OpenStudio API Training Exercises

Contents

[Purpose and Scope 2](#_Toc362680014)

[1.0 Exercises 3](#_Toc362680015)

[1.1 Exercise 1 – Create and Save a New Model 3](#_Toc362680016)

[1.2 Exercise 2 – Create a Floorplan 4](#_Toc362680017)

[1.3 Exercise 3 – Create Construction Elements 8](#_Toc362680018)

[1.4 Exercise 4 – Create Loads and Outdoor Air Requirements 15](#_Toc362680019)

[1.5 Exercise 5 – Create Operation Schedules 18](#_Toc362680020)

[1.6 Exercise 6 – Create Thermal Zones 25](#_Toc362680021)

[1.7 Exercise 7 – Run Simulation From OpenStudio Application 26](#_Toc362680022)

[1.8 Exercise 8 – Run Simulation Programmatically 27](#_Toc362680023)

[1.9 Exercise 9 – Add HVAC Systems 29](#_Toc362680024)

[1.10 Exercise 10 – Remove Existing HVAC Systems and Add New VAV System 30](#_Toc362680025)

[1.11 Exercise 11 – PAT Create New Measure and Run Simulations 32](#_Toc362680026)

[1.12 Exercise 12 – Access Predefined Results Data 39](#_Toc362680027)

[1.13 Exercise 13 – Make a Custom Results Query 40](#_Toc362680028)

[1.14 Exercise 14 – Create and Read a Report Variable 41](#_Toc362680029)

[1.15 Exercise 15 – Browse Existing BCL Content 42](#_Toc362680030)

[1.15.1 Components 42](#_Toc362680031)

[1.15.2 Measures 46](#_Toc362680032)

[1.16 Exercise 16 – Upload New BCL Content 48](#_Toc362680033)

[1.16.1 Content Control and Groups 48](#_Toc362680034)

[1.16.2 Components 49](#_Toc362680035)

[1.16.3 Measures 52](#_Toc362680036)

Purpose and Scope

This document is a training tool that will explain how to programmatically access the OpenStudio software interface to create and modify building energy models using the Ruby programming language. The format consists of a series of exercises beginning with creating and saving a new model, then reading results data and making modifications using OpenStudio Measures. This document is not a complete reference to the OpenStudio software interface, but rather it is a tutorial to outline a basic workflow. In order to complete these exercises you must have a copy of OpenStudio installed that is at least as recent as version 1.0. Each exercise is designed to build on the previous one.

# Exercises

## Exercise 1 – Create and Save a New Model

The goal of this exercise is self-explanatory. Create the most basic possible model and save it in the form of a .osm file.

1. Open a text editor, preferably one with syntax highlighting for Ruby. Notepad++ is a free editor that works well for Windows, <http://notepad-plus-plus.org/>.
2. Open Path/To/OSTraining/EX1/ex1.rb and look at it.
3. Open a command prompt (Windows) or terminal (Mac)
   1. Start->cmd->ENTER (Windows) or Applications Folder > Terminal (Mac)
4. Inside the command prompt, change directories to EX1.

C:\users\aparker> cd C:\OSTraining\EX1 (Press ENTER)

1. Now, run the ruby script

C:\OSTraining> ruby ex1.rb (Press ENTER)

1. You should get output that looks like the following:

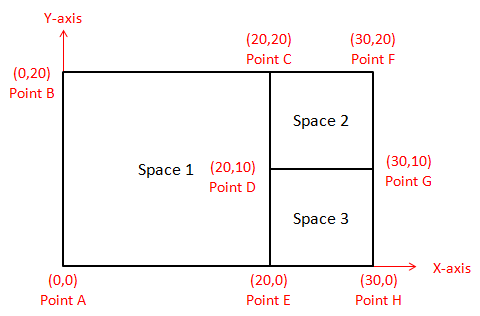
C:\OSTraining> ruby ex1.rb

file saved to C:/OSTraining/ex1.osm

1. Open the folder C:/OSTraining/EX1. You should see a file called ex1.osm.

## Exercise 2 – Create a Floorplan

The goal of this exercise is to make a basic building. The floorplan of the building looks like this:



1. Open Path/To/OSTraining/EX2/ex2.rb.
2. Make a story, set the floor-to-floor height, and make the points

#set the space height (meters)

height **=** 5

#Create a new story for all the spaces

story **=** OpenStudio**::**Model**::**BuildingStory**.**new**(**model**)**

story**.**setNominalFloortoFloorHeight**(**height**)**

story**.**setName**(**"First Floor}"**)**

#create the points (all dimensions in meters)

point\_a **=** OpenStudio**::**Point3d**.**new**(**0**,**0**,**height**)**

point\_b **=** OpenStudio**::**Point3d**.**new**(**0**,**20**,**height**)**

point\_c **=** OpenStudio**::**Point3d**.**new**(**20**,**20**,**height**)**

point\_d **=** OpenStudio**::**Point3d**.**new**(**20**,**10**,**height**)**

point\_e **=** OpenStudio**::**Point3d**.**new**(**20**,**0**,**height**)**

point\_f **=** OpenStudio**::**Point3d**.**new**(**30**,**20**,**height**)**

point\_g **=** OpenStudio**::**Point3d**.**new**(**30**,**10**,**height**)**

point\_h **=** OpenStudio**::**Point3d**.**new**(**30**,**0**,**height**)**

1. Make a polygon for space 1, then create a Space from the polygon. It is important to note that the points are added to the vector in clockwise (when viewed from above) fashion. Putting the points in the opposite direction will result in upside-down surfaces.

#make the polygon for space 1

space\_1\_polygon **=** OpenStudio**::**Point3dVector**.**new

#add the points to the polygon in a clockwise direction

space\_1\_polygon **<<** point\_a

space\_1\_polygon **<<** point\_b

space\_1\_polygon **<<** point\_c

space\_1\_polygon **<<** point\_d

space\_1\_polygon **<<** point\_e

#create the space from the polygon

opt\_space\_1 **=** OpenStudio**::**Model**::**Space**::**fromFloorPrint**(**space\_1\_polygon**,** height**,** model**)**

#if the space now exists, set name and story

**if** **not** opt\_space\_1**.**empty?

space\_1 **=** opt\_space\_1**.**get

space\_1**.**setName**(**"Space 1"**)**

space\_1**.**setBuildingStory**(**first\_story**)**

**end**

1. Repeat for Space 2 and Space 3

#make the polygon for space 2

space\_2\_polygon **=** OpenStudio**::**Point3dVector**.**new

#add the points to the polygon in a clockwise direction

space\_2\_polygon **<<** point\_d

space\_2\_polygon **<<** point\_c

space\_2\_polygon **<<** point\_f

space\_2\_polygon **<<** point\_g

#create the space from the polygon

opt\_space\_2 **=** OpenStudio**::**Model**::**Space**::**fromFloorPrint**(**space\_2\_polygon**,** height**,** model**)**

#if the space now exists, set name and story

**if** **not** opt\_space\_2**.**empty?

space\_2 **=** opt\_space\_2**.**get

space\_2**.**setName**(**"Space 2"**)**

space\_2**.**setBuildingStory**(**first\_story**)**

**end**

#make the polygon for space 3

space\_3\_polygon **=** OpenStudio**::**Point3dVector**.**new

#add the points to the polygon in a clockwise direction

space\_3\_polygon **<<** point\_e

space\_3\_polygon **<<** point\_d

space\_3\_polygon **<<** point\_g

space\_3\_polygon **<<** point\_h

#create the space from the polygon

opt\_space\_3 **=** OpenStudio**::**Model**::**Space**::**fromFloorPrint**(**space\_3\_polygon**,** height**,** model**)**

#if the space now exists, set name and story

**if** **not** opt\_space\_3**.**empty?

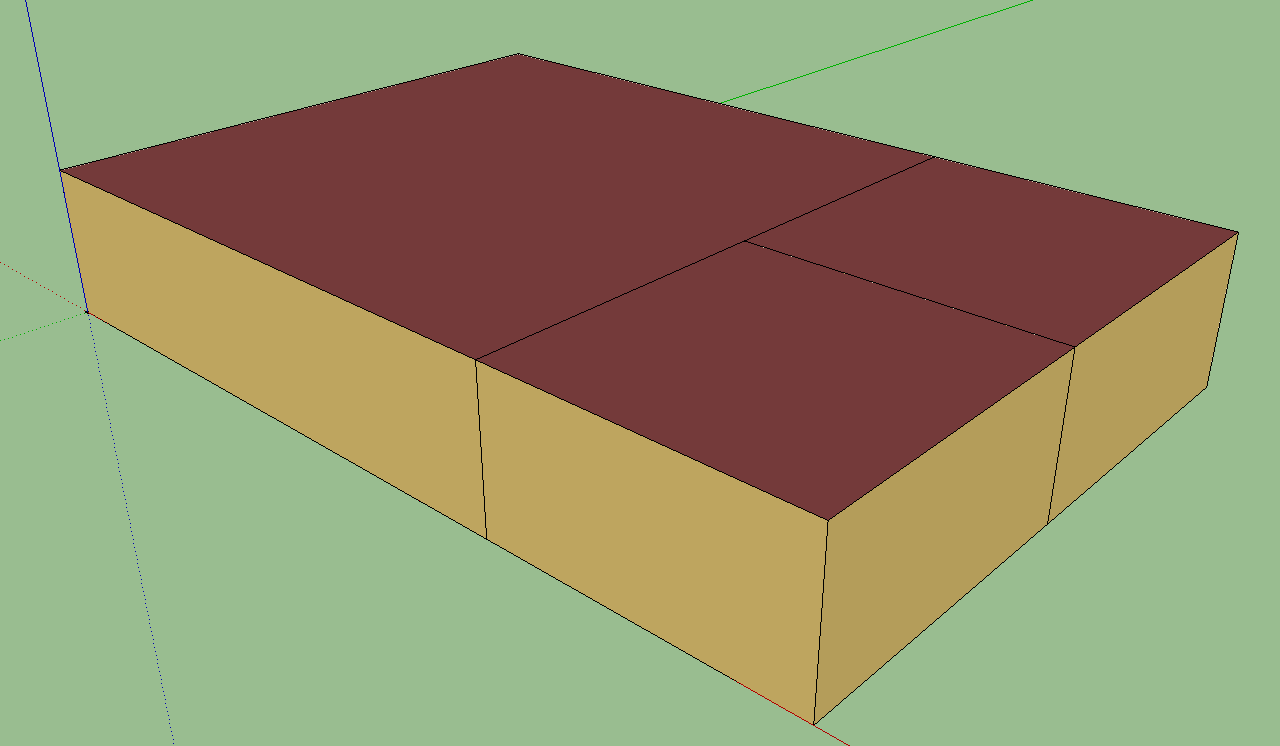
space\_3 **=** opt\_space\_3**.**get

space\_3**.**setName**(**"Space 3"**)**

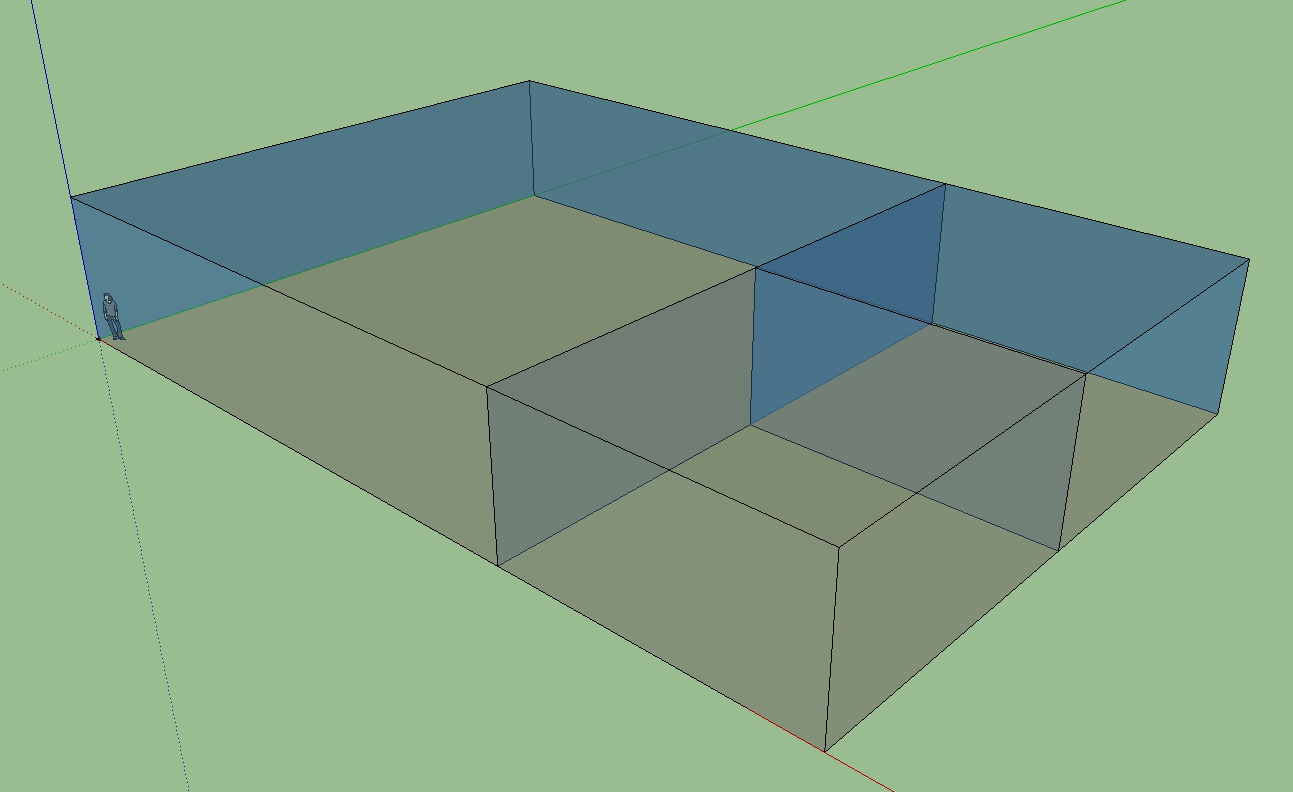
space\_3**.**setBuildingStory**(**first\_story**)**

**end**

1. Open the model in the SketchUp plugin. It should look like this:



If you go into “Render by Boundary Condition” (dark blue cube) and turn on x-ray mode (next to calendar button), you’ll see that all the internal walls are still blue, which means that they have an outdoor boundary condition.



