BuildPatchTool (BPT) Instructions [1.5]

Game Setup

Before You Start: A Note About Paths on the Command Line

Due to the way the command prompt interprets \" (a backslash directly followed by a double quote) on the command line, you need to follow at least one of the following pieces of guidance when specifying paths in BuildPatchTool arguments:

- Do not append a trailing slash to your directory names.
- Use forward slashes for directory separators.
- Do not enclose paths in quotes (this is not possible if your path contains one or more spaces).
- Escape only the trailing backslash with another backslash if it is double-quote enclosed (not recommended, as is error prone)

Examples:

```
    -BuildRoot="D:\MyFolder\"   WILL NOT WORK
    -BuildRoot="D:/MyFolder/"   WORKS
    -BuildRoot=D:\MyFolder\     WORKS
    -BuildRoot=D:\MyFolder      WORKS
    -BuildRoot=D:/MyFolder/     WORKS
    -BuildRoot=D:/MyFolder      WORKS
    -BuildRoot="D:\MyFolder\\"  WORKS
```

Initial Setup

Extract the contents of the *BuildPatchTool ZIP file* to a machine that has access to the binary you want to upload. The machine will also need Internet access to communicate with the Epic backend.

Note: The ZIP file link is always kept up to date with the newest version of the BPT. If you cannot access the file, please create a case.

Authentication

Authentication is carried out by supplying a Client ID and Client Secret to BuildPatchTool. These must be provided for every operation using command line arguments. Epic will issue you this ID and secret to use as your credentials.

Note: BuildPatchTool uses a unique Client ID and Client Secret, separate from any EOS Client IDs your game may use. Refer to the Build Patch Tool Credentials section of your Product Settings in Dev Portal to obtain the correct Client ID and Secret. Clients listed under SDK Credentials will not function with the BuildPatchTool.

The `ClientId` parameter *must* be provided with every operation that interacts with Epic's backend services. The client secret must be provided by using either of the `ClientSecret` or `ClientSecretEnvVar` parameters.

If using the `ClientSecret` parameter, then your client secret should be passed verbatim as the value of the parameter:

```
Engine\Binaries\Win64>BuildPatchTool.exe
                            -ClientId="<YourClientId>"
                            -ClientSecret="<YourClientSecret>"
```

If using the `ClientSecretEnvVar` parameter, then you pass the name of an OS environment variable which contains the client secret:

```
Engine\Binaries\Win64>set MyEpicSecret=<YourClientSecret>
Engine\Binaries\Win64>BuildPatchTool.exe
                            -ClientId="<YourClientId>"
                            -ClientSecretEnvVar="MyEpicSecret"
```

We recommend using the `ClientSecretEnvVar` parameter whenever possible, as this prevents the value of your secret appearing in system logs and command line history. It is a best practice to store your client secret in a secrets management system (such as Hashicorp Vault), and securely inject into your build machine's environment during each run.

How to Upload a Binary

Navigate to the `Engine/Binaries/Win64/` directory. From a command line, run BuildPatchTool.exe in *UploadBinary* mode, and point it at your binary (see CLI argument details below).
This will process your binary , upload it to our cloud storage, and register the binary with our backend:

```
    Engine\Binaries\Win64>BuildPatchTool.exe
                            -OrganizationId="<YourOrg>"
                            -ProductId="<YourProduct>"
                            -ArtifactId="<YourArtifact>"
                            -ClientId="<YourClientId>"
                            -ClientSecretEnvVar="<YourSecretEnvVar>"
                            -mode=UploadBinary
                            -BuildRoot="<LocationOfLocalBuild>"
                            -CloudDir="<YourCloudDir>"
                            -BuildVersion="<YourBuildVersion>"
                            -AppLaunch="<AppToRun>"
                            -AppArgs="<LaunchArguments>"

                            Optional:
                            -FileList="<LocationOfFileList>"
                            -FileAttributeList="<LocationOfAttributesFile>"
                            -FileIgnoreList="<LocationOfIgnoreFile>"
```

Descriptions for each of the parameters that you have to specify are below. You can also add the `"-help"` option after any mode (e.g. `"BuildPatchTool.exe -mode=PatchGeneration -help"`) to get information about the required arguments.

