Title	Author	Date
Robotics : DevOps command line tool	Rumi	** May 2018**

Description
This tool is an addition to the `Blueprism` eco-system.

1. Background

Up to `Blueprism` version 6.2 there was limited to no feature to integrate the tool properly into a devops pipeline. Here are the major gaps :

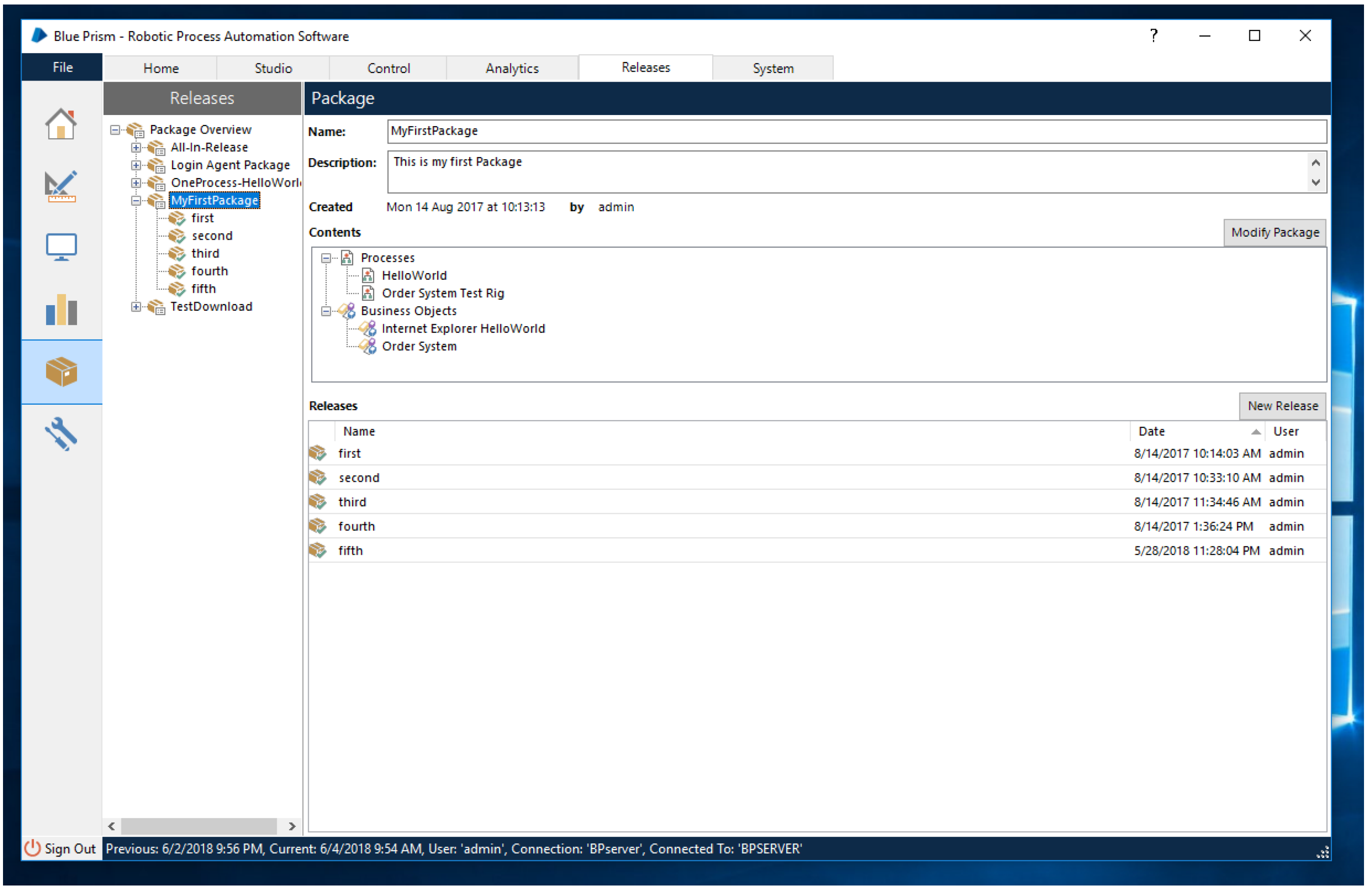
- limited capability to manage code versioning. Whenever the code is updated, it overrides the previous copy.
- There is an audit trail that provides a traceability, old version of code is stored side by side with the new version. This is real a great feature that we can leverage for sure.
- There is a so called release tool, however there is no guarantee that the release includes the intended content. The release is produced at the time the user pushes on the button "Release". However if anyone has modified the code since the "official code" has been last updated, the release will contains the lasted version of the code. There is no mechanism to attach a golden copy of the code to the release package. The system always take the latest version of the code available.
- The releases are produced once and are exported only once in an xml file. There is no mechanism to regenerate the package of any past release.
- The release file generation is changing the code being stored in the database by adding token in the xml that was not in the official code of `blueprism` as well as pretending that the code is built with the latest version while the database still contain the original one. This a risk that one more time there is no traceability to a baseline artefact. Even worse the same code imported and exported in two releases is not the same.

