HSpace Reference Document

For Version 4.2.2

November 21, 2002

Edited by Mark Hassman
mark@moosh.net

Table of Contents

HSpace Overview

HSpace represents years of diligent work by many people and just as many years of hard play by the players who have enjoyed it!

HSpace is currently maintained and developed by Kyle Forbes (Gepht), who can be found at http://www.hspace.org and via gepht@hspace.org.

What is HSpace? Simply put, HSpace is a space simulation package that is an easy addition to your MU*. It is fully 3-dimensional with full combat and navigation support for space vessels. Not only does it allow game players to travel realistically through space, but HSpace also provides many flexible methods for building your own tools and MU* code to produce things like ship upgrade terminals, navigational aids, etc.

HSpace comes from the name Hemlock Space, and it was first developed on the Hemlock MUSH series of games. It was released in 1997 and has been increasing in popularity ever since. Now, it is written in a more object-oriented gaming language -- C++. This allows even more realistic and flexible space modeling to enhance your gaming experience!

The goals of HSpace development are: easy administration, quick installation, realistic but simple playability.

New to HSpace?

Flying through space in an HSpace ship may initially seem difficult and complex, but it's really not! You simply must remember the following things:

- HSpace is 3D, that means X, Y, and Z coordinates.
- Most flight is based on headings and bearings.
  - Headings are where you want to go.
  - Bearings are where some object is located relative to you.
- All headings and bearings have two parts: an XY angle and a Z angle.
- You need speed to move.

Assuming you are in your vessel and "manning" a navigation console, here are some simple things you can do to get going:

1) Type *sysrep*

   This shows you your current engineering systems status. You need power to use your systems.

2) Type *sysperc reactor=100*

   This sets your reactor to 100% desired power output. You're going to need a lot of power from it to power your other systems. You can also use *syspower* <*system*>=<#> to set a fixed amount of power allocated to the system.

3) Type *computer*

   If the game tells you there is insufficient power, wait a while for the reactor to power up more. You can use *sysrep* to keep checking your power output.

4) Type *cpower on*

This tells the computer to allocate power to your console and all of its gadgets.

5) Type *stat*

You get a navigation status display, indicating your current heading, speed, etc.

6) If you're landed on the ground, type *undock*.

These are just some of the basic commands to get you started. From here, you'll probably want to power up the rest of your systems, set a heading, and set your speed. You can get a full listing of commands by typing help hs-commands.

Player Command Index

The following is a list of all commands that may exist on a given console. Although many commands are considered specified to a certain type of console (e.g. engineering), they may be available from other types of consoles. Where applicable, a G, E, or N has been placed next to the command to indicate a general (G) console command, an engineering (E) command, a navigation (N) command, or an operations (O) command.

UNLOCK(G)	UNLOAD(G)	TARGET(G)	QREP(G)	CPOWER(G)
CHEAD(G)	GREP(G)	FIRE(G)	CONFIGURE(G)	AUTOLOAD(G)
LOCK(G)	LOAD(G)	AUTOROTATE(G)	SYSREP(E)	SYSPOWER(E)
SYSPERC(E)	VSTATS(G)	SS(N)	HEAD(N)	AFTERBURN(N)
VIEW(N)	SCAN(N)	LAND(N)	DOCK(N)	MAPRANGE(N)
ABORT(N)	JUMP(N)	LINK(O)	UNLINK(O)	STAT(N)
NOBURN(N)	UNDOCK(N)	TAXI(N)	SREP(G)	CSTAT(G)
SPOSE(N)	ICOM(G)	BLINKS(N)	CLOAK(N)	DECLOAK(N)
CREWREP(E)	ASSIGN(E)	GATE(N)	FUELCONV(N)	BAYSTAT(O)
CLOSE BAY(O)	SET CODE(O)	SELFDESTRUCT(E)	ETA(N)	PLOT(N)
HSTAT(O)	TLOCK(O)	TDOCK(O)	TMODE(O)	

The following are Quick-start commands:

POWER	COMPUTER	SENSOR	ENGINES	THRUST
LIFE	FORE	AFT	PORT	STAR
COMM				

The following are general ship usage commands:

BOARD	DISEMBARK	BOARDING CODE	SECURITY	SHIPNAME
CHOWN	SPECS	DEADMAN		

If you are brand new to flying with HSpace, refer to the "New to Hspace" section of this document.

ABORT

Usage: abort

Takes the ship out of hyperspace if the ship is currently "jumping."

AFT

Usage: <shield>

Sets the specified shield power to 100% of capable power.

AFTERBURN

Usage: afterburn

If afterburners are present on the vessel, this command puts the vessel into afterburn mode, which basically dumps fuel into the engines to create additional thrust. Afterburning significantly increases the speed of the vessel but typically results in poor fuel efficiency.

ASSIGN

Usage: assign <#> to <system>

Assigns crew <#> to repair damage on <system>.

AUTOLOAD

Usage: autoload <on/off>

Sets the console's autoloading status to on or off. If set to on, the console will automatically attempt to load any loadable weapons after they are fired. If no munitions for that weapon remain in the ship's munitions storage, autoloading fails.

If autoload is set to off, loading must be performed manually with the load command.

AUTOROTATE

Usage: autorotate <on/off>

Sets the console's autorotating status to on or off. Autorotating is a convenient way to avoid manually changing the heading of the console each time you want to attack a target on sensors. If autorotating is turned on, the console automatically determines the needed heading to attack the target and attempts to turn to that heading.

BAYSTAT

Usage: baystat OR baystat <bay #>

Displays a summary of the ships bays OR the status of bay <#>.

BLINKS

Usage: blinks

Displays a list of boarding links currently active between the current ship and other ships in the area.

BOARD

Usage: board <ship>[=<code>]

Allows you to board the given ship with a specified name. The ship will obviously be docked in a docking bay or landed on a planet-landing pad. An optional code may be required to board the vessel.

See Also: DISEMBARK, BOARDING CODE, SECURITY

BOARDING CODE

Usage: boarding code or boarding code <code>

Sets the boarding code for the vessel, which means that a player wishing to board the vessel from the outside will need to specify a code. By supplying no <code>, the boarding code for the vessel will be deleted, and anyone may board.

CHEAD

Usage the same as HEAD. Refer to HEAD help file.

CHOWN

Usage: chown <player name>

Allows the owner of a ship to change the ownership to another player in the game.
See Also: BOARD, SECURITY

CLOSE BAY

Usage: close bay <bay #>

Closes the doors of bay <#>.

CLOAK

Usage: Cloak

Cloaks the vessel decreasing its sensor signature.

CONFIGURE

Usage: configure <weapon #>=<munitions #>

Attempts to configure the given weapon on the grep display to the specified <munitions #>, which is a type of munition in the ship's storage. Typically the weapon must be empty before it can be reconfigured. Once configured, the weapon will load and fire that munitions type.

COMPUTER

Usage: computer

Sets computer power to 100% of capable.

CONVFUEL

Usage: convfuel <qty> to <type>

Converts <qty> of Cadmium in your ships hold into <type> of fuel, where type is 'burnable' or 'reactable'.

CPOWER

Usage: cpower <on/off>

Turns the console on or off. This is required before using any navigation or console commands. Engineering commands do not require console power. Power for consoles is extracted from the internal computer. For each of the weapons located on a console, additional power is typically needed. Thus, if damage to the internal computer is encountered or the internal computer, for whatever reason, cannot accommodate enough power, some consoles may not be able to obtain the needed power.

CREWREP

Usage: crewrep

Displays damage control crew status.

CSTAT

Usage: cstat

Shows the current communications array status, including transmission and receiving frequencies.

DEADMAN

Usage: deadman

Purpose, to make you dead! Just kidding.

The deadman command allows you to safely leave your ship in space when needed (certainly useful for capital ships that cannot land). The deadman command takes 2 minutes to go into affect, preventing someone from using it instantly to avoid getting shot at. Once the deadman mode is activated for the ship, the ship cannot be locked on by another ship who has not already locked weapons (ships with locks maintain them and can continue attacking).The ship cannot be used again by the crew without deactivating the deadman mode. Thus, setting deadman mode requires that you discontinue using the ship once the mode is active.

DECLOAK

Usage: Decloak

Decloaks the vessel eliminating all effects of the cloaking device.

DISEMBARK

Usage: disembark

When located in a room on a ship marked as a disembarking room, the disembark command will take you from the ship to the docking bay or planet room where the ship is docked. This command will not work if the ship is actively in space.

See Also: BOARD

DOCK

See LAND

ENGINES

Usage: engines

Sets engine power to 100% of capable power level.

ETA

Usage: eta

Returns the eta of the planet that lies on your current heading, or the one closest to it. You must be moving at a speed greater than 0 for this command to work.

FIRE

Usage: fire <weapon #>

Attempts to fire the given <weapon #> on the console. This assumes that weapons are present. Some weapons may require a target lock, while others may not.

FORE

Usage: <shield>

Sets the specified shield power to 100% of capable power.

GATE

Usage: gate <contact #>

Gates with a contact (i.e. a Wormhole).

GREP

Usage: grep

For consoles controlling weapons, the grep command gives a gunnery report, indicating which weapons are present, their status, weapon attributes, and other information about the gunnery status of the console. Information about the ship's munitions storage is also presented.

HEAD

Usage: head <XY> mark <Z>

Changes the ship's or console's heading to the desired angles. Valid XY angles are from 0 to 359, while valid Z angles are from 90 to -90 (straight up, straight down).

Many consoles can move independently of where the ship is heading, thus making gunnery turrets possible. Even though the ship may be heading 0 mark 0, it is possible for a side-facing turret to turn in other directions to defend from attacking enemies.

For consoles, it may or may not be possible to turn to the desired heading due to restrictions on where the console is located on the sides of the ship. If the console has been setup to deny rotation, it may not be rotated from the current heading.

HSTAT

Usage: hstat

Displays the status of docking links on the ship.

ICOM

Usage: icom <message>

Allows consoles on multi-console ships to communicate with each other.

INTER

Usage: inter <contact>

Plots an intercept course for <contact>.

JUMP

Usage: jump OR jump <contact>

Takes the vessel into hyperspace if jump drives are present. The vessel much first achieve a minimum sub light speed before jumping to hyperspace. Alternatively, if no jump drives are present then jump <contact> can be used with jump gates.

LAND

Usage: land <contact>/<location>=[<code>]

The land command attempts to land the vessel at the given contact number, which could be a planet or another vessel. The <location> refers to which of the landing pads or docking bays is desired, beginning with 1.To land at the given location, a code may need to be given. If a code is not required for the location, this parameter is ignored.

The LAND and DOCK commands are identical and work for both planets and vessels.

LIFE

Usage: life

Sets life support power to 100% of capable.

LINK

Usage: link <contact>=<local port>/<remote port>

Links your ship to another.

LOAD

Usage: load <weapon #>

Attempts to load the given weapon, provided sufficient munitions is available and the weapon can be loaded.

