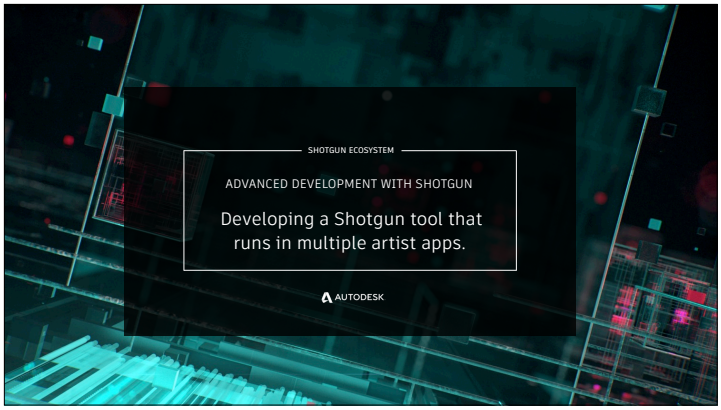
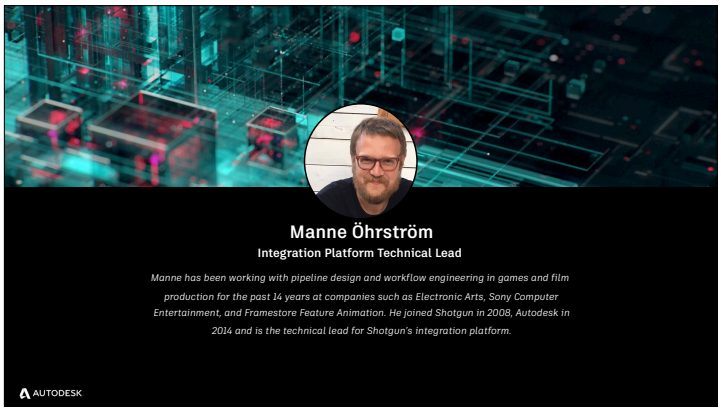
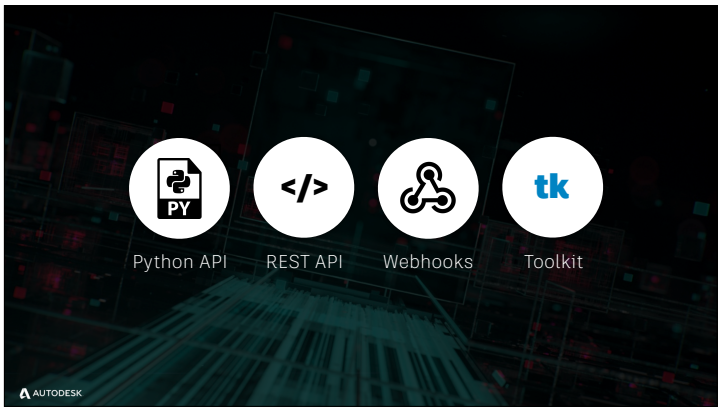




Let me start by introducing myself!
My name is Manne. I have been designing pipelines and tools for vfx, feature animation and games companies for the past 15 years.
I am the tech lead for the Shotgun platform integrations.
It's great to see so many people here today! very exciting and hopefully you'll find this session useful.

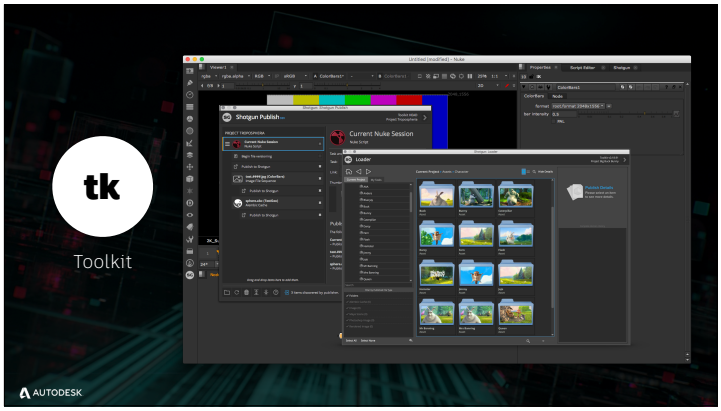
So a quick note before we begin - all the code that I will be showing, as well as the notes of this deck, is in github, so no need to snap pictures of the slides!

So this session is about Shotgun Toolkit. How many of you know what Shotgun Toolkit is?



Shotgun has a whole bunch of different tech and integration points. It can sometimes be confusing

- The python API is a generic way to read and write shotgun data
- The rest API mirrors the python API but is useful if you are using other languages, for example javascript.
- Webhooks and the shotgun event deamon can be used to react to things that happen inside of Shotgun
- Which leaves us with the Shotgun Toolkit.

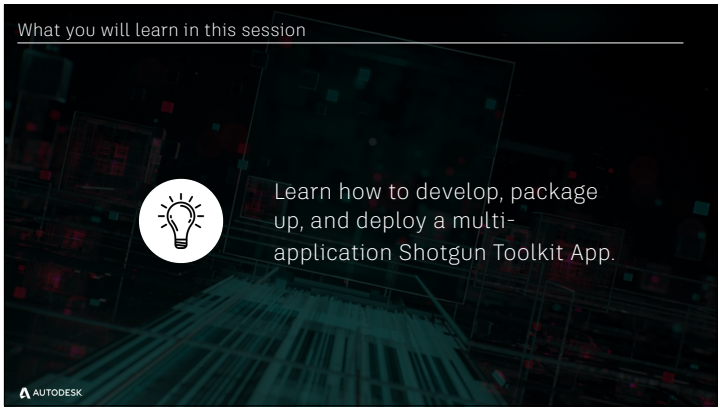


Toolkit is a platform that lets you rapidly develop tools and user experiences that run inside of artist applications like Maya. It's a system that lets you quickly build UIs, panels and other tools. It uses python and QT. It's got a series of API frameworks of reusable functionality to further accelerate development. It helps streamline things like authentication and other common things that most tools need.

