Muse Introduction

Adapted, by rowrbazzle, often verbatim, from part I of

Muse Manual C, Macintosh Version,

Written by Falryx, edited and Macintized by Kismet

...expressly for use in university technology-in-education classes, where students need to experience MUSE in a very limited amount of time -- crucial to get people over the beginning stumbling blocks _quickly_, ready to explore, interact, and build.

The original Falryx/Kismet "Muse Manual C" is online at

http://www.musenet.org (cited 30 Sep 2001)
Introduction

Welcome to Muse

Welcome to the wonderful world of MUSE: Multi-User Simulation Environment. This handout was inspired by the unusual MUSE needs of classes in Technology in Education, in which students need to experience the MUSE in a very restricted timeframe. It draws heavily on Falryx’s Muse Manual C as edited and Macintized by Kismet, downloadable from http://www.musenet.org [cited 22 Aug 01] In generating this adaptation, the idea is to clarify, for people in a hurry, introductory material in Falryx from the absolute basics to ventures into intermediate MUSE programming. For more advanced programming information, refer to Muse Manual C by Falryx and MUSE Programming Guide v1.4 by Wyvern, both downloadable from www.musenet.org.

Caveat: rowrbazzle is only an advanced beginner to MUSE. This documentation effort has two goals: to help others past the stumbling blocks I encountered, and to help myself deepen my own MUSE understanding.

IMPORTANT! Command syntax conventions herein

At times you will see MUSE commands written in a "computery" typeface. These will employ some special symbols to concisely specify options. These symbols are not part of the actual commands. For instance:

```
help <command | topic>
```

describes a command which asks for help on either a command or a topic (the '|' means "or").

```
help {command | topic}
```

is subtly different; it describes the complete MUSE help command, in which the command or topic is optional; it's OK to simply ask for

```
help
```

In other words, the {} mean that whatever's enclosed in them is optional. Here are the special character meanings:

- <> enclose required information,
- {} enclose optional information
- | (a vertical bar, typed Shift-backslash) means OR, in the usual sense of the word. In the example above, you could type either of the following:
  
  help <command>
  help <topic>

- () enclose command abbreviations, as in

  
  look (1) <object> ...could be typed as
  1 <object>

  
  When typing in a command according to the above syntax - do NOT include the <>'s, {}'s, or ()'s. Their use in examples and help pages is only a command documentation convention. (Curly braces are sometimes required in MUSE code itself. Not to worry -- by the time you run across any in code, they won't be ambiguous.)

Actual examples, using the very useful help command:

```
help look ...look is a command,
help happy fun ball ...happy fun ball is a topic
```

There's a complete section about MUSE help coming in a few pages.
Part I: Basic MUSE knowledge

The Environment: What’s around you?

MUSE takes place in an alternate, virtual reality, where you are 'transported' via your computer into a world of text, where vision and imagination commingle to cover the jobs that normally five senses do. When in MUSE, players are connected to a computer holding the MUSE database, full of all the objects: characters, rooms, exits and things that currently exist in the MUSE.

Characters are virtual reality extensions of you, the player. It is the characters of the MUSE who are able to manipulate the environment and interact with one another. MUSE supports a multi-player/character setting where many users can connect simultaneously to the server and partake of the virtual reality together.

Rooms are 'places' within the virtual reality, the MUSE version of actual physical locations. They may represent any space, of any size, that represents an actual 'physical' location where your character can go. Examples: a meadow bursting with wildflowers, a single cell within your body, a many-dimensional mathematical space, or even a plain old room in a house...

Exits are one-way "connectors" leading from one "place" to another. They can be the cyber-version of physical exits (doors, windows, hallways...) or utterly fantastic. You can use exits to represent anything one might pass through: Alice's Looking Glass, a wormhole from "Star Trek", to a catapult which launches you through the sky to land at your new location...

Things (aka objects). These are things (furniture, toys, tools...), limited only by your imagination. They are typically manipulated to make them unique, or do things, or just look nice -- as you'll soon see!