To fix this, add the the following to your file:

#put all of the spaces in the model into a vector

spaces **=** OpenStudio**::**Model**::**SpaceVector**.**new

model**.**getSpaces**.**each **do** **|**space**|**

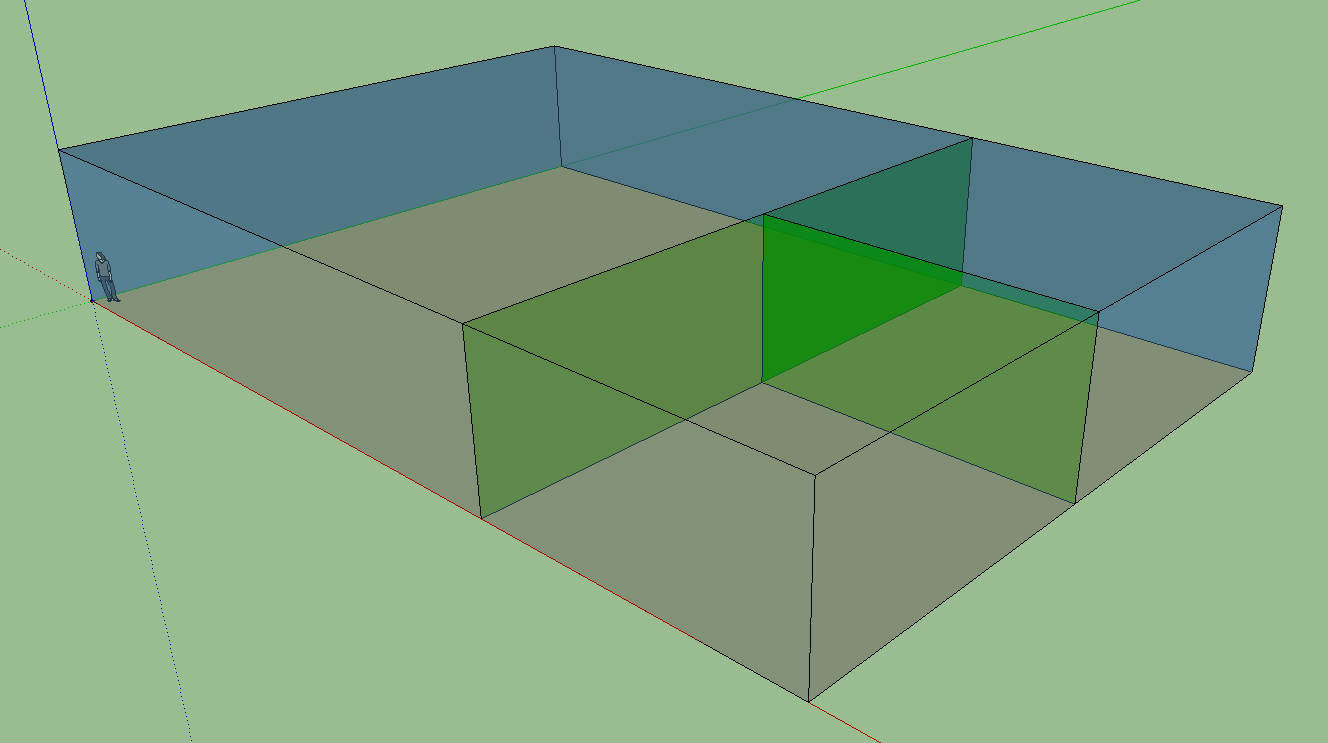
spaces **<<** space

**end**

#match surfaces for each space in the vector

OpenStudio**::**Model**.**matchSurfaces**(**spaces**)**

1. Run the script again. Reopen the model in SketchUp. Looking at it now, notice that the internal surfaces are green, which means inter-zone boundary condition.



## Exercise 3 – Create Construction Elements

At the moment, the building has surfaces (walls, roofs, floors, etc) but they have no properties (thickness, R-value, etc). The goal of this exercise is to create some building constructions and assign them to the building surfaces. In the interest of time, everything in the building will be very simple.

1. Open Path/To/OSTraining/EX3/ex3.rb.
2. Create materials. A material is the basic unit used to make constructions for the energy model. A material represents a particular material and a particular thickness. For example, 2in. insulation and 3in. insulation would be two separate materials.

Properties of the few materials we’ll be making. These properties were taken from the ASHRAE Handbook of Fundamentals.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Thickness**  **(mm)** | **Conductivity (W/m\*K)** | **Density**  **(kg/m^3)** | **Specific Heat**  **(J/kg\*K)** |
| Plywood - 5/8 in | 15.875 | 0.12 | 544.0 | 1210.0 |
| Expanded Polystyrene – 2in | 50.8 | 0.0352 | 24.0 | 1210.0 |
| Heavyweight Concrete – 6in | 152.4 | 1.95 | 2242.6 | 900.0 |

#make plywood and set properties

plywd **=** OpenStudio**::**Model**::**StandardOpaqueMaterial**.**new**(**model**)**

plywd**.**setName**(**"Plywood - 5/8in"**)**

plywd**.**setThickness**(**0.015875**)** #meters

plywd**.**setThermalConductivity**(**0.12**)** #W/m\*K

plywd**.**setDensity**(**544.0**)** #kg/m^3

plywd**.**setSpecificHeat**(**1210.0**)** #J/kg\*K

#make expanded polystyrene and set properties

eps **=** OpenStudio**::**Model**::**StandardOpaqueMaterial**.**new**(**model**)**

eps**.**setName**(**"Expanded Polystyrene - 2in"**)**

eps**.**setThickness**(**0.0508**)** #meters

eps**.**setThermalConductivity**(**0.0352**)** #W/m\*K

eps**.**setDensity**(**24.0**)** #kg/m^3

eps**.**setSpecificHeat**(**1210.0**)** #J/kg\*K

#make plywood and set properties

concrete **=** OpenStudio**::**Model**::**StandardOpaqueMaterial**.**new**(**model**)**

concrete**.**setName**(**"Heavyweight Concrete 6in"**)**

concrete**.**setThickness**(**0.1524**)** #meters

concrete**.**setThermalConductivity**(**1.95**)** #W/m\*K

concrete**.**setDensity**(**2242.6**)** #kg/m^3

concrete**.**setSpecificHeat**(**900.0**)** #J/kg\*K

1. Create a simple glazing material.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **U-Factor**  **(W/m^2\*K)** | **SHGC** | **VLT** |
| Glass | 0.1 | 0.3 | 0.7 |

#make a simple glazing material and set properties

glass **=** OpenStudio**::**Model**::**SimpleGlazing**.**new**(**model**)**

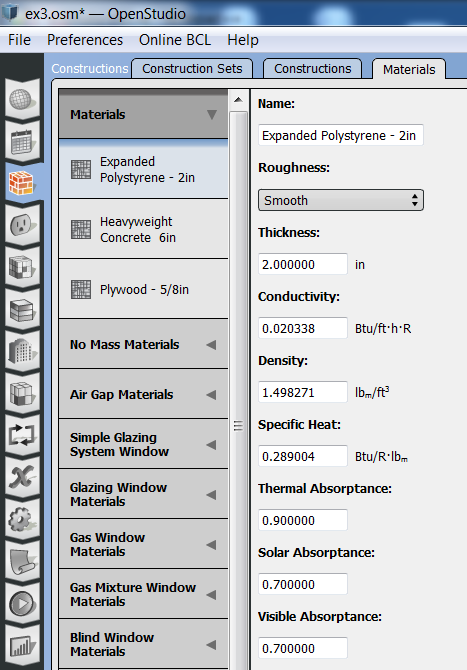
glass**.**setName**(**"Glass"**)**

glass**.**setUFactor**(**0.1**)** #W/m^2\*K

glass**.**setSolarHeatGainCoefficient**(**0.3**)**

glass**.**setVisibleTransmittance**(**0.7**)**

1. Run the script and open ex3.osm in the OpenStudio App. On the Constructions tab, under the Materials sub-tab, you should see the materials in the model.



1. Make constructions using these materials:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Outermost Material** |  |  | **Innermost Material** |
| Ext Wall Construction | Plywood - 5/8 in | Expanded Polystyrene – 2in | Heavyweight Concrete – 6in |  |
| Int Wall Construction | Plywood - 5/8 in | Plywood - 5/8 in |  |  |
| Roof Construction | Plywood - 5/8 in | Expanded Polystyrene – 2in | Expanded Polystyrene – 2in | Plywood - 5/8 in |
| Window Construction | Glass |  |  |  |

#make the exterior wall construction

ext\_wall\_const **=** OpenStudio**::**Model**::**Construction**.**new**(**model**)**

ext\_wall\_const**.**setName**(**"Ext Wall Construction"**)**

ext\_wall\_const**.**insertLayer**(**0**,**plywd**)**

ext\_wall\_const**.**insertLayer**(**1**,**eps**)**

ext\_wall\_const**.**insertLayer**(**2**,**concrete**)**

#make the interior wall construction

int\_wall\_const **=** OpenStudio**::**Model**::**Construction**.**new**(**model**)**

int\_wall\_const**.**setName**(**"Int Wall Construction"**)**

int\_wall\_const**.**insertLayer**(**0**,**plywd**)**

int\_wall\_const**.**insertLayer**(**1**,**plywd**)**

#make the roof construction

roof\_const **=** OpenStudio**::**Model**::**Construction**.**new**(**model**)**

roof\_const**.**setName**(**"Roof Construction"**)**

roof\_const**.**insertLayer**(**0**,**plywd**)**

roof\_const**.**insertLayer**(**1**,**eps**)**

roof\_const**.**insertLayer**(**2**,**eps**)**

roof\_const**.**insertLayer**(**3**,**plywd**)**

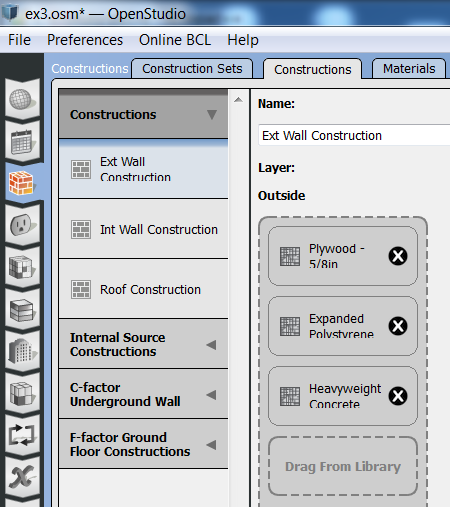
#make the window construction

window\_const **=** OpenStudio**::**Model**::**Construction**.**new**(**model**)**

window\_const**.**setName**(**"Window Construction"**)**

window\_const**.**insertLayer**(**0**,**glass**)**

1. Run the script and open ex3.osm in the OpenStudio App. On the Constructions tab, under the Constructions sub-tab, you should see the constructions in the model.



1. Make default surface constructions, default subsurface constructions, and a default construction set and assign it to the building.

#make a default surface construction for the exterior and ground

default\_ext\_surface\_consts **=** OpenStudio**::**Model**::**DefaultSurfaceConstructions**.**new**(**model**)**

default\_ext\_surface\_consts**.**setFloorConstruction**(**ext\_wall\_const**)**

default\_ext\_surface\_consts**.**setWallConstruction**(**ext\_wall\_const**)**

default\_ext\_surface\_consts**.**setRoofCeilingConstruction**(**roof\_const**)**

#make a default surface construction for the interior

default\_int\_surface\_consts **=** OpenStudio**::**Model**::**DefaultSurfaceConstructions**.**new**(**model**)**

default\_int\_surface\_consts**.**setFloorConstruction**(**int\_wall\_const**)**

default\_int\_surface\_consts**.**setWallConstruction**(**int\_wall\_const**)**

#make a default sub surface construction for interior and exterior

default\_subsurface\_consts **=** OpenStudio**::**Model**::**DefaultSubSurfaceConstructions**.**new**(**model**)**

default\_subsurface\_consts**.**setFixedWindowConstruction**(**window\_const**)**

default\_subsurface\_consts**.**setDoorConstruction**(**window\_const**)**

#make a default construction set

default\_const\_set **=** OpenStudio**::**Model**::**DefaultConstructionSet**.**new**(**model**)**

default\_const\_set**.**setDefaultExteriorSurfaceConstructions**(**default\_ext\_surface\_consts**)**

default\_const\_set**.**setDefaultExteriorSubSurfaceConstructions**(**default\_subsurface\_consts**)**

default\_const\_set**.**setDefaultInteriorSurfaceConstructions**(**default\_int\_surface\_consts**)**

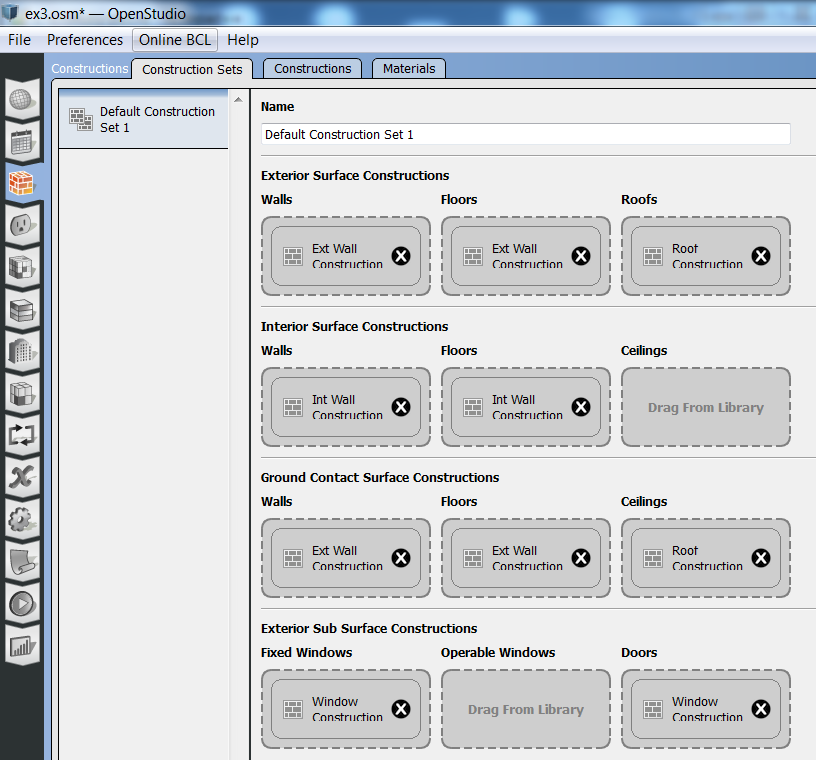
default\_const\_set**.**setDefaultInteriorSubSurfaceConstructions**(**default\_subsurface\_consts**)**

default\_const\_set**.**setDefaultGroundContactSurfaceConstructions**(**default\_ext\_surface\_consts**)**

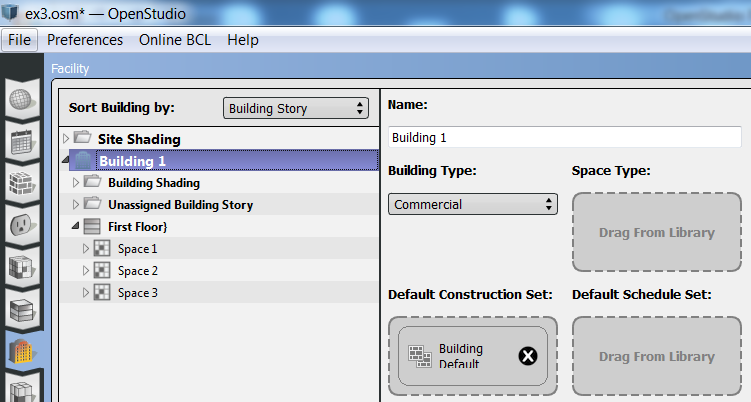
#assign the default construction set to the building

model**.**getBuilding**.**setDefaultConstructionSet**(**default\_const\_set**)**

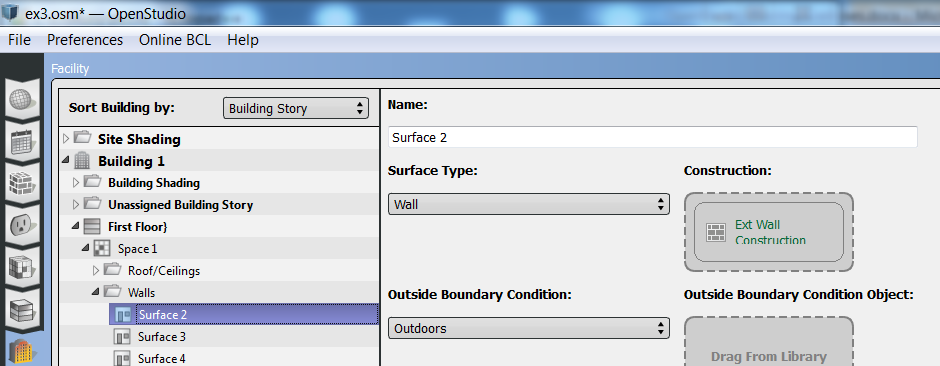
1. Run the script and open ex3.osm in the OpenStudio App. On the Constructions tab, under the Constructions sub-tab, you should see the Constructions in the model.



