Celestia

.CEL Scripting Guide

for Celestia, created by Chris Laurel

(version 1.3.1)

Written by Don Goyette, with contributions from the Celestia developers and forum members.
Editing by Selden Ball, Jr. and other Celestia forum members
Version 1.0
Last revised on December 14, 2003

Table of Contents

Introduction

Viewing This Document

If you downloaded this file (ie. not reading the HTML version on-line), please make sure you are reading the file with a program that fully supports the file type. For example, if this is the Word document (.doc) file, then you should be using Microsoft Word, Word Viewer, or OpenOffice to view this file. (**Warning:** The .doc file will **not** display properly in WordPad.)

If this file is the Rich Text Format (.rtf) file, make sure that the program you are using fully supports this format, so the file is displayed properly. For example, Windows WordPad works fine, but Windows Notepad will not display it properly. (**Note:** URLs (links) do not function in the RTF text file.)

Here are a couple of checks to make sure the program you are using is formatting and displaying the document correctly:

1) The heading page of this guide should contain a graphic image which includes the text "Celestia" on one line, ".CEL Scripting Guide" on the next line, with graphics of telescopes and some space objects above and to either side of the text.

2) Below, there should be two graphic images. The first is a Stop sign with an open hand in it, and the second is a sticky-note with a thumb tack in it.

If you see all of these images, then it should be okay for you to continue using the document display program you are currently using.

If you do **not** see these images, try a different program with known support for the file type you are trying to view.

Acknowledgements

First, I would like to thank Chris Laurel, and the entire Celestia development team, for creating such an incredible 3-D space simulation program. If it wasn't for Chris, we'd all be doing something **boring**, like watching TV, or writing an e-mail to Aunt Nellie! 🙂

By the way, if you are an experienced C++ programmer with a little extra time, who has an interest in advancing Celestia to new heights, please check out the Celestia Source Forge Project page and

consider offering some of your valuable time and experience. Celestia users seem to generate more good ideas for new features than we have programmers to implement them!

Thank you to the folks who provided Celestia scripting information in the past:

- **Unknown** for their *Scripting.html* document:
  http://63.224.48.65/celestia/docs/scripting.html
- **Selden Ball, Jr.** for his *Celestia Notes* web page:
  http://www.lns.cornell.edu/~seb/celestia/celestia_notes.html#4.0

A sincere **thank you** also goes to all the folks who post messages (past and present) on the Celestia Forums that relate to the topics covered in this Guide – which are many. Your names are too numerous to list here, but we are all indebted to you for sharing your Celestia and astronomical knowledge and experience with us.

If you haven't checked out the Celestia Forums yet, you should! They are a lot of fun and you will meet some interesting characters there too. 

Disclaimer

As much as I and the editors have tried to make sure the information and example code contained in this guide is correct, there are bound to be mistakes, omissions, or some things that are confusing or not explained well enough. If you happen to find one of these, **please** tell us about it by leaving a message on the Celestia Scripting forum so we can fix it. Thank you!

Introduction

If you are new to Celestia, the best place for you to start is the ***Celestia User's Guide***, written by Frank Gregorio. It can be downloaded from the **Celestia Documentation** web page at http://www.shatters.net/celestia/documentation.html. The *Celestia User's Guide* is available in the following languages and file formats:

- English (MS Word, HTML, and Adobe PDF
- French (MS Word)
- Japanese (MS Word and Plain Text)

The *Celestia User's Guide* takes you on a step-by-step guided tour of Celestia and its user interface, explaining and demonstrating what many of the mouse, keyboard, and menu options do. It also includes a comprehensive, printable list of the mouse, keyboard and joystick commands available in Celestia.

Once you are familiar with the keyboard and menu options available in Celestia, and how they function, you may want to automate some of your Celestia journeys so other folks can enjoy them too. The easiest way to do this is through Celestia's simple, built-in scripting capability called *.CEL Scripting*, which is what this guide is all about.

If you find that .CEL Scripting is not robust enough for your needs, or if you already know another programming language, you might want to explore the Lua programming language interface to

Celestia, which was introduced in version 1.3.1. This guide does not currently cover any of Lua's capabilities or how to write a Celestia script using Lua. However, you can find Lua information at http://www.lua.org/, and a few example Lua scripts for Celestia at the Celestia project site on SourceForge at http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/celestia/celestia/scripts/.

Symbols Used in this Guide

 **Warning**

 **Note**

Terms Used in this Guide

Argument

Sometimes referred to as a variable or parameter, an argument is merely a human-readable name that is assigned to the place in your computer's memory where the value you are assigning to the argument is stored. For example, the `wait` command has one argument, named `duration`. Other commands, like the `print` command, can have more than a single argument.

Data Types

A *Data Type* describes the type, or kind, of data that is used by the argument. The Data Types used by Celestia .CEL scripts include **<*String*>**, **<*Number*>**, and **<*Vector*>**, each of which is described in detail below:

- **<String>** This *Data Type* consists of any number of characters (text) enclosed in double quote marks ("). For example, **"Hello world."** is a <String> Data Type, and the text contained between the quote marks is sent to the script interpreter as a <String>.
  For example:
  ```
  select { object "Mars" }
  ```

  Some commands require the <String> value to be one item from a specific list of strings (text), and will reject any other values you might assign. For example, the `set` command uses an argument called `name`, which is a <String> value, but the value ***must*** be one of the following values:
  ```
  MinOrbitSize
  AmbientLightLevel
  FOV
  StarDistanceLimit
  ```

  If you assign text to the `name` argument that is not one of these values, such as "Hello", you will receive a script error and the script will not run.

- **<Number>** This *Data Type* consists only of digits, the numbers 0-9. It may, or may not, include any of the following, depending on the argument:
  * Decimal point
  * Sign prefix ("-" for negative numbers, such as -98.53)
  * Exponent suffix (such as 31.0e+15)

When you assign a <Number> value, you do not use double quote marks around the value, as is done with a <String> value. For example, to assign a value of 5.5 to a <Number> argument, you simply enter the digits, such as:

```
wait { duration 5.5 }
```

**** Do not use a negative number for the `time` argument. This will freeze your Celestia display, without warning, and you will then need to press the Esc key to reset the display.***

**** Do not use a negative number for the `duration` argument, as the command will be ignored.***

- **<Vector>** This *Data Type* consists of an array (or grouping) of three individual <Number> values, enclosed within square brackets "[ ]". Each <Number> value is separated by a space (spacebar). The <Vector> Data Type in Celestia, is mostly used to set 3-dimensional coordinates in space via three <Number> values that represent the *X*, *Y*, and *Z Axes*, from left to right in the array. The X, Y, and Z values represent different things depending on the argument.

  For example, the `setposition` command requires two <Vector> values, `base` and `offset`. The values shown below will position the camera outside of the Milky Way:

```
setposition {
  base [ -64132.43 47355.11 196091.57 ]
offset [ 0 0 -1.52e-005 ] }
```

  In some commands, a <Vector> is used to specify a single X, Y, or Z axis, by entering a value of 1 in the corresponding axis position and entering a 0 in the other axis positions. For example, the `up` argument of the `goto` command uses a <Vector> value for describing which axis of the selected Coordinate System you want to point "up". The example below sets the "up" axis to Y, since the <Vector> represents the three axes in the [X Y Z] format:

```
select { object "Sol/Earth" }
goto   { time 5
         distance 5.5
         upframe "follow"
         up [0 1 0] }
wait   ( duration 5 }
```

- **<Base64>** This *Data Type* consists of characters copied from a Celestia Cell://URL which are encoded in Base-64.

  For example, the `setposition` command can use `x`, `y`, and `z` values from a Cell://URL that are encoded in Base-64. The values shown below will position the camera outside of the Milky Way:

```
setposition {
  x "AMDCXoJK/3+IyGgR8f///w"
  y "BvP2LRdAAAD/n5UGCw"
  z "VUrGoeQJ+/8DyvenLQ" }
```

Default

A default value is the value automatically assigned to an argument by the script engine if you do *not* include that argument in the command code. However, not all arguments have a default value (ie. `select`) so if you do not specify an argument in the command you are using, make sure it has a default value, or you will get a script error. For example, if you do not specify the `duration` argument in the `wait` command, a value of 1.0 seconds will automatically be assigned, as in:

```
wait {}
```

Null

A null value means that no arguments are used with that command. For example, the `lock` command does not accept any arguments, so it is coded as follows ...

```
lock {}
```

... with *nothing* between the curly braces.

Celestia .CEL Scripts

What Is a Celestia .CEL Script?

A Celestia .CEL script is simply a plain-text file that contains a list of instructions for Celestia to carry out *automatically*. For example, when a default installation of Celestia is run, the on-screen display (which will be called the **camera** in this guide), moves to Jupiter's moon Io. This action is taken automatically when Celestia is run, via the included startup script, named `start.cel`, which is located in the main Celestia program folder. You also may have learned that pressing the [**D**] key on your keyboard runs a *Demo* of Celestia. This too, is a Celestia .CEL script. The filename of this script is `demo.cel` and it is also located in the main Celestia program folder.