Logging in to the MUSE

Start the MUSE program by double-clicking the MicroMUSE icon in the MUDDweller folder (assuming you've downloaded the MUDDweller client from http://www.musenet.org. Other clients, including Windows clients, will work similarly.

See the Useful Documents Appendix for specific downloading information.

To log in as a visitor, type

connect visitor <proposed-character-name>

If your proposed name is not already in use, you'll see the login screen shown on the next page (if the name is taken, you'll have to pick another one).

To log in as your student character, type

connect <your-char-name> <password>

and you'll see a screen much like the screen two pages hence

In either case, after the "splash" login screen appears, click below the double line to get a blinking cursor, and type your connect command (below that double line is where all your typing will go.

For the visitor example (next page), you see the login screen, some statistics, a description of your current location. Visitors are immediately "whisked away" to a short introductory tutorial -- take it!

For the student example (two pages hence), it shows login screen, some statistics, a description of your current location, followed by a brief announcement.
Hello! MicroMUSE is our vision of the 24th century, a blend of high technology and social consciousness with emphasis on education, concern for the environment, and communication. Our charter is available online or by anonymous FTP to ftp.musenet.org.

*** end of messages ***

connect visitor alskdf


<table>
<thead>
<tr>
<th>Date</th>
<th>&lt;Topic&gt;</th>
<th>&lt;Story&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/11/10</td>
<td>cmr</td>
<td>move</td>
<td>SERVICE INTERRUPTIONS</td>
</tr>
<tr>
<td>07/24/99</td>
<td>admin</td>
<td>visit</td>
<td>Visiting Room</td>
</tr>
<tr>
<td>09/22/99</td>
<td>admin</td>
<td>quota</td>
<td>Quota Increase</td>
</tr>
</tbody>
</table>

To read the news, type: 'news'

Cyberion City Main Transporter Receiving Station

The bright outlines of the Cyberion City Transporter Station slowly come into focus. You have been beamed up here (at considerable expense) from one of the Earth Transporter Stations. You are among the adventurous few who have decided to visit (and perhaps dwell in) Cyberion City, the largest space city in the solar system.

You are welcomed by the transporter attendant, who gives directions to all newcomers to this space city.

Contents:

Attendant

Welcome to MicroMUSE; your name is V.alskdf
Welcome to MicroMUSE, V.alskdf!
You are using 217 bytes.
You are whisked away to the tutorial.

Visitor's Trail - Syntax 1
This is the start of a tutorial trail that will introduce you to MicroMUSE and help you get started. The first thing you will need to learn is how to use the trail.

At various points in the trail, you will be told to type something. Sometimes that 'something' will be in upper case, LIKE THIS; sometimes it will be in quotes, 'like this.'

If that 'something' is set off in quotes, do NOT type the quotes. Type only what is contained between the quotation marks.

For example, type 'knock.'
Welcome! MicroMUSE is our vision of the 24th century, a blend of high technology and social consciousness with emphasis on education, concern for the environment, and communication. Our charter is available online or by anonymous ftp to ftp.musenet.org.

### end of messages ###

connect rowbazzle Visiting


------------------------------- Top 3 News Items -----------------------------

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Story</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/11/00</td>
<td>cne</td>
<td>move</td>
<td>SERVICE INTERRUPTIONS</td>
</tr>
<tr>
<td>07/24/99</td>
<td>admin</td>
<td>visit</td>
<td>Room</td>
</tr>
<tr>
<td>07/22/99</td>
<td>admin</td>
<td>quota</td>
<td>quota increase</td>
</tr>
</tbody>
</table>

------------------------------- To read the news, type: 'news' -----------------------------

Last login: Tue Aug 21 23:18:12 2001

Your Student character expires in 60 days, 1 hour, 39 minutes, 25 seconds.

rowbazzle's room(*10237Rj)

...a place for experimenting with peculiar gizmos and phenomena(one of which is MUSE itself, of course!) -- People exploring MUSE programming might examine things -- most are 'visible' (but bear in mind that things are "under construction" by a beginning programmer :-) Right smack in the middle of the room is a sparkling brand-new-looking bicycle wheel, hanging about chest-high on a strong string tied to the axle (go figure...!)