> Note that `blueprism` mentioned a feature helping to address the devops needs in the release 6.2, the only thing available is the ability to run from the command line the import of a release file.

2. Use Cases to consider

2.1. UC #1 : Regenerate the release file from the command line

Let's have a look to the following snapshot of Blueprim, let's pick on Package `MyFirstPackage`



Name: MyFirstPackage

Description: This is my first Package

Created Mon 14 Aug 2017 at 10:13:13 by admin

Contents

- Processes
  - HelloWorld
  - Order System Test Rig
- Business Objects
  - Internet Explorer HelloWorld
  - Order System

Releases

Name	Date	User
first	8/14/2017 10:14:03 AM	admin
second	8/14/2017 10:33:10 AM	admin
third	8/14/2017 11:34:46 AM	admin
fourth	8/14/2017 1:36:24 PM	admin
fifth	5/28/2018 11:28:04 PM	admin

Previous: 6/2/2018 9:56 PM, Current: 6/4/2018 9:54 AM, User: 'admin', Connection: 'BPserver', Connected To: 'BPSERVER'

```
# here is the command to type to get the raw data about the release 'first' from the package
'MyFirstPackage
./bpdevops GetReleaseDetails -n "first" -N "MyFirstPackage"
# getting the release exactly like blueprism will generate the famous release file
./bpdevops GetReleaseDetails -n "first" -N "MyFirstPackage" -f xml
# same as previous but rather than having the data at the console, it stores the data in a file
that can be later on loaded into blueprism
./bpdevops GetReleaseDetails -n "first" -N "MyFirstPackage" -f xml -F test.xml
```

here are the different options to consider for the command `GetReleaseDetails`

```
$./bpdevops GetReleaseDetails --help
bpdevops GetReleaseDetails

-n, --releasename <release name>
-N, --packagename <package name> [-V|--VersionControlEnabled: Provide the full
release

Options:
  --help                        Show help                               [boolean]
  --version                     Show version number                     [boolean]
  -s, --server                  Microsoft SQL Server host name
                                       [string] [required] [default: "localhost"]

  -u, --username                SQL Server valid User Name
```

```
                                                        [string] [default: "test"]
    -p, --password               SQL Server valid User Password
                                                    [string] [default: "changeme"]
    -P, --port                   SQL Server valid User Password
                                                          [number] [default: 1433]
    --format, -f                 output format
                                 [choices: "raw", "json", "xml"] [default: "json"]
    --database, -D               Blueprism database [default: "BluePrismTraining"]
    --File, -F                   output file                       [default: null]
    --loglevel, -l               Log4js Level
                                 OFF|FATAL|ERROR|WARN|INFO|DEBUG|TRACE|ALL
                                                                 [default: "INFO"]
    -n, --releasename            Provide the release name               [required]
    -N, --packagename            Provide the package name               [required]
    -V, --VersionControlEnabled  Make sure the release is built with the relevant
                                 code versioning. the latest code published before
                                 the release was created                 [boolean]
    -R, --releasepath            provide a path where all necessary files for
                                 building the release are located, by default the
                                 release will be derived from blueprism database
                                 rather than file in directory     [default: null]
    -w, --WrapProcessXml         wrap xml process preferredid and xmlns domain
                                 reference              [boolean] [default: false]
```

