An IBM Document from

IBM Z DevOps Acceleration Program

March, 2022 - Release 1.1

zAppBuild Introduction and Custom Version Maintenance Strategy

Nelson Lopez
nelson.lopez1@ibm.com

Timothy Donnelly
donnellt@us.ibm.com

Dennis Behm
dennis.behm@de.ibm.com

Lauren Li
lauren.k.li@ibm.com

Abstract

An introduction to the implementation of the sample build framework zAppBuild leveraging IBM DBB APIs and a guide on how to manage and refresh your own fork of it

Table of content

Table of Contents

1 Part 1 - Introduction to zAppBuild 3

2 Part 2 – Installation and Maintenance 5

2.1 Make the zAppBuild repository available in your preferred Git provider 5

2.2 Update your customized version of zAppBuild with latest official zAppBuild enhancements 8

3 Additional References 10

1 Part 1 - Introduction to zAppBuild

This is a brief introduction to the public DBB Groovy framework called zAppBuild available at the public GitHub repository https://github.com/IBM/dbb-zappbuild

DBB-zAppBuild is a free, generic mainframe application build framework that customers can extend to meet their DevOps needs. It is available under the Apache 2.0 license.

It is a sample to get you started building source on a zOS Unix System which got added via Git. It is made up of Groovy scripts which leverage the IBM Dependency Based Build toolkit APIs and property files to configure the build behavior.

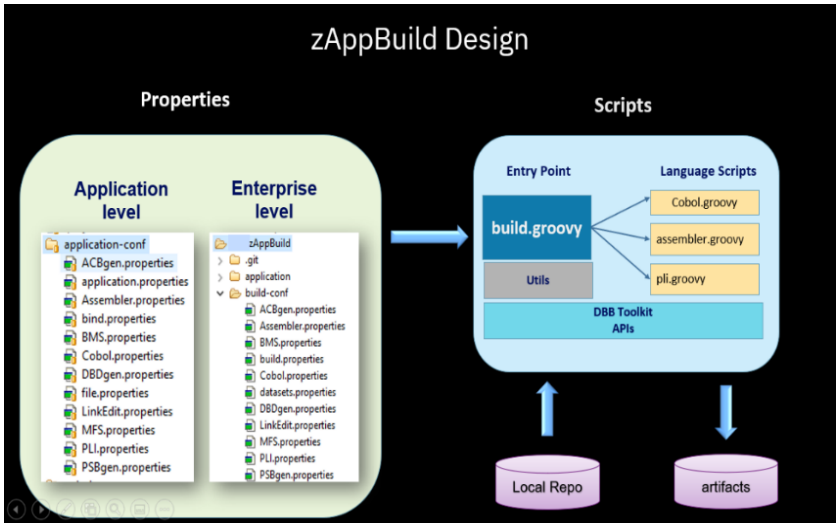
Build properties can span across all applications, one application, or individual programs.

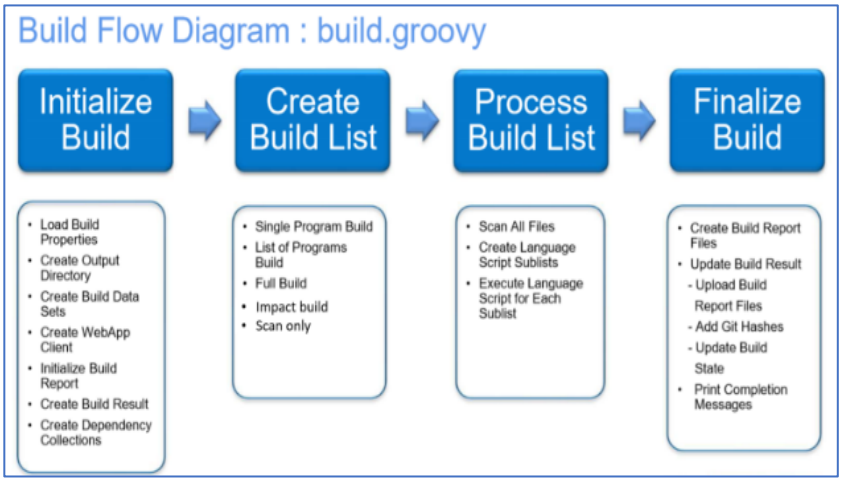
Properties that cross all applications are managed by administrators and define enterprise-wide settings like the PDS name of the compiler, data set allocation attributes and more. Application and program level properties are typically managed within the application repository itself.

