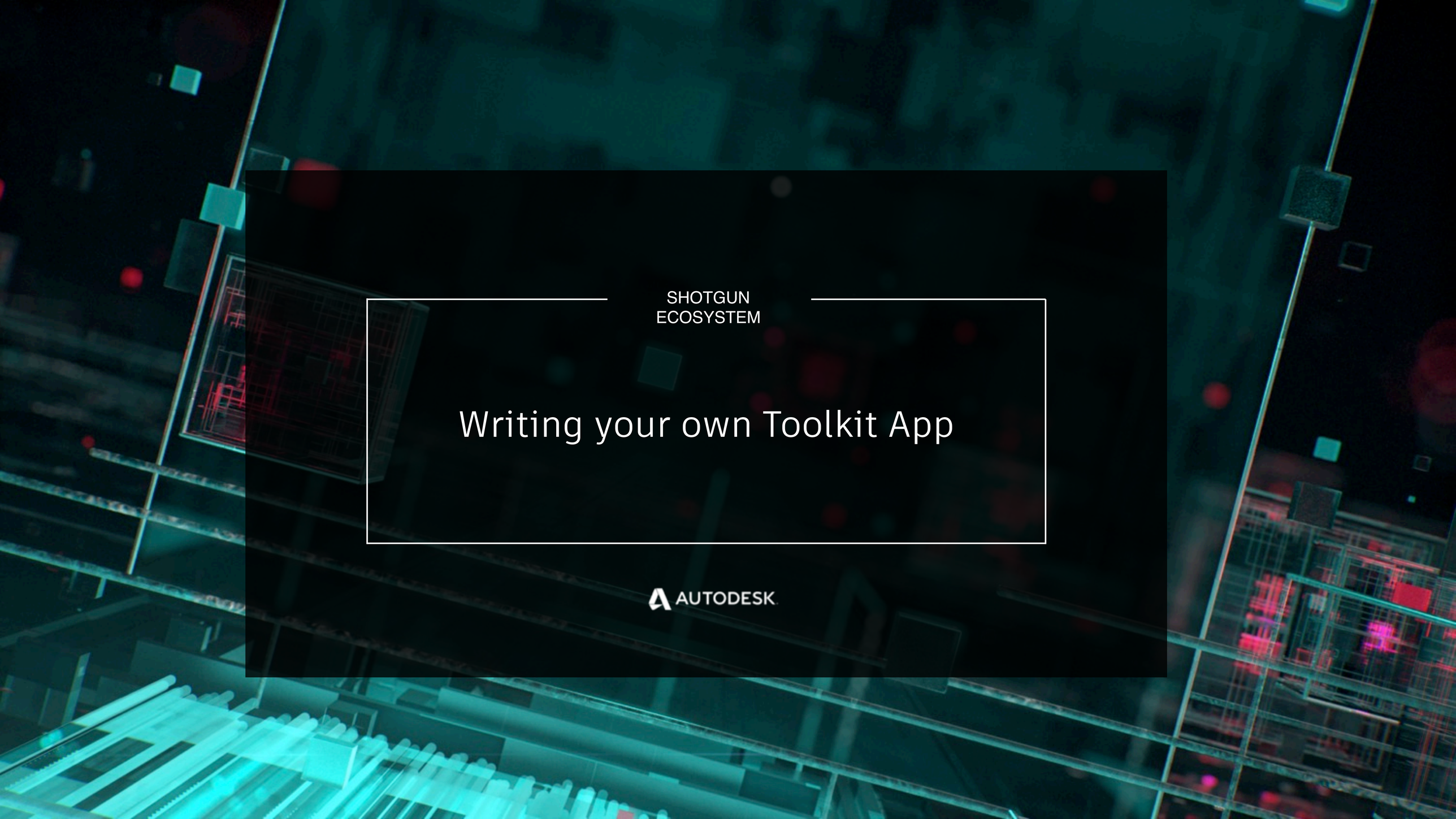
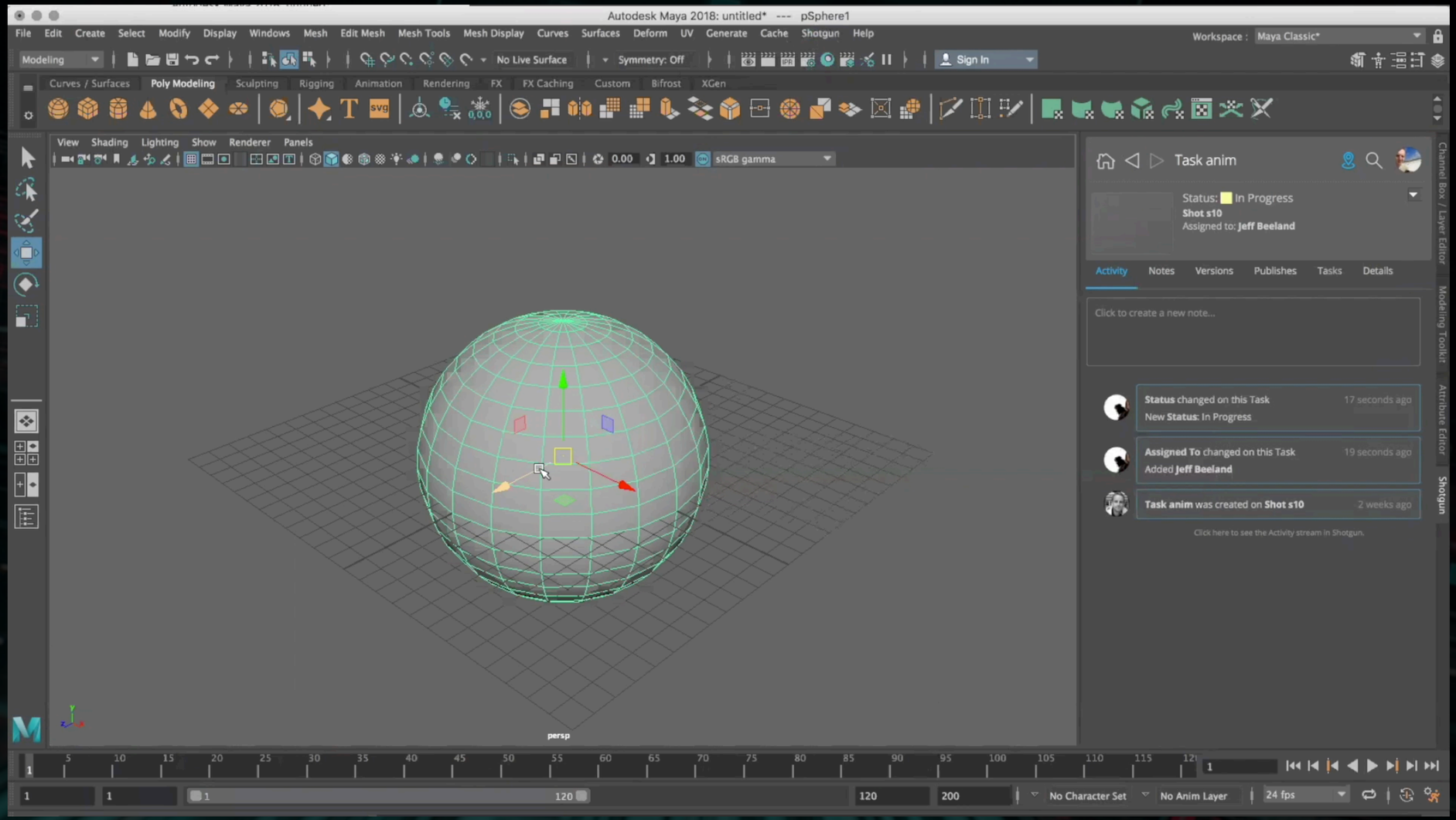

SHOTGUN
ECOSYSTEM
Writing your own Toolkit App
AUTODESK.

Jeff Beeland

Software Development Manager

Jeff spent 11 years developing and supporting VFX and Animation pipelines at Rhythm and Hues and Blur Studio. For the past 3 years, he has worked as an engineer working on Shotgun as part of the Toolkit and Ecosystem teams.

Introducing **tk-multi-bugreporter**!



