Setting Up Turnkey For Your Organization

Information about how to write Turnkey manifests and set up copy providers for your organization.

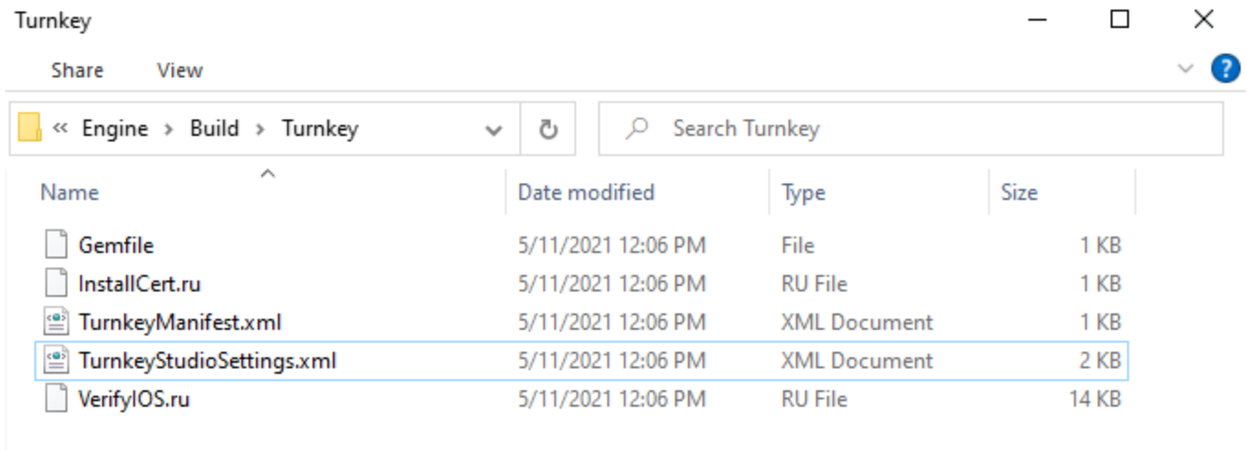
Platform owners distribute their SDKs to organizations on a case-by-case basis. While you still need to obtain your SDKs from a provider, you can place them in a common file source location for your organization. **Turnkey** can then access these files to download and install the SDKs to individual users' devices.

You can host SDK files for Turnkey in three different ways:

- On a Perforce repository for your organization.
- On a user's local machine.
- On a shared drive on Google Drive.

Turnkey then uses a series of chained XML files to determine which SDKs are available for each platform. This process starts by reading a `TurnkeyManifest.xml` file and a `TurnkeyStudioSettings.xml` file located in your **Unreal Engine (UE)** installation directory, under the `Engine/Build/Turnkey`

directory.



`TurnkeyManifest.xml` contains information about which SDKs are available, or points to other manifest files that contain the needed information, while `TurnkeyStudioSettings.xml` provides studio-wide information such as credentials needed to access Google Drive or signed certificates. Both `TurnkeyManifest.xml` and `TurnkeyStudioSettings.xml` can be chained to point to other `.xml` files in either local or remote directories.

In an ideal use case, your organization would distribute this `TurnkeyManifest.xml` to all its users alongside Unreal Engine source files within your version control system. This base manifest never changes, but instead contains references to other manifests located alongside your SDK files for each platform. Using this setup, you can quickly edit and update these manifests without altering any Unreal Engine source files.

Overview of a Manifest File

Turnkey manifests contain the following information:

- A `TurnkeyManifest` tag encompassing the entire manifest.
  - An `AdditionalManifests` tag
    - Individual `Manifest` entries pointing to any other manifests you want to include alongside the current one.
  - `FileSource` entries containing information about available SDKs, including:

- `Platform`
- `Type`
- `Version`
- `Name`
- `Source`

For example, the following would be an example of a working TurnkeyManifest supporting Win64 and Mac:

```
<?xml version="1.0" encoding="utf-8" ?>
<TurnkeyManifest>
<AdditionalManifests>
<Manifest>$(ThisManifestDir)/MyStudio_TurnkeyManifest.xml</Manifest>
<Manifest>$(UE_STUDIO_TURNKEY_LOCATION)</Manifest>
</AdditionalManifests>

<FileSource>
<Platform>Mac</Platform>
<Type>Full</Type>
<Version>$(ExpVersion)</Version>
<Name>MacOS SDK v$(ExpVersion)</Name>
<Source>fileexpansion:googledrive:/SdkInstallers/Mac/$[ExpVersion]/Installer.zip</Source>
</FileSource>

<FileSource>
<Platform>Win64</Platform>
<Type>Full</Type>
<Version>1.00</Version>
<Name>Win64 SDK V1.00</Name>
<Source>googledrive:/SdkInstallers/Win64/1.00/Installer.zip</Source>
</FileSource>
```

```
23
24 </TurnkeyManifest>
25
```