- `OrganizationID`
  - Use the Organization ID string that was provided along with your credentials.
- `ProductID`
  - Use the Product ID string that was provided along with your credentials.
- `ArtifactID`
  - Specify the Artifact ID string that was provided along with your credentials.
- `ClientId`
  - See Authentication section.
- `ClientSecretEnvVar`
  - See Authentication section.
- `BuildRoot`
  - The path to the directory containing the binary to be read and processed. It can be an absolute path from the drive root or a relative path from the current working directory. Additionally, it is recommended that this path be located near the drive root, to avoid any files exceeding the OS MAX_PATH limit (typically 260 characters).

- `CloudDir`
  - A directory where BuildPatchTool can save files to be uploaded, this can be empty each run. As with the BuildRoot, this can be an absolute or a relative path. (This location is used to cache information about existing binaries, and should be a different directory from the *BuildRoot* parameter. It is OK if this directory is initially empty; BuildPatchTool will download information as needed from the Epic backend and store it in the CloudDir.)
- `BuildVersion`
  - The version string for the build. This needs to be unique for each build of a specific artifact. The build version string has the following restrictions:
    - Must be between 1 and 100 characters in length
    - Should only contain characters from the following sets `a-z`, `A-Z`, `0-9`, or `.+-_`
    - Whitespace is not allowed
- `AppLaunch`
  - The path to the app executable that should be launched when running your game, relative to (and inside of) the *BuildRoot*. For Mac binaries, this should be the executable file contained within the .app folder, usually in the location Game.app/Contents/MacOS/Game.
- `AppArgs`
  - The commandline to send to the app on launch. This can be set to “” when no additional arguments are needed.
- `FileList` (*Optional*)
  - A path to a text file containing files to be included in the binary. The files must be *BuildRoot* relative. This is an alternative to using *FileIgnoreList*.

- `FileAttributeList` (*Optional*)
  - A path to a text file containing a list of files and corresponding special attributes (e.g. executable bit) that should be set. See Setting Special File Attributes section for a description of the file contents.
  - Note that the attributes file should not be inside your *BuildRoot* to ensure that it does not get erroneously included in your binary upload.
- `FileIgnoreList` (*Optional*)
  - A path to a text file containing a list of files that should be excluded from the generated patch data. Each entry should be on a new line, and be a relative path from *BuildRoot*. A forward slash ("/") separator should be used.
  - Note that if the ignore file is located inside *BuildRoot* then its own relative file path needs to be included in the ignore list to avoid it being included in the patch data.

Setting Special File Attributes

You can use the optional -FileAttributeList parameter with the PatchGeneration mode to apply special attributes to files within your binary, such as executable, or tagging groups of files. Note that the launch app (as specified by the -AppLaunch argument) is automatically marked as executable and does not need to be explicitly included in this configuration file. If you do not need to set any special attributes, the -FileAttributeList argument can be omitted.

Your custom FileAttributeList file must be a text file with one or more newline-separated entries. Each line must contain a quoted (BuildRoot-relative) filepath, followed by one or more space-delimited attributes to be applied to the referenced file:

```
"<QuotedRelativePathToFile>" <list of attributes>
```

Example entries:

```
"Relative/Path/To/My/file.bin" executable tag:my_tag
"Second/Path/To/A/file.txt" tag:txt_files
```

The following file attribute flags are supported:

- `executable`
- `tag:my_tag`

File tagging is used for providing input to Epic Games Launcher's selective download feature, exposed in installation options screens. If you feel you need to use this feature, please contact an Epic Games representative, as the feature is still in beta.

Setting the Desktop Shortcut Icon

By default, the desktop shortcut icon will be set to the icon of the app executable indicated by the AppLaunch argument. There are two ways of updating this icon:

- Updating the icon embedded in the app executable
- Adding a separate .ico file with same file name as the app executable (for example: launch.ico should be used if the executable is launch.exe)

How to Label a Binary

After the chunking and upload process has completed, you must apply a label to your binary to make it accessible from the Epic Games Launcher. Applying a label to a binary associates a label *name* (e.g. "Live") and a *platform* (e.g. "Win32") with a single binary version. Multiple labels can be assigned to the same binary.

The EpicGamesLauncher client will only ever query for binaries associated with the platform on which it is currently running. For example, the Win32 Launcher client will only ever query with a platform of *Win32*, so it will never receive binaries labeled with *Windows* or *Mac*. A Windows x64 client will only query for binaries using the *Windows* platform, and so on. If you do actually have a single binary that is compatible with multiple platforms (for example, one compatible with both *Windows* and *Win32*), then you can apply two labels (one for each platform) to the same binary to make it available for both platforms without having to re-upload it.