The structure of a Toolkit App

Using Frameworks

Developing and configuring the App


https://github.com/shotgunsoftware/sg-devday-2018

The structure of a Toolkit App

app.py → *Public interface*

info.yml → *Configuration*

Python module → *Business logic*

Contents of an **app.py**

```python
from sgtk.platform import Application

class BugReporter(Application):
    """
    An App that can be used to submit a bug ticket to support team in Shotgun.
    """
    def init_app(self):
        """
        Called as the application is being initialized
        """
        # first, we use the special import_module command to access the app module
        # that resides inside the python folder in the app. This is where the actual UI
        # and business logic of the app is kept. By using the import_module command,
        # toolkit's code reload mechanism will work properly.
        app_payload = self.import_module("app")

        # now register a *command*, which is normally a menu entry of some kind on a Shotgun
        # menu (but it depends on the engine). The engine will manage this command and
        # whenever the user requests the command, it will call out to the callback.

        # first, set up our callback, calling out to a method inside the app module contained
        # in the python folder of the app
        menu_callback = lambda : app_payload.dialog.show_dialog(self)

        # now register the command with the engine
        self.engine.register_command("Report Bugs!", menu_callback)
```

Contents of an **app.py**

```
from sgtk.platform import Application

class BugReporter(Application):
    """
    An App that can be used to submit a bug ticket to support team in Shotgun.
    """
    def init_app(self):
        """
        Called as the application is being initialized
        """
        # first, we use the special import_module command to access the app module
        # that resides inside the python folder in the app. This is where the actual UI
        # and business logic of the app is kept. By using the import_module command,
        # toolkit's code reload mechanism will work properly.
        app_payload = self.import_module("app")

        # now register a *command*, which is normally a menu entry of some kind on a Shotgun
        # menu (but it depends on the engine). The engine will manage this command and
        # whenever the user requests the command, it will call out to the callback.

        # first, set up our callback, calling out to a method inside the app module contained
        # in the python folder of the app
        menu_callback = lambda : app_payload.dialog.show_dialog(self)

        # now register the command with the engine
        self.engine.register_command("Report Bugs!", menu_callback)
```

Contents of an **info.yml**

```yaml
# expected fields in the configuration file for this engine
configuration:
  cc:
    type: str
    default_value: ""
    description: A comma-separated list of Shotgun user names to CC by default.

# More verbose description of this item
display_name: "Shotgun Toolkit Bug Reporter"
description: "An App that can be used to quickly submit a new bug ticket to the support team."

# the frameworks required to run this app
frameworks:
  - {"name": "tk-framework-qtwidgets", "version": "v2.x.x", "minimum_version": "v2.7.0"}
```

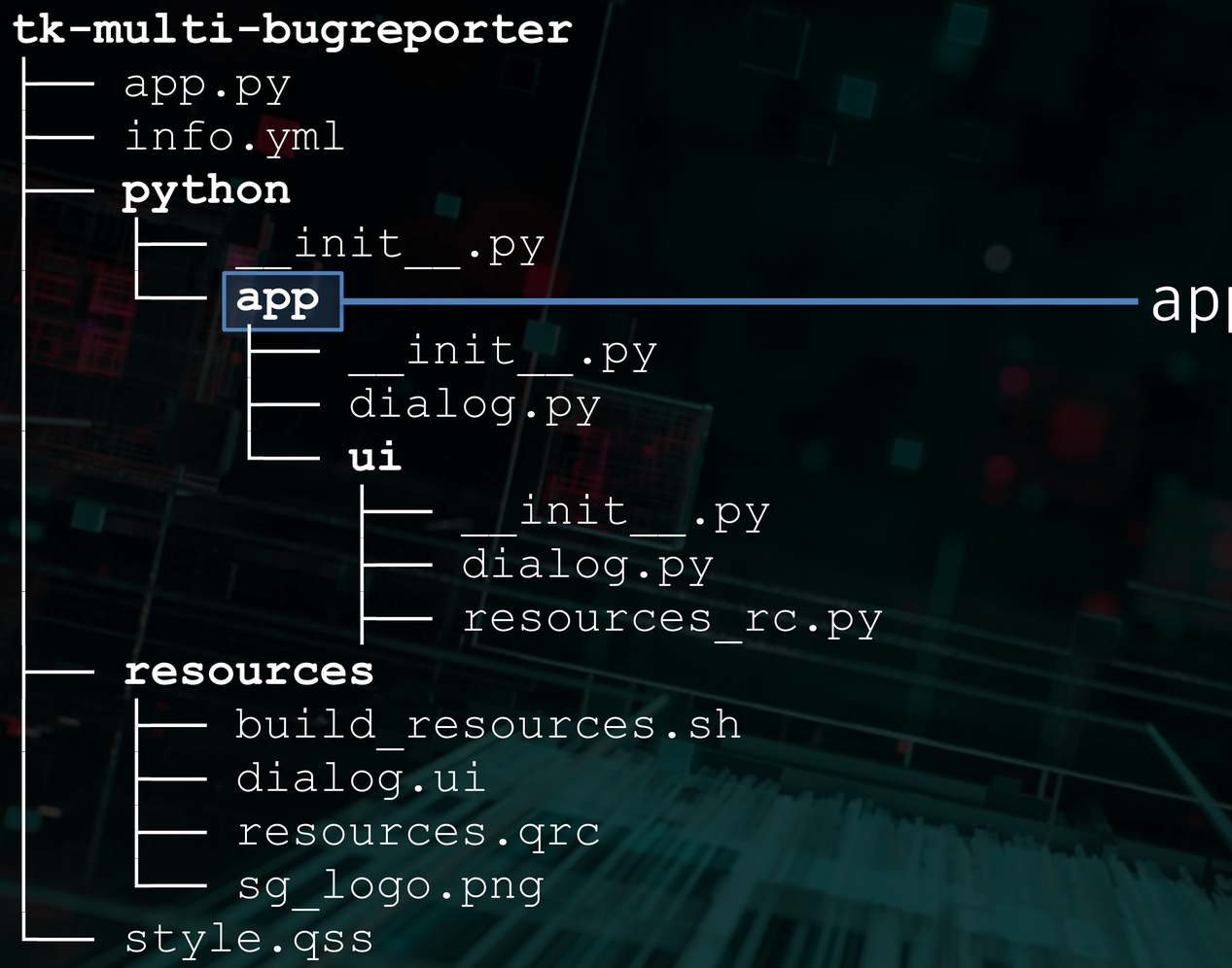
Contents of an **info.yml**

```yaml
# expected fields in the configuration file for this engine
configuration:
  cc:
    type: str
    default_value: ""
    description: A comma-separated list of Shotgun user names to CC by default.

# More verbose description of this item
display_name: "Shotgun Toolkit Bug Reporter"
description: "An App that can be used to quickly submit a new bug ticket to the support team."

# the frameworks required to run this app
frameworks:
  - {"name": "tk-framework-qtwidgets", "version": "v2.x.x", "minimum_version": "v2.7.0"}
```

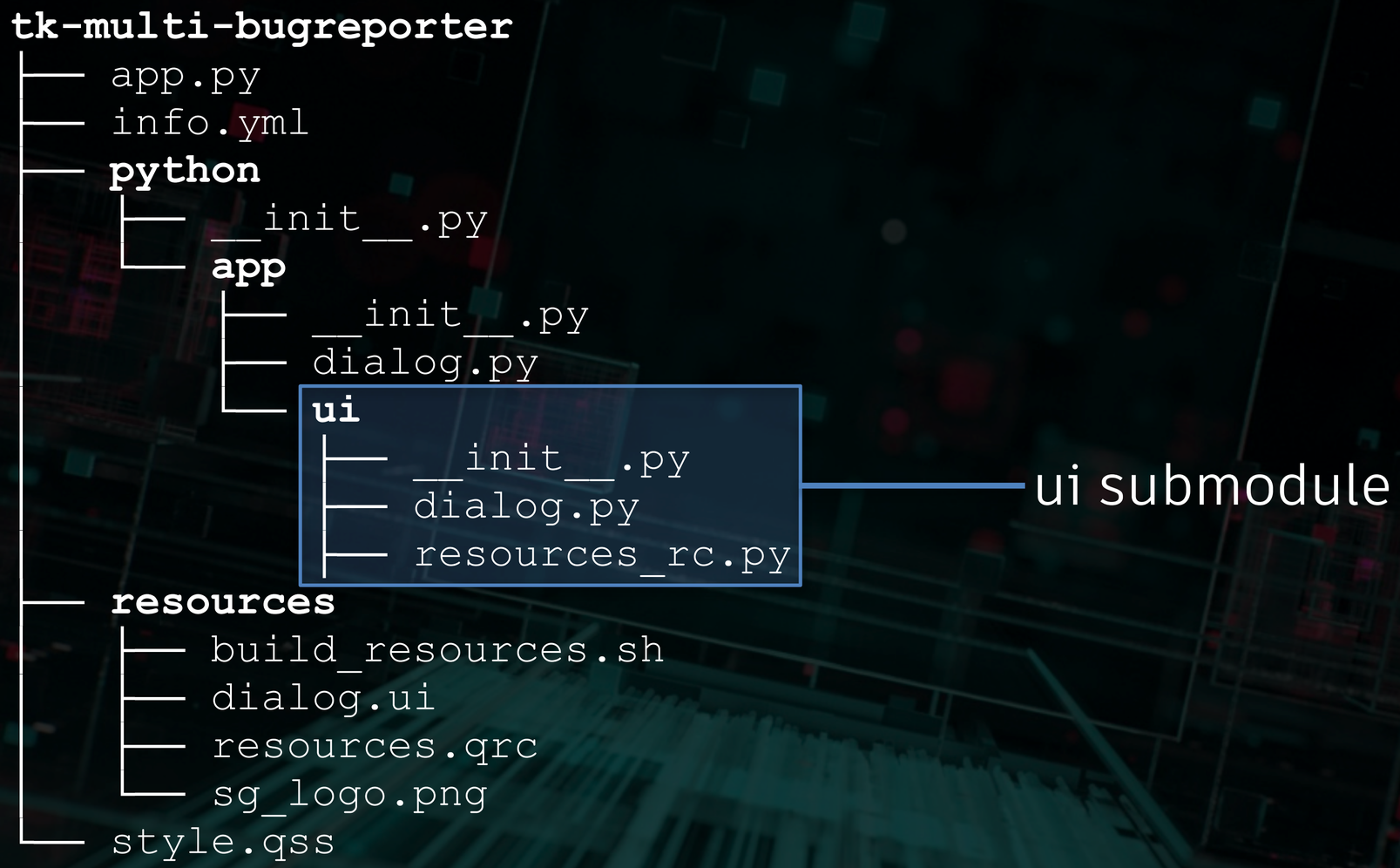
Contents of an **info.yml**

```yaml
# expected fields in the configuration file for this engine
configuration:
  cc:
    type: str
    default_value: ""
    description: A comma-separated list of Shotgun user names to CC by default.

# More verbose description of this item
display_name: "Shotgun Toolkit Bug Reporter"
description: "An App that can be used to quickly submit a new bug ticket to the support team."

# the frameworks required to run this app
frameworks:
  - {"name": "tk-framework-qtwidgets", "version": "v2.x.x", "minimum_version": "v2.7.0"}
```