You can open a Celestia script with any plain-text editor, such as Notepad, Wordpad or Word (for Windows users). Just make sure that when you save your script file to disk, that its filename uses the file extension "**`.cel`**", as this is how your computer's operating system knows the file is one of Celestia's script files, and also that you save it as a plain text file (Rich Text Format files will not work). If you are interested in seeing what a .CEL script looks like, run your favorite plain-text editor now, and open the `start.cel` script in Celestia's main folder.

Before modifying either of these scripts (`start.cel` or `demo.cel`), please copy the **original** script files to a safe place **FIRST**, just in case you need them at a later date.

The `start.cel` script that comes with Celestia should look similar to the following:

```
{
   select { object "Sol/Jupiter/Io" }
   follow {}
   goto { time 5 }
   # gotolonglat { time 0 distance 1e11 longitude 0 latitude 0 }
   # gotolonglat { time 0 distance 2.5 longitude -122 latitude 47 } #
      Seattle!
   # wait { duration 0.1 }
   # orbit { axis [ 0 1 0 ] rate 10 duration 7 }
   # goto { time 5 distance 10 }
   # wait { duration 5.0 }
}
```

All of the commands and arguments shown in this script will be discussed later in this guide. If you are familiar with Celestia's keyboard commands, you should understand a lot of it just by looking at it. The `select` command is the same as pressing **[Enter]** and typing in an object name. The `follow` command is the same as pressing the **[F]** key on your keyboard. And, the `goto` command is the same as pressing the **[G]** key.

The lines that start with the "#" character are comment lines, that are not executed. When the "#" character is removed, that line of code will be executed the next time you run the script. Chris Laurel, the creator of Celestia, included these commented lines for user experimentation.

Writing Celestia Script Commands

The very first character in a Celestia .CEL script **must** be an opening curly brace "`{`". The very last character in a script **must** be a closing curly brace "`}`". Therefore, an empty, but valid, Celestia .CEL script file looks like the following:

```
{ }
```

That's right, the opening and closing curly braces would be the only text in the script file. However, this script does absolutely nothing at all, which makes it pretty useless. Every script you create for Celestia will have at least one command line in it, which is a line of text instructing Celestia to ***set*** or ***do*** something.

A typical single Celestia script command line consists of the following pieces:

- A *command name*, such as `wait`
- A space (spacebar)
- An opening curly brace `{`
- Zero or more *argument names*, such as `duration`, along with the value you want the argument set to ... For example: `duration 5.5` or `object "Mars"`
- A closing curly brace `}`

Here are some command line examples:

- Assign a <*Number*> value of 5.5, to the argument `duration`, in the command `wait`, which tells Celestia to pause for the number of seconds you enter:
  ```
  wait { duration 5.5 }
  ```
  Notice there are no double quote marks (") around the <*Number*> value (5.5) because double quote marks are used **only** to specify a <*String*> value.

- Assign a <*String*> value of `"Mars"`, to the argument `object`, in the `select` command:
  ```
  select { object "Mars" }
  ```
  Here, double quote marks tell Celestia that a <*String*>, or text value (Mars), is being sent to it. When sending a <String> to Celestia, the value must be enclosed in double quote marks.

Some Celestia .CEL scripting commands, such as `print`, send more than one argument. In these cases, you enclose **all arguments and their associated values** between the command's opening and closing curly braces, such as:

```
print { text "Hello universe."  row -4  column 1  duration 5 }
```

or

```
print { text "Hello universe."
        row -4
        column 1
        duration 5 }
```

All arguments for each command must be enclosed within the command's curly braces, such as the example above shows. You may include as much *white space* (spaces, blank lines, etc.) as you

deem necessary, and you may even specify each argument and its associated value on a different line – all for clarity.

Comment Lines

The "#" character can be used as the first character in a script line to make that line a ***comment line***, which the script engine will ignore (not execute). For example:

```
# ---------------------------------------
#   This is my Celestia script
# ---------------------------------------
  select { object "Sol/Earth" }
  center { time 3 }
  wait   { duration 3 }
# ---------------------------------------
```

Every line beginning with the "#" character will be ignored by Celestia's script engine (the internal part of Celestia that reads and interprets a script file into Celestia commands).

Your First Celestia .CEL Script

The first Celestia script you create should probably be a *template* script (see below for an example), that contains the specific settings you want Celestia to have every time you run one of your scripts. Actually, you don't have to create this script at all, as it has already been done for you. All you need to do is modify it to suit your needs, once you learn more about Celestia script commands. Here is a very short example of a template script:

```
{
 #-----------------------------------------------------------------
 # Written by: <your name>
 #         on: <date>
 #
 # The purpose of this script is ...
 #-----------------------------------------------------------------
 # Set Celestia values (modify to meet your needs)...
 #-----------------------------------------------------------------
 # Render the following objects...
     renderflags { set "cloudmaps|planets|stars" }

 # Unmark all objects...
     unmarkall {}

 # Set Field of View...
     set { name "FOV" value 30.0 }

 # Set Time Rate (1x, 100x, 1000x, etc.)...
     timerate { rate 1.0 }
 #-----------------------------------------------------------------
 # Script Body - Your code goes here ...
 #-----------------------------------------------------------------
   <your command lines go here>

 #-----------------------------------------------------------------
```

```
  # End of Script - Let the user know the script is done running ...
  #-----------------------------------------------------------------
    print  { text "The script is finished."
             row  -3
             column 25
             duration 5 }

    wait   { duration 5 }
}
```

A template script saves you time because you don't have to remember or code all of the initial start-up values you want to set, in every script you write. And, the settings in the template script can be easily modified whenever you use it to begin a new script.

The example template script above also demonstrates the value of using comment lines to describe the script, who wrote it, when it was written, what the script is supposed to do, and what each section of the script code is doing. Comments are particularly useful when you want to share your scripts with other Celestia users, who did not write it, and may not fully understand what your intent was when you wrote it.

To obtain a *free*, much more in-depth template script, visit the Celestia Scripting Forum. The script is located in the following topic: http://ennui.shatters.net/forum/viewtopic.php?t=2961 (**Template Script for .cel scripts**), or visit Don G's Celestia Page and download the latest version.

When you save a Celestia .CEL script file to disk, the file extension **must be** "`.cel`", which allows your operating system to recognize the file as belonging to Celestia, and allows Celestia to know this file is one of its .CEL scripts. Also, you **must** save the file as **plain text**. Rich Text Format files do not function in Celestia because they also contain embedded formatting information.

When you double-click on a script filename in your file browser, the operating system will automatically run Celestia (if it's not already running), which will in turn run the script file. In Microsoft Windows, this is called *File Association*.

The example template script above can be copied, pasted into your text editor, and then saved to a file on your computer named `template.cel`. Or, just download the complete, *free* template from Don G's Celestia Page. When you learn more about Celestia .CEL scripting commands, you will want to edit your template script to setup Celestia the way *you* like to see things.

You now have a ready-made, beginning script that you can use whenever you want to create a new script. Simply open the `template.cel` file and immediately save the file as another name (File / Save As) – and enter the name of your **new** script.

Coordinate Systems

Celestia's Coordinate Systems

The following definition is from Astronomy Answers -- Astronomical Dictionary ...

> *A Coordinate System is a tool that allows fixing of positions by measuring distances in different directions. Those distances are the coordinates. There are two often-used classes of coordinate systems: those that use rectangular (or cartesian) coordinates, and those that use polar coordinates. Polar coordinates use angles (for the direction) and one distance, and usually have a base plane that the angles are tied to. Rectangular coordinates use only distances, measured in mutually orthogonal directions from a common origin. Coordinate systems are often named for the thing in or on which they measure positions, or for the base plane or origin with which they are associated.*
>
> *The names of polar coordinate systems that fix positions on a celestial body are often made up of the Latin or Greek name of the body, followed by graphical, for example geographical for the Earth. The names of the corresponding rectangular coordinate systems (with the center of the body as origin) usually have the same first part, but followed by centric, for example geocentric for the Earth, planetocentric for planets in general, or heliocentric for the Sun.*

When running Celestia interactively, you can see that it has several different *modes*, including *Follow*, *Sync Orbit*, *Lock*, *Chase*, and *Free* (none, Esc key was just pressed). Together with one or two selected objects, these modes define a Celestia ***Coordinate System***. In Celestia's interactive mode, you control your exact position (longitude, latitude, orientation, orbit type, etc.) with keystrokes. In .CEL scripting, there are no interactive keystrokes, so you need to tell Celestia exactly where you want to be located in the 3-dimensional universe, and how you want the camera to be oriented. This is where Celestia's *Coordinate Systems* come into play. This section describes how the *X*, *Y*, and *Z*, axes of each 3-dimensional Coordinate System are aligned.