Contents:

- Bicycle wheel(#10115ev)

Obvious exits:

- Out

You are using 2640 bytes.

Please read 'news renew' for information about becoming a Member of MicroMUSE.
**Description of your current location** has four parts

- **its name**: Cyberion City Main Transporter Receiving Station or rowrbazzle's room, e.g.
- **its description** (often abbreviated to desc) of the room. This is written in an attempt to give you a visual representation of the room, to help you picture just what the room looks like. Often, the desc will contain very handy information (for example, where the exits go, or suggestions of interesting activities)
- a list of its **contents**. This list tells you what is in the room. In rowrbazzle's room above, you see only one item, bicycle wheel(#10115ev).
- **the obvious exits** in the room. In this case 'Out'.

**Interacting With The Environment**

Looking/Examining: what's out there, and how do I see it?

Since MUSE is a text based virtual reality, the only way you can interact with it is "visually" via text. In the style of old adventure games the command to look at an object is (not surprisingly): look.

**Look** by itself provides you with the description of your current location, This gives you excellent information about your surroundings. "I".

The command syntax is:

```
look (l) { thing | room | person | exit}
```

Some shortcuts:

(recall that the the "(l)" means 'l' is short for "look").

- look me ...shows you your description
- look here ...shows description of your current location

You can also abbreviate the object being looked at to anything unambiguous. For example, in rowrbazzle's room, you could look at the bicycle wheel in (at least) two ways:

```
look bicycle wheel
```

**Examine** will show you different things depending on the object you are examining. Syntax:

```
look (l) <thing | room | person | exit>
{<attribute>}
```

(that shoulda been all on one line).

If you examine an object you don't own (for example: examine shkoo) then you will be shown a much abbreviated list of information. If the display is long, and you are interested only in a single part of it (and that part is an attribute), you can append its name, as in

```
look here/desc
```

to see just the description of the current location

**Communicating with other players**

Now that you can see them, how do you talk to other people?

say lets you speak into the room you are in and to the people around you. Syntax:

```
say (") <message>
```

"Say" has a two-part action; e.g. if you type

```
say hello!:
```

- you see: You say "Hello!"
- the people in the same location as you see:
  
  ```
  Your_Name says "Hello!"
  ```

Shortcut: type " instead of say:

```
"hello! ...results in
You say "hello"
```

**pose** lets you express gestures, textually. Syntax:

```
pose (:) <action>
```

"Pose" has just a one-part action; e.g. if you type

```
pose bursts into laughter
```

...everyone in the room will see:

```
Your_Name bursts into laughter
```

Shortcut: type ':' instead of pose:

```
:bursts into laughter ...results in
Your_name bursts into laughter
```

**whisper** and **page** are similar to say, except you get to specify which single player will see it. "Whisper" works within the same room as you are, while "page" allows you to send a message to someone anywhere in the MUSE. Syntax:

```
page <player> = <message | :action>
whisper <player> = <message | :action>
```

When you page/whisper someone you can either choose to page or whisper to them with a message or an action. Here are some examples of what some page and whisper commands might look like to the sender (bazzlerowr, in this case, ("rb" is
an alias for rowrbazzle):

```
wh rb=hi there
You whisper "hi there" to rowrbazzle.
```

```
page rb=hi there
You paged rowrbazzle with "hi there".
```

```
wh rb=:struggles mightily
You whisper-posed rowrbazzle with "bazzlerowr
struggles mightily".
```

```
page rb=:is cruising
You page-posed rowrbazzle with "bazzlerowr
is cruising".
```

…and Rowrbazzle saw the following:

```
bazzlerowr whispers "hi there"
bazzlerowr pages: hi there
bazzlerowr whisper-poses: bazzlerowr
struggles mightily
bazzlerowr page-poses: bazzlerowr is cruising
```

**@emit** Unlike the previous four commands, which plug your actual message into various sentence forms, @emit shows everyone in the room exactly what you typed, without change. One exception: @emits beginning with a player's name will not work. Syntax:

```
@emit <message>
```

@Emit is generally not used interactively, but it's a mainstay of object-building/programming, however. It has several variants, with related behaviors. Well-programmed objects often rely on thoughtful exploitation of the various available sentence forms.