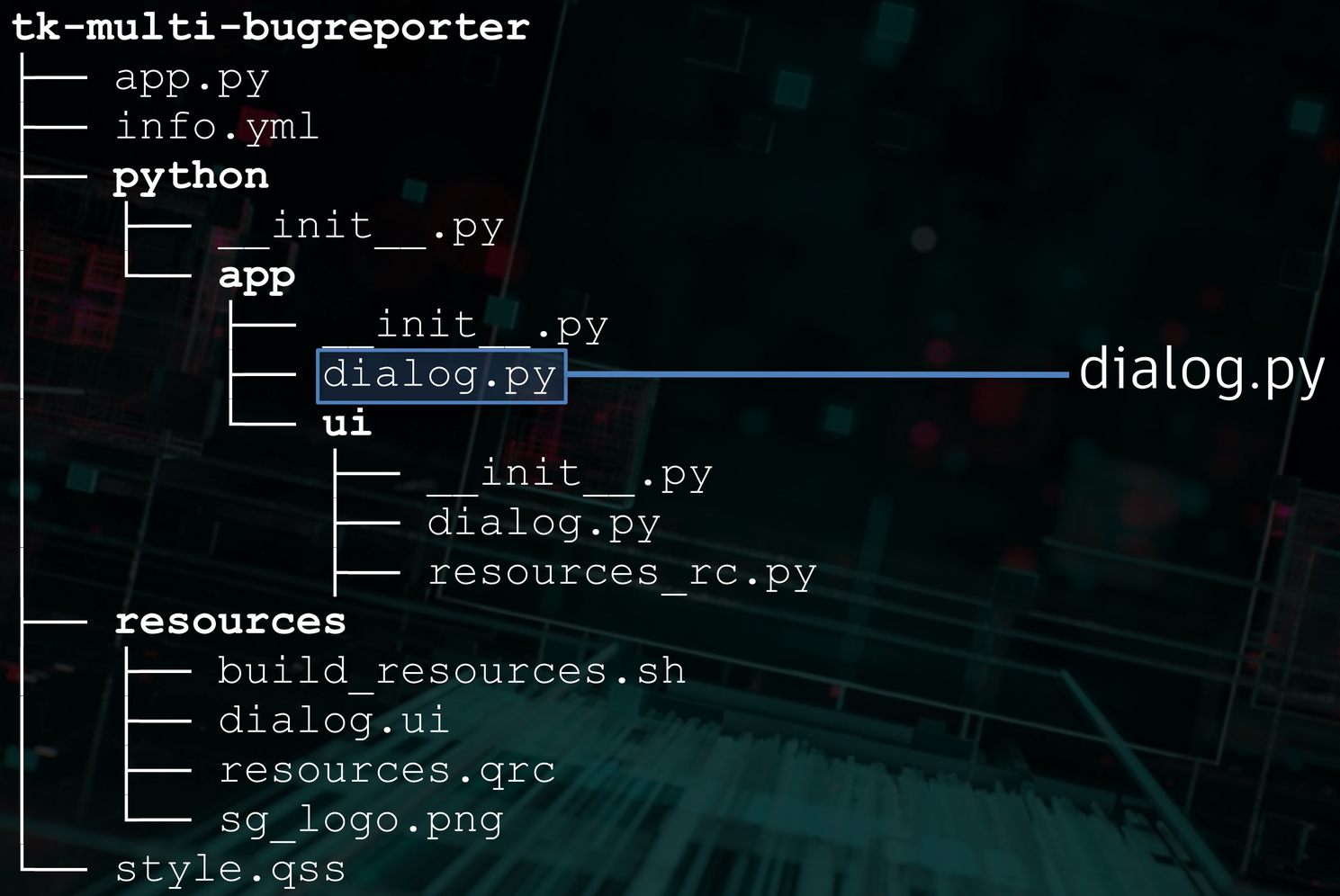
Contents of the **Python module**



Contents of the **Python module**

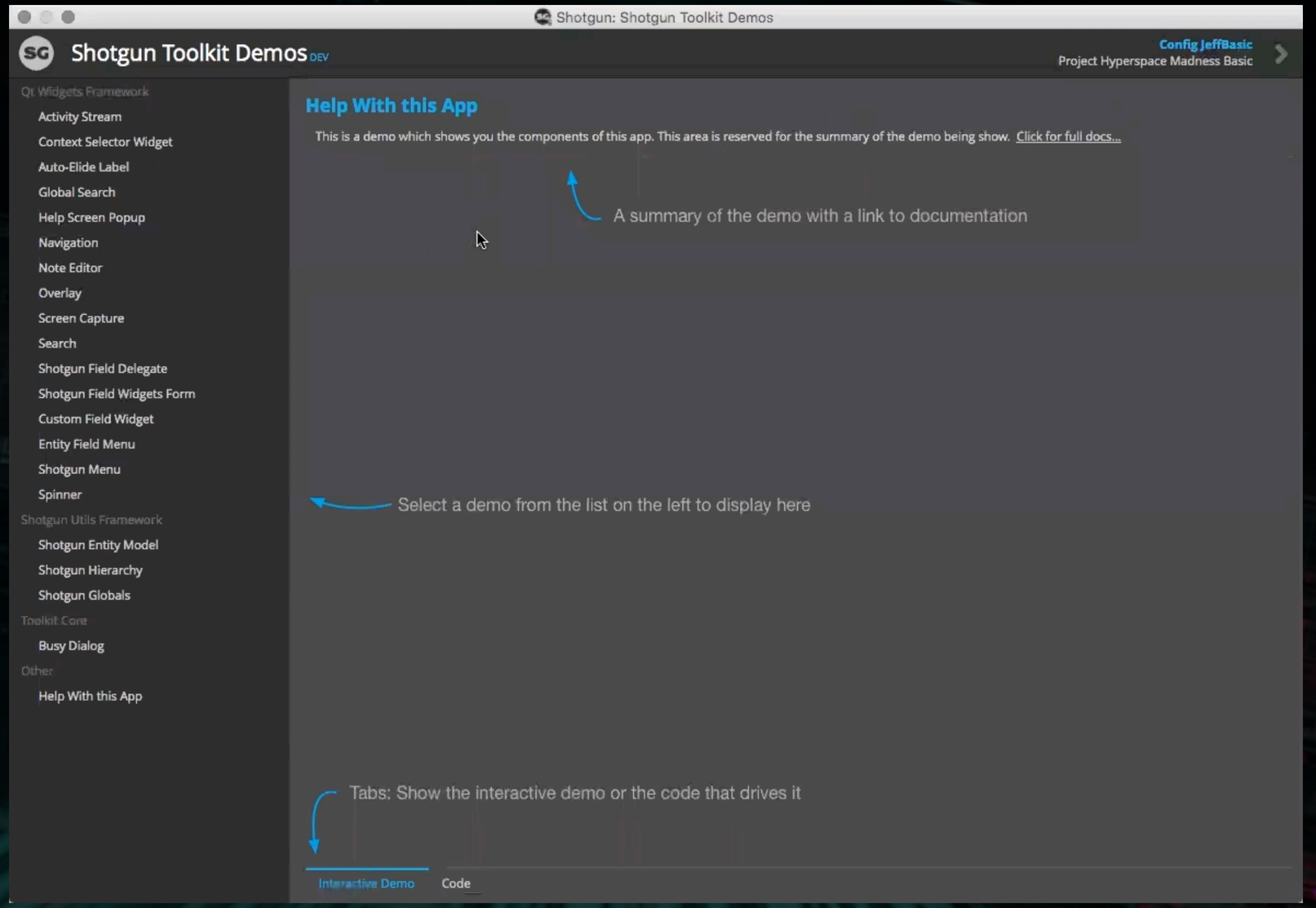


Contents of the **Python module**



Using Frameworks

Reusable components out of the box



https://github.com/shotgunsoftware/tk-multi-demo

https://github.com/shotgunsoftware/tk-framework-qtwidgets

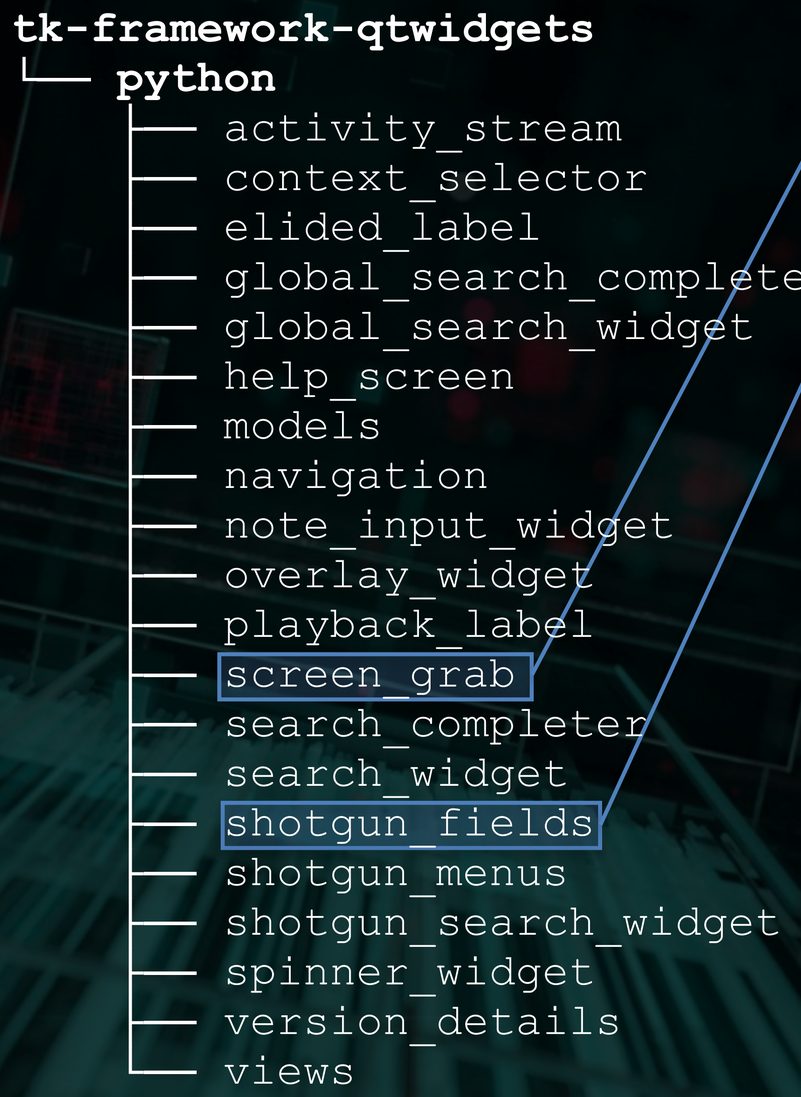
http://developer.shotgunsoftware.com/tk-framework-qtwidgets

Required frameworks in **info.yml**

```
# expected fields in the configuration file for this engine
configuration:
  cc:
    type: str
    default_value: ""