The build framework is invoked either by a developer using IDz's 'User Build' capability or by an automated CI/CD pipeline. It supports different build types.[1]

The main script 'build.groovy' of zAppBuild, initializes the build environment, identifies what to build and invokes language scripts. Utilities and DBB APIs[2] are used to produce runtime artifacts.

The build process also creates logs and an artifact manifest (BuildReport.json) for deployment processes managed in IBM UrbanCode Deploy.





[1] https://github.com/IBM/dbb-zappbuild#build-scope

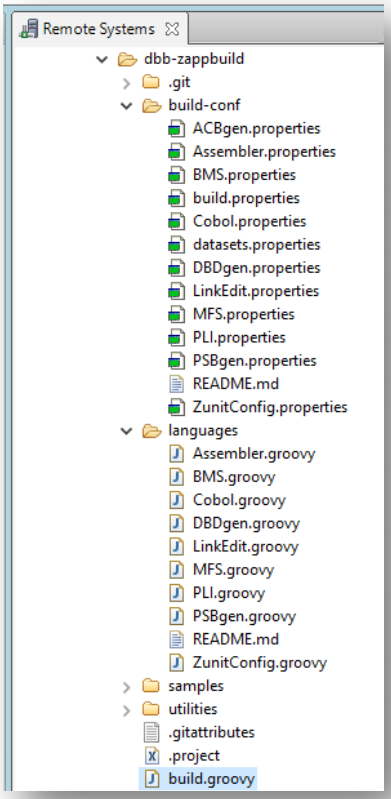
[2] DBB Toolkit Javadoc https://www.ibm.com/support/knowledgecenter/SS6T76_1.1.0/javadoc/index.html

zAppBuild Architecture

The build framework is split into two parts. The core build framework called "**dbb-zappbuild**" which contains the build scripts and stores enterprise-level settings and resides in a permanent location on the USS file system. These are typically owned and controlled by the central build team.

The other is "**application-conf**" which resides within each application repository, to provide application-level settings to the central build framework. These settings are owned by the application team and, as such, are maintained and updated by them.

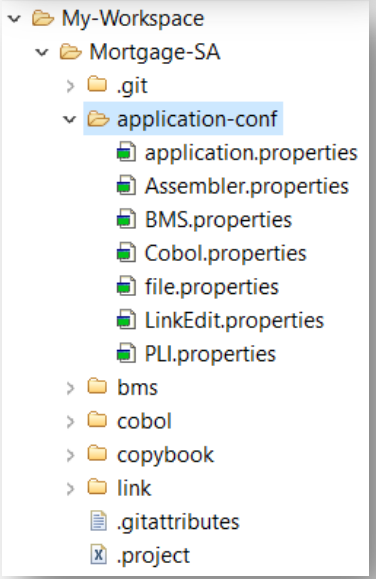
dbb-zappbuild folder structure overview



- **build-conf:**
  - "**build-properties**" defines the DBB Web server's URL, password and other initialization properties. Make sure to update the DBB URL and password information. When using a password file connect to the DBB WebApp, ensure it is tagged as ISO88591-1 (ascii).
  - **"dataset.properties"** describes system datasets like the PDS name of the COBOL compiler or libraries used for the subsystem. You must update this with your site's dataset names.
  - Several language specific property files that define the compiler or link-editor/binder program names, system libraries and general system level properties for Cobol, Assembler …
- **languages -** Groovy scripts used to build programs. For example, Cobol.groovy is called by build.groovy to compile the Cobol source codes. The application source code is mapped by its file extension to the language script in the "**application-conf /file.properties"**.
- **samples -** contains an "application-conf" template, .gitattributes and a reference sample application MortgageApplication.
- **utilities -** contains helper scripts used by build.groovy and other scripts to calculate the build list.
- **build.groovy** - the main build script of zAppBuild. It takes several required command line parameters to customize the build process.

application-conf overview

This folder within the application repository defines application-level properties like:



- **application.properties -** various directory rules, default git branch, impact resolution rules like the copybook lookup rules and more
- **file.properties** - maps files to language scripts in "dbb-zappbuild/languages" and provides file level property overrides.
- **Property files** for further customization of the language script processing, e.q. Cobol.properties is one of the language properties to define compiler and link-edit options among other things

2 Part 2 – Installation and Maintenance

Many clients started to use zAppBuild and have enhanced it to their needs by adding new language scripts, or by modifying the existing build processing logic.

This section provides a set of instructions of how you can make zAppBuild available in your Git provider and how to synchronize new features of zAppBuild into your customized fork.

Be aware, that zAppBuild releases new versions through the main branch. New contributions are added first to the develop branch, which then will be merged to the main branch.