LOCK

Usage: lock <contact ID>

Locks the console's weapons (if available) on the target object with the given <contact ID> on sensors. The contact must be within range of the console's weapons, which is determined by the weapon on the console with the maximum range.

MAPRANGE

Usage: maprange <range>

The maprange command allows the scale and limits of navigation map to be changed. The <range> indicates how far from the ship objects are included in the map. For example, a range of 100 means that any objects on sensors within 100 units of the ship will be displayed in the map. To view close range objects, it may be necessary to reduce the <range> so that those objects appear more "spread out" on the map. As the <range> increases, objects near to each other will get grouped into the same location on the map.

NOBURN

Usage: noburn

If the ship is currently afterburning, this command will cease the process and take the ship back into regular speed levels.

OPEN BAY

Usage: open bay <bay #>

Opens the doors of bay <#>.

PLOT

Usage: plot <x> <y> <z>

Shows details of the course required to move from current position to coordinates <x> <y> <z> along with the distance, and fuel required.

PORT

Usage: <shield>

Sets the specified shield power to 100% of capable power.

POWER

Usage: power

Sets the desired reactor output to 100% of capable power.

QREP

Usage: qrep

Displays gunnery heads-up for the current target lock. The heads up contains a forward display of where the target is located relative to the current heading of the console. Information about your heading and the target status is also displayed.

SCAN

Usage: scan <contact ID>

Scans a target object in space to provide information about that contact. The <contact ID> comes from the sensor report for the vessel doing the scanning.

SECURITY

Usage: security <on/off>

Turns the ship security on or off, effectively activated or deactivating the required boarding code. If no boarding
code is set on the ship, this command has no effect. If a boarding code is set, and security is turned off, the boarding code is not required for players wanted to board. Once security is turned on, the boarding code is reactivated and required.

See Also: BOARD, BOARDING CODE

SELFDESTRUCT

Usage: selfdestruct <seconds>=<boarding code>

Initiates the vessel's Self Destruct sequence, it can be aborted by entering 0 as the value of <seconds>, otherwise the ship will be destroyed after <seconds> have passed, and all onboard crew killed. This command will give warnings to everyone on the ship at:

> 1 HOUR left, 20 MINUTES left, 5 MINUTES left, 1 MINUTE left, 30 SECONDS left, 20 SECONDS left, and each second below 10.

SENSOR

Usage: sensor

Sets the sensor array power to 100% of capable.

SET CODE

Usage: set code <bay #>=<code>

Sets the code for bay <#> to <code>.

SHIPNAME

Usage: shipname <name>

Sets the name of the vessel from that navigation console. The name is limited to a length of 32 characters.

SPECS

Usage: specs [<system>]

Provides vessel statistics in general or, if <system> is specified, for a specific system on the vessel.

SPOSE

Usage: spose <action>

Emits a pose for your ship into the nearby space so that other vessels that have you identified on sensors see the message.

Ex: spose flies near you, blazing its laser cannons.

SREP

Usage: srep

Prints a sensor report for the vessel's sensor array, if present. Administrators may or may not have customized this command to print only certain contact types.

A map of the surrounding space is available for navigation, and the "map range" can be set to change the scale and limits of the area within the map.

SS

Usage: ss <velocity>

Sets the desired velocity for the vessel, assuming the <velocity> is valid. Engine damage, available power, and other factors may limit the maximum capable speed of the engines.

STARBOARD

Usage: <shield>

Sets the specified shield power to 100% of capable power.

STAT

Usage: stat

Gives the navigation status display for the vessel, including current course and velocity. If present, shield information is included.

SVIEW

Usage: sview <contact #>

Displays the description of the target.

SYSPERC

Usage: sysperc <system name>=<% power>

Sets the power allocation for the given system to a percentage of the maximum power needs for the system. If 100% is specified, the reactor attempts to allocate 100% of the system's power needs. This is useful for when you want to quickly set a system's power usage to a given level (e.g. 100%).

Example: sysperc life support=100

SYSPOWER

Usage: syspower <system name>=<#>

Sets the megawatts of power allocated to a given system on the ship. If insufficient power is available from the reactor, this operation fails.

Example: syspower life support=10

You can abbreviate system names with most engineering commands. For example, you can specify life instead of life support.

SYSPRIORITY

Usage: syspriority <system name><+/->

Changes the system priority of the specified system for the ship. The <system name> must be one of the systems listed on the sysrep display for the ship. It's status can be changed upward (+) or download (-) on the list. This is useful for when the ship may lose power. If power is lost for any reason, the ship's computer will begin deallocating power from systems at the top of the sysrep list. For systems you do not want power immediately drained from, lower their priority on the list.

For example, you may want life support to be the last thing that gets power drained from it. Thus, you can lower its priority repeatedly with:

```
syspriority life support-
```

Do this until the priority is lowered to your satisfaction.

SYSREP

Usage: sysrep

Displays a list of systems currently present on the ship, information about their status, damage, and power allocation.

TARGET

Usage: target <system name>

Attempts to target the given <system name> on the target object. Any weapons on the console that support system targeting will attempt to do damage to that system on the enemy object. Not all weapons may support system targeting.

To break a system target lock, specify a value of "none" for the system name.

TAXI

Usage: taxi <direction>

Taxis a docked or landing ship in the <direction> desired, which is the name of an exit in the room where the ship is located.

TDOCK

Usage: tdock <bay>=<contact id>

Docks a contact in a local bay using the tractor beam.

See also: HS-TRACTORHS_CLONE

THRUST

Usage: thrust

Sets maneuvering thruster power to 100% of capable.

TLOCK

Usage: tlock <contact id>

Locks the tractor beam on a contact.

See also: HS-TRACTOR

TMODE

Usage: tmode <repulse/tractor/hold>

Specifies the mode to which the tractor beam should be set.

See also: HS-TRACTOR

TRUSTED

Usage: trusted/add <person> OR trusted/remove <person>

Adds or removes <person> from the trusted persons list for your ship.

UNDOCK

Usage: undock

Undocks a landed or docked vessel from its location, which must be either a docking bay or landing pad on a planet.

UNLOAD

Usage: Unload <weapon #>

Attempts to unload the currently loaded weapon. This is only applicable for weapons that support this feature. Once unload, the munition from the weapon is put back into the ship's munitions storage for later use.

UNLOCK

Usage: unlock

If the console is currently locked onto a target object, the unlock command disengages the console's lock. This need not
be used before establishing another target lock. If another lock is obtained while currently locked on a target, the lock is changed from the previous to the new target.

UNLINK

Usage: unlink <local port>

Unlinks specified local port.

VIEW

Usage: view

Provides a description of the surrounding area where the vessel is currently docked or landed. This is useful in combination with the taxi command.

VSTATS

Usage: vstats

Displays a picture of the ship class for the ship, upon which shield and hull information may be included.

Weapon Attributes

Weapons loaded from the weapons database contain a variety of attributes to govern how they respond when fired. Each weapon type has a different set of attributes, but all weapons share some base sets of attributes. The following attributes can be set and retrieved for all weapon types:

Gettable/Settable attributes:

- NAME : The name of the weapon.

Gettable attributes only:

- TYPE : The type of the weapon (0 = lasers, 1 = missiles)

See Also: Laser Attributes, Missile Attributes

Laser Attributes

The following attributes may be set and retrieved on laser weapons loaded from the database:

Gettable/Settable attributes:

- REGEN TIME: Time it takes for the weapon to regenerate.
- RANGE: Range of the weapon for attacking.
- STRENGTH: Damage strength when attacking.
- ACCURACY: Accuracy of the laser.
- POWER: How much reactor power is required to power up.
- NOHULL: If set to 1 only shield/system damage is done.

See also: Weapon Attributes, Missile Attributes

Missile Attributes

The following attributes may be set and retrieved on missile weapons loaded from the database:

Gettable/Settable attributes:

- RELOAD TIME: Time it takes for the weapon to reload.
- RANGE: Maximum range the missile can reach.
- STRENGTH: Damage strength when the missile hits.
- TURNRATE: The degrees the missile can turn at one time.
- SPEED: Speed of the missile in units per hour.

See also: Weapon Attributes, Laser Attributes

Functions

HS_ADD_WEAPON(<console>, <class>)

Allows a weapon of ID <class> on the global list of weapons to be added to the console specified by <console>. This is identical to the @space/addweapon command. To be able to add a weapon, the player or object using the function must control the object the weapon is being added to.

HS_ADDSYS(<object>, <system>)

Adds a system to a vessel as in @space/addsys.

HS_CLONE(<object>)

This function currently only works on ships.

The hs_clone() function is a handy way of using @space/clone within a function. If successful, the dbref of the new clone is returned.

HS_COMM_MSG(<SID>,<DID>,<X>,<Y>,<Z>,<RANGE>,<FRQ>,<MSG>)

Sends a communications message into space, to be received by any objects with communications capability. The
object sending the message MUST have the HSPACE_COMM flag, or the message will be rejected. The parameters are:

- SID: Source universe ID.
- DID: Destination universe ID.
- X: Source X coordinate.
- Y: Source Y coordinate.
- Z: Source Z coordinate.
- Range: Maximum range to transmit.
- FRQ: Frequency upon which to transmit the message.
- MSG: Message to transmit.

HS_DELSYS(<object>, <system>)

Deletes a system from a vessel as in @space/delsys.

HS_ENG_SYS(<object>)

Returns a comma-delimited list of engineering systems located on the given HSpace <object>.

HS_GET_ATTR(<object>, <attribute>)

The HS_GET_ATTR function allows attributes for the specified object to be retrieved. The <object> parameter is any object in the game that represents an HSpace object (e.g. a ship). The <attribute> parameter is the name of a valid, retrievable attribute for that type of HSpace object.

For ships, any console can be used in place of the ship object to allow players without see_all powers to get attributes from the ship through the consoles. Players may only get attributes from HSpace objects they are standing in the same room as, unless they are granted the ability to see anywhere in the game.

HS_GET_MISSILE(<object>,<ID>)

Retrieves the current and maximum missile storage values for the specified weapon ID, which can be obtained from @space/list weapons. The values are returned as:

<current>/<max>

See Also: @space-setmissile

HS_SET_ATTR(<object>/<attribute>, <value>)

Only wizards may set attributes on the specified game <object>, which represents an object in HSpace (e.g. a ship). The <attribute> parameter is any valid, settable attribute for that type of HSpace object, while

<value> is a valid value for that attribute. Not all attributes that are retrievable for an object are settable and vice versa.

HS_SET_MISSILE(<object>, <type>, <num>)

Sets the current storage for the given weapon type (a missile) on the object (a ship).If <num> is greater than the current maximum storage value, the maximum value is raised to <num>.

See Also: HS_GET_MISSILE()

HS_SPACEMSG(0, <uid>, <X>, <Y>, <Z>, <dist>, <message>)

HS_SPACEMSG(1, <message>)

HS_SPACEMSG(2, <message>)

The HS_SPACEMSG function allows messages to be sent into space in a variety of ways. This can allow what is commonly referred to as "ship posing" such that players can type a command and have the ship send a message as if it were performing some action in space.