    description: A comma-separated list of Shotgun user names to CC by default.

# More verbose description of this item
display_name: "Shotgun Toolkit Bug Reporter"
description: "An App that can be used to quickly submit a new bug ticket to the support team."

# the frameworks required to run this app
frameworks:
  - {"name": "tk-framework-qtwidgets", "version": "v2.x.x", "minimum_version": "v2.7.0"}
```

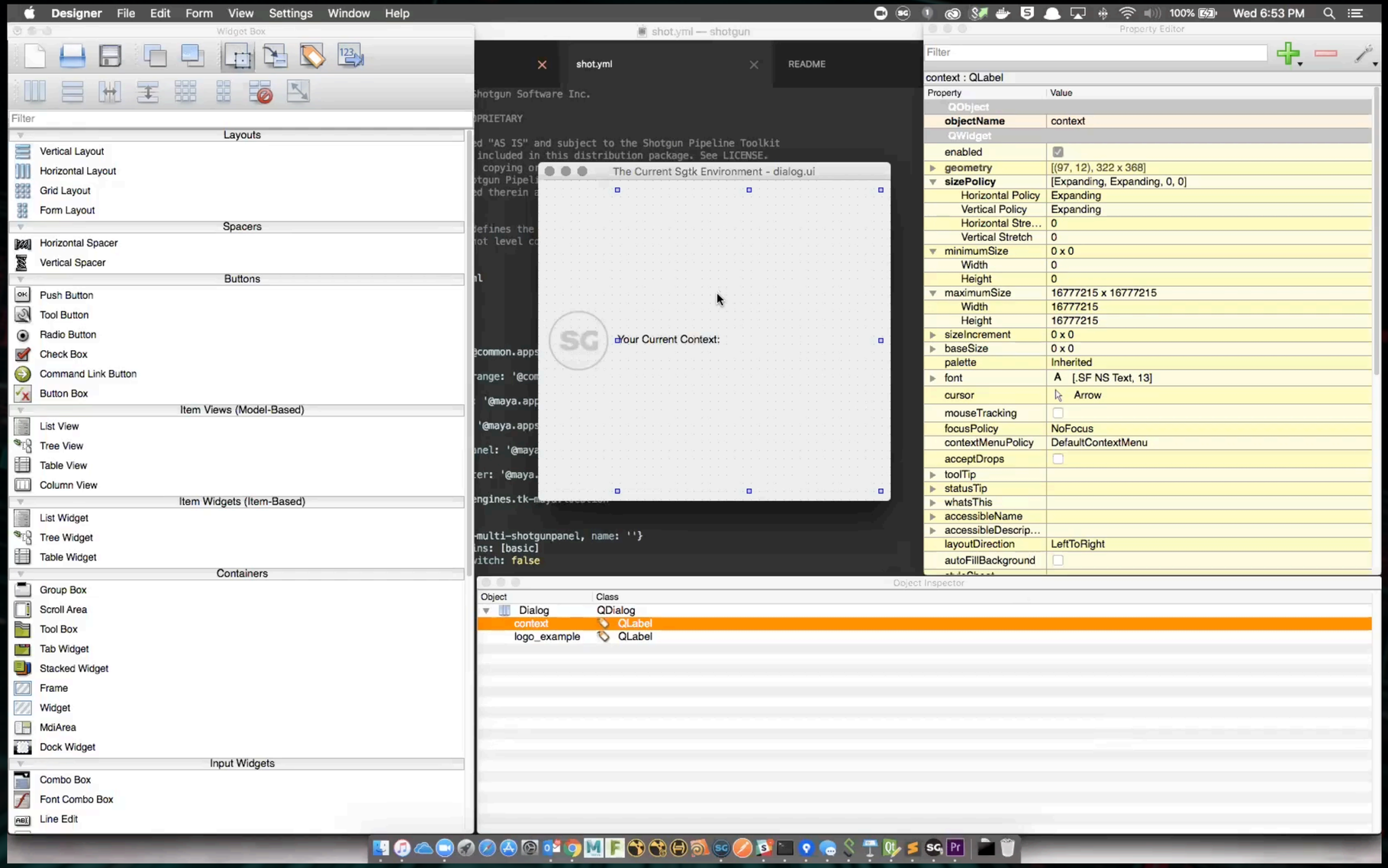
Importing from a framework

```
screen_grab = sgtk.platform.import_framework("tk-framework-qtwidgets", "screen_grab")
shotgun_fields = sgtk.platform.import_framework("tk-framework-qtwidgets", "shotgun_fields")
```