Think "affordance" of each command!

**Exploring things**

To delve deeper into what's around you, see if you can pick them up. Try this:

```
take <object>
```

If it works, as it often will, you'll see some interesting response. If it doesn't, it'll just say so (rather tersely, I thought):

```
You can't pick that up
```

To put it back down,

```
drop <object>
```

Two very important caveats:

- never try to take or get players -- very rude!
- when you pick something up, you're carrying it, meaning that if you go to another location, you'll bring it with you (in effect, stealing it). Not only rude, but it can do damage to others' work by trashing their object arrangement.

Having picked something up, do a

```
look me ...shows that you're carrying it, and

look here ...shows that it's no longer "in the room"
```

There are some _objects_ which you will be allowed to enter (like a car, or bus). These are typically vehicles of some sort, or perhaps a small building or structure (dragon, phone booth, elevator, etc.). Getting in and out of these objects is relatively simple:

```
enter <object>
```

You won't be able to enter all objects. You may always enter an object you own (unless you specifically lock it _against_ yourself). No one else may enter an object which you own unless it is set ENTER_OK (e). [Flags are described in more detail later on in this manual.]

Leaving an object is simple:

```
leave
```

This command does not work in rooms, only in objects that you are in. This essentially forces yourself to be 'dropped' from the object (picture what would happen if a car I was in typed: drop Falryx -- I'd be dropped just like you can drop an object you are carrying). Leave is useful only for leaving objects which you have entered.

Lastly, there is a quick way to move to a specific room called your "home". Your "home" is generally a private room where you have some of your more treasured belongings, etc. The home on a MUSE is probably very similar in function to your bedroom in your real life. To get to your home you type the command "home".

```
home
```

When you log in to a MUSE as a new character, your home is automatically set to a specific room - generally a public area that doesn't get too much traffic, but is near anything important. Once you have made yourself a new home, or if someone offers you a place to call home, you need to use the @link command. This command will change your "home" for you.

```
@link me = <here | room database reference number>
```

The @link command is used for a lot of things, but for now we will just assume it changes your home, and wait until later to discuss @link more in depth. You may @link yourself to any room that you own, or to any other room on the MUSE that has the LINK_OK (L) flag. Like the JUMP_OK flag, the LINK_OK flag will display the room number of the room next to its name so you have an easy reference:

```
A Room(#2345RL)
```

If you change the room your are @linked to then that becomes your new home, and the "home" command will automatically take you to that room.
Moving around

There are (at least) three ways to navigate in the MUSE:

- walking from room to room,
- jumping to any area or neighborhood,
- teleporting to any specific room.

Also, there's the matter of knowing where you are and where you might be going.

Walking -- When you look at a room, the display gives a list of Obvious Exits at the end. To walk through an exit, simply type it if it were a command. Syntax:
<some-exit>

When you see exits like the following, with stuff in < >,
Science Center <SC> <P>ort <B>oard Out
...the enclosed stuff are shortcuts, so type
sc or p or b

Jumping -- If you know the general area you'd like to reach, in the form of a sector/arc pair, you can go directly there. Syntax:
jump <sector>/<arc>

For instance, jump 1/2 will take you directly to the front door of the Science Center. See the Navigation section just ahead for a description of what sector, arc, port, starboard, etc. mean.

Teleporting -- If you know the specific location you'd like to reach, in the form of a data base reference number, you can go directly there. Syntax:
@teleport (@tel) <room database reference #>

For example,
@tel #9999
will take you to the Help Desk in the Cyberion City Hyatt Hotel.

