How to sync Tempo Worklogs between two Jira Cloud instances.

Introduction

I'm excited to present this tutorial, where we will explore the seamless synchronization of Tempo Worklogs between two Jira Cloud instances using Exalate.

To set the stage, let's consider a scenario where we have Jira Cloud instance A (JCA) and Jira Cloud instance B (JCB). When a ticket is exalated from JCA to JCB, it is seamlessly populated within JCB, allowing users to interact with it. However, a critical aspect of worklog synchronization arises when a user adds time to the ticket in JCB. It is imperative that this time entry is reflected in JCA to maintain consistency and facilitate effective project tracking.

Conversely, if a user adds or modifies a worklog in JCA, it should be automatically synchronized with JCB to ensure that all stakeholders have up-to-date information across both instances.

This bidirectional synchronization enables teams working on different Jira Cloud instances to collaborate seamlessly, regardless of their physical location or the specific instance they are using.

By implementing Exalate as the integration solution, organizations can establish a robust and reliable connection between JCA and JCB, facilitating real-time updates of Tempo Worklogs.

Tempo Worklogs

After installing Tempo Worklogs, you will see it getting added under your apps selection.

Exalate uses Tempo Cloud REST API to get access to the tempo worklogs.

The use of the Tempo Cloud REST API enables Exalate to retrieve worklogs from the source instance and transfer them to the target instance seamlessly. It also allows for the synchronization of worklog updates, ensuring that any modifications made in one instance are accurately reflected in the other.

Generate Access Token

Exalate requires access to Tempo. To grant secure, temporary access to Tempo you need to create a user access token. This access token is based on current permissions of the Jira Cloud user.

Required permissions

The user who generated the access token must have the following permissions:

- Jira permissions:
  - Create worklogs
  - View all worklogs
  - Log work for others
  - Work on issues
- Tempo permissions:
  - view team worklogs
  - manage team worklogs

Generate the Access Token under **Tempo settings** - **API integration** tab in your Jira Cloud settings.

The Scripts

Jira Cloud A

Outgoing Sync Jira Cloud A

Outgoing Sync

```
replica.workLogs = issue.workLogs
TempoWorkLogSync.send(
        "dJWtvBhJUkoHroDcEd8iYyYfnd0Bm",  // replace the "token" with the previously generated access token
        replica,
        issue,
        httpClient,
        nodeHelper
)
```

Incoming Sync Jira Cloud A

Add the imports in the beginning of the code.

Add the functions at the end of the code.