The first form of the function allows a wizard or HSpace object to send a message into the universe with the given <uid>, starting at the specified X, Y, and Z coordinates, and reaching all other HSpace objects within the specified <dist> range.

The second and third forms of the function allow an HSpace object to send a message from itself into space such that other objects with sensors receive the message. The second form sends a message to all HSpace objects with the source object on sensors, while the third form sends to all HSpace objects that have the source object IDENTIFIED on sensors. Thus, the third form is very useful for ship posing.

Ex:

think hs_spacemsg(2, The Excalibur comes screaming in for the kill.)

HS_SREP(<object>, <type>)

Retrieves the current sensor contacts for the specified HSpace <object>. The <type> parameter specifies the type of object contacts you wish to retrieve from the sensors. Refer to HS-TYPES for each of the HSpace types of objects. If the No Type of object is specified for the <type>, all sensor contacts are retrieved. A list of contacts is returned in the following format:

<contact> <contact> .. <contact>

Where <contact> contains the following information:

ID:Dbref:Type:Name:XY Bearing:Z Bearing:Distance

HS_SYS_ATTR(<object>/<system>, <attribute name>, <adjusted>)

Returns the value of the requested attribute with <attribute name> if the system is present on <object> (typically a ship) and possesses the specified attribute. The <system> must be a valid engineering system. The <adjusted> parameter can be set to 0 (unadjusted) or 1 (adjusted) to return the system information according to current damage and power levels. A value of 1 specifies that adjusted values should be returned, while a value of 0 specifies that the raw attribute value should be returned.

See Also: HS_SYSSET()

HS_SYSSET(<object>/<system>, <attribute name>[:<value>])

Sets the <value> of the given <attribute name> for the specified <system> on the <object>, which is usually a ship of sorts. The <value> can be left empty to clear the attribute and reset the system values to the defaults of the class it belongs to. For example, a ship of a certain class has default maximum speed values, but setting this value on the ship overrides the setting of the class. Clearing the value resets the maximum speed to that of the ship class.

See Also: HS_SYS_ATTR()

HS_WEAPON_ATTR(<class>,<attrname>[:<value>])

Allows information to be retrieved and set on a weapon loaded from the weapon database. The <class> is the ID of the weapon as shown on the @space/list weapons command. The <attrname> is a valid attribute that contains information on the weapon. Optionally, a value can be specified to set the new value for that attribute on that weapon. This is
restricted to wizards only.

See Also: HS-WEAPON-ATTRS

XYANG(<X1>, <Y1>, <X2>, <Y2>)

Returns the XY planar angle from the first pair of XY coordinates to the second.

ZANG(<X1>, <Y1>, <Z1>, <X2>, <Y2>, <Z2>)

Returns the Z planar angle from the first pair of XYZ coordinates to the second.

HSpace Administration

These HSpace administrative documents are technical documents. While they may include some discussion of approaches and various tips for running HSpace, they are intended more as a reference guide for command usage and step-by-step management of the various HSpace components.

This guide is divided into the following sections:

```
HS-QUICKSTART HS-FUNCTIONS HS-OBJECTS HS-CLASSES
HS-SYSTEMS HS-UNIVERSES HS-TYPES HS-CONSOLES
HS-WEAPONS HS-SHIELDS HS-TERRITORIES HS-HATCHES
CLASS PICTURES
```

Type 'help <section>' for information on each section.

HS-ASTEROID-ATTRS

The following attributes are settable and retrievable for planet objects.

- DENSITY: Density of the asteroid belt.

HS-BLACKHOLE-ATTRS

This object only has the basic attributes, it pulls in any ship within its size * 100, the damage varies from 0 at the edge of the gravity field to 1000 * the size in the center.

HS-CLASSES

What is a class? A "class" is an abbreviated reference to a ship class, so the next question is what is a ship class? We can think of a ship class by taking a simple look at modern day naval vessels. We have a variety of naval vessels present today, many of which are commonly known, such as the destroyer and battleship. Typically, each naval vessel belongs to a certain class of ship. For example, we may find some ship traveling the oceans called the USS Onlooker. It is a large ship, classified as a battleship. However, battleships can vary in how they're designed and what detailed purposes they serve. Thus, each type of battleship belongs to a "class" of battleship. The USS Onlooker might be of the Jersey Class Battleship, which means that it possesses certain attributes common to all battleships of the Jersey class.

Why the use of classes? It is typically uncommon to produce a single vessel with a given set of attributes and characteristics. Often we find at least several vessels that share similarities, and manufacturers can refer to designs for a class of vessel rather than a specific, single vessel. A manufacturer may produce 20, Tiger class sailboats rather than just a single sailboat called the Tiger. The manufacturer, then, profits from the ability to produce multiple ships of that design with relative ease.

In HSpace it is no different. Designing and constructing the initial ship class may be slightly tedious, but the work is much less for producing vessels of that class. Once the base class is in place, vessels of that class can be created quickly. After each vessel is created, administrators have the option to slight alter, or "componentized," pieces of each vessel to make them unique.

In addition to a variety of attributes that can be set on a given ship class, engineering systems must also be added to the class. This is because each type of vessel may contain its own set of engineering systems that differ from other ship classes. For example, ships of Ship Class A may possess jump drives, while ships of Ship Class B do not. For this reason, it is necessary to add engineering systems to the ship class that all ships of that class should possess. When ships of that class are created, they will possess the engineering systems given to that class.

To create a new ship class, refer to the '@SPACE-NEWCLASS' help file.

Refer to 'HS-CLASS-ATTRIBUTES' for information on setting specific attributes for classes once created.

HS-CLASS-ATTRIBUTES

The following attributes may current be set for any ship class:

- MAXHULL: Maximum number of hull points.
- NAME: Name of the class (e.g. Onlooker Class Battleship)
- SIZE: Size of the vessel from 1 .. n.
- CREWS: Number of repair crews stationed on the ship.
- CARGO: Size of the cargo bay on the ship.
- CAN DROP: Whether the ship can drop to a planet or not.
- MINMANNED: Amount of manned consoles needed to undock.
- SPACEDOCK: Allows all size ships to dock at this vessel.

Classes may also possess engineering systems. Refer to the @SPACE-ADDSYSCLASS help file for information about this feature.

CLASS PICTURES

The Vstats command that comes with HSpace 4.0 provides the ability for you to show a picture of the ship class to the player. This is easily accomplished by creating a pics directory in your space directory (configurable in hspace.cnf) that will include your class pictures. For each ship class (0 .. n), you simply create a file called class_#.pic, containing the ASCII art you wish to have displayed. Additionally, you can have the hull and shield strength percentages displayed by using the %h (hull), %f (foreshield), %p (port shield), %a (aft shield), %i (ident number), %n (name), %x (XY heading), %z (Z heading), %v (velocity) and %s (starboard shield) substitutions. For each of these substitutions, a 4 character percentage (e.g 100%) will be put in its place. Thus, keep this 4-character requirement in mind when drawing your art. What you see may not be what you get when it is displayed.

You do not need to reboot your game for changes to the artwork to be displayed. It is taken directly from the file when the vstats command is issued.

HS-CONSOLES

Consoles in HSpace allow players to interact with objects in HSpace. For example, a player on a ship may want to navigate that ship, so a navigation console could be created with a set of commands to allow the player to navigate the vessel.

In HSpace version 3.x and earlier, the navigation console was the "heart" of any vessel. In HSpace 4.0 this concept ceases to exist. It is the "ship object" which is the heart of the ship. All consoles on a ship then belong to that ship object (a MUSH object).

Any HSpace console can use any of the @nav, @eng, and @console commands, thus allowing you to derive many types of consoles from navigation to engineering and so on.

Consoles possess an added ability to make administration and development of MUSH tools easier. Most @space commands will work on consoles as if those consoles were the ship. For example, using the @space/set command requires that the ship object be specified, but any console on the ship can serve in its place.

See also: HS-CONSOLE-ATTRS, HS-CONSOLE-OFFSETS

HS-CONSOLE-ATTRS

Consoles provide the following attributes for setting and retrieving:

Settable Attributes:

- XYHEADING: The current XY angle heading of the console.
- ZHEADING: The current Z angle heading of the console.
- XYOFFSET: The XY offset angle from the front of the ship.
- ZOFFSET: The Z offset angle from the front of the ship.
- CAN ROTATE: A value of 1 if the console can change headings. Otherwise, a value of 0.

Gettable Attributes:

- CXYHEADING: The XY angle heading of the console.
- CZHEADING: The Z angle heading of the console.
- XYOFFSET: The XY offset angle from the front of the ship.
- ZOFFSET: The Z offset angle from the front of the ship.
- CAN ROTATE: A value of 1 if the console can change headings. Otherwise, a value of 0.
- LOCK: The current sensor contact ID of the console target lock.
- WEAPONS: A list of weapon types currently on the console.
- USER: DBRef of thing/player currently manning the console.
- ISPOWERED: 1 if the console is powered and 0 if not.

See also: HS-CONSOLE-OFFSETS

HS-CONSOLE-OFFSETS

HSpace consoles can serve many functions for the vessel they are attached to. They may serve as navigation consoles, engineering consoles, communications consoles, or gunnery consoles. For the latter type, the gunnery console, it is desirable to have the console turn as if it were a turret on a ship. To this end HSpace allows consoles to have XY and Z heading attributes that indicate where they are currently facing (if we can imagine a huge gun projecting out from the turret).If the CAN ROTATE attribute of a console is set to 1, the console can change its heading with the @console/head command. If set to 0, the console can not change it's heading, which may be appropriate if the guns are in a fixed position or the console has no weaponry. For example, engineering consoles don't need to change headings -- there is no purpose served by this feature.

The idea of a console offset is not quickly evident, but it will be evident after a short explanation. Every console that serves as a gunnery turret somewhere on the ship has a limited ability to turn to various angles. Even a turret located on the bottom of the ship can only attack targets relatively below the ship. Why? Because the bottom of the ship that the turret is attached to prevents that turret from rotating upward, into the ship, and firing at targets above the ship. Thus, each turret has an effective "hemisphere" of rotate angles. That is to say that it can rotate to angles on its side of the ship, which is 90 degrees left, 90 degrees right, 90 degrees up, and 90 degrees down. It cannot turn 120 degrees right because it would be pointing into the ship it is attached to.

