
SHOTGUN ECOSYSTEM
INTRODUCTION TO TOOLKIT CONFIGURATIONS
AUTODESK.

Tannaz Sassooni

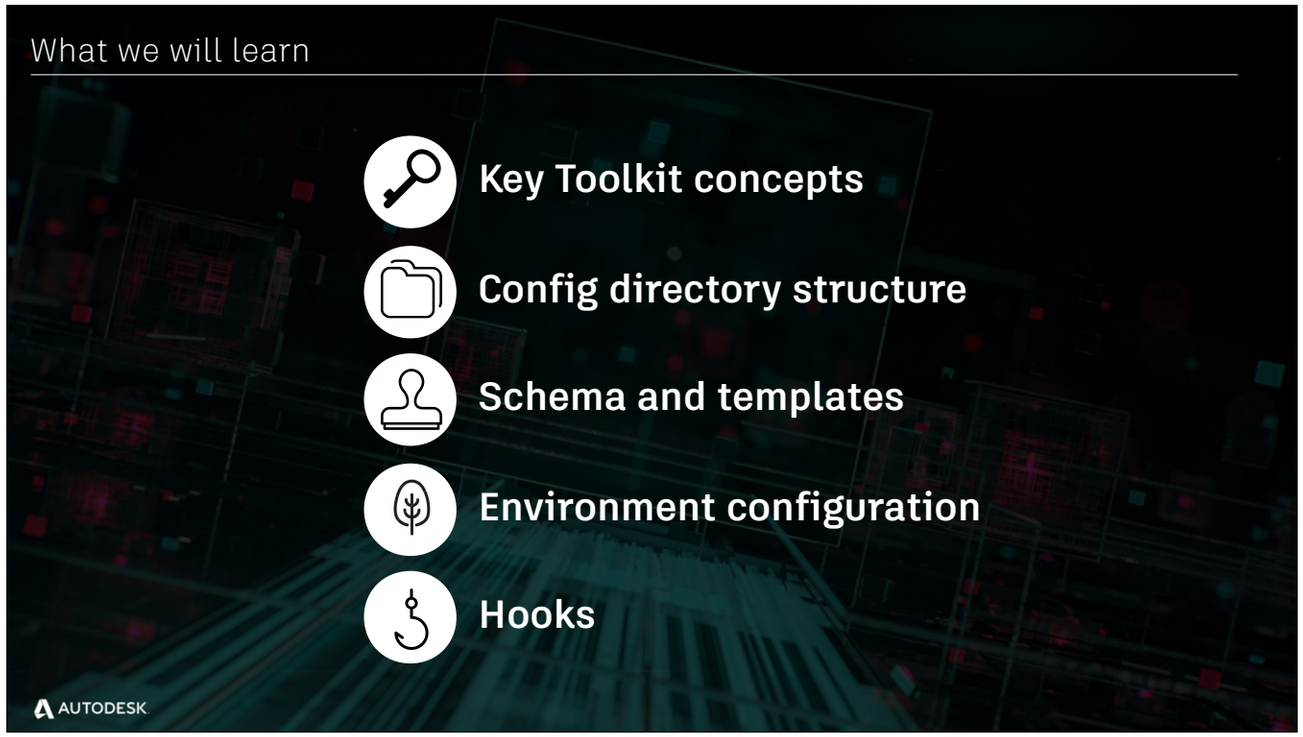
Street Team Toolkit Specialist

Tannaz has worked in visual effects and animation technology for over 14 years. At Rhythm+Hues Studios, she was a pipeline lead on films such as "The Chronicles of Narnia: The Lion, the Witch, and the Wardrobe" and "Evan Almighty". She worked at Dreamworks Animation for nine years, designing and implementing production workflows for layout, animation, modeling, and cloth and hair artists. For the last two years, she's been a member of the Shotgun Street Team's technical support group, specializing in the Toolkit platform.

AUTODESK.

DEV DAY RESOURCES

github.com/shotgunsoftware/sg-devday-2018



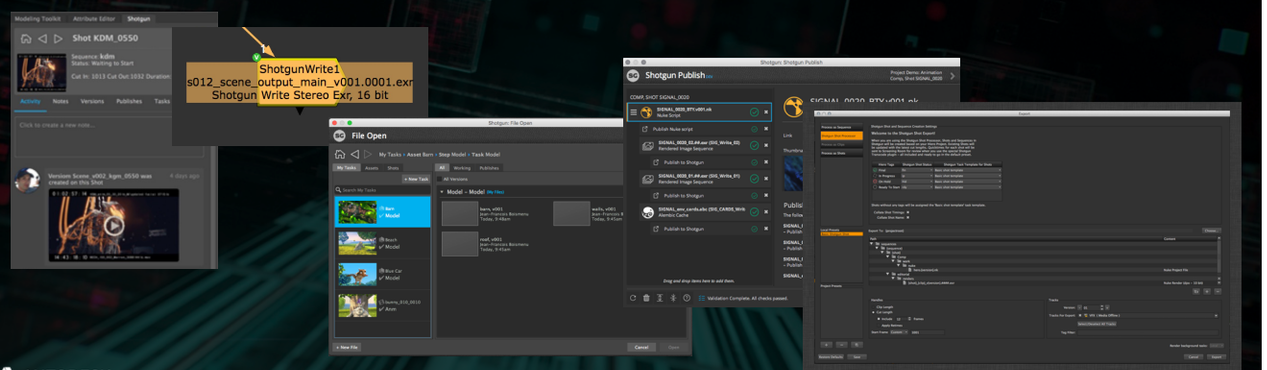
So Phil talked us through the basic integrations: install and go, nothing on disk to configure, Publish, Loader, Panel, easy peasy. He spoke a little about advanced configs: Say you want Toolkit to manage your folder structure. Or to add an app beyond the basic three, such as our special, shotgun-aware write node for Nuke, or to add a non-default integration, such as the just-announced Unreal Engine integration. At that point, you're ready to take over the pipeline configuration and begin customizing it. In this talk, we're going to dive in and look at these configs.

before we dive in to the configs, let’s talk about some key Toolkit concepts, and the notation we use for each in our configs, so you know what each looks like.


Key Concepts
• Apps
• Engines
• Environments
• Core
AUTODESK

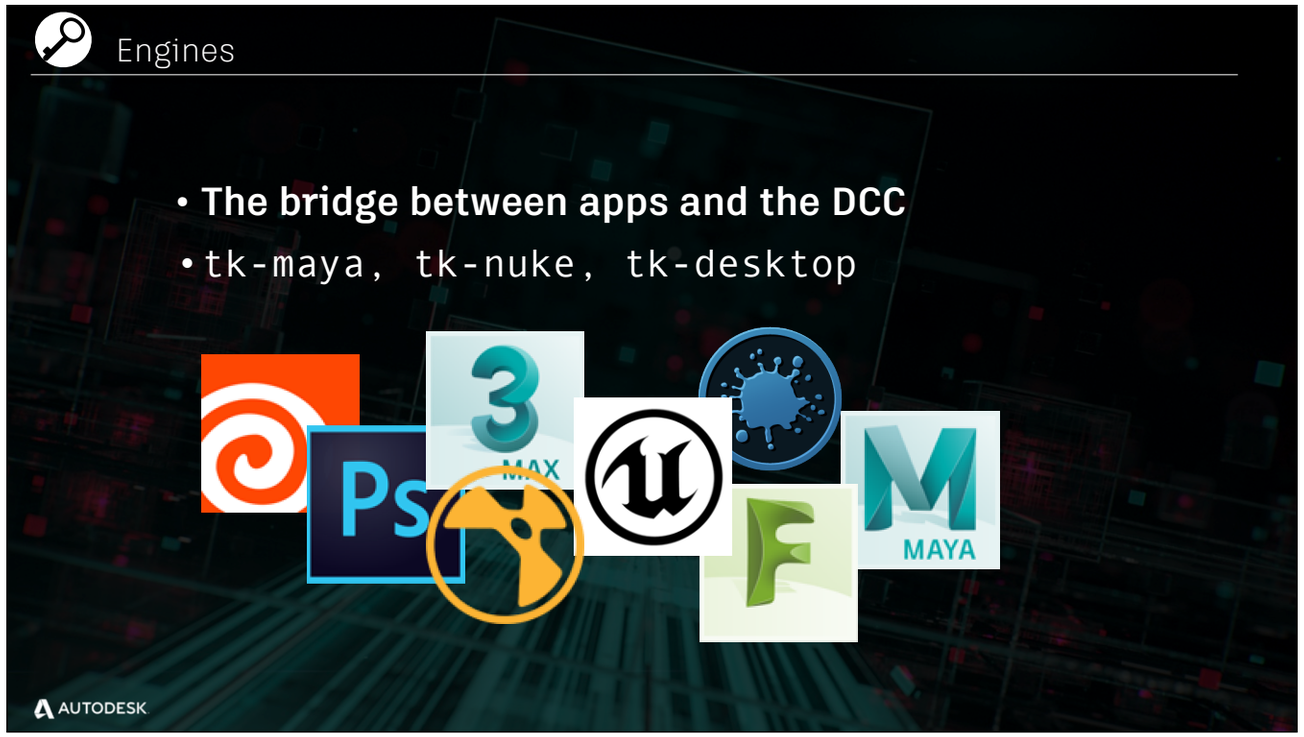
Apps

- **Public-facing user interfaces that do something pipeliney**
- `tk-multi-workfiles2, tk-nuke-writenode`



AUTODESK

An app is…
an app can be anything [CLICK]
a panel in maya [CLICK]
a node in a nuke script [CLICK]
a dialog [CLICK]
a standalone app [CLICK]
deep integration into a native dialog (eg, extra stuff in the Hiero export) [CLICK]
Notation