2.2. UC #2 : Extract any code change in the last <period interval> (e.g. last hour, last 24 hours)

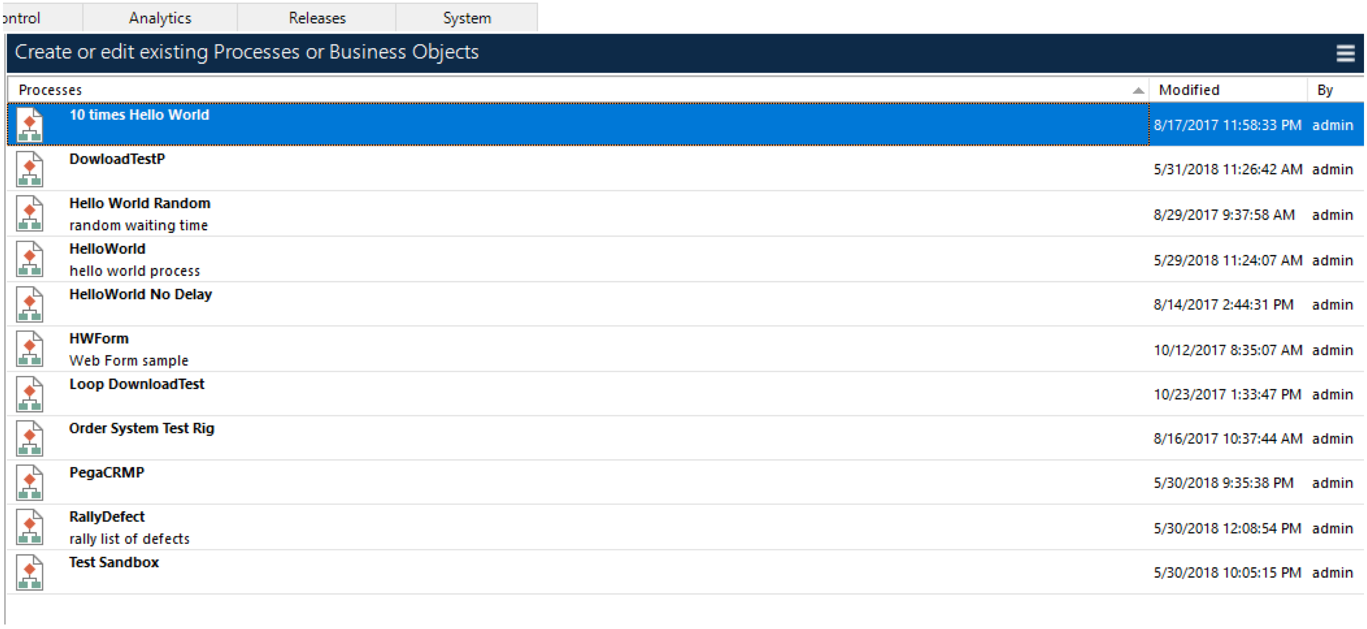
2.2.1. Overall workflow

Let's pretend we are a devops engineer that wants to extract all the recent changes of all the processes and then create a file for each process and then push them all into github. Here is how we will proceed:

1. Get the list of processes that have changed in the last hour, last day or last 7 days (or whatever you like).
2. Iterate for each process in the list and generate a file in xml for each process (the file will have the structure .xml)

2.2.2. Get the list of processes that changed for 7 days, 5 days or one year.

here is the list of recent process changes in `Blueprism`



Create or edit existing Processes or Business Objects

Processes	Modified	By
10 times Hello World	8/17/2017 11:58:33 PM	admin
DowloadTestP	5/31/2018 11:26:42 AM	admin
Hello World Random random waiting time	8/29/2017 9:37:58 AM	admin
HelloWorld hello world process	5/29/2018 11:24:07 AM	admin
HelloWorld No Delay	8/14/2017 2:44:31 PM	admin
HWForm Web Form sample	10/12/2017 8:35:07 AM	admin
Loop DownloadTest	10/23/2017 1:33:47 PM	admin
Order System Test Rig	8/16/2017 10:37:44 AM	admin
PegaCRMP	5/30/2018 9:35:38 PM	admin
RallyDefect rally list of defects	5/30/2018 12:08:54 PM	admin
Test Sandbox	5/30/2018 10:05:15 PM	admin

here is how to get the list of changes

```
# Those commmands have been ran as of 4/6/2018 around noon
$ date
Mon Jun  4 12:48:50 EDT 2018
$ ./bpdevops GetProcessList -A "-7dd" -f raw -l OFF
Test Sandbox;2018-05-30 22:05:15.15
PegaCRMP;2018-05-30 21:35:38.38
DowloadTestP;2018-05-31 11:26:42.42
PegaCRM;2018-05-30 16:44:50.50
RallyDefect;2018-05-30 12:08:54.54
HelloWorld;2018-05-29 11:24:07.07

./bpdevops GetProcessList -A "-5dd" -f raw -l OFF
Test Sandbox;2018-05-30 22:05:15.15
PegaCRMP;2018-05-30 21:35:38.38
DowloadTestP;2018-05-31 11:26:42.42
PegaCRM;2018-05-30 16:44:50.50

./bpdevops GetProcessList -A "-1yy" -f raw -l OFF
Test Sandbox;2018-05-30 22:05:15.15
Loop DownloadTest;2017-10-23 13:33:47.47
PegaCRMP;2018-05-30 21:35:38.38
DowloadTestP;2018-05-31 11:26:42.42
Order System Test Rig;2017-08-16 10:37:44.44
HWForm;2017-10-12 08:35:07.07
RallyDefect;2018-05-30 12:08:54.54
Hello World Random;2017-08-29 09:37:58.58
10 times Hello World;2017-08-17 23:58:33.33
HelloWorld;2018-05-29 11:24:07.07
HelloWorld No Delay;2017-08-14 14:44:31.31
```