Navigation -- Aslan's "Geography" of Cyberion City may help you orient yourself (first, here's a feeble picture ;)

"...Cyberion City is a gigantic cylinder. From flat end to flat end, it measures 36 km... [about 22.5 miles] The central axis of the cylinder -- which we will refer to hereafter as simply the "axis" -- is the line between the centers of the two flat ends... The circumference of the cylinder... around the curved surface is 48 km [about 30 miles]...

People live and work mainly on the inside curved surface of the cylinder [a truly giant tin can!]. Standing on this surface [possible because the cylinder is spinning clockwise -- "spinward"], what would you see? Looking up, if there were no internal walls or ceilings and the visibility were good, you would see straight across to the curved surface halfway around the can, 15 km [about 9 miles] away. If you were facing one of the flat ends of the City, you would see it. Behind you would be the other flat end; and the gently sloping curved surface would be rising to your left and right.

Because the City is spinning, there is a constant push towards the outside curved surface, which, if you are standing on it, feels exactly like the push of gravity on earth... Thus, you can walk in two dimensions -- just as you can on earth. On earth, we call one of these dimensions north and south, and the other dimension east and west.

In the cylinder, there are new names for these dimensions. One flat end of the cylinder has been... designated the Port side; the other... is the Starboard side. The cylinder is divided into twelve smaller cross-sections [aka sectors or sections]. Each Section, numbered 0 through 11, is 3 km [about 2 miles] wide. Section 0 is on the Port side, section 11 on the Starboard side."

In our picture above, the port side is on the left, and evidently we drew only the first three sectors, nos. 0, 1, & 2, numbered across the top. Continuing with Aslan's "geography":

"You can walk from one end to the other, passing through the sections in turn... [along the Port--Starboard dimension]. You can also walk around the curved surface of the cylinder. As you walk, you start from the Main Transportation Strip (MTS)... [running from end to end], and walk either [in] the direction in which the cylinder is spinning, referred to as the
Spin direction, or Spinward, or in the opposite direction, called Antispin or Antispinward. The second dimension is thus Spin and Antispin. The cylinder is divided into eight Arc sections. If you walk Spinward, you pass from the MTS to Arcs 0, 1, 2, etc., through Arc 7, and back to the MTS (you've been walking in a circle, remember)... [hence the <sector>/<arc> location format!]


Sections I and II are well worth reading in their entirety. The remainder is more technical; great for high-school math teachers etc.

Getting MUSE help

Online Help

Type help to see the main help display, describing the major help subsections you can explore by typing

- help commands ...shows a good command list, plus some important command subsections
- help token commands ...shows a list of very handy command shortcuts for say, pose, etc.
- help topics ...shows a list of topics
- help membership ...explains how to upgrade your student character to full MUSE membership.
- help <list-entry> ...shows help on specific items from the above lists.

You can carry two very handy helpers around with your character by typing @add me=#5665 for everyday advice, and @add me=#5373 for a map of sector and arc contents. Having done so, you can then..type helper! (including the '!') to see a list of advice topics; and map 1 (or 2, or 3... through 7). These are the best concise source we know of for info on "what to do" with yourself and everything around you in the MUSE.

Here are the advice contents:

helper!
Topics are:
me!
objects!
objects2!
objects3!
rooms!
lost!

places!
more!
more2!
important!
spam!