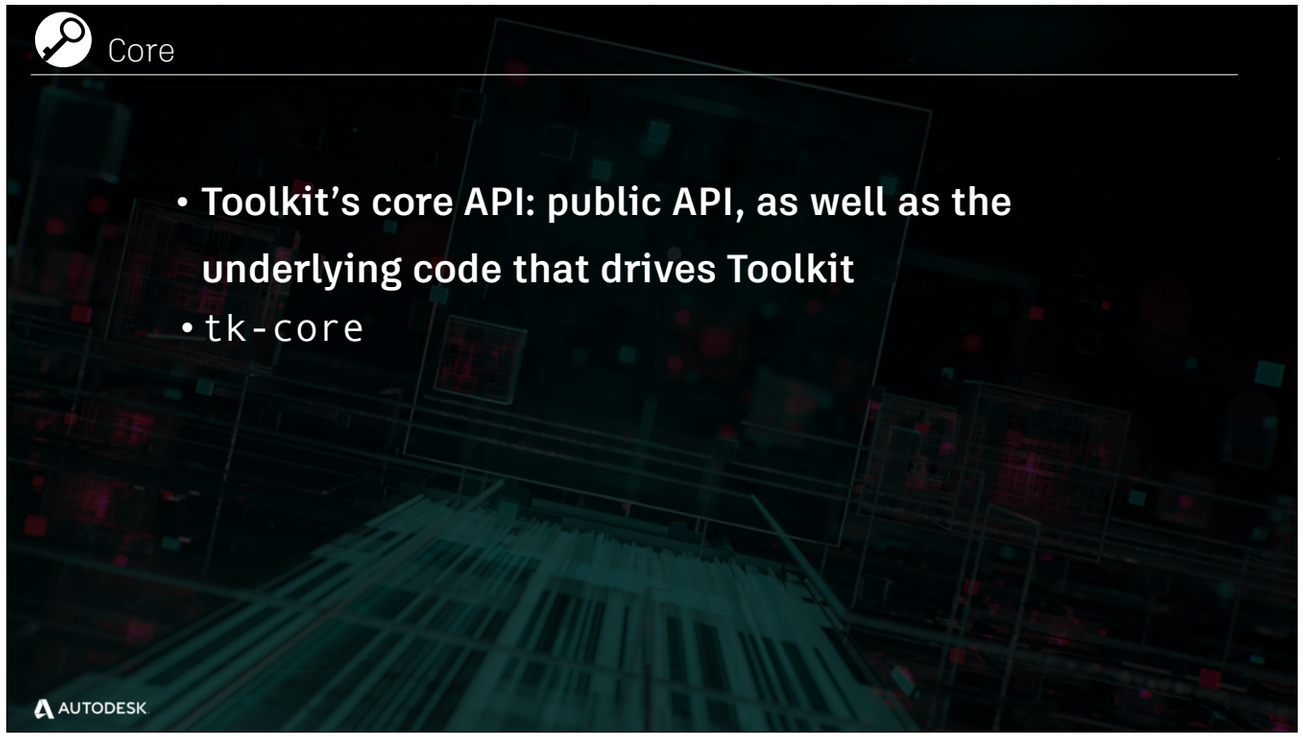
An engine is the bridge between Toolkit apps and the DCC. Our engines hold the DCC-specific logic, so that we can have multi-apps that behave the same from DCC to DCC.

There are some special engines, too: tk-shell, tk-shotgun, tk-desktop

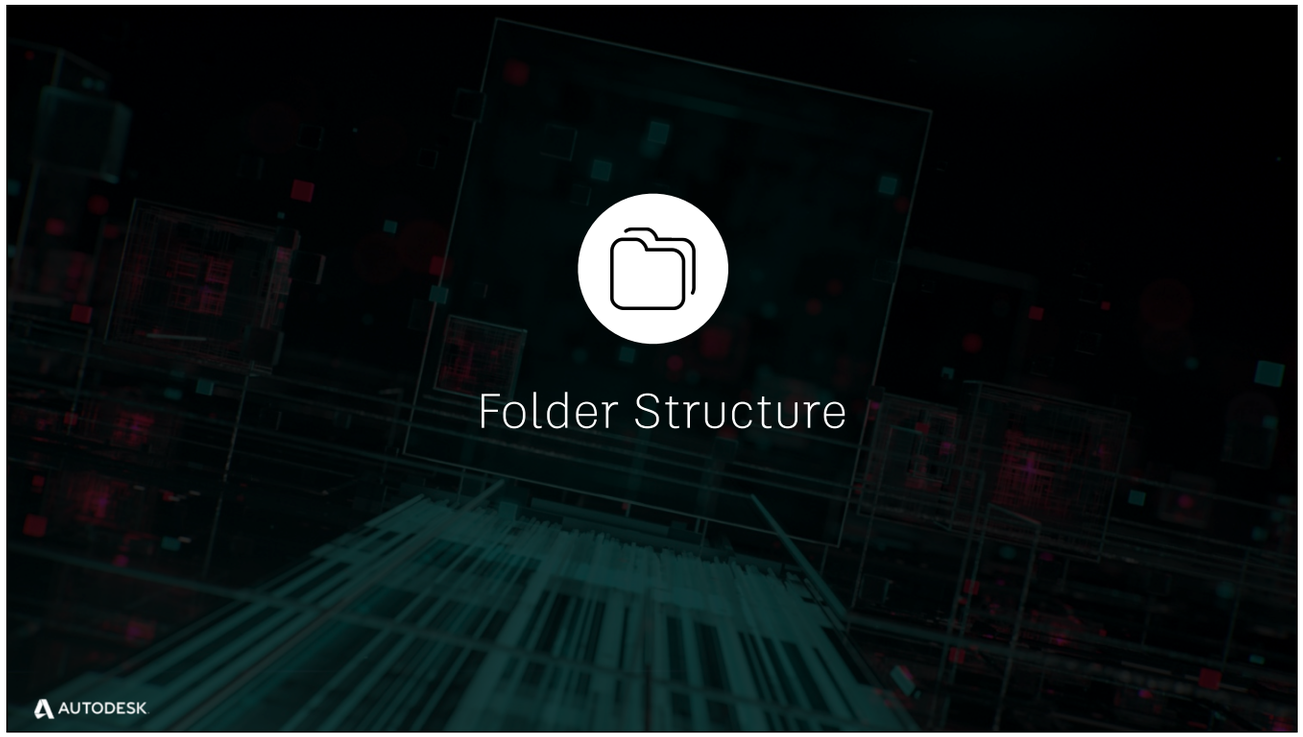
Notation

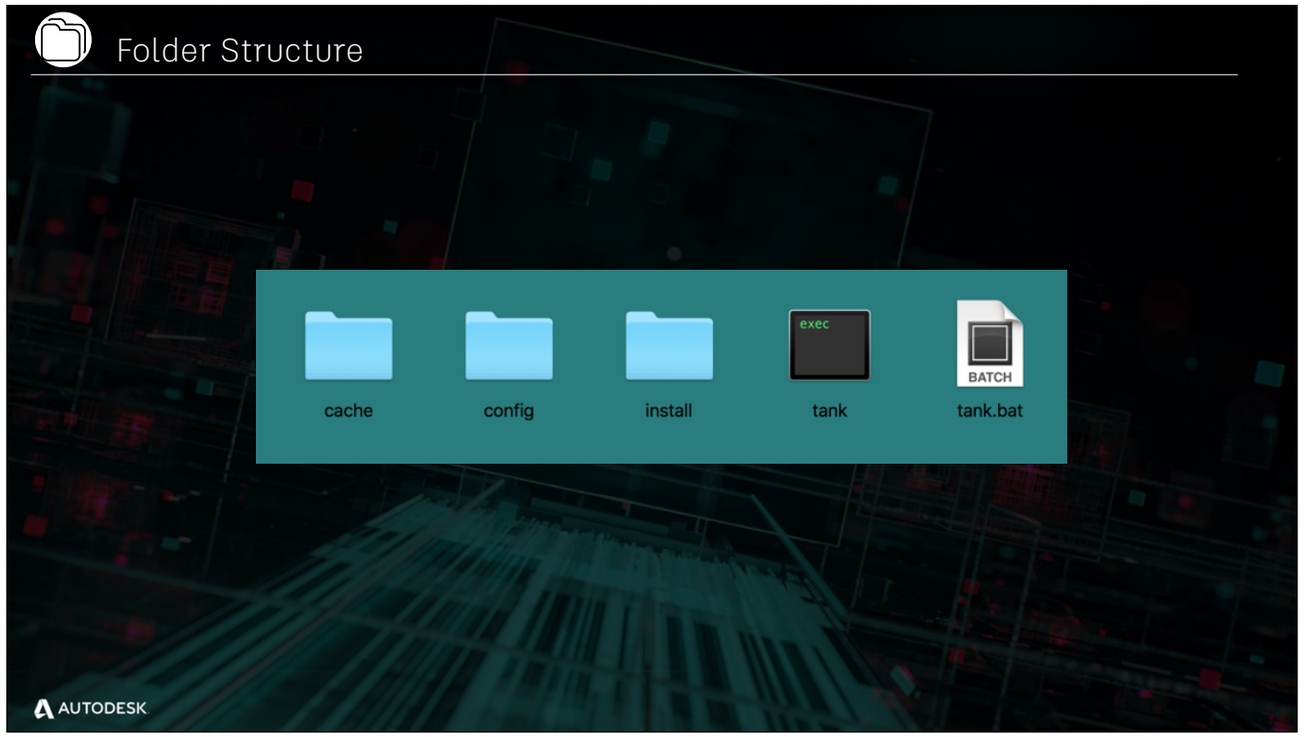
The context in which Toolkit is running, based on what you're working on. The toolkit configuration is organized by environment and why. Your context also determines how files should be named, and where they should go. But more on that later.