1. On the Facility tab, click on “Building” and notice that the Default Construction Set is assigned to the Building.



1. Under the Facility tab, expand the building until you can click on a surface. Notice that the surface has inherited its Construction from the Building’s Default Construction Set.



## Exercise 4 – Create Loads and Outdoor Air Requirements

At the moment, the building has surfaces that make up spaces, but there are no loads (people, lights, equipment, etc) in these spaces. The goal of this exercise is to create some loads and assign them to the spaces in the building.

1. Open Path/To/OSTraining/EX4/ex4.rb.
2. Create a People Definition and two Lights Definition objects and set their properties

#create a people definition

people\_def **=** OpenStudio**::**Model**::**PeopleDefinition**.**new**(**model**)**

people\_def**.**setName**(**"People Def - 1 person/200sf"**)**

people\_def**.**setPeopleperSpaceFloorArea**(**0.053820**)** #people/m^2

people\_def**.**setFractionRadiant**(**0.3**)**

people\_def**.**setSensibleHeatFraction**(**0.1**)**

#create a lights definition

lights\_def **=** OpenStudio**::**Model**::**LightsDefinition**.**new**(**model**)**

lights\_def**.**setName**(**"Lights Def- 1 W/sf"**)**

lights\_def**.**setWattsperSpaceFloorArea**(**11.0**)** #W/m^2

lights\_def**.**setFractionRadiant**(**0.30**)**

lights\_def**.**setFractionVisible**(**0.20**)**

lights\_def**.**setReturnAirFraction**(**0.0**)**

#create a second lights definition

lights\_def\_2 **=** OpenStudio**::**Model**::**LightsDefinition**.**new**(**model**)**

lights\_def\_2**.**setName**(**"Lights Def - 2 W/sf"**)**

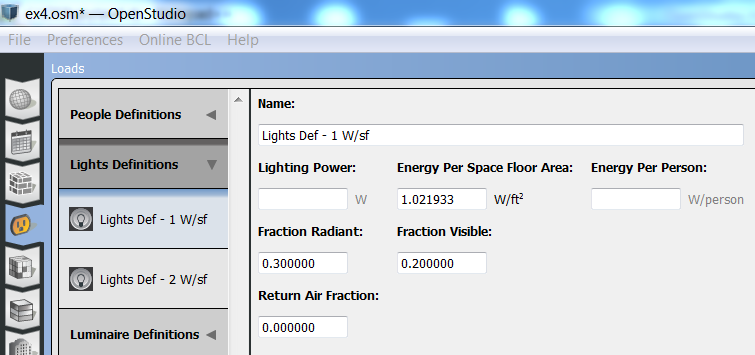
lights\_def\_2**.**setWattsperSpaceFloorArea**(**22.0**)** #W/m^2

lights\_def\_2**.**setFractionRadiant**(**0.30**)**

lights\_def\_2**.**setFractionVisible**(**0.20**)**

lights\_def\_2**.**setReturnAirFraction**(**0.0**)**

1. Run the script and open ex4.osm in the OpenStudio App. On the Loads tab you should see the Lights and People Definitions.



1. Create a Ventilation object.

#create a design outdoor air object

dsn\_oa **=** OpenStudio**::**Model**::**DesignSpecificationOutdoorAir**.**new**(**model**)**

dsn\_oa**.**setName**(**"Design OA"**)**

dsn\_oa**.**setOutdoorAirFlowperPerson**(**0.010765**)** #m^3/s\*person = 22.8 cfm/person

1. Create two Space Type objects. Create a People object and two Lights objects and assign to the new Space Type.

#create the first space type

space\_type\_a **=** OpenStudio**::**Model**::**SpaceType**.**new**(**model**)**

space\_type\_a**.**setName**(**"Space Type A"**)**

#create a people instance and assign to the first space type

people\_a **=** OpenStudio**::**Model**::**People**.**new**(**people\_def**)**

people\_a**.**setName**(**"People for Space Type A"**)**

people\_a**.**setSpaceType**(**space\_type\_a**)**

#create a lights instance and assign to the first space type

lights\_a **=** OpenStudio**::**Model**::**Lights**.**new**(**lights\_def\_a**)**

lights\_a**.**setName**(**"Lights for Space Type A"**)**

lights\_a**.**setSpaceType**(**space\_type\_a**)**

#create the second space type

space\_type\_b **=** OpenStudio**::**Model**::**SpaceType**.**new**(**model**)**

space\_type\_b**.**setName**(**"Space Type B"**)**

#create a people instance and assign to the second space type

people\_b **=** OpenStudio**::**Model**::**People**.**new**(**people\_def**)**

people\_b**.**setName**(**"People for Space Type B"**)**

people\_b**.**setSpaceType**(**space\_type\_b**)**

#create a lights instance and assign to the second space type

lights\_b **=** OpenStudio**::**Model**::**Lights**.**new**(**lights\_def\_b**)**

lights\_b**.**setName**(**"Lights for Space Type B"**)**

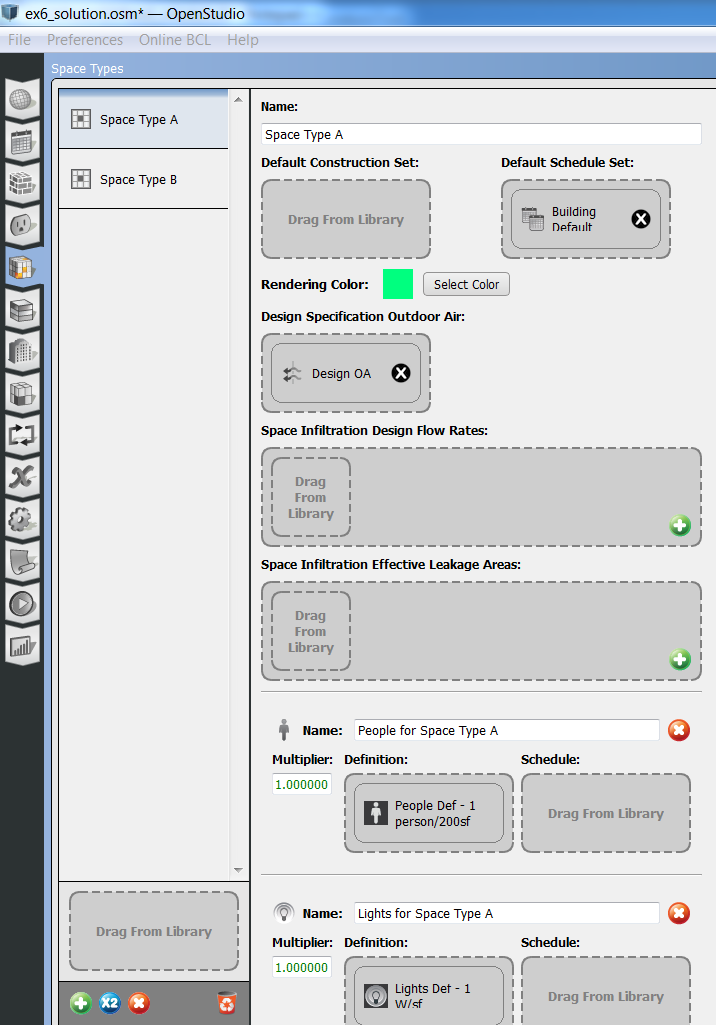
lights\_b**.**setSpaceType**(**space\_type\_b**)**

#assign the design outdoor air to both space types

space\_type\_a**.**setDesignSpecificationOutdoorAir**(**dsn\_oa**)**

space\_type\_b**.**setDesignSpecificationOutdoorAir**(**dsn\_oa**)**

1. Run the script and open ex4.osm in the OpenStudio App. On the Space Types tab you should see the Space Types with the associated Load Instances.



1. Assign the Space Type A to Space 1 and Space 2, and assign Space Type B to Space 3.

#assign space type A to space 1 and space 2

**if** model**.**getSpaceByName**(**"Space 1"**)**

space\_1 **=** model**.**getSpaceByName**(**"Space 1"**).**get

space\_1**.**setSpaceType**(**space\_type\_a**)**

**end**

**if** model**.**getSpaceByName**(**"Space 2"**)**

space\_2 **=** model**.**getSpaceByName**(**"Space 2"**).**get

space\_2**.**setSpaceType**(**space\_type\_a**)**

**end**

#assign space type B to space 3

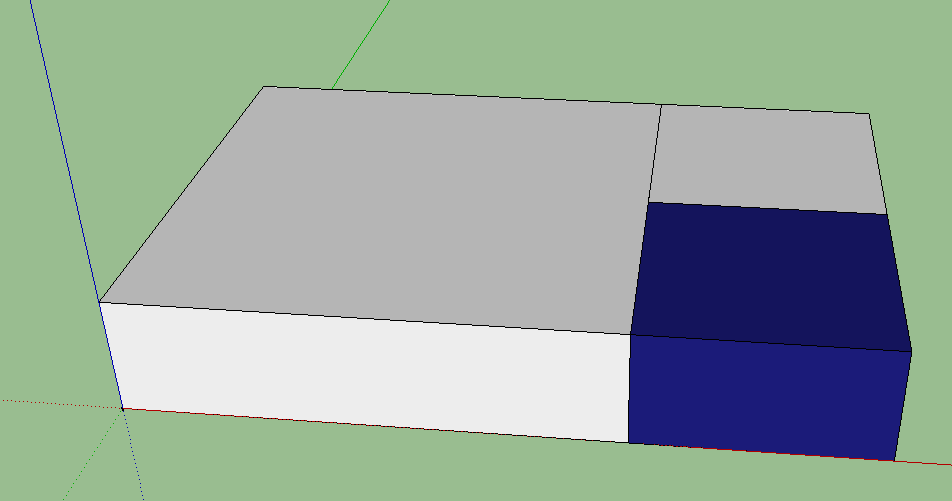
**if** model**.**getSpaceByName**(**"Space 3"**)**

space\_1 **=** model**.**getSpaceByName**(**"Space 3"**).**get

space\_1**.**setSpaceType**(**space\_type\_b**)**

**end**

1. Open ex4.osm in the SketchUp plugin. In render-by-space-type mode, you should see that space types have been assigned as specified in the code.



## Exercise 5 – Create Operation Schedules

At this point, the model has geometry, materials, and loads, but it does not have any schedules to determine when these loads operate. The goal of this exercise is to add those schedules.

The schedules we’ll be creating:

|  |  |
| --- | --- |
| Occupancy Schedule | Sat, Sun: 50% 9am-5pm  Weekdays: 90% 9am-6pm |
| Lighting Schedule | Sat, Sun: 50% 9am-5pm  Weekdays: 90% 9am-6pm |
| Heating Setpoint Schedule | All days, 65F 12am-7am, 70F 7am-8pm, 65F 8pm-12am |
| Cooling Setpoint Schedule | All days, 80F 12am-7am, 75F 7am-8pm, 80F 8pm-12am |
| People Activity Schedule | All days, 120W 24/7 (each person produces 120W 24/7) |

1. Open Path/To/OSTraining/EX5/ex5.rb.
2. Create some schedule type limits. This allows unit conversion in the GUI.

#create some schedule type limits

frac\_sch\_lim **=** OpenStudio**::**Model**::**ScheduleTypeLimits**.**new**(**model**)**

frac\_sch\_lim**.**setName**(**"Fractional Sch Limits"**)**

frac\_sch\_lim**.**setLowerLimitValue**(**0.0**)**

frac\_sch\_lim**.**setUpperLimitValue**(**1.0**)**

frac\_sch\_lim**.**setNumericType**(**"Continuous"**)**

temp\_sch\_lim **=** OpenStudio**::**Model**::**ScheduleTypeLimits**.**new**(**model**)**

temp\_sch\_lim**.**setName**(**"Temperature Sch Limits"**)**

temp\_sch\_lim**.**setLowerLimitValue**(**0.0**)**

temp\_sch\_lim**.**setUpperLimitValue**(**50.0**)**

temp\_sch\_lim**.**setNumericType**(**"Continuous"**)**

temp\_sch\_lim**.**setUnitType**(**"Temperature"**)**

activity\_sch\_lim **=** OpenStudio**::**Model**::**ScheduleTypeLimits**.**new**(**model**)**

activity\_sch\_lim**.**setName**(**"Activity Level Sch Limits"**)**

activity\_sch\_lim**.**setLowerLimitValue**(**0.0**)**

activity\_sch\_lim**.**setUpperLimitValue**(**600.0**)**

activity\_sch\_lim**.**setNumericType**(**"Continuous"**)**

activity\_sch\_lim**.**setUnitType**(**"ActivityLevel"**)**

1. Add a new schedule for occupancy. Schedule Ruleset is used because it can be viewed and edited easily in the OpenStudio App.