LabelBinary mode:

```
Engine\Binaries\Win64>BuildPatchTool.exe
                        -OrganizationId="<YourOrg>"
                        -ProductId="<YourProduct>"
                        -ArtifactId="<YourArtifact>"
                        -ClientId="<YourClientId>"
                        -ClientSecretEnvVar="<YourSecretEnvVar>"
                        -mode=LabelBinary
                        -BuildVersion="<YourBuildVersion>"
                        -Label="<LabelName>"
                        -Platform="<Platform>"

                        Optional:
                        -SandboxId="<YourSandbox>"
```

Description of parameters:

- `OrganizationID`
  - Use the Organization ID string that was provided along with your credentials.
- `ProductID`
  - Use the Product ID string that was provided along with your credentials.
- `ArtifactID`
  - Specify the Artifact ID associated with the binary to be labeled.
- `ClientId`
  - See Authentication section.
- `ClientSecretEnvVar`
  - See Authentication section.
- `BuildVersion`
  - The version string of the binary that should be labeled. A binary with this version must already have been registered, or the labeling operation will be rejected.
- `Label`
  - The label name to apply to your binary. This must be the string "Live" when setting a binary as live to the public. Additionally, the string “Archive” can be used to label a single binary per platform that should be retained even after it is no longer live. Unlabeled binaries will be deleted after roughly a week by automatic cleanup jobs.

- `Platform`
  - The platform associated with the binary to be labeled. Currently must be one of: [Windows, Win32, Mac]
- `SandboxId`
  - Specifies the id of the sandbox to label this binary under. If not provided, the binary will be labeled for the public.

How to Unlabel a Binary

To remove previously set label you can use *UnlabelBinary* Mode with the same parameters that were used for *LabelBinary* Mode:

```
Engine\Binaries\Win64>BuildPatchTool.exe
                        -OrganizationId="<YourOrg>"
                        -ProductId="<YourProduct>"
                        -ArtifactId="<YourArtifact>"
                        -ClientId="<YourClientId>"
                        -ClientSecretEnvVar="<YourSecretEnvVar>"
                        -mode=UnlabelBinary
                        -BuildVersion="<YourBuildVersion>"
                        -Label="<LabelName>"
                        -Platform="<Platform>"

                        Optional:
                        -SandboxId="<YourSandbox>"
```

Description of parameters:

- `OrganizationID`
  - Use the Organization ID string that was provided along with your credentials.
- `ProductID`
  - Use the Product ID string that was provided along with your credentials.
- `ArtifactID`
  - Specify the Artifact ID associated with the binary to be labeled.
- `ClientId`
  - See Authentication section.
- `ClientSecretEnvVar`
  - See Authentication section.
- `BuildVersion`
  - The version string of the binary that should be labeled. A binary with this version must already have been registered, or the labeling operation will be rejected.
- `Label`
  - The label name to apply to your binary. This must be the string "Live" when setting a binary as live to the public. Additionally, the string "Archive" can be used to label a single binary per platform that

should be retained even after it is no longer live. Unlabeled binaries will be deleted after roughly a week by automatic cleanup jobs.

- `Platform`
  - The platform associated with the binary to be labeled. Currently must be one of: [Windows, Win32, Mac]
- `SandboxId`
  - Specifies the id of the sandbox to unlabel this binary from. If not provided, the binary will be unlabeled from the public.

How to List Existing Binaries

You can also query our backend to retrieve the list of binaries that you have already registered using the *ListBinaries* Mode:

```
Engine\Binaries\Win64>BuildPatchTool.exe
                        -OrganizationId="<YourOrg>"
                        -ProductId="<YourProduct>"
                        -ArtifactId="<YourArtifact>"
                        -ClientId="<YourClientId>"
                        -ClientSecretEnvVar="<YourSecretEnvVar>"
                        -mode=ListBinaries

                        Optional:
                        -OnlyLabeled
                        -Num="<NumberOfBinariesReturned>"
                        -OutputFile="<OutputFileName>"
```

Description of parameters:

- `OrganizationID`
  - Use the Organization ID string that was provided along with your credentials.
- `ProductID`
  - Use the Product ID string that was provided along with your credentials.
- `ArtifactID`
  - Specify the Artifact ID associated with the binary to be labeled.
- `ClientId`
  - See Authentication section.
- `ClientSecretEnvVar`
  - See Authentication section.
- `OnlyLabeled` *(Optional)*
  - Only list binaries that are associated with a label.
- `Num` *(Optional)*
  - Restrict the number of binaries returned. Binaries are returned in date order with the most recent first.
- `OutputFile` *(Optional)*
  - The list of binaries will be output to the specified file as a JSON array.

