Dynamic filesystem management

In this guide you will learn how to configure dynamic folder creation, automate file naming, and automate saving files to the appropriate place in your filesystem. You will also learn how to add a custom entity to your projects configuration that facilitates adding a new folder to the supported filesystem and adding a new variable to the file naming.

About the Guide

We understand that one of the hardest things about managing a pipeline is keeping track of the thousands of files that are created. This guide will provide the knowledge necessary to modify a project's configuration in a way that allows for Toolkit to automatically create the proper folder structure, name your files, keep track of versioning, and allow apps running on Toolkit to manage saving and retrieving them.

This guide will demonstrate how to edit configuration settings that Toolkit apps use to manage and track elements associated with a production set. There are several scenes in a dining room, a living room, and outdoor scenes. The goal is to add an entity in the project's configuration and use this entity to manage and name all the assets created for any identified set. This will enable Toolkit to dynamically build the folder structure you specify for the assets in the dining room, living room, or any set, and put the assets where you want them, automatically naming them according to the format you choose. It's important for large productions to organize 3D elements in a way that makes sense for that production. This exercise will create a separate folder structure for the 3D elements in the dining room, distinguishing them from elements in the living room or any of the outdoor scenes. The name of the file will be dynamically created using the set entity to distinguish files for the dining room from files used in other sets. You will edit a pipeline configuration that allows a Maya Shotgun integration to dynamically do all that.

There are three parts to this guide:

1. Creating a custom entity in Shotgun called Set that you will use to associate with the dining room elements the artists are creating.
2. Editing the schema enabling Toolkit to dynamically create the folder structure used to save the files associated with the dining room.
3. Editing the template used for naming the 3D dining room asset work files, enabling Toolkit to name the associated files.

Prerequisites

To use this guide and perform an edit on a pipeline configuration, the following is required:

1. An active [Shotgun](https://www.shotgunsoftware.com/signup/) site.

2. A basic understanding of how a Shotgun site is used to [manage assets](https://support.shotgunsoftware.com/hc/en-us/articles/219031098-Using-the-importer-to-create-and-track-things).
3. A cloned pipeline configuration for the identified project, or complete the [Getting started with configurations](./advanced_config.md) guide and clone the configuration created in that exercise.
4. [Shotgun Desktop](https://support.shotgunsoftware.com/hc/en-us/articles/115000068574-Integrations-user-guide#Installation%20of%20Desktop) installed on your system.
5. Familiarity with YAML, Toolkit uses YAML files for configuring Shotgun integration settings. Read and write permissions set appropriately for the filesystem where the Pipeline Configuration is stored.
6. An active subscription for Maya. Get a 30 day trial of [Maya](https://www.autodesk.com/products/maya/free-trial-dts?adobe_mc_ref=https%3A%2F%2Fwww.google.com%2F&adobe_mc_sdid=SDID%3D577C0A84DDF5D35D-50E96EA2052056FE%7CMCORGID%3D6DC7655351E5696B0A490D44%2540AdobeOrg%7CTS%3D1543444689)
7. A solid understanding of how to create a [new task](https://support.shotgunsoftware.com/hc/en-us/articles/219031288-Tasks-and-Pipeline-Steps) for a 3D model asset in Shotgun.
8. Read and write permissions set appropriately to allow Toolkit to read and write to the production filesystem.

{% include info title="Note" content="This guide is based on the `tk-config-default2` pipeline configuration. If your config was modified, the location of files, folders, and blocks of YAML settings may vary from what is described here." %}

Additional Resources

* [Shotgun Support Site](https://support.shotgunsoftware.com)

* [App and Engine Configuration Reference](https://support.shotgunsoftware.com/hc/en-us/articles/219039878-App-and-Engine-Configuration-Reference)

Using this guide

* [Shotgun entities](https://support.shotgunsoftware.com/hc/en-us/articles/219031098-Using-the-importer-to-create-and-track-things)

* [The schema](https://support.shotgunsoftware.com/hc/en-us/articles/219031358-Shotgun-schema)

* Filesystem reference](https://support.shotgunsoftware.com/hc/en-us/articles/219039868-Integrations-File-System-Reference#The%20Keys%20Section)

About file schemas and templates

The filesystem schema and templates provided in the [Default Configuration](https://support.shotgunsoftware.com/hc/en-us/articles/115004077494-Overview-of-Toolkit-s-New-Default-Configuration-) are the building blocks that allow you to take advantage of the Shotgun architecture for managing project based elements as [entities](https://support.shotgunsoftware.com/hc/en-us/articles/219031098-Using-the-importer-to-create-and-track-things): **Asset, Sequence, and Shot.** These building blocks facilitate creating an automated system for managing files and folder creation associated with entities. Other entities such as **Character**, gaming **Level**, fx asset and Set can be added.

The Toolkit platform allows you to build your folder structure dynamically by utilizing a skeleton of the filesystem you want to create. The [schema](https://support.shotgunsoftware.com/hc/en-us/articles/219031358-Shotgun-schema) demonstrates how entities relate to other data and how the data is going to be organized in the filesystem. The schema is an explicit guide that Toolkit accesses enabling the dynamic creation of folders. It's the skeleton of a folder structure that uses YAML files to determine what folders to create and how they relate to each other. The Default Configuration includes a pre-configured schema that supports folder creation for both asset and shot pipelines. You will be modifying the portion of the schema that supports creating the asset folder structure, `/assets/<asset_type>/<asset>/<step>`, to add support for the new **Set** entity you're creating.

The [templates](https://support.shotgunsoftware.com/hc/en-us/articles/219040648#File%20System%20Template) allow you to dynamically name and save files as they're created using Shotgun metadata, data from the app being used to save the file and information from the schema structure. The templates are also used by apps to retrieve and publish files. The Default Configuration provides templates you can edit to meet the needs of your pipeline.

The Basic Configuration doesn't have the schema or templates for managing files and filesystems. The first guide, [Getting started with configurations](./advanced_config.md), discussed the difference between the [Basic Configuration](https://support.shotgunsoftware.com/hc/en-us/articles/115000067493-Integrations-Admin-Guide#The%20Basic%20Config), that is used when a new project is accessed through Shotgun Desktop and the [Default Configuration](https://support.shotgunsoftware.com/hc/en-us/articles/115004077494-Overview-of-Toolkit-s-New-Default-Configuration-), that's used with the Advanced Project Setup Wizard. Using the Default Configuration will give you a set of building blocks including settings for a

default schema, templates, and many other settings that allow for more ease when extending configurations for a Toolkit pipeline.

Begin exercise

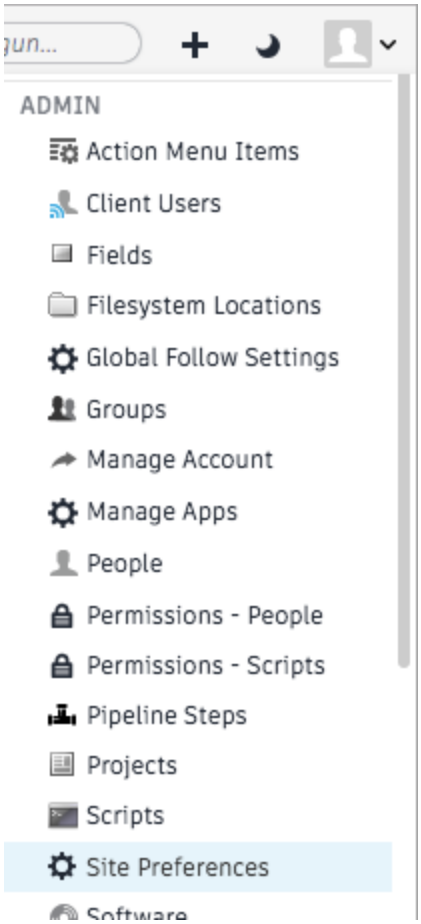
Configuring a schema and templates will allow you to dynamically manage the files generated when creating the dining room set: a place setting, a steaming hot Filet Mignon, a bottle of Penfolds Grange Hermitage 1951, potatoes au gratin, lemon garlic asparagus, etc... The files for the dining room set will be associated with the dining room set entity enabling you to manage them more easily. Set is not an entity type that comes standard with the Default Configuration, so you need to create a custom entity called Set as well as extending the schema and templates necessary to automatically manage, link and save the files for the dining room set.

For more clarification, using an “entity” allows you to make the association in Shotgun, “Set” is the name of the entity type you’re creating, and “Dining Room” is the proper name you will be giving the Set entity type.

Creating a custom entity

****Step 1:**** Open the Shotgun site with access to the the_other_side project, or the project you will be using for this exercise. Click on your **avatar** and go to **ADMIN > Site Preferences** then select the arrow next to **Entities** to open the dialog box.

