Wonderland Development
Part 1: Basic Concepts

These tutorials aim to explain everything simply and clearly, with as little tech-speak as possible (the tech-speak will be introduced as we go along). Wonderland is an incredibly complex system that only real propellor-heads truly understand – but we, the users who want to build our own virtual worlds with the Wonderland toolkit, don't need to know all that.

Part 1 covers the basic concepts at the heart of Wonderland development. Part 2 will look in more detail at the concept of modules in Wonderland. Modules are what you'll spend most of your time on as a Wonderland developer.

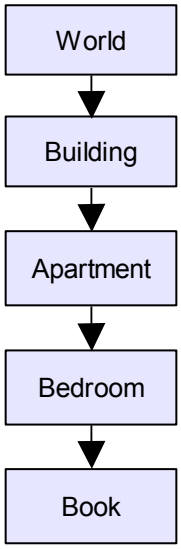
Please note that these tutorials owe a great debt to Jordan Slott's set of tutorials on module development at https://wonderland.dev.java.net/index.html, and also to the help and patience of Nicole Yankelovich, Jonathan Kaplan and the rest of the Wonderland development team :)

1. The Building Blocks

Imagine you are lounging on your bed, reading a book.

- Your book exists inside your bedroom
- Your bedroom exists inside your apartment
- Your apartment exists inside your building
- Your building exists inside the world

This can be represented as a graph.



This graph shows the spatial relationships between objects.

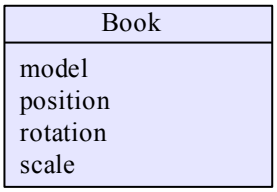
World contains *Building*, which contains *Apartment*, and so on.

In 3D computer graphics, this is called a SCENE GRAPH.

The *scene* is everything that exists in the world, including the world itself.

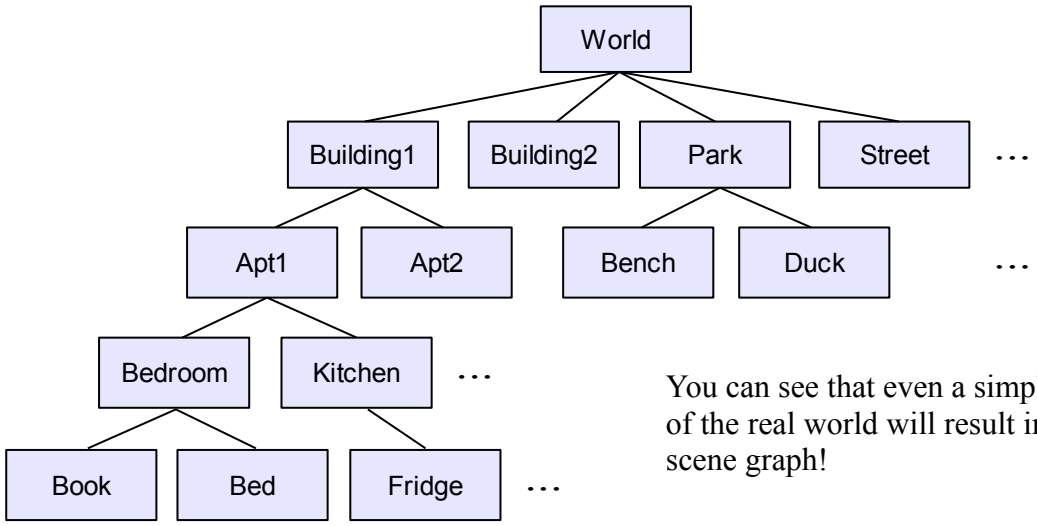
We can think of each object in the scene as a *node* in the scene graph.

A node is simply data that represents an object in the scene. For example, the *Book* node might consist of the following data:



- a **3D model**, which is data about what *Book* looks like (its shape)
- **position -** where *Book* is located in the scene
- **rotation** – how *Book* is oriented
- **scale** – how big or small *Book* is inside the scene

A scene can obviously be much more complex:



You can see that even a simple representation of the real world will result in a very complex scene graph!

1.2 Wonderland Cells

A scene in Wonderland is also represented by a scene graph.

But in Wonderland, a node in a scene graph is called a CELL. A cell is simply data that represents a visible object in the scene.

The name “cell” is less mathematical and abstract than “node” (which is used by mathematicians and 3D graphics programmers). The word *cell* also makes you think about a small volume of space (just like a prison cell).

So, a cell is where we store data about a visible object in a scene. This visible object can be either 3D (eg. a box), or 2D (eg. a 2D window, perhaps for displaying an image or a 2D application such as a web browser). This handles the data for an object's shape and transform (*transform* is basically a mathematical term for position, rotation and scale).