images of env from different places?
- file save
- work area info
- settings
- web ui task?
- shotgun menu

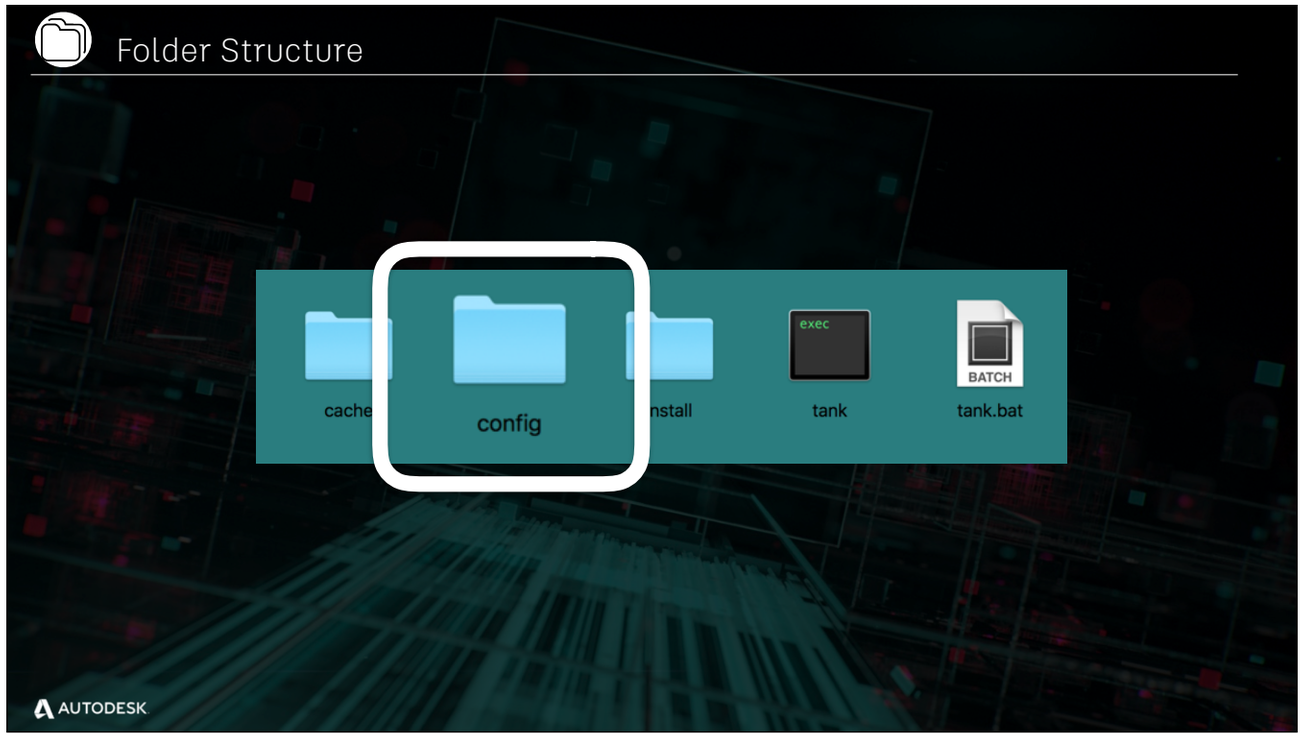


This is the foundation for all toolkit functionality.


Folder Structure
AUTODESK

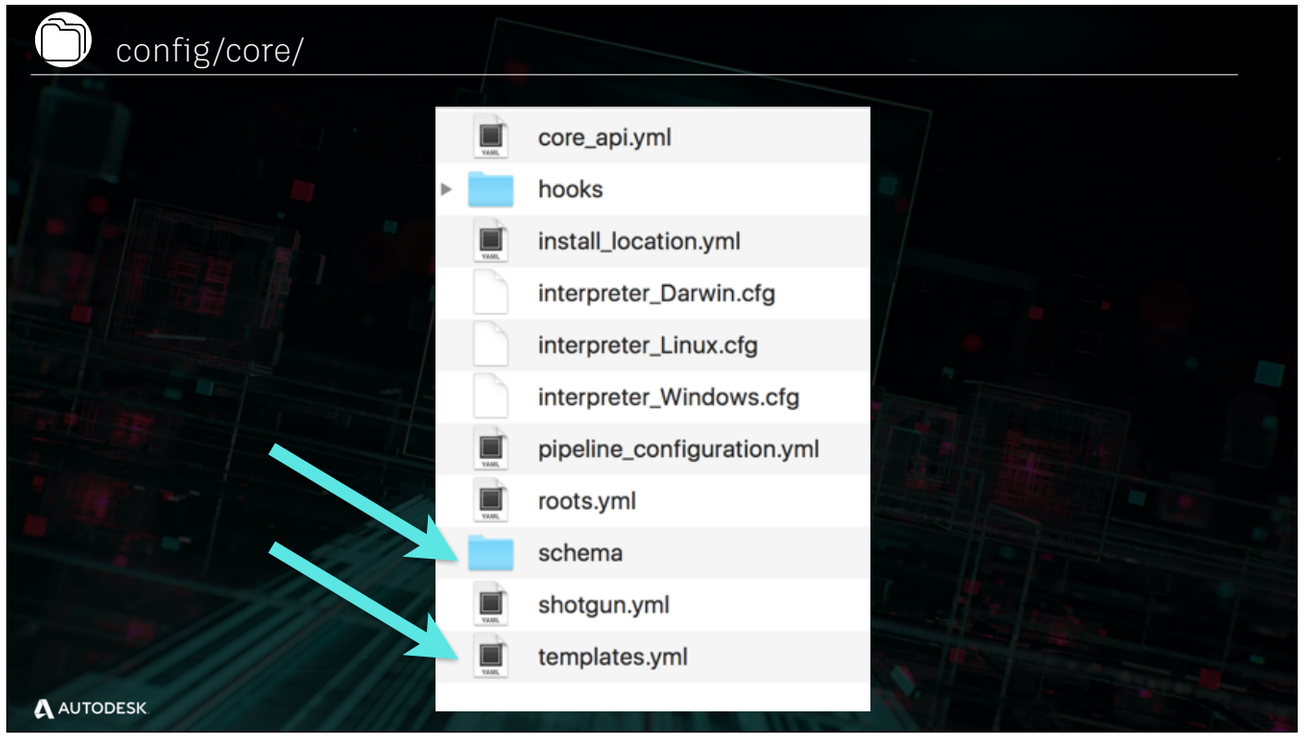


When you first open your pipeline config, you'll see 3 directories and a couple files. The cache dir is for internal use, and we're not gonna worry about it today. Config, let's skip for a moment. The install directory holds the toolkit core, as well as all the apps and engines your configuration is using. You have tools to manage it, so you shouldn't be modifying it directly. The tank command is a collection of commandline tools for managing your configuration, and tank.bat is just tank as a batch file for Windows.

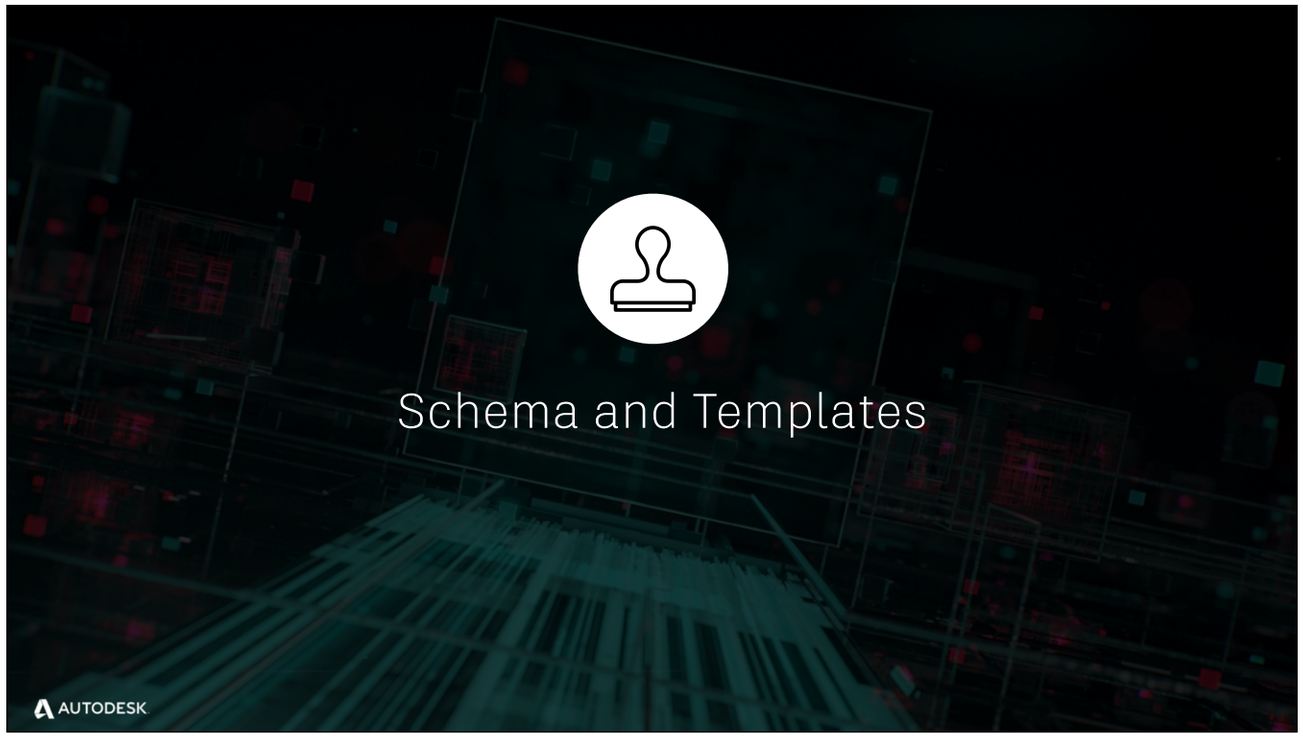


Which leaves us with the config directory, and that’s where we’ll dive in. config is where you’ll customize your Toolkit pipeline, through a set of yaml files and simple but extensible python scripts.

