BizDock Event handler

This Plugin is a powerful system that allows to trigger events (send an email, call a web service...) just after a specific action done in BizDock (create an Initiative, change the name of an Employee...).

Data Type

The BizDock Event handler Plugin is working with the following Data Type:

- User: User
- Role: SystemLevelRoleType
- Employee: Actor
- Activity allocated Employee: TimesheetActivityAllocatedActor
- Application Block: ApplicationBlock
- Budget Bucket: BudgetBucket, BudgetBucketLine
- Org Unit: OrgUnit
- Portfolio: Portfolio
- Requirement: Requirement
- Purchase Order: PurchaseOrder, PurchaseOrderLineItem
- Initiative/Release: PortfolioEntry
- Initiative/Release Budget: PortfolioEntryBudget, PortfolioEntryBudgetLine
- Initiative/Release Event: PortfolioEntryEvent
- Initiative/Release Package: PortfolioEntryPlanningPackage
- Initiative/Release Report: PortfolioEntryReport
- Initiative/Release Risk: PortfolioEntryRisk
- Initiative/Release allocated Employee: PortfolioEntryResourcePlanAllocatedActor
- Initiative/Release allocated Org Unit: PortfolioEntryResourcePlanAllocatedOrgUnit
- Initiative/Release allocated Competency: PortfolioEntryResourcePlanAllocatedCompetency
- Initiative/Release Stakeholder: Stakeholder
- Initiative/Release Work Order: WorkOrder
- Initiative Iteration: Iteration
- LifeCycleMilestoneInstance: LifeCycleMilestoneInstance

The full details of attributes of each type could be found here.

Data exchange

Events

The Plugin is treating the following Events:

- IN events: *none*
- OUT events:
  - CREATE: when a new BizDock instance of one of the above Data Types is created.
  - DELETE: when an existing BizDock instance of one of the above Data Types is deleted.

- UPDATE: when an existing BizDock instance of one of the above Data Types is updated.

Admin Commands

This Plugin does not provide any Admin Command.

Configuration

Hook script

A script which defines the events that could be listened and what actions should be performed for a named event.

The register method

The register method allows to define the Data Types that are supported (listened for events).

> The supported Data Types should a subset of the Plugin Data Types.

Signature

```
function register(supportedDataTypes)
```

Attributes

- supportedDataTypes: the supported Data Types. This parameters is given by reference: the goal is to fill it.

Usage

Add the wished Data Types to support in the supportedDataTypes object. Example:

```
function register(supportedDataTypes){
  supportedDataTypes.add("PortfolioEntry");
  supportedDataTypes.add("Actor");
}
```

The notify method

The notify method is the action that is called after each event on the supported Data Types (whatever the type and the event).

Signature

```
function notify(objectType, objectId, eventType, modifiedAttributes)
```

Attributes

- objectType: the concerned Data Type
- objectId: the concerned instance (defined with its id) of the Data Type
- eventType: the event type, could be CREATE, DELETE, UPDATE
- modifiedAttributes: all modified attributes of the concerned object with old and new values for each.

> The attributes allow to specify the context of a resulting action, for example in “send an email when an Initiative is updated” the context is represented by the condition: objectType == “PortfolioEntry” && eventType == “UPDATE”.

scriptUtils

scriptUtils is a set of utilities that could be used in the notify method.

Utility	Description	Example
getObjectFromId(objectType, objectId)	Get the instance of the object (in order to get access to its attributes).	getObjectFromId(objectType, objectId)
getCustomAttributes(objectType,objectId)	Get the custom attributes of an instance of the object	getObjectFromId(objectType, objectId)
logMessage(isError, message)	Create a Plugin log.	logMessage(false, “Hello world!”)
logEventMessage(isError, objectType, objectId, eventType, message)	Create a full Plugin log, meaning by including the full trace that generates it.	logEventMessage(true, objectType, objectId, eventType, “An error has occurred”)