2.1 Make the zAppBuild repository available in your preferred Git provider

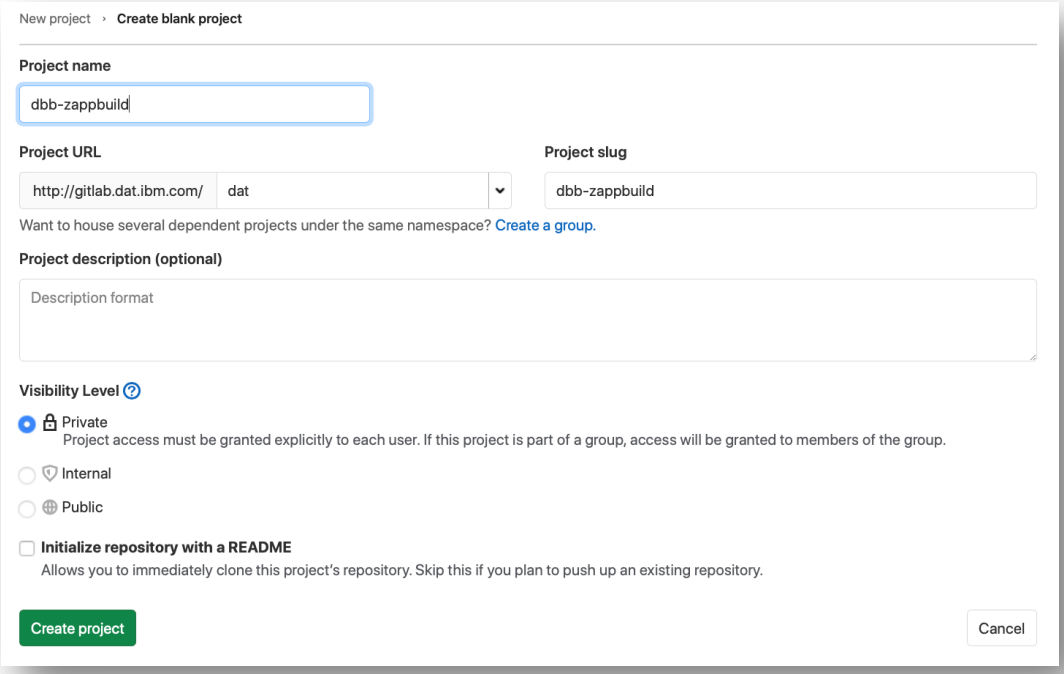
Before you start your customization of zAppBuild, it requires to clone your repository and store it in your Git provider of choice. This could be any Git provider, like GitHub, GitLab, Bitbucket or AzureDevOps etc. If you have done this already, feel free to move to the next section.

Here are the sample steps to make the zAppBuild repository available in an on-premise GitLab environment.

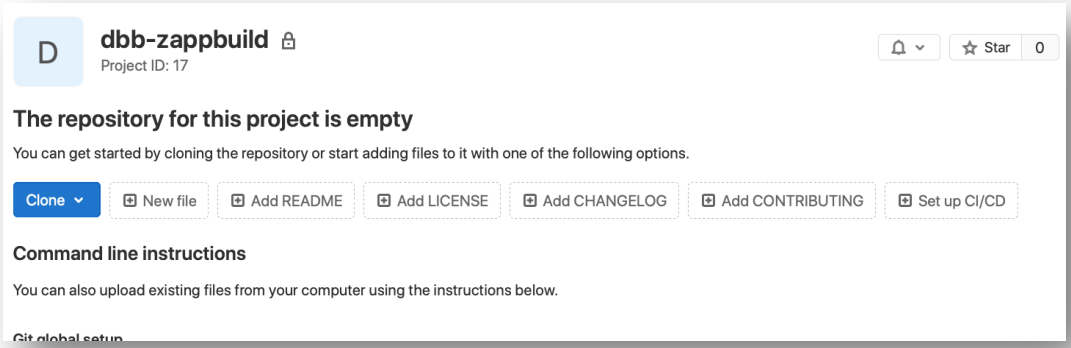
1. Use your Browser and log on to your Git provider. Create a new repository in GitLab, which will be the new "home" of your customized version of zAppBuild.

    a. In GitLab, navigate to the *Group* you would like to use.

    b. Create a *New Project* with your preferred name. We use "dbb-zappbuild". Don't initialize the repository. Set the visibility according to your needs.



c. Your Git provider will create the repository, but it is not yet initialized.

