Customizing the BPM

You can create a custom BPM by forking the Kibo-Fulfillment-Workflows repository of your codebase and creating a new BPM process. This eliminates the need to submit a request for a custom BPM to Kibo Support and wait for it to be created within the development cycle. This new BPM is uploaded and installed through the Kibo Dev Center, enabled through API, and then executed via the Kibo Fulfiller UI.

This guide provides the tools for a developer to get started, though you can reference the ReadMe file within the repository at `/docs/How-to-extend-Kibo-Fulfillment-workflows.md` for convenience during development or additional updates.

This documentation uses the jBPM Business Central application for authoring and testing BPM workflows locally. Alternatively, you can use an Integrated Development Environment (IDE) such as Eclipse which is documented at jbpm.org. Click **Read Documentation** on the jBPM home page and search the referenced document for **Eclipse Developer Tools** to get more details.

Additionally, since custom BPMs are implemented on a separate fork of Kibo's fulfillment workflows, that means that any future enhancements Kibo may add to the default BPM will not be reflected on the fork. In this case, you will have to code the changes into your version of the forked BPMs in order to add them to the new fulfillment workflows.

Step 1: Set Up jBPM with Business Central

The jBPM Server distribution is the easiest way to start with jBPM, as the included Business Central application is useful for authoring processes. To get up and running quickly, use the jBPM single distribution which can be downloaded at jbpm.org. Look at the Getting Started guide to get yourself familiar with Business Central.

By default, Business Central is available here.

Step 2: Create a Kibo Application

Create a Kibo Dev Center application for remote synchronization:

1. Install node.js and npm via download or an operating system specific package manager.
2. Install the Yeoman command line tool by typing `npm install -g yo`
3. Install the Mozu Actions Generator and Grunt command line tools by typing `npm install -g generator-mozu-actions grunt-cli`
4. Request a Kibo Dev Center Account and Access Credentials.
5. Log in to the Dev Center, then locate and record your account ID.
   - For example, Name: `Your Developer Account` and Account ID: `9999`
6. Double-click the Developer Account entry to enter the Developer Account Console.
7. Click the **Develop** pull-down and select **Applications**.
8. Click the **Create Application** button.
9. Supply a valid **Name** and **Application ID**, such as `YOUR_DEVCENTER_APP_NAME`
10. Click **Save**.
11. Double-click your new application within the displayed list of applications. For example, an Dev Center application URI may look like:
    https://developer.mozu.com/console/app/edit/YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAM
12. Within the Application interface, click the **Packages** tab on the left-side of the screen, select

the **Capabilities** tab, and then click **Add Capability**.

13. Select the **Fulfillment Business Process Workflow** capability.
14. Record the application key, such as `YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME.1.0.0.Release`

Step 3: Fork the Fulfillment Workflows Repository

Forking the repository is a simple two-step process:

1. On GitHub, navigate to the Kibo Fulfillment Workflows repository.
2. In the top-right corner of the page, click **Fork**.

You can optionally rename your fork to match a desired Kibo application name:

1. On GitHub, navigate to your forked repository and select **Settings.**
2. Enter the desired name under **Repository name** and then click the **Rename** button.
    - For example: `YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME`

Keep Your Fork Synchronized

It's a good practice to regularly synchronize your fork with the upstream repository. To achieve this, you'll need to use Git via the command line by following the below steps:

1. Set Up Git
2. Create a Local Clone of Your Fork
3. Configure Git to Synchronize with the Original Repository
4. Make Changes to the Fork

Set Up Git

If you haven't yet, first set up Git. Don't forget to set up authentication to GitHub from Git as well.

Create a Local Clone of Your Fork

Right now, you have a fork of the Kibo Fulfillment Workflows repository on GitHub, but you don't have the files in that repository on your computer. Let's create a clone of your fork locally on your computer.