2.2.3. Get the most recent copy of a process

Let get the latest copy of a particular process

```
# The process name in blueprism is HelloWorld
./bpdevops GetProcess -n HelloWorld -f xml -l OFF
```

```
<process name="HelloWorld" version="1.0" bpversion="6.2.0.2669" narrative="hello world
process">
  <view>
    <camerax>0</camerax>
    <cameray>0</cameray>
    <zoom version="2">1.25</zoom>
  </view>
  <preconditions>
    <condition narrative="No special Preconditions" />
  </preconditions>
  <endpoint narrative="No special postconditions" />
  <stage stageid="f25d3168-3e43-40a3-ab8a-dbfdab308334" name="Start" type="Start">
    <narrative></narrative>
    <displayx>15</displayx>
    <displayy>-105</displayy>
    <displaywidth>60</displaywidth>
    <displayheight>30</displayheight>
    <font family="Segoe UI" size="10" style="Regular" color="000000" />
    <onsuccess>aac21e24-ef74-4c97-8ea2-c775cb054dac</onsuccess>
  </stage>
  <stage stageid="bc7e8d0e-7bfb-408f-a09a-a0eb412a05cc" name="End" type="End">
    <narrative></narrative>
    <displayx>15</displayx>
    <displayy>135</displayy>
    <displaywidth>60</displaywidth>
    <displayheight>30</displayheight>
    <font family="Segoe UI" size="10" style="Regular" color="000000" />
  </stage>
  <stage stageid="1aefa1aa-4ca5-476c-9d95-aca3766501da" name="Stage1" type="ProcessInfo">
    <narrative></narrative>
    <displayx>-195</displayx>
    <displayy>-105</displayy>
    <displaywidth>150</displaywidth>
    <displayheight>90</displayheight>
    <font family="Segoe UI" size="10" style="Regular" color="000000" />
  </stage>
  <stage stageid="aac21e24-ef74-4c97-8ea2-c775cb054dac" name="Open Explorer on javascript Rest
API" type="Action">
    <loginhibit />
    <narrative></narrative>
    <displayx>15</displayx>
    <displayy>-45</displayy>
    <displaywidth>60</displaywidth>
    <displayheight>30</displayheight>
    <font family="Segoe UI" size="10" style="Regular" color="000000" />
    <onsuccess>5672072e-6851-4904-bdfe-81561955c4aa</onsuccess>
    <resource object="Internet Explorer HelloWorld" action="OpenExplorer" />
  </stage>
  <stage stageid="c0b06897-9af2-4683-a3fa-70c3c2d9b961" name="Terminate Explorer"
type="Action">
    <narrative></narrative>
    <displayx>15</displayx>
    <displayy>75</displayy>
    <displaywidth>60</displaywidth>
    <displayheight>30</displayheight>
    <font family="Segoe UI" size="10" style="Regular" color="000000" />
    <onsuccess>bc7e8d0e-7bfb-408f-a09a-a0eb412a05cc</onsuccess>
    <resource object="Internet Explorer HelloWorld" action="Terminate Explorer" />
  </stage>
  <stage stageid="5672072e-6851-4904-bdfe-81561955c4aa" name="Pause " type="Action">
    <narrative></narrative>
    <displayx>15</displayx>
    <displayy>15</displayy>
    <displaywidth>60</displaywidth>
    <displayheight>30</displayheight>
    <font family="Segoe UI" size="10" style="Regular" color="000000" />
    <onsuccess>c0b06897-9af2-4683-a3fa-70c3c2d9b961</onsuccess>
    <resource object="Internet Explorer HelloWorld" action="Pause" />
  </stage>
</process>

# get a copy into a file, same as before excepting that the result is stored in a file called
'helloworld.xml"
./bpdevops GetProcess -n HelloWorld -F helloworld.xml -f xml
# get same as before excepting that it adds the attribute preferredid and xmlns like it does
when a process is exported from blueprism
./bpdevops GetProcess -n HelloWorld -F helloworld.xml -f xml -w
```

2.2.4. Building a combine script

> Note : the tool has been built with the intend to be independent of the devops tools. This is the whole purpose to have a commandline tool rther than a fully integrated solution based on COTS.

Build the script that leverages the two previous sample on `GetProcessList` and `GetProcess`.

The following script when invoked, will create a file by process that has changed in the last <time interval>

```
# Create a file for each process that has been modified in the last 7 days
./GetDelta.sh -A "-7dd" -C
```

Here is the script a devops engineer may end up writing :

```
#!/bin/bash
# Rumi 2018

# manage parameters

usage()
{
    echo "usage: $0 [-A <datepart>| [-h]] [-C]"
}

while [ "$1" != "" ]; do
    case $1 in
        -A | --audit   )        shift
                                DATEPART=$1
                                ;;
        -C | --create-file)     CREATEFILE="true"
                                ;;
        * )                     usage
                                exit 1
    esac
    shift
done

# Global Declaration

COMMAND=./bpdevops

# capture the list of changes

processlist=$($COMMAND GetProcessList -A $DATEPART -f raw -l OFF| awk -F ";" '{print $1}')
IFS=$'\n'

for PROCESS in $processlist
do
  echo "Process $PROCESS in progress ..."
  if [[ -n "$CREATEFILE"  ]]
  then
        $COMMAND GetProcess -n $PROCESS -f xml -l OFF  > ./$PROCESS.xml
  else
        $COMMAND GetProcess -n $PROCESS -f xml -l OFF

  fi
done
```

