Salesforce Zendesk: Syncing the Account-Contact relationship from Salesforce to Organization-User in Zendesk

This use case revolves around syncing the accounts and contacts from Salesforce to Zendesk, and ensuring that the Zendesk entities are kept up to date when the corresponding Salesforce entity is updated. Let us start by breaking down the exact requirements:

- When a new Account is provisioned in Salesforce, a corresponding Organization should be provisioned in Zendesk
    - The website and phone number fields from the Account need to be brought over to organization fields in Zendesk
- When a new Contact is provisioned in a certain account, a corresponding User should be provisioned in Zendesk
    - The phone number field from the Contact should be brought over to the user field in Zendesk.
- When the Salesforce Account or Contact are updated, the corresponding entities on the Zendesk side must be updated.

The following diagram depicts the use case:

For the Salesforce part, we need to send out two things i.e. Account and Contact. The script for the outgoing part for Account is as follows:

Salesforce Outgoing

```
if(entity.entityType == "Account") {
    replica.key   = entity.Id
    replica.Name  = entity.Name
    replica.Website = entity.Website
    replica.Phone = entity.Phone
    replica."tType" = "Account"
}
```

Here we have sent an additional variable called replica."tType" to designate the type of entity we are sending across. This would help on the Zendesk side to decide what entity type we are dealing with.

In order to send over the Contact information, the script is as follows:

Salesforce Outgoing

```
if(entity.entityType == "Contact") {
    replica.key   = entity.Id
    replica.Name  = entity.Name
    def res = httpClient.get("/services/data/v54.0/query/?q=SELECT+Name+from+Account+where+id=%27${entity.
AccountId}%27")
    replica.AccountName = res.records[0].Name
    replica.Email = entity.Email
    replica.MobilePhone = entity.MobilePhone
    replica."tType" = "Contact"
}
```

Here, we run an API call on the Account table to fetch the Account Name of the Contact. This is needed in order to correctly link the Zendesk objects together.

Moving the Zendesk part now, we will be receiving the Salesforce payload, and running custom API calls in order to create the Organization and Users correctly. For this we will need the ZdClient class firstly:

ZdClient

```
class InjectorGetter {
    static Object getInjector() {
        try {
```

```
            return play.api.Play$.MODULE$.current().injector()
        } catch (e) {
            def context = com.exalate.replication.services.processor.CreateReplicaProcessor$.MODULE$.
threadLocalContext.get()
            if (!context) {
                context = com.exalate.replication.services.processor.ChangeIssueProcessor$.MODULE$.
threadLocalContext.get()
            }
            if (!context) {
                context = com.exalate.replication.services.processor.CreateIssueProcessor$.MODULE$.
threadLocalContext.get()
            }
            if (!context) {
                throw new com.exalate.api.exception.IssueTrackerException(""" No context for executing external
script CreateIssue.groovy. Please contact Exalate Support.""".toString())
            }
            context.injector
        }
    }
}
class ZdClient {
    // SCALA HELPERS
    private static <T> T await(scala.concurrent.Future<T> f) {
        scala.concurrent.Await$.MODULE$.result(f, scala.concurrent.duration.Duration$.MODULE$.Inf())
    }

    private static <T> T orNull(scala.Option<T> opt) { opt.isDefined() ? opt.get() : null }

    private static <T> scala.Option<T> none() { scala.Option$.MODULE$.<T> empty() }

    @SuppressWarnings("GroovyUnusedDeclaration")
    private static <T> scala.Option<T> none(Class<T> evidence) { scala.Option$.MODULE$.<T> empty() }

    private static <L, R> scala.Tuple2<L, R> pair(L l, R r) { scala.Tuple2$.MODULE$.<L, R> apply(l, r) }

    // SERVICES AND EXALATE API
    private static play.api.inject.Injector getInjector() {
        InjectorGetter.getInjector()
    }

    private static def getGeneralSettings() {
        def gsp = InjectorGetter.getInjector().instanceOf(com.exalate.api.persistence.issuetracker.
IGeneralSettingsRepository.class)
        def gsOpt = await(gsp.get())
        def gs = orNull(gsOpt)
        gs
    }

    private static String getIssueTrackerUrl() {
        final def gs = getGeneralSettings()

        def removeTailingSlash = { String str -> str.trim().replace("/+\$", "") }
        final def issueTrackerUrl = removeTailingSlash(gs.issueTrackerUrl)
        issueTrackerUrl
    }

    private httpClient

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

    //Usage examples: https://gist.github.com/treyturner/4c0f609677cbab7cef9f
    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])
                    parsedUri.metaClass.mailMap = [
                            recipient: schemeSpecificPartList[0],
                            cc       : tempMailMap.find { it.key.toLowerCase() == 'cc' }.value,
                            bcc      : tempMailMap.find { it.key.toLowerCase() == 'bcc' }.value,
                            subject  : tempMailMap.find { it.key.toLowerCase() == 'subject' }.value,
                            body     : tempMailMap.find { it.key.toLowerCase() == 'body' }.value
                    ]
                }
                if (parsedUri.fragment?.contains('?')) { // handle both fragment and query string
                    parsedUri.metaClass.rawQuery = parsedUri.rawFragment.split('\\?')[1]
                    parsedUri.metaClass.query = parsedUri.fragment.split('\\?')[1]
                    parsedUri.metaClass.rawFragment = parsedUri.rawFragment.split('\\?')[0]
                    parsedUri.metaClass.fragment = parsedUri.fragment.split('\\?')[0]
                }
                if (parsedUri.rawQuery) {
                    parsedUri.metaClass.queryMap = parseQueryString(parsedUri.rawQuery)
                } else {
                    parsedUri.metaClass.queryMap = null
                }

                if (parsedUri.queryMap) {
                    parsedUri.queryMap.keySet().each { key ->
                        def value = parsedUri.queryMap[key]
                        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 $e ", e)
            }
            parsedUri
        }
    }

    ZdClient(httpClient, debug) {
        this.httpClient = httpClient
        this.debug = debug
    }
    def debug

    String http(String method, String path, java.util.Map<String, List<String>> queryParams, String body, java.
util.Map<String, List<String>> headers) {
        http(method, path, queryParams, body, headers) { Response response ->
            if (response.code >= 300) {
                throw new com.exalate.api.exception.IssueTrackerException(
                        """Failed to perform the request $method $path (status ${response.code}), and body was:
```$body``` Please contact Exalate Support: """.toString() + response.body
)
            }
            response.body as String
        }
    }