Type one to read about it. To get rid of this parent, type '@del me = #5665'.

map 2
Main Transportation Spine <Section 2>
<Sec 2 - Arc 0> Cyberion Information Center
<Sec 2 - Arc 1> Communications Complex
<Sec 2 - Arc 2>
<Sec 2 - Arc 3> Communications Companies
<Sec 2 - Arc 4> Broadcasting Services
<Sec 2 - Arc 5> Movie Studio
<Sec 2 - Arc 6> Robot Housing
<Sec 2 - Arc 7>

WWW-based Help

http://www.musenet.org has a lot of downloadable material
To save your browsing time, we particularly recommend those listed in the Appendix

Particularly Interesting Places to visit

- #9999... Help Desk, has some tutorial stuff
- Science Center 1/2 ...a bunch of interesting "exhibits"
- <Sec 5 - Arc 5> Puzzles - Entertainment Entrance; walk through Puzzles, take the Turbolift, and walk into Cannons. This is a splendid piece of MUSE creative "writing", especially if you enjoy puzzles about motion.
- the University (<sec 1 - arc 5>, often written "1/0") ...help on building & programming: "rooms to rent"
Part II: Basic Building and Programming

Building - the basics

One of the extreme beauties of MUSE (and other similar network environments) is the ability to create your own additions to it. You are empowered to leave your ‘mark’ on the MUSE, for better or worse. How you choose to do that is up to you, but one of the most common ways is to build interesting things.

Builders create objects, rooms, gadgets, environments, whole cities, etc. Without people to Build, a MUSE is doomed to a small database and a short life. It is the users who, through their contributions, ensure the continued survival of the MUSE.

Building Things

To create a thing (an object),

first type in:

@create <thing name> , as in

@create gizmo

*gizmo(#9759v) created.*

This asks the MUSE to make space for a new thing with the name you give it. MUSE reports that it has reserved this space with its 'dbref' (database reference number).

Second, because you are automatically "carrying" newly created thing, you should put it down so it's "in the room" (and can be looked at):

drop gizmo

Dropped.

Only now will *look here* show the new thing:

look here
rowrbazzle's room(#10237RJ)
...a place for experimenting [...]

Contents:
gizmo(#9759v)
 [...] 

Third, comes the fun part (harder, to be sure) -- transforming this bare gizmo into something interesting -- responsive to its environment. Here are a few basic ways to tailor your new object; by setting the following attributes:

@desc -- text that describes the object to whoever looks at it (a few sentences)

@succ -- text to display if someone (you) successfully picks it up (with take or get)

@drop -- text to display when someone carrying it (having taken it) puts it down

@fail
@desc
@lock

FOR NOW, WE'RE DEALING WITH PLAIN OBJECTS - THESE ATTRIBUTES CAN GENERALLY BE SET ON ROOMS, EXITS, AND PLAYERS (ALL ULTIMATELY OBJECTS). THERE ARE A GREAT VARIETY OF POSSIBLE ATTRIBUTES -- NOT TO WORRY: LOTS OF VERY NEAT STUFF CAN BE DONE WITH JUST THESE FEW!

IT'S ALSO EASY TO GET OBJECTS TO LISTEN TO THE ROOM, AND REACT WHEN SPECIFIC WORDS OR PHRASES ARE HEARD

Here's Falryx's excellent example of @creating a simple object (see the Appendices for the actual transcript of building, and two characters playing with the elephant):

@create An Elephant ...(object names can include spaces)
An Elephant(#445) created.
drop An Elephant
Dropped.
@describe An Elephant=This is a large grey elephant. As you watch the great beast, it sways back and forth in a lacadazical rhythm on its ponderous journey to a nearby stream. The elephant's large grey trunk snak... 
Description set.
@odesc elephant=looks at the big elephant.
Set.
@lock elephant=me 
...(this means only you can pick up the elephant)
Locked.
@succ elephant=You pick up an elephant! Wow, you must be strong!
Set.
@osucc elephant=picks up an elephant! Watch out!
Set.
@odrop elephant=teeters back and forth under the weight of a large elephant and drops it with a loud *THUD*
@drop elephant=The elephant gets very heavy, so you decide to drop it.

@fail elephant=You can't pick up an elephant!!

@ofail elephant=tried to pick up the elephant, who doesn't even notice.

