The MineBuilder Tutorial

Introduction:

Minecraft is a simple game where players can build, craft, explore and survive in an infinite sandbox world to their heart's content. With over 200 million copies sold, it is the best-selling video game of all time! Owing to that huge success, thousands of players design and build elaborate and beautiful structures in Minecraft, ranging from castles to cities to college campuses to a 1:1 scale model of the Starship Enterprise. Unfortunately, not everyone has the time or the talent to design or implement such grand builds. That's where MineBuilder comes in! MineBuilder intends to redefine the way you build structures in Minecraft: Java Edition.

Our easy-to-use language allows users to rapidly design and build structures in Minecraft without even having to move the character or press an in-game button! Structures are easily designed and built with various characters that correspond to actual blocks inside of the game. Instead of having to spend hours manually placing every block within the game, programmers will be able to easily design and manipulate structures out of ASCII characters in MineBuilder, and programmatically build them in seconds within the world of Minecraft!

Prerequisites

MineBuilder is an incredibly easy to learn programming language that lets you build structures in Minecraft: Java Edition. In fact, all building operations rely on only one Minecraft command: `/setblock <x> <y> <z> <tileName>` That's it! The MineBuilder language utilizes several different technologies in order to convert your programs into structures in Minecraft: Java Edition, including F#, Python, Minecraft Spigot servers, and the Raspberry Juice Minecraft API.

Before you can get started writing programs in MineBuilder, there's a few prerequisites you'll need to make sure everything goes smoothly:

- MineBuilder Console Project Files
- Minecraft: Java Edition
- .NET Framework 3.0
- Python
- RaspberryJuice Minecraft API*
- Minecraft Spigot Server*

**The last two prerequisites can be a little tricky to set up, so we recommend following Carlos Argueta's tutorial to get them set up correctly*

Running Programs

To run programs in MineBuilder, make sure you are in the MineBuilder project directory, minebuilder/, then use the command `dotnet run <program_file_path>`.

Writing Programs

All programs in MineBuilder are made up of combinations of four simple commands:
`define <var> as {<blueprint>}`,
`build <var> at (<x>,<y>,<z>)`,
`repeat <var> starting at (<x>,<y>,<z>) for <int> times, <int> apart in <dir> direction`, and
`end`.

Define

The `define <var> as {<blueprint>}` commands allow you to assign a variable name to a `<blueprint>`. Blueprints can be inserted in-line right between the '{ }' or you can use a `load <filename.mpb>` statement to load in a valid `<blueprint>` from a .mpb file located in the MineBuilder minebuilder/ directory. `.mpb` stands for Minecraft Python Blueprint.

Blueprints

Blueprints will generally make up the bulk of your MineBuilder code. Blueprints are designed literally from the ground up, layer by layer, with each ASCII character representing an actual block in Minecraft: Java Edition! Below you can find the list of MineBuilder supported blocks, and the characters you can use to represent them in your programs.

Block	ASCII Character		Block	ASCII Character
AIR	'.'		**STONE_SLAB_DOUBLE**	'?'
STONE	'S'		**STONE_SLAB**	'@'
GRASS	'R'		**BRICK_BLOCK**	'B'
DIRT	'D'		**TNT**	't'
COBBLESTONE	'C'		**BOOKSHELF**	'b'
WOOD_PLANKS	'W'		**MOSS_STONE**	'M'
SAND	's'		**OBSIDIAN**	'O'
GRAVEL	'r'		**CHEST**	'+'
GOLD_ORE	'1'		**DIAMOND_ORE**	'5'

IRON_ORE	'2'		**DIAMOND_BLOCK**	'd'
COAL_ORE	'3'		**CRAFTING_TABLE**	'$'
WOOD	'w'		**FURNACE_ACTIVE**	'F'
LEAVES	'l'		**REDSTONE_ORE**	'6'
GLASS	'~'		**ICE**	'i'
LAPIS_LAZULI_ORE	'4'		**SNOW_BLOCK**	'^'
LAPIS_LAZULI_BLOCK	'L'		**CLAY**	'c'
SANDSTONE	'N'		**FENCE**	'H'
WOOL	'P'		**GLOWSTONE_BLOCK**	'&'
GOLD_BLOCK	'G'		**STONE_BRICK**	'*'
IRON_BLOCK	'I'		**GLASS_PANE**	'p'
GLOWING_OBSIDIAN	'o'		**MELON**	'm'