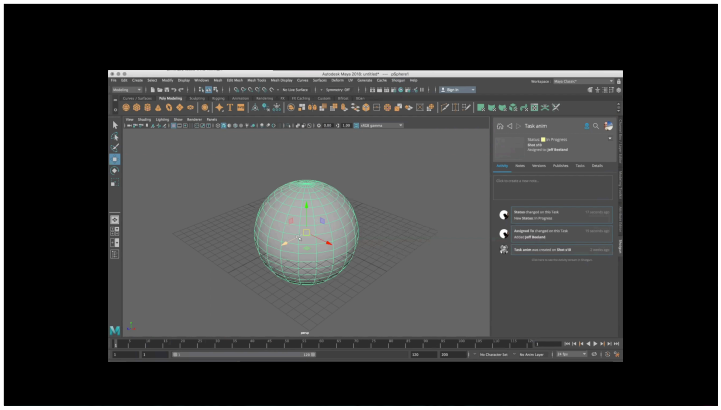
We use toolkit to write all Shotgun in-

application integration tools. If you have used our publisher or loader, then you have used Toolkit.

One of the great things about of Toolkit is that once you have written a tool for maya, it's very easy to get it to run in a wide range of applications.



So for the next 30 minutes we will be learning about this - how to develop a toolkit app and how to deploy it to production.



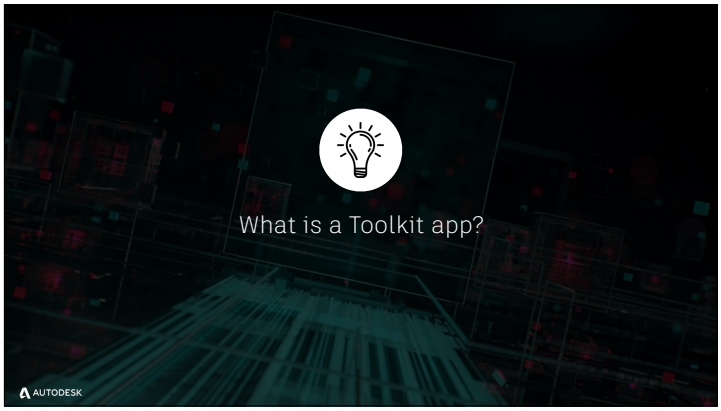
There’s a story to tell, which is that an artist has found something wrong while they’re working:

* They report the issue using the Bug Reporter app
* Mister Support was on the CC list, so they get an email notification
* Mister Support goes to Shotgun and assigns the ticket to himself, and gives it a quick reply

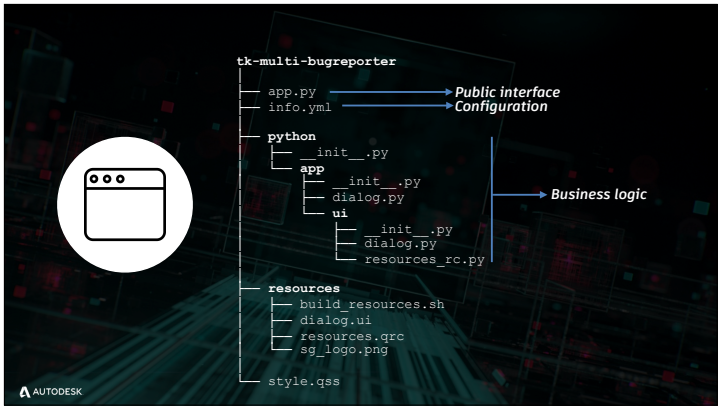
---

So first we’ll look at what an app is exactly
Next, we’ll take a look at toolkit’s reusable frameworks

After that bit of quick theory, we’ll dive in and set up our dev environment
We’ll write the actual app code
And finally we’ll deploy it back to production.



So let’s begin with a very short intro - what is a shotgun toolkit app exactly?



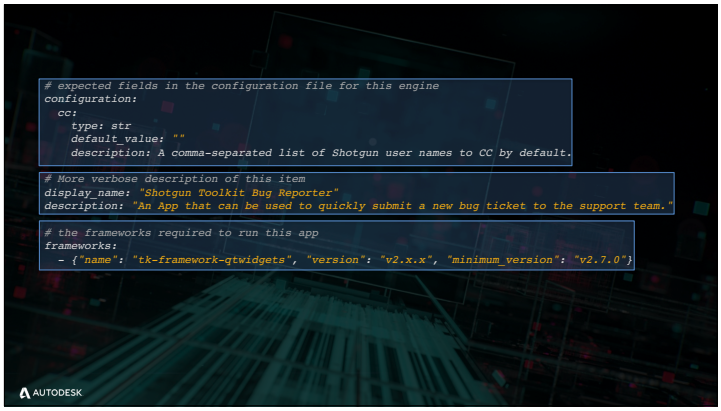
There are three primary components to an App that we're going to look at:

* The app.py contains the public interface (from a code standpoint) and fulfills the contract that any Toolkit App is responsible for
* The info.yml file contains a description of the configuration of the App
* The python module within the App bundle contains the "business logic", where user interface and features are coded

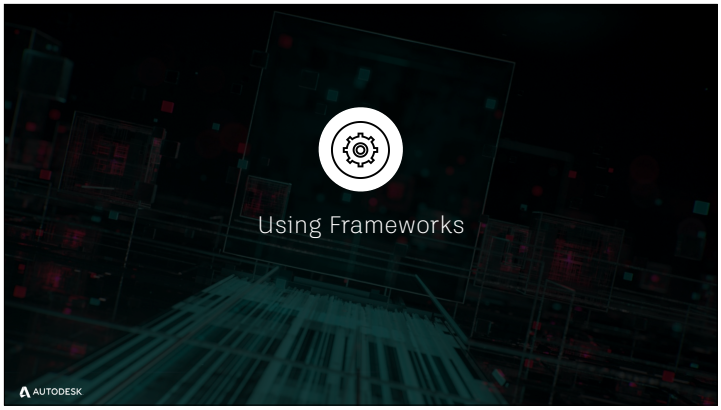
---

The bare minimum:

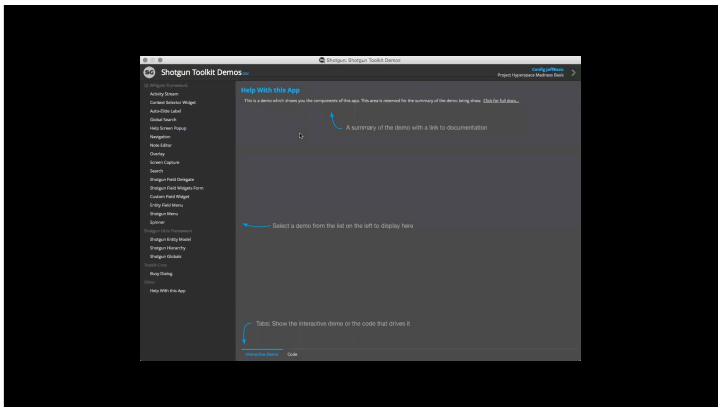
* A class subclassing sgtk.platform.Application
* An init_app method



