Packaging 101

UPack

Swift Kick

Kevin Griffin

December 12th, 2018

Agenda

1. What is UPack?
2. Building a Universal Package
3. DEMO: Building a Universal Package with UPack
4. UPack Command-line Interface
5. Package Validation
6. Repackaging
7. Universal Package Explorer
8. LAB: Installing a Universal Package with UPack

What is UPack?

Swift Kick

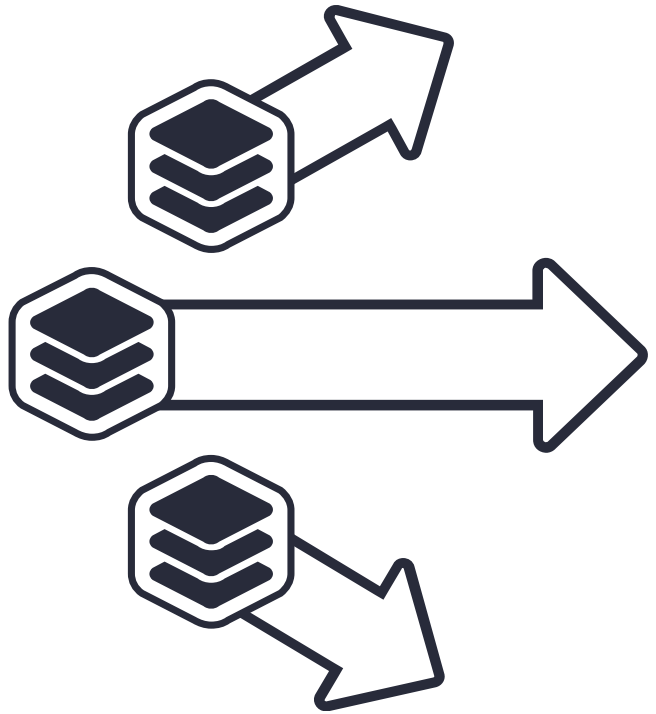
"UPack is a technology-neutral packaging platform that allows you to uniformly distribute your applications and components across environments to enable consistent deployment and testing."

Packaging 101 Recap

- Manually managing dependencies requires a lot of human effort and process
- By using packages, dependencies can be encapsulated and installed effortlessly
- Most major platforms have public feeds available to download packages
- Feeds aid in the discovery of new packages, and manage the versions of packages available to download

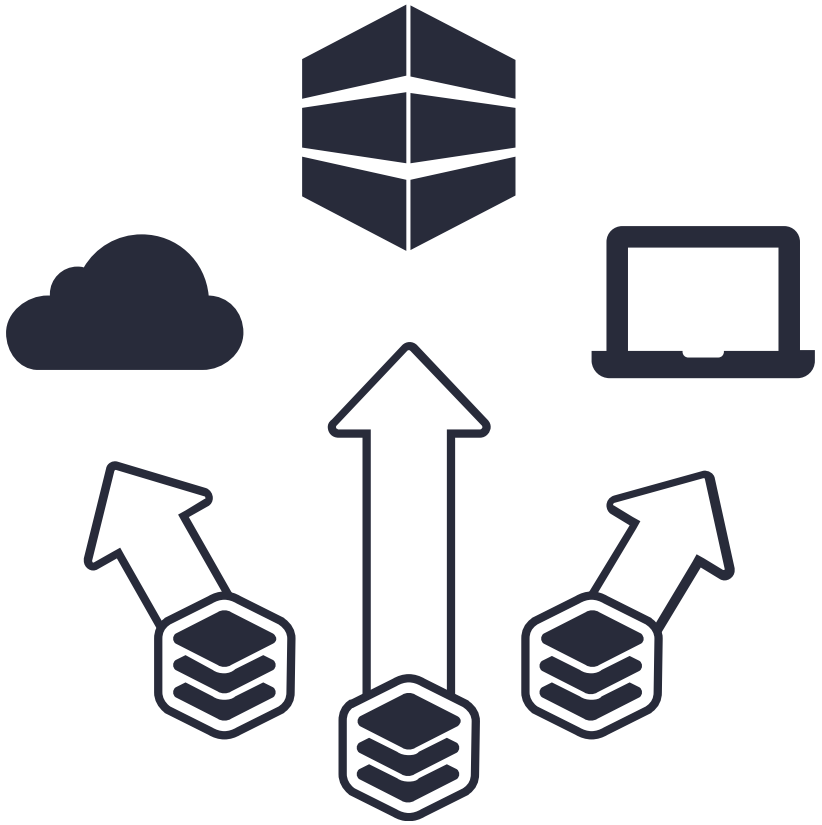
What is UPack?