    public <R> R http(String method, String path, java.util.Map<String, List<String>> queryParams, String body,
java.util.Map<String, List<String>> headers, Closure<R> transformResponseFn) {
        def gs = getGeneralSettings()
        def unsanitizedUrl = issueTrackerUrl + path
        def parsedUri = parseUri(unsanitizedUrl)

        def embeddedQueryParams = parsedUri.queryMap

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

```
                : (queryParams ?: [:] as java.util.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, // Ignore the query part of the input url
                    uri.getFragment()).toString()
        }
        def sanitizedUrl = urlWithoutQueryParams(unsanitizedUrl)

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

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

            def authorizationHeader = await(httpClient.zendeskClient.getAuthHeaderFromGs())
            request = request.addHttpHeaders(scala.collection.JavaConversions.asScalaBuffer([pair
("Authorization", authorizationHeader) as scala.Tuple2<String, String>]))
            //debug.error("${request.method()} ${request.url()} ${request.headers()}")
            response = await(request.execute())
        } catch (Exception e) {
            throw new com.exalate.api.exception.IssueTrackerException(
                    """Unable to perform the request $method $path with body:```$body```, please contact
Exalate Support: """.toString() + e.message,
                    e
            )
        }
        java.util.Map<String, List<String>> javaMap = [:]
        for (scala.Tuple2<String, scala.collection.Seq<String>> headerTuple : scala.collection.JavaConverters.
bufferAsJavaListConverter(response.allHeaders().toBuffer()).asJava()) {
            def javaList = []
            javaList.addAll(scala.collection.JavaConverters.bufferAsJavaListConverter(headerTuple._2().
toBuffer()).asJava())
            javaMap[headerTuple._1()] = javaList
        }
```

```
        def javaResponse = new Response(response.body(), new Integer(response.status()), javaMap)
        return transformResponseFn(javaResponse)
    }

    public static class Response {
        final String body
        final Integer code
        final java.util.Map<String, List<String>> headers

        Response(String body, Integer code, java.util.Map<String, List<String>> headers) {
            this.body = body
            this.code = code
            this.headers = headers
        }
    }
}
```

Once we have this class in place, we are able to make any API calls to the Zendesk API (or any other for that matter). The following script now goes and creates the organization and the users:

Zendesk Incoming

```
if (replica."tType" == "Account"){
    if (issue.customFields."Salesforce Account Number".value == null){
        def _rJson
        def res = new ZdClient(httpClient, debug)
            .http(
                    "POST",
                    "/api/v2/organizations",
                    [:],
                    "{\"organization\": {\"name\": \"${replica.Name}\"}}",
                    ["Accept": ["application/json"], "Content-type" : ["application/json"]]
            )
            {
                response ->
                if (response.code >= 400) {
                    throw new com.exalate.api.exception.IssueTrackerException("Failed to get orgnaizations with
name ")
                }
                else {
                    def _r = response.body as String
                    def js = new groovy.json.JsonSlurper()
                    _rJson = _r == null ? null : js.parseText(_r)
                    _rJson
                    }
            }
        issue.customFields."Salesforce Account Number".value = _rJson.organization.id
        issue.summary      = replica.summary
        issue.description  = replica.description ?: "No description"
        store(issue)

        res = new ZdClient(httpClient, debug)
            .http(
                    "PUT",
                    "/api/v2/organizations/${issue.customFields.'Salesforce Account Number'.value}",
                    [:],
                    "{\"organization\": {\"organization_fields\": {\"org_phone\": \"${replica.Phone}\", \"
test_copy\": \"${replica.Website}\"}}",
                    ["Accept": ["application/json"], "Content-type" : ["application/json"]]
            )
            {
                response ->
                if (response.code >= 400) {
                    throw new com.exalate.api.exception.IssueTrackerException("Failed to get orgnaizations with
name ")
                }
                else {
                    def _r = response.body as String
                    def js = new groovy.json.JsonSlurper()
```

```
                    _rJson = _r == null ? null : js.parseText(_r)
                    _rJson
                    }
            }
    }
    else{
        def _rJson
        def res = new ZdClient(httpClient, debug)
            .http(
                    "PUT",
                    "/api/v2/organizations/${issue.customFields.'Salesforce Account Number'.value}",
                    [:],
                    "{\"organization\": {\"organization_fields\": {\"org_phone\": \"${replica.Phone}\", \"
test_copy\": \"${replica.Website}\"}}",
                    ["Accept": ["application/json"], "Content-type" : ["application/json"]]
            )
            {
                response ->
                if (response.code >= 400) {
                    throw new com.exalate.api.exception.IssueTrackerException("Failed to get orgnaizations with
name ")
                }
                else {
                    def _r = response.body as String
                    def js = new groovy.json.JsonSlurper()
                    _rJson = _r == null ? null : js.parseText(_r)
                    _rJson
                    }
            }
    }
}
else if (replica.tType == "Contact"){
    if (issue.customFields."Salesforce Account Number".value == null){
        def _rJson
        def res = new ZdClient(httpClient, debug)
            .http(
                    "POST",
                    "/api/v2/users",
                    [:],
                    "{\"user\": {\"email\": \"${replica.Email}\", \"name\": \"${replica.Name}\", \"
user_fields\": {\"test_copy\": \"${replica.MobilePhone}\"},\"organization\": {\"name\" : \"${replica.
AccountName}\"}}}",
                    ["Accept": ["application/json"], "Content-type" : ["application/json"]]
            )
            {
                response ->
                if (response.code >= 400) {
                    throw new com.exalate.api.exception.IssueTrackerException("Failed to get orgnaizations with
name ")
                }
                else {
                    def _r = response.body as String
                    def js = new groovy.json.JsonSlurper()
                    _rJson = _r == null ? null : js.parseText(_r)
                    _rJson
                    }
            }
        issue.summary      = replica.summary
        issue.description  = "No description"
        issue.customFields."Salesforce Account Number".value = _rJson.user.id
    }
    else{
        def _rJson
        def res = new ZdClient(httpClient, debug)
            .http(
                    "PUT",
                    "/api/v2/users/${issue.customFields.'Salesforce Account Number'.value}",
                    [:],
                    "{\"user\": {\"email\": \"${replica.Email}\", \"name\": \"${replica.Name}\", \"
user_fields\": {\"test_copy\": \"${replica.MobilePhone}\"},\"organization\": {\"name\" : \"${replica.
AccountName}\"}}}",
```

```
                    ["Accept": ["application/json"], "Content-type" : ["application/json"]]
            )
            {
                response ->
                if (response.code >= 400) {
                    throw new com.exalate.api.exception.IssueTrackerException("Failed to get orgnaizations with
name ")
                }
                else {
                    def _r = response.body as String
                    def js = new groovy.json.JsonSlurper()
                    _rJson = _r == null ? null : js.parseText(_r)
                    _rJson
                    }
            }
    }
}
```

The above script will create a Zendesk ticket when a new Account/ Contact is received from Salesforce side, and will use a custom field called "Salesforce Account Number" to hold the accountId from SF (to maintain the correlation).

A video demonstration of the solution is as follows:

A complete listing of the scripts:

- Salesforce
- Zendesk