Utility	Description	Example
sendMail(subject, body, to)	Send an email.	sendMail("New initiative", "A new initiative has been created", "test@domain.com")
sendNotification(title, message, actionLink, uid)	Send a Notification to a BizDock user.	sendNotification("New initiative", "A new initiative has been created", "/portfolio-entry/overview?id=152", "test_sall")
wsCall(url)	Call a web service.	More details below.
createQuery(dataType)	Create a custom select query on the BizDock DB.	More details below.
addPortfolioEntryEvent(portfolioEntryId, portfolioEntryEventTypeId, eventMessage, jsCreationDate)	Add an Event to the specified portfolio entry	Mode details below.
setSharedRecord(key, stateObject)	Store an object or a String into a shared storage. This storage is accessible to any plugin relying the script utils library. This is thus the same for the scheduler plugin.	More details below.
getSharedRecord(key)	Get a previously stored object (or String) from the shared storage.	More details below.
deleteSharedRecord(key)	Delete the record associated with the specified key.	More details below.

wsCall

The wsCall utility allows to call a web service. Here is the full description:

First, initialize the request:

```
var request=scriptUtils.wsCall(url);
```

where `url` is a string that represents the URL of the web service, for example:

```
var request=scriptUtils.wsCall("https://test.bizdock.io/api/core/actor");
```

Then, parametrize the request:

Method	Description	Example
setMethod(method)	Set the HTTP method (GET, POST, PUT...). More info here.	request.setMethod("POST")
request.setHeader(name, value)	Add an header. This method could be called many times.	request.setHeader("lang", "en")
request.setContentType(contentType)	Set the content type. More info here.	request.setContentType("application/json")
request.setBody(content)	Set the request body, depending of the content type.	request.setBody("{\"firstName\": \"James\",\"lastName\": \"Bond\"")

Finally, execute the request:

```
request.execute(success, error)
```

where `success` and `error` are functions called respectively after a success or an error of the execution, for example:

```
request.execute(
  function(response){
    var result=response.getStatus();
    ...
  },
  function(errorMessage){
    ...
  }
)
```

createQuery

The createQuery utility allows to create a custom select query on the BizDock database. Here is the full description:

First, initialize the query:

```
var query=scriptUtils.createQuery(dataType);
```

where `dataType` is the type of the wished object, for example:

```
var query=scriptUtils.createQuery("PortfolioEntry");
```

Then, customize the query: Simply refers to the Ebean documentation, for example:

```
var current_date_minus_7_days = new Date(today.getTime() -
```

```
7*24*3600*1000);
  var clause = query.expr().or(query.expr().eq("name", "CRM upgrade"),
query.expr().between("creationDate", current_date_minus_7_days, new
Date()));
```

Finally, execute the request:

```
  var list=query.executeQuery(clause);
```

It is possible to exploit the result with:

```
  for (i = 0; i < list.size(); i++) {
    //get(i).name
    ...
  }
```

addPortfolioEntryEvent()

This method expect 4 parameters:

- the id of the portfolio entry (to which the event will be added)
- the type of the event (see in the reference data configuration screen for all the possibilities - this is extensible)
- the message itself (which can contains HTML tags)
- a javascript date (which is used as the date of creation of the event)

Here is an example:

```
scriptUtils.addPortfolioEntryEvent(portfolioEntryId,1,'New <a
href="/portfolio-entry/report/view?id='+portfolioEntryId+'&reportId='+
object.id+'">report</a>published',now);
```

setSharedRecord(), getSharedRecord(), deleteSharedRecord()

If for example, you have 2 plugins A and B and you need to exchange/share data between them, simply do:

```
  //in Plugin A, put the value "Bond" associated with the key "James"
    scriptUtils.setSharedRecord("James","Bond");
    ...
  }
```

```
  //in Plugin B, get the value via the key "James"
    scriptUtils.logMessage(false,"Response is :
"+scriptUtils.getSharedRecord("James"));
  //it will create a log message with "Response is : Bond"  ...
  }
```

It is also possible to store bigger objects. These ones are serialized into the database. Be careful however to the transaction issues which may happens if two plugins compete for the same data at

the very same time.

> This mechanism is storing some data in the database and thus can increase the volume of the BizDock database. Please use it carefully and ensure that you have a scheduled cleanup plugin for the old records if possible.