2.3. UC #3 : Push to github the latest change captured in the last interval

This use case is about pushing to git the most recent change that has happened on any process or object in the last period of time (e.g. last hour, last 5min). Let's have an example

```
# this script will leverage the tool bpdevops already created :
#   1. Extract from blueprism the changes affecting any  process or object in the last 5min
#   2. Copy the object or process that has changed into the git repository with a file <name or
object or process>.xml.
#      in essence it will override the previous copy if any
#   3. push the change to git
# only one entry is created in git over the period being scanned (here 5 min).
./GetDeltaPushtoGit.sh -A "-5mi" -C <complete path to git workspace>/git/bpdevops_src/
```

Here is the script a devops engineer may end up writing :

```
#!/bin/bash -x
# Rumi 2018
# manage parameters

usage()
{
    echo "usage: $0 [-A <datepart>| [-h]] [-C <path where to create files>]"
}

while [ "$1" != "" ]; do
    case $1 in
        -A | --audit   )       shift
                                DATEPART=$1
                                ;;
        -C | --create-file)     CREATEFILE="true"
                                shift
                                GITPATH=$1
                                ;;
        * )                     usage
                                exit 1
    esac
    shift
done

# Global Declaration

COMMAND=./bpdevops

# capture the list of changes

NOTE=''
processlist=$($COMMAND GetProcessList -A $DATEPART -f raw -l OFF| awk -F ";" '{print $1}')
IFS=$'\n'
for PROCESS in $processlist
do
  if [[ -n "$CREATEFILE"  ]]
  then
    NOTE=$(echo -en "$NOTE" "`$COMMAND GetProcessAudit -n $PROCESS -A $DATEPART -f xml -l OFF
-F $GITPATH$PROCESS.xml`" )
  else
        $COMMAND GetProcess -n $PROCESS -f xml -l OFF
  fi
done
git -C $GITPATH add .
git -C $GITPATH commit -am "Blueprism Autocommit from last delta$NOTE"
git -C $GITPATH push
```

2.3.1. Example : Forever loop or cron

In case you would like to implement the process running in the background, you can either run a forever loop like 👍

```
#!/bin/bash
# Rumi 2018
while true; do
echo 'Start punlishing' `date`
./GetDeltaPushtoGit.sh -A "-1hh" -C /Users/rumi/git/bpdevops_src/
 sleep 300 # 300 sec or 5minutes
 done
```

2.4. UC #4 : Push any change to github

> Note : in this section we will provide an example of a script to get all changes that happened let's for 1 year (this can be configured). However, as the number of change can be significant, a paging feature is available to bach the changes and manage any scalability or memory issue. The first example does not leverage the paging. Without the paging approach the resulting dataset is a json which drives a slow navigation if being too big.

2.4.1. Leveraging GetProcessAudit with no paging capability

This use case is about pushing any change being audited by blueprism into github. In other words, anytime someone saves an object or a process and add a comment, it will be saved separately to git with the comment attached to git audit trail. This will work as much as the audit trail of blueprism allows it. If the audit trail has never been purged, it should be able to recover all changes made since the beginning and push them all into git.

```
# this script will leverage the tool bpdevops already created :
#   1. Extract from blueprism the changes affecting any  process or object in the last year.
#   2. Copy the object or process that has changed into the git repository with a file <name or
object or process>.xml and attached the justification of the change to git
#   3. push the change to git
# The difference with UC #4 is that if the process or object have changed many times during the
period, separate entry will be created in git.
 ./GetAllChangesPushtoGit.sh -A "-1yy" -C /Users/rumi/git/bpdevops_test1/
```