1.3 Wonderland Components

But an object can also have other features, such as interactive behaviour, and the ability to communicate with other parts of the Wonderland system.

For example:

- **interactive behaviour** – an iPod object can *play* music when *clicked*
- **communication** – an iPod object must *communicate* with Wonderland's audio subsystem (jVoicebridge) to play a sound file

An object may have other types of features, too.

One important feature that an object might have is that it can be moved. The object is said to have a *moveable* feature - a ball might be moveable, while a house would usually be static (unless you tie a bunch of balloons to it).

Moving something in the real world is easy (just give it a push), but moving an object in a virtual world requires quite a bit of program code.

It makes sense to be able to share this code among different objects, eg. a ball and a wheel.

The code to make an object moveable can be attached to a ball cell, and also attached to a wheel cell. This will give the *moveable* feature to the ball and wheel cells.

We might also want to have the ball make a bouncy noise as it moves – this requires the ball to communicate with the audio subsystem to play a sound file.

So, we can simply add a feature that enables the ball to communicate with the audio subsystem. The ball cell would then have two features: moveable and audio playback.

In Wonderland, these features are implemented as COMPONENTS. A component is simply program code that can be shared among cells.

When you create a new Wonderland virtual world, you will spend most of your time creating new cells, and perhaps new components. Wonderland already contains many components, which can be used to add different types of features, or capabilities, to your cells.

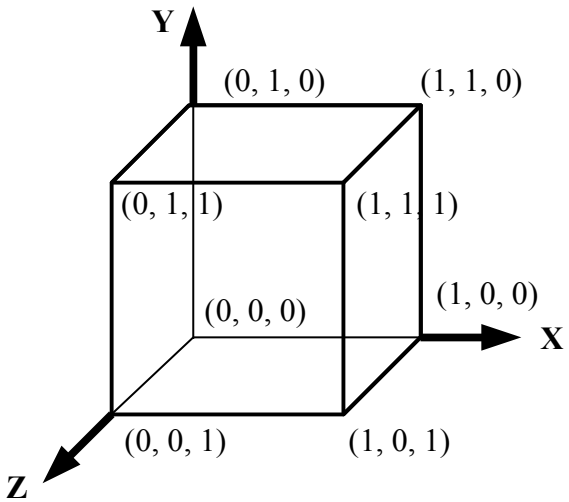
But if an existing component doesn't provide the feature you want, you'll have to create a new one. The new component must then be added to your cell. (Or any other cell that might need such a feature – components are made to be shared!)

1.4 Wonderland Renderers

Things are a little bit more complicated, though.

Data is just that – data. You can't actually *see* data. Data exists inside the computer memory as a collection of numbers. For example, to represent a box (cube), all we need is the position of each corner.

In 2D, we use the X and Y coordinate axes. In 3D, we have an extra axis for the third dimension, called Z. Wonderland uses what graphics people call a right-handed coordinate system, where X points to the right, Y points up, and Z points out from the screen, as shown below.



So, each corner needs an x, y, and z value, eg:

(0, 1, 1) ⟶ (x, y, z)

So, we need 8 x 3 = 24 numbers to represent a cube in a scene. (It's actually more complicated, but this is the basic idea.)

This data is stored inside a cell.

But we still can't see anything yet!

Graphics data, such as the data for the box, must be *rendered*.

To render something means to make it visible on-screen. This involves a lot of complex maths and graphics code, that luckily we don't have to worry about.

Wonderland includes a rendering subsystem, which is used to display the cell's object on-screen.

This is usually called a *rendering engine,* or *graphics engine.*

The rendering engine has an API that enables you to execute its graphics commands. Wonderland uses a rendering engine called jMonkeyEngine, which is also used to make games.

So, we must use the graphics engine API to actually display the object in a cell.

The rendering code for the cell is written inside a CELL RENDERER class that we create.

1.5 Wonderland Resources

Cells, components and renderers are all created with computer code. Wonderland also makes use of other resources, which are created using different tools.

The most common types of resources are 3D models, animations, images, and sounds, i.e. all the artwork used inside a Wonderland world.

These are usually created in tools such as Maya, Photoshop, Audacity, etc.

For example, all the Wonderland avatars are 3D models, with animations for the various gestures and facial expressions. These models have texture files applied, which are images that specify the surface appearance of an avatar (e.g. t-shirt design, skin details).