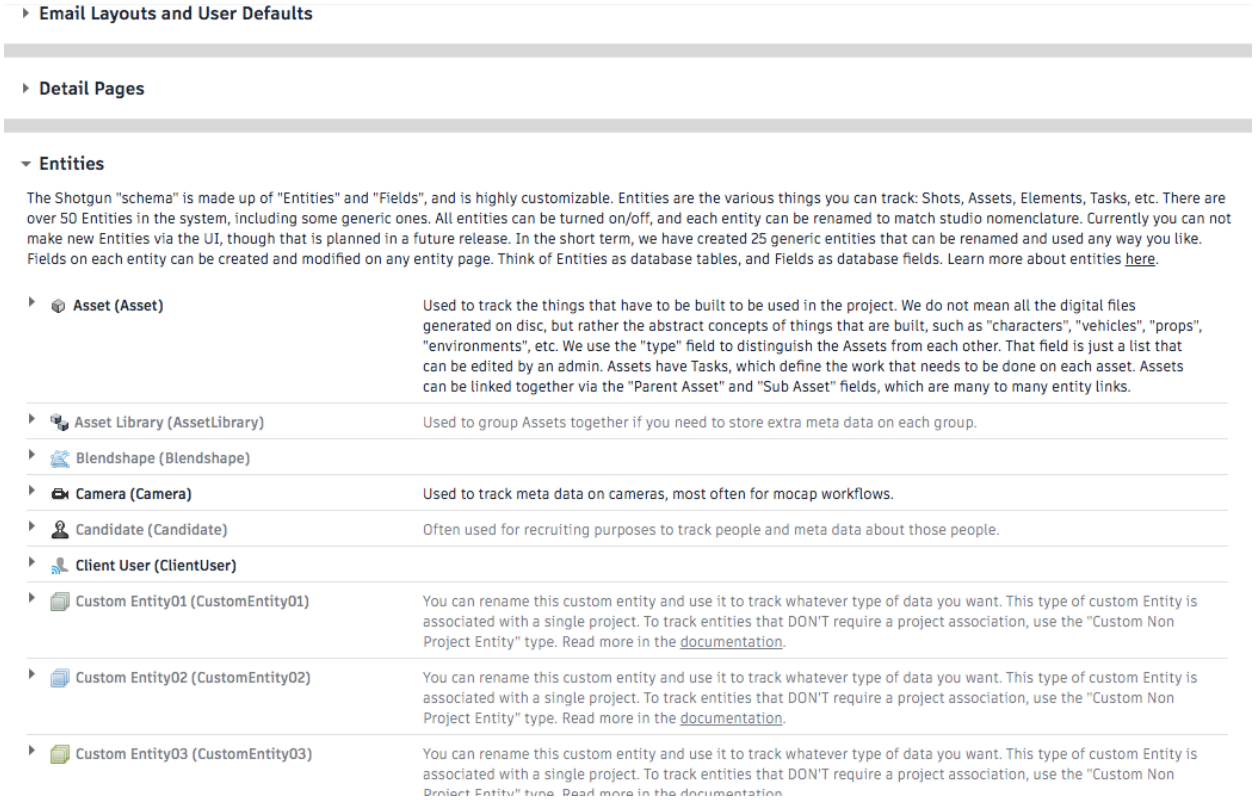
(../../../../images/toolkit/learning-resources/guides/ ??)



Displayed is a list of entity types that are available in Shotgun. At the top of the list in the image below are some entity types that are configured for the current Shotgun site. Underneath these entity types are several **Custom Entities** that are not configured or enabled.

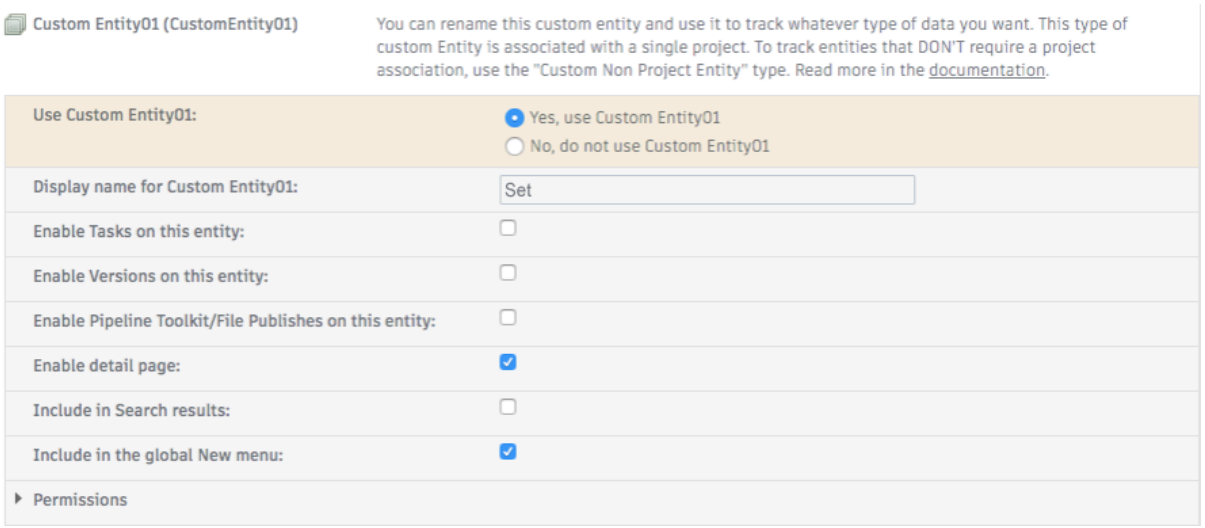
Choose one of the custom entity types, configure it, and enable it.

(../../../../images/toolkit/learning-resources/guides/ ??)



Step 2: Select the arrow to open the settings on a grayed out disabled custom entity. Select the radio button next to **Yes, use Custom Entity…**, change the **Display name** to **Set** then scroll to the top of the window and select **Save Changes**.

(../../../../images/toolkit/learning-resources/guides/ ??)



This gives that custom entity the display name ***Set*** and makes that entity active and enabled in Shotgun. Essentially you are creating an alias for the custom entity because the system name of the entity remains `CustomEntity01`. In this pipeline configuration, `CustomEntity01` is used for the settings, you might be using a different custom entity.

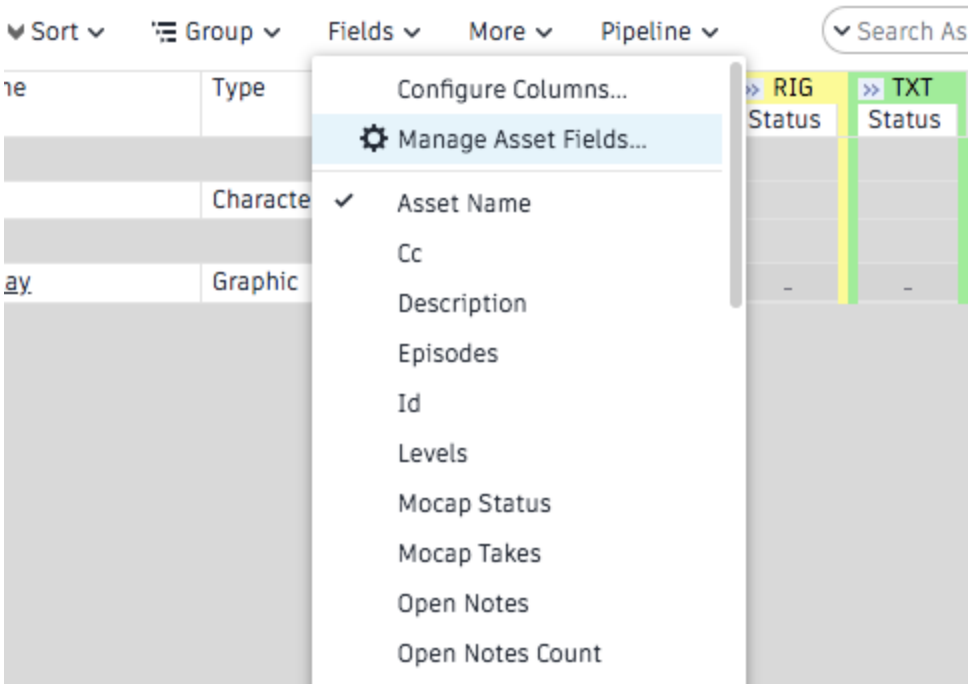
{% include info title="Note:" content="Remember the system name of the custom entity you chose." %}

Add the data field used to associate assets with the dining room set

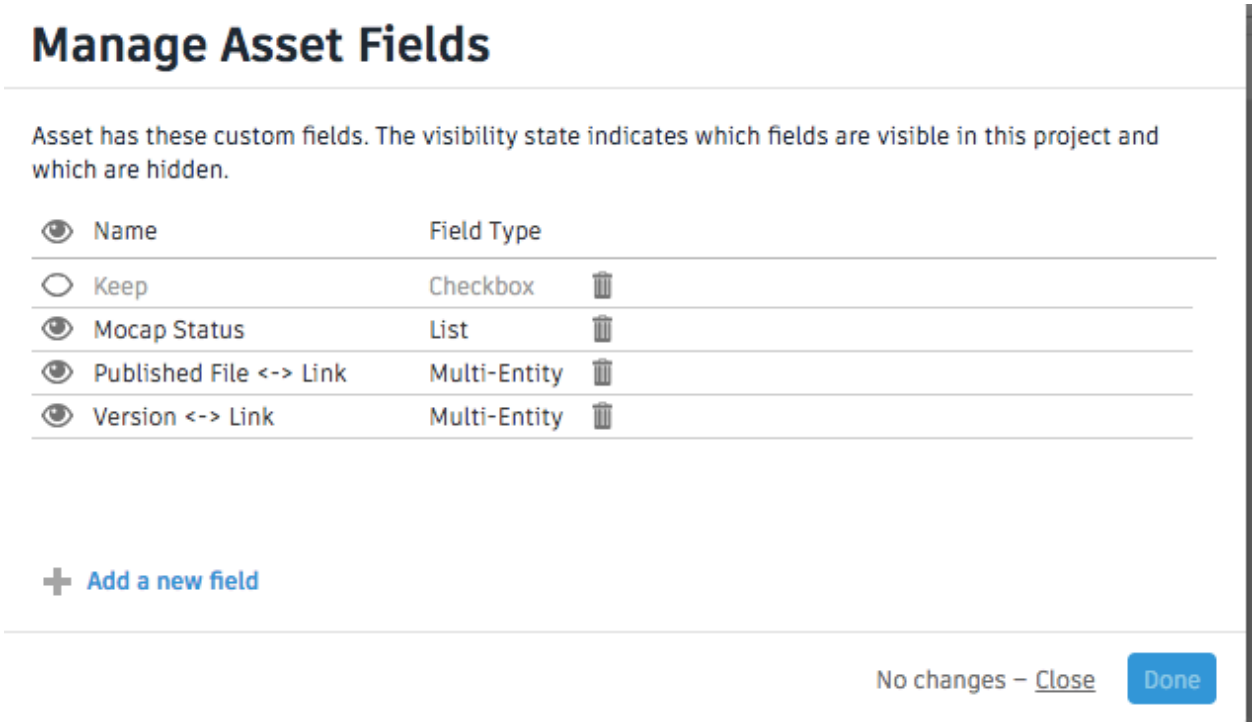
Step 3: Select the ****Projects**** drop down at the top of the page to open the project you want to use for this exercise.