#make a new schedule ruleset for occupancy

occ\_sch **=** OpenStudio**::**Model**::**ScheduleRuleset**.**new**(**model**)**

occ\_sch**.**setName**(**"Occupancy Schedule"**)**

occ\_sch**.**setScheduleTypeLimits**(**frac\_sch\_lim**)**

#weekdays

occ\_week\_day **=** occ\_sch**.**defaultDaySchedule

occ\_week\_day**.**setName**(**"Occupancy Schedule Week Day"**)**

occ\_week\_day**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 9**,** 0**,** 0**),** 0.0**)**

occ\_week\_day**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 18**,** 0**,** 0**),** 0.90**)**

occ\_week\_day**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 24**,** 0**,** 0**),** 0.0**)**

#saturdays

occ\_sat\_rule **=** OpenStudio**::**Model**::**ScheduleRule**.**new**(**occ\_sch**)**

occ\_sat\_rule**.**setName**(**"Occupancy Schedule Saturday Rule"**)**

occ\_sat\_rule**.**setApplySaturday**(true)**

occ\_sat **=** occ\_sat\_rule**.**daySchedule

occ\_sat**.**setName**(**"Occupancy Schedule Saturday"**)**

occ\_sat**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 9**,** 0**,** 0**),** 0.0**)**

occ\_sat**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 17**,** 0**,** 0**),** 0.50**)**

occ\_sat**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 24**,** 0**,** 0**),** 0.0**)**

#sundays

occ\_sun\_rule **=** OpenStudio**::**Model**::**ScheduleRule**.**new**(**occ\_sch**)**

occ\_sun\_rule**.**setName**(**"Occupancy Schedule Sunday Rule"**)**

occ\_sun\_rule**.**setApplySunday**(true)**

occ\_sun **=** occ\_sun\_rule**.**daySchedule

occ\_sun**.**setName**(**"Occupancy Schedule Sunday"**)**

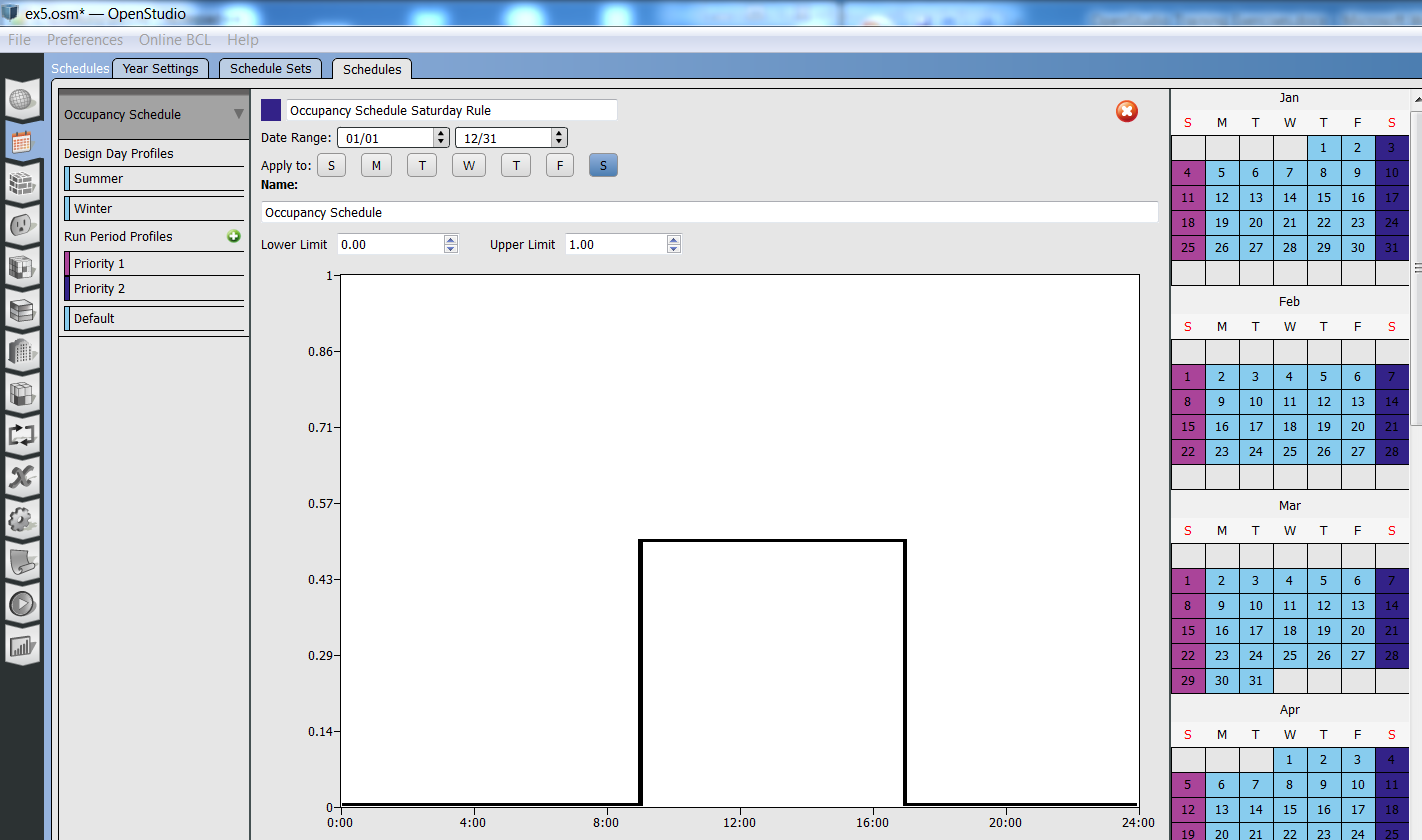
occ\_sun**.**setScheduleTypeLimits**(**frac\_sch\_lim**)**

occ\_sun**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 9**,** 0**,** 0**),** 0.0**)**

occ\_sun**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 17**,** 0**,** 0**),** 0.50**)**

occ\_sun**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 24**,** 0**,** 0**),** 0.0**)**

1. Run the script and open ex5.osm in the OpenStudio App. On the Schedules tab under the Schedules sub-tab, you should see the new schedule.



1. Add the other schedules, following the same pattern.

#make a new schedule ruleset for lighting

lts\_sch **=** OpenStudio**::**Model**::**ScheduleRuleset**.**new**(**model**)**

lts\_sch**.**setName**(**"Lighting Schedule"**)**

lts\_sch**.**setScheduleTypeLimits**(**frac\_sch\_lim**)**

#weekdays

lts\_week\_day **=** lts\_sch**.**defaultDaySchedule

lts\_week\_day**.**setName**(**"Lighting Schedule Week Day"**)**

lts\_week\_day**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 9**,** 0**,** 0**),** 0.0**)**

lts\_week\_day**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 18**,** 0**,** 0**),** 0.90**)**

lts\_week\_day**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 24**,** 0**,** 0**),** 0.0**)**

#saturdays

lts\_sat\_rule **=** OpenStudio**::**Model**::**ScheduleRule**.**new**(**lts\_sch**)**

lts\_sat\_rule**.**setName**(**"Lighting Schedule Saturday Rule"**)**

lts\_sat\_rule**.**setApplySaturday**(true)**

lts\_sat **=** lts\_sat\_rule**.**daySchedule

lts\_sat**.**setName**(**"Lighting Schedule Saturday"**)**

lts\_sat**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 9**,** 0**,** 0**),** 0.0**)**

lts\_sat**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 17**,** 0**,** 0**),** 0.50**)**

lts\_sat**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 24**,** 0**,** 0**),** 0.0**)**

#sundays

lts\_sun\_rule **=** OpenStudio**::**Model**::**ScheduleRule**.**new**(**lts\_sch**)**

lts\_sun\_rule**.**setName**(**"Lighting Schedule Sunday Rule"**)**

lts\_sun\_rule**.**setApplySunday**(true)**

lts\_sun **=** lts\_sun\_rule**.**daySchedule

lts\_sun**.**setName**(**"Lighting Schedule Sunday"**)**

lts\_sun**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 9**,** 0**,** 0**),** 0.0**)**

lts\_sun**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 17**,** 0**,** 0**),** 0.50**)**

lts\_sun**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 24**,** 0**,** 0**),** 0.0**)**

#make a new schedule ruleset for heating setpoint

htg\_setpoint\_sch **=** OpenStudio**::**Model**::**ScheduleRuleset**.**new**(**model**)**

htg\_setpoint\_sch**.**setName**(**"Heating Setpoint Schedule"**)**

htg\_setpoint\_sch**.**setScheduleTypeLimits**(**temp\_sch\_lim**)**

#all days

htg\_setpoint\_all\_days **=** htg\_setpoint\_sch**.**defaultDaySchedule

htg\_setpoint\_all\_days**.**setName**(**"Heating Setpoint Schedule All Days"**)**

htg\_setpoint\_all\_days**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 7**,** 0**,** 0**),** 18.33**)**

htg\_setpoint\_all\_days**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 20**,** 0**,** 0**),** 21.11**)**

htg\_setpoint\_all\_days**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 24**,** 0**,** 0**),** 18.33**)**

#make a new schedule ruleset for cooling setpoint

clg\_setpoint\_sch **=** OpenStudio**::**Model**::**ScheduleRuleset**.**new**(**model**)**

clg\_setpoint\_sch**.**setName**(**"Cooling Setpoint Schedule"**)**

clg\_setpoint\_sch**.**setScheduleTypeLimits**(**temp\_sch\_lim**)**

#all days

clg\_setpoint\_all\_days **=** clg\_setpoint\_sch**.**defaultDaySchedule

clg\_setpoint\_all\_days**.**setName**(**"Cooling Setpoint Schedule All Days"**)**

clg\_setpoint\_all\_days**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 7**,** 0**,** 0**),** 26.67**)**

clg\_setpoint\_all\_days**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 20**,** 0**,** 0**),** 23.89**)** clg\_setpoint\_all\_days**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 24**,** 0**,** 0**),** 26.67**)**

#make a new schedule ruleset for people activity

people\_activity\_sch **=** OpenStudio**::**Model**::**ScheduleRuleset**.**new**(**model**)**

people\_activity\_sch**.**setName**(**"People Activity Schedule"**)**

people\_activity\_sch**.**setScheduleTypeLimits**(**activity\_sch\_lim**)**

#all days

people\_activity\_all\_days **=** people\_activity\_sch**.**defaultDaySchedule

people\_activity\_all\_days**.**setName**(**"People Activity Schedule All Days"**)**

people\_activity\_all\_days**.**addValue**(**OpenStudio**::**Time**.**new**(**0**,** 24**,** 0**,** 0**),** 120.0**)**

1. Create a new Default Schedule Set, assign the occupancy and lighting schedules to it, and then assign this Default Schedule Set to the space types in the model.

#make a default schedule set object

default\_sch\_set **=** OpenStudio**::**Model**::**DefaultScheduleSet**.**new**(**model**)**

default\_sch\_set**.**setName**(**"Building Default Schedule Set"**)**

#assign the occupancy and lighting schedules to the default schedule set

default\_sch\_set**.**setNumberofPeopleSchedule**(**occ\_sch**)**

default\_sch\_set**.**setLightingSchedule**(**lts\_sch**)**

#assign this default schedule set to the two space types created earlier

**if** model**.**getSpaceTypeByName**(**"Space Type A"**)**

space\_type\_a **=** model**.**getSpaceTypeByName**(**"Space Type A"**).**get

space\_type\_a**.**setDefaultScheduleSet**(**default\_sch\_set**)**

**end**

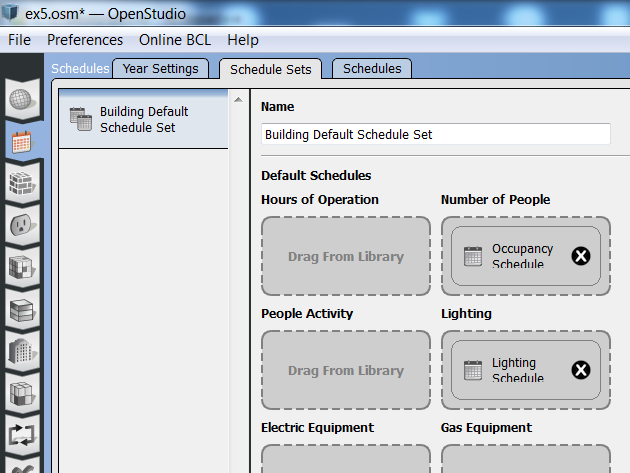
**if** model**.**getSpaceTypeByName**(**"Space Type B"**)**

space\_type\_b **=** model**.**getSpaceTypeByName**(**"Space Type B"**).**get

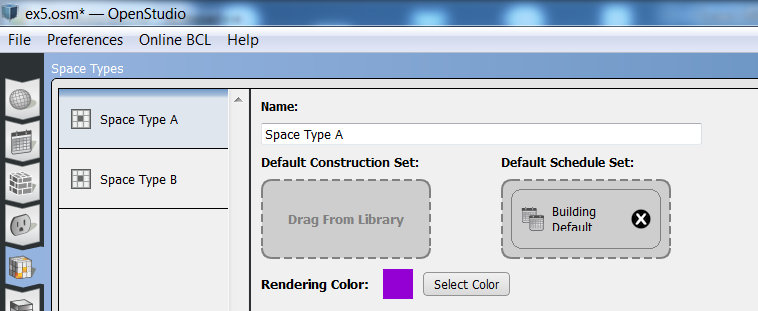
space\_type\_b**.**setDefaultScheduleSet**(**default\_sch\_set**)**

**end**

1. Run the script and open ex5.osm in the OpenStudio App. On the Schedules tab under the Schedules Sets sub-tab, you should see the new Default Schedule Set.



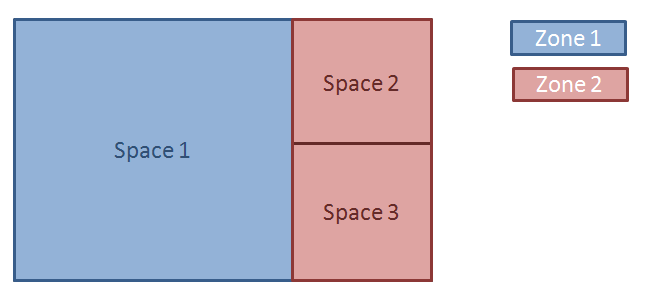
1. On the Space Types tab you should see that the Space Types have been assigned to use the new Default Schedule Set.



## Exercise 6 – Create Thermal Zones

The building now has geometry, envelope properties, and internal loads with schedules. In order to run simulation, the Spaces in the building must be aggregated into Thermal Zones. Thermal Zones are the basic unit in EnergyPlus simulation.

Here is the Thermal Zoning Configuration we’ll be making. Notice that Zone 1 contains only Space 1, while Zone 2 contains Space 2 and Space 3.