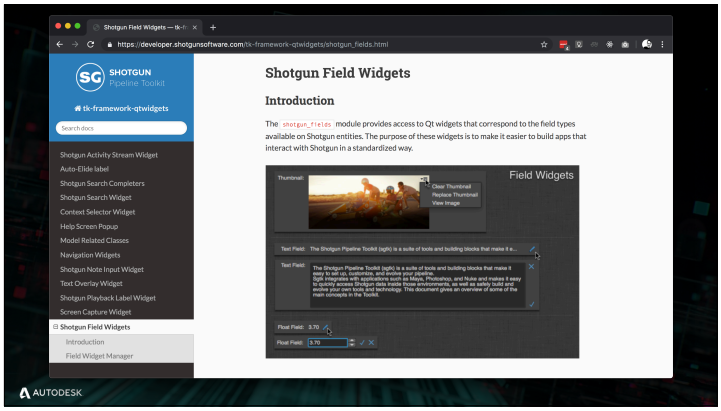
* Configuration: contains the definition of any config settings for the app
* Name/Desc: Pretty name and description for users
* Frameworks: The app's required frameworks (more later)



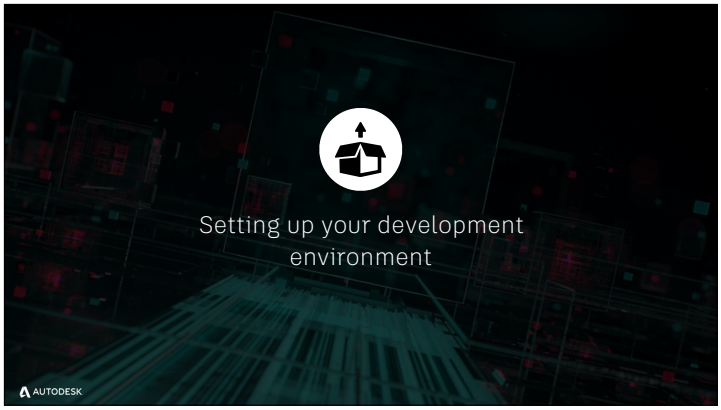
Frameworks play a key part in Shotgun Toolkit as they help reuse functionality and widgets.

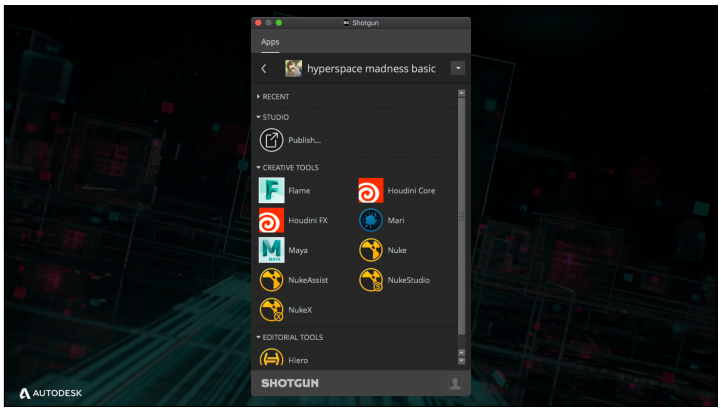


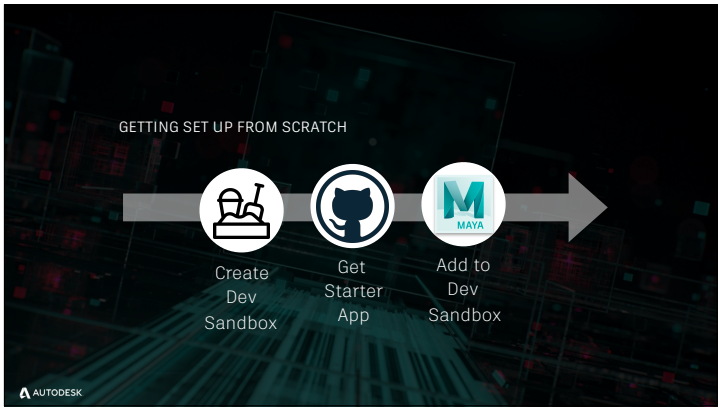
* We provide a BUNCH of stuff out of the box, and the demo app is a good place to start
* We're going to be focusing the qtwidgets framework for this app
* 10x speed with widgets
* We are showing you one use case, here's how you discover all the other things!
* Shotgun widgets.
* This is how all the integrations are built.

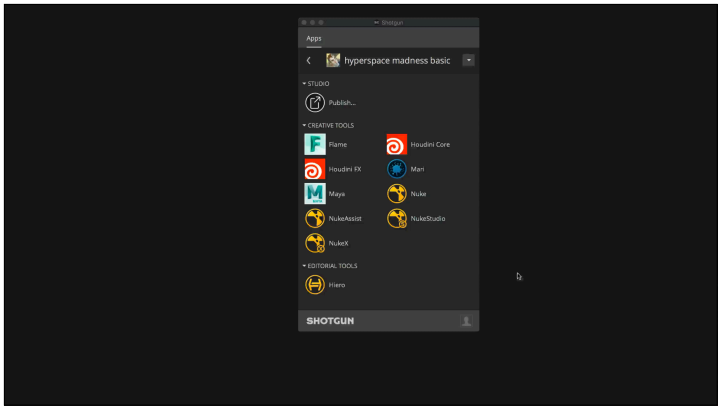


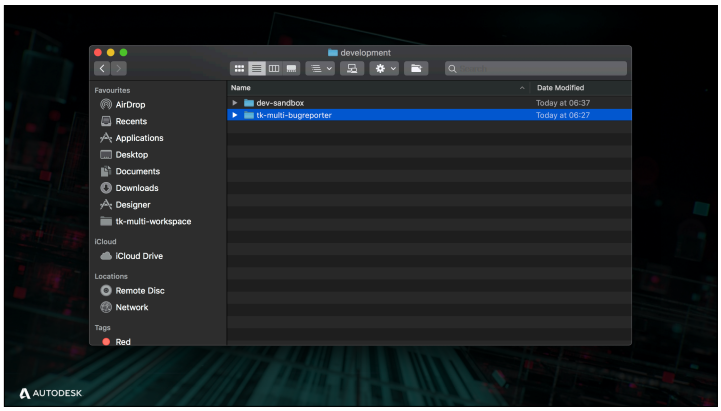
There is also extensive API documentation on our dev site.
And all the source code is available in github.