How to Delete a Binary

You can delete a previously uploaded binary by using the *DeleteBinary* Mode:

```
Engine\Binaries\Win64>BuildPatchTool.exe
                        -OrganizationId="<YourOrg>"
                        -ProductId="<YourProduct>"
                        -ArtifactId="<YourArtifact>"
                        -ClientId="<YourClientId>"
                        -ClientSecretEnvVar="<YourSecretEnvVar>"
                        -mode=DeleteBinary
                        -BuildVersion="<VersionToDelete>"
```

Description of parameters:

- `OrganizationID`
  - Use the Organization ID string that was provided along with your credentials.
- `ProductID`
  - Use the Product ID string that was provided along with your credentials.
- `ArtifactID`
  - Specify the Artifact ID associated with the binary to be labeled.
- `ClientId`
  - See Authentication section.
- `ClientSecretEnvVar`
  - See Authentication section.
- `BuildVersion`
  - The version string of the binary that should be labeled. A binary with this version must already have been registered, or the labeling operation will be rejected.

How to Copy a Binary

The CopyBinary mode allows you to copy binaries from one artifact to another without having to re-upload the binary. This is useful when you want to move a final binary from a Staging artifact to a Live artifact.

```
Engine\Binaries\Win64>BuildPatchTool.exe
                        -OrganizationId="<YourOrg>"
                        -ProductId="<YourProduct>"
                        -ClientId="<YourClientId>"
                        -ClientSecretEnvVar="<YourSecretEnvVar>"
                        -mode=CopyBinary
                        -BuildVersion="<VersionToCopy>"
                        -SourceArtifactId="<SourceArtifact>"
                        -DestArtifactId="<DestArtifact>"
```

Description of parameters:

- `OrganizationID`
  - Use the Organization ID string that was provided along with your credentials.
- `ProductID`
  - Use the Product ID string that was provided along with your credentials.
- `ClientId`
  - See Authentication section.
- `ClientSecretEnvVar`
  - See Authentication section.
- `BuildVersion`
  - The version string of the binary that should be labeled. A binary with this version must already have been registered, or the labeling operation will be rejected.

- `SourceArtifactId`
  - Specifies the ID of the artifact the binary is being copied from
- `DestArtifactId`
  - Specifies the ID of the artifact the binary is being copied to

How to Diff Binaries

The *DiffBinaries* Tool Mode calculates the difference between two binaries that were already uploaded and outputs statistics for applying the update between them.

```
Engine\Binaries\Win64>BuildPatchTool.exe
                        -OrganizationId="<YourOrg>"
                        -ProductId="<YourProduct>"
                        -ArtifactId="<YourArtifact>"
                        -ClientId="<YourClientId>"
                        -ClientSecretEnvVar="<YourSecretEnvVar>"
                        -mode=DiffBinaries
                        -BuildVersionA="<FirstBuildVersionToCompare>"
                        -BuildVersionB="<SecondBuildVersionToCompare>"

                        Optional:
                        -InstallTagsA="<FirstInstallTagSet>"
                        -InstallTagsB="<SecondInstallTagSet>"
                        -CompareTagSet="<DiffInstallTagSet>"
                        -OutputFile="<OutputFileName>"
```

Description of parameters:

- `OrganizationID`
  - Use the Organization ID string that was provided along with your credentials.
- `ProductID`
  - Use the Product ID string that was provided along with your credentials.
- `ArtifactID`
  - Specify the Artifact ID associated with the binary to be labeled.
- `ClientId`
  - See Authentication section.
- `ClientSecretEnvVar`
  - See Authentication section.
- `BuildVersionA`
  - The version string for the base binary image.
- `BuildVersionB`
  - The version string for the binary image to be updated.
- `InstallTagsA` *(Optional)*
  - Specifies in quotes a comma separated list of install tags used on BuildVersionA. You should include an empty string if you want to count untagged files.
  - Leaving the parameter out will use all files.
  - -InstallTagsA="" will be untagged files only.
  - -InstallTagsA=",tag" will be untagged files plus files tagged with 'tag'.
  - -InstallTagsA="tag" will be files tagged with 'tag' only.

- `InstallTagsB` *(Optional)*

    - Specifies in quotes a comma separated list of install tags used on BuildVersionB. Same rules apply as InstallTagsA.
- `CompareTagSet` *(Optional)*
    - Specifies in quotes a comma separated list of install tags used to calculate differential statistics between the binaries. Multiple lists are allowed. Same rules apply as InstallTagsA.
- `OutputFile` *(Optional)*
    - The differential will be output to the specified file as a JSON object.