Step 4: Select **Assets** in your project menu bar. In the Assets menu select ****Fields > Manage Asset Fields…****

Adding a data field to the Asset entity enables the ability to associate specific assets with the new entity. The assets the artists create for the dining room will be associated with the **Dining Room** set entity.



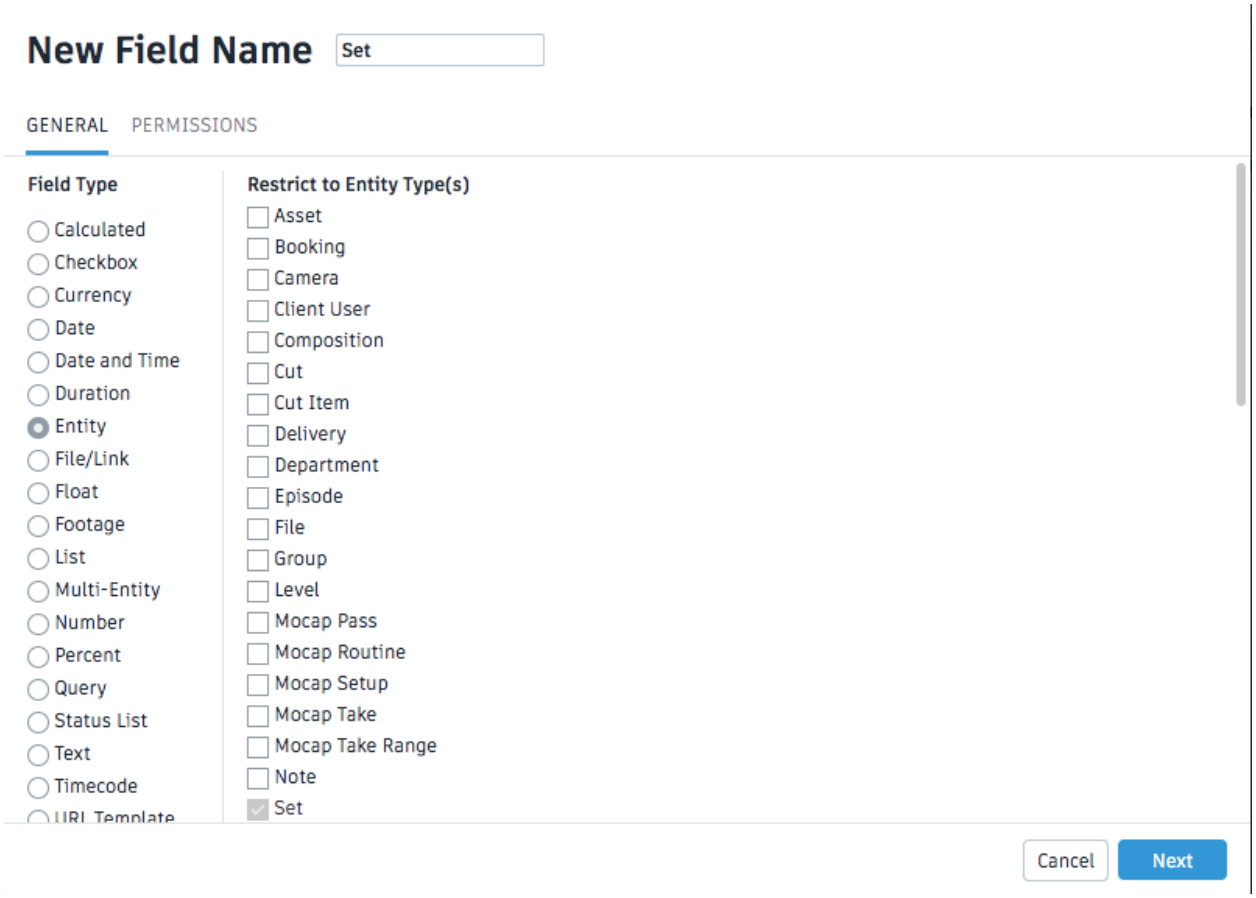
This action displays the asset field manager



Select **+ Add a new field**

Select the parameters for the new field.

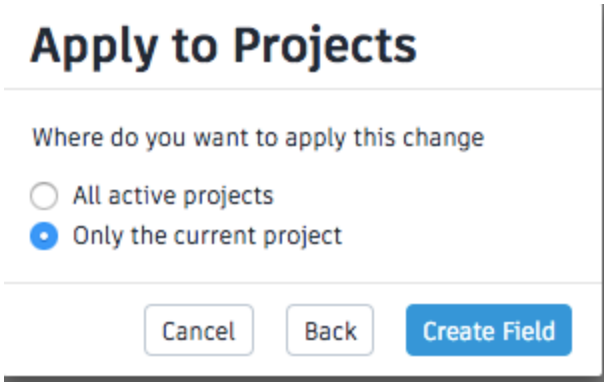
In **New Field Name** type Set. In the **GENERAL** menu under **Field Type** select **Entity** and scroll down to **Restrict the type** to **Set**.

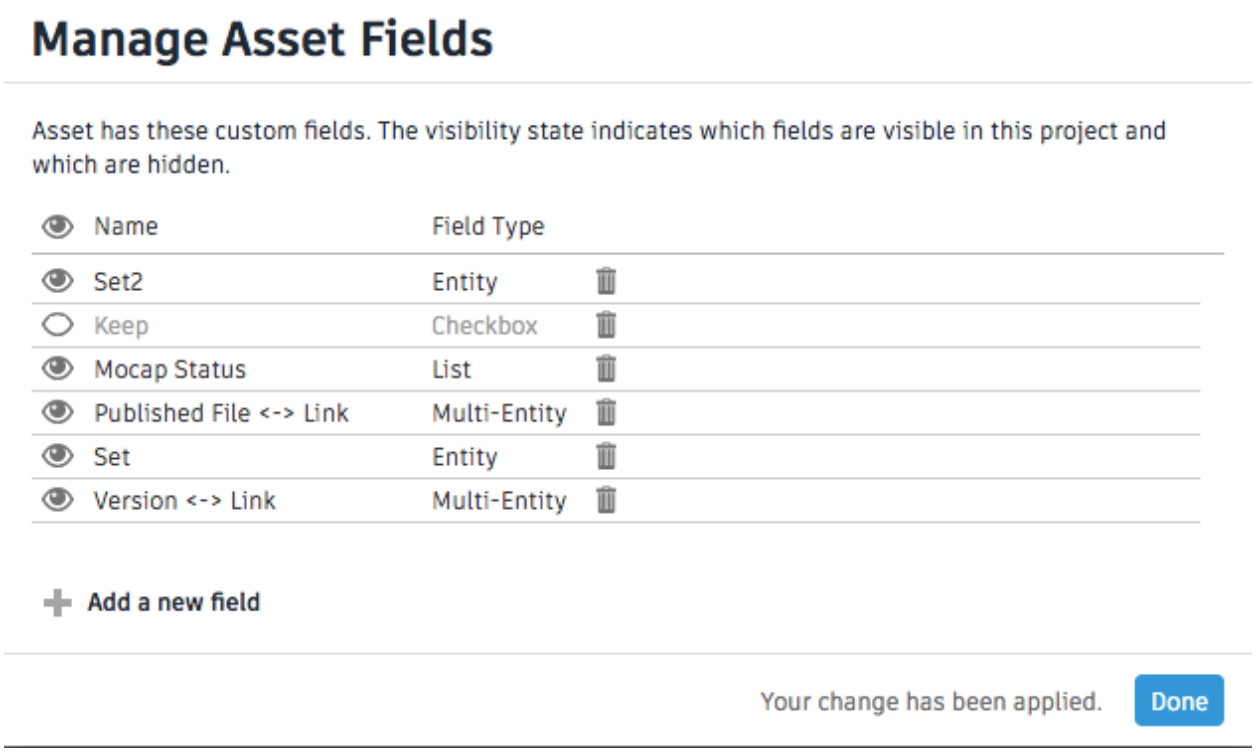


Select **Next**

For this guide, apply it to **Only the current project** and select **Create Field**.

Shotgun will configure the new field.

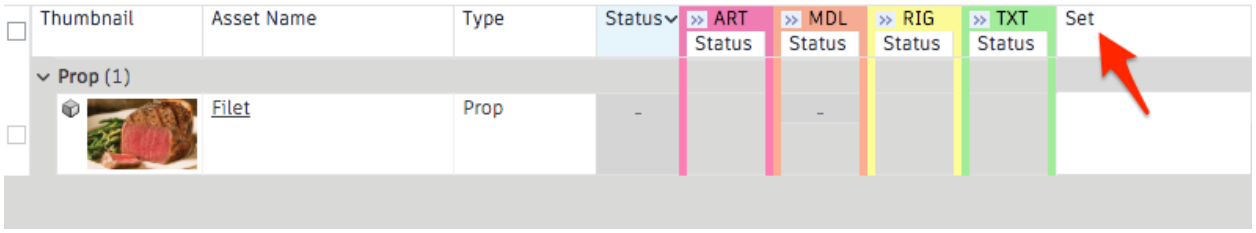




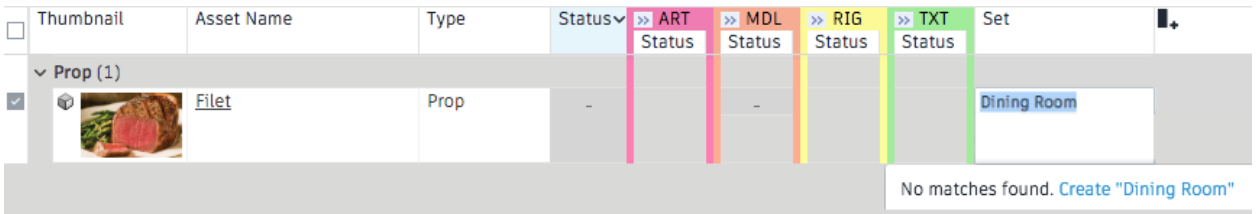
Your change has been applied and you can select **Done**.

