repoImport

Command Reference

Table of contents

1. Description

Import packages into the repository.

The repoImport command recursively scans directories under `-targetdir` looking for files that match the regexes: (`filebase`)(?:`buildstamp`)?.(`extension`) loading them into the repository if the necessary metadata is provided.

There are two setup modes for repoImport:

1. Single package import: If only one build artifact is generated by the Builder, then ensure the follwoing: `-min` and `-max` both use a value of "1", `-type` refers to an existing Package type, `-installroot` refers to a valid file path.
2. Multiple package import: If multiple build artifacts are generated by the Builder, then it is possible to use a `-propfile` as a metadata template for each kind of package to import.

> **Note:**
> The repoImport command assumes that the Package subtype already exists and will fail if the type is not defined in the project model.

The command will fail if the `-min` and `-max` criteria are not met.

static: This command can be run outside of an object context.

2. Usage

```
ctl -m Builder -c repoImport [-buildstamp <>] [-extension <>]
[-filebase <>] [-installrank <>] [-installroot <>] [-max <>]
[-min <>] [-packageBuildtimePattern <>] [-propfile <>]
[-requirebuildstamp <>] [-separator <->] [-targetdir <>]
[-type <>] [-vendor <>] [-version <>]
```

2.1. Options

Option	Description	Type	Default
`buildstamp`	*build identifier*	string	`${entity.attribute.buildstamp}`
`extension`	*package file extension*	string	`${entity.attribute.packageExte`
`filebase`	*package file base*	string	`${entity.attribute.packageFile`
`installrank`	*package install-rank*	string	`${entity.attribute.packageInst`
`installroot`	*package install-root*	string	`${entity.attribute.packageInst`

`max`	*maximum number of packages to import after build*	string	`${entity.attribute.importMax}`
`min`	*minimum number of packages to import after build*	string	`${entity.attribute.importMin}`
`packageBuildtimePa`	*timestamp pattern*	string	`${entity.attribute.packageBuil`
`propfile`	*metadata template file used during package registration*	string	
`requirebuildstamp`	*whether or not to require that files be matched by buildstamp*	string	
`separator`	*package name separator character*	string	`-` / `${entity.attribute.packageSepa`
`targetdir`	*top level directory containing build artifacts*	string	`${entity.attribute.targetdir}`
`type`	*package type* This must be a pre-existing type in the project model.	string	`${entity.attribute.packageType`
`vendor`	*package vendor*	string	/ `${entity.attribute.packageVend`
`version`	*package version*	string	