This is a typical way to create a simple object. By addressing the @lock, the Fails and the Sucess we have thoroughly made the elephant so people will always get a reaction that is unique for the elephant. If you are on a MUSE now, try creating something like the elephant, and try out some of the commands. If there are people in the room with you, ask them to try taking the elephant so you can see the 'ofail' or 'osucce' message.

Rooms

Part V: Appendices

Glossary

! event - ! events match what they hear and when a given pattern matches the one given by the ! event, actions are taken by the object on which the ! event resides.

$ event - $ events are essentially new commands or verbs. $ events match what is typed in (as opposed to what is heard) and if matched properly execute commands as dictated by the object the $ event is on.

Action - Used in the manual to refer to 'commands' which an object would do after an event was triggered.

Attribute - Attributes are registers, attached to MUSE objects, in which you can store data or commands. There are built-in attributes, such as 'succe', 'osucce', 'fail', 'ofail', etc. These can be set on any muse object. The @defattr command can be used to define additional attributes on particular objects.

Character - These are the people who are wandering about the MUSE with you. People like 'Falryx' and 'shkoo' are characters.

Child - An object which has a parent (and therefore has inherited any number of attributes, commands, ! events and functions from the parent).

Class - Classes are ways to separate the users of the MUSE from the officials who oversee the smooth running of the MUSE. In general someone with a class of 'Director' is a head honcho.

Database Reference Number (dbref) - The number appearing after the names of objects. Each number is unique to that object and allows MUSE to properly store thousands of different objects, even with the same names. This number can almost always be used in place of the name of an object.

Exit - An 'object' in MUSE which allows you to move from one room to another. Flag - Everything in the universe of MicroMUSE (Rooms, Exits, Objects, Players, etc...) are represented in the same way at the program level. A room merely has the room flags set and a player has the player flags set. In addition, flags also give objects abilities or qualities.

Function - Functions are routines which can be quickly evaluated by MUSE to return values based upon their function. (i.e. the 'add()' function will add two numbers together and return the new value)

Lock - A lock is an attribute that determines who may take, use, enter, or otherwise control an object.

Parent - A parent is an object which has children and which has been made specifically for the purpose of acting as a 'central reference point' for its children.

Player - The person in control of a character. 'Falryx' is a Character. 'John Doe' is a Player. At times this distinction is
blurred within MUSE. (for example the 'Player' flag is set on characters, not the players. :))

**Power** - A special permission to be able to use commands or do specific actions which might not otherwise be allowed (i.e. spoofing).

**Puppet** - An object with the 'PUPPET (p)' flag set. Puppets transmit what they hear to their owners, acting as 'ears from afar'.

**Room** - An object with the 'Room' flag set. Rooms can represent any number of various locations - out of doors, in a cave, your bedroom etc. In MUSE, rooms are the backbone of the structure.

**Thing** - Any object which is not a player, an exit or a room is a thing. Things include tables, zone objects, pet dragons, etc.

**User-Defined Attribute** - An attribute which has been defined by a user to have a certain name. User-defined attributes may also have various options set upon them which modify their behavior and utility.

**User-Defined Function** - A function created by a user. UDFs may be used only by the object they are defined upon. If they are on a zone object and set inherit, they may be used in any room in at zone.

**Zone** - An object which acts as a reference point for a collection of rooms, unifying them into a greater whole. Zones may have functions, ! events and commands which are common to all of the rooms in the zone.

---

**Useful Documents from** [http://www.musenet.org](http://www.musenet.org)

- [http://musenet.org/tinyfugue.html](http://musenet.org/tinyfugue.html) **Download page for MUSE clients:** Towards the bottom of the page are get TinyFugue source for your own Unix client.
  
  For a Macintosh with direct Internet service, try MudDweller.
  
  For a Windows with direct Internet service, try The TUCOWS Collection. Do a search on "Mud Clients" for your platform.
  