Creating the **Dining Room** set entity

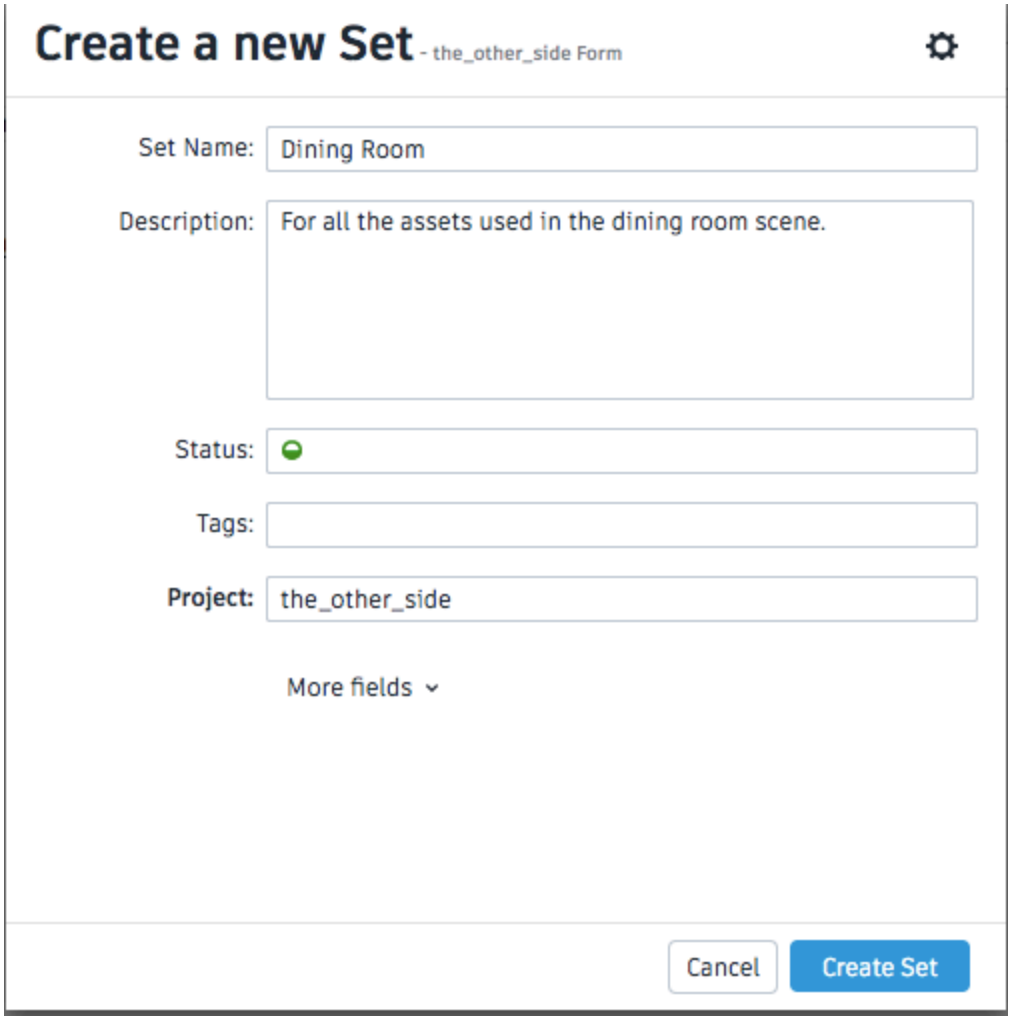
Step 5: Select the new **Set** field of an asset and start typing Dining Room.



A dialog box is displayed stating, **No matches found. Create “Dining Room”**

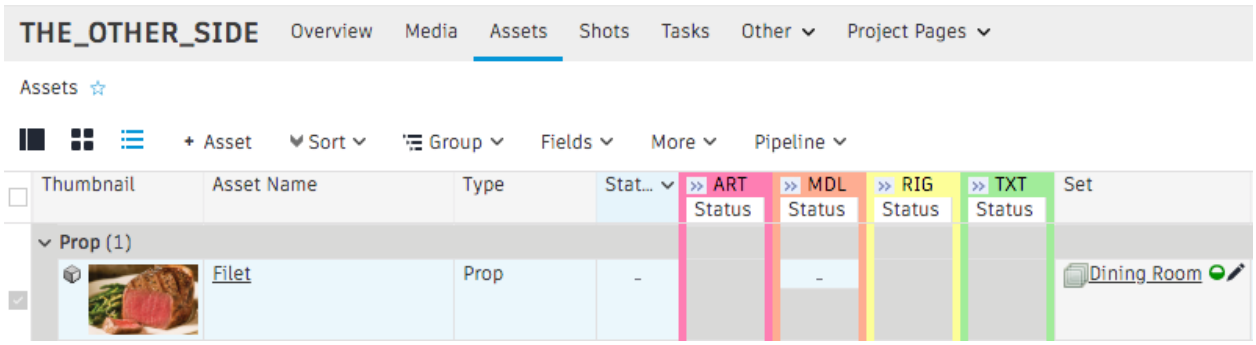


Select **Create “Dining Room”**

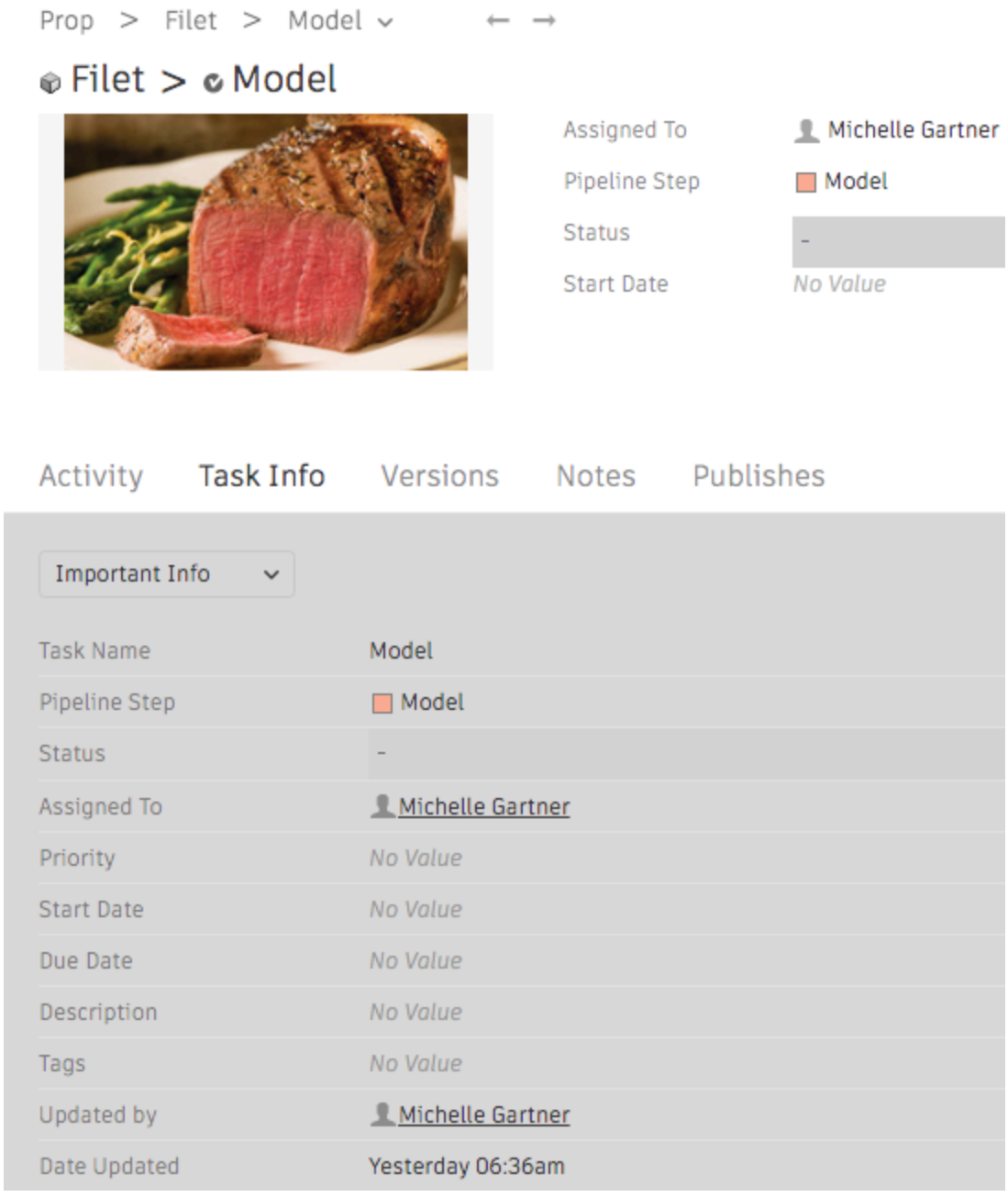


Select **Create Set**

Adding **Dining Room** in the Set field of an asset creates an [association](https://support.shotgunsoftware.com/hc/en-us/articles/115000010973-Linking-a-custom-entity) with the Dining Room set entity.



Step 6: Select the **Dining Room** link and add a task to create a Model asset called **filet** and assign it to yourself, so you can find it easily for testing purposes.