Copy full snippet

Providing Additional Manifests

The `AdditionalManifests` section contains individual `Manifest` entries that point to other XML files you want to include when Turnkey scans for files. The base `TurnkeyManifest.xml` in the `Engine/Build/Turnkey` directory is always the starting point for this process. and you can otherwise chain manifests as many times as you want, and you can organize them in any way.

The `Manifest` entries in this list follow the same formatting rules as file references for source files. Additional manifests can therefore be located anywhere that your source files are located, including remote repositories or a Google Drive folder. See the Formatting Rules section for more information about how to format a directory string.

FileSource Entries

FileSource entries list either specific SDKs that are available for your organization, or grouped SDK entries, depending on how you fill them out. The rules for each parameter within a FileSource entry are described below.

Platform

The name of the supported platform for this SDK. Turnkey uses this information to automatically determine the appropriate versioning rules for a given platform.

Platform names must match one of Unreal Engine's built-in platform names. For example, you should use "Win64" rather than "Windows," and "IOS" rather than "iPhone."

Type

The Type field describes whether you are listing a full SDK installation or a more minimal SDK, such as a flash kit. The valid SDK types are as follows:

Type	Description
Full	A complete installation of the SDK, as would be used by a developer.
AutoSDK	A package that Unreal Engine can use to configure your SDK installation as needed. Turnkey prefers AutoSDKs over Full SDKs when both are present.
Flash	A minimal installation with components for flashing a dev kit.

For more information on AutoSDKs, refer to the Unreal Engine AutoSDK Reference.

Version

The Version field contains the version number for the SDK. Every platform has a different expected version format, so it is important to list this information accurately.

You can capture the version number in the Source section by using a capture variable such as `$[ExpVersion]`, then list `$(ExpVersion)` in this section as a substitution variable. See the section on File Expansions and Capture Variables for more details.

The version number is also converted into an integer when Turnkey scans for SDKs. Turnkey uses different rules for each platform, and if the Full and Flash SDKs use different versioning formats, it takes those into account as well. When Turnkey compares multiple valid versions of an SDK, it will favor the highest converted integer value among the versions listed.

For information about each platform's expected version number formatting, check the Help command in the Turnkey commandline.

Name

The Name field contains a human-readable display name for your SDK. This is purely for display purposes, and can be set to anything you want.

The Name field will accept substitution variables corresponding to capture variables that you used elsewhere in the FileSource. For example, if you used `$[ExpVersion]` to represent the version number, the Name field can use `$(ExpVersion)` to substitute the value that the capture variable finds. For more information about capture and substitution variables, refer to the File Expansions and Capture Variables section.

Source

The Source field contains a string describing the location of the SDK. This string is separated into two parts: a set of prefixes that tell Turnkey how to read the file path, and the file path itself.

For example, the following would be a source entry that looks for the Win64 SDKs on the local file system (using the `file:` prefix) and expects a 7zip file called `Install.zip`:

```
<Source>file:X:\SdkInstallers\Win64\1.10\Install.zip</Source>

```

Copy full snippet

You can use the `fileexpansion:` prefix to enable **capture variables** to iterate through many SDKs with data from the file provider. For example, the following would be a source entry that uses the version number specified in the Version field:

```
<Source>fileexpansion:file:X:\SdkInstallers\Win64\$[ExpVersion]\Install.zip</Source>

```

Copy full snippet

Instead of looking at just one specific file path, it would look through every file path that matches this format.

The Source parameter can also use an optional `CopyOverride` specifier to restrict the directories Turnkey expands when using a Perforce repository. See the example below for a FileSource block that uses this specifier.

For a full list of prefixes and capture variables that are useable with source strings, refer to the Formatting Reference below.

Other Parameters

Although the above parameters are most commonly used, there are several others used in special cases.