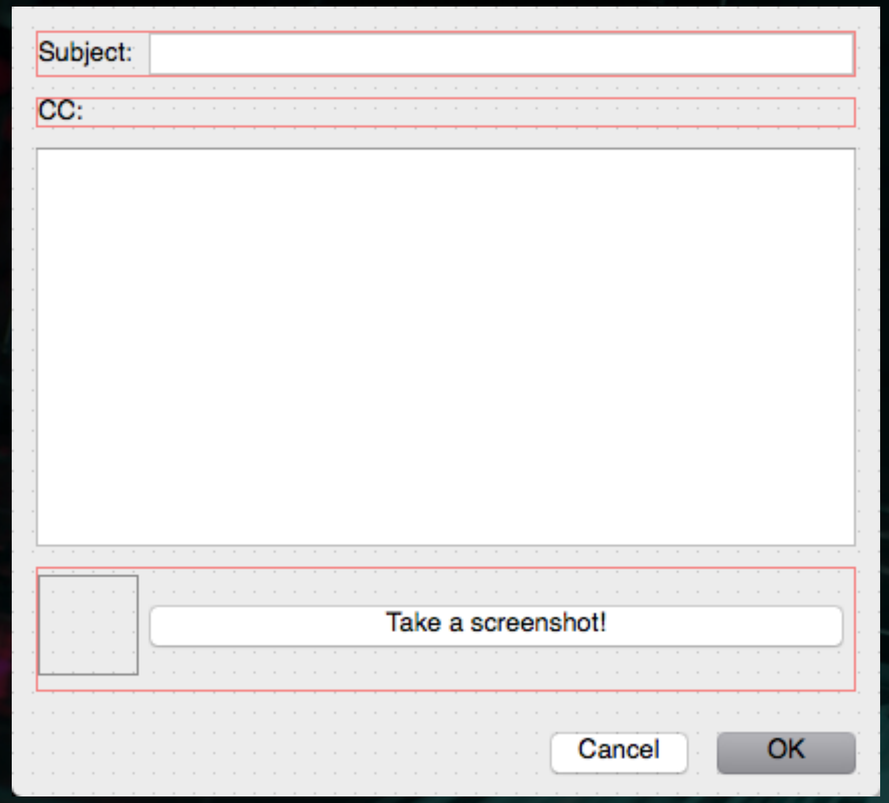
Developing and Configuring the App

From **starterapp** to **bugreporter**

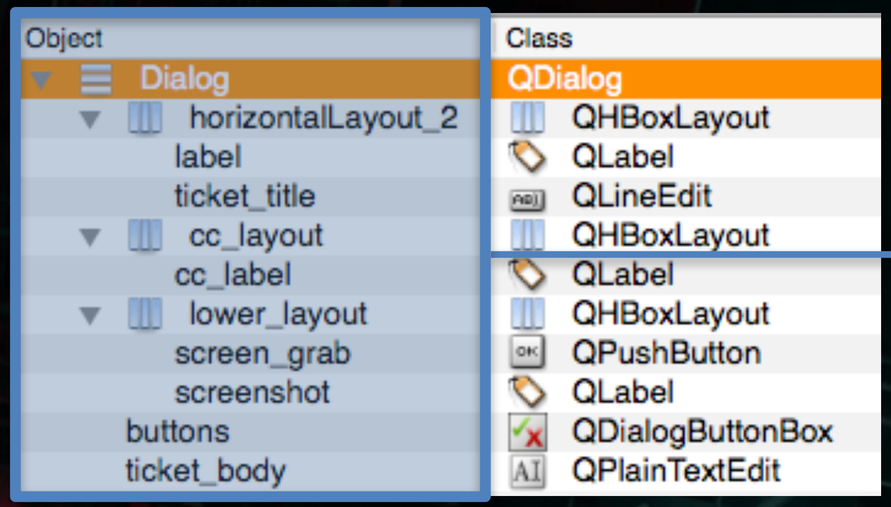


Start with a template: **tk-multi-starterapp**

https://www.github.com/shotgunsoftware/tk-multi-starterapp

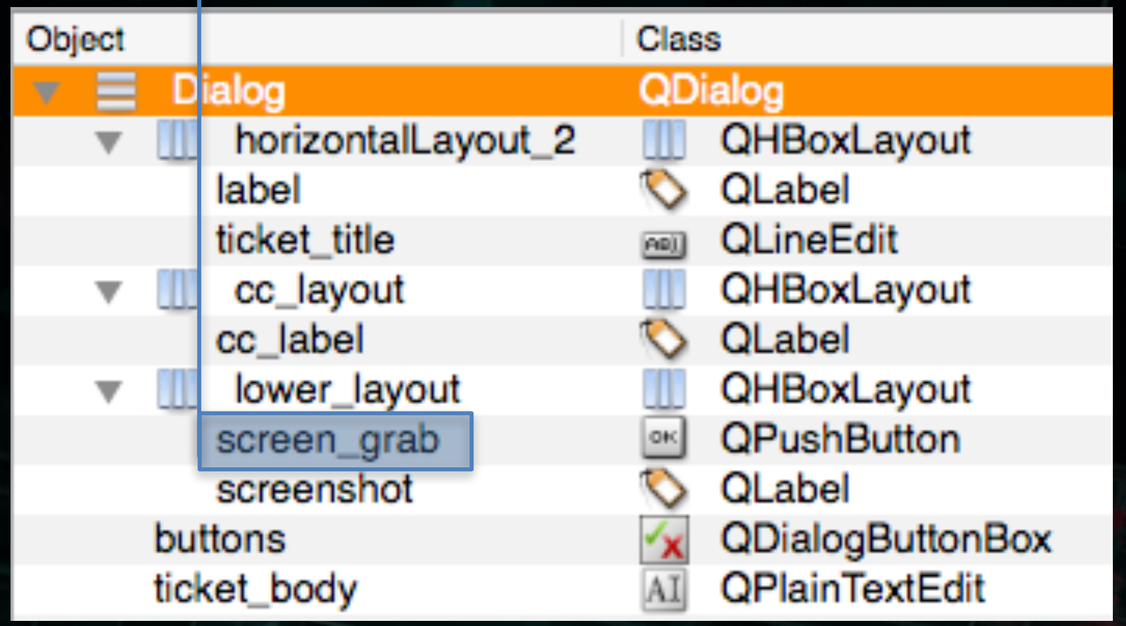


Construct the basic layout of the App



Name widgets and layouts identifiably

```
self.ui.screen_grab.clicked.connect(self.screen_grab)
```



```
tk-multi-bugreporter
├── app.py
├── info.yml
├── python
│   ├── __init__.py
│   └── app
│       ├── __init__.py
│       ├── dialog.py
│       └── ui
│           ├── __init__.py
│           ├── dialog.py
│           └── resources_rc.py
├── resources
│   ├── build_resources.sh
│   ├── dialog.ui
│   ├── resources.qrc
│   └── sg_logo.png
└── style.qss
```

dialog.ui

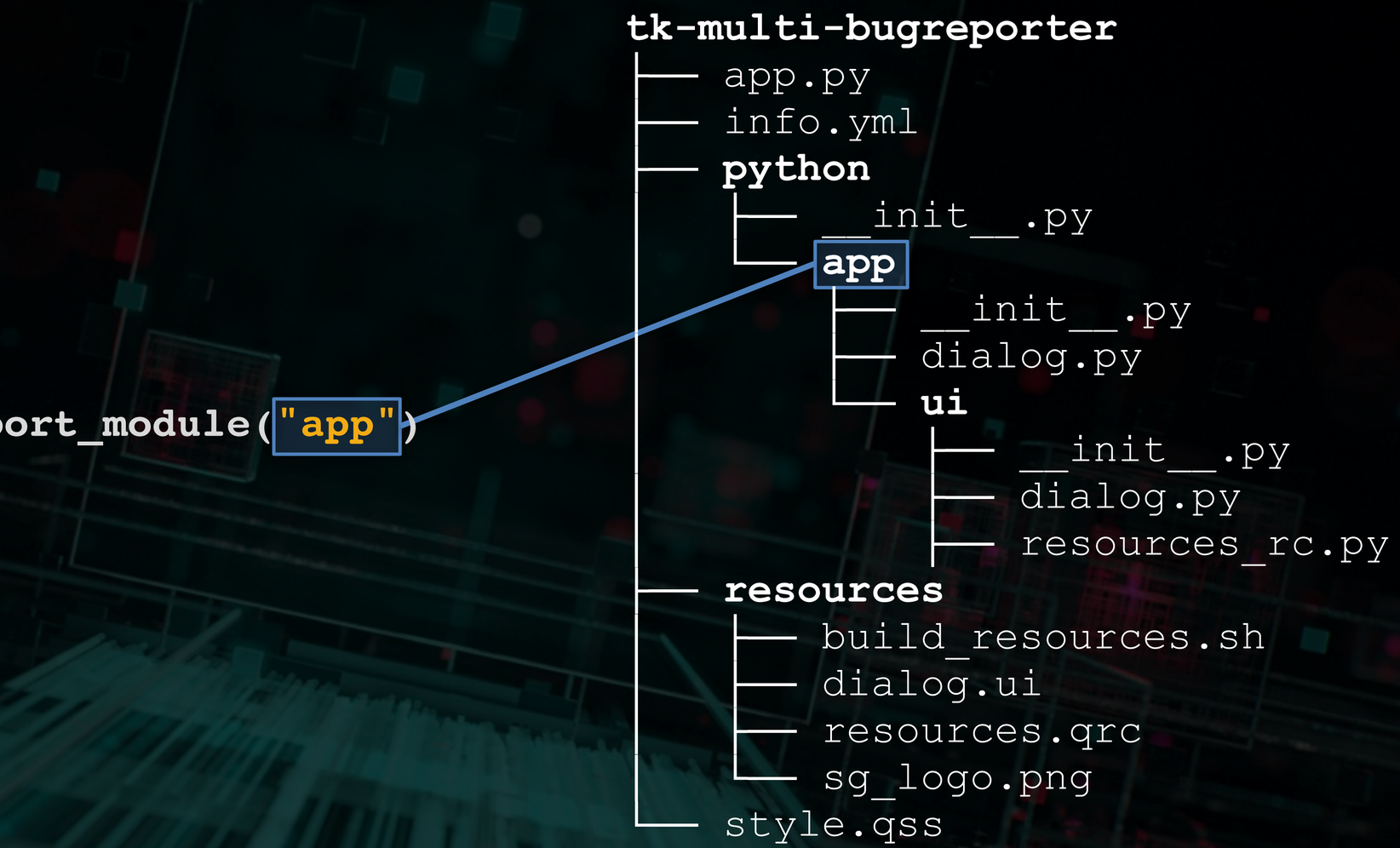
What's happening in **app.py**?

```python
from sgtk.platform import Application

class BugReporter(Application):
    """
    An App that can be used to submit a bug ticket to support team in Shotgun.
    """
    def init_app(self):
        """
        Called as the application is being initialized
        """
        # first, we use the special import_module command to access the app module
        # that resides inside the python folder in the app. This is where the actual UI
        # and business logic of the app is kept. By using the import_module command,
        # toolkit's code reload mechanism will work properly.
        app_payload = self.import_module("app")

        # now register a *command*, which is normally a menu entry of some kind on a Shotgun
        # menu (but it depends on the engine). The engine will manage this command and
        # whenever the user requests the command, it will call out to the callback.

        # first, set up our callback, calling out to a method inside the app module contained
        # in the python folder of the app
        menu_callback = lambda : app_payload.dialog.show_dialog(self)

        # now register the command with the engine
        self.engine.register_command("Report Bugs!", menu_callback)
```