Each of Celestia's interactive modes is represented by a Coordinate System in .CEL scripting, which is set via the `setframe` command. The following list shows each of Celestia's interactive Modes along with its corresponding Coordinate System:

Celestia Mode	Related Coordinate System (`setframe`)
Chase	`chase`
Follow	`ecliptical`
Lock	`lock`
Free (none set)	`universal`

Sync Orbit	`geographic`
n/a	`equatorial`
n/a	`observer/local (both used in code and Bookmarks)`

Below, is a brief description of the Coordinate Systems Celestia uses, which you will find helpful when using the `goto`, `gotoloc`, `gotolonglat`, and `setframe` commands. You may want to skip this section until you actually have a need to select a Coordinate System in your script, then come back here to learn more about how each system is aligned.

All of Celestia's Coordinate Systems are orthonormal, meaning that the *X*, *Y*, and *Z* axes are all at right angles to each other and all have unit length (there is no stretching).

Chase (Chase)

The Chase Coordinate System (which is also a .CEL script command) tells Celestia to **chase** the currently selected object through space, which keeps your orientation constant with respect to the direction in which an object is moving. This Coordinate System is similar to **lock** (explained below), except that the Z axis points in the direction of motion relative to the *target* object's parent body (the Sun for planets, a planet for moons). The keyboard command for `chase` is the quote mark **["]**.

In Chase mode, your position remains fixed relative to the target's direction of motion. If you're directly in front of or behind the target (in terms of its motion), you'll remain there regardless of how it moves. In the case of the Moon, Chase is similar to Sync Orbit mode (Geographic Coordinate System, although the Moon wobbles around quite a bit) because the Moon's direction of motion is nearly fixed with respect to its surface, in terms of longitude, at least.

Try Chase mode with a comet other than Halley. Because comets generally have highly eccentric orbits, the direction of motion will not always be almost perpendicular to the direction to the Sun. In Lock mode the Sun will stay fixed, but it will drift across the field of view in Chase mode.

Note: When you click the Chase Moon link below, if Celestia is not currently running, the program will be started. Celestia will then be directed to display the scene as described in a saved Cell://URL. All of the visual, on-screen examples in this guide are presented in this manner, as saved Cell://URLs.

Here is a "Chase Moon" example with the Earth in view and time sped up.

Here is an example of chasing the International Space Station as it passes over the southwest U.S.A. (a closer view of the ISS).

Following the ISS provides quite a different perspective, while locking ISS and Earth is similar to Chasing ISS.

Ecliptical (Follow)

The Ecliptical Coordinate System tells Celestia to **follow** the selected object. As the object moves through space, the camera moves with it. The object can rotate on its axis below you, which means

the longitude will constantly change – but you will remain the same distance and latitude above the object. The keyboard command for `follow` is the **[F]** key.

The X axis points away from the Sun in the direction of the Julian 2000.0 vernal equinox. The Y axis is normal to the ecliptic with positive Y north. The Z axis completes the right-handed coordinate system.

Example: Moon orbiting Earth (starting from the lower right corner).

Another example of how Ecliptical/Follow works: Follow Earth with Orbits displayed.

Using the Ecliptical Coordinate System on our sun (Sol), allows you to watch the planets orbit the sun.

Equatorial (n/a)

??? This Coordinate System is selectable in the `setframe` command but is not available as a keystroke command.

Lock (Lock)

As when running Celestia interactively, this coordinate system involves *two* objects – the first object selected is called the **reference** object, and the second is called the **target** object. When two objects are **lock**ed together as a pair, as you rotate the scene, the target object stays *locked* to the reference object, and the distance displayed is the distance *between the two objects*. The keyboard command for `lock` is the colon **[:]**.

The Z axis points from the **reference** object (first object selected) to the **target** object; the Y axis lies in the plane defined by the Z axis and the rotation axis of the **reference** object, at a right angle to the Z axis. The X axis completes the right-handed coordinate system.

In Lock mode, your position remains fixed relative to a line between the reference object and the target object. As the target object moves around the reference object (relatively speaking) so do you. Thus if the target object is Sol, the illuminated fraction of the reference object (its "phase") remains the same because you're moving around the object in sync with Sol.

From the developers: Unfortunately, this definition does not work when the rotation axis is perfectly aligned with the **reference** to **target** direction. It also produces strange results with precessing objects. We are looking into changing it so that this Coordinate System is defined by the axis of precession instead of the axis of rotation.

Example: Lock Earth-Moon showing Moon Phases. Earth is the **reference** object and Moon is the **target** object.

Another example using Earth as the reference object, but Sol as the target.

Universal (Free - Esc key pressed)

Setting the Coordinate System to **universal** de-selects the current Coordinate System and places Celestia into *free* (or *absolute*) mode, which is what happens when you press the **[Esc]** key on the keyboard. This mode is not time dependent – a given position (using one of the `goto` commands) will always place you at the same point in space regardless of the current simulation time.

Observer/Local

??? This Coordinate System is selectable in the `setframe` command but is not available as a keystroke command.

Geographic (Sync Orbit)

The Geographic Coordinate System allows you to remain in a **stationary**, or **geosynchronous** orbit above the selected object (not just Earth). As the object rotates below, the camera moves with it, as if it were attached to the object. The keyboard command for Sync Orbit is **[Y]**.

In the Geographic Coordinate System, the axes rotate *with* the selected object. The Y axis is the axis of rotation – counter-clockwise, so it points north for prograde rotators like Earth, and south for retrograde rotators like Venus. The X axis points from the center of the object to the intersection of its zero longitude meridian and equator. The Z axis (at a right angle to the XY plane) completes the right-handed coordinate system. An object with constant geographic coordinates will thus remain fixed with respect to a point on the surface of the object.

The **origin** of all Celestia Coordinate Systems, **except `universal`**, is the center of the **reference** object. For example, if you are **follow**ing Mercury and execute a `gotoloc` command with a `position` of `[0 0 0]`, you will travel to the center of Mercury. In **`universal`** mode, `[0 0 0]` is a location near the Sun (Sol).

The `position` argument in `gotoloc` is a point in the current Coordinate System specified in units of kilometers.

Web pages that discuss Coordinate Systems in more detail:

Astronomy Answers -- Astronomical Dictionary (text only):
http://www.astro.uu.nl/~strous/AA/en/woordenboek.html#coordinate_system

Coordinate Systems Overview (text and graphics):
http://www.colorado.edu/geography/gcraft/notes/coordsys/coordsys_f.html

Celestial Coordinate System (text and graphics):
http://csep10.phys.utk.edu/astr161/lect/time/coordinates.html

Coordinate Systems in Astronomy (text and graphics):
http://www.astro.virginia.edu/~teacha/130_manual/node9.html

Earth Coordinate System (text and graphics):
http://zebu.uoregon.edu/~js/ast121/lectures/lec02.html

Beyond Our Skies: Discovering the Cosmos -- Coordinate Systems (text and graphics):
http://library.thinkquest.org/29033/begin/coordinate.htm

Coordinate system transformation (equations and graphics):
http://astronomy.swin.edu.au/~pbourke/projection/coords/

.CEL Script Commands

.CEL Script Command Descriptions

This section lists all Celestia .CEL script commands, as of version 1.3.1, along with a description of what each command does, what its arguments are, default argument values, valid values for the arguments (if any exist), and examples of how to code the command.

Remember, experimentation can be the most enjoyable part of learning, so please take your time and experiment with each of the commands. If you discover a particularly interesting effect when implementing a series of commands, please, by all means, share it with all the other Celestia users in the Celestia Scripting Forum!

The first part of this section provides an *alphabetical* reference list of script commands, and a *categorized* list, by the type of command.

*** All** Celestia script commands **must be entered in lower-case** only, as shown, or they will not be recognized by Celestia. You should also enter all argument names in lower-case only, even though some are not case sensitive.

***** When you code a command that includes a `time` or `duration` argument, such as `goto`, `print`, `center`, etc., you **must** include a `wait` command after it, with a duration equal to or greater than the length of time used in the command. If you forget to do this, the command may be ignored or may not function as you intend.

Command Reference -- Alphabetical

`cancel`	Cancels `goto` and `track` commands, and resets the Coordinate System to `universal`.
`center`	Center the currently selected object in the display.
`changedistance`	Change your distance from the selected object.
`chase`	Set the Coordinate System to `chase`.
`cls`	Clear the display screen of all text printed using the `print` command.
`follow`	Follow the currently selected object – Sets the Coordinate System to `ecliptical`.
`goto`	Go to the currently selected object using the current Coordinate System.
`gotoloc`	Go to the specified position and orientation of the currently selected object using the current Coordinate System.