Incoming Sync

```
import com.exalate.api.domain.webhook.WebhookEntityType
import com.exalate.basic.domain.hubobject.v1.BasicHubIssue
import com.exalate.basic.domain.hubobject.v1.BasicHubUser
import com.exalate.basic.domain.hubobject.v1.BasicHubWorkLog
import com.exalate.replication.services.replication.impersonation.AuditLogService
import org.slf4j.LoggerFactory
import org.slf4j.Logger
import play.libs.Json
import scala.concurrent.Await$;
import scala.concurrent.duration.Duration$;
import java.text.SimpleDateFormat
import java.time.Instant

//Your normal Incoming Sync code
//Add these functions at the end

Worklogs.receive(
          "dJWtvBhJUkoHroDcEd8iYyYfnd0Bm",  // replace the "token" with the previously generated access token
          replica,
          issue,
          httpClient,
          traces,
          nodeHelper
)


class Worklogs {




    static receive(String token, BasicHubIssue replica, BasicHubIssue issue, httpClient, traces, nodeHelper){
```

```
        receive(
                token,
                replica,
                issue,
                httpClient,
                traces,
                nodeHelper,
                { BasicHubWorkLog w ->
                    def getUser = { String key ->
                        def localAuthor = nodeHelper.getUser(key)
                        if (localAuthor == null) {
                            localAuthor = new BasicHubUser()
                            localAuthor.key = "557058:c020323a-70e4-4c07-9ccc-3ad89b1c02ec"
                        }
                        localAuthor
                    }
                    w.author = w.author ? getUser(w.author.key) : null
                    w.updateAuthor = w.updateAuthor ? getUser(w.updateAuthor.key) : null
                    w
                }
        )
    }


    static receive(String token, BasicHubIssue replica, BasicHubIssue issue, httpClient, traces, nodeHelper,
Closure<?> onWorklogFn){
        def http = { String method, String path, Map<String, List<String>> queryParams, String body,
Map<String, List<String>> headers ->

            def await = { future -> Await$.MODULE$.result(future, Duration$.MODULE$.apply(60, java.util.
concurrent.TimeUnit.SECONDS)) }
            def orNull = { scala.Option<?> opt -> opt.isDefined() ? opt.get() : null }
            def pair = { l, r -> scala.Tuple2$.MODULE$.<?, ?>apply(l, r) }
            def none = { scala.Option$.MODULE$.<?> empty() }

            def getGeneralSettings = {
                def classLoader = this.getClassLoader()
                def gsp
                try {
                    gsp = InjectorGetter.getInjector().instanceOf(classLoader.loadClass("com.exalate.api.
persistence.issuetracker.jcloud.IJCloudGeneralSettingsRepository"))
                } catch(ClassNotFoundException exception) {
                    gsp = InjectorGetter.getInjector().instanceOf(classLoader.loadClass("com.exalate.api.
persistence.issuetracker.jcloud.IJCloudGeneralSettingsPersistence"))
                }
                def gsOpt = await(gsp.get())
                def gs = orNull(gsOpt)
                gs
            }
            final def gs = getGeneralSettings()

            def removeTailingSlash = { String str -> str.trim().replace("/+\$", "") }
            final def tempoRestApiUrl = "https://api.tempo.io/core/3"

            def parseQueryString = { String string ->
                string.split('&').collectEntries{ param ->
                    param.split('=', 2).collect{ URLDecoder.decode(it, 'UTF-8') }
                }
            }


            def parseUri
            parseUri = { String uri ->
                def parsedUri
                try {
                    parsedUri = new URI(uri)
                    if (parsedUri.scheme == 'mailto') {
                        def schemeSpecificPartList = parsedUri.schemeSpecificPart.split('\\?', 2)
                        def tempMailMap = parseQueryString(schemeSpecificPartList[1])
                        //noinspection GrUnresolvedAccess
                        parsedUri.metaClass.mailMap = [
```

```
                                recipient: schemeSpecificPartList[0],
                                cc       : tempMailMap.find { //noinspection GrUnresolvedAccess
                                    it.key.toLowerCase() == 'cc' }.value,
                                bcc      : tempMailMap.find { //noinspection GrUnresolvedAccess
                                    it.key.toLowerCase() == 'bcc' }.value,
                                subject  : tempMailMap.find { //noinspection GrUnresolvedAccess
                                    it.key.toLowerCase() == 'subject' }.value,
                                body     : tempMailMap.find { //noinspection GrUnresolvedAccess
                                    it.key.toLowerCase() == 'body' }.value
                            ]
                        }
                        if (parsedUri.fragment?.contains('?')) {
                            //noinspection GrUnresolvedAccess
                            parsedUri.metaClass.rawQuery = parsedUri.rawFragment.split('\\?')[1]
                            //noinspection GrUnresolvedAccess
                            parsedUri.metaClass.query = parsedUri.fragment.split('\\?')[1]
                            //noinspection GrUnresolvedAccess
                            parsedUri.metaClass.rawFragment = parsedUri.rawFragment.split('\\?')[0]
                            //noinspection GrUnresolvedAccess
                            parsedUri.metaClass.fragment = parsedUri.fragment.split('\\?')[0]
                        }
                        if (parsedUri.rawQuery) {
                            //noinspection GrUnresolvedAccess
                            parsedUri.metaClass.queryMap = parseQueryString(parsedUri.rawQuery)
                        } else {
                            //noinspection GrUnresolvedAccess
                            parsedUri.metaClass.queryMap = null
                        }

                        //noinspection GrUnresolvedAccess
                        if (parsedUri.queryMap) {
                            //noinspection GrUnresolvedAccess
                            parsedUri.queryMap.keySet().each { key ->
                                def value = parsedUri.queryMap[key]
                                //noinspection GrUnresolvedAccess
                                if (value.startsWith('http') || value.startsWith('/')) {
                                    parsedUri.queryMap[key] = parseUri(value)
                                }
                            }
                        }
                    } catch (e) {
                        throw new com.exalate.api.exception.IssueTrackerException("Parsing of URI failed:
$uri\n$e", e)
                    }
                    parsedUri
                }

                def unsanitizedUrl = tempoRestApiUrl + path
                def parsedUri = parseUri(unsanitizedUrl)