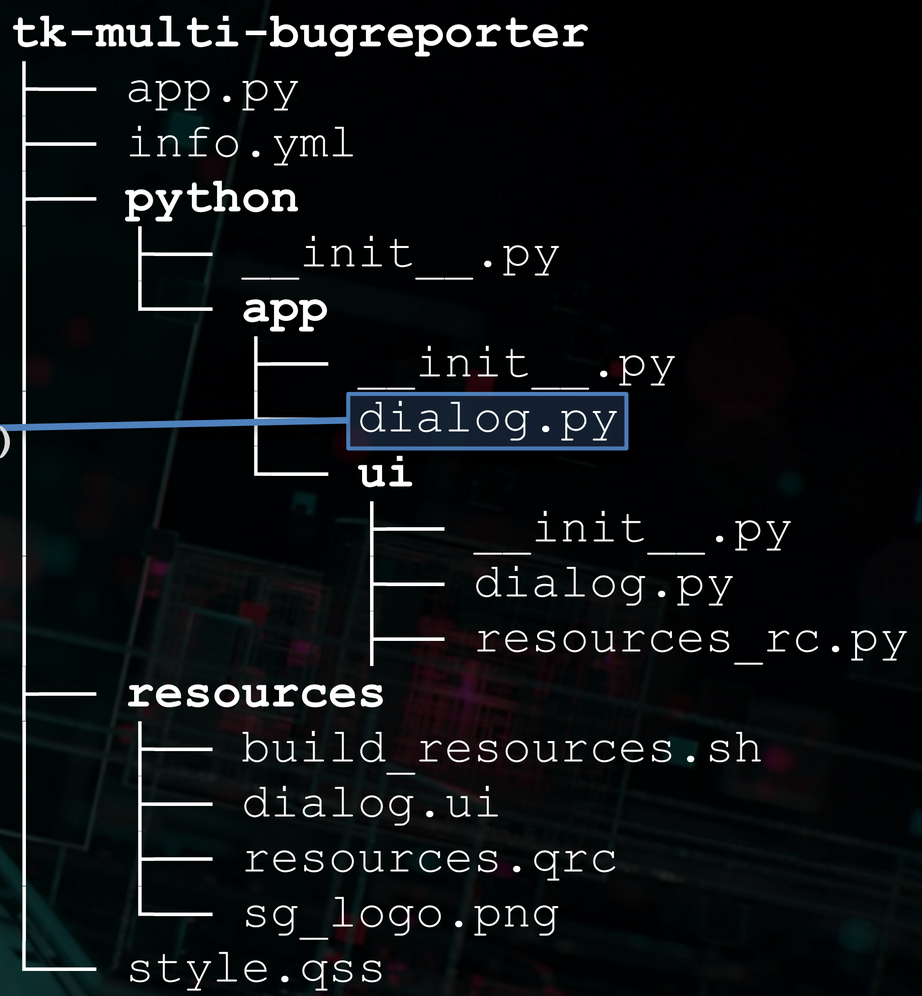
What's happening in **app.py**?

```
from sgtk.platform import Application

class BugReporter(Application):
    """
    An App that can be used to submit a bug ticket to support team in Shotgun.
    """
    def init_app(self):
        """
        Called as the application is being initialized
        """
        # first, we use the special import_module command to access the app module
        # that resides inside the python folder in the app. This is where the actual UI
        # and business logic of the app is kept. By using the import_module command,
        # toolkit's code reload mechanism will work properly.
        app_payload = self.import_module("app")

        # now register a *command*, which is normally a menu entry of some kind on a Shotgun
        # menu (but it depends on the engine). The engine will manage this command and
        # whenever the user requests the command, it will call out to the callback.

        # first, set up our callback, calling out to a method inside the app module contained
        # in the python folder of the app
        menu_callback = lambda : app_payload.dialog.show_dialog(self)

        # now register the command with the engine
        self.engine.register_command("Report Bugs!", menu_callback)
```

```
app_payload = self.import_module("app")
```



```
menu_callback = lambda : app_payload.dialog.show_dialog(self)

# now register the command with the engine
self.engine.register_command("Report Bugs!", menu_callback)
```



What's happening in **dialog.py**?

```
# by importing QT from sgtk rather than directly, we ensure that
# the code will be compatible with both PySide and PyQt.
from sgtk.platform.qt import QtCore, QtGui
from .ui.dialog import Ui_Dialog

screen_grab = sgtk.platform.import_framework("tk-framework-qtwidgets", "screen_grab")
shotgun_fields = sgtk.platform.import_framework("tk-framework-qtwidgets", "shotgun_fields")

def show_dialog(app_instance):
    """
    Shows the main dialog window.
    """
    # in order to handle UIs seamlessly, each toolkit engine has methods for launching
    # different types of windows. By using these methods, your windows will be correctly
    # decorated and handled in a consistent fashion by the system.

    # we pass the dialog class to this method and leave the actual construction
    # to be carried out by toolkit.
    app_instance.engine.show_dialog("Report Bugs!", app_instance, AppDialog)
```

What's happening in **dialog.py**?

```
# by importing QT from sgtk rather than directly, we ensure that
# the code will be compatible with both PySide and PyQt.
from sgtk.platform.qt import QtCore, QtGui
from .ui.dialog import Ui_Dialog

screen_grab = sgtk.platform.import_framework("tk-framework-qtwidgets", "screen_grab")
shotgun_fields = sgtk.platform.import_framework("tk-framework-qtwidgets", "shotgun_fields")

def show_dialog(app_instance):
    """
    Shows the main dialog window.
    """
    # in order to handle UIs seamlessly, each toolkit engine has methods for launching
    # different types of windows. By using these methods, your windows will be correctly
    # decorated and handled in a consistent fashion by the system.

    # we pass the dialog class to this method and leave the actual construction
    # to be carried out by toolkit.
    app_instance.engine.show_dialog("Report Bugs!", app_instance, AppDialog)
```

What's happening in **AppDialog**?

```
class AppDialog(QtGui.QWidget):
    """
    Main application dialog window
    """
    def __init__(self):
        """
        Constructor
        """

    def _get_shotgun_fields(self):
        """
        Populates the CC list Shotgun field widget.
        """

    def screen_grab(self):
        """
        Triggers a screen grab to be initiated.
        """

    def create_ticket(self):
        """
        Creates a new Ticket entity in Shotgun from the contents of the dialog.
        """
```

```
def __init__(self):
    """
    Constructor
    """
    QtGui.QWidget.__init__(self)

    # now load in the UI that was created in the UI designer
    self.ui = Ui_Dialog()
    self.ui.setupUi(self)

    self._app = sgtk.platform.current_bundle()

    self.ui.buttons.accepted.connect(self.create_ticket)
    self.ui.buttons.rejected.connect(self.close)
    self.ui.screen_grab.clicked.connect(self.screen_grab)

    self._screenshot = None
    self._cc_widget = None

    # The ShotgunFieldManager is a factory used to build widgets for fields
    # associated with an entity type in Shotgun. It pulls down the schema from
    # Shotgun asynchronously when the initialize method is called, so before
    # we do that we need to hook up the signal it emits when it's done to our
    # method that gets the widget we want and adds it to the UI.
    self._field_manager = shotgun_fields.ShotgunFieldManager(parent=self)
    self._field_manager.initialized.connect(self._get_shotgun_fields)
    self._field_manager.initialize()
```

```
def __init__(self):
    """
    Constructor
    """
    QtGui.QWidget.__init__(self)

    # now load in the UI that was created in the UI designer
    self.ui = Ui_Dialog()
    self.ui.setupUi(self)

    self._app = sgtk.platform.current_bundle()

    self.ui.buttons.accepted.connect(self.create_ticket)
    self.ui.buttons.rejected.connect(self.close)
    self.ui.screen_grab.clicked.connect(self.screen_grab)

    self._screenshot = None
    self._cc_widget = None

    # The ShotgunFieldManager is a factory used to build widgets for fields
    # associated with an entity type in Shotgun. It pulls down the schema from
    # Shotgun asynchronously when the initialize method is called, so before
    # we do that we need to hook up the signal it emits when it's done to our
    # method that gets the widget we want and adds it to the UI.
    self._field_manager = shotgun_fields.ShotgunFieldManager(parent=self)
    self._field_manager.initialized.connect(self._get_shotgun_fields)
    self._field_manager.initialize()
```

```
# The ShotgunFieldManager is a factory used to build widgets for fields
# associated with an entity type in Shotgun. It pulls down the schema from
# Shotgun asynchronously when the initialize method is called, so before
# we do that we need to hook up the signal it emits when it's done to our
# method that gets the widget we want and adds it to the UI.
self._field_manager = shotgun_fields.ShotgunFieldManager(parent=self)
self._field_manager.initialized.connect(self._get_shotgun_fields)
self._field_manager.initialize()
```