1. Open Path/To/OSTraining/EX6/ex6.rb.
2. Create two new thermal zones in the model.

#create two new thermal zones

zone\_1 **=** OpenStudio**::**Model**::**ThermalZone**.**new**(**model**)**

zone\_1**.**setName**(**"Zone 1"**)**

zone\_2 **=** OpenStudio**::**Model**::**ThermalZone**.**new**(**model**)**

zone\_2**.**setName**(**"Zone 2"**)**

1. Add the spaces to the new thermal zones.

#add space 1 to thermal zone 1

**if** model**.**getSpaceByName**(**"Space 1"**)**

space\_1 **=** model**.**getSpaceByName**(**"Space 1"**).**get

space\_1**.**setThermalZone**(**zone\_1**)**

**end**

#add spaces 2 and 3 to thermal zone 2

**if** model**.**getSpaceByName**(**"Space 2"**)**

space\_2 **=** model**.**getSpaceByName**(**"Space 2"**).**get

space\_2**.**setThermalZone**(**zone\_2**)**

**end**

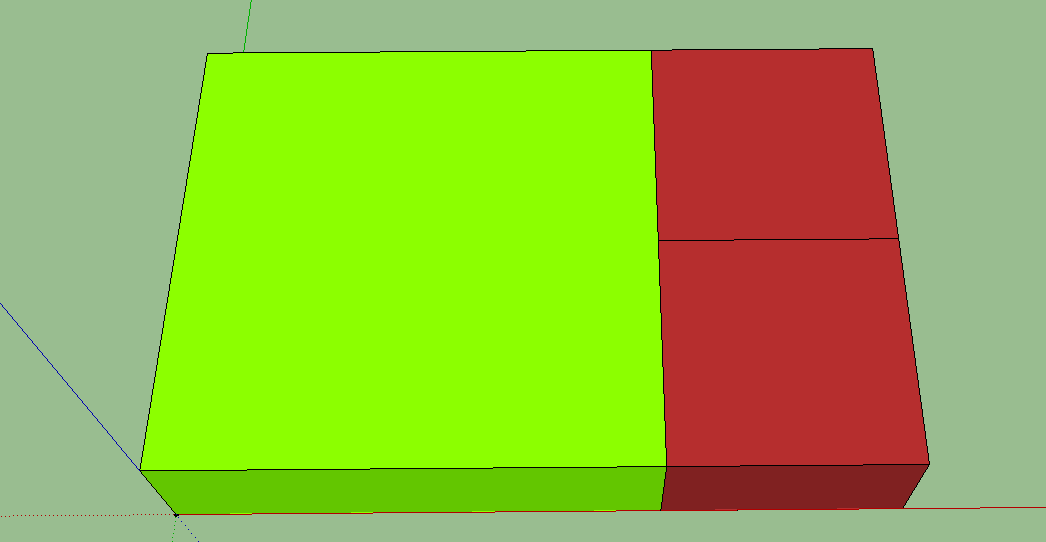
**if** model**.**getSpaceByName**(**"Space 3"**)**

space\_3 **=** model**.**getSpaceByName**(**"Space 3"**).**get

space\_3**.**setThermalZone**(**zone\_2**)**

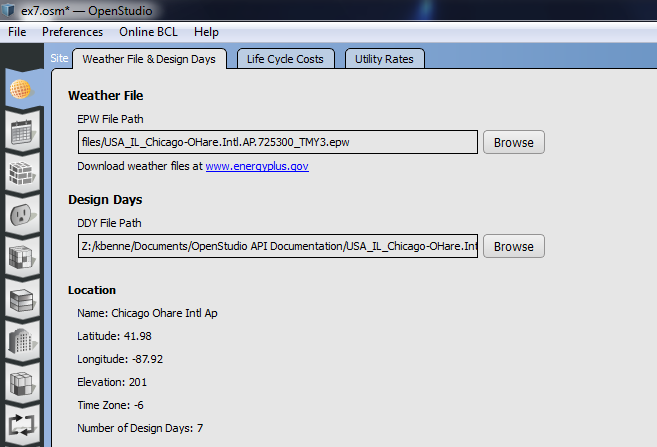
**end**

1. Load ex6.osm in the SketchUp plugin. In render-by-thermal-zone mode you should see the thermal zoning as specified in the diagram above.



## Exercise 7 – Run Simulation From OpenStudio Application

1. Launch the OpenStudio application.
2. Choose File->Open and browse to the NoHVACExample.osm file in the exercises folder.
3. Choose File->Save As and save a copy of the example as ex7.osm in the EX7 folder.
4. Use the “Browse” button to set the EPW and DDY file paths. These files are included with the OpenStudio exercises.



1. Switch to the “Run Simulation” tab and click the Run button to execute a new EnergyPlus simulation. Note: If this is the first time running a simulation with this version of OpenStudio you will need to select File->Scan for Tools to locate EnergyPlus.
2. When the simulation is complete you may click on the “Results Summary” tab to view a summary of the results.
3. The included NoHVACExample.osm file should be the same as the ex6.osm file that you generated in exercise 6. If you have successfully completed exercise 6 you may use this same procedure to simulate your own ex6.osm file.

## Exercise 8 – Run Simulation Programmatically

In this exercise you will run a simulation of the NoHVACExample.osm file programmatically. This exercise will be the programmatic equivalent to running a simulation by clicking the run button in the OpenStudio Application.

1. Setup paths to the NoHVACExample.osm file and the output location

# Setup model and output paths relative to this script

model\_path **=** OpenStudio**::**Path**.**new**(**'/OSTraining/NoHVACExample.osm'**)**

output\_path **=** OpenStudio**::**Path**.**new**(**'/OSTraining/EX8/'**)**

1. Setup a path to the weather file. Use the weather file included with the exercises.

# Find weather file

epw\_path **=** OpenStudio**::**Path**.**new**(**"C:/OSTraining/USA\_IL\_Chicago-OHare.Intl.AP.725300\_TMY3.epw"**)**

1. Add design days to your model using the ddy file included with the exercises. The basic steps are to load the ddy file, use the OpenStudio Reverse translator to create a ddy model, and then add the DesignDay objects to the NoHVACExample model.

# add design days to model

ddy\_path **=** OpenStudio**::**Path**.**new**(**"/OSTraining/USA\_IL\_Chicago-OHare.Intl.AP.725300\_TMY3.ddy"**)**

ddy\_file **=** OpenStudio**::**IdfFile**::**load**(**ddy\_path**).**get

ddy\_workspace **=** OpenStudio**::**Workspace**.**new**(**ddy\_file**)**

reverse\_translator **=** OpenStudio**::**EnergyPlus**::**ReverseTranslator**.**new

ddy\_model **=** reverse\_translator**.**translateWorkspace**(**ddy\_workspace**);**

design\_days **=** ddy\_model**.**getDesignDays

version\_translator **=** OpenStudio**::**OSVersion**::**VersionTranslator**.**new

model **=** version\_translator**.**loadModel**(**model\_path**).**get

model**.**addObjects**(**design\_days**)**

new\_model\_path **=** OpenStudio**::**Path**.**new**(**"/OSTraining/EX8/ex8.osm"**)**

model**.**save**(**new\_model\_path**)**

1. Create a RunManger

# Create a runmanager

db **=** output\_path **/** OpenStudio**::**Path**.**new**(**'runmanager.db'**)**

runmanager **=** OpenStudio**::**Runmanager**::**RunManager**.**new**(**db**,true)**

runmanager**.**setPaused**(true)**

1. Find EnergyPlus

# Find EnergyPlus

config\_options **=** runmanager**.**getConfigOptions**()**

config\_options**.**fastFindEnergyPlus**()**

tools **=** config\_options**.**getTools**()**

1. Create a Workflow that will translate the OpenStudio model into an EnergyPlus idf file and then execute an EnergyPlus simulation.

# Create an EnergyPlus workflow

workflow **=** OpenStudio**::**Runmanager**::**Workflow**.**new**(**"ModelToIdf->EnergyPlus"**)**

workflow**.**add**(**tools**)**

1. Create a job that will carry out the workflow on a specific model.

# Create a job

job **=** workflow**.**create**(**output\_path**,**new\_model\_path**,**epw\_path**)**

1. Put the job in the queue, start RunManager, and wait until the simulation is complete

# Put the job in the run queue, start runmanager, and wait until the simulation is complete

runmanager**.**enqueue**(**job**,true)**

runmanager**.**setPaused**(false)**

runmanager**.**waitForFinished**()**

1. Run the script from the EX8 folder

## Exercise 9 – Add HVAC Systems

1. Open ex9.rb in your text editor and edit it to perform the following operations using the comments in the file as a guide. Some steps are already completed because they have been covered in previous exercises.
2. Load the NoHVACExample.osm file as an OpenStudio model.
3. Retrieve the thermal zones and save them to a variable.

# Retrieve the thermal zones

zones **=** model**.**getThermalZones

1. Retrieve the heating and cooling setpoint schedules

# Add a new HVAC system and thermostat to each zone

heating\_schedule **=** model**.**getScheduleRulesetByName**(**'Heating Setpoint Schedule'**).**get

cooling\_schedule **=** model**.**getScheduleRulesetByName**(**'Cooling Setpoint Schedule'**).**get

1. Loop over the thermal zones and add a new HVAC system type 3 using a template. This system type corresponds to a packaged rooftop air conditioner. Also create a new thermostat for each zone.

# Loop over the thermal zones, adding system 3 and a thermostat

zones**.**each **do** **|**zone**|**

air\_loop\_hvac **=** OpenStudio**::**Model**::**addSystemType3**(**model**).**to\_AirLoopHVAC**.**get

air\_loop\_hvac**.**addBranchForZone**(**zone**)**

thermostat **=** OpenStudio**::**Model**::**ThermostatSetpointDualSetpoint**.**new**(**model**)**

thermostat**.**setHeatingSetpointTemperatureSchedule**(**heating\_schedule**)**

thermostat**.**setCoolingSetpointTemperatureSchedule**(**cooling\_schedule**)**

zone**.**setThermostatSetpointDualSetpoint**(**thermostat**)**

**end**

1. Save the model as ex9.osm.

# Specify where to save the model

save\_path **=** OpenStudio**::**Path**.**new**(**'ex9.osm'**)**

# Save the model

model**.**save**(**save\_path**,true)**

1. Run the script from the EX9 folder.
2. Visualize the new systems. Open the model in the OpenStudio application and navigate to the “HVAC Systems” tab. Using the selector at the top of the interface, you should be able to switch between the new packaged rooftop air conditioner systems.
3. You may optionally run the simulation using the application or by editing ex9.rb.

## Exercise 10 – Remove Existing HVAC Systems and Add New VAV System

1. Open ex10.rb in your text editor and edit it to perform the following operations using the comments in the file as a guide. Some steps are already completed because they have been covered in previous exercises.
2. Load the PSZExample.osm file as an OpenStudio model.
3. Add a new VAV system to the model using basic components. It is also possible to create a complete VAV system using the Model::addSystemType7 method, but this exercise will demonstrate the flexibility of building the system yourself. The Ruby code that is required for this step is included in the vav\_example file. You need to transcribe the text into ex10.rb. Study this script because it contains many examples of interacting with OpenStudio’s HVAC API.
4. Retrieve all of the thermal zones in the model and save them to an array.

# Get all of the thermal zones, we are going to remove the current

# PSZ system and add each zone to the VAV system

zones **=** model**.**getThermalZones

1. Loop over all of the thermal zones.

# Loop over all thermal zones

zones**.**each **do** **|**zone**|**

* 1. Retrieve each zone’s AirLoopHVAC object.

# Get the current system for each zone

old\_system **=** zone**.**airLoopHVAC

* 1. Determine if the current system is a packaged single zone system

# Figure out if the current system is PSZ

old\_system\_is\_psz **=** **false**

**if** old\_system**.**is\_initialized

old\_system **=** old\_system**.**get

old\_systems\_zones **=** old\_system**.**thermalZones

**if** old\_systems\_zones**.**size **==** 1

constant\_speed\_fans **=** old\_system**.**supplyComponents**(**OpenStudio**::**Model**::**FanConstantVolume**::**iddObjectType**)**

dx\_coils **=** old\_system**.**supplyComponents**(**OpenStudio**::**Model**::**CoilCoolingDXSingleSpeed**::**iddObjectType**)**

**if** constant\_speed\_fans**.**size **>=** 1 **and** dx\_coils**.**size **>=** 1

old\_system\_is\_psz **=** **true**

**end**

**end**

**end**

* 1. If the current system is a packaged single zone system then remove it.

# Remove the old system if it is PSZ

**if** old\_system\_is\_psz

old\_system**.**remove

**end**

* 1. Add each zone to the new VAV system

# Add zone to new VAV

new\_vav\_system**.**addBranchForZone**(**zone**)**

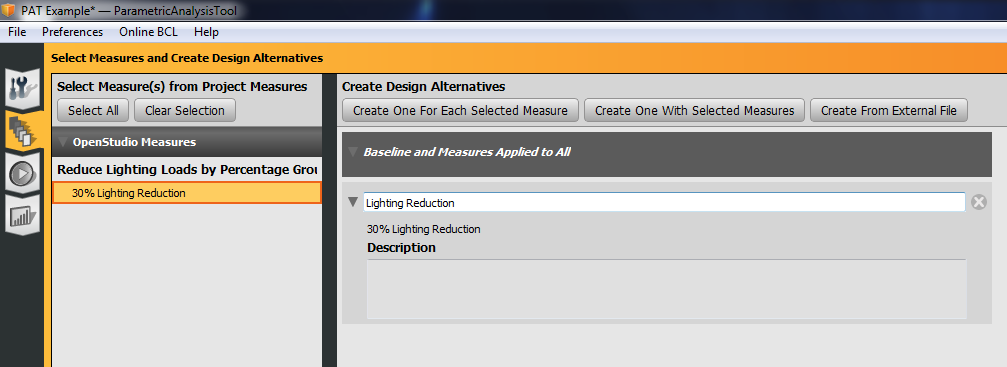
1. Save the model to a file named ex10.osm
2. Visualize the model by opening ex10.osm in the OpenStudio application.
3. You may optionally run the simulation using the application or by editing to ex10.rb script.

## Exercise 11 – PAT Create New Measure and Run Simulations

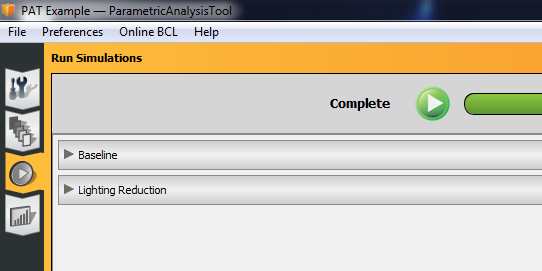
1. In this exercise you will learn how to use the PAT application to run multiple simulations and evaluate design alternatives. In the first half of the exercise you will use an existing OpenStudio Measure to evaluate the effect of reducing the lighting level. This basic procedure will familiarize you with how to use PAT. In the second half of the exercise you will create a new Measure to remove PSZ systems and replace them with a VAV system. Although OpenStudio Measures are implemented with Ruby code, you will write very little new code because we will recycle the work you did in exercise 11, turning it into a measure.
2. Open the existing OpenStudio project file by double clicking on the project.osp file located in PAT Example Project/ex11/project.osp.
3. The example project already has a seed model and it also includes a lighting measure group, which will be displayed in the first tab. The lighting measure group contains one measure, 30% Lighting Reduction. The lighting measure has not been applied to a design alternative at this point.



1. The second tab is for creating design alternatives using the measures that were grouped within the project on tab one. Create a new lighting reduction design alternative by selecting “30% Lighting Reduction” on the left column and then clicking “Create One With Selected Measures.” This will create a new design alternative that is the seed model with a 30 percent lighting reduction.



1. Tab three displays the design alternatives that will be simulated. You should see two entries, the baseline seed model, and the lighting reduction design alternative. Click the run button and run the two simulations. You can see information about each step of the simulation by expanding each section. If there are errors they will show up here.