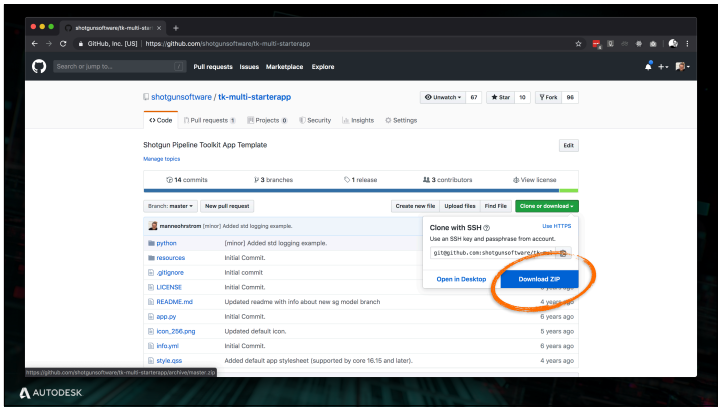

Setting up your development environment
AUTODESK

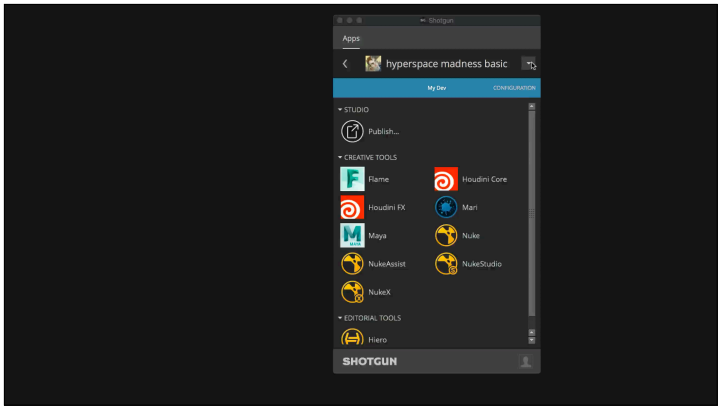

Shotgun
Apps
hyperspace madness basic
RECENT
STUDIO
Publish...
CREATIVE TOOLS
Flame
Houdini Core
Houdini FX
Mari
Maya
Nuke
NukeAssist
NukeStudio
NukeX
EDITORIAL TOOLS
Hiero
SHOTGUN
AUTODESK

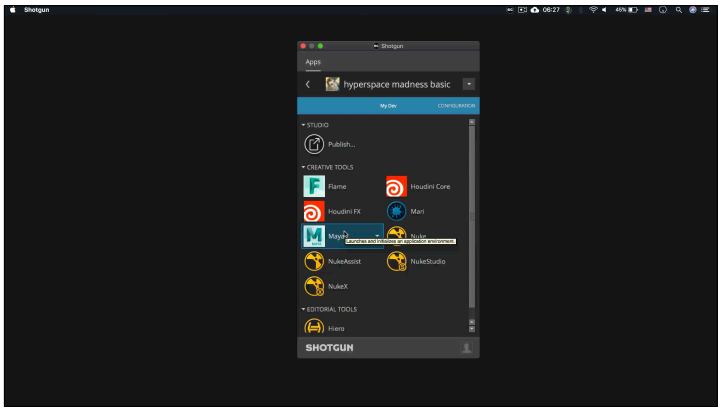

GETTING SET UP FROM SCRATCH
Create Dev Sandbox
Get Starter App
Add to Dev Sandbox
AUTODESK

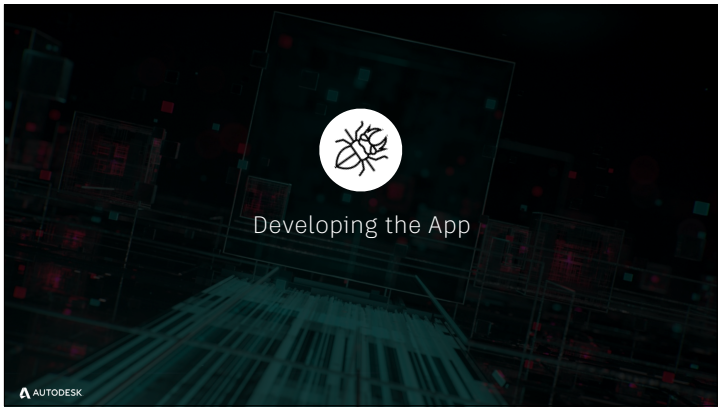

Apps
hyperspace madness basic
STUDIO
Publish...
CREATIVE TOOLS
Flame
Houdini Core
Houdini FX
Mari
Maya
Nuke
NukeAssist
NukeStudio
NukeX
EDITORIAL TOOLS
Hiero
SHOTGUN


development
Favourites
AirDrop
Recents
Applications
Desktop
Documents
Downloads
Designer
tk-multi-workspace
iCloud
iCloud Drive
Locations
Remote Disc
Network
Tags
Red
Name
Date Modified
dev-sandbox
Today at 06:37
tk-multi-bugreporter
Today at 06:27
AUTODESK

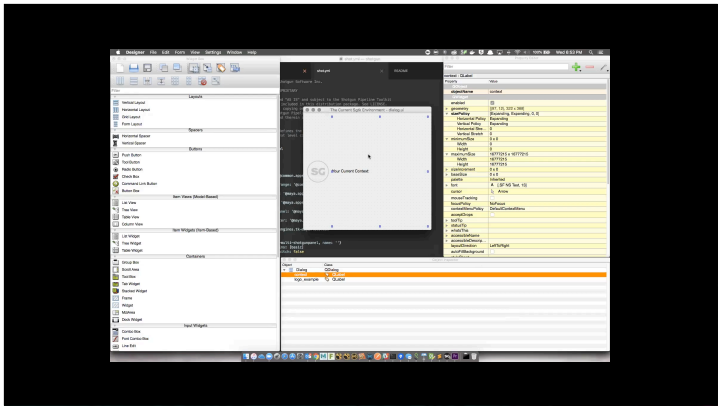

shotgunsoftware / tk-multi-starterapp
Shotgun Pipeline Toolkit App Template
Clone with SSH
Download ZIP
AUTODESK