Setting up the schema

You added a field to the asset for a set entity and created a dining room set. When a file is created, Toolkit uses the associated asset fields and task fields to determine where the file will

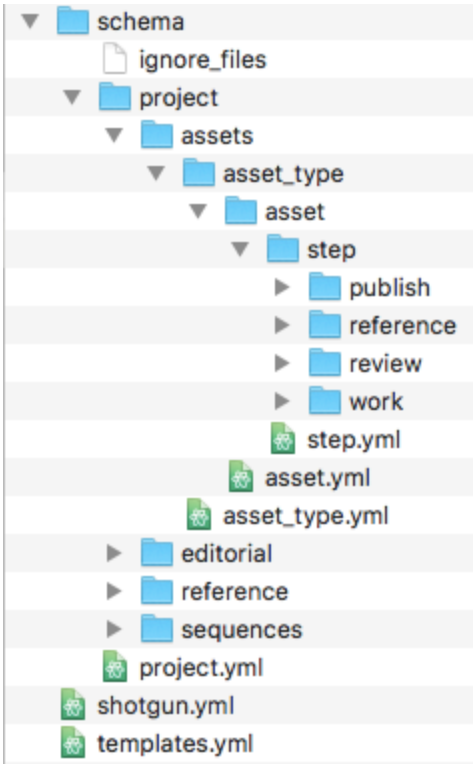
be saved in the filesystem. New folders are created and named automatically while an artist is working based on the schema structure detailed in a project configuration and files are named based on other variables you will set later in this guide. For example: The first time an artist creates a dining room asset for the new project `the_other_side`, all necessary folders are created for that specific asset as they are outlined in the schema and the file is named automatically created based on variables in the templates.

Let's take a look at how this is done

First we will need a cloned configuration to play with. If you remember from the third guide, [Adding an app](./installing_app.md), a cloned configuration was used as not to affect the production configuration. The cloning process created a copy of the configuration that you edited and tested then pushed to the live site. Here you can use that same clone and do the changes in the copied configuration, then push it live once the changes are made. If you don't have a [clone](https://support.shotgunsoftware.com/hc/en-us/articles/219033168-Configuration-staging-and-rollout#Cloning%20your%20Configuration) of your configuration you can refer to the [Adding an app](./installing_app.md#clone-the-pipeline-configuration-you-want-to-add-an-app-to) guide for how to create one.

Now it's time to create the folder structure you want Toolkit to dynamically generate as an artist steps through the production pipeline and files are created. This is done by using a skeletal structure that mirrors the filesystem structure to be dynamically generated. The folders in the skeleton contain variables for the folder names. These folder names are defined through queries on Shotgun data.

Step 7: Navigate to the copy of the project configuration for the project where a set entity type is to be used. Drill down to the schema folder, `<copy of project configuration>/config/core/schema` and open the project folder.



The current schema, with this skeleton:

`<project>/assets/<asset_type>/<asset>/<step>`

Supports dynamically creating this filesystem structure:

`the_other_side/assets/prop/filet/MDL`

Here's the structure you want to achieve on the filesystem for this project:

`the_other_side/assets/Dining-Room/Prop/filet/MDL`

To achieve this you would build the skeleton in the schema like this:

`<project>/assets/<CustomEntity01>/<asset_type>/<asset>/<step>`

The set entity is represented as `CustomEntity01`. In the first part of the guide you gave CustomEntity01 the display name Set in the Shotgun site. You didn't change the name of CustomEntity01 in the system, so when setting up the folder schema you will use `CustomEntity01`. What's happening under the hood is still relative to CustomEntity01.

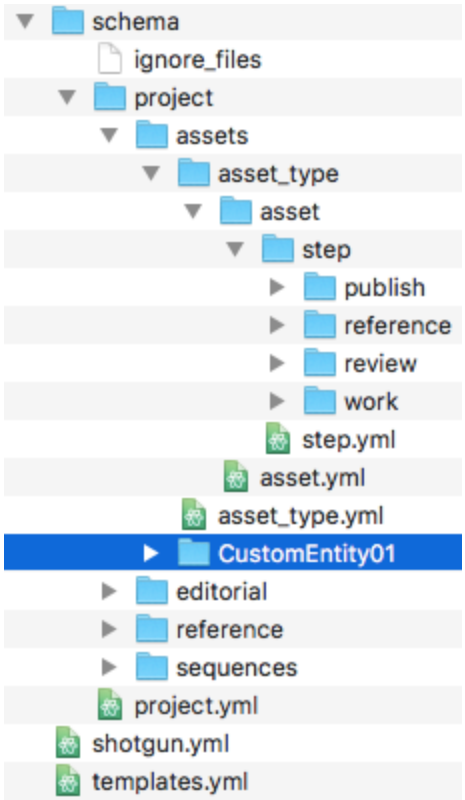
How the schema uses YAML files

There are static folders and dynamic folders, assets is the only static folder in this schema. The YAML files contain the variables and instruction sets for Toolkit to determine the names of the folders and which entities to create folders for. The YAML files are in the same folder as the skeletal folders they affect and each dynamic folder has a YAML file associated with it. The YAML file gives the instructions for determining the name of the folder and how folders relate to each other. So, how do they know they're connected? A Toolkit algorithm reads the YAML files based on the hierarchy of the schema folder structure. The algorithm determines the relationship of the information using the hierarchy of the skeletal folders and the YAML files. Toolkit determines the name of each folder by looking at the fields in the Shotgun data as specified by the YAML files.

Create the new folder and YAML file for the Set entity

The schema has a `project` folder that contains folders relative to the different entities Shotgun tracks. You are adding the new asset entity, CustomEntity01, to enable Shotgun to track the items in a Set. These items are assets, so you will edit the folders and YAML files under assets.

Step 8: Add a `CustomEntity01` folder inside the `project/assets/asset_type/asset` folder of your schema.

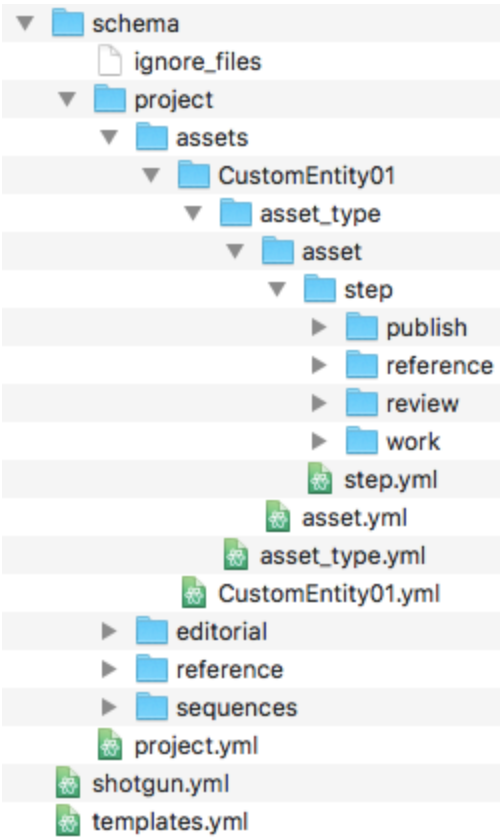


Step 9: Using a text editor or terminal window add a `CustomEntity01.yml` file next to the `CustomEntity01` folder, in the assets folder, and put the content below in that file.

```
# the type of dynamic content
type: "shotgun_entity"

# the shotgun field to use for the folder name
name: "code"

# the shotgun entity type to connect to
entity_type: "CustomEntity01"

# shotgun filters to apply when getting the list of items
# this should be a list of dicts, each dict containing
# three fields: path, relation and values
# (this is std shotgun API syntax)
# any values starting with $ are resolved into path objects
filters:
    - { "path": "project", "relation": "is", "values": [ "$project" ] }
```

The YAML file will give the instructions to Toolkit for what to name the `CustomEntity01` folder. This YAML file has several variables that Toolkit utilizes to [query](https://support.shotgunsoftware.com/hc/en-us/articles/219039868-Integrations-File-System-Reference#Shotgun%20Query%20Folders) Shotgun data.

Step 10: Move `asset_type` and `asset_type.yml` into the `CustomEntity01` folder



Final schema folder structure.

The `asset_type` is at a lower level then the `CustomEntity01` because it is the child of what will be part of a specified Set and holds the folders for the types of assets that could be in a Set like a dining room: prop, fx asset, character, etc...