                def embeddedQueryParams = parsedUri.queryMap

                def allQueryParams = embeddedQueryParams instanceof Map ?
                        ({
                            def m = [:] as Map<String, List<String>>;
                            m.putAll(embeddedQueryParams as Map<String, List<String>>)
                            m.putAll(queryParams)
                        })()
                        : (queryParams ?: [:] as Map<String, List<String>>)

                def urlWithoutQueryParams = { String url ->
                    URI uri = new URI(url)
                    new URI(uri.getScheme(),
                            uri.getUserInfo(), uri.getHost(), uri.getPort(),
                            uri.getPath(),
                            null,
                            uri.getFragment()).toString()
                }
                def sanitizedUrl = urlWithoutQueryParams(unsanitizedUrl)
```

```
            def response
            try {
                def request = httpClient
                        .ws()
                        .url(sanitizedUrl)
                        .withMethod(method)

                if (headers != null && !headers.isEmpty()) {
                    def scalaHeaders = scala.collection.JavaConversions.asScalaBuffer(
                            headers.entrySet().inject([] as List<scala.Tuple2<String, String>>) { List<scala.
Tuple2<String, String>> result, kv ->
                                kv.value.each { v -> result.add(pair(kv.key, v) as scala.Tuple2<String,
String>) }
                                result
                            }
                    )
                    request = request.withHeaders(scalaHeaders)
                }

                if (!allQueryParams.isEmpty()) {
                    def scalaQueryParams = scala.collection.JavaConversions.asScalaBuffer(queryParams.
entrySet().inject([] as List<scala.Tuple2<String, String>>) { List<scala.Tuple2<String, String>> result, kv ->
                        kv.value.each { v -> result.add(pair(kv.key, v) as scala.Tuple2<String, String>) }
                        result
                    })
                    request = request.withQueryString(scalaQueryParams)
                }

                if (body != null) {
                    def writable = play.api.libs.ws.WSBodyWritables$.MODULE$.writeableOf_String()
                    request = request.withBody(body, writable)
                }
                response = await(request.execute())


            } catch (Exception e) {
                throw new com.exalate.api.exception.IssueTrackerException("Unable to perform the request
$method $path, please contact Exalate Support: ".toString() + e.message, e)
            }
            if (response.status() >= 300) {
                throw new com.exalate.api.exception.IssueTrackerException("Failed to perform the request
$method $path ${body ? "with body `$body`".toString() : ""}(status ${response.status()}), please contact
Exalate Support: ".toString() + response.body())
            }
            response.body() as String
        }

        def gsp = InjectorGetter.getInjector().instanceOf(AuditLogService.class)

        def js = new groovy.json.JsonSlurper()
        def jo = new groovy.json.JsonOutput()

        def listAdditionalParams = replica.customKeys."tempoWorklogParams" as Map<String, Map<String, Object>>;

        replica.workLogs.findAll{it.id == null}.each{ BasicHubWorkLog worklog ->
            def transformedWorklog


            try {
                transformedWorklog = onWorklogFn(worklog)
            } catch (com.exalate.api.exception.IssueTrackerException ite) {
                throw ite
            } catch (Exception e) {
                throw new com.exalate.api.exception.IssueTrackerException(e)
            }
            if (transformedWorklog instanceof BasicHubWorkLog) {
                worklog = transformedWorklog as BasicHubWorkLog
            } else if (transformedWorklog == null) {
                return
            }
```

```
            def auditLogOpt = gsp.createAuditLog(scala.Option$.MODULE$.<String>apply(issue.id as String),
                    WebhookEntityType.WORKLOG_CREATED,
                    worklog.getAuthor().getKey()
            )

            def attributes = ((listAdditionalParams?.get(worklog.remoteId.toString())?.get("attributes") as
Map<String, Object>)?.get("values") as List<Map<String, String>>)?.inject([]){
                List<Map<String, String>>result, Map<String, String> attribute ->
                    result.add(
                            [
                                    key: attribute.get("key"),
                                    value: attribute.get("value")
                            ]
                    )
                    result
            } ?: []
            def properties = [
                    issueKey : issue.key,
                    timeSpentSeconds : worklog.getTimeSpent(),
                    billableSeconds: listAdditionalParams?.get(worklog.remoteId.toString())?.get
("billableSeconds") ?: worklog.getTimeSpent(),
                    startDate : new java.text.SimpleDateFormat("yyyy-MM-dd").format(worklog.startDate),
                    startTime : new java.text.SimpleDateFormat("hh:mm:ss").format(worklog.startDate) ?:
listAdditionalParams?.get(worklog.remoteId.toString())?.get("startTime"),//strDateSplitted[1].split("\\.")[0],
                    description : worklog.getComment(),
                    authorAccountId : worklog.getAuthor().getKey(),
                    remainingEstimateSeconds:  replica.remainingEstimate ?: listAdditionalParams?.get(worklog.
remoteId.toString())?.get("remainingEstimateSeconds"),
                    attributes : attributes
            ]
            def jsonTempoPost = jo.toJson(properties)


            def response = js.parseText(http(
                    "POST",
                    "/worklogs",
                    null,
                    jsonTempoPost,
                    [
                            "Authorization":["Bearer ${token}".toString()],
                            "Content-Type":["application/json"],
                    ]
            ))
            println(response)

            gsp.updateAuditLog(scala.Option$.MODULE$.apply(auditLogOpt), issue.id as String, response
["jiraWorklogId"] as String, Json.stringify(Json.toJson(response)))


            String localIdStr = response["tempoWorklogId"]
            String remoteIdStr = worklog.remoteId.toString()

            com.exalate.api.domain.twintrace.INonPersistentTrace trace = new com.exalate.basic.domain.
BasicNonPersistentTrace()
                    .setLocalId(localIdStr)
                    .setRemoteId(remoteIdStr)
                    .setType(com.exalate.api.domain.twintrace.TraceType.WORKLOG)
                    .setAction(com.exalate.api.domain.twintrace.TraceAction.NONE)
                    .setToSynchronize(true)
            traces.add(trace)
        }



    }


}
```