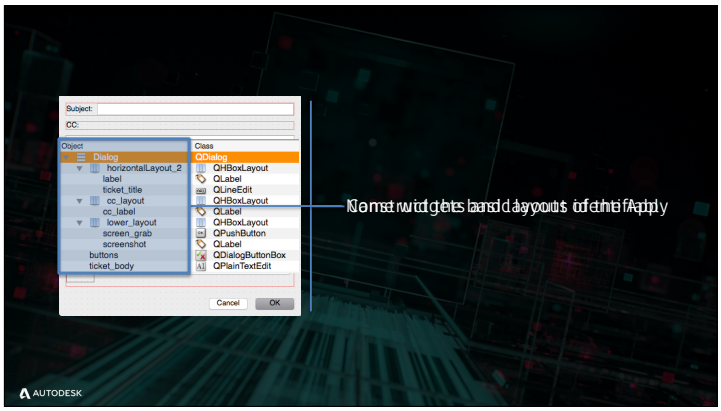


Sweet! So we have the starter app added to our config

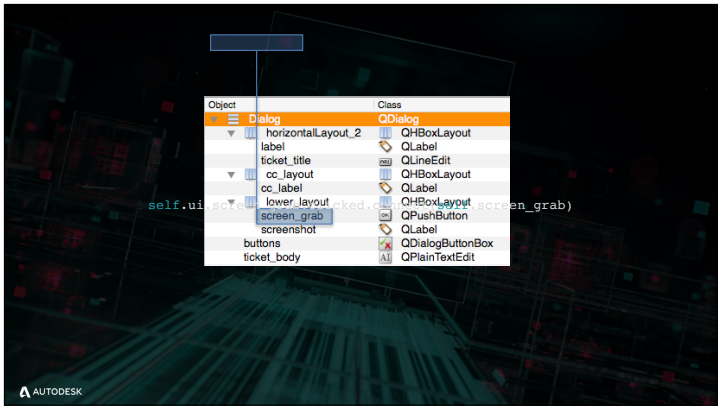


* The starter app gives us a very basic .ui file that can be used in Qt Designer as a starting place
* It provides the basic scaffolding required of a Toolkit App, which can be extended to do what you want your app to do

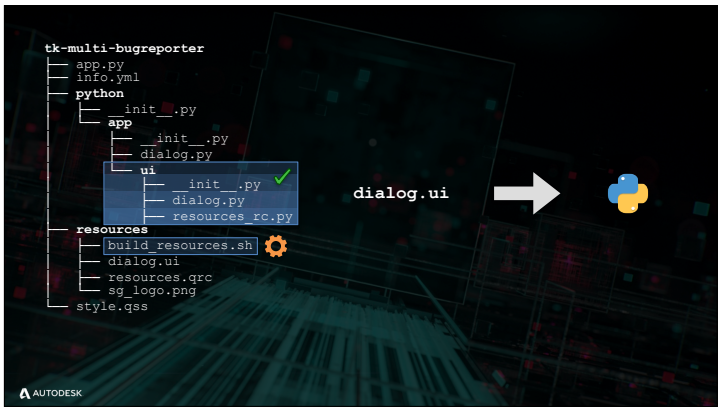
---



* We won't be going into detail about how to use Qt Designer
  * It's not required: if you prefer to code all of your UI layout, then go for it, but we find Designer to be a faster workflow
  * Qt Designer is bundled with Shotgun Desktop, so if you look in the install location for that, you can find what you need without installing any additional software
* Constructing the layout is largely a drag-and-drop affair
* Naming the widgets/layouts you add makes them easy to reference from code later — this is KEY, as we'll discuss further as we continue

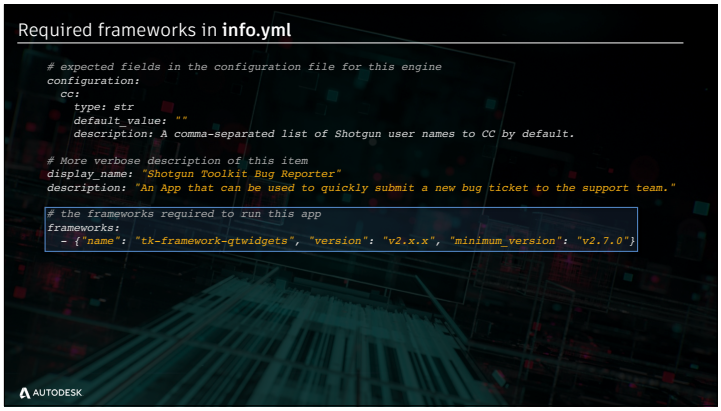


Accessing the widgets created in Qt Designer is done via the ui submodule and by the name specified in Designer.



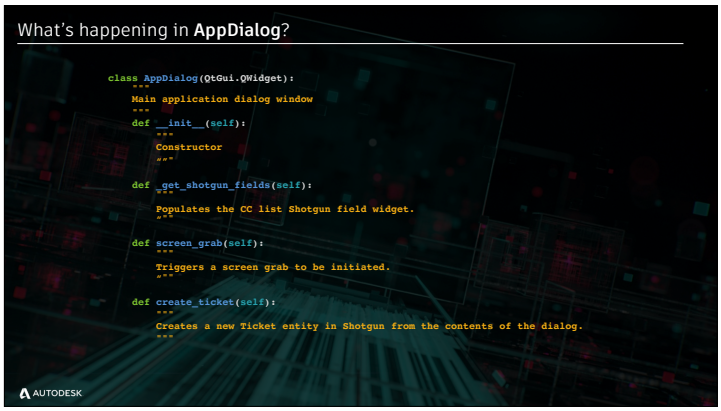
* build_resources.sh converts the work you did in Qt Designer into Python
* That python that is generated goes into the app's ui submodule

---



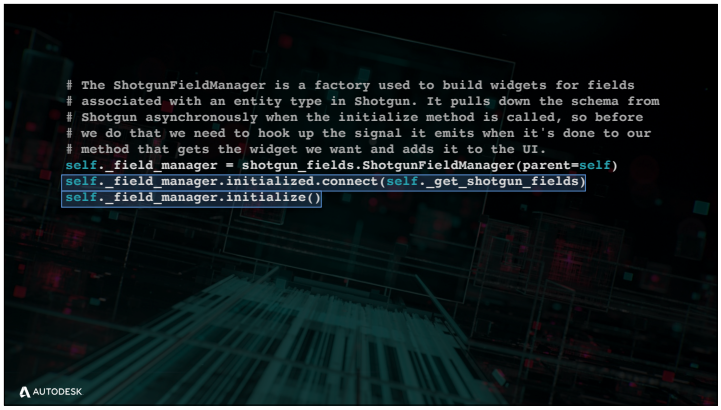
* For the app's business logic to have access to a framework, it must be included in the frameworks structure in info.yml
  * The name of the framework
  * The required version: we allow for wildcards for non-major version components
  * Optionally a bare minimum version requirement