- Universal packaging
  - Not specifically for .NET, JavaScript, etc
- Extensible
  - Custom metadata
- Easily Distributable
  - ProGet Universal Feeds
  - Local storage
  - Network share



What is UPack?

- Deployable
  - Deployed with Hedgehog or the UPack Command-line interface
- Discoverable
  - Local package discovery with Command-line interface
  - ProGet provides the ability to host UPack feeds.
  - Audit history



Where to get UPack

- https://inedo.com/upack/download to download
  - Command-line interface
  - Universal Package Explorer
  - Visual Studio Extension
  - UPackLib.Net library from .NET
  - Jenkins Plugin

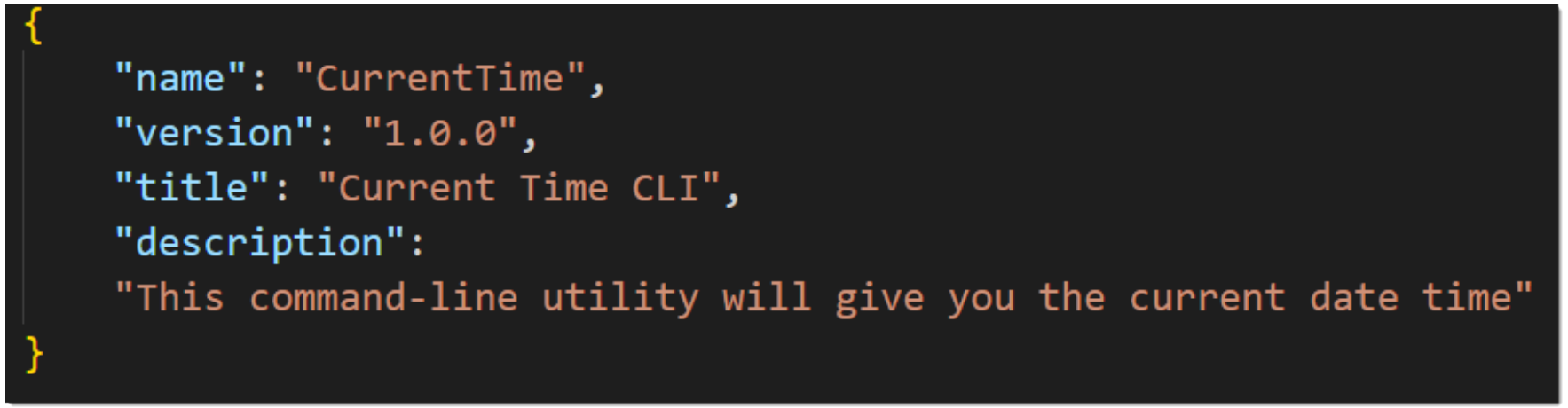
- Chocolatey (Command-line interface only)
  - `choco install upack`

What is inside a UPack file?

- A UPack file is a package file (zip) that contains metadata (upack.json) and a collection of files that would be unpacked when the package is installed.