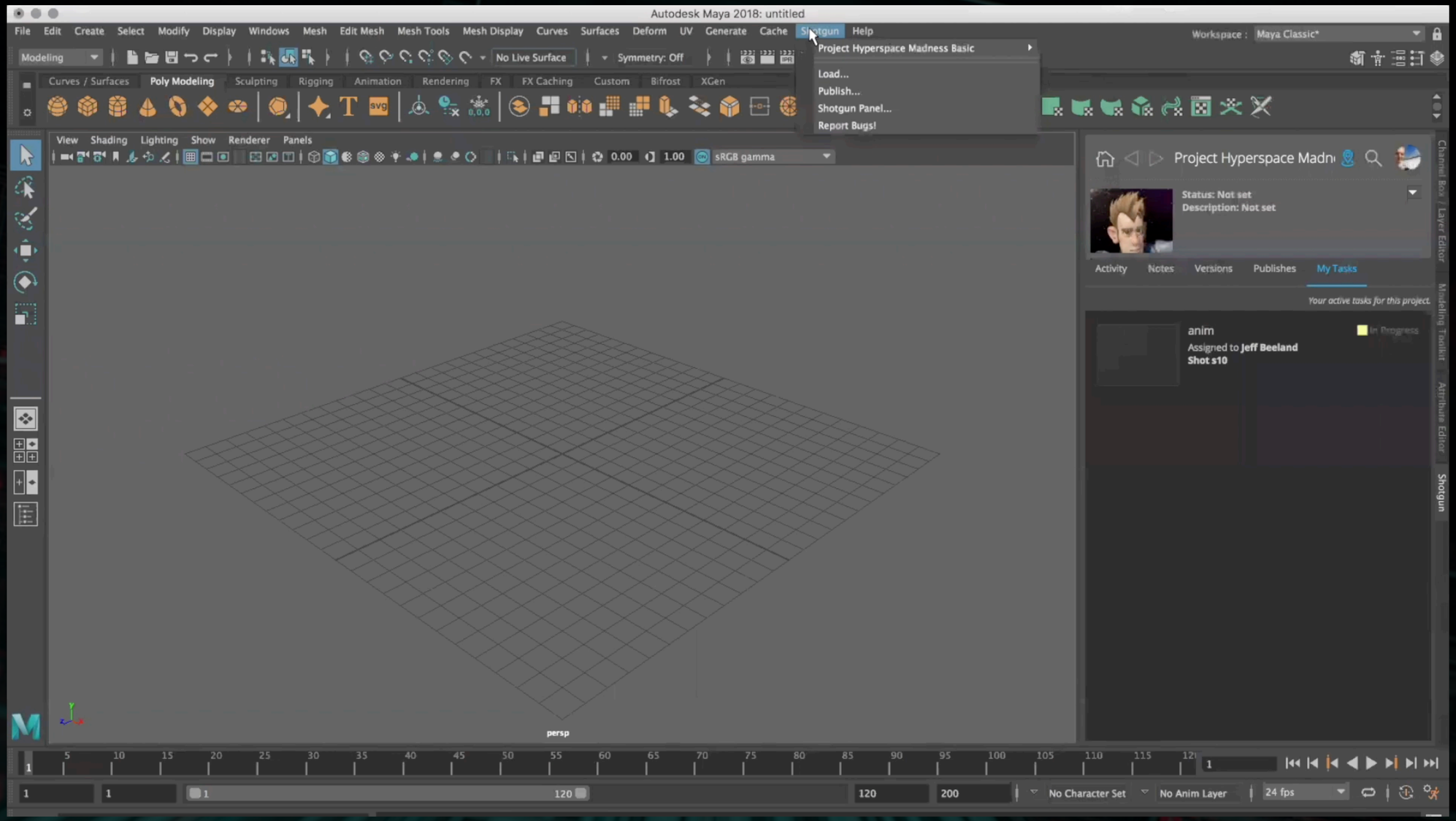
```
# The ShotgunFieldManager is a factory used to build widgets for fields
# associated with an entity type in Shotgun. It pulls down the schema from
# Shotgun asynchronously when the initialize method is called, so before
# we do that we need to hook up the signal it emits when it's done to our
# method that gets the widget we want and adds it to the UI.
self._field_manager = shotgun_fields.ShotgunFieldManager(parent=self)
self._field_manager.initialized.connect(self._get_shotgun_fields)
self._field_manager.initialize()
```

```
def _get_shotgun_fields(self):
    """
    Populates the CC list Shotgun field widget.
    """
    # Get a list of user entities from Shotgun representing the default
    # list that we'll pull from the "cc" config setting for the app.
    raw_cc = self._app.get_setting("cc", "")
    users = self._app.shotgun.find(
        "HumanUser",
        [["login", "in", re.split(r"[,\s]+", raw_cc)]],
        fields=("id", "type", "name")
    )

    # Create the widget that the user will use to view the default CC
    # list, plus enter in any additional users if the choose to do so.
    self._cc_widget = self._field_manager.create_widget(
        "Ticket",
        "addressings_cc",
        parent=self,
    )

    # Add our list of default users to the CC widget and then add the
    # widget to the appropriate layout.
    self._cc_widget.set_value(users)
    self.ui.cc_layout.addWidget(self._cc_widget)
```

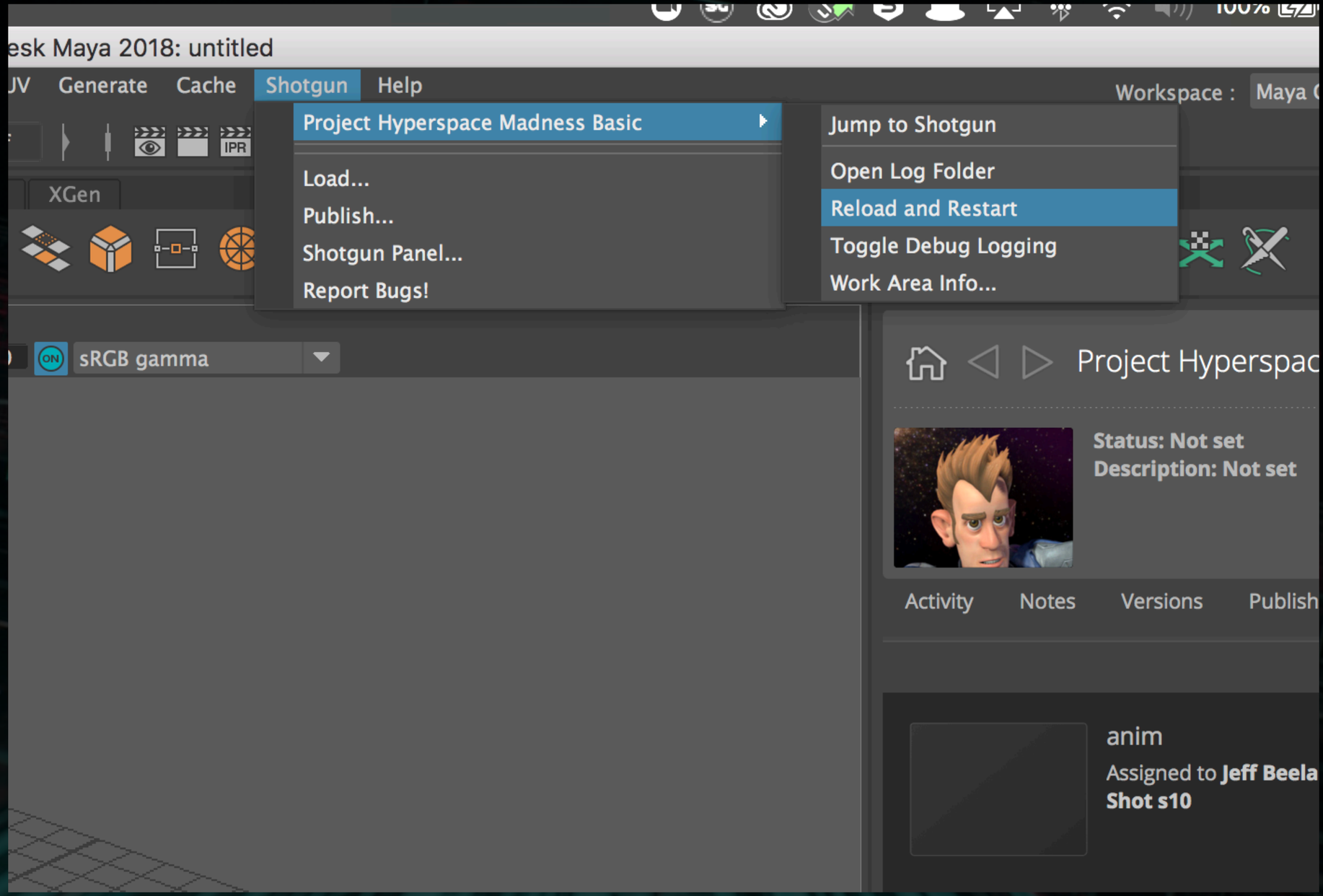
```
def _get_shotgun_fields(self):
    """
    Populates the CC list Shotgun field widget.
    """
    # Get a list of user entities from Shotgun representing the default
    # list that we'll pull from the "cc" config setting for the app.
    raw_cc = self._app.get_setting("cc", "")
    users = self._app.shotgun.find(
        "HumanUser",
        [["login", "in", re.split(r"[,\s]+", raw_cc)]],
        fields=("id", "type", "name")
    )

    # Create the widget that the user will use to view the default CC
    # list, plus enter in any additional users if the choose to do so.
    self._cc_widget = self._field_manager.create_widget(
        "Ticket",
        "addressings_cc",
        parent=self,
    )

    # Add our list of default users to the CC widget and then add the
    # widget to the appropriate layout.
    self._cc_widget.set_value(users)
    self.ui.cc_layout.addWidget(self._cc_widget)
```

```python
def _get_shotgun_fields(self):
    """
    Populates the CC list Shotgun field widget.
    """
    # Get a list of user entities from Shotgun representing the default
    # list that we'll pull from the "cc" config setting for the app.
    raw_cc = self._app.get_setting("cc", "")
    users = self._app.shotgun.find(
        "HumanUser",
        [["login", "in", re.split(r"[,\s]+", raw_cc)]],
        fields=("id", "type", "name")
    )

    # Create the widget that the user will use to view the default CC
    # list, plus enter in any additional users if the choose to do so.
    self._cc_widget = self._field_manager.create_widget(
        "Ticket",
        "addressings_cc",
        parent=self,
    )

    # Add our list of default users to the CC widget and then add the
    # widget to the appropriate layout.
    self._cc_widget.set_value(users)
    self.ui.cc_layout.addWidget(self._cc_widget)
```

```
def screen_grab(self):
    """
    Triggers a screen grab to be initiated.
    """
    pixmap = screen_grab.ScreenGrabber.screen_capture()
    self._screenshot = pixmap
    self.ui.screenshot.setPixmap(pixmap.scaled(100, 100))
```