AllowedFlashDeviceTypes

The `AllowedFlashDeviceTypes` value is only used for SDKs with the Flash type. These are strings whose values only make sense to the targeted platform. The possible values can be found in the Help section of the Turnkey command line, listed under the information for the given platform. When Turnkey receives device information, it will get the Device Type as well as the installed software version. It will then use these to determine if it is up to date and select a Flash FileSource to install from.

Example

The following is an example of a FileSource entry that looks for AutoSDK files on a Perforce repository. This example looks for `Setup.bat` and `Setup.sh` files to expand into one FileSource per SDK version, per platform, and uses `CopyOverride` to restrict Turnkey from expanding recursively unless the user explicitly chooses one of these SDKs. This prevents Turnkey from unnecessarily expanding on too many Perforce files. This behavior is specific to Perforce because it can only operate on files, not directories.

```
<FileSource>
<Platform>listexpansion:ExpPlatform=$(AutoSDKPlatforms)</Platform>
<Version>$(ExpVersion)</Version>
<Name>$(ExpPlatform) AutoSdk version $(ExpVersion)</Name>
<Type>AutoSdk</Type>
<Source CopyOverride="perforce://depot/CarefullyRedist/Host$(HOST_PLATFORM_NAME)/$(ExpPlatform)/$(Version)/...">
fileexpansion:perforce://depot/CarefullyRedist/Host$(HOST_PLATFORM_NAME)/$(ExpPlatform)/$[ExpVersion]/Setup.*
</Source>
</FileSource>

```

Copy full snippet

String Formatting Reference

Strings in Turnkey manifests use several prefixes to determine how to read the given file path. Depending on which prefixes you use, file paths can incorporate custom variables that expand into additional information at read time. This makes it possible for you to specify an expected format for a file path, rather than explicitly listing file paths for specific files, and Turnkey will discover all files that match that format.

Copy Providers

Copy provider prefixes determine what type of location Turnkey should look for when it parses the file path that follows it. Turnkey recognizes three types of locations, and the prefixes for these location types each have rules for what kind of file path they expect.

Prefix	Description	Example File Path

`file:`	Turnkey will look at the local file system for the SDK installers.
	
	This prefix expects a standard file path, including the drive where the file is located. This can include shared network drives.
	
	MacOS requires an additional `smb:` prefix to mount an SMB share.
		`file:X:\SdkInstallers\$[ExpPlatform]\$[ExpVersion].Install.zip`
		
		`file:smb:X:\SdkInstallers\$[ExpPlatform\$[ExpVersion].Install.zip`

`perforce:`	Turnkey will connect to a Perforce stream to find the SDK installers. This prefix expects a standard Perforce filespec, which must always include a specific file component, complete with a file extension. You may also be prompted to provide a clientspec the first time you use this to allow Turnkey to find the Perforce client's location.	`perforce:\\depot\SdkInstallers\$[ExpPlatform]\$[ExpVersion].Install.zip`

Prefix	Description	Example File Path
`googledrive:`	Turnkey will connect to Google Drive to locate the SDK installers. The first component in the file path represents the name of a shared drive to look in, while all subsequent components of the path are names of subfolders on that drive.	`googledrive:/SdkInstallers/$[ExpPlatform]/$[ExpVersion].Install.zip`

Using local files requires the least additional setup, as you can use SDK installers located directly on a user's machine. However, this setup requires the most additional maintenance, as each user must maintain a consistent local folder with the SDK installers located in it. You can use a shared network drive instead of a local drive on the user's machine, but depending on the speed of the shared drive, this might be slow.

Using Google Drive requires the most additional setup, as you must set up an account to use the Google Drive API and provide OAuth 2.0 credentials to users alongside their engine files. However, this setup provides a simpler point of maintenance for your organization's administrators once you have it set up. Refer to the guide on Setting Up Google Drive for Unreal Turnkey for more information.

Using a Perforce stream provides a balanced solution between these two methods, as it creates a central location for your organization to keep SDK files, but requires your organization to set up a Perforce stream and provide users with credentials to access it.

If you need to add support for other source control options to Turnkey, it is possible to add new types of copy providers with changes to the engine source code. See `GoogleCopyProvider`, `PerforceCopyProvider`, and `NullCopyProvider` for examples.

File Expansions and Capture Variables

The `fileexpansion:` prefix provides a way for Turnkey to search for paths matching a general format rather than paths to specific SDKs. This reduces the number of server calls for SDKs that you do not need to download, and tracks a wide range of SDKs rather than needing to write all their entries individually.