1. On GitHub, navigate to your fork of the repository.
2. Under the repository name, click **Code** and then the desired **Clone** or **Download** option.
3. To clone the repository using HTTPS, click the clipboard icon under **Clone with HTTPS**. To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **Use SSH** and then click **Clone URL**.
4. Open Terminal.
5. Type `git clone`, and then paste the URL you copied earlier. It will look like this, with your GitHub username instead of `YOUR_USERNAME`:

    ```
    $ git clone https://github.com/YOUR_USERNAME/YOUR_FORK
    ```

6. Press **Enter**. Your local clone will be created.

    ```
    $ git clone https://github.com/YOUR_USERNAME/YOUR_FORK
    > Cloning into `YOUR_FORK`...
    > remote: Counting objects: 1033, done.
    > remote: Total 1033 (delta 0), reused 0 (delta 0), pack-reused 1033
    > Receiving objects: 100% (1033/1033), 1.22 MiB | 173.00 KiB/s, done.
    > Resolving deltas: 100% (405/405), done.
    ```

Configure Git to Synchronize with the Original Repository

When you fork a project, you can configure Git to pull changes from the original (or upstream) repository into the local clone of your fork.

1. On GitHub, navigate to the Kibo Fulfillment Workflows repository.
2. Under the repository name, click **Code** and then the desired **Clone** or **Download** option. To clone the repository using HTTPS, click the clipboard icon under **Clone with HTTPS**. To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **Use SSH** and then click **Clone URL**.
3. Open Terminal.
4. Change directories to the location of the fork you cloned in Create a Local Clone of Your Fork.
   - To go to your home directory, type just `cd` with no other text.
   - To list the files and folders in your current directory, type `ls`.
   - To go into one of your listed directories, type `cd your_listed_directory.`
   - To go up one directory, type `cd ..`
5. Type `git remote -v` and press **Enter**. You'll see the current configured remote repository for your fork.

   ```
   $ git remote -v
   > origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
   > origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
   ```

6. Type `git remote add upstream` , paste the URL you copied, and press **Enter**. It will look like this:

   ```
   $ git remote add upstream https://github.kibocommerce.com/KiboSoftware/kibo-fulfillment-workflows.git
   ```

7. To verify the new upstream repository you've specified for your fork, type `git remote -v` again. You should see the URL for your fork as origin, and the URL for the original Kibo Fulfillment Workflows repository as upstream.

   ```
   $ git remote -v
   > origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
   > origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
   > upstream  https://github.kibocommerce.com/KiboSoftware/kibo-fulfillment-workflows (fetch)
   > upstream  https://github.kibocommerce.com/KiboSoftware/kibo-fulfillment-workflows (push)
   ```

Make Changes to the Fork

You have the flexibility to make various changes to your fork, which includes creating and opening branches. You will have to synchronize your custom fork with the upstream repository as well as with your jBPM Business Central repository.

- **Creating Branches:** Branches allow you to build new features or test out ideas without putting your main project at risk.
- **Opening Pull Requests:** If you are hoping to propose a change to the original repository, you can send a request to Kibo to pull your fork into their repository by submitting a pull request.

Step 4: Modify Forked Repository Files

To update your pom.xml file:

1. Modify **pom.xml** by changing the following elements to match your project requirements:

```
<groupId>YOUR_DEVCENTER_ACCOUNT_KEY</groupId>
<artifactId>YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME</artifactId>
<version>1.0.0-SNAPSHOT</version>
<packaging>kjar</packaging>
<name>YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME</name>
```

2. Commit changes to your local copy of the forked repository.

```
$ git add pom.xml
$ git commit -m "Provide a meaningful commit message here"
$ git push origin develop
```

Step 5: Import Assets into Business Central

You can easily import the forked business assets project into Business Central, as it's a valid Git repository:

1. Create a git branch named **master** from the default **develop** branch.

```
$ git checkout -b master
Switched to a new branch 'master'
---
$ git branch
  develop
* master
```

> The assumption here is that there's no existing `master` branch in your forked repository. The name `master` is used to align with the default branch name used by the development jBPM instance.

2. Log in to Business Central and go to **Menu** > **Design** > **Projects**.
3. Select `Import Project` from the Add Project menu and enter the filesystem location of the project git repository within the `Repository URL` field. For example:

```
file://{filesystem location of forked repository}
```

4. Click **Import**, confirm the project to be imported, and click **Ok.**

> Note that if attempting upload within a Docker container, a volume must be mapped.

5. Once the business assets project is successfully imported into Business Central, you can begin working on it. Navigate to the project and make additions or modifications to assets such as business processes, forms, rules, decision tables, and more.