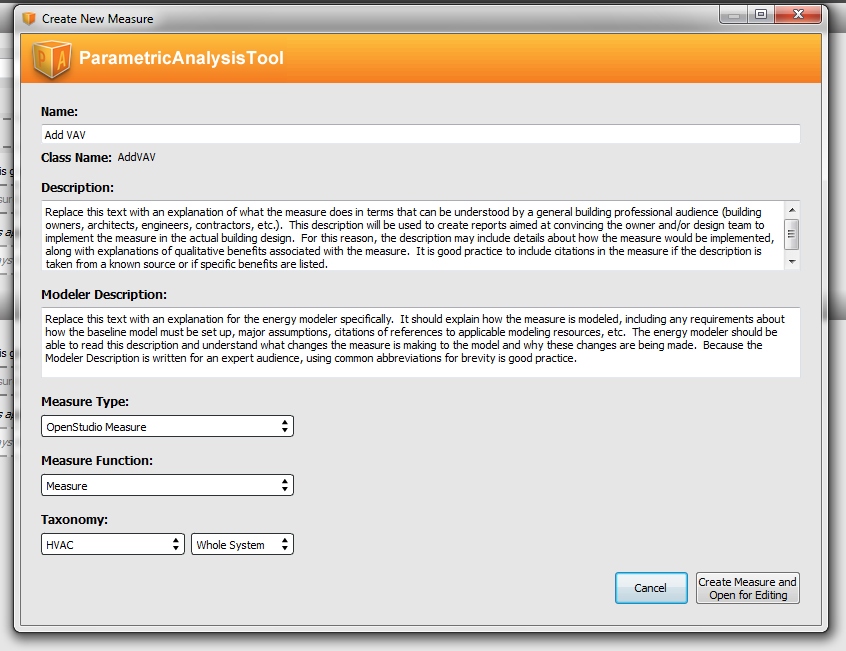
1. View the results on the fourth tab.



1. Now you are going to create a new OpenStudio Measure to remove packaged single zone systems and replace them with a single VAV system. The script to perform this operation was completed in exercise 10. Here you are going to formalize that script into a Measure that can be uploaded to the Building Component Library and reused.
2. Go back to the first tab and click on the “My Measures” button under the “Measure Library” tab on the right column. Click on the “Create New Template Script File” button at the bottom of the window.



1. Fill out the input form so that it looks like the following image. The most important part is to name the measure “Add VAV” and to choose “HVAC” and “Whole System” in the taxonomy. A complete measure should also have a clear description and modeler description, but you can leave the default text for this exercise. When you are done click “Create Measure and Open for Editing.”



1. The measure will open in your file browser. Open the measure.rb file in your text editor.
2. Details about authoring OpenStudio Measures are provided in the Measure Writing Guide. For this exercise edit the arguments method so that the measure has no arguments. This means that the new Add VAV measure will not require user input.

#define the arguments that the user will input

**def** **arguments(**model**)**

args **=** OpenStudio**::**Ruleset**::**OSArgumentVector**.**new

**return** args

**end** #end the arguments method

1. Edit the run method by removing the example Ruby code and replacing it with the following. This is equivalent to the work done in exercise 10, but here you will take a shortcut by using the built in template. When you have completed editing, save the measure.rb file and return to PAT.

#define what happens when the measure is run

**def** **run(**model**,** runner**,** user\_arguments**)**

**super(**model**,** runner**,** user\_arguments**)**

new\_vav\_system **=** OpenStudio**::**Model**::**addSystemType7**(**model**).**to\_AirLoopHVAC**.**get

# Get all of the thermal zones, we are going to remove the current PSZ system

# and add each zone to the VAV system

zones **=** model**.**getThermalZones

zones**.**each **do** **|**zone**|**

# Get the current system for each zone

old\_system **=** zone**.**airLoopHVAC

# Figure out if the current system is PSZ

old\_system\_is\_psz **=** **false**

**if** old\_system**.**is\_initialized

old\_system **=** old\_system**.**get

old\_systems\_zones **=** old\_system**.**thermalZones

**if** old\_systems\_zones**.**size **==** 1

constant\_speed\_fans **=** old\_system**.**supplyComponents**(**OpenStudio**::**Model**::**FanConstantVolume**::**iddObjectType**)**

dx\_coils **=** old\_system**.**supplyComponents**(**OpenStudio**::**Model**::**CoilCoolingDXSingleSpeed**::**iddObjectType**)**

**if** constant\_speed\_fans**.**size **>=** 1 **and** dx\_coils**.**size **>=** 1

old\_system\_is\_psz **=** **true**

**end**

**end**

**end**

# Remove the old system if it is PSZ

**if** old\_system\_is\_psz

old\_system**.**remove

**end**

# Add zone to new VAV

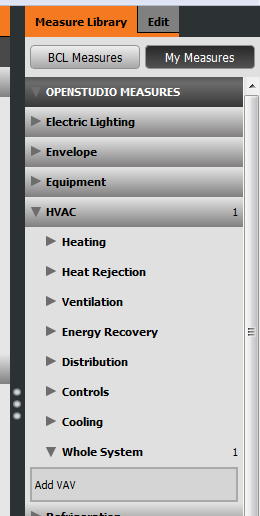
new\_vav\_system**.**addBranchForZone**(**zone**)**

**end**

**return** **true**

**end** #end the run method

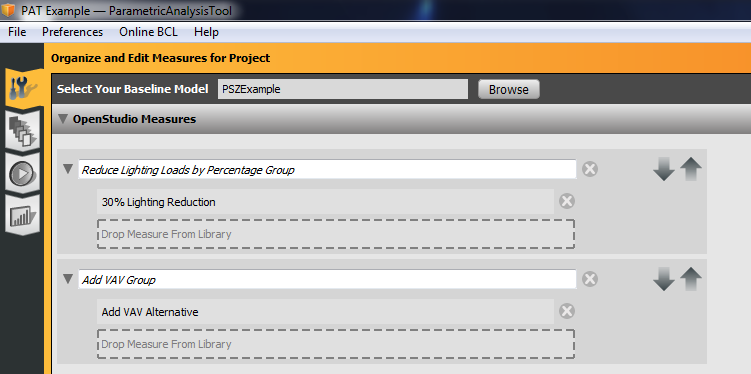
1. The new “Add VAV” measure will show up in PAT when you are on the first tab viewing “My Measures.”



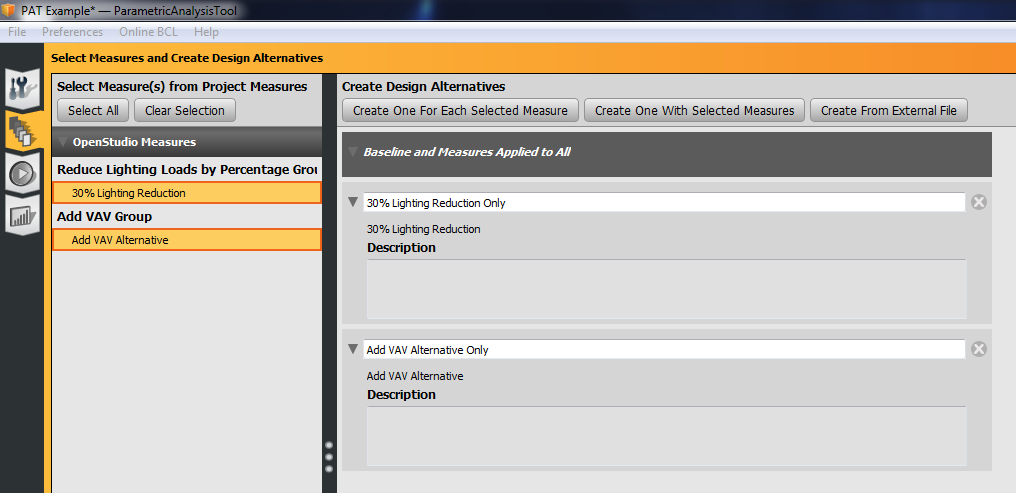
1. Drag the “Add VAV” measure from the left column into the drop zone to create a new measure group. If notified, accept the prompt that the project will be saved and existing design alternatives will be removed.



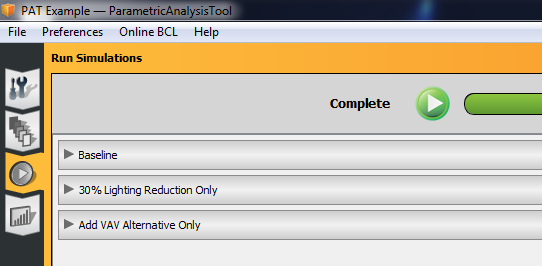
1. You should now see two measure groups. A lighting group and the new VAV group.



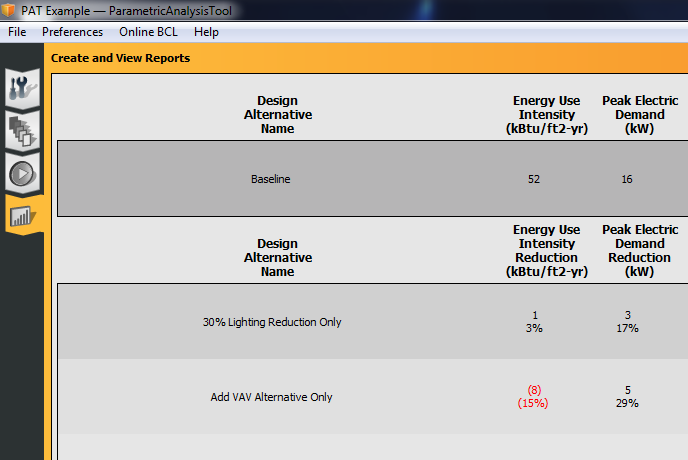
1. Go to the second tab. Select the “30% Lighting Reduction” measure and the “Add VAV Alternative” measures on the left, then click the “Create One For Each Selected Measure” button. You should have two new design alternatives.



1. Run the simulations from the third tab. Use the messages to debug as required.



1. View the results. This example didn’t perform very well with the VAV system. You can investigate to understand the cause.



## Exercise 12 – Access Predefined Results Data

1. Open ex12.rb in your text editor and edit it to perform the following operations using the comments in the file as a guide. Some steps are already completed because they have been covered in previous exercises.
2. Add Ruby code to run a simulation of the PSZExample.osm file. Make sure to output the results to the EX12 folder. This step is mostly complete, you only need to update the file paths.
3. Load the Sqlite output file produced by EnergyPlus using the OpenStudio::SqlFile class. If the simulation was successful it will be located in the EX12 folder under “ModelToIdf/EnergyPlus-0/eplusout.sql.”

# Load the sqlite file which contains results from EnergyPlus

sql\_path **=** output\_path **/** OpenStudio**::**Path**.**new**(**'ModelToIdf/EnergyPlus-0/eplusout.sql'**)**

sql **=** OpenStudio**::**SqlFile**.**new**(**sql\_path**);**

**if** **!** sql**.**connectionOpen

puts 'Unable to open Sqlite file.'

**end**

1. Use the SqlFile::netSiteEnergy method to retrieve the total net site energy in GJ.

# Retrieve the net site energy in GJ

puts sql**.**netSiteEnergy

1. Run the script from the EX12 folder.

## Exercise 13 – Make a Custom Results Query

1. Open ex13.rb in your text editor and edit it to perform the following operations using the comments in the file as a guide. Some steps are already completed because they have been covered in previous exercises.
2. Load the Sqlite output file from EnergyPlus using the OpenStudio::SqlFile class. If the simulation was successful it will be located in the EX13 folder under “ModelToIdf/EnergyPlus-0/eplusout.sql.”

# Load the sqlite file which contains results from EnergyPlus

sql\_path **=** output\_path **/** OpenStudio**::**Path**.**new**(**'ModelToIdf/EnergyPlus-0/eplusout.sql'**)**

sql **=** OpenStudio**::**SqlFile**.**new**(**sql\_path**);**

**if** **!** sql**.**connectionOpen

puts 'Unable to open Sqlite file.'

exit

**end**

1. Create a sql querry and use the SqlFile::execAndReturnFirstDouble method to execute the query against the sqlite output file. This method should return the number of occupied hours when the zone heating setpoint is unmet.

# Retrieve the net site energy in GJ

query **=** "SELECT Value FROM tabulardatawithstrings WHERE "

query **<<** "ReportName='SystemSummary' and " # Notice no space in SystemSummary

query **<<** "ReportForString='Entire Facility' and "

query **<<** "TableName='Time Setpoint Not Met' and "

query **<<** "RowName='Facility' and "

query **<<** "ColumnName='During Occupied Cooling' and "

query **<<** "Units='hr';"

unmet\_heating\_hours **=** sql**.**execAndReturnFirstDouble**(**query**)**

**if** unmet\_heating\_hours**.**empty?

puts 'No Value'

**else**

puts unmet\_heating\_hours**.**get

**end**

1. Run the script from the EX13 folder.

## Exercise 14 – Create and Read a Report Variable

1. Open ex14.rb in your text editor and edit it to perform the following operations using the comments in the file as a guide. Some steps are already completed because they have been covered in previous exercises.
2. Load the PSZExample.osm file as an OpenStudio::Model object.
3. Add a new OutputVariable object to the model.

# Add a new OutputVariable object for "Total Electric Power Purchased"

output\_variable **=** OpenStudio**::**Model**::**OutputVariable**.**new**(**"Total Electric Power Purchased"**,**model**)**

1. Save the model as ex14.osm.

# Save the model as ex14.osm

new\_model\_path **=** output\_path **/** OpenStudio**::**Path**.**new**(**'ex14.osm'**)**

model**.**save**(**new\_model\_path**,true)**

1. Run a simulation of ex14.osm.
2. Load the sqlite output file using the OpenStudio SqlFile class.

# Load the sqlite file which contains results from EnergyPlus

sql\_path **=** output\_path **/** OpenStudio**::**Path**.**new**(**'ModelToIdf/EnergyPlus-0/eplusout.sql'**)**

sql **=** OpenStudio**::**SqlFile**.**new**(**sql\_path**);**

**if** **!** sql**.**connectionOpen

puts 'Unable to open Sqlite file.'