`gotolonglat`	Go to the specified `longitude` and `latitude` of the selected object.
`labels`	Set or clear `labels` to be displayed.
`lock`	Lock two objects together in the camera's view. Sets the Coordinate System to `lock`.
`lookback`	Change the current camera view by 180 degrees.
`mark`	Mark the defined object with the specified symbol.
`move`	Move the camera at the specified velocity.
`orbit`	Orbit the currently selected object using the current Coordinate System.
`preloadtex`	Pre-load a texture file from disk to memory.
`print`	Display a text message on the display screen.
`renderflags`	Set or clear items to be rendered/displayed on-screen.
`rotate`	Rotate the camera view.
`select`	Select an object (planet, moon, galaxy, etc.).
`set`	Set the Min Orbit Size, Ambient Light Level, Field Of View, Star Distance Limit, or Star Style.
`setfaintestautomag45deg`	Set the Magnitude of stars to be displayed when Auto-Magnitude is ON.
`setframe`	Set the Coordinate System to be used.
`setorientation`	Set the camera's orientation.
`setposition`	Set the camera's position in space.
`setsurface`	Set the name of an alternative surface for the selected object.
`setvisibilitylimit`	Set the Magnitude of stars to be displayed when Auto-Magnitude is OFF.
`seturl`	Move the camera to the location of a saved "location URL" (or Cell://URL).
`synchronus`	Orbit the currently selected object in Synch Orbit mode. Sets the Coordinate System to `geographic`.
`time`	Set the date and time in JulianDay format.
`timerate`	Set the time multiplication factor, ie. 100x.
`track`	Keep the currently selected object centered in the display.
`unmark`	Unmark the specified object.
`unmarkall`	UnMark all objects and disable the display of Marks.
`wait`	Pause script execution for the specified number of seconds.

Below, is a ***categorized*** listing of script commands that should help you find the ***type*** of command you need, with some commands being listed in multiple categories. Each category is separated by a page-break in case you want to print them out on paper:

Command Reference -- Category: *Object Control and Display*

`center`	Center the currently selected object in the display.
`chase`	Set the Coordinate System to `chase`.
`follow`	Follow the currently selected object – Sets the Coordinate System to `ecliptical`.
`lock`	Lock two objects together in the camera's view. Sets the Coordinate System to `lock`.
`lookback`	Change the current camera view by 180 degrees.
`orbit`	Orbit the currently selected object using the current Coordinate System.
`rotate`	Rotate the camera view.
`select`	Select an object (planet, moon, galaxy, etc.).
`setframe`	Set the Coordinate System to be used.
`setorientation`	Set the camera's orientation.
`setposition`	Set the camera's position in space.
`setsurface`	Set the name of an alternative surface for the selected object.
`seturl`	Move the camera to the location of a saved "location URL" (or Cell://URL).
`synchronus`	Orbit the currently selected object in Synch Orbit mode. Sets the currently active Coordinate System to `geographic`.
`track`	Keep the currently selected object centered in the display.

Command Reference -- Category: *Overall Display - Rendering - Flags*

`cls`	Clear the display screen of all text printed using the `print` command.
`labels`	Set or clear `labels` to be displayed.
`lookback`	Change the current camera view by 180 degrees.
`mark`	Mark the defined object with the specified symbol.
`preloaxtex`	Pre-load a texture file from disk to memory.
`print`	Display a text message on the display screen.
`renderflags`	Set or clear items to be rendered/displayed on-screen.
`set`	Set the Min Orbit Size, Ambient Light Level, Field Of View, Star Distance Limit, or Star Style.
`setfaintestautomag45deg`	Set the Magnitude of stars to be displayed when Auto-Magnitude is ON.
`setframe`	Set the Coordinate System to be used.
`setsurface`	Set the name of an alternative surface for the selected object.
`setvisibilitylimit`	Set the Magnitude of stars to be displayed when Auto-Magnitude is OFF.
`unmark`	Unmark the specified object.
`unmarkall`	UnMark all objects and disable the display of Marks.

Command Reference -- Category: *Movement - Navigation*

`cancel`	Cancels `goto` and `track` commands, and resets the Coordinate System to `universal`.
`center`	Center the currently selected object in the display.
`changedistance`	Change your distance from the selected object.
`chase`	Set the Coordinate System to `chase`.
`follow`	Follow the currently selected object – Sets the currently active Coordinate System to `ecliptical`.
`goto`	Go to the currently selected object using the current Coordinate System.
`gotoloc`	Go to the specified position and orientation of the currently selected object using the current Coordinate System.
`gotolonglat`	Go to the specified `longitude` and `latitude` of the selected object.
`lookback`	Change the current camera view by 180 degrees.
`move`	Move the camera at the specified velocity.
`orbit`	Orbit the currently selected object using the current Coordinate System.
`rotate`	Rotate the camera view.
`setframe`	Set the Coordinate System to be used.
`setorientation`	Set the camera's orientation.
`setposition`	Set the camera's position in space.
`seturl`	Move the camera to the location of a saved "location URL" (or Cell://URL).
`synchronus`	Orbit the currently selected object in Synch Orbit mode. Sets the currently active Coordinate System to `geographic`.
`track`	Keep the currently selected object centered in the display.

Command Reference -- Category: *Date and Time*

`time`	Set the date and time in Julian Date or UTC format.
`timerate`	Set the time direction (forward or backward) and set the time multiplication factor (ie. 100x).
`wait`	Pause script execution for the specified number of seconds.

Alphabetical Listing of Commands

---

cancel

`(null)`

Cancels `goto` and `track` commands, and resets the Coordinate System to `universal`, which means it also cancels `follow`, `synchronous` and other Coordinate System related commands. This command is much like pressing the ESC key when running Celestia in its interactive mode.

When moving from one object to another, it's a good idea to use the `cancel` command between them, in order to release any functional *hold* that exists on the currently selected object. If you want to maintain your current Coordinate System setting (if other than `universal`), you will need to reset it via the `setframe` command after executing the `cancel` command.

For example, the first part of this script selects the Earth, goes to it, and then performs other commands, maybe to change it's view, speed up time, etc. The second section executes a `cancel` command, selects Mars, and then performs other commands on Mars:

```
   select { object "Sol/Earth" }
   goto   { time 3 }
   wait   ( duration 3 }
 # ...
 # <other commands doing other things>
 # ...
   cancel { }
   select { object "Sol/Mars" }
   goto   { time 3 }
   wait   { duration 3 }
 # ...
 # <other commands doing other things>
 # ...
```

If the currently selected object has `follow` enabled, then it will be *automatically* overridden by `select`ing and `follow`ing a new object.

If the currently selected object has `track` enabled, as of Celestia version 1.3.1 you can use the following code to cancel only a `track` command:

```
 select { object "" }
 track  { }
```

---

center

`time <Number> (default 1.0)`

Centers the currently selected object in the display. You must first use the `select` command to select an object to be centered.

`time`	Number of seconds to take centering the object.

Do not use a negative number for the `time` argument. This will freeze your Celestia display, without warning, and you will then need to press the Esc key to reset the display.

The following example selects the Earth and takes 5-1/2 seconds to center it on the screen. If you do not see the Earth, press the "g" key on your keyboard and you should see the Earth zoom into view from the center of the display:

```
select { object "Sol/Earth" }
center { time 5.5 }
```

---

changedistance

```
duration <Number> (default 1.0)
rate     <Number> (default 0.0)
```

Changes the camera's distance from the currently selected object. You must first use the `select` command to select an object.

`duration`	Number of seconds to take when changing the distance.
`rate`	Speed at which to change the distance. Small values work best, from decimal values, to 5 or 6. A negative value moves the camera closer to the object, while a positive value moves the camera further away.

Do not use a negative number for the `duration` argument, as the command will be ignored.

The following example selects the Earth, travels to it, and then spends 2-1/2 seconds changing your display distance to be further away from the Earth:

```
select { object "Sol/Earth" }
goto   { time 3 }
wait   { duration 3 }
changedistance { duration 2.5 rate 0.5 }
wait   { duration 2.5 }
```

---

chase

```
(null)
```

Tells Celestia to set the active Coordinate System to `chase`, which *chases* the currently selected object through space (please refer to the Coordinate Systems section earlier in this guide). This keeps your orientation constant with respect to the direction in which an object is moving. This Coordinate System is similar to **lock** (please see below), except the Z axis points in the direction of motion relative to the *target* object's parent body (the Sun for planets, a planet for moons). The keyboard command for `chase` is the double quote mark **["]**.

In Chase mode, your position remains fixed relative to the target's direction of motion. If you're directly in front of or behind the target (in terms of its motion), you'll remain there regardless of how it moves. In the case of the Moon, Chase is similar to Sync Orbit mode (Geographic Coordinate System, although the Moon wobbles around quite a bit) because the Moon's direction of motion is nearly fixed with respect to its surface, in terms of longitude, at least.

Try Chase mode with a comet, other than Halley. Because comets generally have highly eccentric orbits, the direction of motion will not always be almost perpendicular to the direction to the Sun. In Lock mode the Sun will stay fixed, but it will drift across the field of view in Chase mode.

You must first `select` an object before using `chase`.