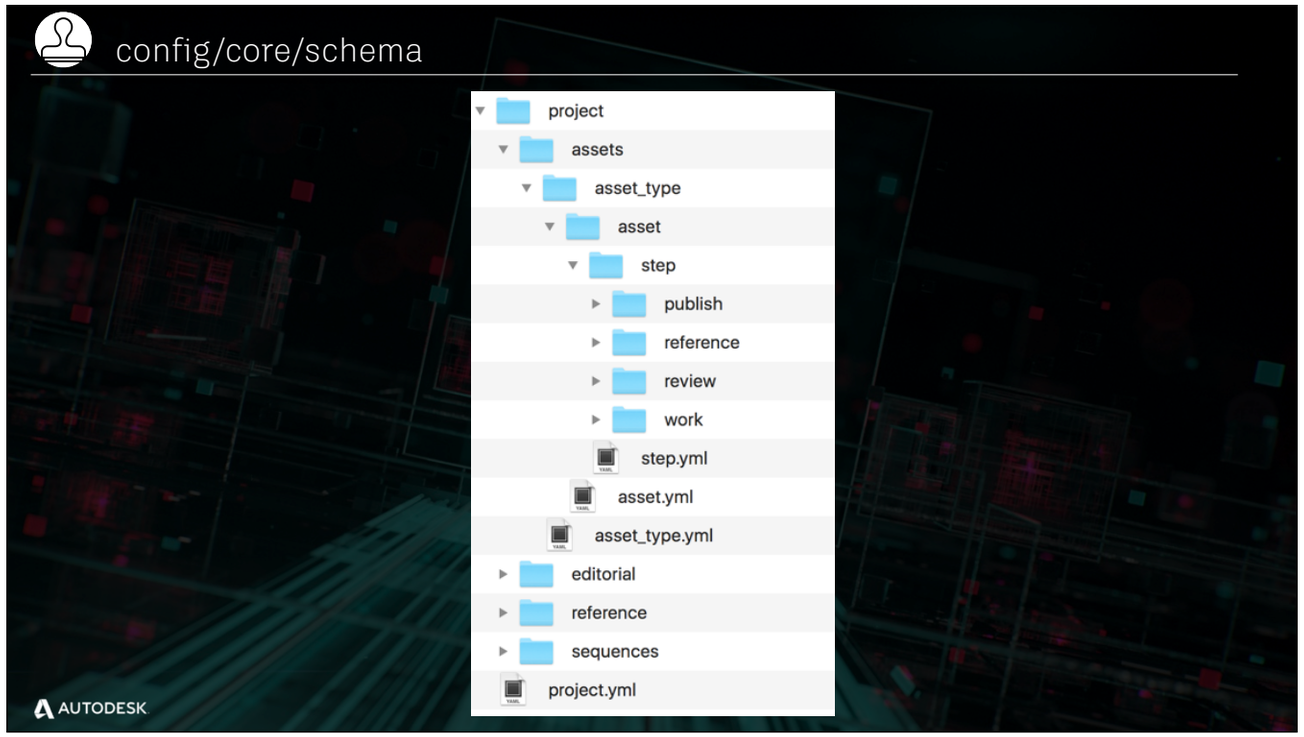
So what do we see when we step into the config dir? Some familiar words here? We know what core is, and we know what environments are. So, as you might guess, the env dir contains your per-environment configuration settings, while your core dir contains configurations for core as well as other non-environment specific stuff. So, let's dig into core first.



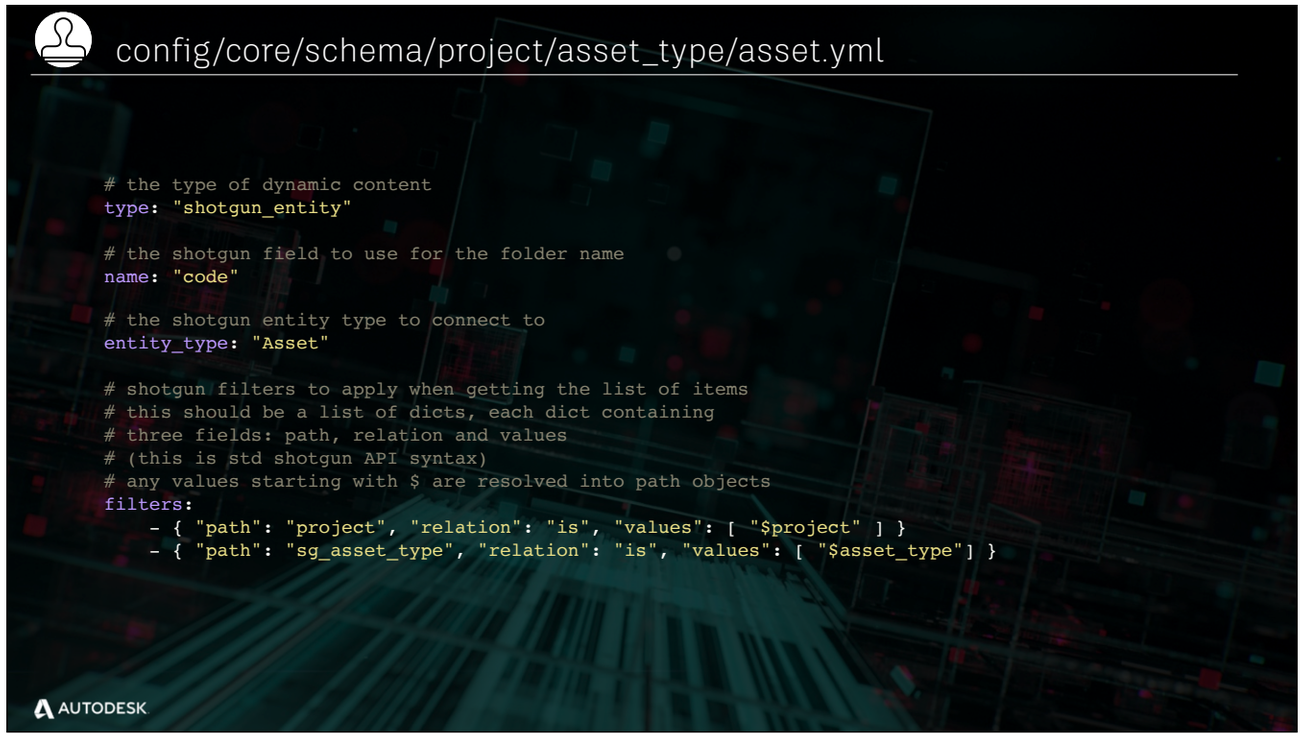
you see a bunch of stuff here, but we’re going to focus on two things: [CLICK] your schema, and [CLICK] your templates.



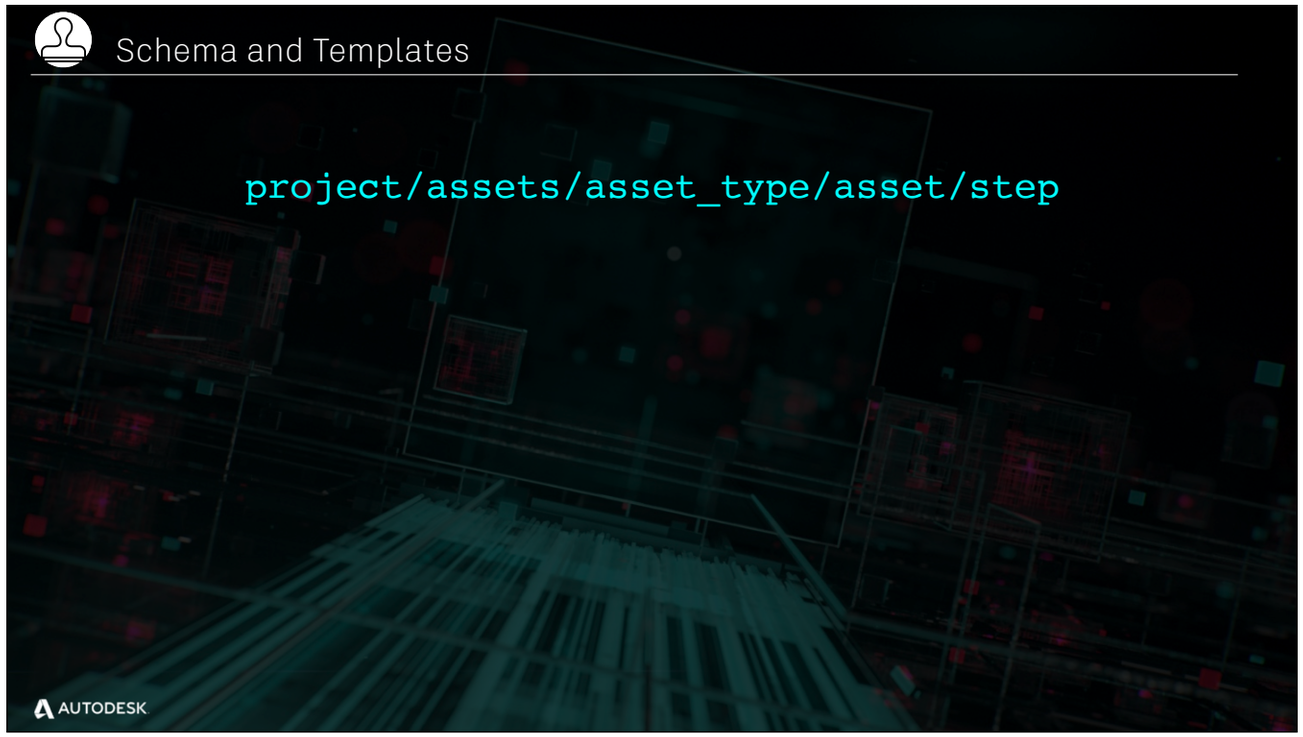
We’ve mentioned that Toolkit can manage your folder structure and file naming. These two components of the pipeline config do just that.