Using in-line blueprints

Blueprints are designed layer-by-layer, separated by a new line and a '-' character. The first 'slice' is the 'bottom' layer of your structure, with subsequent slices positioned one block space in the y-direction above the last. Here's an example of a very simple program that includes a blueprint designed in-line within a `define` command. This is the definition from Example 0 from the MineBuilder language specification.

```
define woodStack as
{
     // this is the bottom layer made of wood
     www
     www
     -
     // this is the top layer made of wood planks
     WWW
     WWW
     -
}

end
```

Here, we have defined a structure called `woodStack` that is made up of a 2x3 layer of wood planks on top of a 2x3 layer of wood.

As demonstrated, each 2D Slice is in the X-Z direction (x in the horizontal, and z in the vertical), while each additional slice is in the Y direction. Thus, this building is 2x3x2 size using (x,z,y) coordinates.

Using external blueprints

MineBuilder also makes it easy to share blueprints through `load <filename.mpb>` statements. .mpb files contain valid blueprints, so instead of designing your raw blueprints in your MineBuilder program as in the example above, you can refer to a .mpb file with the relevant code instead. These files must be included in the working directory of your code. For example, the program below is semantically identical to the program in ***Using in-line blueprints***.

```
define woodStack as
{
    // this statement grabs blueprint code from the specified .mpb
    load wood.mpb
}

end
```

Now let's take a look inside `wood.mpb`

```
// this is the bottom layer made of wood
WWW
WWW
-
// this is the top layer made of wood planks
WWW
WWW
-
```

You can go ahead and run this program, but you'll notice that it doesn't really do much. That's because we haven't told MineBuilder to build anything yet!

Build

Once you have defined a couple structures using the `define` command, you'll likely want to actually build them in a location in Minecraft. `build <var> at (<x>,<y>,<z>)` statements take in a name of a structure that you have previously defined with the `define` command, and a set of coordinates corresponding to a point in Minecraft. At this point it's important to make sure your Spigot server is running so that MineBuilder can interact with your Minecraft world. While not necessary, it is helpful to load up Minecraft and actually join your world so you can find a set of coordinates and eventually see your structure built!

Tip: You can press F3 to see your current coordinates

Now that you have your server running, let's modify our example program to build our `woodStack`. This is Example 0 from the MineBuilder language specification.

```
define woodStack as
{
      // this is the bottom layer made of wood
      WWW
      WWW
      -
      // this is the top layer made of wood planks
      WWW
      WWW
      -
}

// builds the previously defined structure at the given coordinates
build woodStack at (120,79,-244)

end
```

If you run this program, you should see your `woodStack` at the coordinates you provided!

Repeat

Now that you've built a simple structure with MineBuilder, let's try something a bit more complicated with the `repeat <var> starting at (<x>,<y>,<z>) for <int> times, <int> apart in <dir> direction` command! `repeat` builds a given structure a given amount of times, with a given amount of space between iterations, in a particular direction. Just like `build`, `repeat`, needs a previously defined structure, and some coordinates. However, it also needs to know how many structures you would like, the amount of space between structures, and the direction subsequent structures should be built. For our next example, let's make a row of houses using `house.mpb`, and `repeat`.

```
define house as
{
     // this statement grabs blueprint code from the specified .mpb
     load house.mpb
}

// builds the previously defined structure at the given coordinates
repeat house starting at (115,79,-244) for 4 times, 3 apart in z direction

end
```

After you run this program, you should see a row of houses at the location you specified.

End

The `end` command signals that the program has finished execution. It will write the result of `build` and `repeat` commands to a Python file, `mine.py`, which is then executed using the operating system's Python interpreter.

And that's it! The above four commands are all that are necessary to begin to write and build MineBuilder programs. Note that MineBuilder programs are of the `.mc` file extension (`.mc` for **M**ine**C**raft.) What follows are more screenshots of example structures built using MineBuilder!