Jira Cloud B

Outgoing Sync Jira Cloud B

Outgoing Sync

```
replica.workLogs = issue.workLogs
TempoWorkLogSync.send(
        "B0V4tQJ22LhPnznllWoT2s0N29So5",  // replace the "token" with the previously generated access token
        replica,
        issue,
        httpClient,
        nodeHelper
)
```

Incoming Sync Jira Cloud B

Add the imports in the beginning of the code.

Add the functions at the end of the code.

Incoming Sync

```
import com.exalate.api.domain.webhook.WebhookEntityType
import com.exalate.basic.domain.hubobject.v1.BasicHubIssue
import com.exalate.basic.domain.hubobject.v1.BasicHubUser
import com.exalate.basic.domain.hubobject.v1.BasicHubWorkLog
import com.exalate.replication.services.replication.impersonation.AuditLogService
import org.slf4j.LoggerFactory
import org.slf4j.Logger
import play.libs.Json
import scala.concurrent.Await$;
import scala.concurrent.duration.Duration$;
import java.text.SimpleDateFormat
import java.time.Instant

//Your normal Incoming Sync code
//Add these functions at the end

Worklogs.receive(
          "B0V4tQJ22LhPnznllWoT2s0N29So5",  // replace the "token" with the previously generated access token
          replica,
          issue,
          httpClient,
          traces,
          nodeHelper
)


class Worklogs {
```

```
    static receive(String token, BasicHubIssue replica, BasicHubIssue issue, httpClient, traces, nodeHelper){
        receive(
                token,
                replica,
                issue,
                httpClient,
                traces,
                nodeHelper,
                { BasicHubWorkLog w ->
                    def getUser = { String key ->
                        def localAuthor = nodeHelper.getUser(key)
                        if (localAuthor == null) {
                            localAuthor = new BasicHubUser()
                            localAuthor.key = "557058:c020323a-70e4-4c07-9ccc-3ad89b1c02ec"
                        }
                        localAuthor
                    }
                    w.author = w.author ? getUser(w.author.key) : null
                    w.updateAuthor = w.updateAuthor ? getUser(w.updateAuthor.key) : null
                    w
                }
        )
    }


    static receive(String token, BasicHubIssue replica, BasicHubIssue issue, httpClient, traces, nodeHelper,
Closure<?> onWorklogFn){
        def http = { String method, String path, Map<String, List<String>> queryParams, String body,
Map<String, List<String>> headers ->

            def await = { future -> Await$.MODULE$.result(future, Duration$.MODULE$.apply(60, java.util.
concurrent.TimeUnit.SECONDS)) }
            def orNull = { scala.Option<?> opt -> opt.isDefined() ? opt.get() : null }
            def pair = { l, r -> scala.Tuple2$.MODULE$.<?, ?>apply(l, r) }
            def none = { scala.Option$.MODULE$.<?> empty() }

            def getGeneralSettings = {
                def classLoader = this.getClassLoader()
                def gsp
                try {
                    gsp = InjectorGetter.getInjector().instanceOf(classLoader.loadClass("com.exalate.api.
persistence.issuetracker.jcloud.IJCloudGeneralSettingsRepository"))
                } catch(ClassNotFoundException exception) {
                    gsp = InjectorGetter.getInjector().instanceOf(classLoader.loadClass("com.exalate.api.
persistence.issuetracker.jcloud.IJCloudGeneralSettingsPersistence"))
                }
                def gsOpt = await(gsp.get())
                def gs = orNull(gsOpt)
                gs
            }
            final def gs = getGeneralSettings()

            def removeTailingSlash = { String str -> str.trim().replace("/+\$", "") }
            final def tempoRestApiUrl = "https://api.tempo.io/core/3"

            def parseQueryString = { String string ->
                string.split('&').collectEntries{ param ->
                    param.split('=', 2).collect{ URLDecoder.decode(it, 'UTF-8') }
                }
            }


            def parseUri
            parseUri = { String uri ->
                def parsedUri
                try {
                    parsedUri = new URI(uri)
                    if (parsedUri.scheme == 'mailto') {
                        def schemeSpecificPartList = parsedUri.schemeSpecificPart.split('\\?', 2)
                        def tempMailMap = parseQueryString(schemeSpecificPartList[1])
                        //noinspection GrUnresolvedAccess
```

```
                    parsedUri.metaClass.mailMap = [
                            recipient: schemeSpecificPartList[0],
                            cc       : tempMailMap.find { //noinspection GrUnresolvedAccess