* this is the business logic of the UI
* Import Qt stuff from sgtk.platform.qt because of the shim
* Import the auto-gen dialog from ui for future use (more to come)
* Import frameworks (already explained earlier)
* show_dialog implementation, which makes use of AppDialog (more to come)

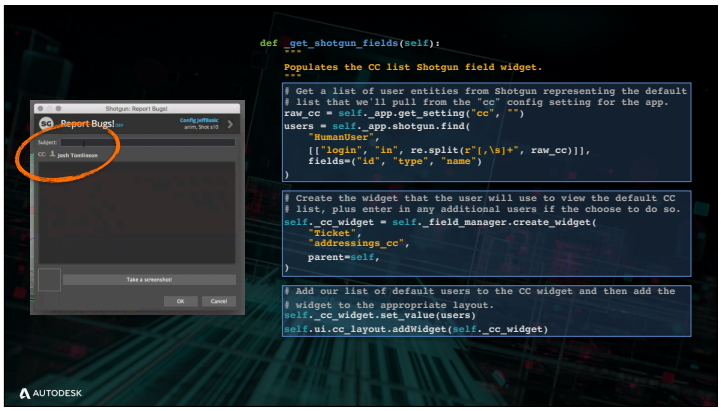


* It's a QWidget!
* four methods, we'll go through each one by one

* we're using the auto-gen dialog from Designer and calling its setupUi method to populate everything
* We can then reference any of the widgets/layouts we created in Designed via self.ui
* The fields manager will require some explanation, so we'll look at it on its own now

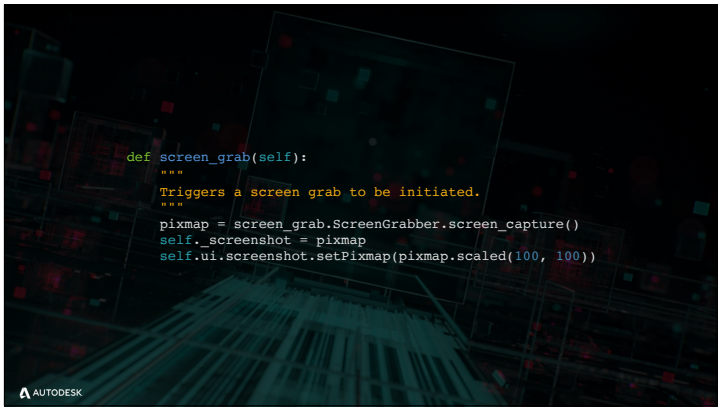


* the initialize call will trigger loading the schema from Shotgun, which can take some time if the data isn't already cached
* When it's done, it emits the initialized signal, which we'll use to trigger some logic (more to come)

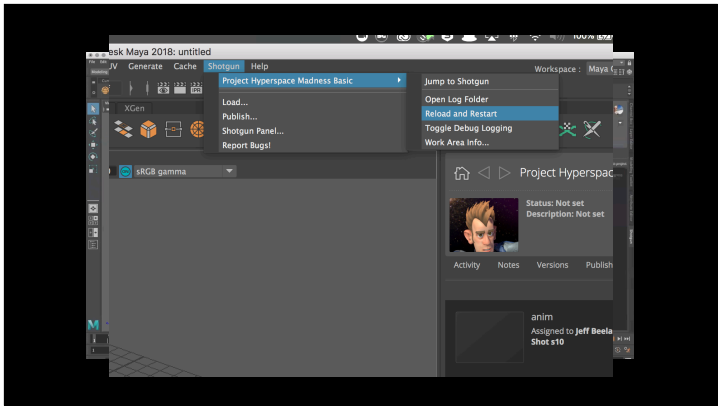


* Accessing our "cc" setting from the config to get the default set of users to CC from SG
* Creating the field widget for the "addressings_cc" field that is on the Ticket entity in SG
* Setting the value of that widget to the list of users we want to CC by default
* Add the cc widget to the layout we previously built in Designer where we wanted it to go

---



* This one is simple: we call the screen_capture static method from ScreenGrabber, which gives us a pixmap object
* We keep track of the pixmap because we'll need it later to create a png we can upload to Shotgun
* We set our screenshot QLabel that we created in Qt Designer as the label's pixmap at the correct resolution



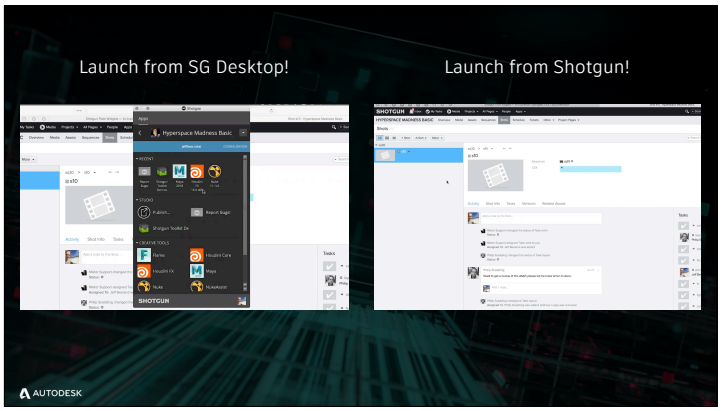
* We'll take a moment to step away from explaining code and look at the development workflow
* An important and super-useful aspect of developing apps and integrations with Toolkit is the reload and restart menu action.
* We try our app without the screen_grab slot hooked up to the button and it doesn't do anything, as expected
* We make the code change, save the file, and use the reload and restart menu action in Maya — now the button triggers a screen grab!
* This highlights how quickly a developer can iterate when writing an app or modifying existing code