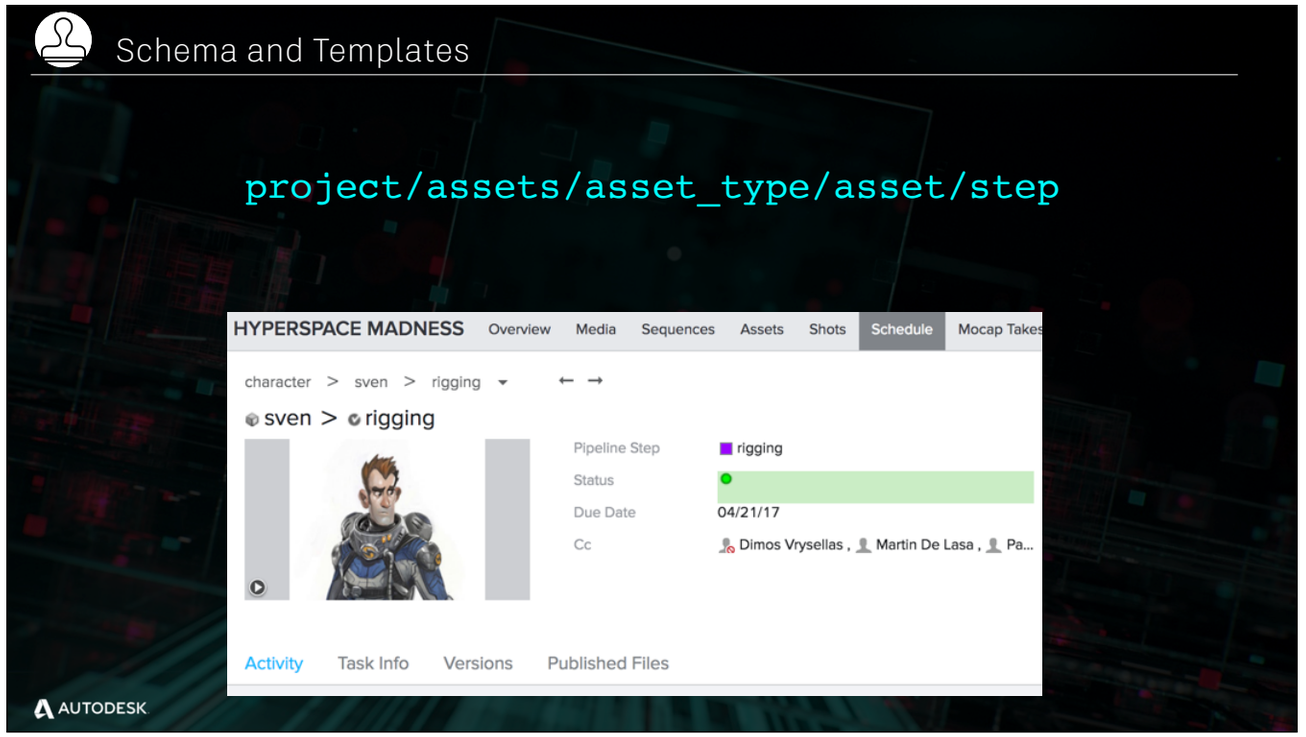
- Inside the schema folder, you'll find a templated mini version of what you want your production filesystem to look like. The folder names are generic - sequence, shot, step, etc., but they are dynamic, and will be replaced by the names of actual entities in your production. yaml files. some are static.
- We're looking at the schema inside our default config, but you can customize this easily for your production: just add/rename folders, change the rules in the yaml files, and you're off to the races
- Let's look at one of these yaml files, namely, asset.yml



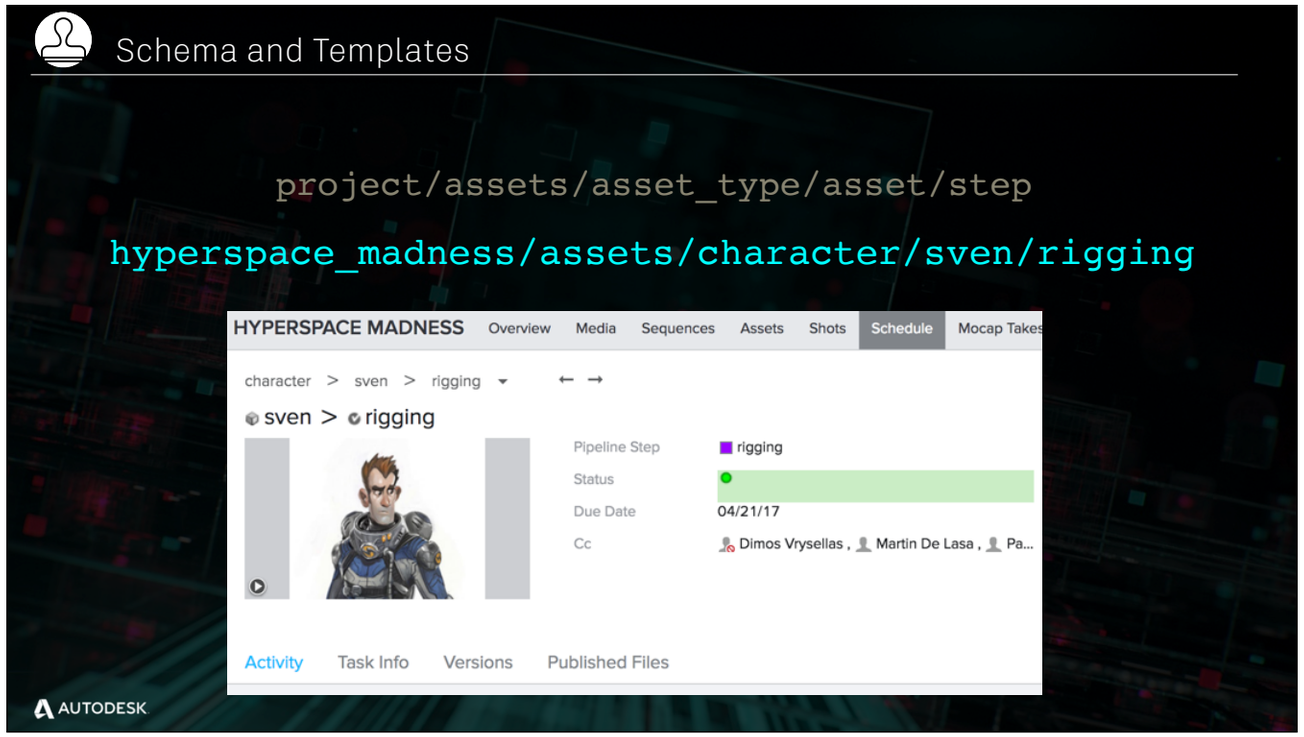
talk about fields, and a little bit about create folders. and say that, these concepts carry through to all of those yaml files in your schema.



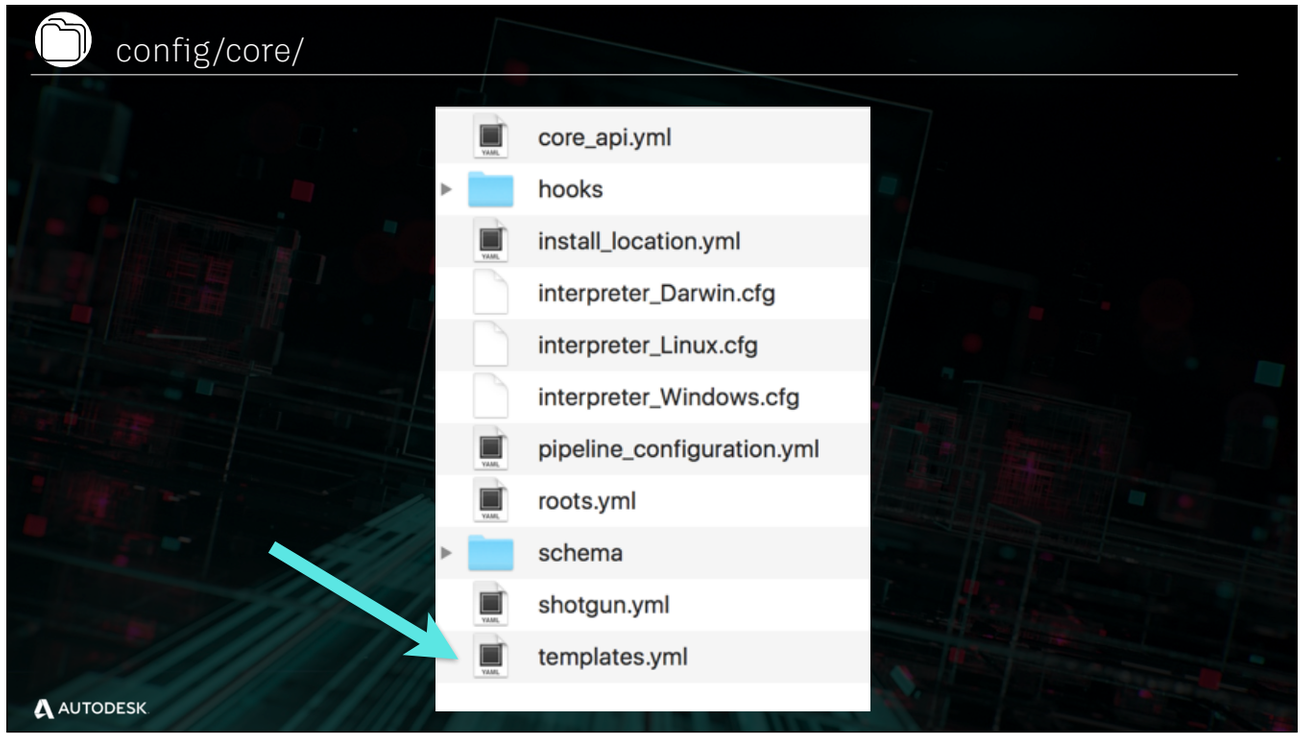
so let’s drive this point home. If you have your schema built out to make a path for pipeline steps on assets that looks like this