                                it.key.toLowerCase() == 'cc' }.value,
                            bcc      : tempMailMap.find { //noinspection GrUnresolvedAccess
                                it.key.toLowerCase() == 'bcc' }.value,
                            subject  : tempMailMap.find { //noinspection GrUnresolvedAccess
                                it.key.toLowerCase() == 'subject' }.value,
                            body     : tempMailMap.find { //noinspection GrUnresolvedAccess
                                it.key.toLowerCase() == 'body' }.value
                    ]
                }
                if (parsedUri.fragment?.contains('?')) {
                    //noinspection GrUnresolvedAccess
                    parsedUri.metaClass.rawQuery = parsedUri.rawFragment.split('\\?')[1]
                    //noinspection GrUnresolvedAccess
                    parsedUri.metaClass.query = parsedUri.fragment.split('\\?')[1]
                    //noinspection GrUnresolvedAccess
                    parsedUri.metaClass.rawFragment = parsedUri.rawFragment.split('\\?')[0]
                    //noinspection GrUnresolvedAccess
                    parsedUri.metaClass.fragment = parsedUri.fragment.split('\\?')[0]
                }
                if (parsedUri.rawQuery) {
                    //noinspection GrUnresolvedAccess
                    parsedUri.metaClass.queryMap = parseQueryString(parsedUri.rawQuery)
                } else {
                    //noinspection GrUnresolvedAccess
                    parsedUri.metaClass.queryMap = null
                }

                //noinspection GrUnresolvedAccess
                if (parsedUri.queryMap) {
                    //noinspection GrUnresolvedAccess
                    parsedUri.queryMap.keySet().each { key ->
                        def value = parsedUri.queryMap[key]
                        //noinspection GrUnresolvedAccess
                        if (value.startsWith('http') || value.startsWith('/')) {
                            parsedUri.queryMap[key] = parseUri(value)
                        }
                    }
                }
            } catch (e) {
                throw new com.exalate.api.exception.IssueTrackerException("Parsing of URI failed:
$uri\n$e", e)
            }
            parsedUri
        }

        def unsanitizedUrl = tempoRestApiUrl + path
        def parsedUri = parseUri(unsanitizedUrl)


        def embeddedQueryParams = parsedUri.queryMap

        def allQueryParams = embeddedQueryParams instanceof Map ?
                ({
                    def m = [:] as Map<String, List<String>>;
                    m.putAll(embeddedQueryParams as Map<String, List<String>>)
                    m.putAll(queryParams)
                })()
                : (queryParams ?: [:] as Map<String, List<String>>)

        def urlWithoutQueryParams = { String url ->
            URI uri = new URI(url)
            new URI(uri.getScheme(),
                    uri.getUserInfo(), uri.getHost(), uri.getPort(),
                    uri.getPath(),
                    null,
                    uri.getFragment()).toString()
        }
        def sanitizedUrl = urlWithoutQueryParams(unsanitizedUrl)
```

```
            def response
            try {
                def request = httpClient
                        .ws()
                        .url(sanitizedUrl)
                        .withMethod(method)

                if (headers != null && !headers.isEmpty()) {
                    def scalaHeaders = scala.collection.JavaConversions.asScalaBuffer(
                            headers.entrySet().inject([] as List<scala.Tuple2<String, String>>) { List<scala.
Tuple2<String, String>> result, kv ->
                                kv.value.each { v -> result.add(pair(kv.key, v) as scala.Tuple2<String,
String>) }
                                result
                            }
                    )
                    request = request.withHeaders(scalaHeaders)
                }

                if (!allQueryParams.isEmpty()) {
                    def scalaQueryParams = scala.collection.JavaConversions.asScalaBuffer(queryParams.
entrySet().inject([] as List<scala.Tuple2<String, String>>) { List<scala.Tuple2<String, String>> result, kv ->
                        kv.value.each { v -> result.add(pair(kv.key, v) as scala.Tuple2<String, String>) }
                        result
                    })
                    request = request.withQueryString(scalaQueryParams)
                }

                if (body != null) {
                    def writable = play.api.libs.ws.WSBodyWritables$.MODULE$.writeableOf_String()