---

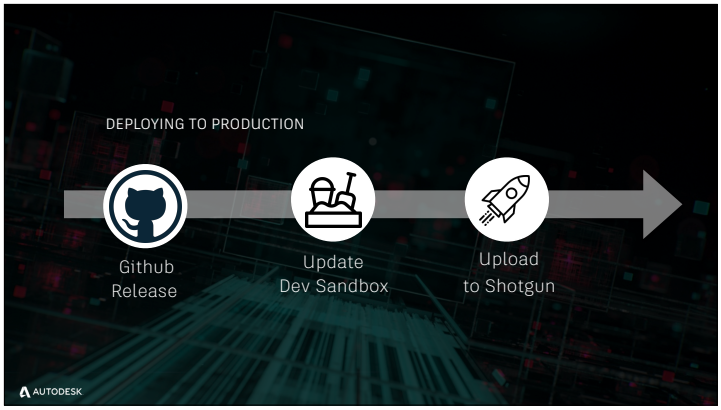
* First: create the new Ticket entity:
  * We get the project entity from the current context
  * Title and description from the widgets we added for those in Qt Designer
  * The cc widget we got from the shotgun fields manager from the qtwidgets framework will give us back a value that's a list of entities that we can use directly, no modification required!
* If we had a screenshot taken, then

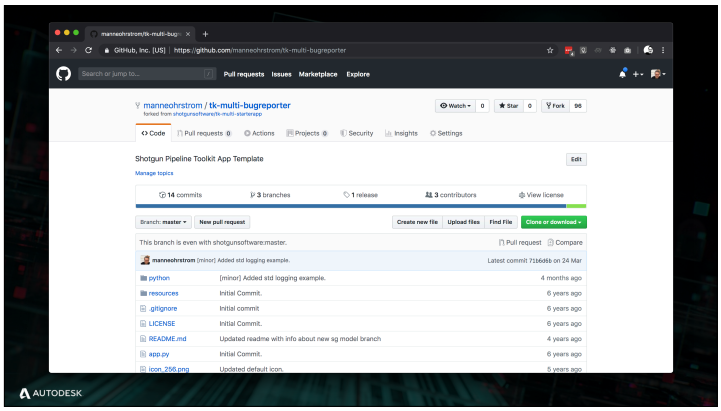
we need to write that out as a png to a temp file and then upload it as an attachment to our Ticket

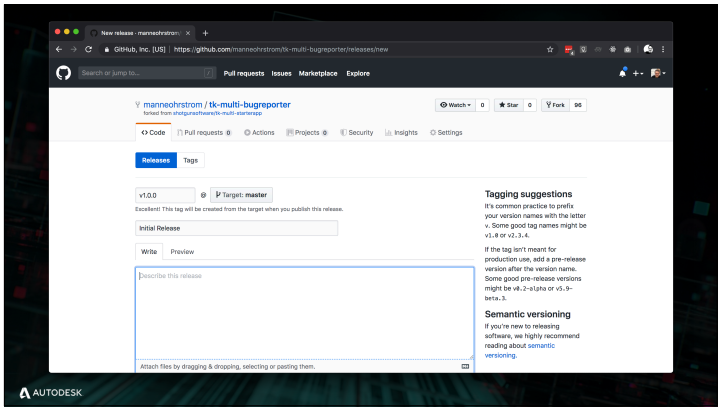
---

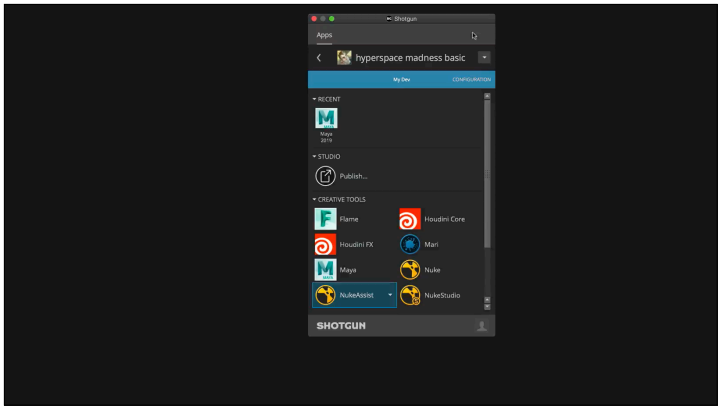


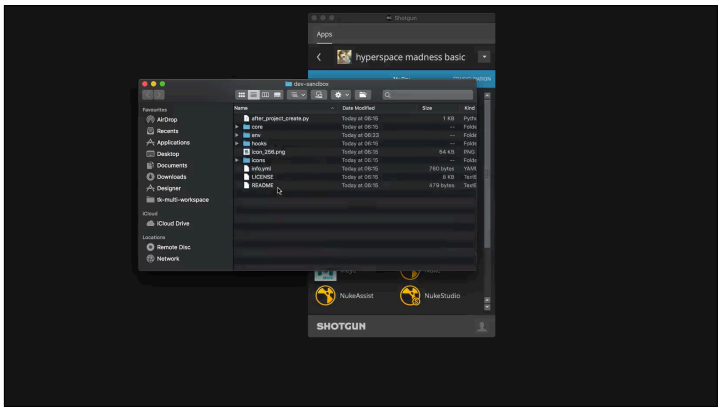
So now all we need to do is to push this back from our dev sandbox into the rest of production

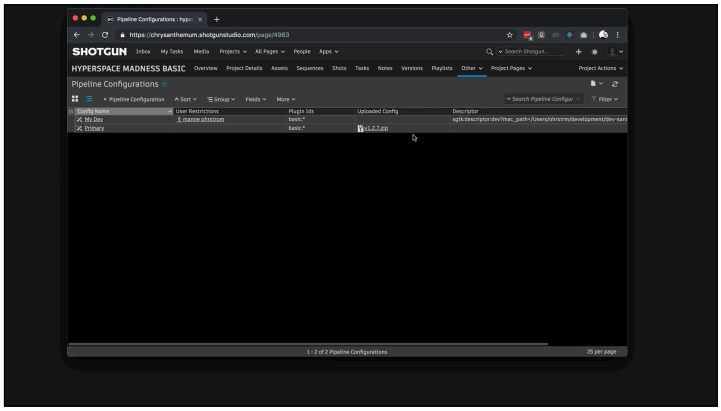

DEPLOYING TO PRODUCTION
Github Release
Update Dev Sandbox
Upload to Shotgun
AUTODESK