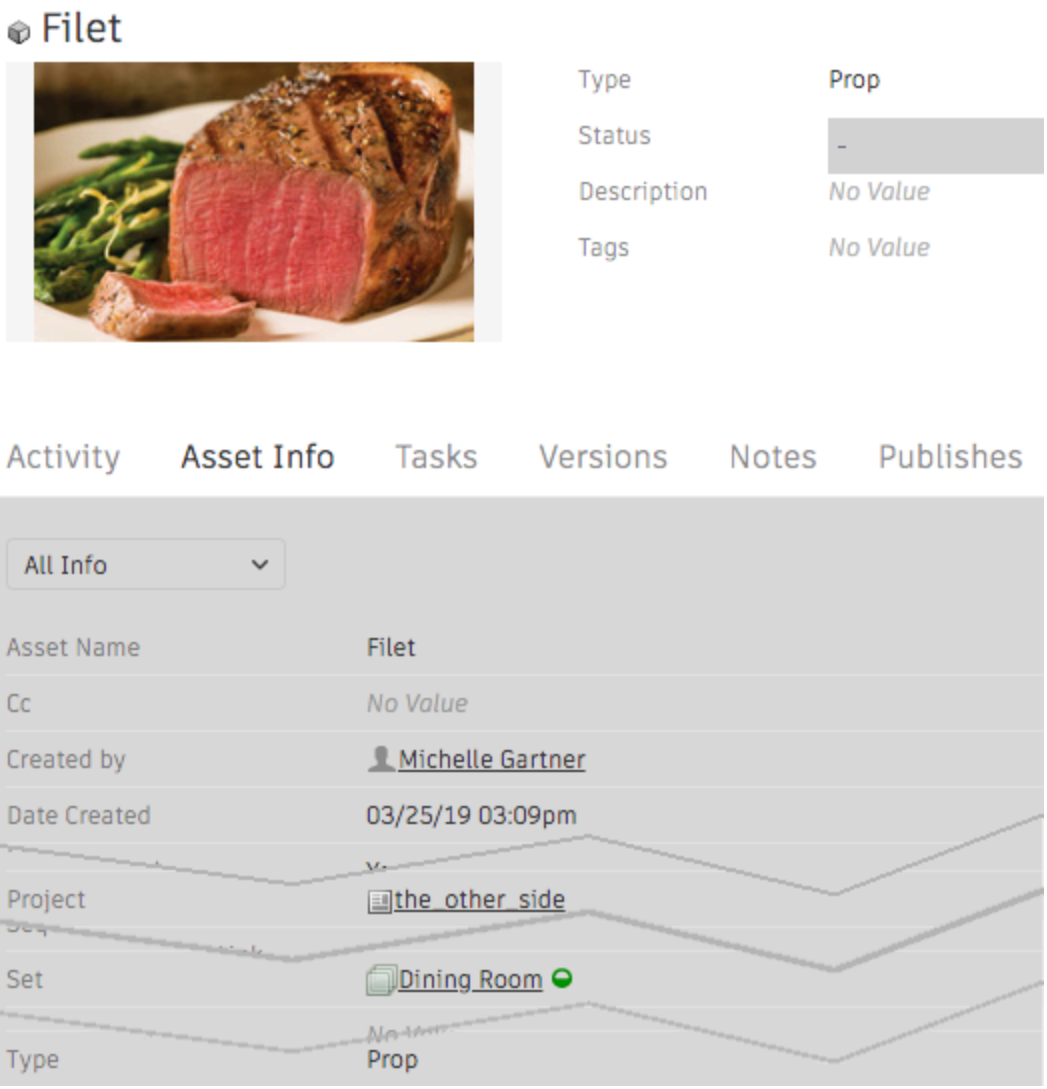
{% include info title="Note:" content="Creating and managing new asset folders and YAML files can be accomplished using a terminal window or text editor." %}

Edit `asset.yml` file

Editing the `asset.yml` file will allow the dynamic creation of the dining room folder when the first artist begins work on the first asset for the dining room, and uses the **+New File** function of the Workfiles app. You may be asking, “how does Toolkit know to generate the folder?”

The Toolkit algorithm that creates the folders knows to read the YAML files in the schema and query the Shotgun data fields based on the task the artist is performing. As described above, Toolkit builds the filesystem based on the results of the queries and the hierarchy of the schema, gathering information on the project, the task, the root directory, and any filters you’ve added or edited in the YAML files.

The queries in the algorithm search for the field values of the task the artist initiates. In this case, the task you created earlier is for a 3D Model asset. If an asset is being created, like the model of the filet, Toolkit queries the asset for the field values the YAML files specify. Below is the asset information for creating the filet.



Looking at the task and the asset Toolkit knows: There's a **Model Task** being created, the **Asset Name** is **Filet**, it's for the **Project the_other_side**, the **Set** is a **Dining Room**, and the **Type** of asset is a **Prop**. Toolkit gets the information about the project root directory through the data gathered from the `project.yml` file and uses the schema to understand the hierarchy of the data it gathered.

The **+ New File** action happens at a step in the pipeline, so the algorithm for building the folder structure knows to begin there in the schema, this will be the lowest level child of the

structure being built for saving the model of the filet. Here's how Toolkit figures out the type of data that folder stores and its name.

```# the type of dynamic content
type: "shotgun_step"

# the shotgun field to use for the folder name
name: "short_name"```

The Toolkit algorithm uses the information it gathered from the task and the asset to build the filesystem starting with the content of the folder where the shotgun_step was identified. Names of folders and folder hierarchy are discovered at various points in the process based on the information available. In this case, the **+ New File** triggers actions that create a folder for the new file and name it, then the parent and ancestor directories are created based on the variables in each YAML file as the algorithm climbs up the schema to the project root directory.

Below you will discover how the Toolkit algorithm uncover the parents and ancestors of the lowest folder and their names.

The artist tasked with creating the 3D model of the filet is the first artist to work on the project. The first time the artist uses the Workfiles app to save an asset, Toolkit knows it needs to build the filesystem to have somewhere to save the files associated with this asset. Based on the task information, Toolkit knows that the artist is in the model step in the pipeline and that the artist is in the asset_step environment. Follow along with what happens next…

Toolkit first creates a collection of all the YAML files needed to create the directories. A querying is performed that searches for data matching what's being created: the task being performed, entity_type, and shotgun_entity, etc... The data begins being utilized at the lowest point of the schema where data is needed:

Note: The display names for the data associated with task and asset are different than the code names used under the hood.

The data fields being used in this file structure translate to these identifiers under the hood. All the identifiers could have multiple fields to choose from based on the `type` of data that's identified.

**Pipeline Step** = `"shotgun_step"` this identifier has multiple data fields to pull from and is pulling from the `"short_name"` for this example.
**Asset Name** = `"code"`
**Type** = `"sg_asset_type"`
**Set** = "CustomEntity01"```
```

- Toolkit looks in the `step.yml` file at this line `type: "shotgun_step"` and queries the **Model** task data field, `"short_name"` which is different than what's being displayed in the **Pipeline Step**, the data associated with the name of the `shotgun_step` has multiple values. From that data it takes the `"short_name"` **MDL** and names the folder accordingly.

`step.yml' queries...

```type: "shotgun_step"

name: "short_name"```

- Toolkit also looks in the `asset.yml` file to determine the type of folder to create is a `"shotgun_entity"` and the Shotgun data it needs is the value of the Shotgun entity field **Asset Name** or under the hood, `"code"` and finds **Filet**. This data is data is identified as `entity_type` `"Asset"`.


`asset.yml` queries

```type: "shotgun_entity"

name: "code"

entity_type: "Asset"```

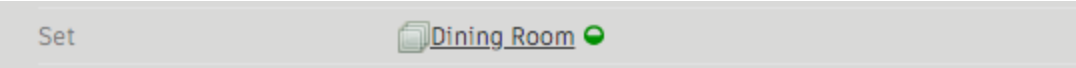
- The next stop is `asset_type.yml`. How Toolkit gets information from the `asset_type` skeleton is a little different. Here it's identified that this folder needs to connect to the entity type `"Asset"` and based on the hierarchy it's parent a of `asset`. It knows its value is from a `"shotgun_list_field"` and will get its name from the `"sg_asset_type"` field of the asset, **Prop**.


```# the type of dynamic content
type: "shotgun_list_field"
```

```
# the shotgun entity type to connect to
entity_type: "Asset"

# the shotgun field to use for the folder name
field_name: "sg_asset_type"```
```

- Toolkit jumps up another folder to find `CustomEntity01.yml` in the `asset_type` folder and learns this is an entity type and it's value for the name is based on the code field where it relates `"CustomEntity01"` to Dining-Room to use for the filesystem folder name.




```
```type: "shotgun_entity"

name: "code"

entity_type: "CustomEntity01"```
```

- Toolkit jumps up another folder into `assets` and there's no YAML file, so it knows to create a static folder called `assets`.

- Toolkit jumps up one more level to the `project.yml` file. Toolkit knows this is a folder that stores the project information because it's of type `"project"` and it will inherently know what project is being worked on when running the algorithms.

```
`type: "project"`
```

It finds the project root in the `roots.yml` file that's identified under the primary setting.

```
```# name of project root as defined in roots.yml
root_name: "primary"

primary:
  default: true
  linux_path: null
  mac_path: /Users/michelle/Documents/Shotgun/projects
  shotgun_storage_id: 4
  windows_path: null```
```

{% include info title="Note:" content="If the primary project path is changed the folders will need to be unregistered and registered to the new path. To unregister folders see Advanced topics below." %}

After Toolkit gathers all the information for what type of content each folder is going to store, it then needs to find the names of the unknown folders and create the hierarchy for this task based on the filters. This is what happens next…

- Starting from the `project` folder, the child is assets because it's static

- The `CustomEntity01.yml` filter identifies that this entity belongs to the `the_other_side` project folder and because of the hierarchy of the schema, it knows it belongs under the `assets` folder and the assets folder doesn't change because there's no YAML file. The static folders are just that… static and are created with the same name and position as they are in the schema.

  `CustomEntity01.yml` filters

  `- { "path": "project", "relation": "is", "values": [ "$project" ] }`

- The `asset_type.yml` doesn't have any filters. It already identified that the folder is called `Prop` and based on the filters in the `asset.yml` file it knows `Prop` is the child of `Dining-Room` and it's associated with the `the_other_side` project.