Only *one* Coordinate System can be active at any given time:

- `chase` (Chase)
- `equatorial`
- `ecliptical` (Follow)
- `geographic` (Sync Orbit)
- `lock` (Lock)
- `observer`
- `universal` (Free – ie. Esc key pressed)

The following example selects the Moon, sets the Coordinate System to `chase`, and then goes to the Moon:

```
select { object "Sol/Earth/Moon" }
chase  { }
goto   { time 2 }
wait   { duration 2 }
```

The following Cel:/URLs (links that display pre-defined scenes on ***your*** Celestia program) demonstrate a couple views of Earth, from the Moon, with the Moon set to `chase`:

Chase Moon, with Earth in view

Chase Moon with Earth and Sun

---

cls

`(null)`

Clears the display screen of any left-over text that was printed via the `print` command. Personally, I have not found a need to use this command in any of the scripts I have written.

Example:

```
cls { }
```

follow

`(null)`

This command tells Celestia to set the Coordinate System to `ecliptical`, and to *follow* the currently selected object (please refer to the Coordinate Systems section earlier in this guide). As the object moves through space, the camera moves with it. The object can rotate on its axis, which means the longitude will constantly change, but the camera will remain the same distance and latitude above the object.

The X axis points away from the Sun in the direction of the Julian 2000.0 vernal equinox. The Y axis is normal to the ecliptic with positive Y north. The Z axis completes the right-handed coordinate system.

You must first use the `select` command to select an object to be followed.

Only *one* Coordinate System can be active at any given time:

- `chase` (Chase)
- `equatorial`
- `ecliptical` (Follow)
- `geographic` (Sync Orbit)
- `lock` (Lock)
- `observer`
- `universal` (Free – ie. Esc key pressed)

When you `follow` Earth, this coordinate system makes Earth the center of the solar system, where the Sun revolves around the Earth: Cell://URL Follow Example.

This example selects Mars, sets the Coordinate System to ecliptical (`follow`), and then travels to Mars:

```
select { object "Sol/Mars" }
follow { }
goto   { time 2 }
wait   { duration 2 }
```

goto

```
time     <Number> (default 1.0)
distance <Number> (default 5.0)
upframe  <String> (default "observer")
up       <Vector> (default [0 1 0] "Y")
```

Moves the camera to the currently selected object, taking `time` seconds, stopping `distance` from the object, using the `upframe` Coordinate System, and defining the axis that points `up`. You must first use the `select` command to select an object.

`time`	The number of seconds to take going to the object.

`distance`	Describes how far away from the object you want to be positioned, in units of the object's radius, plus 1. For example, if the object's radius is 10000 km, and you specify 6 for `distance`, you will be positioned 50000 km from the object's center. An easy way to compute this value is: `distance` = distance you want to be from the object, divided by the object's radius, plus one. For example, if you want to be 1,000,000 km from Earth, which has a radius of 6,378.1 km, then the calculation is: `(1000000 / 6,378.1) + 1 = 157.7865` Special `distance` values: 0 (zero) = Center of the object. In version 1.3.1, using this value causes the program to incorrectly recognize further positioning values, so do not use zero. 1 = Surface of the object. Traveling to the exact surface level of an object may cause some graphics cards to display random polygons on the display screen, so it is best to use a value slightly **above** the surface.
`upframe`	Sets the specified Coordinate System, which **must** be one of the following values (for more information, refer to the Coordinate System descriptions earlier in this guide): • chase • ecliptical • equatorial • geographic • lock • observer • universal
`up`	Defines which axis points up, X [1 0 0], Y [0 1 0] or Z [0 0 1].

Do not use a negative number for the `time` argument. This will freeze your Celestia display, without warning, and you will then need to press the Esc key to reset the display.

***** Before using the `goto` command, you must first use the `select` command to select an object.

***** You can use other non-movement commands, such as **`print`**, while the camera is moving towards its destination.

***** Before using other commands that have a time argument, you

must execute a wait command with a duration equal to or greater than the value of the goto command's `time` argument. Otherwise, the camera **will not** execute the goto command, but will instead execute the next time-based command.

The following example selects Mars and takes five seconds to travel there:

```
select { object "Mars" }
goto   { time 5 }
wait   { duration 5 }
```

Invalid example -- The following is an example of how ***not*** to perform a time-based command in Celestia. One might assume that this series of commands would `select` Mars, taking five seconds to get there; then `select` Earth and take three seconds to get there. However, since the required `wait` command was not performed after the first `goto` command, Celestia ignores the first `goto` command and executes only the second one. You would only see Earth, and not Mars.

```
select { object "Mars" }
goto   { time 5 }
select { object "Earth" }
goto   { time 3 }
wait   { duration 8 }
```

The ***correct*** way to script the above example is as follows:

```
select { object "Mars" }
goto   { time 5 }
wait   { duration 5 }
select { object "Earth" }
goto   { time 3 }
wait   { duration 3 }
```

The next example shows how a non-movement command can be used after a `goto` command, by displaying a message on the screen *while* Celestia is going to Mars:

```
select { object "Mars" }
goto   { time 5
         distance 8.5
         upframe "follow"
         up [0 1 0] }
print  { text "We're on our way to Mars."
         row -3 column 1
         duration 5 }
wait   { duration 5 }
```

---

gotoloc

```
time     <Number> (default 1.0)
position <Vector> (default [0 1 0])
xrot     <Number> (default 0.0)
yrot     <Number> (default 0.0)
zrot     <Number> (default 0.0)
```

-or-

```
time      <Number> (How many seconds to take, no default)
x         <Base64> (no default)
y         <Base64> (no default)
z         <Base64> (no default)
ow        <Number> (no default)
ox        <Number> (no default)
oy        <Number> (no default)
oz        <Number> (no default)
```

The `gotoloc` command moves the camera to the currently selected object, taking `time` seconds, traveling to the specified `position` (or the Base64 values `x`, `y`, and `z` from a Cell://URL), using the orientation specified via `xrot`, `yrot`, and `zrot` (or `ow`, `ox`, `oy`, and `oz`). You must first use the `select` command to select an object. (Note: `gotoloc` does **not** reset the coordinate system. It retains the previously set coordinate system.)

If you are attempting to duplicate a position based on a Cell://URL, you will also need to set the proper Coordinate System, position, and other parameters. (Also see `seturl`.)

The `x`, `y`, and `z` values represent `position` as stored by a Cell://URL. The `ow`, `ox`, `oy` and `oz` values represent orientation (`xrot`, `yrot`, and `zrot`) as stored by a Cell://URL. These values are obtained from the following Cell://URL values:
`?x= &y= &z=` and `&ow= &ox= &oy= &oz=`

`time`	The number of seconds to take going to the object.
`position`	Defines a point in the current Coordinate System specified in units of kilometers. For all but the `universal` Coordinate System, a `position` of [0 0 0] is the center of the object. The first value represents the camera location, in kilometers, along the X axis. The second value represents the camera location, in kilometers, along the Y axis. The third value represents the camera location, in kilometers, along the Z axis. The kilometer value you specify must include the radius of the object. For example, if you want to be positioned 100,000 km above the surface of the Earth, which has a radius of 6378.1 km, you would add the radius of the Earth to your desired altitude of 100,000 km, which results in a `position` value of `106378.1` (see example below).

`xrot`
`yrot`
`zrot`
	`xrot`, `yrot`, and `zrot` are the "Eular Angle" or "Angle-Axis" representation of the camera's orientation. Think of them as Pitch, Yaw, and Roll in aviation. These are used to control which direction the camera is looking. The related keyboard commands are as follows:
	X/Pitch (Up/Down Arrow)
	Y/Yaw (with NumLock **on**: Keypad #4 and Keypad #6)
	Z/Roll (Left/Right Arrow)

Do not use a negative number for the `time` argument. This will freeze your Celestia display, without warning, and you will then need to press the Esc key to reset the display.