Usage

The usage is divided in 2 parts:

1. Identify the context: The notify method is called after EVERY events on ALL supported Data Types, so it is necessary to condition the resulting actions according to a context. This last could be defined with the attributes.
2. Perform an action:
   - Send an email
   - Call a web service
   - Create a Plugin log

> In the below examples, we consider that the concerned Data Types are supported.

Example 1

When the name of an Initiative is updated then send an email to its manager in order to alert him.

```
  function notify(objectType, objectId, eventType, modifiedAttributes) {
    if (eventType == "UPDATE" && objectType == "PortfolioEntry" &&
modifiedAttributes.name != null) {
      var object = scriptUtils.getObjectFromId(objectType, objectId);
      scriptUtils.sendMail("Initiative name updated", "The name of the
initiative " + object.governanceId + " has been updated",
object.manager.mail);
    }
  }
```

Example 2

Log (in the Plugin logs) the deletion of an Employee.

```
  function notify(objectType, objectId, eventType, modifiedAttributes) {
    if (eventType == "DELETE" && objectType == "Actor") {
      scriptUtils.logEventMessage(false, objectType, objectId, eventType,
"The Employee " + objectId + " has been deleted.");
    }
```

```
  }
```

Example 3

Send a notification to the Requirement author with the difference (new value - old value) of the Requirement estimation when this last has changed.

```
  function notify(objectType, objectId, eventType, modifiedAttributes) {
    if (eventType == "UPDATE" && objectType == "Requirement" &&
modifiedAttributes.initialEstimation != null &&
modifiedAttributes.initialEstimation.oldValue !=
modifiedAttributes.initialEstimation.newValue) {
      var object = scriptUtils.getObjectFromId(objectType, objectId);
      scriptUtils.sendNotification("Requirement: initial estimation
modified", "The initial estimation of the Requirement \"" + object.name +
"\" has been modified. The difference is: "  +
(modifiedAttributes.initialEstimation.newValue -
modifiedAttributes.initialEstimation.oldValue), "/portfolio-
entry/requirement/view?id=" + object.portfolioEntry.id + "&requirementId=" +
object.id, object.author.uid);
    }
  }
```

Example 4

Create a new fake Employee (thanks the BizDock API) when an Initiative is created.

```
  function notify(objectType, objectId, eventType, modifiedAttributes) {
    if (eventType == "CREATE" && objectType == "PortfolioEntry") {
      var
request=scriptUtils.wsCall("https://test.bizdock.io/api/core/actor");
      request.setMethod("POST");
      request.setHeader("X-bizdock-application","86egm-uXiOyks-
iTvPCglZDwu5Cn6oil7LGN8buRl-S1puicheaog-O2nOWalOWOre6rr-
aWgOOmguOOvOmcjfK-
gKzlsafwoLWC6ZmT6Lqz6Lax7JGd47SC8JeTpem_l-W0o_KokLvlnqrlmITqjorijJ7jk73tjL7u
nZDxgZGi5ri04LGI5JeU65CN5Y6w5oKnx5vin5ourqbrtI7qsLzhirrxl7GJ7oyA");
      request.setBody("{\"isActive\": false,\"firstName\":
\"James\",\"lastName\": \"Bond\",\"mail\":
\"james.bond@mi6.uk\",\"employeeId\": \"JB\",\"actorTypeId\":
1,\"managerId\": 1}");
      request.setContentType("application/json");
      request.execute(
        function(response){
          var result=response.getStatus();
          scriptUtils.logEventMessage(false, objectType, objectId,
eventType,"WS response status is : " + result);
        },
        function(errorMessage){
          scriptUtils.logEventMessage(true, objectType, objectId,
```

```
eventType,"WS call error : " + errorMessage);
        }
      );
    }
  }
```

Example 5

Get the value of the custom attribute of an object (UID of the custom attribute us “CUSTOM_ATTRIBUTE_Actor_0”

```
if(objectType=="Actor" && eventType=="UPDATE"){
  var result=scriptUtils.getCustomAttributes(objectType,objectId);
  scriptUtils.logMessage(false,"Response is :
"+result.get("CUSTOM_ATTRIBUTE_Actor_0").name+":
"+result.get("CUSTOM_ATTRIBUTE_Actor_0").value+"\n");
  }
```