Autodesk Maya 2018: untitled
File Edit Create Select Modify Display Windows Mesh Edit Mesh Mesh Tools Mesh Display Curves Surfaces Deform UV Generate Cache Shotgun Help
Workspace : Maya Classic*
Modeling
No Live Surface
Symmetry: Off
Curves / Surfaces Poly Modeling Sculpting Rigging Animation Rendering FX FX Caching Custom Bifrost XGen
View Shading Lighting Show Renderer Panels
sRGB gamma
persp
Project Hyperspace Madness Basic
Load...
Publish...
Shotgun Panel...
Report Bugs!
Project Hyperspace Madn
Status: Not set
Description: Not set
Activity Notes Versions Publishes My Tasks
Your active tasks for this project.
anim
In Progress
Assigned to Jeff Beeland
Shot s10
Channel Box / Layer Editor
Modeling Toolkit
Attribute Editor
Shotgun
No Character Set
No Anim Layer
24 fps

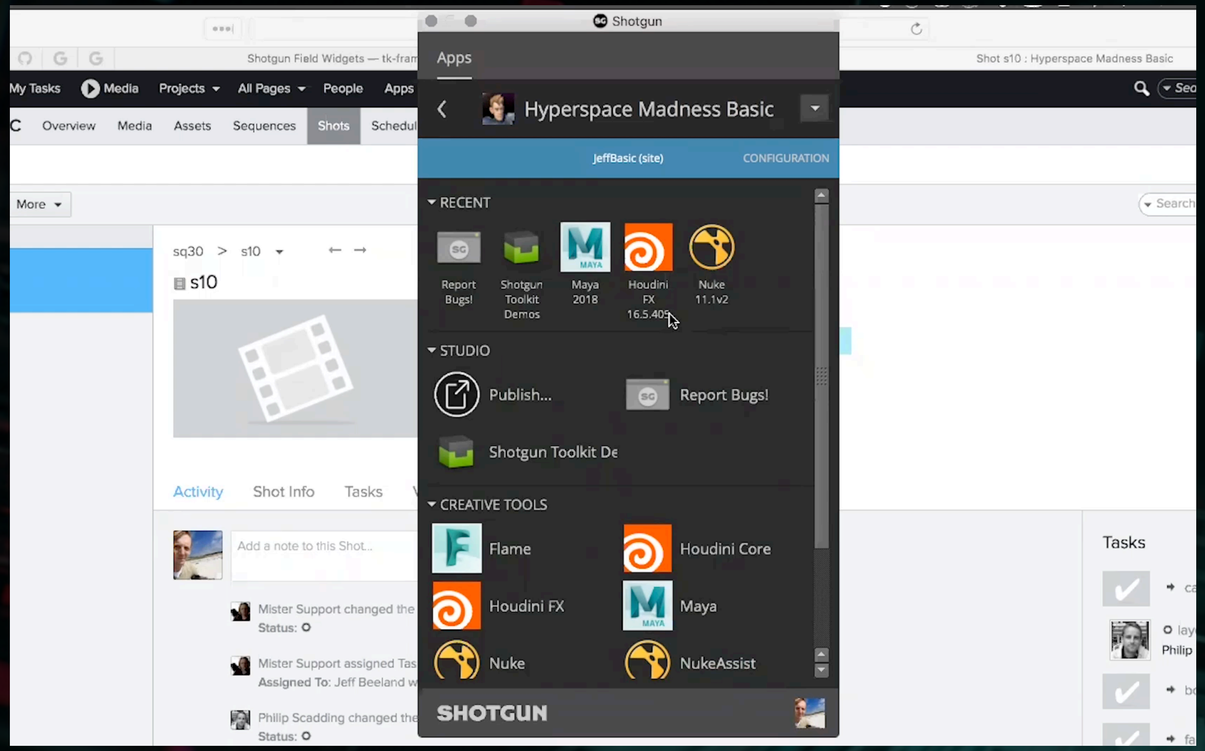
AUTODESK


esk Maya 2018: untitled
JV
Generate
Cache
Shotgun
Help
Workspace :
Maya
Project Hyperspace Madness Basic
Load...
Publish...
Shotgun Panel...
Report Bugs!
Jump to Shotgun
Open Log Folder
Reload and Restart
Toggle Debug Logging
Work Area Info...
XGen
IPR
sRGB gamma
Project Hyperspac
Status: Not set
Description: Not set
Activity
Notes
Versions
Publish
anim
Assigned to Jeff Beela
Shot s10

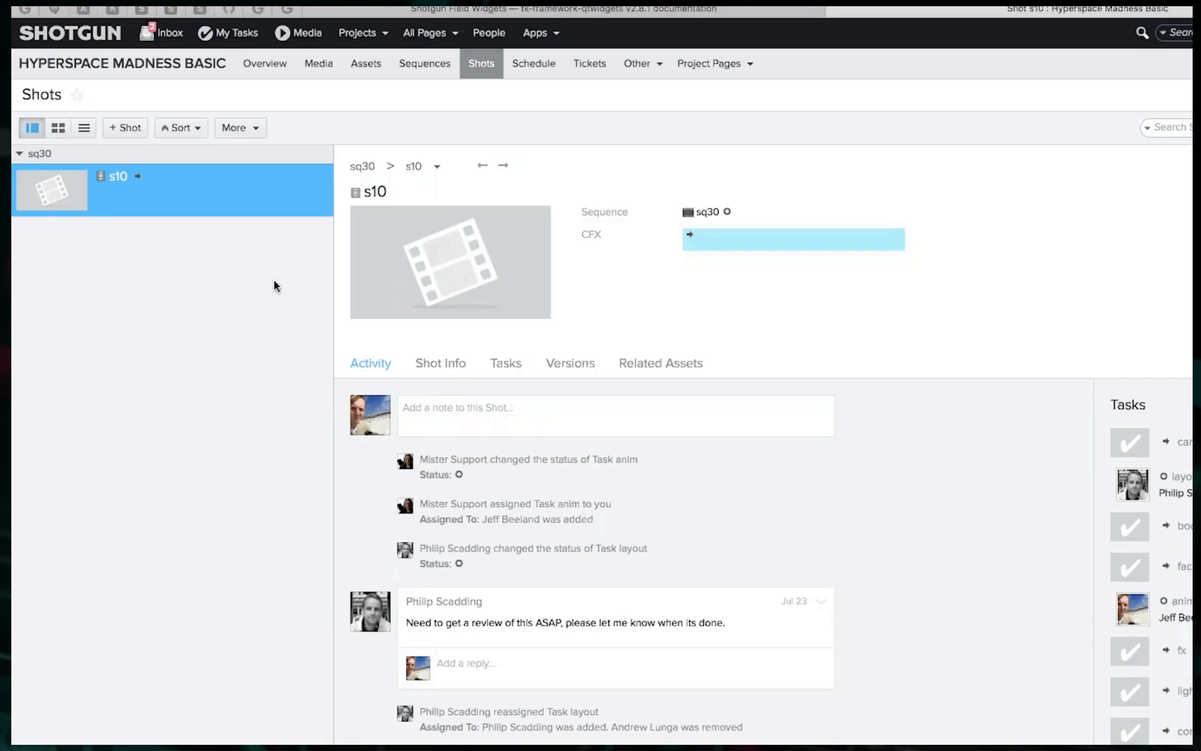
```
def create_ticket(self):
    """
    Creates a new Ticket entity in Shotgun from the contents of the dialog.
    """
    result = self._app.shotgun.create(
        "Ticket",
        dict(
            project=self._app.context.project,
            title=self.ui.ticket_title.text(),
            description=self.ui.ticket_body.toPlainText(),
            addressings_cc=self._cc_widget.get_value(),
        )
    )

    if self._screenshot:
        path = tempfile.mkstemp(suffix=".png")[1]
        file_obj = QtCore.QFile(path)
        file_obj.open(QtCore.QIODevice.WriteOnly)
        self._screenshot.save(file_obj, "PNG")
        self._app.shotgun.upload(
            “Ticket",
            result["id"],
            path,
            "attachments",
        )

    QtGui.QMessageBox.information(
        self,
        "Ticket successfully created!",
        "Ticket #%s successfully submitted!" % result["id"],
    )
    self.close()
```

```
def create_ticket(self):
    """
    Creates a new Ticket entity in Shotgun from the contents of the dialog.
    """
    result = self._app.shotgun.create(
        "Ticket",
        dict(
            project=self._app.context.project,
            title=self.ui.ticket_title.text(),
            description=self.ui.ticket_body.toPlainText(),
            addressings_cc=self._cc_widget.get_value(),
        )
    )

    if self._screenshot:
        path = tempfile.mkstemp(suffix=".png")[1]
        file_obj = QtCore.QFile(path)
        file_obj.open(QtCore.QIODevice.WriteOnly)
        self._screenshot.save(file_obj, "PNG")
        self._app.shotgun.upload(
            “Ticket",
            result["id"],
            path,
            "attachments",
        )

    QtGui.QMessageBox.information(
        self,
        "Ticket successfully created!",
        "Ticket #%s successfully submitted!" % result["id"],
    )
    self.close()
```

Launch from SG Desktop!



Launch from Shotgun!



AUTODESK

Potential **additional features**

* Gather information about the DCC session and include it in the Ticket:
  * Which DCC is being used, and which version?
  * Collect the Toolkit log file and upload it as a Ticket attachment?
  * Include the user’s current context in the Ticket body?
* Allow for multiple screenshots
* Support for arbitrary file attachments
* User-settable priority/severity
* Progress bar when creating the new Ticket and uploading attachments

AUTODESK.


https://github.com/shotgunsoftware/sg-devday-2018


AUTODESK