Step 6: Pull Custom Assets to the Fork

Updated business assets need to be pulled back to the forked project source code repository:

1. Go to Settings of the project within Business Central.
2. Copy the **URL** value from the **General Settings** view.
3. Go to the filesystem location of the forked and imported repository.
4. Type `git remote -v` and press **Enter**. You'll see the current configured remote repositories.

```
$ git remote -v
> origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
> origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
> upstream https://github.com/KiboSoftware/kibo-fulfillment-workflows.git (fetch)
> upstream  https://github.com/KiboSoftware/kibo-fulfillment-workflows.git (push)
```

6. Type `git remote add jbpm`, and then paste the URL you copied in Step 2. Modify the value to include `wbadmin@` and press **Enter**. It will look like this:

```
$ git remote add jbpm ssh://wbadmin@localhost:8001/MySpace/YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME
```

7. To verify the new **jbpm** repository you've specified for your fork, type `git remote -v` again. You should see the URL for the jBPM Business Central project as **jbpm**, the URL for your fork as **origin**, and the URL for the original repository as **upstream**.

```
$ git remote -v
 > jbpm    ssh://wbadmin@localhost:8001/MySpace/YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME (fetch)
 > jbpm    ssh://wbadmin@localhost:8001/MySpace/YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME (push)
 > origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
 > origin    https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
 > upstream  https://github.com/KiboSoftware/kibo-fulfillment-workflows.git (fetch)
 > upstream  https://github.com/KiboSoftware/kibo-fulfillment-workflows.git (push)
```

8. Pull or fetch your custom business assets from jBPM Business Central to your forked git repository.

```
$ git checkout master
$ git pull jbpm master - when prompted enter wbadmin as password
```

or

```
$ git checkout master
$ git fetch jbpm
$ git rebase jbpm/master
```

If you encounter issues connecting to the jBPM generated Git repository over SSH, you can change the protocol to **http** within the same Business Central **Settings** view for your project.

9. Synchronize the **develop** branch of your fork with the **origin** repository on GitHub.

```
$ git checkout develop
$ git pull origin develop
```

10. Rebase your updated local **master** branch commits on the synchronized **develop** branch.

```
$ git checkout master
$ git rebase develop
```

11. Squash all dedicated jBPM Business Central changes in the **develop** branch of your fork.

```
$ git checkout develop
$ git merge --squash master
```

12. Add & commit the merged changes to the **develop** branch and then push to your fork on GitHub.

```
$ git add -A
$ git commit -m "some useful comment"
$ git push origin develop
```

13. Reset the jBPM Business Central **master** branch using the updated **develop** branch.

```
$ git checkout master
$ git reset --hard develop
$ git push -f jbpm master
```

14. With your custom business assets now part of the forked project source tree, Maven commands can be used to build and publish the KJAR artifact to a Maven repository without using the standalone jBPM server.

```
$ mvn clean install
```

Step 7: Deploy Custom Assets to KIE Server

After adding assets to your project in Business Central, you can easily deploy it to a running KIE server instance:

1. Navigate to your project and click **Deploy**.
2. After a few seconds, you should see the project successfully deployed.

Step 8: Interact with Deployed Assets

You can use **Process Definitions** and **Process Instances** perspectives of Business Central to interact with your newly deployed business assets, such as processes or user tasks.

Step 9: Set Up Application Scaffolding

Follow these steps to create the necessary directory structure and configure your application for development:

1. Create a `devcenter-app` subdirectory within the root of your forked project directory. For example, use the command: `mkdir -p /Users/YOUR_USERNAME/Projects/Kibo-Applications/Fulfillment/YOUR_APPLICATION_NAME/devcenter-app`
2. Change your current working directory to the `devcenter-app` subdirectory. For example, use the command: `cd /Users/YOUR_USERNAME/Projects/Kibo-Applications/Fulfillment/YOUR_APPLICATION_NAME/devcenter-app`
3. Create the DevCenter application scaffolding. Type `yo mozu-actions` and then answer prompts specific to your application. For example:

```
$ yo mozu-actions

 ,,,,,,,,,,,,,,,,,      ,,,,,,    ,,,,,,,,,,,  ,,,     ,,,
 SSSSSSSSSSSSSSSSSSSQ   ;QSSSSSSSSSQ,  SSSSSSSSSSS  SSSQ   ]SSSQ
 SSS#"""""@SSS"""""YSSSQ  SSSS""""""QSSS  ^T777T@SSS#^  SSSQ   ]SSSQ
 SSS[    @SSS     SSS[  SSSb     ]SSS     ,QSSSN    SSSQ    ]SSSQ
 SSS[    @SSS     SSS[  SSSb     ]SSS     #SSSM|    SSSQ    ]SSSQ
 SSS[    @SSS     SSS[  SSSQ     @SSS   #QSSS^      SSSQ    ]SSSQ
 SSS[    @SSS     SSS[  QSSSQQQQQSSSF  SSSSSSSSSSS  %SSSSSSSSSSQ
 PPP"    "PPP    PPP+   ^FBWWWWBEP`   "PPPPPPPPPPT    "+PPPPPPPP"
 .-------------------------------------------------------------------.
 |     Follow the prompts to scaffold a Mozu Application that     |
 |   contains Actions. You'll get a directory structure, action   |
 |            file skeletons, and a test framework!               |
 '-------------------------------------------------------------------'


? Application package name (letters, numbers, dashes): YOUR_DEVCENTER_ACCOUNT_KEY.Y
OUR_DEVCENTER_APP_NAME
? Short description:
? Initial version: 1.0.0
? Developer Center Application Key for this Application: YOUR_DEVCENTER_ACCOUNT_KEY.YO
UR_DEVCENTER_APP_NAME.1.0.0.Release
? Enter your Developer Account login email: YOUR_DEVCENTER_ACCOUNT_EMAIL_ADDRESS
? Developer Account password: [hidden]
>> Looking up developer accounts...


? Select a developer account for YOUR_DEVCENTER_ACCOUNT_EMAIL_ADDRESS: Your Develo
per Account (9999)
? Choose a test framework: None/Manual
? Enable actions on install? (This will add a custom function to the embedded.platform.applica
tions.install action.) Yes


Unit tests are strongly recommended. If you prefer a framework this generator does not supp
ort, or framework-free tests, you can still use the mozu-action-simulator module to simulate a
 server-side environment for your action implementations.


...
? Choose domains: platform*applications
? Actions for domain platform*applications
...
```

Step 10: Modify Pom for Synchronization

Modify the project's pom.xml to add support for synchronizing KJAR with the application:

```
 <profiles>
  <profile>
   <id>devcenter</id>
   <activation>
    <property>
     <name>devcenter</name>
    </property>
   </activation>
   <properties>
    <devcenterAssetsDirectory>${project.basedir}/devcenter-app/assets</devcenterAssetsDire
ctory>
   </properties>
   <build>
    <plugins>
     <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-dependency-plugin</artifactId>
      <version>3.1.2</version>
      <executions>
       <execution>
        <id>copy-assets</id>
        <phase>install</phase>
        <goals>
         <goal>copy</goal>
        </goals>
        <configuration>
         <artifactItems>
          <artifactItem>
           <groupId>${project.groupId}</groupId>
           <artifactId>${project.artifactId}</artifactId>
           <version>${project.version}</version>
           <type>${project.packaging}</type>
           <overWrite>true</overWrite>
          </artifactItem>
         </artifactItems>
         <outputDirectory>${devcenterAssetsDirectory}</outputDirectory>
        </configuration>
       </execution>
      </executions>
     </plugin>
    </plugins>
   </build>
  </profile>
 </profiles>
```

Step 11: Build BPM Project with Assets

Built the BPM project and ensure that the assets are present:

1. Type `mvn clean install -P devcenter`.
2. Verify existence of KJAR file in `devcenter-app/assets` directory.

Step 12: Install Custom Workflows

Upload and install your custom workflows through the Dev Center:

1. Change your current working directory to the `devcenter-app` directory.
2. Validate the content of the file `mozu.config.json`
3. Type `grunt -f` to upload your application assets to Kibo DevCenter.
4. Log in to the Dev Center and open your application.
5. Verify the existence of your uploaded application assets in the **Packages > Assets** view.
6. Click the **Install** button.
7. Select the appropriate Sandbox and click **OK**.

Step 13: Enable Workflows for Location Groups

Update your location group configuration settings to use customized processes, referencing the new containerId(s) and processId(s) by shipmentType. Some example cURL requests are listed below.