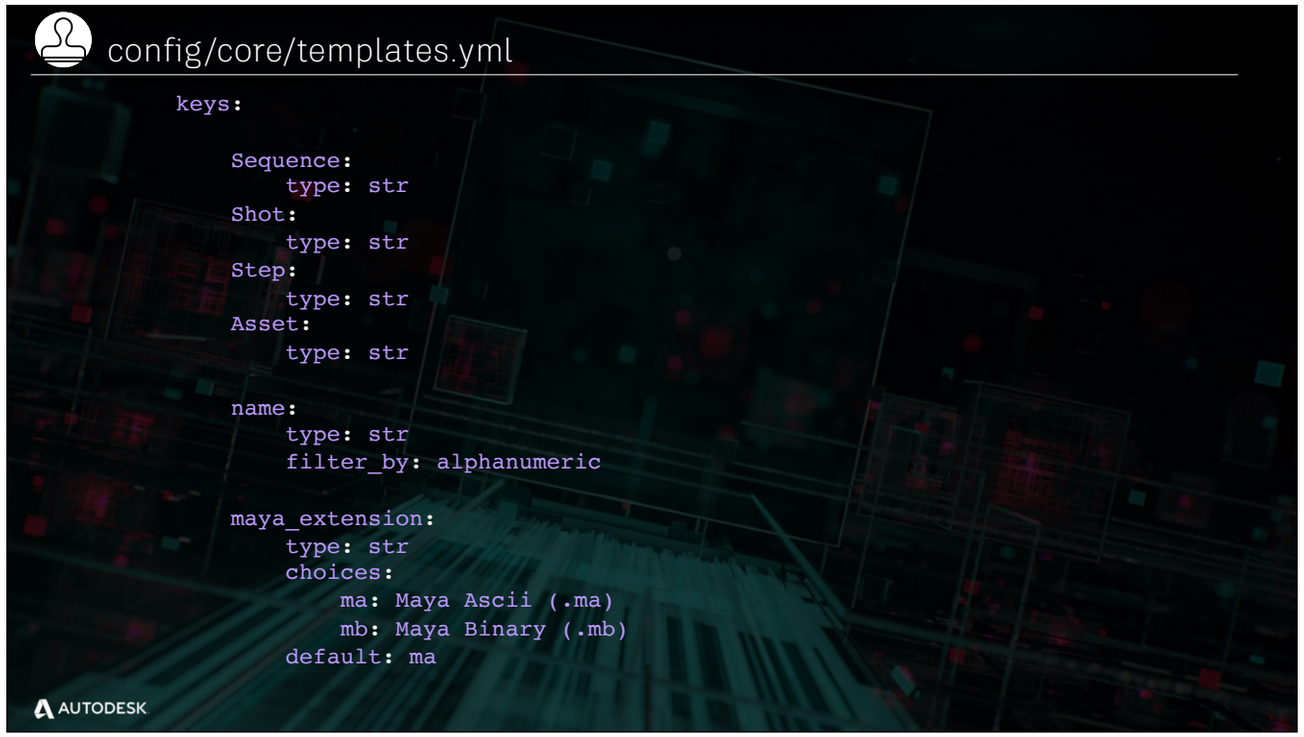
and you create folders for the rigging step on the sven character on the Hyperspace Madness project



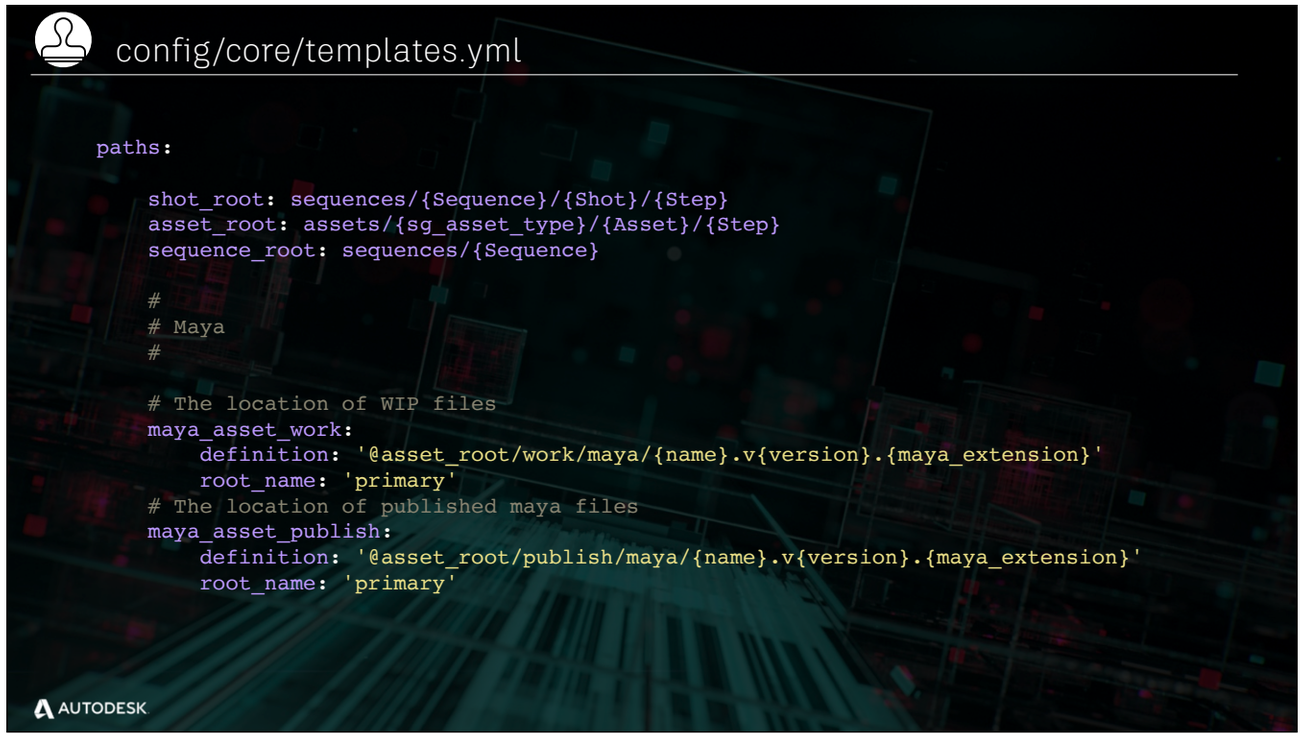
you'll end up with actual folders on your filesystem that look like this



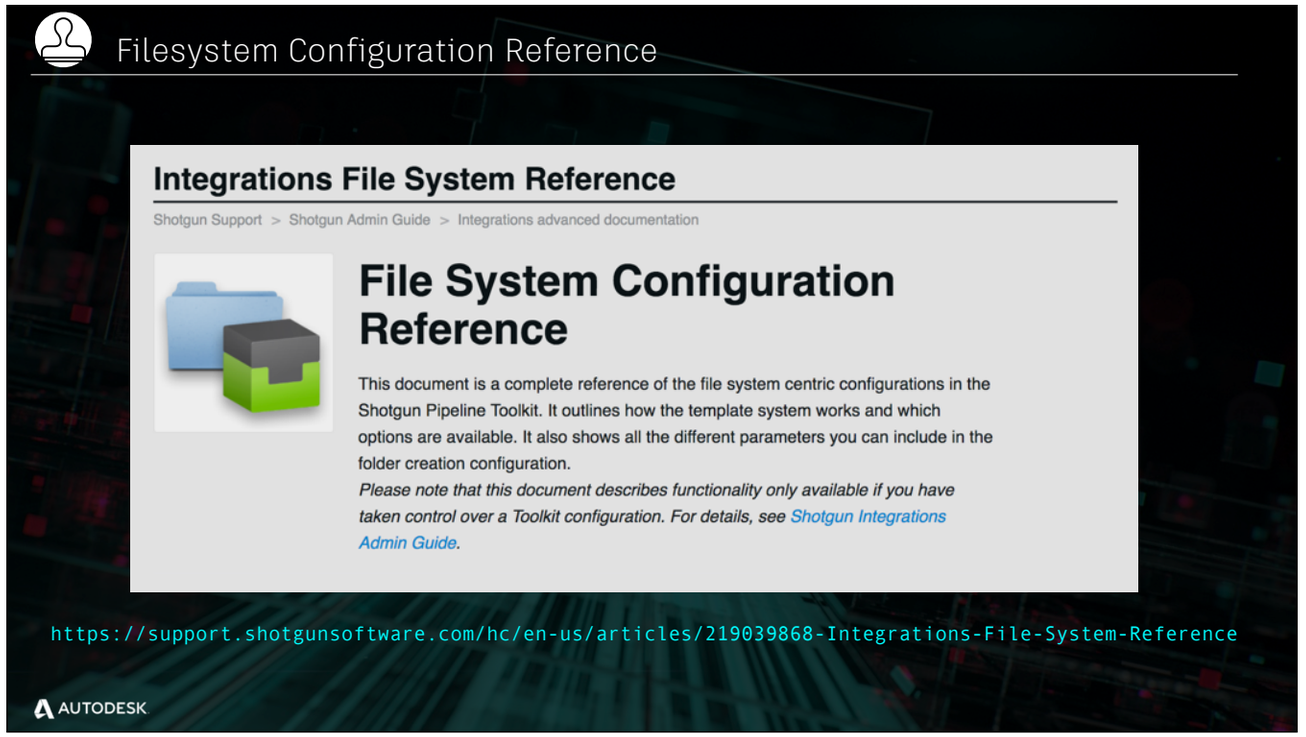
So we've seen how the schema controls the folders in your filesystem, but what about naming of files? That's where templates come in. Similar to what the schema is for folders, the templates.yml file holds templated keys that are used to build paths and file names for use in your Toolkit workflows. Let's take a look at that file.



