DPM Conflict Resolutions

Dependency Resolution

Dependency resolution in DPM is fairly straight forward. DPM reads the `DEPENDENCIES` file from the package's metadata and then checks if a file whose `NAME`, `PROVIDES`, or `REPLACES` entries name a package with a version met by that named dependency.

The DON repository metadata provides larger dependency chains for larger package transactions, such as in the event of system updates are large framework installations, and this will be discussed more in the specification for DON.

In DPM, the behaviour is very simple if a dependency is not found: it will fail, unless an override switch to force the transaction is supplied.

Dependency Version Comparison Resolution

Version comparison will be an implementation of the behaviour documented in the public Semantic Versioning 2.0 (SemVer) specification, with a default patch number of 0 that increments with every official package created for that package version. So, if a typo is identified or some other change is made to the package, it increments by 1. If a backwards compatible patch is applied, it increments by 1 if it hasn't already. The patch here is used to indicate package versions of the same software version, and not necessarily patching state.

If a software's versioning scheme does not align with SemVer, it will be converted to a form that is, and this will be noted in the package description.

File Conflict Resolution

While undesired, it is possible for package creators to create more than one package that provides a file.

1. For instance, if `PackageA` is installed and provides a file at `$file_path`, and `PackageB` is being attempted to be installed and provides a different (or even the same) file at the same `$file_path`, this will create a file conflict.
   In this installation file conflict, DPM detects that `$file_path` is already owned by `PackageA`. DPM will then fail with an error that names the conflicting file, and which package (`PackageA`) owns it. It will identify the package (`PackageB`) that is trying to additionally provide it.

2. There are exceptions. In the event that `PackageB` lists `PackageA` in its `REPLACES` metadata, then it will replace the file along with any other files in `PackageA` and PackageA will be removed. This handles legitimate cases where one package is meant to replace another.
3. Another exception is if the user supplies a commandline option to force installation of PackageB:
   If `PackageA` lists the file as `non-controlled` and `PackageB` lists the file as `non-controlled`, it will place the file from `PackageB` at `${file_path}.dpmnew`.

   If `PackageA` lists the file as `controlled`, and `PackageB` lists the file as `non-controlled`, the original file will be preserved and `PackageB` will place the file at `${file_path}.dpmnew`.

   If `PackageA` lists the file as `non-controlled`, and `PackageB` lists the file as `controlled`, the original file at `${file_path}` will be renamed to `${file_path}.dpmbackup` and `PackageB`'s version will take ownership priority of the file at the conflicting path.

   If both packages list the file as controlled, the transaction will fail as described in (1). The user must uninstall `PackageA` and install `PackageB` or one of the packages must be modified and (re)installed to allow installation without a package conflict.