The following example displays the Earth 100,000 kilometers away:

```
select    { object "Sol/Earth" }
center    { }
follow    { }

gotoloc   { time 2
            position [0 106378.1 0]
            xrot 90
            yrot 0
            zrot 0 }
wait      { duration 2 }
```

The next example is a bit more complex and demonstrates both `gotoloc` methods. It travels to the Earth, setting a position so you can see the night lights over the USA, view the Sun in the upper left hand corner of the display, and the Moon in the upper right hand corner:

```
{
  cancel { }

  renderflags { clear "boundaries|comettails|constellations" }
  renderflags { clear "eclipseshadows|orbitspointstars|ringshadows" }
  renderflags { clear "automag|grid" }
  renderflags { set   "cloudmaps|galaxies|nightmaps|planets|stars" }
  renderflags { set   "atmospheres|markers" }

  labels    { clear "planets|moons|spacecraft|asteroids" }
  labels    { clear "constellations|stars|galaxies" }

# Pause time and set the date and time...
  timerate  { rate 0 }
  time      { jd 2452874.692638889 }

# Place a blue square marker around the Moon...
  unmarkall { }
  mark      { object "Sol/Earth/Moon"
              size   20.0
              color  [0 0 1]
              symbol "square" }
```

```
# Set the Field Of View...
  set       { name "FOV" value 47.0 }

  select    { object "Sol/Earth" }
  center    { }
  follow    { }

# Goto location: +7000km-X, +9000km-Y, +11000km-Z
# Pitch Up +13X, Yaw Left -32Y, Roll Left -95Z
  gotoloc   { time 2
              position [7000 9000 11000]
              xrot 13
              yrot -32
              zrot -95 }
  wait      { duration 2 }

  print     { text "Blue square is the Moon ---"
              row -24 column 32 duration 3 }
  wait      { duration 3 }

#****************************************
  print { text "First result done."
          duration 2  row -3  column  2 }
  wait { duration 2 }
#****************************************

  cancel { }
  gotoloc {
    time 2
    x "wJjebddBM3XLDA"
    y "Z2/b0Q34Pw"
    z "GShiyVOKuxUI"
    ow 0.954531
    ox 0.112769
    oy -0.272262
    oz -0.045019 }

#****************************************
  print { text "Second result done."
          duration 2  row -3  column  2 }
  wait { duration 2 }
#****************************************

}
```

---

gotolonglat

```
time      <Number> (default 1.0)
distance  <Number> (default 5.0)
up        <Vector> (default Y [0 1 0])
longitude <Number> (no default)
latitude  <Number> (no default)
```

Go to the currently selected object, taking `time` seconds, stopping `distance` from the object, using `up` orientation, positioning yourself above the specified `longitude` and `latitude` surface coordinates. (**Note:** gotoloc does **not** reset the coordinate system. It retains the previously set coordinate system.)

`time`	The number of seconds to take going to the object.
`distance`	Describes how far away from the object you want to be positioned, in units of the object's radius, plus 1. For example, if the object's radius is 10000 km, and you specify 6 for `distance`, you will be positioned 50000 km from the object's center. An easy way to compute this value is: `distance` = distance you want to be from the object, divided by the object's radius, plus one. For example, if you want to be 1,000,000 km from Earth, which has a radius of 6,378.1 km, then the calculation is: `(1000000 / 6,378.1) + 1 = 157.7865` Special `distance` values: 0 (zero) = Center of the object. In version 1.3.1, using this value causes the program to incorrectly recognize further positioning values, so do not use zero. 1 = Surface of the object. Traveling to the exact surface level of an object may cause some graphics cards to display random polygons on the display screen, so it is best to use a value slightly **above** the surface.
`up`	Defines which axis points up, X [1 0 0], Y [0 1 0] or Z [0 0 1].
`longitude` `latitude`	These two values describe the surface coordinates you want to be positioned above, and should be specified as decimal values. Longitude is specified as a negative number for the Western hemisphere and as a positive number for the Eastern hemisphere. Latitude is specified as a negative number for the Southern hemisphere, and as a positive number for the Northern hemisphere.

Do not use a negative number for the `time` argument. This will freeze your Celestia display, without warning, and you will then need to press the Esc key to reset the display.

***** Before using the `gotolonglat` command, you must first use the `select` command to select an object.

***** You can use other non-movement commands, such as **`print`**, while the camera is moving towards its destination.

***** Before using other commands that have a time argument, you ***must*** execute a wait command with a duration equal to or greater than the value of the goto command's `time` argument. Otherwise, the camera **will not** execute the goto command, but will instead execute the next time-based command.

This example selects the Earth and positions the camera over Seattle, Washington, USA:

```
select      { object "Sol/Earth" }
synchronous { }
gotolonglat { time 5
              distance 3
              up [0 1 0]
              longitude -122
              latitude 47 }

print       { text "Traveling to Seattle, Washington, USA."
              row -3  column 1  duration 5 }
wait        { duration 5 }

print       { text "Hovering over Seattle, Washington, USA."
              row -3  column 1  duration 5 }
wait        { duration 5 }
```

---

labels

```
set   <String> (no default)
clear <String> (no default)
```

`Set` (turn **on**) or `clear` (turn **off**) labeling of one or more of the items below. The `set` and `clear` string values can be any combination of the following items. Multiple values are specified within a single set of quotes (") by separating them with a vertical bar "|" (ie. `"moons|stars"`):

- `asteroids`
- `comets`
- `constellations`
- `galaxies`
- `moons`
- `planets`
- `spacecraft`
- `stars`

The following two examples demonstrate how to `clear` and `set` labels:

```
labels { clear "comets|constellations|galaxies|stars" }
labels {   set "asteroids|moons|planets|spacecraft" }
```

---

lock

`(null)`

This command *locks* two objects together (please refer to the Coordinate Systems section earlier in this guide). As when running Celestia interactively, this coordinate system involves *two* objects – the first object selected is called the **reference** object, and the second object selected is called the **target** object. When two objects are **lock**ed together as a pair, as you rotate the scene, the target object stays locked to the reference object, and the distance displayed is the distance *between the two objects*.

The Z axis points from the **reference** object (first object selected) to the **target** object; the Y axis lies in the plane defined by the Z axis and the rotation axis of the **reference** object, at a right angle to the Z axis. The X axis completes the right-handed coordinate system.

In Lock mode, your position remains fixed relative to a line between the reference object and the target object. As the target object moves around the reference object (relatively speaking) so do you. Thus if the target object is Sol, the illuminated fraction of the reference object (its "phase") remains the same because you're moving around the object in sync with Sol.

Only *one* Coordinate System can be active at any given time:

- `chase` (Chase)
- `equatorial`
- `ecliptical` (Follow)
- `geographic` (Sync Orbit)
- `lock` (Lock)
- `observer`
- `universal` (Free – ie. Esc key pressed)

The example below will maintain your position with respect to the center of the Earth, and keep both the Sun (Sol) and the Earth at a fixed location in the camera view. It's the same as if you typed the following keys on your keyboard: "3 f 0 :" (0 is a zero).

```
select { object "Sol/Earth" }
follow { }
select { object "Sol" }
lock   { }
```

---

lookback

`(null)`

Change the current camera view by 180 degrees (like a rear-view mirror).

Example:
```
lookback { }
```

---

mark

```
object <String> (no default)
size   <Number> (default 10.0)
color  <Vector> (default [1 0 0] - Red)
symbol <String> (default "diamond")
```

This command marks the `object` with the specified `symbol` of the specified `size` and `color`.

`object`	Name of the object to be marked.
`size`	Diameter, in pixels, of the symbol.

`color`	Defines the color of the symbol. The three number values define the strength of Red, Green and Blue (RGB) respectively. Allowable values are from 0 to 1, with decimals being specified with a leading 0, such as 0.5. Any value over 1 is handled the same as if it were specified as 1. Here are some example color values:
	`Black:  [0 0 0]`
	`Red:    [1 0 0]` *(default)*
	`Green:  [0 1 0]`
	`Blue:   [0 0 1]`
	`Yellow: [1 1 0]`
	`White:  [1 1 1]`

`symbol`	Defines what shape of symbol to mark the object with, which must be one of the following string values: * `diamond` *(default)* * `plus` * `square` * `triangle` * `x`

If the object to be marked was previously marked, or if you are unsure, you **must** `unmark` the object first.

The following example marks the Earth with a green "x":

```
unmark { object "Sol/Earth" }
mark   { object "Sol/Earth"
         size 15
         color [0 1 0]
         symbol "x" }
```

---

move

```
duration <Number> (no default)
velocity <Vector> (no default)
```

Move the camera at the specified `velocity` for the specified `duration`. A `wait` command is not necessary after a `move` command.

`duration`	Number of seconds to take during the move.
`velocity`	Speed, in km/second.

Do not use a negative number for the* `duration` *argument, as the command will be ignored.

This example moves the camera for 10 seconds at a speed of 100,000 km/second:

```
move { duration 10 velocity 100000 }
```

---

orbit

```
duration <Number> (default 1.0)
rate     <Number> (default 0.0)
axis     <Vector> (no default)
```

Orbit the currently selected object using the currently defined Coordinate System. You must first use the `select` command to select an object, and optionally use the `setframe` command to define a Coordinate System, if it is not currently defined.

`duration`	Number of seconds to orbit the object.
`rate`	Speed at which to orbit the object. Positive and negative values are used to indicate the direction of orbit.
`axis`	Define which axis to orbit around [X Y Z]. Set the `X`, `Y`, or `Z` value to 1 for yes, 0 for no. You may also specify multiple axes. For example, to specify `X` as the axis, use `[1 0 0]`, for `Y` use `[0 1 0]`, and for `Z`, use `[0 0 1]`. To specify both, the `X` and `Y` axes, use `[1 1 0]`.

Do not use a negative number for the* `duration` *argument, as the command will be ignored.