Get all location groups for a tenant and site:

```
curl --request GET 'http://t123.mozu.com/api/commerce/admin/locationGroups' \
--header 'x-vol-tenant: 123' \
--header 'x-vol-site: 456' \
--header 'Authorization: Bearer *******'
```

Get configuration for a specific location group:

```
curl --request GET 'http://t123.mozu.com/api/commerce/admin/locationGroupConfiguration/2' \
--header 'x-vol-tenant: 123' \
--header 'x-vol-site: 456' \
--header 'Authorization: Bearer *******'
```

Set a custom BPM configuration for a location group:

```
curl --request PUT 'http://t123.mozu.com/api/commerce/admin/locationGroupConfiguration/2' \
--header 'x-vol-tenant: 123' \
--header 'x-vol-site: 456' \
--header 'Authorization: Bearer *******' \
--header 'Content-Type: application/json' \
--data-raw '{
   "tenantId": 123,
   "siteId": 456,
   "locationGroupId": 2,
   "locationGroupCode": "2",
   ...
   "bpmConfigurations": [
      {
         "shipmentType": "BOPIS",
         "workflowContainerId": "YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME"
,
         "workflowProcessId": "fulfillment.FulfillmentProcess-BOPIS"
      },
      {
         "shipmentType": "STH",
         "workflowContainerId": "YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME"
,
         "workflowProcessId": "fulfillment.FulfillmentProcess-STH"
      },
      {
         "shipmentType": "Transfer",
         "workflowContainerId": "YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME"
,
         "workflowProcessId": "fulfillment.FulfillmentProcess-Transfer"
      }
   ],
   ...
}'
```

Step 14: Execute Custom Workflows in Fulfiller UI

To execute custom workflows as a fulfiller user, such as for testing the new workflow:

1. Log in to the Admin UI and select the appropriate tenant.
2. Create a new order and shipment type matching the custom workflow configuration.
3. Go to **Main** > **Fulfiller** and locate the corresponding shipment.
4. Proceed through the workflow tasks for the shipment and confirm functionality.

Step 15: Sync Custom Fork with Upstream Repository

To sync your custom forked repository with the upstream repository:

1. Open Terminal.
2. Change the current working directory to your local project.
3. Fetch the branches and their respective commits from the upstream repository. Commits to develop will be stored in a local branch, `upstream/develop`.

```
$ git fetch upstream
> remote: Counting objects: 8, done.
> remote: Compressing objects: 100% (8/8), done.
> remote: Total 8 (delta 3), reused 0 (delta 0), pack-reused 0
> Unpacking objects: 100% (8/8), done.
> From https://github.com/KiboSoftware/kibo-fulfillment-workflows
>  * [new branch]      develop                    -> upstream/develop
```

4. Check out your fork's local `develop` branch.

```
$ git checkout develop
> Switched to branch 'develop'
```

5. Merge the changes from `upstream/develop` into your local `develop` branch. This brings your fork's `develop` branch into sync with the upstream repository, without losing your local changes.

```
$ git merge upstream/develop
> Merge made by the 'recursive' strategy.
```

6. If your local branch didn't have any unique commits, Git will instead perform a "fast-forward":

```
$ git merge upstream/develop
> Updating 34e91da..16c56ad
> Fast-forward
>  README.md              |    5 +++--
>  1 file changed, 3 insertions(+), 2 deletions(-)
```

Syncing your fork updates only your local copy of the repository. To update your fork on GitHub, you must push your changes. For more information about syncing a fork, see the GitHub documentation.

Step 16: Sync jBPM Repository with Forked Repository

Follow these command-line instructions to synchronize the local jBPM Business Central repository with the custom forked repository:

```
$ git checkout master
> Switched to the 'master' branch.
$ git reset --hard develop
> HEAD is now at a5fff3d Merge remote-tracking branch 'upstream/develop' into develop.
$ git push -f jbpm master
> Counting objects: 60, done.
> Delta compression using up to 8 threads.
> Compressing objects: 100% (52/52), done.
> Writing objects: 100% (60/60), 155.44 KiB | 8.18 MiB/s, done.
> Total 60 (delta 26), reused 0 (delta 0)
> remote: Resolving deltas: 100% (26/26)
> remote: Updating references: 100% (1/1)
> To http://localhost:8080/business-central/git/MySpace/YOUR_DEVCENTER_ACCOUNT_KEY.YOUR_DEVCENTER_APP_NAME
>  + ace62bb...a5fff3d master -> master (forced update)
```