But how do consoles know where they are located on a vessel? We must tell the console where it is located with the two offset attributes. The first, the XYOFFSET, indicates which side of the vessel the console is on based on an offset value from the front of the ship on the XY plane. That is to say that when you are standing in the middle of the ship, where is the console located to your left and right? A value of 0 indicates that the console is in front of you (on the XY plane), while a value of 90 indicates the console is directly to you right. A value of -90 indicates the console is directly to your left, and a value of 180 (or -180) indicates the console is located behind you. Similarly, the ZOFFSET attribute indicates where the console is located in the up and down directions. A value of 0 indicates the console is directly in front of you, while values of 90 and -90 indicate the console is either directly above you or directly below you. Again, these are OFFSET values from the front of the ship, such that XYOFFSET/ZOFFSET values of 0 put the console at the front of the ship. Whatever offset value the console possesses indicates its natural heading. Thus, offset values of 90/0 indicate that the console naturally points 90 degrees clockwise from the current ship heading and in the same Z heading of the ship. By specifying combinations of the XY and Z offset values, you can position your console anywhere on the sides of the vessel.

See also: HS-CONSOLE-ATTRS

HS-FUNCTIONS

Functions in HSpace allow administrators and players alike to develop "softcode" for working with HSpace. The following functions are currently provided:

- `HS_GET_ATTR`
- `HS_SET_ATTR`
- `HS_SPACEMSG`
- `XYANG`
- `HS_SREP`
- `HS_ENG_SYS`
- `ZANG`
- `HS_ADD_WEAPON`

- HS_WEAPON_ATTR
- HS_SYS_ATTR
- HS_SYSSET
- HS_GET_MISSILE
- HS_COMM_MSG
- HS_ADDSYS
- HS_DELSYS
- HS_CLONE

For more information on a specific function, refer to each function's help file. For example 'help HS_SPACEMSG()'.

HS-GATES

Jump Gates are a new softcoded feature with Hspace 4.2. You'll find that the jump command on the navigation has been altered to include jump <contact>.

This allows for jump gates to transport vessels without jump drives from one location to another.

They are set up as follows:

1. Create an object to represent the jump gate and add it to space as type 0.
2. Set the following attributes, jump_buoy, dest_x, dest_y, and dest_z.

The last three are the x y and z coordinates for the destination and jump_buoy just needs to be set to 1 to indicate the object is a jump gate.

HS-HATCHES

Hatches were implemented in version 4.11 to replace the old boarding links, hatches(represented by exits) can be added to ships with the @space/addhatch command, and there are several gettable attributes on them. Two ships can links together through the hatches while floating in space, they must be within 5 hm of each other.

See: HS-HATCH-ATTRS

HS-HATCH-ATTRS

There are 2 different gettable attributes on hatches.

- TARGETOBJ: The ship object of the ship the hatch is linked to.
- TARGETHATCH: The hatch # of the hatch on the other side.
- CLAMPED: If 1 this hatch can't be connected or disconnected.

See: HS-HATCHES

HS-LANDINGLOC-ATTRS

Landing locations are a generic type of HSpace object that allows ships to dock in the bays of other ships or to land on planets. Landing locations, like all HSpace objects, have attributes that can be set and retrieved. These are as follows:

Settable AND Gettable attributes:

- CODE: Code required to land in the location.
- ACTIVE: 1 if location is open for landing, 0 otherwise.
- CAPACITY: Object size capacity of the location.
- MAX CAPACITY: Maximum object size capacity of the location.
- VISIBLE: 1 if the location is visible on scan, 0 otherwise.

The "object size capacity" refers to the volume of the landing location and its ability to hold HSpace objects (i.e. ships). For each unit of "size capacity," the location can hold 1 unit of an HSpace object size. Thus, a landing location with a current capacity of 2 could hold two, size 1 objects or one, size 2 object. As objects (ships) enter and leave the landing location, the current capacity decreases and increases to reflect the space taken up in the location by the object.

HS-MESS-TYPES

These are the types of messages consoles in HSpace can receive:

#	Name
0	General
1	Sensors
2	Engineering
3	Combat
4	Communications

HS-NEBULA-ATTRS

The following attributes are settable and retrievable for planet objects.

- DENSITY: Density of the nebula.
- SHIELDAFF: Shield functionality in the nebula in %.

HS-OBJECTS

Everything in HSpace is modeled as an object, thus making it an object-oriented system. Everything from a ship, to a planet, to the smallest particle of bug goo hitting the foreward shield of your vessel is modeled as an object in space. Each object has its own behavior and attributes, though many objects may be very similar to other objects (e.g. stars and planets).

Each object in space must belong to a universe, and it has one of two states: active or inactive. Active objects are those that can be seen (generally speaking) in space. They are not docked in another ship. They are not landed on the surface of a planet. They are, essentially, actively in the space that makes up the universe the object belongs to.

Each object in the HSpace must also have an object in the game. For example, a planet called Planet X must have an actual "thing" in the game called Planet X. The object creation command will require that you specify the object # to identify that HSpace object.

Some objects, like ships, can possess engineering systems and engage in combat, while other objects, such as wormholes, serve entirely different purposes. For this reason, each object must be "setup" differently than other objects. When you create a new object, you will most likely have to initialize that object's attributes, just like assembling a new car. You must paint the car, install the radio, etc. Just as you may set an attribute on an object, you can also retrieve attributes from objects. Each object provides a variety of attributes that can, then, be set and retrieved, though these to sets of attributes may not be identical.

See also: @SPACE-ADDOBJECT, @SPACE-ACTIVATE, HS-TYPES

HS-OBJECT-ATTRS

Probably one of the most enjoyable features of HSpace is that every object can be manipulated and modified through its attributes. From changing the name of the object to changing the size, most aspects of an object are available to change. Each type of HSpace object, be it a ship, a nebula, or a missile, possesses attributes that you can change and retrieve. Because objects are often derived from "base objects," many objects will share similar attributes with other objects. All objects share at least the base, NOTYPE object attributes. These are as follows:

- NAME: Name of the object as viewed in space.
- SIZE: Size of the object from 1 .. n.
- X: X coordinate of the object in space.
- Y: Y coordinate of the object in space.
- Z: Z coordinate of the object in space.
- UID: Universe ID of the universe the object belongs to.

Beyond these attributes, the various types of objects may provide a plethora of additional attributes for you to modify and manipulate. Some objects may provide many attributes, while others may provide few.

For a list of attributes that are settable and retrievable for each object, refer to each object's attributes section:

HS-<OBJECT>-ATTRS

See also: HS-OBJECTS, HS-TYPES

HS-QUICKSTART

Looking for a quick start for your day? Here are some quick start answers to HSpace admin questions to get you moving along quickly.

Q: I've got HSpace installed, and I've rebooted my game. Here I am, what do I do?

A: Type @space. You'll see all of the current HSpace information presented to you. You'll have no classes, no universes, no ships, no planets, and some default weapons that HSpace offers. Your first step is to create a universe, so checkout the 'help hs-universes' help file.

Q: What are the HSpace commands I'll want to know first?

A: There's one command you'll use, and she's called @space. Refer to 'help @space' for the usage of this command and all of the wonderful options it provides.

Q: Where can I set all of the configuration options I want for HSpace?

A: HSpace comes with an hspace.cnf file that you can use to configure HSpace.

Q: How do consoles interact with HSpace?

A: Checkout the 'help hs-consoles' help file, but here's your answer. Consoles use the @nav, @eng, and @console commands to tell HSpace what they (or their users) want to do.

Q: Is HSpace hard to administrate? Am I going to spend my life trying to figure out how to do everything and get it all setup?

A: We hope not! Just keep a tight grip on that @space command and follow the many examples in the help files, and you will fairly easily setup your ships, planets, and other HSpace goodies.

HS-SHIELDS

HSpace supports two types of shields on vessels: Deflector(0) and Absorption(1).Ships that have shields must possess pairs of shields. That is, they must either have no (0) shields or 2 or 4 shields. HSpace does not currently support odd numbers of shields nor greater than 4 shields.

Absorption shields were used in versions of HSpace less than 4.0, and they work much like the skin on your body. The only difference is that you can't take your skin off, but let us pretend you can. When power is initially supplied to the absorption shield, the shield generator slowly begins to form a protective layer, the absorption shield. While the regeneration process may be slow, the shield can be lowered and raised subsequently without losing the charge of the shield. That is to say that the skin can be taken off and put back on without losing the strength of the skin. When an absorption shield is hit by an object or blast, it weakens. The shield regenerators will slowly begin to regenerate the shield. The shield loses effectiveness when its strength reduces to 0, at which point any objects or blasts pierce the shield to the underlying hull.

Deflector shields work by producing a semi-permeable shielding layer around the ship. They can be raised and lowered quickly and never need to recharge. They work by deflecting all incoming objects and blasts up to a given strength, which is their deflection strength. For example, a small deflector shield may be able to deflect the normal space debris that the ship encounters, but it may not be able to fully deflect a laser blast from another ship. The deflector shield will deflect all damage up to its strength value. Additional damage pierces the shield to the underlying hull. If the shield regenerators are damaged, the strength of the deflector shield is weakened.

HS-SHIP-ATTRS

Ship objects in HSpace provide the following attributes that are settable and retrievable.

Settable attributes:

- IDENT: Identification string for the ship.
- DESTROYED: 1 if the ship is destroyed, 0 otherwise.
- BOARDING CODE: Boarding code for the ship.
- CAN DROP: 1 if the ship can land, 0 otherwise. This attribute overrides the setting in the ship's class.
- DROPLOC: Dbref of the landing pad where the ship is located.
- DOCKLOC: Dbref of the docking bay where the ship is located.
- HULL: Current hull points for the ship.
- SPACEDOCK: All vessels regardless of size may dock here.

Gettable attributes (All of the above, plus):

- XYHEADING: The current XY angle heading of the ship.
- ZHEADING: The current Z angle heading of the ship.
- DXYHEADING: The desired XY angle heading of the ship.
- DZHEADING: The desired Z angle heading of the ship.
- CONSOLES: A list of console dbrefs on the ship.
- SROOMS: A list of registered rooms on the ship.
- LANDINGLOCS: A list of landing locations on the ship.

HS-PLANET-ATTRS

The following attributes are settable and retrievable for planet objects.

Gettable attributes:

- LANDINGLOCS: A list of landing locations (rooms) on that planet.

HS-SYSTEMS

Engineering systems in HSpace are very conveniently modeled to provide the highest level of flexibility. Objects, such as ships, can often be very different from one another, and thus there is no way to predict which engineering systems will be present on a given space object. For all of our examples, we will refer to ships, since they are most likely to possess systems.

What is an engineering system? An engineering system is really any "system" on the ship that helps it function. This may be a computer, a sensor array, a fuel management system, shields, etc. The list is nearly endless, though fortunately it's not! Systems can be given power, or they may provide power, such as with a ship's reactor. Systems may be stressed, and how easily they stress depends on how tolerant they are. Each engineering system at least possesses the following attributes:

- OPTIMAL POWER
- TOLERANCE
- STRESS
- DAMAGE LEVEL
- CURRENT POWER
- NAME
- VISIBLE

"Visible" systems are those systems that you can interact with -- you can transfer power to and from it. Not all systems are visible, such as with an internal fuel management system. It manages itself and doesn't let you touch it.