2. Wonderland Modules

So, simply put, a Wonderland world consists of:

- Cells, that represent visible objects
- Components, attached to cells, that add specific features to cells
- Cell renderers, that use the graphics engine API to make cells visible on-screen
- Art and other resources

Once these have all been created, they need to be packaged in some way for easy upload to the server.

This is done using a Wonderland MODULE.

A module is basically a wrapper file that contains all the Wonderland building blocks according to a specific structure. This structure lets the server know where all the different buiding blocks are.

A module is actually a Java jar file - but we'll look at the details in Part 2.

3. Server and Client Programs

Before creating a new cell, there's just one more thing we must understand: how Wonderland is shared among many different users.

If lots of people are visiting the same virtual world, we have to make sure that everything in the world is the same for everyone.

For example, if you're playing a game inside Wonderland and your friend tells you to pick up a health pack, you both have to be able to see the same health pack, at the same time, in the same location. And if someone else comes along and takes the pack, everyone else in the game has to know that the pack is no longer available.

So, Wonderland has to keep track of all the objects in the world.

This is very difficult – but Wonderland has been designed to make it easy.

Wonderland is an example of a **client-server** application.

To see what this means, we'll look at some of the main requirements for a virtual word.

- Every user in the virtual world must see the same things as everyone else.
  If Nicole and Jon are in a room together, and Nicole sees a book in the room, then Jon must see the same book in the same location.

- If anything changes, everyone should be kept up to date.
  If Nicole picks up the book, Jon should see that it is now no longer on the shelf, but is being held by Nicole.

- Users should be able to connect at any time and see the current state of the world.
  If Jon has to leave the world for some reason, and while he is gone Nicole puts the book on a table, when Jon visits the room again, he should see the book on the table (assuming it hasn't moved since then).

- Users do not need to know anything about the physical location of any other user in the *real* world.
  Nicole does not need to know that Jon is really in San Francisco or that another user is in Singapore, nor do they need to know that Nicole is in Massachusetts. They do not need to communicate directly with each other in the real world. Their computers also do not need to communicate with each other directly.

Wonderland is split into two parts, a *server* program, and a *client* program (we say that Wonderland has a client-server *architecture*) :

Server

- The virtual world is stored on a computer at a particular IP address. The actual address is defined as a URL.
- The server knows everything about the virtual world: it stores all the objects that will be displayed, controls how different parts of the Wonderland system communicate with each other, and knows which users are inside the virtual world.

Client

- The client is on another computer, at a different IP address.
- The client must log into the server to join a virtual world.
- After the client logs on, the server sends a copy of the virtual world to the client. (Actually, the client can cache data locally, but on entering a new virtual world, the data is sent to the client by the server.)
- Note that there can be lots of different clients, each running on a different computer. There can also be more than one server, though this is rather complex and won't be discussed here.

This isn't difficult to understand – in fact, it's a lot like mail-order shopping!

To buy something mail-order, you must send your request to the mail-order company's address. The company then sends the items you ordered to your address. You must know the company's address, and they must know your address.

You are a **client**, asking the mail-order company to *serve* you (give you something). So, the mail-order company can be thought of as a **server**.

It's the same with Wonderland.

Your computer is the client, asking the Wonderland server to send you all the data you need to display and interact with the virtual world.

- The client is the program that you run on your computer – a Wonderland client.
- The server is the program that runs on the other computer – a Wonderland server.

If another person (say, Jon) wants to join you in the virtual world, they must log on to the same server as you, and request their own copy of all the virtual world data.

Now, you both have the same virtual world displayed on your computer.

If you change something about the world, e.g. move a book, Jon needs to know what happened (or he'll get confused).

So, when you pick up the book, four things are done:

- your client sends the new position of the book object to the Wonderland server
- the server updates its own copy of the book object
- the server then sends the new position of the book to Jon's client program
- Jon's client updates its own copy of the book object, and displays it in its new location

Now everyone is up-to-date.

All this communication is handled by Wonderland.

But we must still write code to plug our own cells into the Wonderland communication system. This is also packaged inside a module, along with everything else.

We'll look at how to do this later.

4. Review

So far, we've covered the basic building blocks of Wonderland worlds:

- Cells
- Components
- Renderers
- Other Resources

We saw how these building blocks are packaged into modules.

Finally, we explored the client-server nature of Wonderland.

In Part 2 we'll take a more detailed look at modules, and in Part 3 we'll actually start looking at some code! But before creating a new module, we'll examine an existing module and how it is compiled, built and uploaded to the server. Once the actual coding and building starts, a poor understanding of the details can cause a fair bit of confusion and frustration. I've been there, so will try to help you through it. Modules are a very powerful feature of Wonderland, so it's worth taking the time to really understand them well :)