- Metadata
  - The UPack metadata file contains all the information necessary for UPack and ProGet to properly manage and install the package.
  - JSON format
  - Called upack.json
- Files
  - Stored in the /package directory

Sample upack.json



```
{
    "name": "CurrentTime",
    "version": "1.0.0",
    "title": "Current Time CLI",
    "description":
    "This command-line utility will give you the current date time"
}
```

Swift Kick
SOFTWARE TRAINING AND CONSULTING

DEMO: Building a Universal Package

UPack Command-line Interface

Swift Kick

- upack pack <<source>>
  [--metadata=«metadata»]
  [--targetDirectory=«targetDirectory»]
  [--group=«group»]
  [--name=«name»]
  [--version=«version»]
  [--title=«title»]
  [--description=«description»]
  [--icon=«icon»]

UPack Command-line Interface

- `upack push «package» «target»`
  `[--user=«authentication»]`

UPack Command-line Interface

- `upack unpack «package» «target» [--overwrite]`

UPack Command-line Interface

- `upack metadata «package» [«version»]`
  `--source=«source»`
  `[--user=«authentication»]`
  `[--file=«file»]`

UPack Command-line Interface

- upack install «package» [«version»]
  --source=«source»
  --target=«target»
  [--user=«user»]
  [--comment=«comment»]
  [--overwrite]
  [--prerelease]
  [--userregistry]
  [--unregistered]
  [--cache]

Deploying Universal Packages

- Command-line tools
- Otter
  - Provision servers and manage their configuration by ensuring that certain packages are installed
- Hedgehog
  - Deploys your packages to the server and cloud
- UPackLib.Net
  - Programmically interact with and deploy packages

UPack Command-line Interface

- `upack list`
  `[--userregistry]`

Package Validation and Security

- Package version is based off Semantic Versioning (semver).
- Packages are meant to be read-only, and cryptographically verifiable.
- Each package has a SHA-1 based hash of the files inside of the package.
  - SHA-1 is cryptographically unique, like a fingerprint. If the files inside the package change, even a little, then the hash will change as well.

Package Validation and Security

- Why is this important?

- While it's increasingly difficult to do so – there have been cases in the past of package deployment credentials being compromised.
- If this happens, someone could attempt to upload and replace a package with malicious code.
- Many packages are not installed by humans, but by build systems and automated processes.

- By creating a hash of the package contents, we can ensure that the package has not been tampered with when installing.

UPack Command-line Interface

- `upack hash «package»`

UPack Command-line Interface

- `upack verify «package» «source» [--user=«user»]`

Swift Kick
SOFTWARE TRAINING AND CONSULTING

DEMO: Package Validation

Repackaging

- Because packages are read-only and cryptographically verified, how do we handle simple changes, such as promoting the version of a package from prerelease to release.

- Repackaging is a feature of UPack that allows this to happen without affecting the integrity of the packages.