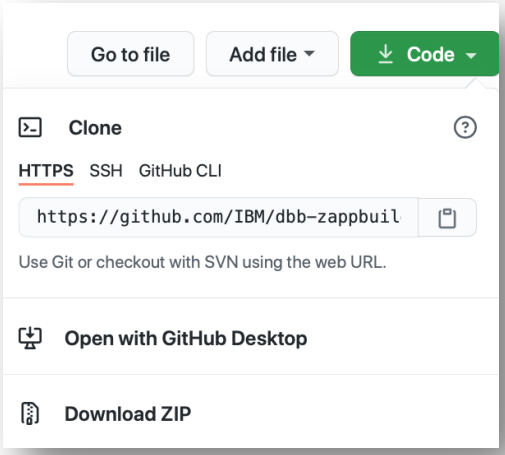


This page contains information on how to share an existing Git repository. Leave the browser open.

2. Clone the public dbb-zappbuild repository. You can use IDZzor also a terminal to complete the step. We document the steps using a terminal (Gitbash or appropriate.)

   a. In the terminal navigate to the folder where you would like to clone the repository.

   b. Retrieve the Git repository URL or SSH path from the public zAppBuild repository from https://github.com/IBM/dbb-zappbuild



   c. In your terminal enter the command for cloning the repository.

```
git clone https://github.com/IBM/dbb-zappbuild.git
```

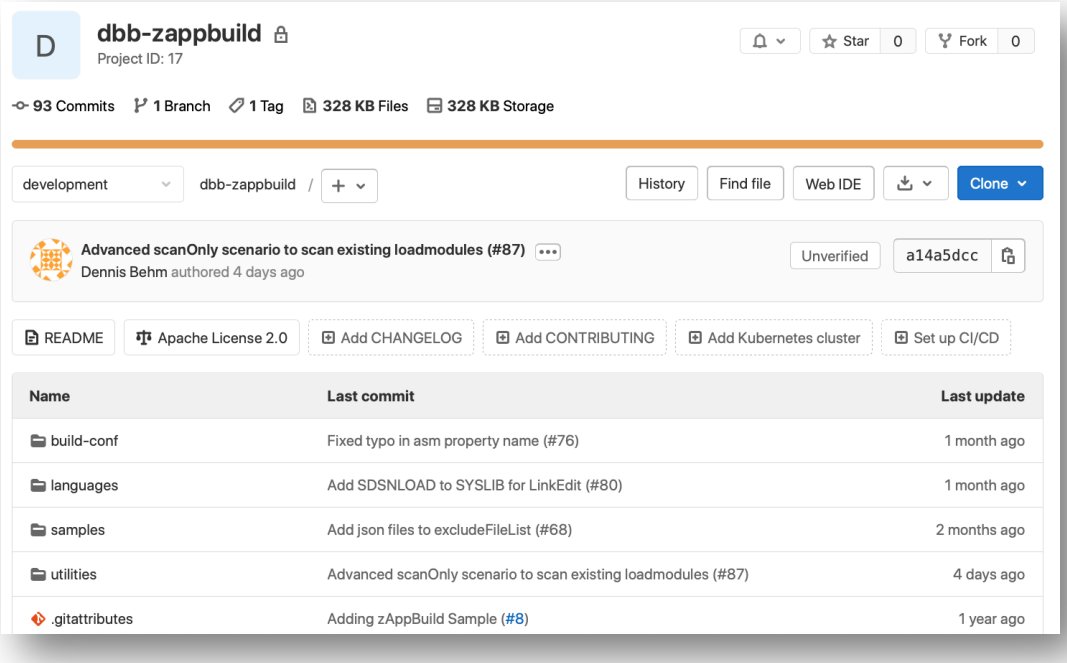
```
dennisbehm@Denniss-MacBook-Pro ▶ ~/git/buildframework ▶ git clone https://github.com/IBM/dbb-zappbuild.git
Cloning into 'dbb-zappbuild'...
remote: Enumerating objects: 604, done.
remote: Total 604 (delta 0), reused 0 (delta 0), pack-reused 604
Receiving objects: 100% (604/604), 217.50 KiB | 898.00 KiB/s, done.
Resolving deltas: 100% (395/395), done.
```

3. Follow the instructions of your Git provider to push the existing repository. (* instructions may vary from Git provider to Git provider)

   a. Within the terminal session execute the provided commands to Push an existing Git repository

```
cd <existing_repo>
git remote rename origin old-origin
git remote add origin <Your on-premise git repository> (like git@gitlab.dat.ibm.com:DAT/dbb-zappbuild.git )
git push -u origin -all
git push -u origin --tags
```

   b. You will find dbb-zappbuild with the entire history like on the public git repository.



2.2 Update your customized version of zAppBuild with latest official zAppBuild enhancements

It is assumed at this point, that you have enhanced your fork and now would like to integrate the latest features into your version of zAppBuild. So, let's get started.