  (the Sanx Client looks good at brief glance)

- [ftp://ftp.musenet.org/micromuse/MuseGuide](ftp://ftp.musenet.org/micromuse/MuseGuide) **"MuseGuide":** GENERAL GUIDE, CYBERION. CITY MAP. Skip down to **== BASIC NAVIGATION ==** (do a text-search for NAVIGATION in your browser). There're some helpful moving-around examples, PLUS details of what is where in Cyberion City. (Slightly outdated, but pretty good. A few places may have moved)

- [ftp://ftp.musenet.org/micromuse/CybCity.Txt](ftp://ftp.musenet.org/micromuse/CybCity.Txt) **"Layout of Cyberion City":** DETAILED CYBERION. CITY GUIDE. The first few pages give a nice physical description of our virtual world. Read about the tin can and the gerbil! Read the math only if you enjoy this type of technical description, and/or are interested in teaching high-school math/physics

- [ftp://ftp.musenet.org/muse/MusePG14](ftp://ftp.musenet.org/muse/MusePG14) **Programming Guide** by Wyvern -- for people interested in programming; assumes some prior programming experience, as from the Falryx manual

- [http://www.musenet.org/otherdocs.html](http://www.musenet.org/otherdocs.html) "Other Interesting Documents -- Including Muse Manuals " by Falryx, others

---

**The Elephant Example (first building example)**

our elephant is in rowrbazzle's room(#10237RJ): do @tel #10237 to get there and play with it!
ROWRBAZZLE talking:

@create An Elephant
An Elephant(#9762v) created.
drop An Elephant
Dropped.
@describe An Elephant=This is a large gray elephant. As you watch the great beast, it sways back and forth in a lackadaisical rhythm on its ponderous journey to a nearby stream. Its large gray trunk snakes around nearby plants, grabbing on for a bite or two.
Description set.
@desc elephant=looks at the big elephant.
An Elephant - Set.
@lock ele=me
Locked.
@succ ele=You pick up an elephant! Wow, you must be strong!
Message set.
@osucc ele=picks up an elephant! Watch out!!
Message set.
@drop ele=The elephant gets very heavy, so you decide to drop it.
An Elephant - Set.
@odrop ele=teeters back and forth under the weight of a large elephant and drops it with a loud -->> THUD!! <<--
An Elephant - Set.
@fail ele=You can't pick up an elephant!
Message set.
@ofail elephant=tried to pick up the elephant, who doesn't even notice.
Message set.
l ele
An Elephant(#9762v)
This is a large gray elephant. As you watch the great beast, it sways back and forth in a lackadaisical rhythm on its ponderous journey to a nearby stream. Its large gray trunk snakes around nearby plants, grabbing on for a bite or two.
take ele
You pick up an elephant! Wow, you must be strong!
drop ele
The elephant gets very heavy, so you decide to drop it.
bazzlerowr has connected.
bazzlerowr tried to pick up the elephant, who doesn't even notice.
take ele
You pick up an elephant! Wow, you must be strong!
drop ele
The elephant gets very heavy, so you decide to drop it.
BAZZLERO W R talking, in a separate login window: See if you can work out the sequence of interaction between the two characters:

take ele
You can't pick up an elephant!
rowrbazzle picks up an elephant! Watch out!!
rowrbazzle teeters back and forth under the weight of a large elephant and drops it with a loud --> THUD!! <<--

help <command | topic>
help {command | topic}
connect visitor <proposed-char-name>
connect <your-char-name> <password>
say (*) <message>
pose (:) <action>
page <player> = <message | :action>
whisper <player> = <message | :action>
look (l) { thing | room | person | exit } 
examine (ex) <thing | room | person | exit> { /<attribute> }
@emit <message>
get <object>
take <object>

the Object-oriented metaphor in MUSE.
Characters, Rooms, Exits, Things, :: Object, Object, Object, Object.

More specifically, there's the underlying generic notion of "object" or "entity", and their specialization into varieties of entity. Hence, a Room is

all entities have generic behavior -- the commands they "know"

Specialized entities have specialized behavior in addition (and sometimes in place of) the generic.

Like a wireframe rendering of a three-dimensional object, or an animal skeleton, or a web of lies!