A ship may possess from none to many engineering systems, depending on how functional that ship is. For example, it may have jump drives, allowing it to travel at faster than light (FTL) speeds. It may, or may not, have engines, allowing it to move at sub light speeds. Many systems are included in HSpace that can be added to some HSpace objects, such as ships.

See also: @SPACE-ADDSYSCLASS, @SPACE-SYSSETCLASS, @SPACE-SYSSET, HS-SYSTEM-TYPES HS-SYSTEM-ATTRS

HS-SYSTEM-ATTRS

While each of the engineering systems available is derived from the generic engineering system object, each system may have additional attributes that can be set beyond the standard tolerance, stress, damage, etc. All systems have the following attributes:

- OPTIMAL POWER
- TOLERANCE
- STRESS
- DAMAGE LEVEL
- CURRENT POWER
- NAME
- VISIBLE

Additionally the following systems provide additional attributes as indicated:

Sensor Array:

- SENSOR RATING

Reactor:

- MAX OUTPUT
- DESIRED OUTPUT
- CURRENT OUTPUT

Maneuvering Thrusters:
- TURNING RATE

All Shields:
- REGEN RATE
- SHIELD TYPE
- MAX STRENGTH
- STRENGTH

*Refer to HS-SHIELDS for more information.

Engines:
- MAX VELOCITY
- ACCELERATION
- DESIRED SPEED
- CURRENT SPEED
- CAN AFTERBURN
- AFTERBURNING
- EFFICIENCY

Jump Drive:
- ENGAGED
- MIN SPEED
- EFFICIENCY

Fuel System:
- BURNABLE FUEL
- REACTABLE FUEL
- MAX REACTABLE FUEL
- MAX BURNABLE FUEL

Cloaking Device:
- EFFICIENCY
- ENGAGED

Tachyon Sensors:
- EFFICIENCY

Comm. Array:
- MAX RANGE

Damage Control:
- NUMCREWS

Comm. Jammer:
- RANGE

Tractor Beam:
- STRENGTH
- MODE (See HS-TRACTOR)

Fictional System: All standard attributes.

HS-SYSTEM-TYPES

HSpace currently supports the following types of engineering systems. Most are recommended for the common space vessel.

- Life Support: Supplies life support to people on the ship.
- Engines : Moves the ship (possibly afterburn)
- Jump Drive: Allows the ship to travel in hyperspace
- Fuel System : Provides fuel storage and allocation for the ship.
- Reactor : Provides power for the ship's engineering systems.
- Fore Shield : Protects the front of the ship.
- Aft shield: Protects the aft of the ship.

- Starboard Shield: Protects the starboard side of the ship.
- Port Shield : Protects the port side of the ship.
- Maneuv. Thrusters: Allows the ship to turn/maneuver.
- Sensor Array: Provides the ability to sensor sweep.
- Internal Computer: Provides power to consoles and ship logic.
- Cloaking Device: Provides the ability to hide ship from sensors.
- Tachyon Sensors: Provides the ability to counter the effect of cloak.
- Damage Control: Provides the ability to repair systems in flight.
- Comm. Jammer: Provides the ability to block comms within its range.
- Tractor Beam: Provides the ability to tractor/repulse/hold other ships.
- Fictional Sys : Allows an admin to add a system and give it any name he/she likes and set optimal power, useful for soft-coding extra effects, no hardcode effects.

It is important to note that the absence of port and starboard shields leaves the fore and aft shields to protect the entire ship. Thus, the starboard and port shields are not required to fully protect the ship, though they may be desired for added flexibility.

System names can be abbreviated for any @space commands.

See Also: HS-SYSTEM-ATTRS

HS-TERRITORIES

What is a territory? A territory is some piece of space, defined by specified boundaries.

HSpace models space territories through one of two ways. These are radial and cubic territories. A radial territory defines a center of the territory and a radius to which the boundaries extend from that center. A cubic territory defines a minimum and maximum set of 3D coordinates (6 coordinates in all) that define a cube in space.

When a ship travels, HSpace looks through the territories in the game to determine which territory the ship is in. If the ship enters or leaves a territory, a message is displayed to the console users on the ship. Further, HSpace triggers an attribute on the game object representing the territory, which can be useful for softcoding border alerts, welcome messages, etc.

The ENTER and LEAVE (@enter, @leave) messages are displayed to console users on the ship when entering or leaving a territory, and the AENTER and ALEAVE attributes are likewise triggered.

See Also: TERRITORY-SETUP, TERRITORY-ATTRS

TERRITORY-ATTRS

The following territories have the following attributes that can be set or retrieved.

Radial:
- X: The central X coordinate.
- Y: The central Y coordinate.
- Z: The central Z coordinate.
- RADIUS: The radius of the territory boundaries.
- UID: The universe ID of the territory.

CUBIC:
- MINX: Minimum X coordinate of the boundaries.

- MINY: Minimum Y coordinate of the boundaries.
- MINZ: Minimum Z coordinate of the boundaries.
- MAXX: Maximum X coordinate of the boundaries.
- MAXY: Maximum Y coordinate of the boundaries.
- MAXZ: Maximum Z coordinate of the boundaries.
- UID : The universe ID of the territory.

See Also: @SPACE-SET, HS_GET_ATTR(), HS_SET_ATTR()

TERRITORY-SETUP

Setting up a territory is much like setting up any other object in HSpace. You first need to create a game object to represent your territory. Perform the following steps:

1) @create My Territory
2) @space/addterritory My Territory=0 (A radial .. 1 for cubic).
3) @space/set My Territory/uid=0 (or a valid universe you have)
4) @space/set My Territory/cx=0
5) @space/set My Territory/cy=0
6) @space/set My Territory/cz=0
7) @space/set My Territory/radius=1000

There you go! You now have a radial territory with its center at 0,0,0 and a radius of 1000 units. To set the entering and leaving messages, just set the ENTER and LEAVE attributes on the object to some messages. If you want, you can set the AENTER and ALEAVE attributes with some code that gets run when a ship enters or leaves.

See Also: HS-TERRITORIES, TERRITORY-ATTRS

HS-TRACTOR

Tractor beams are systems that have 3 functions: Tractor ships towards you, Repulse ships away from you, and Hold ships at the same relative position to you. A tractor beam system has a certain strength, this together with the target's mass decides the range the beam can lock at and the speed it can tag it along or tractor/repulse it towards itself, once at 0.00000 hm it stops tractoring…of course. Ships can also be tractor docked by carriers or other ships with landing bays, provided they have enough space. This can be done from within 2 hm.

See: TMODE, TDOCK, TLOCK

HS-TYPES

Many types of HSpace objects can exist in the game. The following types are currently supported by HSpace code:

Type	Description
0	No Type (any object)
1	Ship
2	Missile
3	Planet
4	Wormhole
5	Blackhole
6	Nebula
7	Asteroid Belt

- LandingLoc

Other: Jump Gate - See: HS-GATES

For any commands that require an object type, the above types can be used. The "No Type," or type 0, can often be specified to represent all objects. Each object has its own attributes that can be set and retrieved. Because all HSpace objects are derived from a single, NOTYPE base object, they all share some similar attributes that can be set and retrieved. Some objects, such as celestial objects, will share similar celestial attributes that can be set and retrieved.

See also: HS-OBJECT-ATTRS

HS-UNIVERSES

The universe is the spice. The spice is the universe…or something like that.

A universe is just that…a universe. HSpace objects exist within universes, so they have to exist before any objects can exist. Whether a ship is flying through space or sitting on the surface of a planet, it belongs to a universe.

Each universe in HSpace relates to exactly one room in the game, created for the sole purpose of having a universe in the game. For each universe that exists, there must exist a room for that universe. This is needed for when ships are actively flying through space. It is also useful for looking at the room and the contents of that room to see what objects are actively in the universe.

You may have many universes, or you may have one. Often it is helpful to have at least two universes, one for real ships flying through space, and one for simulator combat that shouldn't interfere with the real universe(s).

See also: @SPACE-NEWUNIVERSE, @SPACE-DELUNIVERSE

HS-WORMHOLE-ATTRS

Wormholes are 2 black holes with immense gravity at each other folding the fabric of time and space, creating a passage from one point in space to the other. They have stability, in percentage, which represents the chance of successfully gating the wormhole, failure results in DESTRUCTION.

- BASESTABILITY: The percentage around which the stability of the wormhole fluctuates.
- STABILITY: The current stability of the wormhole
- FLUCTUATION: The margin within which the stability fluctuates.
- DESTX: The X coordinate of the destination of the wormhole.
- DESTY: The Y coordinate of the destination of the wormhole.
- DESTZ: The Z coordinate of the destination of the wormhole.
- DESTUID: The UID of the destination of the wormhole.

HS-WEAPONS

Weapons in HSpace provide the essential means for inter-vessel combat, and thus may vary in nature. While the HSpace software will provide most weapons, it is entirely possible to create your own, softcoded weapons using the provided HSpace interface functions. Querying and damaging systems can easily be performed, making softcoded weapons possible. This help file will clearly only discuss weapons provided with the HSpace software.

All weapons are stored in the weapondb located in your game/space/ directory. At present it is necessary to hand edit this database to add weapons. This will not be necessary in the final release of HSpace 4.0.The following types of weapons are currently supported:

Lasers:

Energy based weapons in the form of a gun. These weapons fire blasts of energy at the target and may or may not be strong enough to break through the target's shields.

Missiles:

Also known as torpedoes, these are large projectile weapons that must be loaded onto a hard point and fired at the target. Missiles are limited in supply. They also possess a limited "turning rate," which means they must steer toward a target. This also makes it possible to evade less maneuverable missiles.

See Also: HS-WEAPON-ATTRS, HS-LASER-ATTRS, HS-MISSILE-ATTRS

@SPACE Commands

Usage: @space/<switch> [<parameters>]

The almighty @space command controls just about every administrative aspect of the system. There is no other command in the software that allows so much. The command has the following switches:

ACTIVATE	ADDCONSOLE	ADDLANDINGLOC	ADDOBJECT
ADDSROOM	ADDSYS	ADDSYSCLASS	ADDWEAPON
DELCLASS	DELCONSOLE	DELSROOM	DELUNIVERSE
DELSYS	DELSYSCLASS	DUMPCLASS	DUMPWEAPON
HALT	LIST	SETMISSILE	MOTHBALL
NEWCLASS	NEWUNIVERSE	REPAIR	SET
SETCLASS	SYSINFOCLASS	SYSINFO	SYSSET
SYSSETCLASS	START	ADDTERRITORY	CLONE
NEWWEAPON	SETWEAPON	DELWEAPON	ADDHATCH
DELHATCH	ADDMESSAGE	DELMESSAGE	DELLANDINGLOC

For help on each command switch, refer to the help files with the following syntax 'help @space-<switch>'.

@SPACE-ACTIVATE

Usage: @space/activate <thing>=<class>

Activates <thing> as a ship with the given <class> identifier.