Reducing Update Size

Requires v1.4.0 and above

BuildPatchTool uses a general patching system for all binaries uploaded using the *UploadBinary* mode. The system allows the Epic Games store to update any version of your binary on a user's machine to any other version, minimising the download size to do so.

The *BinaryDeltaOptimise* mode is designed to take two binaries already uploaded to the Epic Services and produce a smaller download for users who are updating between these two specific versions. We call this an A-to-B patch. Running this command is an optional step that can be taken to improve the user experience when releasing a larger than usual patch, with the majority of your players known to be on a specific binary version (binary "A").

The following is a sample command line to improve delta patching between two binaries:

```
Engine\Binaries\Win64>BuildPatchTool.exe
                        -OrganizationId="<YourOrg>"
                        -ProductId="<YourProduct>"
                        -ArtifactId="<YourArtifact>"
                        -ClientId="<YourClientId>"
                        -ClientSecretEnvVar="<YourSecretEnvVar>"
                        -mode=BinaryDeltaOptimise
                        -CloudDir="<YourCloudDir>"
                        -BuildVersionA="<YourSourceBuildVersion>"
                        -BuildVersionB="<YourDestinationBuildVersion>"

                        Optional
                        -DiffAbortThreshold="<UpperLimitInBytes>"
```

Description of parameters:

- `OrganizationID`
  - Use the Organization ID string that was provided along with your credentials.
- `ProductID`
  - Use the Product ID string that was provided along with your credentials.
- `ArtifactID`
  - Specify the Artifact ID associated with the binary to be labeled.
- `ClientId`
  - See Authentication section.
- `ClientSecretEnvVar`
  - See Authentication section.
- `CloudDir`
  - A directory where BuildPatchTool can save files to be uploaded, this can be empty each run. As with the BuildRoot, this can be an absolute or a relative path. (This location is used to cache information about existing binaries, and should be a different directory from the *BuildRoot* parameter. It is OK if this directory is initially empty; BuildPatchTool will download information as needed from the Epic backend and store it in the CloudDir.)

- `BuildVersionA`
    - The version string for the base binary image.
- `BuildVersionB`
    - The version string for the destination binary image.
- `DiffAbortThreshold` *`(Optional)`*
    - Specified in bytes, an upper limit for original diffs to try to enhance. This allows short circuiting lengthy optimisation attempts on large diffs which may not benefit. Range accepted is n >= 1GB, defaults to never abort.

When running in *BinaryDeltaOptimise* mode, BuildPatchTool will stream the existing binary data from our cloud network, re-analyze, and upload new A-to-B specific patch data. This process can take longer than the standard PatchGeneration mode if the original download size between `BuildVersionA` and `BuildVersionB` is large.

It is generally advised to run *BinaryDeltaOptimise* mode using your current Live binary as BuildVersionA, and a binary you will release next as BuildVersionB. Ideally, plan to complete this process several days before your release in case of any issues running the tool, such as network loss.

To check on your current Live binary version, you can make use of the *ListBinaries* mode.
If *BinaryDeltaOptimise* was already run between the two provided BuildVersion values, the process will succeed short circuiting the operation, indicated in the output.

```
Running optimisation for patching 1.3.0-Windows -> 1.3.1-Windows
...
** Chunk delta optimization already completed for provided manifests. **
Loaded optimised delta file
E:/CloudDir/Deltas/jd63jdu7-jsg58dh58dki8dk323/a5nbd0jbweof98fvytsvnthcsvf.delta
```

Once the processing is completed, the output will show the improvement achieved.

- "Final unknown compressed bytes, plus meta [Bytes]"
    - This will refer to the new download size achieved for patching between BuildVersionA and BuildVersionB.
- "Original unknown compressed bytes [Bytes]"
    - This will refer to the download size for patching between BuildVersionA and BuildVersionB before the optimisation was run.
- "Improvement: [Percentage]"
    - This will show the improvement that was achieved. The percentage refers to the size of the reduction, larger is better.