                    request = request.withBody(body, writable)
                }
                response = await(request.execute())


            } catch (Exception e) {
                throw new com.exalate.api.exception.IssueTrackerException("Unable to perform the request
$method $path, please contact Exalate Support: ".toString() + e.message, e)
            }
            if (response.status() >= 300) {
                throw new com.exalate.api.exception.IssueTrackerException("Failed to perform the request
$method $path ${body ? "with body `$body`".toString() : ""}(status ${response.status()}), please contact
Exalate Support: ".toString() + response.body())
            }
            response.body() as String
        }

        def gsp = InjectorGetter.getInjector().instanceOf(AuditLogService.class)

        def js = new groovy.json.JsonSlurper()
        def jo = new groovy.json.JsonOutput()

        def listAdditionalParams = replica.customKeys."tempoWorklogParams" as Map<String, Map<String, Object>>;

        replica.workLogs.findAll{it.id == null}.each{ BasicHubWorkLog worklog ->
            def transformedWorklog


            try {
                transformedWorklog = onWorklogFn(worklog)
            } catch (com.exalate.api.exception.IssueTrackerException ite) {
                throw ite
            } catch (Exception e) {
                throw new com.exalate.api.exception.IssueTrackerException(e)
            }
            if (transformedWorklog instanceof BasicHubWorkLog) {
                worklog = transformedWorklog as BasicHubWorkLog
            } else if (transformedWorklog == null) {
                return
            }
```

```
            def auditLogOpt = gsp.createAuditLog(scala.Option$.MODULE$.<String>apply(issue.id as String),
                    WebhookEntityType.WORKLOG_CREATED,
                    worklog.getAuthor().getKey()
            )

            def attributes = ((listAdditionalParams?.get(worklog.remoteId.toString())?.get("attributes") as
Map<String, Object>)?.get("values") as List<Map<String, String>>)?.inject([]){
                List<Map<String, String>>result, Map<String, String> attribute ->
                    result.add(
                            [
                                    key: attribute.get("key"),
                                    value: attribute.get("value")
                            ]
                    )
                    result
            } ?: []
            def properties = [
                    issueKey : issue.key,
                    timeSpentSeconds : worklog.getTimeSpent(),
                    billableSeconds: listAdditionalParams?.get(worklog.remoteId.toString())?.get
("billableSeconds") ?: worklog.getTimeSpent(),
                    startDate : new java.text.SimpleDateFormat("yyyy-MM-dd").format(worklog.startDate),
                    startTime : new java.text.SimpleDateFormat("hh:mm:ss").format(worklog.startDate) ?:
listAdditionalParams?.get(worklog.remoteId.toString())?.get("startTime"),//strDateSplitted[1].split("\\.")[0],
                    description : worklog.getComment(),
                    authorAccountId : worklog.getAuthor().getKey(),
                    remainingEstimateSeconds:  replica.remainingEstimate ?: listAdditionalParams?.get(worklog.
remoteId.toString())?.get("remainingEstimateSeconds"),
                    attributes : attributes
            ]
            def jsonTempoPost = jo.toJson(properties)


            def response = js.parseText(http(
                    "POST",
                    "/worklogs",
                    null,
                    jsonTempoPost,
                    [
                            "Authorization":["Bearer ${token}".toString()],
                            "Content-Type":["application/json"],
                    ]
            ))
            println(response)

            gsp.updateAuditLog(scala.Option$.MODULE$.apply(auditLogOpt), issue.id as String, response
["jiraWorklogId"] as String, Json.stringify(Json.toJson(response)))


            String localIdStr = response["tempoWorklogId"]
            String remoteIdStr = worklog.remoteId.toString()

            com.exalate.api.domain.twintrace.INonPersistentTrace trace = new com.exalate.basic.domain.
BasicNonPersistentTrace()
                    .setLocalId(localIdStr)
                    .setRemoteId(remoteIdStr)
                    .setType(com.exalate.api.domain.twintrace.TraceType.WORKLOG)
                    .setAction(com.exalate.api.domain.twintrace.TraceAction.NONE)
                    .setToSynchronize(true)
            traces.add(trace)
        }



    }

  }
```