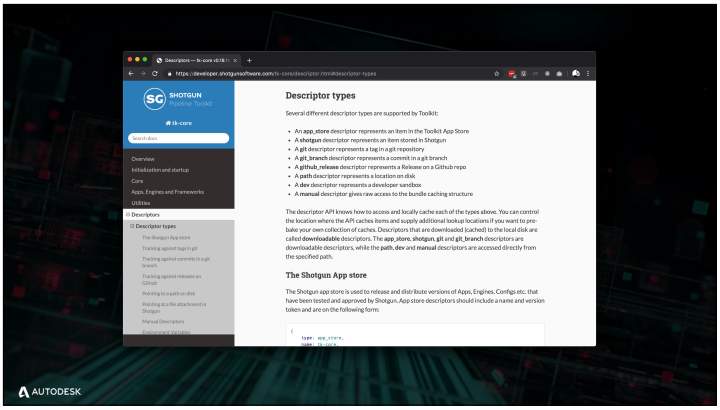

manneohrstrom / tk-multi-bugreporter
Shotgun Pipeline Toolkit App Template
AUTODESK

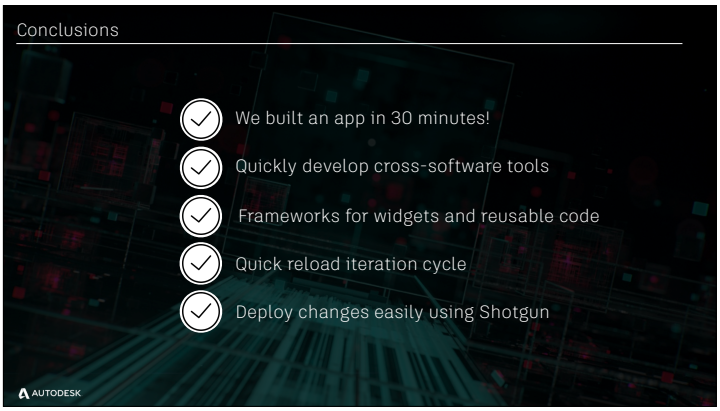

manneohrstrom / tk-multi-bugreporter
Releases
Tags
Tagging suggestions
Semantic versioning
AUTODESK


Shotgun
Apps
hyperspace madness basic
My Dev
CONFIGURATION
RECENT
Maya 2019
STUDIO
Publish...
CREATIVE TOOLS
Flame
Houdini Core
Houdini FX
Mari
Maya
Nuke
NukeAssist
NukeStudio
SHOTGUN

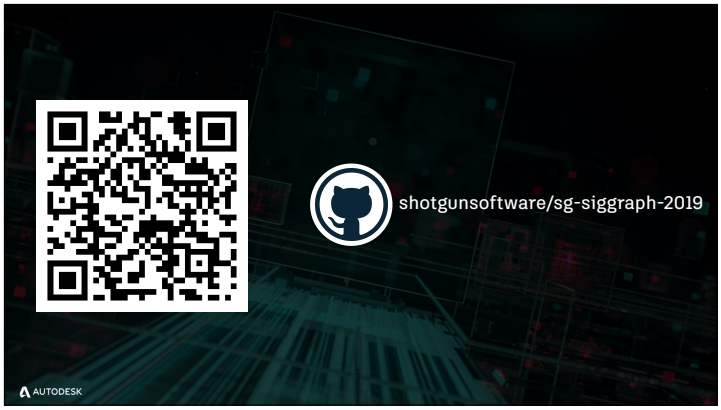

Shotgun
Apps
hyperspace madness basic
dev-sandbox
Favorites
AirDrop
Recents
Applications
Desktop
Documents
Downloads
Designer
9k-multi-workspace
iCloud
iCloud Drive
Locations
Remote Disc
Network
Name
Date Modified
Size
after_project_create.py
core
env
hooks
Today at 06:15
1 KB
icon_256.png
64 KB
icons
info.yml
780 bytes
LICENSE
8 KB
README
479 bytes
NukeAssist
NukeStudio
SHOTGUN


Pipeline Configurations : hype
https://chrysanthemum.shotgunstudio.com/page/4983
SHOTGUN
Inbox
My Tasks
Media
Projects
All Pages
People
Apps
HYPERSPACE MADNESS BASIC
Overview
Project Details
Assets
Sequences
Shots
Tasks
Notes
Versions
Playlists
Other
Project Pages
Project Actions
Pipeline Configurations
Pipeline Configuration
Sort
Group
Fields
More
Search Pipeline Configur
Filter
Config Name
User Restrictions
Plugin Ids
Uploaded Config
Descriptor
My Dev
marine.ohnimm
basic.*
Primary
basic.*
v1.2.2.zip
sgtk:descriptor:dev?mac_path=/Users/ohnimm/development/dev-san
1 - 2 of 2 Pipeline Configurations
25 per page


Descriptors — tk-core v0.18.1
https://developer.shotgunsoftware.com/tk-core/descriptor.html#descriptor-types
SHOTGUN
Pipeline Toolkit
tk-core
search docs
Overview
Initialization and startup
Core
Apps, Engines and Frameworks
Utilities
Descriptors
Descriptor types
The Shotgun App store
Tracking against tags in git
Tracking against commits in a git branch
Tracking against releases on Github
Pointing to a path on disk
Pointing at a file attachment in Shotgun
Manual Descriptors
Environment Variables
Descriptor types
Several different descriptor types are supported by Toolkit:
An app_store descriptor represents an item in the Toolkit App Store
A shotgun descriptor represents an item stored in Shotgun
A git descriptor represents a tag in a git repository
A git_branch descriptor represents a commit in a git branch
A github_release descriptor represents a Release on a Github repo
A path descriptor represents a location on disk
A dev descriptor represents a developer sandbox
A manual descriptor gives raw access to the bundle caching structure
The descriptor API knows how to access and locally cache each of the types above. You can control the location where the API caches items and supply additional lookup locations if you want to pre-bake your own collection of caches. Descriptors that are downloaded (cached) to the local disk are called downloadable descriptors. The app_store, shotgun, git and git_branch descriptors are downloadable descriptors, while the path, dev and manual descriptors are accessed directly from the specified path.
The Shotgun App store
The Shotgun app store is used to release and distribute versions of Apps, Engines, Configs etc, that have been tested and approved by Shotgun. App store descriptors should include a name and version token and are on the following form:
{
type: app_store,
name: tk-core,
AUTODESK


Conclusions
We built an app in 30 minutes!
Quickly develop cross-software tools
Frameworks for widgets and reusable code
Quick reload iteration cycle
Deploy changes easily using Shotgun
AUTODESK


https://app.sli.do/event/10wljptx
AUTODESK



Lastly, here's a QR code to the GitHub repository.