Ex.@space/activate Test Ship=0

The specified class must have already been created. The <thing> is a MUSH object that will represent the ship. In HSpace 3.x this was referred to as the ship object, and the navigation console was the heart of the ship. In HSpace 4.0 and later the shipobj is the heart of the ship, and the navigation console is like any other console on the ship.

The <thing> will be placed in rooms as the ship moves from active space to docking and landing locations. When docked or landed, <thing> will be placed in that room where the ship has landed. Players can use the board command with the object to board the ship.

Once the object is activated as a ship, consoles can be added to it and attributes can be set for it! DO NOT use the <thing> as a console. Because HSpace automatically moves it around the game, it will normally not be available for use as a console.

See also: HS-SHIP-ATTRS, @SPACE-MOTHBALL

@SPACE-ADDCONSOLE

Usage: @space/addconsole <ship obj>=<console>

Adds a new HSpace console, specified by the name or dbref of <console>, to the specified <ship obj>, which represents an HSpace ship in the game.

Consoles can use any of the HSpace commands (including @space if the console is set with the WIZARD flag) to perform a variety of functions.

See also: HS-CONSOLES, @SPACE-DELCONSOLE

@SPACE-ADDHATCH

Usage: @space/addhatch <ship obj>=<exit>

Specify <exit> as a new hatch for <ship obj>.

See Also: @SPACE-DELHATCH

@SPACE-ADDLANDINGLOC

Usage: @space/addlandingloc <object>=<room>

Adds a new landing location (a bay or drop pad), represented by <room> to the given HSpace <object>, which may be a ship or]a planet.

Like all HSpace objects, landing locations have their own attributes that can be set and retrieved. Ships and planets can support up to 15 landing locations at one time.

See also: HS-LANDINGLOC-ATTRS

@SPACE-ADDMESSAGE

Usage: @space/addmessage <console>=<message type>

This sets <console> to receive messages of type <type>, <type> being the number of the desires HSpace Message Type. All consoles can always hear general messages thus this is not a valid message type. Also consoles with nothing set can receive all messages, thus acting like all consoles did before HSpace v4.12.

See also: HS-MESS-TYPES @SPACE-DELMESSAGE

@SPACE-ADDOBJECT

Usage: @space/addobject <thing>=<type>

Adds a new HSpace object to HSpace of the given <type>. The <thing> parameter must be a MUSH object that was created with the @create command. Do not destroy this MUSH object while the HSpace object still exists. You should destroy the HSpace object first with @space/mothball and then @nuke the MUSH object.

Ex.@space/addobject Wormhole Alpha=4

Do not use this command for ships.

See also: HS-TYPES, @SPACE-ACTIVATE, and @SPACE-MOTHBALL

@SPACE-ADDSROOM

Usage: @space/addsroom <object>=<room>

Adds a "sroom" to the given HSpace <object>, which is typically a ship. Srooms stand for "ship rooms" are nothing more than the rooms the make up a vessel. By registering your ship rooms with the given ship

<object>, HSpace will be able to determine which game rooms belong to your ship. Thus, when the ship is destroyed, players in the rooms can be killed. Messages are also sent to all rooms on the ships sometimes, so any non- registered rooms won't receive these messages.

See also: @SPACE-DELSROOM

@SPACE-ADDSYS

Usage: @space/addsys <object>=<system>

Adds the specified <system> to the specified ship. The <system> parameter must be the name of a valid ship system.

Ex: @space/addsys #50=int

Notice that the name "Internal Computer" can be abbreviated as simply "int." HSpace uses the first system name matched by the specified <system> parameter. Thus, int, in, inter, internal computer all work for the Internal Computer name.

@SPACE-ADDSYSCLASS

Usage: @space/addsysclass <class>=<system>

Adds the specified <system> to the specified <class> ID. The <system> parameter must be the name of a valid ship system.

Ex: @space/addsysclass 0=int

Notice that the name "Internal Computer" can be abbreviated as simply "int." HSpace uses the first system name matched by the specified <system> parameter. Thus, int, in, inter, internal computer all work for the Internal Computer name.

After a system is added to a class, ships of that class will not reflect the changes after setting the system up and adding system to all ships of this class they will have right settings. This is because since v4.02 systems are saved for each ship specifically to provide the ability to edit each ship separately.

@SPACE-ADDTERRITORY

Usage: @space/addterritory <object>=<type>

Adds a new territory to space, represented by the MUSH object <object>.Type <type> must be a valid territory type, which is 0 for radial and 1 for cubic.

See Also: HS_TERRITORIES, TERRITORY-ATTRS, TERRITORY-SETUP

@SPACE-ADDWEAPON

Usage: @space/addweapon <console>=<weapon ID>

Adds a weapon of type <weapon ID> to the specified HSpace console. Only consoles can possess weapons, and all consoles are "things" in the game. A console can possess any type of HSpace weapon.

@SPACE-CLONE

Usage: @space/clone <ship>

Clones the ship represented by the object <ship>, including rooms, exits, attributes on rooms and exits, and consoles. The command returns the dbref of the new ship object and places the new ship in the universe and location of the cloned ship.

@SPACE-DELCLASS

Usage: @space/delclass <class ID>

Deletes the specified class of type <class ID> from the game. This can only be performed after all ships of the given class are changed to a new class or deleted from the game.

See also: @SPACE-NEWCLASS

@SPACE-DELHATCH

Usage: @space/delatch <ship object>=<exit>

Removes <exit> as a boarding hatch for <ship object>.

See Also: @SPACE-ADDHATCH

@SPACE-DELLANDINGLOC

Usage: @space/dellandingloc <object>=<room>

Removes a landing location (a room) that was previously added to the specified <object>.

@SPACE-DELMESSAGE

Usage: @space/delmessage <console>=<message type>

This sets <console> to no longer receive messages of type <type>, <type> being the number of the desires HSpace Message Type. All consoles can always hear general messages thus this is not a valid message type. Also consoles with nothing set can receive all messages, thus acting like all consoles did before HSpace v4.12.

See also: HS-MESS-TYPES @SPACE-ADDMESSAGE

@SPACE-DELSROOM

Usage: @space/delsroom <object>=<room #>

Deletes the given <room #> from the list of srooms for the specified <object>, typically a ship object.

See also: @SPACE-ADDSROOM

@SPACE-DELSYS

Usage: @space/delsys <object>=<system>

Removes <system> from the specified ship. The <system> parameter must be the name of a valid ship system.

@SPACE-DELSYSCLASS

Usage: @space/delsysclass <class #>=<system>

Removes <system> from the specified <class> ID. The <system> parameter must be the name of a valid ship system.

The changes will only apply to NEW ships with this class old ships will have to have the system @space/delsyssed from them or attributes set on their own systems since attributes derived from class are deleted.

@SPACE-DELUNIVERSE

Usage: @space/deluniverse <#>

Deletes the universe specified by the supplied <#>. Universes must be completely empty before they are deleted.

See also: @SPACE-LIST, @SPACE-NEWUNIVERSE, and HS-UNIVERSES

@SPACE-DELWEAPON

Usage: @space/delweapon <console>=<weapon #>

Deletes the weapon in the slot indicated by <weapon #> from the console indicated by <console>.

Ex:

- @space/delweapon gunnery turret=1

@SPACE-DUMPCLASS

Usage: @space/dumpclass <class ID>

Prints the information for the specified <class ID>, including what the current attribute settings are for that class and what engineering systems the class possesses.

See also: @SPACE-ADDSYSCLASS, @SPACE-SETCLASS, and @SPACE-SYSSETCLASS

@SPACE-DUMPWEAPON

Usage: @space/dumpweapon <weapon ID>

Prints the information for the specified <weapon ID>, including what the current attribute settings are for that class and what engineering systems the class possesses.

- See also: @SPACE-NEWWEAPON, @SPACE-SETWEAPON

@SPACE-HALT

Usage: @space/halt

Halts the HSpace cycle, which moves ships and performs general cyclic things. This does not prevent players from using HSpace commands. It only prevents HSpace from cycling.

See also: @SPACE-START

@SPACE-LIST

Usage: @space/list <database>

Allows items in the specified <database> to be listed. The <database> argument may be one of the following:

Objects: @space/list objects/<uid>=<type>

Where <uid> is a universe ID and <type> is a type of HSpace object.

Weapons: @space/list weapons
Classes: @space/list classes
Destroyed: @space/list destroyed
Universes: @space/list universes

See also: HS-TYPES

@SPACE-MOTHBALL

Usage: @space/mothball <thing>

Destroys the HSpace object associated with the MUSH object <thing>. The MUSH object is not destroyed, only the HSpace object it represents. Once an HSpace object is mothballed, it is gone forever.

Ex.@space/mothball Test Ship

@SPACE-NEWCLASS

Usage: @space/newclass <name>

The @space/newclass command allows a new ship class to be added to the game. The <name> parameter specifies the name for the new class.

Once a new class is created, its attributes must be set, and engineering systems must be added to it.

For example:

1) @space/newclass Test Class 1(creates class 0)
2) @space/setclass 0/maxhull = 300
3) @space/addsysclass 0=Internal Computer
4) @space/sysinfoclass 0=Internal Computer
5) @space/syssetclass 0:intern/tolerance=1

See also: @SPACE-SETCLASS, @SPACE-SYSSETCLASS, @SPACE-ADDSYSCLASS, and @SPACE-DELCLASS

@SPACE-NEWUNIVERSE

Usage: @space/newuniverse <room #>

Creates a new universe, represented by the room with the specified <room #>.Each universe in HSpace must have a room in the game that corresponds to it. Do NOT delete this room once the universe is created.

Universes allow HSpace objects to exist because every HSpace object must belong to a universe.

See also: @SPACE-DELUNIVERSE, HS-UNIVERSES

@SPACE-NEWWEAPON

Usage: @space/newweapon <type>=<name>

Adds a new weapon to the list of available ship weapons. The <type> must be a valid weapon type, and the <name> is the initial name of the weapon. You can use @space/setweapon to set the attributes on the new weapon after creation.

Valid weapon types:
0 = Laser
1 = Missile

See Also: HS-WEAPONS, HS-WEAPON-ATTRS

@SPACE-REPAIR

Usage: @space/repair <object>

Completely repairs all parts of the specified <object>, which is any HSpace object (typically a ship) that may be damaged. Any engineering systems belonging to that object are also repaired.

@SPACE-SET

Usage: @space/set <object>/<attribute>=<value>

Allows the specified <attribute> on the given <object> to be set to <value>.All HSpace objects have attributes that can be set and retrieved. The @space/set command is a powerful tool for changing the behavior and appearance of any HSpace object. For a list of attributes for the various types of HSpace objects, refer to HS-OBJECT-ATTRS.

For ships, attributes in the class of that ship can often be "overridden" at the ship level to provide more individuality for the ship. For example, the ability to drop to a planet or not is an attribute settable at the class level and also at the ship level. If the attribute is set on the ship, it overrides any setting on the ship's class, allowing that ship to perform unlike most ships of its class. This remains in effect until the attribute is reset on the specific ship by specifying no <value> parameter for the given attribute.