1. Locate the internal git repository and create a new Git branch. This is a good practice to validate the changes first. In our sample it is called *update-zappbuild.*

2. As the next step, we will add a new Git remote definition to connect to the official public GitHub repository. (* Requires internet connectivity)

   a. First, let's list the remotes by issuing `git remote -v`

```
dennisbehm@Denniss-MacBook-Pro  ~/git/gitlab/dbb-zappbuild  35-update-zappbuild  git remote -v
origin  http://gitlab.dat.ibm.com/dat/dbb-zappbuild.git (fetch)
origin  http://gitlab.dat.ibm.com/dat/dbb-zappbuild.git (push)
```

   For more documentation see https://git-scm.com/docs/git-remote

   b. We are now adding a new remote to connect to GitHub by issuing

```
git remote add zappbuild-official https://github.com/IBM/dbb-zappbuild.git
```

   c. Let's verify that it is available through issuing the previous command again. `git remote -v`

```
dennisbehm@Denniss-MacBook-Pro  ~/git/gitlab/dbb-zappbuild  35-update-zappbuild  git remote -v
origin  http://gitlab.dat.ibm.com/dat/dbb-zappbuild.git (fetch)
origin  http://gitlab.dat.ibm.com/dat/dbb-zappbuild.git (push)
zappbuild-official      https://github.com/IBM/dbb-zappbuild.git (fetch)
zappbuild-official      https://github.com/IBM/dbb-zappbuild.git (push)
```

   d. Let's fetch the latest information from the official repository, by executing git fetch for the official dbb-zappubuild repository:

```
git fetch zappbuild-official
```

```
dennisbehm@Denniss-MacBook-Pro  ~/git/gitlab/dbb-zappbuild  35-update-zappbuild  git fetch zappbuild-official
remote: Enumerating objects: 33, done.
remote: Counting objects: 100% (33/33), done.
remote: Total 51 (delta 33), reused 33 (delta 33), pack-reused 18
Unpacking objects: 100% (51/51), 21.58 KiB | 223.00 KiB/s, done.
From https://github.com/IBM/dbb-zappbuild
 * [new branch]      development -> zappbuild-official/development
 * [new branch]      master      -> zappbuild-official/master
 * [new tag]         2.0.0       -> 2.0.0
```

e. Make sure, that your feature branch is checked out, before you merge the changes from zappbuild-official. To merge the changes run into your branch *update-zappbuild,* issue

```
git merge zappbuild-official/main
```

```
dennisbehm@Denniss-MacBook-Pro  ~/git/gitlab/dbb-zappbuild  35-update-zappbuild  git merge zappbuild-official/development
Auto-merging utilities/ImpactUtilities.groovy
CONFLICT (content): Merge conflict in utilities/ImpactUtilities.groovy
Auto-merging utilities/GitUtilities.groovy
Auto-merging utilities/BuildUtilities.groovy
Auto-merging samples/MortgageApplication/application-conf/application.properties
Automatic merge failed; fix conflicts and then commit the result.
```

Potentially, you face merge conflicts. In the above case, the merge processor could not automatically resolve the utilities/ImpactUtilities.groovy

Run a `git status` to see which files changed

```
dennisbehm@Denniss-MacBook-Pro  ~/git/gitlab/dbb-zappbuild  35-update-zappbuild | merge  git status
On branch 35-update-zappbuild
Your branch is up to date with 'origin/35-update-zappbuild'.

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:
        modified:   BUILD.md
        modified:   README.md
        modified:   build-conf/Assembler.properties
        modified:   build.groovy
        modified:   languages/Assembler.groovy
        modified:   languages/BMS.groovy
        modified:   languages/Cobol.groovy
        modified:   languages/PLI.groovy
        modified:   samples/MortgageApplication/application-conf/application.properties
        modified:   samples/application-conf/application.properties
        modified:   utilities/BuildUtilities.groovy
        modified:   utilities/GitUtilities.groovy

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   utilities/ImpactUtilities.groovy
```

f. Open the unmerged files and resolve them manually. Either use the terminal, or an IDE for this task.

g. Commit the changes and verify them with a sample application before you are committing it to your main branch, which is used for all your production DBB builds.

3 Additional References

Always select the latest version of the material referenced below.

- Publications: List of resources on building a open an modern CICD pipeline for the mainframe
- Training: IBM Dependency Based Build Fundamentals - Mainframe DEV
- Documentation: Knowledge Center Documentation including the JavaDoc for the IBM Dependency Based Build APIs
- Tutorial: Basics of writing your first build script