Video

Questions

Recent Questions

Ask a question

1. 0
votes
Non-Trivial Sync in Script Mode
   - **0** answers
   - Chris Sebok
   - May 16, 2024
   - Space: Exalate
   - exalate
   - script
   - customfields
   - statusmapping
2. 0
votes
Lifecycle blocking error
   - **1** answer
   - Jozsef Lehocz
   - May 13, 2024
   - Space: Exalate
   - exalate
   - blocking
   - error
   - upgrade
3. 0
votes
How to sync issues from two Jira cloud sites to one? Is this possible?
   - **1** answer
   - Vedant Kulkarni
   - Mar 27, 2024
   - Space: Exalate
   - jira-cloud
   - connector-cloud-jira
4. 0
votes
Issues are not syncing to local destination desk
   - **1** answer
   - John Lombardo
   - Mar 21, 2024
   - Space: Exalate
   - exalate
   - connector-cloud-jira
   - jira-cloud
5. 0
votes
Not getting the Custom Field sync value from JIRA to Zendesk
   - **1** answer
   - Dinesh Gupalan
   - Mar 18, 2024
   - Space: Exalate
   - connector-onpremise-jira-zendesk
6. 0
votes

Need to split source comment into multiple destination comments. (The entered text is too long.)

- **1** answer
- Kyle K
- Feb 14, 2024
- Space: Exalate
- exalate
- jira-cloud
- comments

7. 0
votes

exalate plugin of jira server

- **1** answer
- Sreenivasaraju P
- Feb 08, 2024
- Space: Exalate
- exalate

8. 0
votes

How to have keys match between jira cloud instances

- **2** answers
- Chris Matthews
- Feb 07, 2024
- Space: Exalate
- connector-cloud-jira

9. 0
votes

how decrease dimension of /var/atlassian/application-data/jira/data/exalate

- **1** answer
- Francesco Doricchi
- Dec 14, 2023
- Space: Exalate
- exalate

10. 1
vote

How to Impersonate Attachments in Jira Cloud?

- **0** answers
- Valeriia Solianikova
- Dec 12, 2023
- Space: Exalate
- exalate
- jira
- attachments

Ask a question