Ex.@space/set Navigation Console/name=USS Excalibur

See also: @SPACE-SETCLASS, @SPACE-SYSSET

@SPACE-SETCLASS

Usage: @space/setclass <class ID>/<attribute>=<value>

Like @space/set, the @space/setclass sets the <value> of the specified <attribute> but for the given <class ID> instead of a given HSpace object. This is useful only for ship classes, which provide the foundation attributes for all ships in the game. Rather than setting attributes on each individual ship, setting the attribute on the class allows all ships of that class to use the value. The only exception is when the class attributes are "overridden" by using @space/set with the same attribute name on the ship object. The value is then set on the specific ship instead of for the class. Any changes to the same attribute on the class will no longer affect the ship.

See also: @SPACE-SET, @SPACE-SYSSETCLASS

@SPACE-SETMISSILE

Usage: @space/setmissile <object>/<weapon ID>=<value>

Manipulates the munitions storage on the specified <object>, which is usually a ship, to set the current storage value for the given <weapon ID> (a missile type) to the specified <value>.The <weapon ID> is a the ID of the weapon in the weapons database, which can be printed with the @space/list command.

If the type of weapon is not already found in the object's munitions storage (missile bay), then the type is added to the storage and the max value set.

If the value of the storage is greater than the current maximum storage value, the maximum value is raised to the value given.

See also: @SPACE-LIST, @SPACE-ADDWEAPON

@SPACE-SETWEAPON

Usage: @space/setweapon <id>/<attribute>=<value>

Sets an attribute value for the specified weapon <id>, which may be listed with the @space/list command. The <attribute> must be a valid attribute for that type of weapon, and <value> is the new value for that attribute.

See Also: HS-WEAPON-ATTRS, HS_WEAPON_ATTR()

@SPACE-START

Usage: @space/start

Starts the main HSpace cycle, which is responsible for all cyclic activity in HSpace (e.g. moving ships, moving missiles, sweeping for sensor contacts, etc.).Without this cycle, nothing in HSpace moves or gets updated.

See also: @SPACE-HALT

@SPACE-SYSINFO

Usage: @space/sysinfo <object>=<system name>

Prints the current settings for the specified <system name> on the given HSpace object -- usually a ship. Each engineering system has its own attributes that can be set and retrieved. All systems are also derived from a base, generic engineering system type. Thus, base engineering system information is printed along with specific information for that type of engineering system.

See also: HS-SYSTEMS

@SPACE-SYSINFOCLASS

Usage: @space/sysinfoclass <class ID>=<system name>

Prints the current settings for the specified <system name> on the class with the specified ship <class ID>. Each ship class contains a list of default systems for ships of that class. Further, these systems must provide default system values for the systems on those ships. The @space/sysinfoclass command allows system information for the classes to be retrieved.

The <system name> should be one of the valid engineering systems in HSpace.

See also: HS-SYSTEMS, @SPACE-ADDSYSCLASS, and @SPACE-SYSSETCLASS

@SPACE-SYSSET

Usage: @space/sysset <object>:<system>/<attribute>=<value>

The sysset command is similar to the @space/set command, except that it works at the system level for the ship. Systems are smaller pieces located on an HSpace object (typically a ship) and not part of the standard attributes of an HSpace object. Thus, it is necessary to specify the <object>, which is an HSpace object, the <system>, which is the name of a system on that object (e.g. engines), the <attribute>, which is a valid attribute name for that system, and the <value> for that attribute.

Any attributes set at the individual ship system level override those of the ship's class systems. For example, the acceleration rate of the engines of Class 0 may be 50.By setting the acceleration rate of the engines on Ship B, which is of Class 0, to 100, that ship will no longer refer to Class 0's engine acceleration rate for accelerating. This is called "overriding" the class information.

Ex.@space/sysset navigation console:engines/acceleration=100

See also: @SPACE-SYSSETCLASS, HS-SYSTEMS

@SPACE-SYSSETCLASS

Usage: @space/syssetclass <class>:<system>/<attribute>=<value>

Sets the value of a given attribute for the specified system on the specified ship class ID. After an engineering system is added to a ship class, its variables must be set. For example:

@space/syssetclass 0:reactor/max output=100

The above command sets the maximum reactor output to 100 MW for ship class 0.

See also: @SPACE-SYSSET

Installation Guide

What is HSpace?

If you are familiar with MUSH, then you probably have some idea of what HSpace is. If you're not familiar with MUSH and online text games (that's what a MUSH is), then you probably have no use for this package.

HSpace (Hemlock Space, from the Hemlock series of games) is a sort of "plug-in" to online text games, adding to the game a virtual space environment with ships that can navigate and engage in combat and a variety of other space-like things. Since most text-based game engines (e.g. PennMUSH or TinyMUSH) only provide a basis for creating an online text game, to create a space game you either need to develop your own space engine, choose not to have any space flight at all, or download and install a package like HSpace. You have obviously chosen the latter of the three. If you are still interested in using HSpace, then continue reading the sections to follow.

SUPPORTED GAME ENGINES

This distribution of HSpace (there are other mutants available) currently only works with PennMUSH. HSpace was built in such a way that it can be easily adapted to other MUSH engines similar to PennMUSH. However, it hasn't yet been adapted to anything than PennMUSH. Perhaps someday soon? There is a rumor that another version of HSpace exists for TinyMUSH, so perhaps you could ask around for that one.

PACKAGE CONTENTS

You have unpacked the HSpace package, which contains a single "hspace" directory with the HSpace source code and this text file that you are reading. There is a Makefile in the same directory, which you will use to build the HSpace library.

BUILDING HSPACE

First, you will need to unpack and compile your game engine, be it PennMUSH or whatever. It is important for you to compile your game without HSpace to be certain that you can even get that far. Build your game, start it up, and get it running well. If you install HSpace without first testing your game without it, you won't be able to distinguish normal game problems from problems that could be caused by the space system!

Ok, you know your game works, right? Shut down your game, and let's build HSpace.

To build HSpace, you need to do the following things:

For PennMUSH:

1. Move your hspace directory into your pennmush/src directory, so that your hspace directory should be at pennmush/src/hspace.

2. Edit your pennmush/config.h and find the line that reads "#define HASATTRIBUTE /**/" Put a /* in front of that line if it's not already there.

3. Go to the hspace directory, and type make.

If all goes well, it should compile. At the beginning, you may see some warnings that certain dependency files do not exist, and that is quite all right. You should see several lines like "Generating dependencies for ..." and "Compiling ..." If you see any errors, you have compilation problems that were not foreseen when this package was created. If that happens, go to the sectioned titled "WHERE TO GET HELP" in this guide.

When your package is done building, you will see a file called in the hspace directory named "libhspace.a." This is the fully compiled HSpace code that will need to be "plugged in" to your game engine. Go to the section titled "INSTALLING HSPACE FOR <X>," where <X> is your game type.

INSTALLATION HSPACE FOR PENNMUSH

You will need to modify one two files for PennMUSH to add HSpace: src/local.c and src/Makefile.

Make the following change to src/Makefile (in the src directory, not the top level directory!):

Change: LIBS=$(CLIBS) $(RLIBS) $(ILIBS)
To: LIBS=$(CLIBS) $(RLIBS) $(ILIBS) -Lhspace -lhspace -lstdc++

Note that this assumes you have your hspace package directory in the PennMUSH src directory! If this is not the case, then you should have -L<path to libhspace.a> (e.g. -L../spacestuff/hspace).

Make the following changes to src/local.c, but do not include the <--- and the text after it!:

```
local_startup()  <--- Find this
{
        hsInit(); <--- Add this
}

local_dump_database() <--- Find this
{
        hsDumpDatabases(); <--- Add this
}

local_shutdown() <--- Find this
{
        hsShutdown(); <--- Add this
}

local_timer() <--- Find this
{
        hsCycle(); <--- Add this
}
```

When you have made these changes, change to your top-level PennMUSH directory, and type 'make install', which should compile local.c again and link the libhspace.a file with your PennMUSH engine.

In the PennMUSH game directory, make a directory named "space," and copy the hspace/hspace.cnf file to this directory. Edit the hspace.cnf, and change things as you please.

Copy all files in the hspace/help_files directory to your pennmush/game/txt/hlp directory. From pennmush/game/txt type 'make' to rebuild the help files.

Restart your game, and you should have HSpace commands available. Type 'help hspace' for the HSpace help files.

WHERE TO GET HELP

HSpace has been in use for quite some time and has had time to evolve, but the package is free, so to some extent you must expect there could be some problems that take time to sort out. One of the main problems could be that the package doesn't compile for you. This would not be a surprise, and the developers will need to know why it failed. Another problem could be that these instructions were not clear enough, so the developers will need to know that as well. Or maybe you just have some burning questions on your mind and want to ask them. In any case, here's the address where you can contact the developers of this package, and this package only!

gepht@hspace.org

Please include the package version, found in hsversion.h, the operating system you are using for your game server, and a decent description of the problem or question.

Additional information can be garnered from http://www.hspace.org!

Configuration File

The configuration file, hspace.cnf, is used to modify how HSpace operates. Many of the configuration options listed below came from version 3.x of HSpace and will likely be used again in the future. Most of the options listed below will operate properly.

AFTERBURN_RATIO

Default=2

Afterburning allows ships to increase their speed by some factor while still at sub light speeds. In real life it is accomplished by dumping fuel into the engines, essentially turning the engines into big rockets. Engines in HSpace can be capable of afterburning. Set this value to the increase in speed you want engines to experience when afterburning. This value is multiplied by the maximum capable speed of the engines to determine afterburn speed. If the engines aren't capable of afterburning, this value is ignored.

AFTERBURN_FUEL_RATIO

Default=7

Set this to the ratio of fuel consumption increase that you want engines to experience while afterburning. If you set it to a value of 1, engines just consume fuel during afterburn like they do during non-afterburn. Anything greater than 1 multiplies fuel consumption during afterburn.

AFTERWORLD

Default=18

The AFTERWORLD. Set this to the dbref number of the room where players go when they die in space. You will most definitely need to change this.

ALLOW_STRAFING

Default=0
Set this to 0 if you do not want to allow pilots to strafe fuel from nearby planets. This is typically disabled if you want to use your own ship-refueling scenario. Leave it set to 1 if you want to use the HSpace strafing ability.

AUTOSTART

Default=1

If you want the space system to start cycling immediately upon reboot, set this to 1. Else, set this to 0.

AUTOZONE

Default=1

Set to 1 rooms @space/addsroom'ed are automatically zoned to the shipobject, if there is no Zone-Lock present they are zonelocked to the Space Wiz.

CYCLE_INTERVAL

Default=1

This is probably set fine, but you may want to change it if you are on a machine with low CPU availability. This setting is measured in seconds and is the time between each cycle of the space system. The higher

this number, the "slower" the space system is. Ships change coordinates only once per cycle, sensors are swept once per cycle, etc.