For example, you can upload a new installer to your server, and as long as its directory structure fits the expected format listed in the FileSource entry, the file expansion will discover it without needing to modify the manifest.

When using the `fileexpansion:` prefix, you must follow it with a copy provider prefix such as `file:` or `googledrive:` as file expansions depend on information from the copy provider.

After adding `fileexpansion:` to your path, you can then substitute parts of your path with **capture variables**. These variables use a format `$[xyz]`, where xyz is replaced with any arbitrary string you define. When Turnkey reads through the manifest, it will expand these with information from the user and the file provider.

As an example, if you use the following source string:

```
fileexpansion:file:X:\Installers\$[ExpPlatform]\$[ExpVersion]\Install.bat

```

Copy full snippet

Turnkey will convert the `$[ExpPlatform] and $[ExpVersion]` capture variables to wildcard values (the `*` character). Turnkey will then pass those to the copy provider used after the `fileexpansion:` prefix, in this case a local file system. The provider then iterates through its available files and folders and returns a list of results that match the provided format, along with the values captured for each capture variable.

```
X:\Installers\Win64\1.00\Install.bat [ $(ExpPlatform) = Win64, $(ExpVersion=1.00)]
X:\Installers\Win64\2.00\Install.bat [ $(ExpPlatform) = Win64, $(ExpVersion=2.00)]
X:\Installers\Android\10.1a\Install.bat [ $(ExpPlatform) = Android, $(ExpVersion=10.1a)]

```

Copy full snippet

Turnkey creates a FileSource objects in C# for each of these entries, which can then be discovered as SDK installer download sources. The portions of the file path expanded on by the capture variables will be converted into `$(xyz)` style **replacement variables**, which the FileSource object can reference in other

fields. For example, if you use `$[ExpVersion]` in a source path, you can also use `$(ExpVersion)` in the Version field.

Capture variables can represent any part of a file path, including filenames, directories, or even parts of a name.

```
fileexpansion:file:X:\Installers\$[ExpPlatform]\Installer_$[ExpVersion].zip

```

Copy full snippet

For example, the above entry expects installers to exist in a folder for their platform, and it expects them to be contained in `.zip` files with names like `Installer_1.00.zip` or `Installer_2.00.zip`.

List Expansions

The `listexpansion:` prefix defines a variable with a list of possible values, which does not need to query a server. When you define a variable with `listexpansion`, you can use it in other entries in the FileSource.

For example, the following text defines a `listexpansion` under Platform, then refers to that list under `Name`:

```
<Platform>
listexpansion:ExpPlatform=Windows,Android,IOS
</Platform>
<Name>$(ExpPlatform) Installer</Name>

```

Copy full snippet

When Turnkey reads this, it will create separate FileSource objects for Windows, Android, and iOS, and substitute `$(ExpPlatform)` appropriately for each one. This functionality does not require Turnkey to query a server, as it uses the list provided by the `listexpansion`. This can potentially make initial startup of

Turnkey faster, if the list of platforms with SDKs is generated from a list expansion.

You can also use an XML variable to store the list items rather than listing them directly.

```
<SavedSetting hostplatform="Win64">

<Variable>AllowedPlatforms</Variable>

<Value>Windows,Android</Value>

</SavedSetting>

...

<Platform>listexpansion:ExpPlatform=$(AllowedPlatforms)</Platform>

```

Copy full snippet

Built-In Variables

Turnkey reserves several built-in variables for defining file paths alongside user-defined capture and substitution variables.

Variable	Description
`$(ThisManifestDir)`	Expands to the directory of the `TurnkeyManifest.xml` file currently being processed. This includes the provider prefix for where the manifest is currently located, so if it was found in a Perforce directory, it would automatically contain the `perforce:` prefix without the need to define it yourself.

Variable	Description
	This variable makes it possible to relocate entire directory structures, as long as each directory's location relative to the manifest remains consistent.
`$(EngineDir)`	The root folder for your Unreal Engine installation. This is a useful point of reference if you are using files that are checked in to source control.
`$(HOST_PLATFORM_NAME)`	The name of the host platform. This will typically be Win64, Mac, or Linux.
`$(UE_STUDIO_TURNKEY_LOCATION)`	An environment variable that can be set to the location of a studio-wide Turnkey directory.