Here is the script a devops engineer may end up writing `GetAllChangesPushtoGit.sh` :

```
#!/bin/bash

# Rumi 2018

# manage parameters

usage()
{
    echo "usage: $0 [-A <datepart>| [-h]] [-C <path where to create files>]"
    }
info()
{
    echo "[104m$1[49m"
}
highlight()
{
    echo "[100m$1[49m"
}


while [ "$1" != "" ]; do
    case $1 in
        -A | --audit   )        shift
                                DATEPART=$1
                                ;;
        -C | --create-file)     CREATEFILE="true"
                                shift
                                GITPATH=$1
                                ;;

        * )                     usage
                                exit 1

    esac
    shift
done
 if [[ -z "$CREATEFILE"  ]]
  then
  usage
  exit 1
fi
# Global Declaration

COMMAND=./bpdevops

# capture the list of changes
set TERM="VT100"
NOTE=''
processlist=$($COMMAND GetProcessList -A $DATEPART -f raw -l OFF| awk -F ";" '{print $1}')
IFS=$'\n' # don't remove this otherwise it will affect the whole process

for PROCESS in $processlist
```

```
do
    # tput sc
    PROCESSFILENAME=$(echo $PROCESS | sed 's/[\/ ]/_/g') # replace any '\' and SPACE by '-' in
order to construct a consistent file name

    NOTE=$($COMMAND GetProcessAudit -n $PROCESS -A $DATEPART -l OFF ) # process versions are
ordered from oldest to most recent
    COUNTER=$(unset IFS; echo $NOTE | jq -r '.recordset | length')
    DONE=0
    info "Number of processes : $COUNTER"
    while [  $COUNTER -gt 0 ]; do
       # tput rc


             # extract to audit record associated to the code change
             CURRENTRECORD=$(unset IFS ;echo $NOTE | jq -r ".recordset[${COUNTER}-1]")

             (unset IFS;echo "$CURRENTRECORD") | jq -r ".processxml" >
$GITPATH$PROCESSFILENAME.xml

             EDITSUMMARY=$(unset IFS;echo $CURRENTRECORD | jq -r ".EditSummary")
             USER=$(unset IFS ;echo $CURRENTRECORD | jq -r ".username")
             DATE=$(unset IFS ;echo $CURRENTRECORD | jq -r ".eventdatetime")
             OBJECT=$(unset IFS ;echo $CURRENTRECORD  | jq -r ".Object")
             AUDITNOTE=$(echo "User: " $USER " - DATE: " $DATE " - OBJECT : " $OBJECT " - NOTE:
" $EDITSUMMARY)
             highlight "Audit added for : $AUDITNOTE"
             #push the change to git
             info "GIT ADD Lasted    : $({ time git -C $GITPATH add . ; } 2>&1 | grep -i real |
awk '{print $2}')"
             info "GIT COMMIT Lasted : $({ time git -C $GITPATH commit -am "$AUDITNOTE" ; }
2>&1 | grep -i real | awk '{print $2}')"
             info "GIT PUSH LASTED   : $({ time git -C $GITPATH push ; } 2>&1 | grep -i real |
awk '{print $2}')"

             let COUNTER-=1
             let DONE+=1
             info "Process changes : ${PROCESS} : Pushed : $DONE"

    done
    info "Process done :  ${PROCESS}"

done
```

2.4.2. Leveraging GetProcessAudit with paging

By levergacing the paging capability of the `GetProcessAudit` command, it allows to manage the process in a most efficient manner. The reason for it is very simple, when testing the previous script on an existing blueprism sandbox, it was highlighted that when a code may have changed let say 254 times (this was the real case by the way 😁), it can create as may changes to push to cithub in a form of a big json containing each and evrery version of the code. As the scripting will usually used `jq` (or something similar) to navigate the json, it can either cause performance issue or memroy issue.

You can use the script exacly as the one before with the paging optimizatioon, the name of the cript is `GetAllChangesPushtoGit-Paging.sh`

2.5. UC #5 : Extract full blueprism release and push to corresponding github branch

Here is what we would like to achieve :

1. Make sure you have a repository created on github and the workspace cloned on your machine
2. A blueprism release has been created as above (Packagename: MyFirstPackage, Releasename : first)

3. run the script `./GetPushReleasetoGit.sh -C <path to git workspace>/git/bpdevops_releases/ -n fifth -N MyFirstPackage`
    - The script will create a github branch in your workspace with the name `MyFirstPackage-first`
    - Extract the whole content of the corresponding blueprism release into the workspace.
    - Push the blueprism release into github

here is the shell script achieving that :

```
<same header as before>
COMMAND=./bpdevops

# capture the list of changes
set TERM="VT100"
RELEASEINFO=$(unset IFS;$COMMAND GetReleaseInfo -n $RELEASENAME -N $PACKAGENAME| jq -r
'del(.recordsets[])')
processlist=$(unset IFS;echo $RELEASEINFO| jq '.recordset[1][].name')
IFS=$'\n' # don't remove this otherwise it will affect the whole process

COUNTER=$(unset IFS; echo $RELEASEINFO | jq -r '.recordset[1] | length')
DONE=0
echo "Number of processes : $COUNTER"
# if branch already exist : delete it
if [ $(git -C $GITPATH branch | grep -i $PACKAGENAME-$RELEASENAME | wc -l) -eq 1 ]
then
    info "Branch $PACKAGENAME-$RELEASENAME Already exist"
    info "Switch to master branch before delete $(GIT -C $GITPATH checkout master)"
    info "Delete branch : $PACKAGENAME-$RELEASENAME $(GIT -C $GITPATH branch -d
$PACKAGENAME-$RELEASENAME)"
    info "Push delete $(GIT -C $GITPATH push origin :$PACKAGENAME-$RELEASENAME)"
fi
info "GIT Checkout Lasted    : $(git -C $GITPATH  checkout -b $PACKAGENAME-$RELEASENAME)"

while [  $COUNTER -gt 0 ]; do

    PROCESS=$(unset IFS ;echo $RELEASEINFO | jq -r ".recordset[1][${COUNTER}-1].name")

    PROCESSFILENAME=$(echo $PROCESS | sed 's/[\/ ]/_/g')

    highlight "Process in progress :  ${PROCESS}"
    # extract to audit record associated to the code change
    CURRENTRECORD=$(unset IFS ;echo $RELEASEINFO | jq -r ".recordset[1][${COUNTER}-1]")
    highlight "Create $GITPATH$PROCESSFILENAME.xml lasted : $(unset IFS ;{ time echo
$CURRENTRECORD | jq -r ".processxml" > $GITPATH$PROCESSFILENAME.xml ; } 2>&1 | grep -i real |
awk '{print $2}')"
    let COUNTER-=1
    let DONE+=1

done
highlight "Create Release info $GITPATH$PACKAGENAME$PACKAGENAME-$RELEASENAME.json $(unset
IFS;echo $RELEASEINFO|jq -r 'del(.recordset[1][].processxml)' >
$GITPATH$PACKAGENAME$PACKAGENAME-$RELEASENAME.json)"


info "GIT ADD Lasted    : $(git -C $GITPATH add .)"
info "GIT COMMIT Lasted : $(git -C $GITPATH commit -am "Automated built")"
git -C $GITPATH  push --set-upstream origin $PACKAGENAME-$RELEASENAME
```