UPack Command-line Interface

Swift Kick

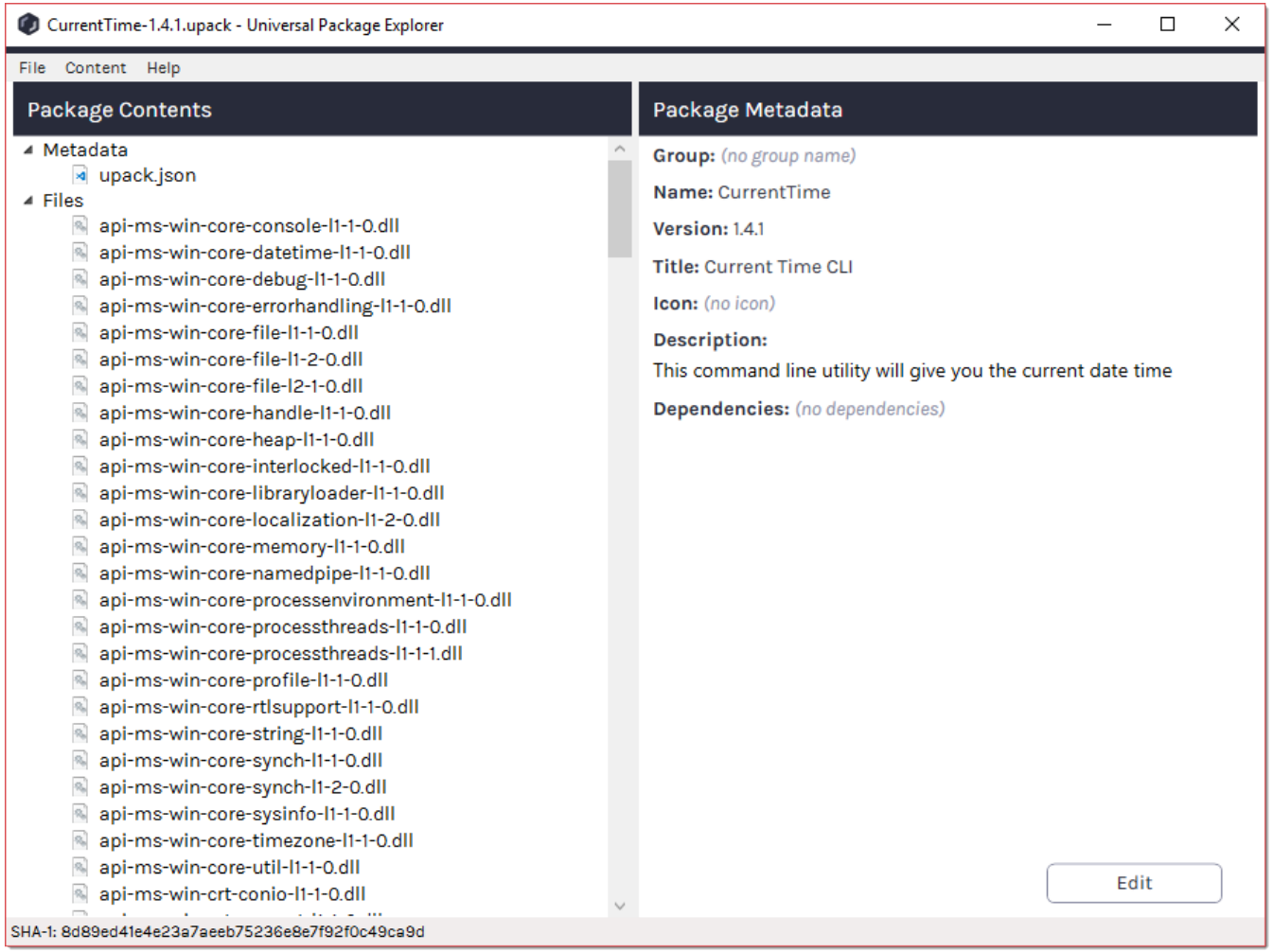
- upack repack «source»
  [--manifest=«manifest»]
  [--targetDirectory=«targetDirectory»]
  [--group=«group»]
  [--name=«name»]
  [--version=«version»]
  [--title=«title»]
  [--description=«description»]
  [--icon=«icon»]
  [--overwrite]

Swift Kick
SOFTWARE TRAINING AND CONSULTING

DEMO: Repackaging

Universal Package Explorer

- The Universal Package Explorer is a graphical interface for creating, viewing, and publishing universal packages.



Swift Kick
SOFTWARE TRAINING AND CONSULTING

DEMO: Universal Package Explorer

Swift Kick
SOFTWARE TRAINING AND CONSULTING

LAB: Creating and Deploying Universal Packages

Recap

- UPack is a universal package format that allows for consistent deployment across environments and platforms.
- UPack packages can be created through the Command Line Interface (CLI) or the Universal Package Explorer user interface.
- UPack enforces immutability and cryptographic verification with repackaging – making universal packages extremely secure and trustworthy.