THE NEW STANDARD FOR MANAGING YOUR WORKSPACE

HENRI RETERINK

REMEMBER THIS…?

NOT THE BEST EXPERIENCE...

WE WANTED TO IMPROVE THIS

HUMBLE BEGINNINGS

HUMBLE BEGINNINGS

- Started as a random hobby / proof of concept project
- First shown at a WODAN
- Presented at EDUC in Iceland
- Terminal only
  - `dfpm install Quill`
- Fetched a library from a self-hosted “server” and
  - Made references to the library in your workspace
  - Copied the necessary JS / CSS / etc. files
  - Updated your index.html with these files

GOOD ENOUGH, SHIP IT… RIGHT?

NOT QUITE…

IT NEEDED TO BE WAY MORE MATURE

WISHLIST(S)

WISHLIST(S)

- Not just terminal
- Studio integration with a nice UI
- Versioning (and constraints)
- Updates
- Pushing and publishing packages (and versions)
- Server to serve packages
- Web UI (for management)
- Community-driven packages
- New sws format?

MANY, MANY DECEMBERS LATER

DataFlex

/*PACKAGE MANAGER*/

A QUICK TOUR

PACKAGE TYPES

PACKAGE TYPES
SERVER

Server package

- Most common package type
- Hosted on our servers (currently ***packages.dataflex.dev***)
- Simply browse and click “Install” to add to your workspace!
  - Handles .sws modifications
  - Installs files like .js and .css.
  - Updates index.html with these entries.
  - Etc.
- Automatically downloaded by PacMan upon opening a workspace
  - *(more on this in a bit)*
- Checks for updates and notifies you if upgrades are available

PACKAGE TYPES
LOCAL

Local package

- Packages / libraries that exist on your filesystem
- Very similar to “legacy” libraries.
- Added via absolute / relative pathing
- Does not perform .js, .css, etc. installations and index.html modifications
- When migrating to DF26, all existing libraries are automatically assumed to be LOCAL.

PACKAGE TYPES
GIT

GIT Package (yep!)

- Added by passing the URL to the repository
  - (can also be a local git repo!)
- Able to pass a specific branch, commit or tag to fetch.
  - Uses default branch (usually master / development / release) if nothing is passed
- Upon first install, gets the latest commit for that branch if available and not pinned to a tag or commit.
- Will not suggest updates (there's no hard git metric for this)

VERSIONING

VERSIONING

2 types to distinguish between:

- Package version
  - Version for the actual package, to indicate updates, new features, bugfixes etc.
  - Follows Semantic Versioning standard
    - *More on this in a bit*
- DataFlex version
  - DF Versions this package is compatible with
  - This matters much less than it previously did

DATAFLEX VERSIONS

VERSIONING
DATAFLEX VERSIONS

Before 26

- Library version had to match target workspace version
- Often times no real changes between DF versions.
- “Old” versions probably would work anyway.
- Led to library_23.sws, library_25.sws, etc.

After 26

- DF Version no longer really matters
- Local & Git packages: DF assumes the library just works.
  - *if not, you’ll probably find out when compiling ☺*
- Server packages: Server manages a DF Min and Max version
  - Installs the best matching version for your DF instance
  - Updates keep this in mind too.

PACKAGE VERSIONS

VERSIONING
PACKAGE VERSIONS

- Actual version for a package
  - To indicate major updates, new features, bugfixes, etc.
- Uses Semantic Versioning notation (SemVer)
  - Eg. 1.0.0, 1.4.5, 2.0.0-alpha1, 2.0.2, etc.
  - Broadly accepted standard for managing dependency hell
- Clicking install will attempt to install latest release
  - Can also install specific version
- Updates will always suggest / attempt latest possible released version.
  - Within constraints ☺

A BIT MORE ON SEMVER

Very basic overview here, in the training we looked at this more in depth

VERSIONING
SEMVER BASICS

2	5	1
Major	**Minor**	**Revision**
For breaking changes such as API overhauls.	Use this when adding new features	Bugfixes and small tweaks go here