  `asset.yml` filters

  ``` - { "path": "project", "relation": "is", "values": [ "$project" ] }
  - { "path": "sg_asset_type", "relation": "is", "values": [ "$asset_type"] }
  - { "path": "sg_set", "relation": "is", "values": [ "$CustomEntity01" ] }```

- `asset.yml` filters identify the relationship of the folders above to the project `the_other_side`, the asset type **Prop** and the new custom entity **Dining Room** and puts the folders in order under the project, `the_other_side`.

Each time a **+New File** action is performed on a new task the Workfiles app checks to see if there are any folders that need to be created and determines where this new file is to be saved. All the applications that save or retrieve files use the same schema and build what they need, extending the filesystem when necessary.

## Test filesystem creation

Before going live with your new settings, run a test to make sure the folders are being created correctly.
  ```

Step11: Inside the copy of the project configuration root folder, run the `tank` command `folders` to create folders for an asset that already exists

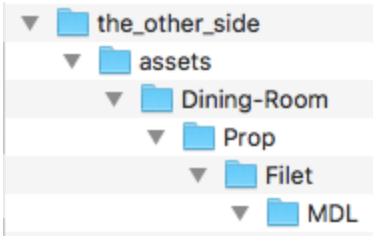
```
> cd /path/to/pipeline/config
```
```> ls
cache                           install
config                          tank
        tank.bat

>  ./tank Asset Filet folders

Welcome to the Shotgun Pipeline Toolkit!
For documentation, see https://support.shotgunsoftware.com
- You are running a project specific tank command. Only items that are part of
this project will be considered.
- Found Asset Filet (Project 'the_other_side')
- Running as user 'Michelle'
- Using configuration 'Primary Clone Config 2' and Core v0.18.159
- Setting the Context to Asset None.
- Running command folders...


----------------------------------------------------------------------
Command: Folders
----------------------------------------------------------------------

Creating folders, stand by...

The following items were processed:
 - /Users/michelle/Documents/Shotgun/projects/the_other_side
 - /Users/michelle/Documents/Shotgun/projects/the_other_side/editorial
 - /Users/michelle/Documents/Shotgun/projects/the_other_side/editorial/publish
 - /Users/michelle/Documents/Shotgun/projects/the_other_side/editorial/work
 - /Users/michelle/Documents/Shotgun/projects/the_other_side/reference
 - /Users/michelle/Documents/Shotgun/projects/the_other_side/reference/artwork
 - /Users/michelle/Documents/Shotgun/projects/the_other_side/reference/footage
 - /Users/michelle/Documents/Shotgun/projects/the_other_side/sequences
 - /Users/michelle/Documents/Shotgun/projects/the_other_side/assets
 -
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room
 -
```

/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/publish
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/publish/caches
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/publish/elements
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/publish/mari
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/reference
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/reference/artwork
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/reference/footage
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/review
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/work
-
/Users/michelle/Documents/Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/work/images

In total, 23 folders were processed.```

The final structure matches what was expected, and Toolkit is so smart that it even added a dash between Dining and Room enabling the label name to be used for the filesystem.

/the_other_side/assets/Dining-Room/Prop/Filet/MDL



{% include info title="Note:" content="You can also add other entities under the Dining-Room to create a folder for the `Place-Setting`: knife, fork, spoon, plate, etc..." %}

Toolkit templates for reading and writing files

Now that we've set up our folder structure, the next step is to edit the templates, so production files will be named appropriately and put in the correct folder when they're created. The templates control how files are named giving metadata to files like version and extension.

How Toolkit apps use a Toolkit template

You first created a way to associate an asset with the dining room by enabling CustomEntity01 to represent Sets, then associating that entity with the type of set, Dining Room, so you can manage the assets specific to the project. After establishing the relationship between the custom entity and Set, you created the file structure schema that allows Toolkit to create the folders necessary to save files based your filesystem. Now you're going to create a way to dynamically name files and allow Toolkit Apps to manage the files automatically. Toolkit Apps utilize the configuration file `templates.yml` when a file is written or accessed by an app during a pipeline process.

The first time an artist uses the Workfiles app **New File** action to create the first asset in a new project, the necessary folder structure is generated and when the Workfiles app **File Save** action is initiated, the file is named automatically. A template accessed through Toolkit's Workfiles app is used to name that file. Render apps like Nuke Write node and Houdini Mantra node also utilize the templates to name and save rendered files.

When files are accessed using the Workfiles **File Open** action, the template is used to find the appropriate file to load. The Publisher, Loader, and Nuke Studio Export apps also use the template to find and manage files. The artist doesn't have to worry about if the files are named correctly or if they're accessing the correct file, Toolkit manages them based on the template and the task being performed.

Any Toolkit app used in a specific integration can use the same project schema to build a file system, but templates are specific to each integration and the Toolkit app being used.

The templates are managed by the configuration file `/<project_config>/config/core/templates.yml`. In the first three guides you managed and created settings that were based on environments and were specific to software integrations. The schema and template settings are stored in the `config/core` folder and are not specific to an environment, but are specific to apps. One schema is used for the entire project and a template can be accessed from any environment that's appropriate. For example, `template_work` is the setting for the Workfiles app that specifies which template in `templates.yml` to use for work files. Depending on the environment and engine in which Workfiles is configured, you might use this configuration setting to point to the `maya_shot_work` template or the `houdini_asset_work` template from `templates.yml`.

Step 12: Navigate to `/<cloned project config>/config/core` and open `templates.yml`

This file is broken down into three sections:

[`Keys`](https://support.shotgunsoftware.com/hc/en-us/articles/219039868-Integrations-File-System-Reference#The%20Keys%20Section) - words that identify variables used in your templates. Key words like: Sequence, Shot, or Step are the variables that identify the entity type that is inserted in the string. Each Key has a required name and type with optional values ranging from a description, string, code etc... There are many ways a [`Key`](https://support.shotgunsoftware.com/hc/en-us/articles/219039868-Integrations-File-System-Reference#The%20Keys%20Section) word can be identified.

Below the `Keys` are `Paths`

[`Paths`](https://support.shotgunsoftware.com/hc/en-us/articles/219039868-Integrations-File-System-Reference#The%20Paths%20Section) - use the `Keys` to create a map to the folder where the file is to be saved or found. There are both static folders and dynamic folders that are used to build out the path to the file.

{% include info title="Important:" content="The paths in the templates need to match the folder structure in the schema." %}

[`Strings`](https://support.shotgunsoftware.com/hc/en-us/articles/219039868-Integrations-File-System-Reference#The%20Strings%20Section) - use keys to create templates for arbitrary text strings. For example, it may be the name field for versioning or the formatting of a file being published. The `Strings` section is similar to the `Paths` section, but while items in the paths section are validated and must correspond with actual paths on disk, strings can be used to store any text data that you want to refer to in your Toolkit workflows.

In the case of a string it can reference a variable from somewhere else in the configuration and finding the variable through the string can be done by instructing Toolkit to access other places in the configuration.

Add template key for the Set entity

The first thing to do is add the Set entity key to let Toolkit know there's a new key being defined.

Step 13: In the template.yml file add `CustomEntity01` under keys:

```keys:
      CustomEntity01:
          type: str```

### Adding the variable

Every individual integration has its own templates for each app that manages files. Since we're modifying our Maya workflow, add the variable `{CustomEntity01}` to the template that will put the file in the correct folder using the `maya_asset_work` template. This will tell Toolkit to, "use the path with the `Dining-Room` folder to save assets associated with the Dining Room."

Saving asset work files is triggered in the asset_step environment using the Workfiles app. You can find the template that the asset_step environment uses by following the includes from the `tk-maya.yml file`. This is the file that contains the starting point for the map of where the Toolkit settings live for Maya.

{% include info title="Note:" content="If other apps in other integrations are saving model assets for the dining room then these apps will also need a template specific to that integration" %}

### Find the template the Maya asset_step environment uses for the Workfiles2 app

**Step 14 In the `<copy of project configuration>/config/env/asset_step` look under the `engines:` for the `tk_maya:` `"@settings.tk-maya.asset_step"` and look in the `includes:` to find `./includes/settings/tk-maya.yml`.

Find the start of the roadmap that will take you to the settings Toolkit uses when the Workfiles app is triggered from the asset_step environment in Maya.

**Step 15: **Open `tk-maya.yml` and look under the `settings.tk-maya.asset_step` to find `tk-multi-workfiles2`.
```

The template settings are nested under `settings.tk-multi-workfiles2.maya.asset_step` in `tk-multi-workfiles2.yml`

{% include info title="Note:" content="In [Editing an app setting](./editing_app_setting.md) we learned a bit about how a Default Configuration is structured. To reiterate, the setting above is a good example of the structure and gives you a window into how the configuration can be used. `tk` = Toolkit, `workfiles2` = the workfiles app, `maya` is the integration, and `asset_step` is the environment." %}

Step 16: Open `tk-multi-workfiles2` and search for `settings.tk-multi-workfiles2.maya.asset_step`

Below this are the template settings for Workfiles. The name of the `template_work` template that's used for managing assets in Maya is what you are ultimately looking for. The setting displayed next to `template_work` is `maya_asset_work`.

Look in the `templates.yml` file for `maya_asset_work`

Step 17: In **template.yml** search for **maya_asset_work**

```
  maya_asset_work:
      definition: '@asset_root/work/maya/{name}.v{version}.{maya_extension}'
```

The `definition:` for `maya_asset_work`: begins with `@asset_root`. The `@` symbol signifies that the setting is nested somewhere. In this case it's nested inside the `template.yml` file.

{% include info title="Note:" content="The template settings don't nest settings the same way the environment settings do. If there's an include, `@`, in the `templates.yml` file it symbolizes there's an included setting within the `templates.yml` file." %}

Since each integration and each app within those integrations use their own settings, you can imagine that this same path can be used in many different places inside the `template.yml` file. The configuration is setup so it can refer to one master path and reuse that throughout the `template.yml` file. You won't have to change each instance of the path generation settings when you can reference the master path.

This is done with the `asset_root` setting in the `path:` section of `template.yml`, ultimately creating the path variables that can be referenced by settings in the `template.yml` file.

Edit the template to match the path in the filesystem schema

Step 18: Search for `asset_root` at the top of the `paths` section of `templates.yml`, this is where the master path settings are referenced. Add `CustomEntity01` to the `asset_root` path and match the schema you created for the project.

Project schema

`<project>/assets/<CustomEntity01>/<asset_type>/<asset>/<step>`

Master path

`asset_root: assets/{CustomEntity01}/{sg_asset_type}/{Asset}/{Step}`

Set up the path for naming the file and adding the metadata

Adding `CustomEntity01` in the setting for `maya_asset_work` adds the entity name to the file.

Step 19: Search for the `maya_asset_work` template and add the variable `CustomEntity01` to the string. This setting is for any asset that uses Workfiles in Maya to save files associated with that specific asset.