exit

**end**

1. Retrieve the timeseries for “Total Electric Power Purchased.”

# Make a query

run\_period **=** 'RUN PERIOD 1'

frequency **=** 'Hourly'

variable\_name **=** 'Total Electric Power Purchased'

key **=** 'Whole Building'

timeseries **=** sql**.**timeSeries**(**run\_period**,**frequency**,**variable\_name**,**key**).**get

values **=** timeseries**.**values

1. Output statistics of timeseries.

# Output results

puts OpenStudio**::**minimum**(**values**)**

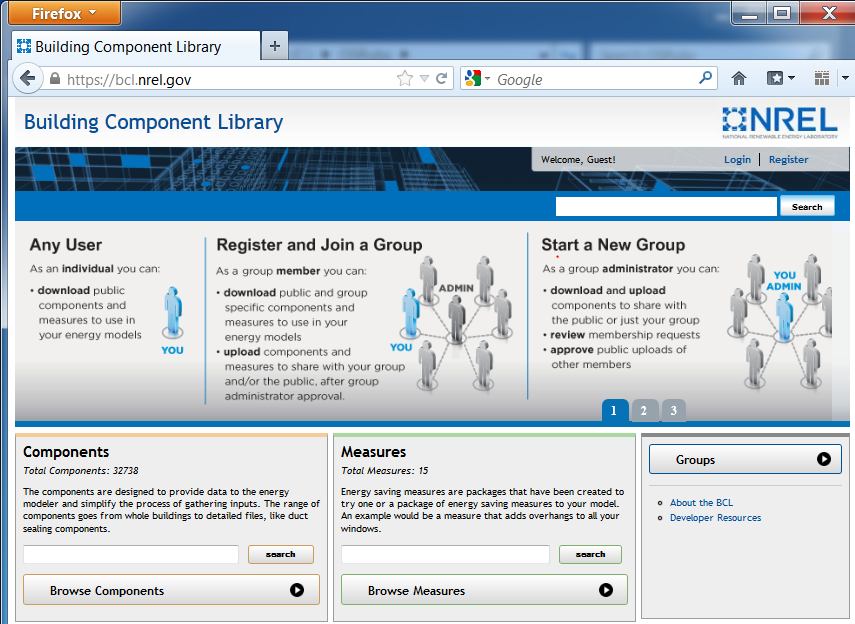
puts OpenStudio**::**maximum**(**values**)**

puts OpenStudio**::**mean**(**values**)**

1. Run script from the EX14 folder.

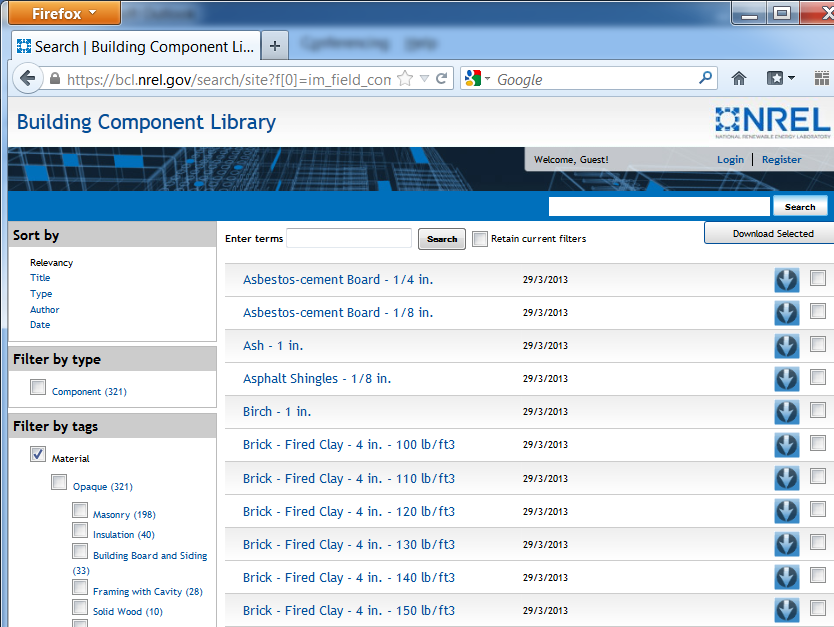
## Exercise 15 – Browse Existing BCL Content

The Building Component Library (BCL) is an online database (<https://bcl.nrel.gov/>) that stores content for building energy modeling. Going to the website, you will see that there are two major categories of content: Components and Measures.

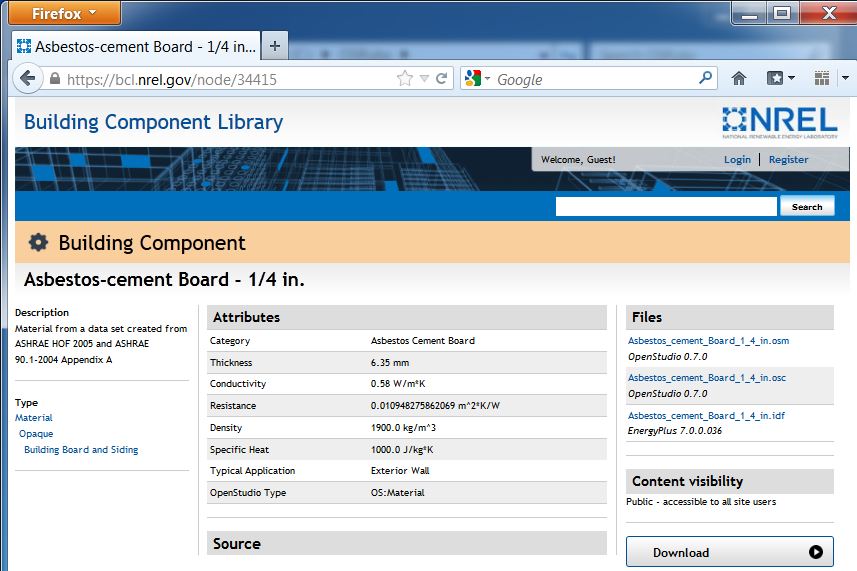


### Components

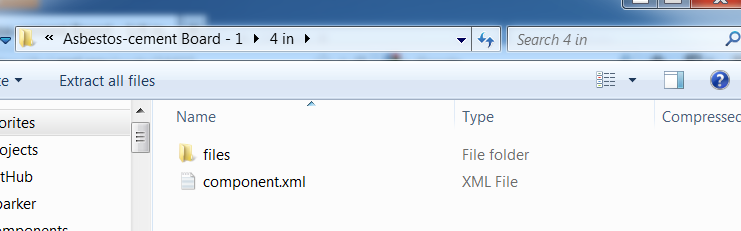
Components on the BCL are input information for building energy modeling. For example, the BCL currently contains a number of different building materials.



Clicking on one of the individual components will show you the attributes of that specific BCL component.



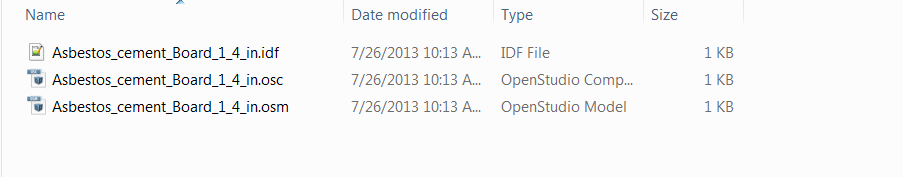
Clicking the “Download” button on BCL will download a zip file containing the simulation content.



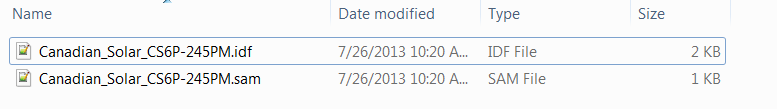
The first file, component.xml, contains information about all of the components attributes, as well as a list of the files associated with this component.



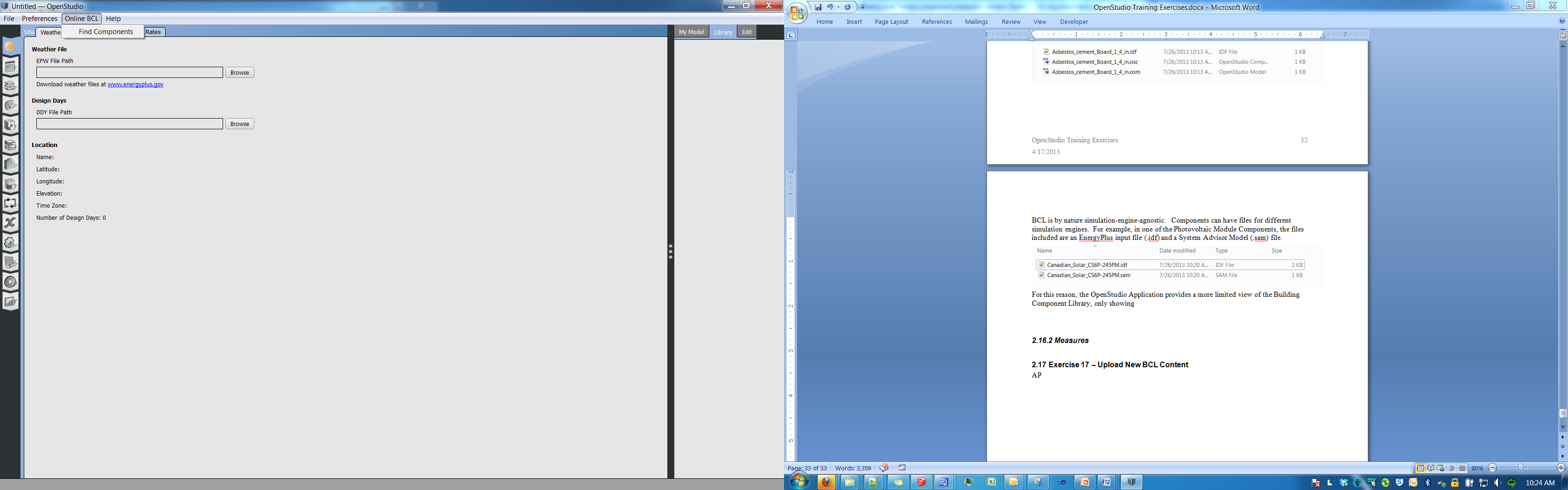
Inside the files directory, there may be one or more files describing the modeling content. For example, the example Material Component shown contains an OpenStudio model file (.osm), an OpenStudio component file (.osc) and an EnergyPlus file (.idf).



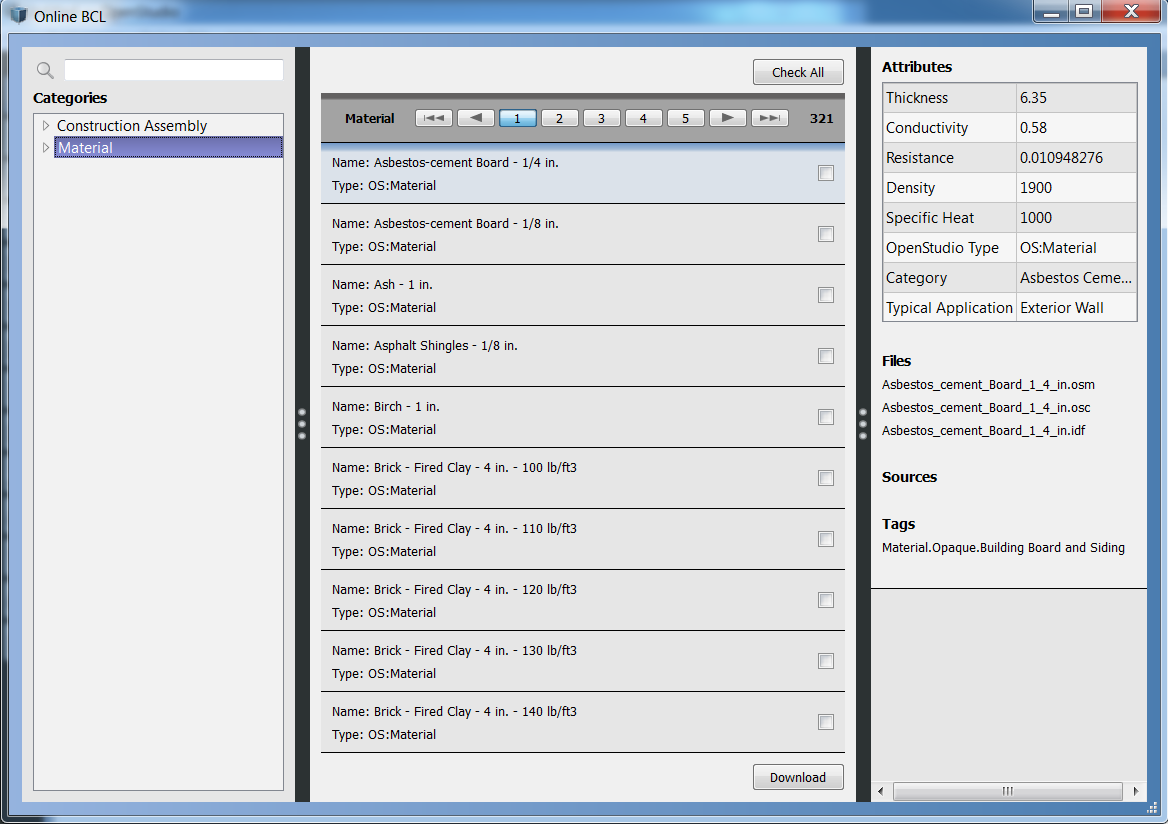
BCL is by nature simulation-engine-agnostic. Components can have files for different simulation engines. For example, in one of the Photovoltaic Module Components, the files included are an EnergyPlus input file (.idf) and a System Advisor Model (.sam) file.



For this reason, the OpenStudio Application provides a more limited view of the Building Component Library, only showing BCL Components that contain the necessary files to be used in OpenStudio.



Looking at the BCL through the OpenStudio Application, we can see the same Materials Components that are visible on the BCL website, but we do not see the Photovoltaic Module Components, because they do not contain the files that make them useful to OpenStudio.



Clicking the check boxes next to a Component, then clicking the “Download” button at the bottom will download that Component and automatically add it to your OpenStudio library, available for use in your model directly.