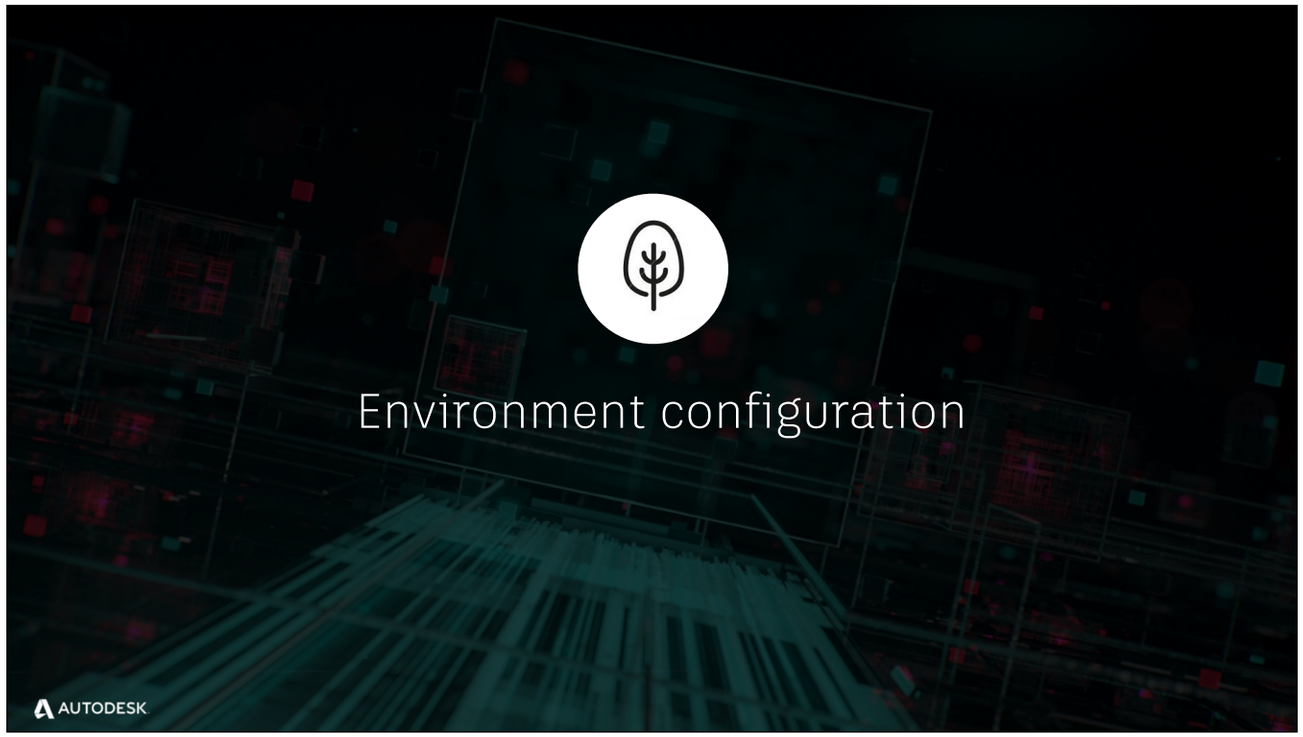
At the top of templates.yml is the keys section. You start by defining a series of template keys - dynamic fields you'll use further down in path templates. type, filter_by, format_spec, alias, choice



And here’s a snippet from the paths section, namely the maya path templates. Common bits at the top. talk through
- maya_shot_work,
- root_name,
- {name}
- revisit {maya_extension}
[NOTE: should we talk about roots.yml and storage roots?]
- Why are we talking so much about folders, I thought this was files?

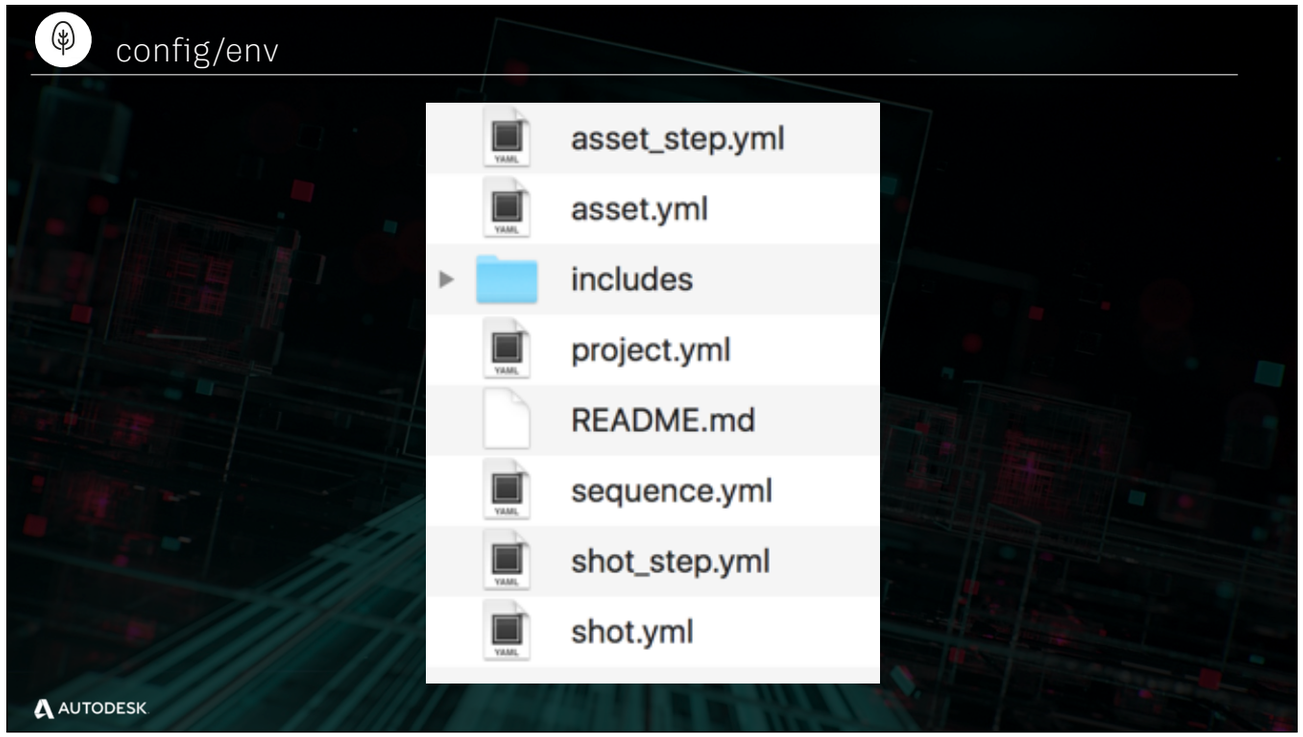


An important note: what we're seeing here is a small subset of the options you have available for customizing your filesystem management. Our Filesystem Configuration Reference doc goes through all of these options; it's a really valuable reference.



So now you've defined some templates, but you're not actually doing anything with them yet. Let's see how templates, and other configuration settings, are put into play

So, going back to our config directory, we talked a bit about core, now let’s talk about env.