Example output:

```
...
Final unknown compressed bytes, plus meta 400000
Original unknown compressed bytes         1000000
Improvement: 60%
```

Downloadable Content (DLC)

The Epic Games store provides full support for Downloadable Content (DLC). **Please contact your Epic Service Delivery Team representatives to get artifacts for your DLC items provisioned, as they require additional up-front configuration that differs from non-DLC items.**

Once your DLC artifacts have been pre-provisioned in the Epic backend, the operation of BuildPatchTool is exactly the same for DLC as it is for full games, with the following additional notes:

Each item of DLC will be given its own, unique Artifact ID. Accordingly, each item of DLC will need its own patch generation process to be executed with the appropriate Artifact ID specified. Because each is generated separately, then it is important when performing patch generation for a DLC item that the base game, and all other DLC items are not present at the same time in the folder specified in the BuildRoot parameter during the patch generation process. Alternatively, if the BuildRoot folder does contain the main game or other DLC items, then a -FileIgnoreList parameter can be specified on the BuildPatchTool command line to exclude them from being included in the generated binary (see the How to Upload a Binary for more details).

When installed on end users' machines, the main game and all items of related DLC will be installed into the same directory. This leads to the restriction that there must be no files within the main game or any item of DLC which share a common file path. The BuildPatchTool will perform checks on this and prevent the patch generation process from occurring if there are any files which violate this rule. This means that the base game must be able to locate installed DLC by relative path from its install location.

In the short-term, Artifact IDs for DLC items will be allocated to you by Epic, and the association between the main game and the DLC item will be provisioned as part of that process. Longer term, we will look to incorporate this provisioning into a self-service operation on the Developer Portal.

Troubleshooting Common Errors

Check these first

- Check Path syntax - `BuildRoot="D:\MyFolder\"` `WILL NOT WORK`
- Ensure spaces have been added between all arguments.
- Files included in the build are subject to Windows MAX_PATH limits (limit of 260 characters); if the upload process is failing unexpectedly, ensure all file path locations do not exceed this limit.
- Most BuildPatchTool arguments are typically provided as absolute paths or relative to the working directory, but the AppLaunch argument must be provided only relative to the BuildRoot location.
- CloudDir contents will grow over time as more builds are processed on the same machine. However, this folder can be cleared at any time without negative impact.

Common errors

When using the BPT, Make sure your command lines are exact. If you do not use exact commands, you will receive a specific error message.
Please be aware these are only the common errors.

- Error - Failed to retrieve list of binaries from Epic services
- Error - Missing parameters
- Error - Missing credentials
- Error - Authentication error
- Error - Unexpected error

Other things to check

- The Client ID and Client Secret being used were provided by Epic specifically for BuildPatchTool use; EOS Client IDs (typically starting with **xyza**) cannot be used with the BuildPatchTool
- All required parameters are provided
- Parameters are free of typos, including incorrect capitalization or spacing issues
- Missing quotation marks for a parameter that includes spaces

Error - Failed to retrieve list of binaries from Epic services

```
Please check your provided OrganizationId, ProductId and ArtifactId are correct, and
that you should have access to create binaries for this artifact.
Tool exited with: 6
```

What it means
This means that the Client credentials you are using do not grant access to upload to the specified artifact. This could be a typo in the credentials or another parameter, or an issue with the permissions granted to the Client.

Error - Missing parameters

```
CloudDir, BuildRoot, ArtifactId, BuildVersion, AppLaunch, and AppArgs are required
parameters
Tool exited with: 2
```

What it means
This simply means that the BPT is not detecting one or more of the parameters that are listed above. This could simply be that they are missing, or that they were added incorrectly.

Error - Missing credentials

```
Please ensure you have specified ClientId and also one of ClientSecret or
ClientSecretEnvVar. If using ClientSecretEnvVar, ensure the env variable is properly
set.
Tool exited with: 9
```

What it means

This simply means that the BPT is not detecting one or more of the parameters that are listed above. This could simply be that they are missing, or that they were added incorrectly.

Error - Authentication error

```
An authentication error occurred - Check you have specified ClientId, and either
ClientSecret or ClientSecretEnvVar parameters correctly. Message: Sorry the client
credentials you are using are invalid
Tool exited with: 10
```

What it means
This means that the Client credentials you are using do not match our records. This could be a typo in the credentials or another parameter, or an issue with the permissions granted to the Client.

Error - Unexpected error

```
An unexpected error occurred. Error code: errors.com.epicgames.common.missing_permission;
Error message: Sorry your login does not possess the permissions
'artifact:o-7kyv8uwtz74njnl544kbxzt5xqephw:3bc138d96fb24d9597d4bbff23b9ecee:asdfasd READ'
needed to perform the requested operation
Tool exited with: 5
```

What it means
This means that the Client credentials you are using do not have access to the artifact you have specified. This is usually the result of an issue with the permissions granted to the Client, and can be addressed by Service Delivery.