The following example orbits Saturn for 12 seconds:

```
select { object "Sol/Saturn" }
center { }
goto   { time 3
         distance 8
         up [ 0 1 0 ]
         upframe "equatorial" }
wait   { duration 3 }
orbit  { axis [ 0 1 0 ]
         rate 30
         duration 12 }
```

preloadtex

```
object <String> (no default)
```

Pre-load the specified texture file from disk into memory.

`object`: The filename itself, not including a path. Celestia uses its internal search lists to find the file. Those search lists are often "context dependant", being partially determined by the location of the add-on directory containing the `.ssc` file defining the selected object. This means, of course, that Celestia may not load the surface texture that the author of the script expects. Note also, that the file extension, the part after the ".", can be an asterisk "*", which will match any of `.png`, `.jpg` or `.dds`, in that order.

Depending on the file size of the texture being loaded, you may want to follow this command with a `wait` command. If you are preloading multiple textures, you should definitely use a `wait` command. The `wait` duration will depend on the size of the texture(s) and may require some testing to get it just right.

This example pre-loads the texture file `"mars.jpg"` into memory:

```
preloadtex { object "mars.jpg" }
```

print

```
text     <String> (no default)
origin   <String> (default "bottomleft")
duration <Number> (default 1.0)
row      <Number> (default 0)
column   <Number> (default 0)
```

The `print` command allows you to display text in Celestia's display window, defining where the text is positioned and for how long it should be displayed. You **must** follow the `print` command with a `wait` command of a duration equal to or greater than the specified duration is required.

Beware: If the text you want printed is longer than the width of the user's window, it will run off the right edge of the window.

`text`	The text you want displayed, surrounded by quote marks. You may also use "\n" to generate a CR/LF at any position within the text string.

origin	Starting position of the first character of your text. Must be one of the following values: * bottom * bottomleft * bottomright * center * left * right * top * topleft * topright Description of the origins: bottom: Bottom-center of the screen (not center justified). bottomleft: Left edge, bottom of the screen. bottomright: Right edge, bottom of the screen. You must use a **negative** column value, or your text will be outside the right edge of the window. center: Center of the display screen (not center justified). left: Left edge, half way down the screen. right: Right edge, half way down the screen. You must use a **negative** column value, or your text will be outside the right edge of the window. top: Top-center of the screen (not center justified). topleft: Left side, top of the screen. topright: Right edge, top of the screen. You must use a **negative** column value, or your text will be outside the right edge of the window. Use row and column to adjust the origin vertical and horizontal positions.
duration	Number of seconds to display the text message.

`row`	Defines the starting display line (vertical position), ***offset*** from the `origin` you specify, on which the text should be displayed. Positive values move the starting row **down**, while negative values move the starting row **up**. For example, if you specify an origin of `left`, but you want it to start three rows *higher*, then you would specify a `row` value of -3. If you wanted the text to start three rows *lower* than the `origin`, you would specify a `row` value of 3.
`column`	The column number (horizontal position) where the first character of the text should be displayed. Column 0 (zero) is the left edge of the display screen.

The `print` command must be followed by a `wait` command with a `duration` equal to or greater than that of the `print` command. Otherwise, you will not see the text you printed to the screen.

Do not use a negative number for the* `duration` *argument, as the command will be ignored.

This example prints the message "Hello Universe!" three rows up from the bottom of the display screen, starting two columns from the left edge:

```
print { text "Hello Universe!"  row -3  column 2 }
wait  { duration 1 }
```

This example prints the word "Hello" five rows up from the bottom of the display screen, starting two columns from the left edge, and on the next line, prints the word "Universe!":

```
print { text  "Hello\nUniverse!"
        origin "left"
        duration 5
        row -5
        column 2 }
wait  { duration 5 }
```

---

renderflags

```
set   <String> (no default)
clear <String> (no default)
```

`set` (turn **on**) or `clear` (turn **off**) one or more of the items below to be displayed on the screen. The `set` and `clear` values can be any combination of the following values. Multiple values are specified within a single set of quotes (") by separating them with a vertical bar "|" (ie. `"orbits|stars"`). The items in the square brackets indicate the related keyboard command:

- `atmospheres ......` Atmospheres [Ctrl+a]
- `automag ..........` Auto Magnitude [Ctrl+y]

- `boundaries ....... ` Constellation boundaries [Ctrl+b]
- `cloudmaps ........ ` Cloud textures [i]
- `comettails ....... ` Comet tails [Ctrl+t]
- `constellations ... ` Constellation labels [Ctrl+/]
- `eclipseshadows ... ` Eclipse shadows [Ctrl+e]
- `galaxies ......... ` Galaxy rendering [u]
- `grid ............. ` Earth-based equatorial coordinate grid [;]
- `markers .......... ` Markers [Ctrl+k]
- `nightmaps ........ ` Night-side planet maps [Ctrl+l (L)]
- `orbits ........... ` Object orbits [o]
- `planets .......... ` Planet labels [p]
- `pointstars ....... ` [no longer used -- see `set` command]
- `ringshadows ...... ` Ring Shadows [no keyboard key]
- `stars ............ ` Stars [none]

Examples:

```
renderflags { set "automag|atmospheres|nightmaps" }
renderflags { clear "boundaries|galaxies|markers" }
```

---

rotate

```
duration <Number> (default 1.0)
rate     <Number> (default 0.0)
axis     <Vector> (no default)
```

Rotates the camera using the currently defined Coordinate System. You must first use the `select` command to select an object, and optionally use the `setframe` command to set Coordinate System, if it is not currently defined.

`duration`	The length of time, in seconds, to execute the `rotate` command.
`rate`	Speed at which to rotate the camera. Positive and negative values are used to indicate the direction of rotation.
`axis`	Define which axis to rotate around [X Y Z]. Set the `X`, `Y`, or `Z` value to 1 for yes, 0 for no. You may also specify multiple axes. For example, to specify `X` as the axis, use `[1 0 0]`, for `Y` use `[0 1 0]`, and for `Z`, use `[0 0 1]`. To specify both, the `X` and `Y` axes, use `[1 1 0]`.

The `rotate` command must be followed by a `wait` command with a `duration` equal to or greater than that of the `rotate` command.

Do not use a negative number for the* `duration` *argument, as the command will be ignored.

The following example rotates the camera to the right (positive `rate` value) along the `Z` axis of the currently selected object for 5 seconds:

```
rotate { duration 5
         rate 10
         axis [0 0 1] }
wait   { duration 5 }
```

---

select

```
object <String> (no default)
```

Selects the specified `object` (name of a moon, planet, galaxy, spacecraft, etc.) in order to perform other commands on it. If you are currently positioned *outside* of our solar system, and want to select an object *inside* of our solar system, you must include "`Sol/`" (our sun's name) in the `object` value, as shown in the example below.

If the object does not exist (ie. a misspelled name), the script will simply continue to the next command without selecting any object at all, even if an object was previously selected.

Examples:

```
select { object "M33" }      #(Galaxy M33)
select { object "Mars" }     #(When in our Solar System)
select { object "Sol/Mars" } #(When outside of our Solar System)
```

---

set

```
name  <String>             (no default)
value <Number> or <String> (default 0.0 or "")
```

Set one of the items listed below to the value specified in the value argument.

`name`	Must be one of the following values: * `MinOrbitSize` * `AmbientLightLevel` * `FOV` * `StarDistanceLimit` * `StarStyle` (How to display stars) Descriptions for the above: `MinOrbitSize`: This value is used when Celestia's *render orbits* option is turned on. In general, when an object is very distant its orbit path will not be shown. MinOrbitSize is the minimum radius (in pixels) which the orbit path must cover before Celestia will draw it. `AmbientLightLevel`: Brightness level of Ambient Light, 0.0 to 1.0 is a good low to high range. `FOV`: Field of View. Celestia computes this value relative to the size of the display window, in pixels. The keyboard keys used to adjust

	FOV are the coma and period ("," and "."). StarDistanceLimit: The furthest distance at which Celestia will display stars. (Default value is 1000000.) StarStyle: This allows you to set how Celestia displays stars on the screen. The StarStyle value must be one of the following: * fuzzypoints * points * scaleddiscs
value	The value you want to assign to the name parameter.

Examples:
```
set { name "FOV"       value 35.5 }
set { name "StarStyle" value "points" }
```

---

setfaintestautomag45deg

```
magnitude <Number> (default 8.5)
```

Set the lowest Magnitude of stars to be displayed when Auto-Magnitude is *on*.

magnitude defines the magnitude at which stars will become visible, when Auto-Magnitude is *on*. The Celestia user interface allows a range from about 1.0 to 15.0.