2.6. UC #6 : Generating Blueprism release from github/file system (enabling better patching mechanism)

> Note : today once a release has been generated, it is quite difficult from the same blueprism instance to create a patch of a particular release.

```
./bpdevops GetReleaseDetails --help
bpdevops GetReleaseDetails

-n, --releasename <release name>
-N, --packagename <package name> [-V|--VersionControlEnabled: Provide the full
release

Options:
  --help                       Show help                                 [boolean]
  --version                    Show version number                       [boolean]
  -s, --server                 Microsoft SQL Server host name
                                      [string] [required] [default: "localhost"]
  -u, --username               SQL Server valid User Name
                                                      [string] [default: "test"]
  -p, --password               SQL Server valid User Password
                                                  [string] [default: "changeme"]
  -P, --port                   SQL Server valid User Password
                                                     [number] [default: 1433]
  --format, -f                 output format
                               [choices: "raw", "json", "xml"] [default: "json"]
  --database, -D               Blueprism database [default: "BluePrismTraining"]
  --File, -F                   output file                         [default: null]
  --loglevel, -l               Log4js Level
                               OFF|FATAL|ERROR|WARN|INFO|DEBUG|TRACE|ALL
                                                               [default: "INFO"]
  -n, --releasename            Provide the release name                 [required]
  -N, --packagename            Provide the package name                 [required]
  -V, --VersionControlEnabled  Make sure the release is built with the relevant
                               code versioning. the latest code published before
                               the release was created                   [boolean]
  -R, --releasepath            provide a path where all necessary files for
                               building the release are located, by default the
                               release will be derived from blueprism database
                               rather than file in directory     [default: null]
  -w, --WrapProcessXml         wrap xml process preferredid and xmlns domain
                               reference              [boolean] [default: false]

# Leveraging the option -R which will generate the release from git workplae rather than
blueprism
# the command will levegate the meta data saved before when release was extracted from
blueprism
# in addition the command will read all files for the objects and processes containted in the
release.
# this will even allow patching a release easily which is not easy today.

./bpdevops GetReleaseDetails -n first -N MyFirstPackage -R /Users/rumi/git/bpdevops_releases/
```

3. Version control

> Note that Blueprism up to version 6.2 does no have any source control mechanism. However we can leverage the audit table as well as creating a simple copy of the current table where the code is stored. This way we will have a mechanism to extract the different versions from blueprism. The reason to create a copy of the table is that the audit does not stored the `BPAProcess` record information but only the xml, good for the future to keep track of the source table, the one really being edited from the studio and the one being leveraged by the engine running the processes. Furthermore, it is that simple to stored the copy without impacting the operation and the ability of the plaform to operate, it just takes disk space that can be managed depending on everyone's strategy.

Here is the trigger that will enable to keep track of all details related to processes and objects :

```
-- at this point we will no consider adding any index yet.
-- let's create the copy of BPAProcess the simplest way
```

```
select * into BPAProcess_devops
from BPAProcess
```

Now we have `BPAProcess_devops` which has the same structure as `BPAProcess`, what the trigger will achieve next is that rather than keeping only one record per process, we are going to keep them all, having all subsequent updates. In other words each time someone will save a process in blueprism, a copy will be persisted in the history table `BPAProcess_devops`. Then it will enable version control and release to be rebuild on demand.

```
DROP TRIGGER BluePrismTraining.dbo.BPAProcess_devops_tri
GO


create trigger BPAProcess_devops_tri on BPAProcess
after update, insert
as
begin
  insert into BPAProcess_devops
  (processid,
ProcessType,
name,
description,
version,
createdate,
createdby,
lastmodifieddate,
lastmodifiedby,
AttributeID,
compressedxml,
processxml,
wspublishname,
runmode,
sharedObject,
forceLiteralForm,
useLegacyNamespace
)
  select t.processid,
t.ProcessType,
t.name,
t.description,
t.version,
t.createdate,
t.createdby,
t.lastmodifieddate,
t.lastmodifiedby,
t.AttributeID,
t.compressedxml,
t.processxml,
t.wspublishname,
t.runmode,
t.sharedObject,
t.forceLiteralForm,
t.useLegacyNamespace
  from  BPAProcess t
  inner join inserted i on t.processid=i.processid
end
GO
```