VERSIONING
SEMVER (CONT)

- When simply clicking install, PacMan will install the latest version
- Can select a specific version to install
- Reflected in your sws and lock file, eg:

```
"dependencies": [
    “MyRepo/SomePackage#1.2.3"
]
```

- Things you can also do…
  - Use expressions!
  - Prevents updating to a version with breaking changes

```
"dependencies": [
    “MyRepo/SomePackage#<2.0.0"
]
```

NEW SWS FORMAT

Modern twist on a familiar face

- Still .sws, but now JSON format instead of INI
- Many familiar, existing settings live here, like the DF version
  - Lots of stuff hidden if the value is default
- Works together with config.ws for things like alternate paths.
- New entries to help PacMan like
  - List of dependencies
  - package information (with version and license subs)
- Studio handles almost everything related to the sws (both reading and writing)
  - Can modify manually and if syntactically correct, df-cli will consume it.
    - E.g. custom SemVer expressions in a dependency

DF-CLI???

DF-CLI

- New command-line powerhouse for anything DF related
- Foundation for a lot of DF workspace related actions
  - Opening a workspace
  - Building (compiling) projects
  - Installing packages
  - Pushing packages
  - Etc.
- Used by the studio and package manager (both client and server!)
- Bram has a cool presentation on this later using it inside a build system!

SERVER AND ADMIN PANEL

SERVER

- Found (currently) at **packages.dataflex.dev/api**
- Basically an API that
  - Handles all incoming traffic from the admin panel and studio (via df-cli)
  - Serves
    - overview of packages (search)
    - package details
    - suggest update candidates
    - package downloads
- We plan to release a clean API spec in the near future for custom integrations

ADMIN PANEL

- Found (currently) at **packages.dataflex.dev**
- Web environment to manage anything package related
- Discover new packages and versions
- Manage your own packages
  - Create your own repository!
    - Register your package
    - Upload (push) a version and publish it

PUTTING IT ALL TOGETHER

Let’s create, register, push and install a package start to finish.

MIGRATING TO 26

MIGRATING TO 26 AND BEYOND

- The migration wizard should simply handle this
  - In theory ☺
- Your SWS will be converted to the new format
- All libraries will be added as local libraries
- For Web / Flextron projects, studio will “suggest” to install the WebUI package

- Recommendation: Make sure your project lives inside a git repo (can be local)
  - Optionally create a new branch
  - If the wizard made a mess of your migration, you can easily roll it back via git

SYSTEM PACKAGES

How the package manager solves a key challenge for us

SYSTEM PACKAGES

- The problem
  - Small bugfixes in certain areas
  - Don’t always need a new DF version
  - We don’t want to wait until a next release
  - We want to get certain fixes and updates out asap

SYSTEM PACKAGES

- Contain core DataFlex functionality
  - E.g WebUI: contains the bulk of the DataFlex WebApp Framework
    - Individual “theme packages”
  - We intend to turn many more areas into packages over the next year or so
- Allows us to separate key areas and make the whole thing more modular
  - Makes your workspaces more lightweight – less unused code.
  - Allows us to update individual pieces outside of release cycles.
  - Gives you the freedom to potentially use (and stick with) older versions
    - *If you really need this*

COMMUNITY DRIVEN

COMMUNITY DRIVEN

- Did you know…
  - A lot of DF libraries also already live on GitHub
    - https://github.com/DataFlex-dev
  - You can easily install these using the package manager
  - We'd love for you to contribute
    - Create a fork to make your own tweaks and changes
    - Need a new feature or found a bug? Create an "issue"!
    - Created a new feature you'd like to share or fixed a bug? Open a "pull request"!
  - We have an extensive "GitHub contribution" Memo that we'll share later.

/* THANK YOU!
QUESTIONS? */