DAMAGE_REPAIR_TIME

Default=60

Set this to the amount of time a damage crew should spend working on a system before that system receives one level of repair. This value is in seconds (e.g. 60 = 1 minute).

DECAY_SHIPS

Default=0

By setting this variable to 1, HSpace will age and decay ships for you by a predefined aging algorithm. Attributes of wear and tear for each ship component are stored on the navigation console. If you won't be using any sort of ship component repair system, it is recommended that you keep this turned off by leaving it set to 0.

DETECTDIST

Default=500

The distance at which a ship with sensor rating 1 DETECTS a ship with size 1.

FIRE_WHILE_CLOAKED

Default=0

Set this to 1 if you want to allow firing weapons while cloaked. Else, set it to 0 to disallow this feature.

FORBID_PUPPETS

Default=0

Some games allow puppets to fly ships. Others do not. If you prefer to restrict use of any ship consoles to only players, set this variable to 1.

FUEL_RATIO

Default=200

This is the fuel ratio, which you can change to increase or decrease fuel used by a ship's reactor. The number here is the number of units of power a reactor can produce per unit of fuel per hour. For example, a value of 50 indicates a ratio of 1:50. That is, the reactor uses 1 unit of fuel each hour for each 50 units of power it currently outputs. If you've specified that the fuel system should model two fuels, this fuel is deducted from reactable fuels and not burnable.

IDENTDIST

Default=100

The distance at which a ship with sensor rating 1 IDENTIFIES a ship with size 1

JUMP_SPEED_MULTIPLIER

Default=5

Set this to the sub light to hyperspace speed ratio for when jump drives are engaged. For example, if you're traveling at 100 units/hour, and this value is set to 500, then you'll travel 50,000 units/hour in hyperspace.

LOG_COMMANDS

Default=0

Set this to 1 if you want all space commands to be logged to the log file. A setting of 0 will not log commands.

MAX_ACTIVE_SHIPS

Default=50

Active ships are those that are undocked and should be checked for things like traveling and other active-ship sorts of status checks. By decreasing this, you can speed your game up slightly. You may need to increase it if you have a lot of people traveling around in the universes. This will typically be just a fraction of MAX_SHIPS, because it's unusual to have every ship in the universe out running around. If you modify this number while the game is running, do NOT reload the configuration file. Shutdown and reboot.

MAX_BOARD_DIST

Default=5

Set this to the maximum distance two vessels can be from each other and still acquire a boarding link.

MAX_CARGO_DIST

Default=2

Set this to the maximum distance two ships may be apart from each other and still transfer cargo between the ships.

MAX_DOCK_DIST

Default=5

Set this as the maximum distance a vessel can be away from its target vessel and still be able to dock in that target vessel. This is measured in whole heptameters.

MAX_DROP_DIST

Default=20

Set this as the maximum distance a vessel can be away from a target planet and still commence landing procedures.

MAX_DOCK_SIZE

Default=4

Set this to the maximum size ship that can dock in another ship. This exists for realism. How realistic is it for a battleship to dock in a destroyer? The size is on a scale from 1 - 10 and is recorded for each ship in the ship classes' database. Size 4 is approximately a corvette, which is considered to be about the upper range of dockable vessels.

MAX_LAND_SPEED

Default=2000

Set this to the maximum speed you wish vessels to be traveling when they attempt to land on a planet. Thus, if they're going too fast they cannot begin to land.

MAX_PLANETS

Default=50

Specify the maximum number of planets you want to allow per universe here. Like MAX_SHIPS, try to keep this number lower to avoid eating up system memory. If you run out of room in your universes, increase this number and reboot your game. If you modify this number while the game is running, do NOT reload the configuration file. Shutdown and reboot.

MAX_SENSOR_RANGE

Default=9999

Set this to the maximum distance you want any ship sensors to see. This is useful for filtering out big objects (like stars) that would normally appear on sensors from far, far away.

MAX_SHIPS

Default=500

When your universes become too full of ships, you will probably want to increase this variable. Try to keep it lower to avoid unnecessarily eating up system memory. You must reboot your game after changing this number. If you modify this number while the game is running, do NOT reload the configuration file. Shutdown and reboot.

MAX_STRAFE_DIST

Default=500

This is the maximum distance a vessel may be from a planet to strafe fuel from its atmosphere. This has no effect if ALLOW_STRAFING is set to 0.

MAX_UNIVERSES

Default=15

By default, HSpace allows 15 universes to exist in the game. Increase or decrease as you desire, but do not decrease it below the number of universe you already have in the game. Otherwise, bad things are going to befall you.

MIN_TRAINING

Default=10

The minimum flight training is a handy way of restricting brand newbies from hopping in a space ship when they have no business being in real space. This is not the number of minutes, hours, or days that a player must spend in a simulator before being able to "man" the navigation of a real ship. Rather, it is an arbitrary level to which a player must attain by using various space commands in a simulator. Each command is worth a very small number of points. As the players use various commands in a simulator, they gain more points until this number is reached. If you do not wish to restrict players to gaining simulator time, set this to 0.

NOTIFY_SHIPSCAN

Default=1

When one ship scans another, HSpace can notify the target ship that they are being scanned. If you would like this, change this option to a value of 1. Otherwise, set it to 0 for no notification.

SECONDS_TO_DROP

Default=30

Set this to the total number of seconds you want to have between the time a ship begins drop-to-planet procedures and the time they touch the ground.\

SENSE_HYPERVESSELS

Default=1

Set this to 2 if you want vessels in hyperspace to identified fully on other vessels' sensors.
Set it to 1 if you want vessels in hyperspace to only be detected.
Set it to 0 if you don't want vessels in hyperspace to be seen at all on other vessels' sensors.
These settings work vice versa for vessels in hyperspace detecting objects not in hyperspace.

SPACE_WIZ

Default=1

Set this to the dbref number of your space administrator. The player who this dbref refers to must be set with the WIZARD flag unless the player is god (dbref 1).

TRAIN_INTERVAL

Default=5

This is the amount of time that must pass before a sim user's simtime may increase. Essentially, it sets training increase per time and prevents sim users from looping a lot of commands to increase their sim training. If you're not partaking of sim training and such, this can be ignored or set to 0.

UNIT_NAME

Default=hm

Sets the name of the unit of distance in your game. This is the abbreviation of the unit (e.g. km).

USE_TWO_FUELS

Default=1

Set this to 1 if you want fuel systems to present both reactable and burnable fuels for ship reactors, engines, etc. If this is set to 0, then only one fuel source (burnable) is presented and used by systems.

USE_COMM_OBJECTS

Default=1

If you are using commlinks in your game and the HSPACE_COMM flag, set this option value to 1. If you do not intend to use commlinks, then set it to 0. Enabling this option does consume extra processing time when messages are broadcast through space, so you don't want to enable it unless you're going to use it.

Message Configuration

AFTERBURN_DISENGAGE

Default=The ship shakes as the afterburners are disengaged.

This is the message shown to players in the ship rooms when the afterburners are disengaged.

AFTERBURN_ENGAGE

Default=The ship shudders as the afterburners are engaged.

This is the message shown to players in the ship rooms when the afterburners are engaged.

BEGIN_DESCENT

Default=The ship shudders as the drop rockets engage and the ship begins its descent.

This is the message shown to players inside the ship when the ship begins the drop toward a planet.

COMPUTER_ACTIVATING

Default=Terminals all around you power up as the computer systems are powered.

Message displayed when power is supplied to the main computer.

END_JUMP

Default=The ship reverberates as it drops out of hyperspace.

This message is shown to players inside the ship when it drops out of hyperspace.

ENGINES_CUT

Default=The incessant hum from the ship's engines suddenly fades away.

This message is shown to players in the ship when power to the engines is cut.

ENGINES_ACTIVATING

Default=You feel a sudden rumble as the engines activate.

This is the message that everyone sees when the ship's engines activate.

ENGINE_FORWARD

Default=You feel a sudden and powerful force as the main engines engage.

This message is shown to players in the ship when the ship's engines switch from reverse to forward momentum.

ENGINE_REVERSE

Default=You feel a sudden and powerful force as the reverse thrusters engage.

This message is shown to players in the ship when the ship' engines switch from forward to reverse momentum.

JUMPERS_CUT
Default=You hear a large powerdrop as the jump drives power down.

This message is shown to players in the ship when power to the jump drives is cut.

LANDING_MSG
Default=The ship sways and bumps as it makes contact with the ground surface.

This message is shown to players inside the ship when it lands on a celestial surface.

LIFE_CUT
Default=The air suddenly becomes stale as life support loses power.

This message is shown to players in the ship when power to life support is cut.

LIFT_OFF
Default=The ship sways and bumps as it lifts from the surface of the ground.

This is the message shown to players inside the ship when the ship lifts off from the ground.

SHIP_IS_JUMPING
Default=Ship is currently traveling in hyperspace.

This message is given to a console operator when an illegal action is attempted while the ship is in hyperspace.

SHIP_JUMPS
Default=The ship hums loudly as it enters into hyperspace.

This is the message shown to players inside the ship when the ship goes into jump mode.

LIFE_ACTIVATING
Default=The air turns fresh and a gentle hum can be heard as life support is activated.

Message displayed when power is supplied to the life support system.

SPEED_INCREASE
Default=You feel a sudden jerk as the ship speeds up.

This message is shown to players in the ship when the ship speeds up.

SPEED_DECREASE
Default=You feel the ship begin to slow.

This message is shown to players in the ship when the ship's speed decreases.

SPEED_HALT
Default=You feel a slight force as the ship glides to a halt ...

This is the message that players inside the ship receive when the ship stops.

ENGINES_OFFLINE

Default=Engines are currently offline.
Console operators receive this message when a movement action is attempted while the engines are offline.

REACTOR_ACTIVATING

Default=Lights around you flicker on as the main reactor powers up.

Sets the reactor activation message.

REACTOR_OFFLINE

Default=Main reactor is currently offline.

Console operators receive this message when an action is attempted while the main power system is offline on the ship.

SHIP_IS_DOCKED

Default=Ship is currently docked.

Console operators receive this message when an illegal action is attempted while the ship is docked.

SHIP_IS_DOCKING

Default=The ship is currently in docking procedures ...

Console operators receive this message when an illegal action is attempted while the ship is docking.

SHIP_IS_UNDOCKING

Default=The ship is currently undocking ...

Console operators receive this message when an illegal action is attempted while the ship is undocking.

THRUSTERS_ACTIVATING

Default=A loud roar is heard as the steering thrusters activate.

Message displayed when power is applied to the thrusters.

Database Locations

The following configurations probably shouldn't be changed unless you are sure of what you're doing.

CLASSDB

Default=space/classdb

OBJECTDB

Default=space/objectdb

PICTURE_DIR

Default=space/pics

TERRITORYDB

Default=space/territorydb

WEAPONDB

Default=space/weapondb

UNIVDB

Default=space/univdb