4. Commands available

The tool expose itself through command line interactions, here is the help that explain what the tool offers:

```
./bpdevops --help
bpdevops [command]

Commands:
  bpdevops GetReleaseInfo         -n, --releasename <release name>
                                  -N, --packagename <package name>: Provide
                                  information about a specific release
  bpdevops GetProcess             -n, --processname <process name> : Provide
                                  information about a particular process
  bpdevops GetReleaseDetails      -n, --releasename <release name>
                                  -N, --packagename <package name>
                                  [-V|--VersionControlEnabled: Provide the full
                                  release
  bpdevops GetPackageList         Provide the list of blueprism packages
  bpdevops GetProcessList         [-A | -audit <number><date part>] Provide the
                                  list of blueprism processes
  bpdevops GetProcessFromHistory  [-A | -audit <number><date part>] -n,
                                  --processname <process name> : Provide the
                                  right version of the process
  bpdevops GetProcessAudit        [-A | -audit <number><date part>] -n,
                                  --processname <process name> : Provide the
                                  right version of the process
  bpdevops GetProcessInRelease    -n, --releasename <release name>
                                  -N, --packagename <package name> : Provide the
                                  list of process in a specific release <package
                                  name>
  bpdevops GetGroupReleaseInfo    -n, --releasename <release name>
                                  -N, --packagename <package name> : Provide
                                  information about the groups in a specific
                                  release <package name>
  bpdevops GetVariablesInRelease  -n, --releasename <release name>
                                  -N, --packagename <package name>  : Provide
                                  the list of variables in a specific release
                                  <package name>

Options:
  --help          Show help                                            [boolean]
  --version       Show version number                                  [boolean]
  -s, --server    Microsoft SQL Server name
                                      [string] [required] [default: "localhost"]
  -u, --username  SQL Server valid User Name          [string] [default: "test"]
  -p, --password  SQL Server valid User Password  [string] [default: "changeme"]
  -P, --port      SQL Server valid User Password          [number] [default: 1433]
  --format, -f    output format[choices: "raw", "json", "xml"] [default: "json"]
  --database, -D  Blueprism database              [default: "BluePrismTraining"]
  --File, -F      output file                                    [default: null]
  --loglevel, -l  Log4js Level OFF|FATAL|ERROR|WARN|INFO|DEBUG|TRACE|ALL
                                                               [default: "INFO"]
```

As you can see there are two parts, command and options, here is an example on how to get the specifics options for a particular command :

```
./bpdevops GetProcessAudit --help
bpdevops GetProcessAudit

Provide the list of changes that happen during the last period, as an example,
if the code of a object called HelloWorld has hanged 10 times, this command will
provide the 10 versions of the code with their audit trail. As this can become
very heavy if someone try to get all changes since the beginning of time, a
paging techniques has been utilized as a potential option to navigate the long
list of changes.

Options:
  --help             Show help                                         [boolean]
  --version          Show version number                               [boolean]
  -s, --server       Microsoft SQL Server name
                                      [string] [required] [default: "localhost"]
  -u, --username     SQL Server valid User Name       [string] [default: "test"]
  -p, --password     SQL Server valid User Password
                                                  [string] [default: "changeme"]
  -P, --port         SQL Server valid User Password     [number] [default: 1433]
```

```
    --format, -f       output format
                                 [choices: "raw", "json", "xml"] [default: "json"]
    --database, -D     Blueprism database           [default: "BluePrismTraining"]
    --File, -F         output file                                 [default: null]
    --loglevel, -l     Log4js Level OFF|FATAL|ERROR|WARN|INFO|DEBUG|TRACE|ALL
                                                                 [default: "INFO"]
    -A, --audit        Provide the list of processes modified and the the list of
                       audit item for the period passed as argument, the processes
                       in scope are the one following transact-SQL following query
                       BPAAudit.eventdatetime > datedd(DATEADD(<date part>,
                       <signed number>, getdate())             [string] [required]
    -n, --processname  Provide the processname                          [required]
    -i, --pagenumber   provide the page number to load, default value is 0 it
                       means no paging approach, This allow loading the different
                       versions per page as opposed to infinite list
                                                             [number] [default: 0]
    -I, --pagesize     propvide the page size                [number] [default: 5]
    -d, --details      provide all details including the xml code otherwise does
                       not attach the xml in order to reduce the size of xml
                                                         [boolean] [default: true]

  Examples:
    GetProcessAudit  bpdevops GetProcessAudit -A -90mi
                     bpdevops GetProcessAudit -f raw -A -90mi -F output.txt
```