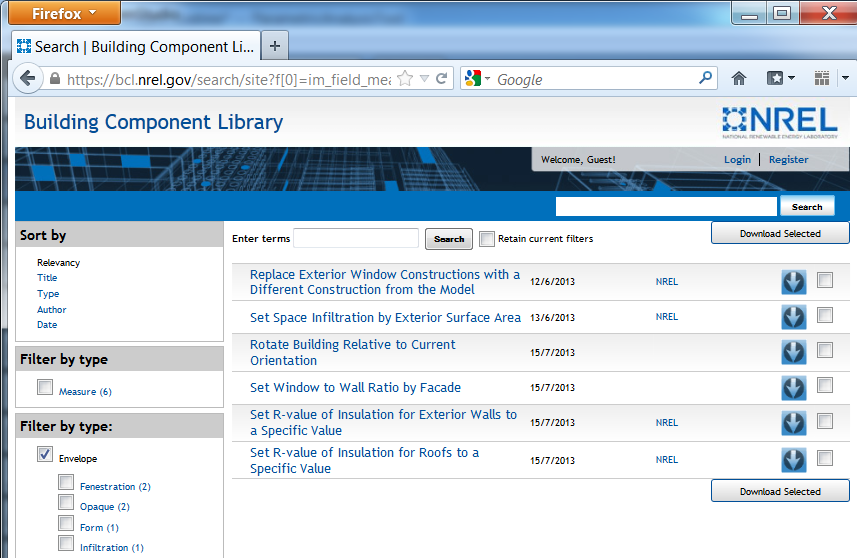


…



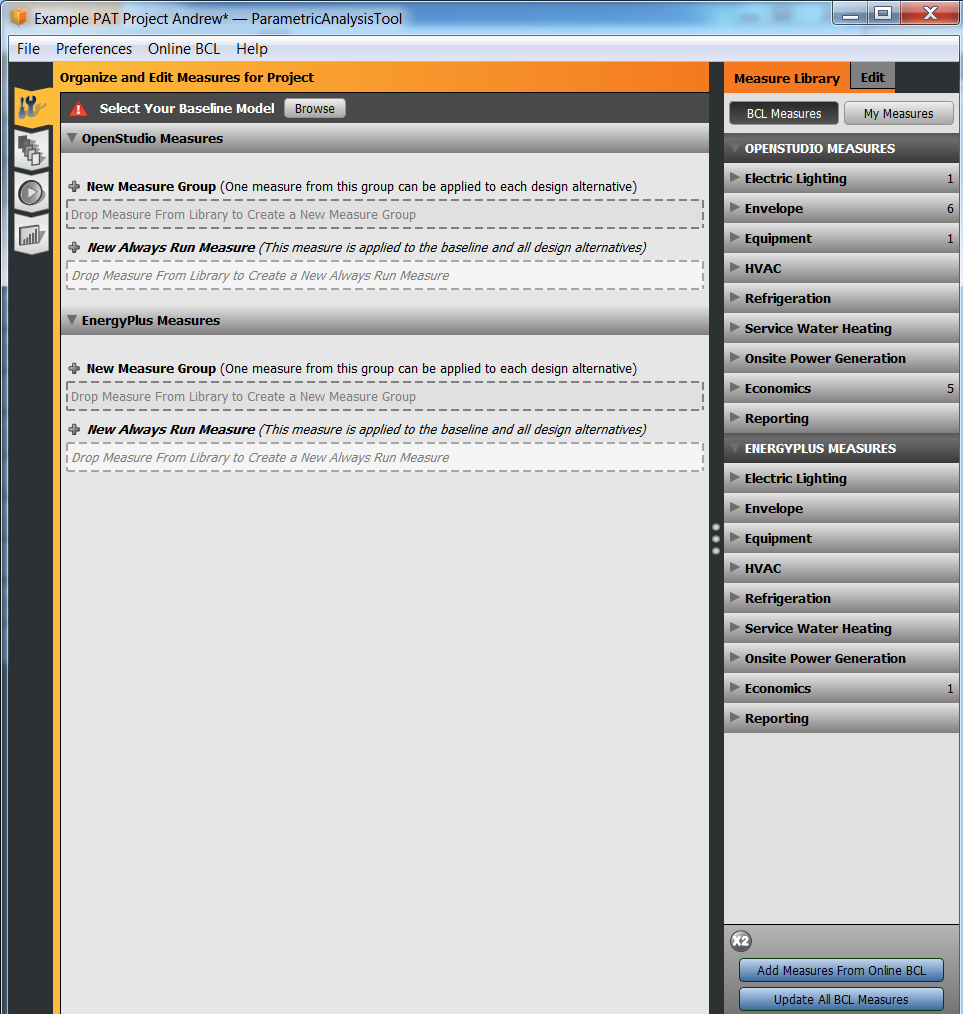
### Measures

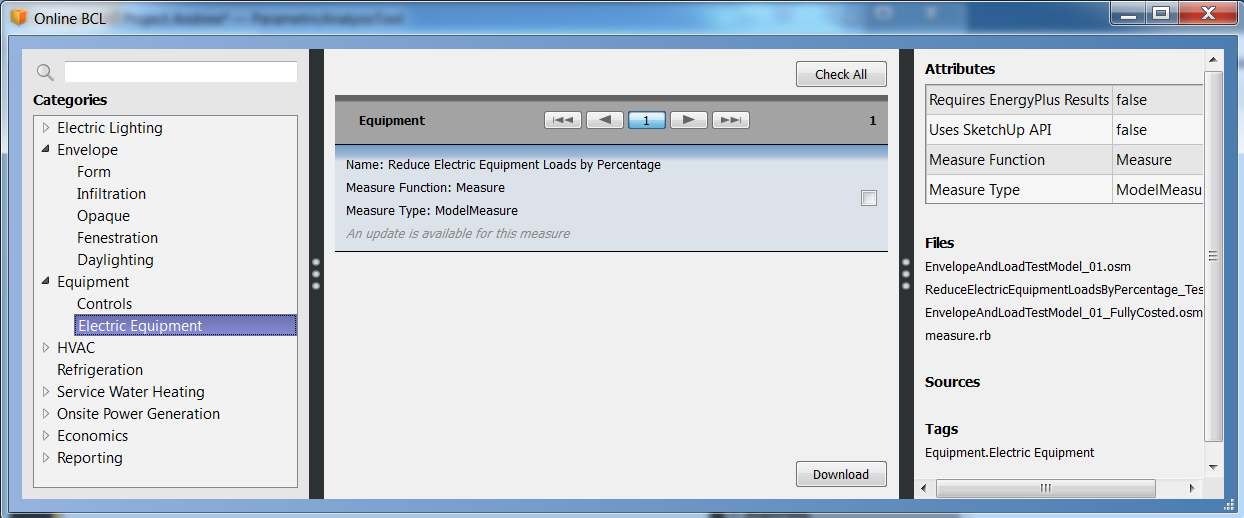
Measures on the BCL are self-contained scripts designed to make a change to a building energy model to simulate the effects of a retrofit or a different design option. For example, one Measure might reduce lighting power density in the building by 15%. Opening the BCL website and browsing through the measures, one sees the current options.

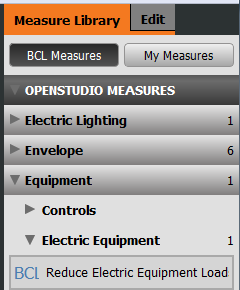


Similar to Components, downloading a measure returns a measure.xml describing the attributes of the measure, as well as a series of files. The measure.rb file contains the actual script that changes the model.

Also similar to components, there is an interface in PAT that allows measures to be downloaded directly from BCL into your local library.







## Exercise 16 – Upload New BCL Content

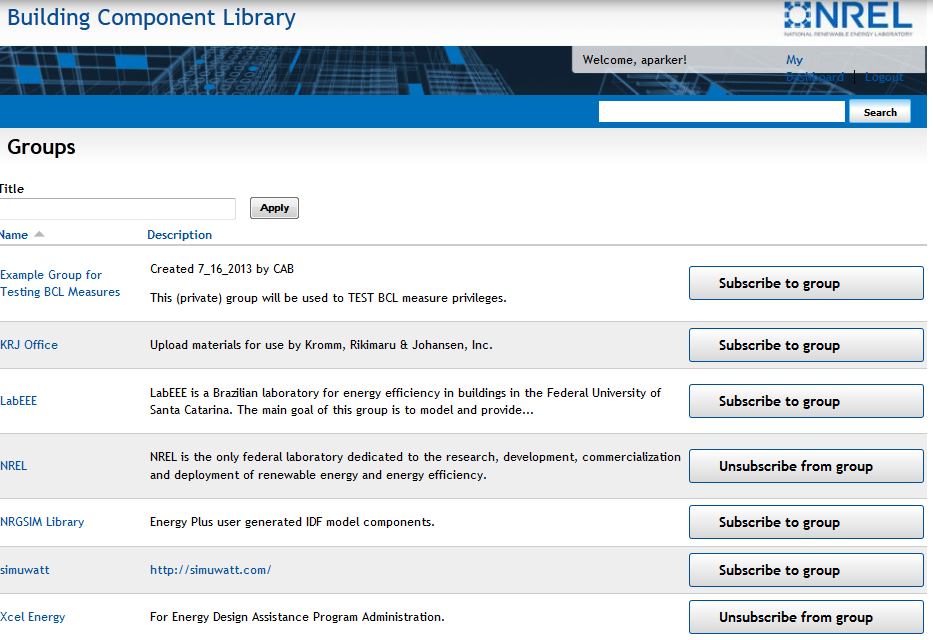
Content can be added to the BCL by anyone with a BCL account. This exercise will show the basics of adding Components and Measures.

### Content Control and Groups

Before going into the specifics of uploading, the topic of content control must be addressed. Any user can upload content to the BCL. If a typical user publishes a Component or Measure, and the site moderator approves, the Component or Measure will be publicly available to all BCL users unless the person who uploaded it removes it.

In some cases, particularly for private organizations, there is a desire to share content with members of the organization, but not with BCL users outside the organization. In order to facilitate this user-based content control, Groups were implemented. A Group can have one or more group administrator. The group administrator can manage who is or isn’t in the Group. Similarly, the Group administrator must approve the publishing of any Components or Measures, and may set the privacy setting of this content. Any content owned by a Group may either be “Public” to everyone on BCL or “Private” to only other Group members and administrators. A user can be a member or administrator of one or more Groups.

A new Group can be created by any BCL user. When this Group request is approved by a BCL moderator, the Group name will appear on the Groups page. Users wishing to join a Group may click the “Subscribe to Group” button, which will send a request to the administrators of that Group.



Important Note:

As of OpenStudio 1.0.0, “Private” content is not available to users via the OpenStudio Application or PAT. This is because of a legal barrier that is currently being reviewed by the NREL legal department. The intent in the near future is to show all “Public” content, as well as any “Private” content to a user based on the BCL login information they provide via the user interface.

### Components

This section will describe one way to upload Components to the BCL. This section assumes that the user has Ruby 1.8.7 or greater installed. This method is only available to users with elevated BCL access privileges (such as other users at National Labs) who are trusted by NREL and who accept the responsibility of quality control and support for their content. This upload process is heavily programmatic; this is necessary for large numbers of components. Although we strive to make this process simple, you will encounter issues. **Please read the messages in the console – they are your debugging friends!**

1. In the Materials folder, open Components.xlsx.

Open the BCL website, and look at the spreadsheet. Notice how the attribute columns list the names of the attributes, and each row is an individual component. Notice that the paths in the spreadsheet are incorrect. Fix them to match your configuration. Save the spreadsheet.

1. Install the bcl ruby gem.

C:\users\aparker> gem install bcl (Press ENTER)

1. Change to the exercise 17 directory.

C:\users\aparker> cd C:\OSRuby\EX17 (Press Enter)

1. List all the possible rake commands.

C:\OSTraining\EX16> rake -T (Press Enter)

1. You should see something like this.

C:\OSTraining\EX16> rake -T (Press Enter)

rake components:delete\_receipts # delete receipt files for all component

rake components:generate # generate all component types listed in

rake components:push # push all component types listed in

rake components:update # update all component types listed in

1. Open the file called C:\Users\myusername\.bcl\config.yml and edit.

---

:server:

:url: https://bcl.nrel.gov

:admin\_user:

:password: mybclpassword

:username: mybclusername

1. Decide which component types we’ll work with. These are specified inside setup\_component\_types.rb

**def** **component\_types()**

#array to hold all the component types

component\_types **=** **[]**

#add the component types to upload

component\_types **<<** "Materials"

**return** component\_types

**end**

1. Generate the components.

C:\OSTraining\EX16> rake components:generate (Press Enter)

You should see something like this.

C:\OSTraining\EX16>rake components:generate

\*\*\*\*\*\*\*\*\*\*\*Generating BCL component files for: Materials\*\*\*\*\*\*\*\*\*\*\*

Reading C:/OSRuby/EX17/Materials/Components.xlsx

[ComponentSpreadsheet] Starting parsing components of type Material.Opaque.

[ComponentSpreadsheet] Finished parsing components of type Material.Opaque.

Loading current taxonomy from C:/Ruby187/lib/ruby/gems/1.8/gems/bcl.2.9/

tarring batch 1 of 1 to components\_1.tar.gz

1. Push the components to BCL.

C:\OSTraining\EX16> rake components:push (Press Enter)

You should see something like this.

C:\OSTraining\EX16 >rake components:push

loading config settings from U://.bcl/config.yml

Connecting to bcl.nrel.gov on port 443

warning: peer certificate won't be verified in this SSL session

SESSION COOKIE: SESSfaaeee5f1efd0340efef1847db711442=GEht9zjL0IlKdQSPWJo9Dxgt5PqDL83qlDQu86yNQ8k;BNI\_bcl=000000000000000000000000423aaec000005000;BNES\_SESSfaaee

e5f1efd0340efef1847db711442=JTJiULQImml2HoOiNvn2TtP8WKSAM3DdsQSnaUZqliH+zMC2KbguhKe/DqDSCiE6kFIXg6Jcl36MUN5jSMcsnmH50cwFZAJAYWyuqtKOGmtLqImICU7/+ryhfbgsO+bol7iv

2xGNa2KE+wM2ZrE3Uw==

\*\*\*\*\*\*\*\*\*\*\*Pushing component files to BCL for: Materials\*\*\*\*\*\*\*\*\*\*\*

Getting ready to push 1 files

pushing component C:/OSRuby/EX17/Materials/components/TEST\_MATERIAL\_FOR\_TRAINING/TEST\_MATERIAL\_FOR\_TRAINING.tar.gz:

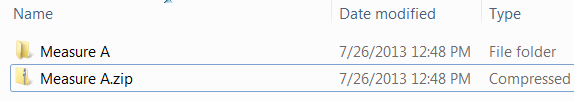
1. Assuming you have BCL access and the correct access privileges, in about 10 minutes, your new content should show up on BCL.
2. If you make changes to components, either in the Components.xlsx spreadsheet or in the files themselves, use this command to upload the updates. Note, if you try to re-upload something you’ve already uploaded instead of updating, it won’t work. This is because BCL keeps every version of every Component or Measure. If it is truly a new Component, create a new row for it in the spreadsheet.

C:\OSTraining\EX16 > rake components:update (Press Enter)

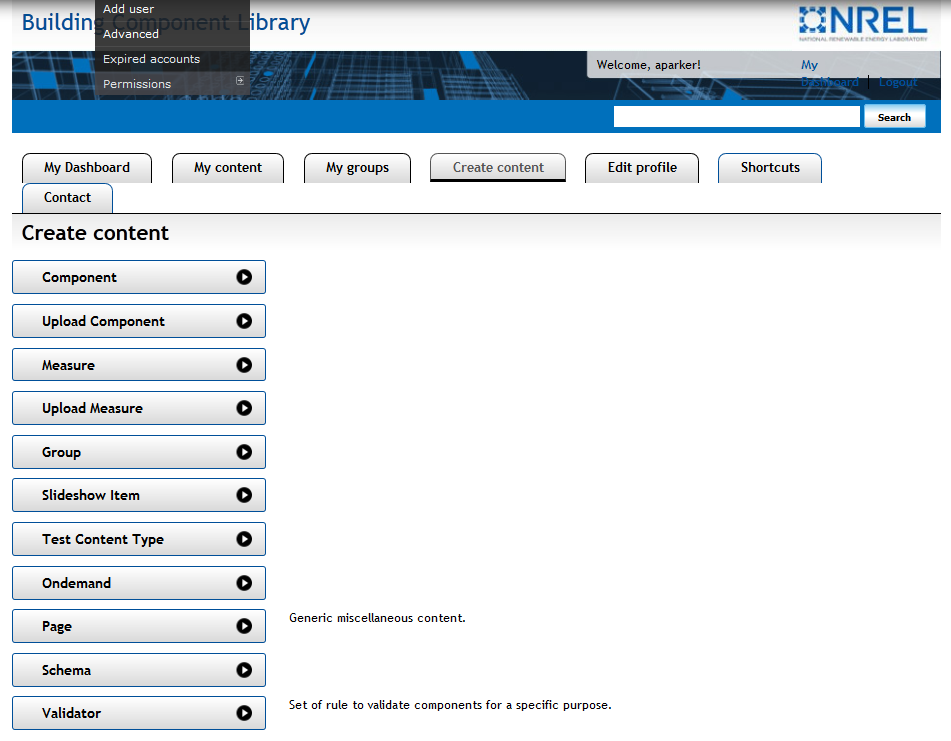
### Measures

For uploading measures, although there is a similarly programmatic method, because the number of measures is much lower, we will demonstrate the manual process.

1. Open PAT and open the “My Measures” directory.
2. Zip the directory of the measure you want to upload.



1. Open the BCL website. Go to “My Dashboard” then “Create Content” then “Upload Measure.”



1. Browse for the “Measure A.zip” file
2. Go through the Data, Group, Menu Settings, etc. tabs.
3. Click “Save”

That’s it! The measure should show up on the BCL moderator page for review. The measure should show up on the BCL home page and in PAT about 10 minutes after the moderator approval.