`maya_asset_work:
    definition: '@asset_root/work/maya/{CustomEntity01}_{name}.v{version}.{maya_extension}'`

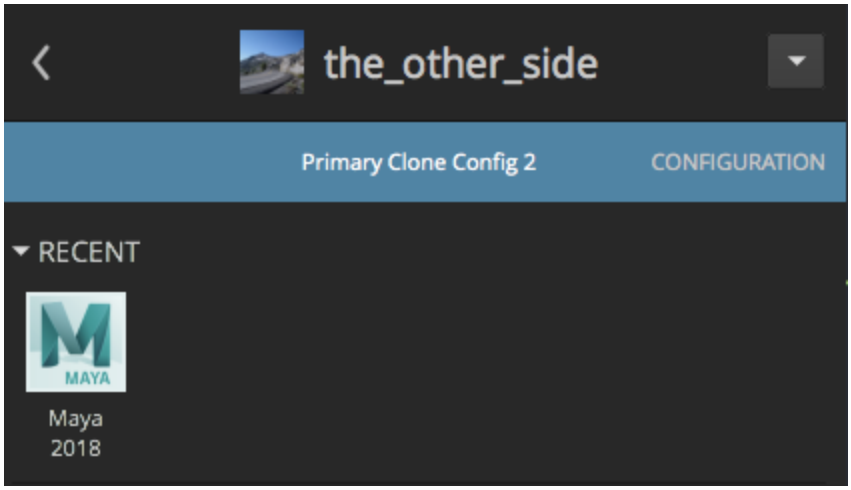
This action allows you to use the Dining-Room entity proper name in the file name. The result would be something like `Dining-Room_Filet.v1.mb`.

{% include info title="Note:" content="The files don’t save metadata for the project or entity they are associated with. The only way to determine the association is by the folder structure the file is saved in or by adding your own file name structure using metadata from Shotgun." %}

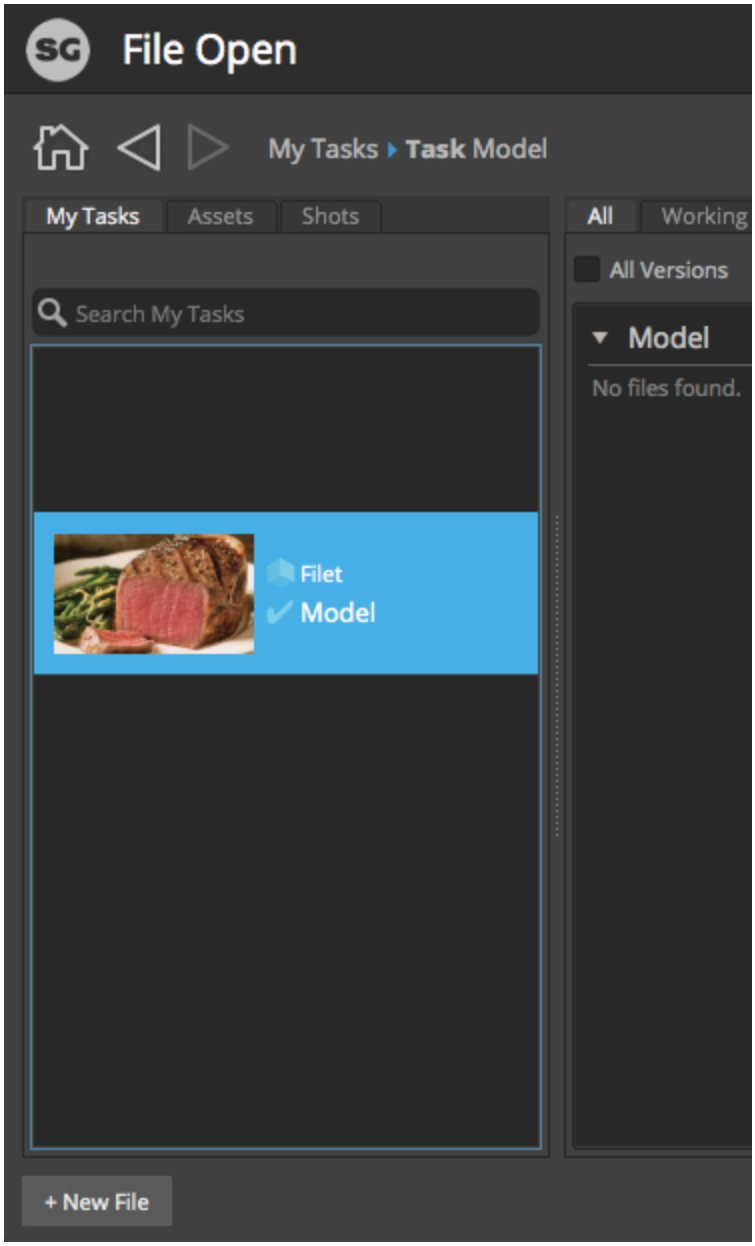
You edited settings for the Maya Workfiles app to use when naming files. A new key word was added for the new entity type, the key word was used as one of the variables for file naming, and a master path was created as a reference. The file name variables were also edited, adding the new entity as part of the name.

Test it

Step 20: Open Maya from Shotgun Desktop.

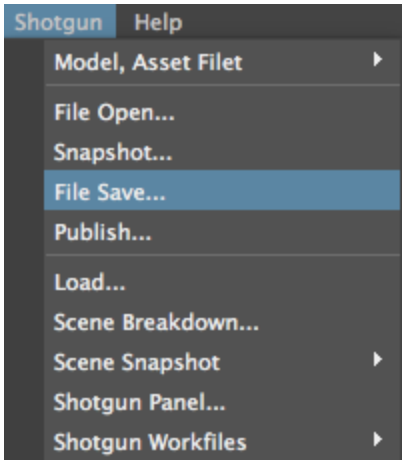


In the **Shotgun menu **>** File Open** dialog, select a task on an asset for which you've specified a set in Shotgun.

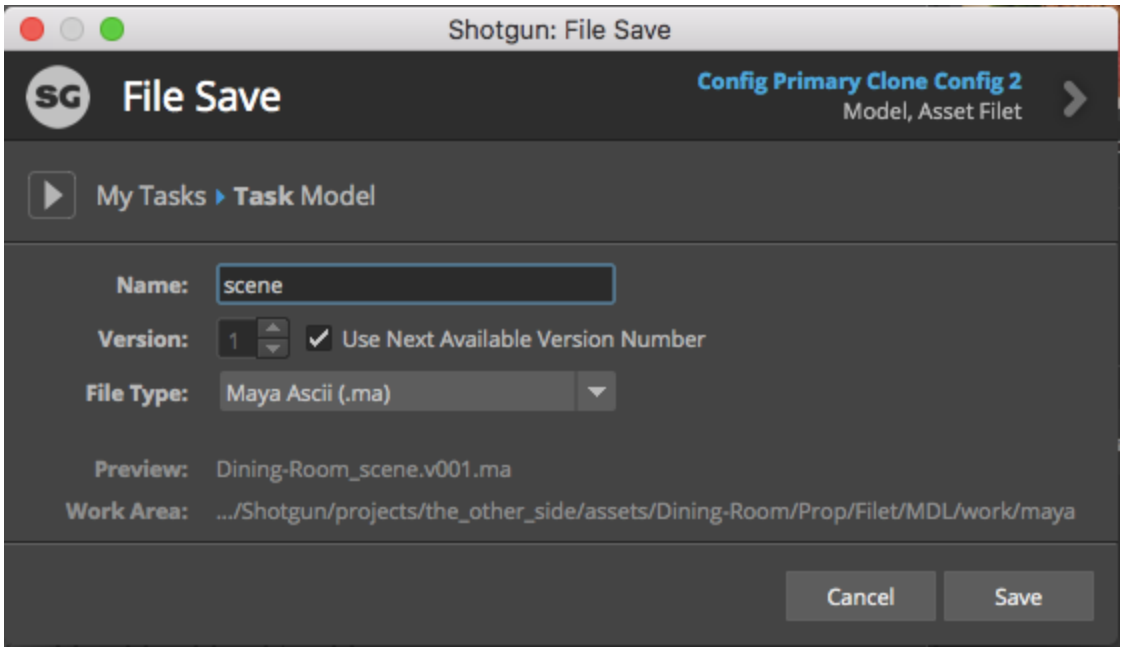


Select **+New File**.

You can create a simple 3D object or just save the file using the **Shotgun Menu > Save File**.



Success!



Notice the **File Save** dialog box is displaying **Preview: Dining-Room_scene.v001.ma** using the variables that were set in the `preview:`

The **Work Area**: is displaying **.../Shotgun/projects/the_other_side/assets/Dining-Room/Prop/Filet/MDL/work/maya** as the path for where Workfiles is saving the file.

Adding folders to an existing filesystem

If there are assets other than props for the dining room: fx asset, character, vehicle etc…, Toolkit creates only the folders that don't exist. For example: if no tasks have been initiated that are associated with an fx asset entity the first time the artist selects **+New File** on a task to create an fx asset, Toolkit looks at the data, compares it to what the current filesystem structure is, and if the folder structure for the fx asset entity doesn't exist, it utilizes the schema to create only the folders needed to save fx assets. It creates the new folders based on what's needed for the ancestors and descendants of fx asset and caches the new path.

Advanced topics

Unregistering folders and creating a new path
https://www.google.com/url?q=https://support.shotgunsoftware.com/hc/en-us/articles/219040418-What-is-the-Path-Cache-What-are-Filesystem-Locations-&sa=D&ust=1555705593472000&usg=AFQjCNGEwrMeZmGRp9QLev0l7Jqvvgo55A

Creating the path for the assets

Toolkit only builds the necessary structure for the task that's being performed then saves the path for that task in the cache, so any Toolkit app being utilized for that task can access the path when needed and doesn't have to run the queries on folders that are already created.

Workfiles does not send information to Shotgun

User Sandboxes
https://support.shotgunsoftware.com/hc/en-us/articles/219033088-Your-Work-Files#User%20sandboxes

Video demo of how to take over integrations and create a custom file structure

https://www.youtube.com/watch?v=7qZfy7KXXX0&t=1961s