Example:
```
setfaintestautomag45deg { magnitude 9.0 }
```

---

setframe

```
ref      <String> (no default)
target   <String> (no default)
coordsys <String> (default "universal")
```

Set the currently active Coordinate System (please refer to the Coordinate Systems section earlier in this guide).

ref	Defines the reference (first) object.
target	Optional - Defines the target (second) object, for 2-object coordinate systems, such as lock.
coordsys	Must be one of the following values: * chase * ecliptical * equatorial * geographic * lock * observer * universal

Only *one* Coordinate System can be active at any given time:

- `chase` (Chase)
- `equatorial`
- `ecliptical` (Follow)
- `geographic` (Sync Orbit)
- `lock` (Lock)
- `observer`
- `universal` (Free – ie. Esc key pressed)

The following example sets the Coordinate System to `lock`, which locks the Earth and Moon together on the display:

```
setframe { ref "Sol/Earth"
           target "Sol/Earth/Moon"
           coordsys "lock" }
```

---

setorientation

```
angle <Number> (no default)
axis  <Vector> (no default)
 -or-
ow <Number> (no default)
ox <Number> (no default)
oy <Number> (no default)
oz <Number> (no default)
```

Sets the camera orientation. If you are attempting to duplicate a position based on a Bookmark or Cell://URL, you will also need to set the proper Coordinate System, position, and other parameters. (Also see `seturl`.)

The `angle` and `axis` values may be obtained from Bookmarks, which are stored in the `favorites.cel` file located in the Celestia directory.

The `ow`, `ox`, `oy` and `oz` values represent the `angle` and `axis` as stored by a Cell://URL. They are obtained from the following Cell://URL values:
`&ow=`, `&ox=`, `&oy=`, and `&oz=`.

Examples:

```
setorientation {
   angle 0.945208
   axis  [ 0.81466 -0.570975 -0.101573 ] }

setorientation {
  ow 0.090610
  ox -0.494683
  oy 0.860207
```

```
    oz -0.084397 }
```

---

setposition

```
base   <Vector> (no default)
offset <Vector> (no default)
 -or-
x <Base64> (no default)
y <Base64> (no default)
z <Base64> (no default)
```

Moves the camera to a specific position in the 3-dimensional universe. If you are attempting to duplicate a position based on a Bookmark or Cell://URL, you will also need to set the proper Coordinate System, orientation, and other parameters. (Also see `seturl`.)

The `base` and `offset` values may be obtained from Bookmarks, which are stored in the `favorites.cel` file located in the Celestia directory.

The `x`, `y` and `z` values represent the `base` and `offset` values as stored by a Cell://URL. They are obtained from the following Cell://URL values: `?x=`, `&y=`, and `&z=`.

Both examples below position the camera to the same exact position in space outside of the Milky Way:

```
setposition {
    base [ -64132.43  47355.11  196091.57 ]
  offset [ 0  0  -1.52e-005 ] }

setposition {
  x "AMDCXoJK/3+IyGgR8f///w"
  y "BvP2LRdAAAD/n5UGCw"
  z "VUrGoeQJ+/8DyvenLQ" }
```

---

setsurface

```
name <String> (no default)
```

Allows you to define the name of an alternative surface for the currently selected object. The `select` command must be used first, to select an object.

`name` defines the filename itself, not including a path. Celestia uses its internal search lists to find the file. Those search lists are often "context dependant", being partially determined by the location of the add-on directory containing the `.ssc` file defining the selected object. This means, of course, that Celestia may not load the surface texture that the author of the script expects. Note also, that the file extension, the part after the ".", can be an asterisk "*", which will match any of `.png`, `.jpg` or `.dds`, in that order.

To use the "limit of knowledge" textures, instead of the interpretive ones, use:

```
setsurface { name "limit of knowledge" }
```

Example:
```
setsurface { name "my_mars.jpg" }
```

---

setvisibilitylimit

`magnitude <Number> (default 6.0)`

Set the Magnitude of stars to be displayed, when Auto-Magnitude is *off*. The Celestia user interface allows a range from about 1.0 to 15.0.

`magnitude` defines the magnitude at which stars will become visible, when Auto-Magnitude is *off*.

Example:
```
setvisibilitylimit { magnitude 6.5 }
```

---

seturl

`url <String> (no default)`

Move the camera to the location of a saved "location URL" (or Cell://URL), which you capture to the clipboard using the Ctrl+C or Ctrl+Ins keys.

`url` defines the Cell://URL to be used.

When you use the `seturl` command, **all** of the saved settings stored in the Cell://URL are transferred to the user's currently running copy of Celestia.

Example:
```
seturl { url
 "cel://Follow/Sol/2002-09-
01T16:58:51.32378?x=AAAAAMCrktOdILb5/////w&y=CkxgAADEXts7E7HX/////w&z
=VcpQAAAAAAAAAMIy/////w&ow=0.090610&ox=-0.494683&oy=0.860207&oz=-
0.084397&select=Sol&fov=49.499992&ts=1.000000<d=0&rf=6043&lm=0" }
```

Some example Cel:// URLs ...

Solar eclipse

Moon and Earth Cross Sol

ISS rotating over Sol

Sol-rise, ISS over Earth

Sol-rise from ISS

Sol-rise from ISS 2

Sol-set, ISS over Earth

synchronous

`(null)`

Orbit the currently selected object in synchronus orbit mode. This activates the `geographic` Coordinate System (please refer to the Coordinate Systems section earlier in this guide).

The Geographic Coordinate System allows you to remain in a **stationary**, or **geosynchronous** orbit above the selected object (not just Earth). As the object rotates below, the camera moves with it, as if it were attached to the object. The keyboard command for Sync Orbit is **[Y]**.

In the Geographic Coordinate System, the axes rotate *with* the selected object. The `Y` axis is the axis of rotation – counter-clockwise, so it points north for prograde rotators like Earth, and south for retrograde rotators like Venus. The `X` axis points from the center of the object to the intersection of its zero longitude meridian and equator. The `Z` axis (at a right angle to the `XY` plane) completes the right-handed coordinate system. An object with constant geographic coordinates will thus remain fixed with respect to a point on the surface of the object.

The `select` command must be used first, to select an object.

Only *one* Coordinate System can be active at any given time:

- `chase` (Chase)
- `equatorial`
- `ecliptical` (Follow)
- `geographic` (Sync Orbit)
- `lock` (Lock)
- `observer`
- `universal` (Free – ie. Esc key pressed)

Example:

```
select { object "Sol/Earth" }
synchronous { }
```

time

`jd <Number> (Julian Date, no default)`

-or-

`utc <String> (Universal Time, no default)`

Sets the Date and Time using a Julian Date (a decimal number) or a UTC time string, in the format `YYYY-MM-DDTHH:MM:SS.SSSSS`.

To obtain a Julian Date from a normal calendar date, try the U.S. Navy Calendar Date/Time to Julian Date/Time converter, located at: http://aa.usno.navy.mil/data/docs/JulianDate.html.

Examples of how to set the date and time to August 11, 2003 at 9:29:24 UTC:

```
time { jd 2452862.89542 }
```

-or-

```
time { utc "2003-08-11T09:29:24.0000" }
```

---

timerate

`rate <Number> (default 1.0)`

Set the multiplier (speed) at which time changes.

`rate` defines the time multiplier (ie. 100x).

Special values:
0 = Pause time
1 = Reset to Real Time
Negative values reverse time

This example sets the time multiplier to "1000x faster":

```
timerate { rate 1000 }
```

---

track

`(null)`

Track the currently selected object, which keeps it centered in the display. The `select` command must be used first, to select an object to be tracked.

For example, release your hold on any currently selected object (`cancel`), select the Earth (`select`), `goto` the Earth, and then `track` it. The Earth will begin to recede from you at the speed it actually travels in space, but Celestia will `track` the Earth by keeping it centered in the display. The code example below demonstrates this, with time sped up by 1000x:

```
cancel   { }
select   { object "Sol/Earth" }
goto     { time 3 distance 7 }
wait     { duration 5 }
track    { }
timerate { rate 1000 }
```

If you want the camera to remain at a constant distance from the object, add a `follow` command *after* the `track` command.

If the currently selected object has `track` enabled, as of Celestia version 1.3.1 you can use the following code to cancel a `track` command:

```
select { object "" }
track  { }
```

The `cancel` command resets the Coordinate System to Universal. Thus, if you want to use a different Coordinate System, you must use the `setframe` command before using `track`.

---

unmark

`object <String> (no default)`

If an object was previously `marked`, this command *un*marks it.

`object` defines the name of the object you want to `unmark`.

Example:
```
unmark { object "Sol/Earth" }
```

---

unmarkall

`(null)`

This command removes any previously assigned marks from **all** objects and disables the display of marks.

Example:
```
unmarkall { }
```

---

wait

`duration <Number> (default 1.0)`

Pauses execution of the script for the specified `duration` value in seconds.

Do not use a negative number for the* `duration` *argument, as the command will be ignored.

The following example pauses script execution for 15 ¼ seconds:
```
wait { duration 15.25 }
```

Deprecated Celestia Script Commands

The commands listed in this section have been replaced with newer, more powerful or more general commands. Though they continue to work for now, you should use the new command.

setambientlight

`brightness <Number> (default 0.0)`

Set the brightness level of Ambient Light. This has been replaced with a setting in the `set` command.

```
-------------------------
---< end of document >---
-------------------------
```