So let’s look at one of these files. Since we’re interested in seeing how the maya_asset_work template is put into use, let’s look at the asset_step.yml environment config file.

config/env/asset_step.yml

```
includes:
- ./includes/frameworks.yml
- ./includes/settings/tk-3dsmaxplus.yml
- ./includes/settings/tk-houdini.yml
- ./includes/settings/tk-mari.yml
- ./includes/settings/tk-maya.yml
- ./includes/settings/tk-motionbuilder.yml
- ./includes/settings/tk-nuke.yml
- ./includes/settings/tk-photoshopcc.yml
- ./includes/settings/tk-shell.yml
- ./includes/settings/tk-shotgun.yml

################################################################################
# configuration for all engines to load in an asset step context

engines:
  tk-3dsmaxplus: "@settings.tk-3dsmaxplus.asset_step"
  tk-houdini: "@settings.tk-houdini.asset_step"
  tk-mari: "@settings.tk-mari.asset_step"
  tk-maya: "@settings.tk-maya.asset_step"
  tk-motionbuilder: "@settings.tk-motionbuilder.asset_step"
  tk-nuke: "@settings.tk-nuke.asset_step"
  tk-nukestudio: "@settings.tk-nuke.nukestudio.asset_step"
  tk-photoshopcc: "@settings.tk-photoshopcc.asset_step"
  tk-shell: "@settings.tk-shell.asset_step"
  tk-shotgun: "@settings.tk-shotgun.asset_step"

################################################################################
# reference all of the common frameworks

frameworks: "@frameworks"
```

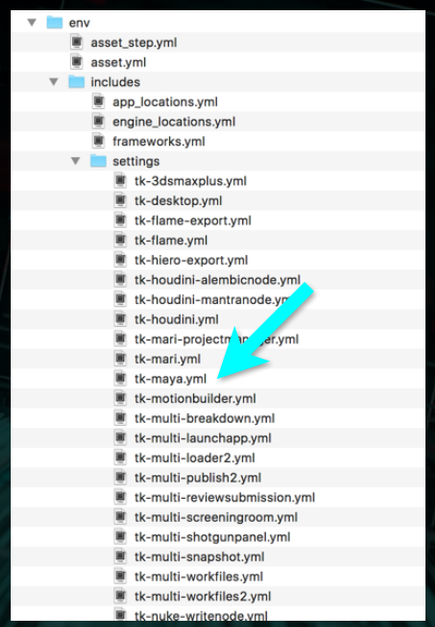
AUTODESK

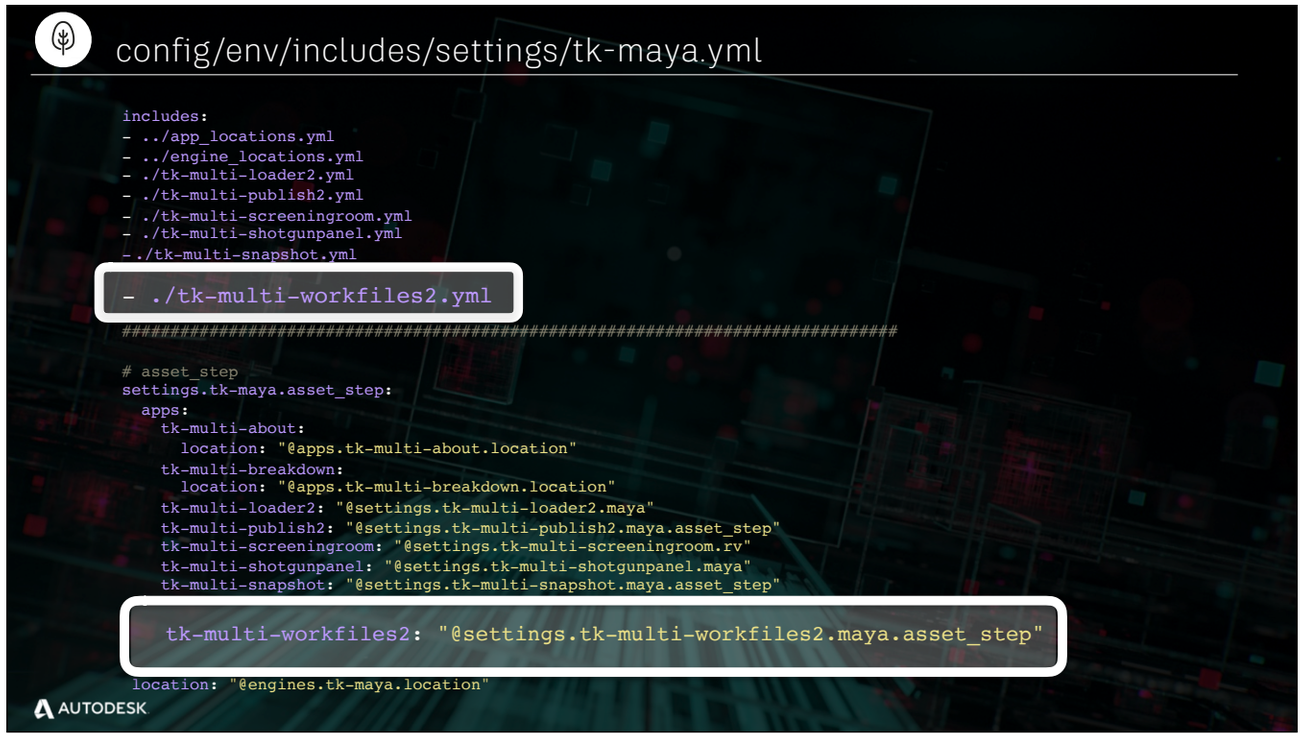
So we open that file. but there is no maya_asset_work here. What we see instead though, is two blocks: at the top are *includes*. Includes are a way for us to define something in a single file, but then *reference* that information in other files. We use includes extensively in the default config, so we can reuse settings, keep files short, and not have to make modifications in multiple files.

Below the includes are the engine blocks: contain settings for all engines available in the env, @ symbol

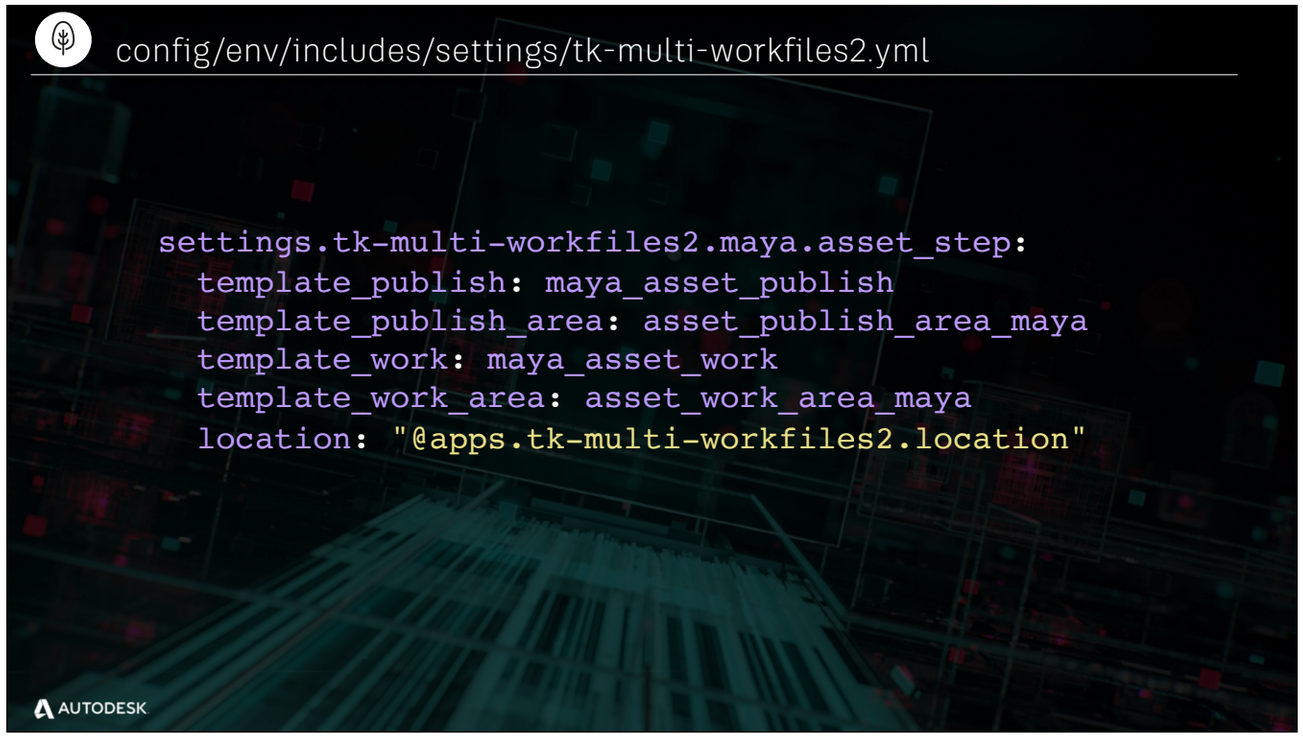
We do find tk-maya in there, and it’s pointing to @settings.tk-maya.asset_step. so let’s go to that included tk-maya.yml file.

Environment configuration





once again, no shot work templates defined here. once again, includes at the top. in this file, they're followed by environment-specific blocks of apps, mostly defined as includes. so keep going down the path of includes, to tk-multi-workfiles2.yml



- at last! some configuration settings! and templates at that!
- So we figured out how to change templates, but this concept carries out to all other settings for our apps, too. We can make changes here to our templates, but Workfiles must have more than 4 configuration settings. Sparsed configs - we only write out the settings that stray from the default. info.yml

the manifest. [CLICK] Every possible configuration setting for the bundle, and a description of each. You can see that just for this one app, there are MANY configuration settings. And this is the power of the Toolkit pipeline configuration: with simple tweaks in yaml files, you can quickly make all kinds of customizations to your pipeline. And that's Pipeline config..

So, going back to our config directory, we talked a bit about core, as well as env, which leaves us with hooks.

The idea with hooks is that it’s easy to change a value for a configuration setting by modifying your yaml files. But sometimes you want a deeper customization: injecting custom code into your workflows. For that we have hooks. In broad strokes, when you customize a hook, it goes in your pipeline configurations “hooks” directory. Taking over hooks is more work than modifying configs, but it gives your pipeline configuration infinite extensibility. In the next half-hour, Josh Tomlinson will demo taking over a Toolkit app hook and customizing it.

Before I hand it over to Josh, are there any questions?


AUTODESK