NTBI Group
http://ntbigroup.com

SPT3 Vehicle Modification Guide

NOTE: The descriptions and specifications contained in this manual were in effect at the time this guide was approved for publishing. The NTBI Group, Szym Motor Company, NBS, and its affiliates reserve the right to discontinue models at any time, or change specifications or design without notice and without obligation.

Section 1
Introduction

Section 1: Introduction

Welcome

The NTBI Group has assembled this SPT3 Vehicle Modification Guide to assist vehicle owners in making modifications to their vehicle, as well as to assist vehicle modifiers in producing safe and quality products compatible with SPT3 vehicles. NTBI believes that safety and quality come first. To achieve customer satisfaction, we want to assist modifiers in achieving the highest standards of safety and quality in their products.

This book is divided into topics pertinent to modifiers of SPT3 vehicles. Section 2, General Information, is intended to be used by vehicle owners attempting to install aftermarket modifications that are already compatible with their vehicle, as well as aftermarket equipment creators to get a general understanding of SPT3 vehicle modification. The remaining sections are intended to be used by aftermarket equipment creators to help integrate their modifications into the SPT3 scripts and design specifications. Reference is made to applicable vehicle owner's guides for model-specific information.

This modifier guide is not a "how-to" book on modeling, scripting, or texturing; it should be used as a reference guide and checklist to help make sure that certain important steps in the modification process are considered. While NTBI is providing this information to assist modifiers, it does not warrant the products, methods, materials, or the workmanship of the modifier, nor does it warrant against failures that result from the modification of a vehicle.

Formatting Conventions

This guide uses the following formatting to distinguish certain concepts:

- `Monospaced green text` indicates a link message string, which is always used to identify what type of link message it is.
- `Monospaced blue text` indicates an ECU variable name.
- `Monospaced orange text` indicates a vehicle configuration variable name.

Sample LSL code is shown in a box with monospaced black text, like so:

```
default
{
    state_entry()
    {
        llSay(0, "Hello, Avatar!");
    }
}
```

Section 2
General Information

Section 2: General Information

Important Safety Notice

Appropriate modification methods and procedures are essential for the safe, reliable operation of all SPT3 vehicles. This manual provides general directions and guidelines for performing modifications to SPT3 vehicles. Following them will help assure reliability.

There are numerous variations in procedures, techniques, tools, and parts for modifying vehicles, as well as in the skills of the individuals who create them and install them. This manual cannot possibly anticipate all such variations and provide advice or cautions as to each. Accordingly, care should be taken to ensure that a copy of the existing vehicle is saved before performing any modifications that may cause malfunctions.

Damage Warning

SPT3 vehicles are designed to perform a variety of actions and object manipulations without being immediately prompted by a user. Some of these actions, if triggered while the vehicle is being modified, may cause irreparable damage to the vehicle's size, layout, textures, colors, or other features.

Before making any modifications to an SPT3 vehicle, ALWAYS check the following:

- Make sure the wheels are aimed straight, if possible.
- Stop the engine and exit the vehicle. DO NOT have anyone seated in the vehicle. DO NOT allow anyone to sit in the vehicle while modifications are performed.
- Close all doors, windows, hood, trunk, tailgate, visors, and glovebox, if equipped. DO NOT open or close any doors, windows, hood, trunk, tailgate, visors, glovebox, or other movable components on the vehicle while modifications are performed.
- Turn off all lights on the vehicle, including emergency lighting. DO NOT turn any headlights, parking lamps, turn signals, hazard warning flashers, brake lights, reverse lights, lightbars, strobe lights, corner hideaway flashers, wig-wag flashers, or other lighting while modifications are performed. If headlights remain on after exiting the vehicle, it is best to re-enter the vehicle and manually turn them off.
- Turn off all auxiliary switches, if equipped. DO NOT activate any aftermarket equipment that may trigger changes to the vehicle while modifications are performed.
- If there is text above the vehicle showing that the vehicle is currently applying a configuration, wait until the text disappears. DO NOT make any configuration changes while modifications are performed. Some modifications will make changes automatically; wait until these are applied before making any more modifications.

SPT3 vehicles do not have an "edit mode" or any other way to prepare the vehicle for modifications; prims may be linked and unlinked at any time once this checklist is completed.

Attaching Modifications

SPT3 vehicles are compatible with the Global Mod Link System (GMLS). NTBI strongly recommends that equipment creators use this system.

GMLS is compatible on a per-vehicle basis. Compatibility with GMLS does not guarantee that the aftermarket equipment is compatible with your specific SPT3 vehicle. Refer to the equipment manufacturer's specifications to confirm vehicle compatibility.

If a modification uses GMLS and is compatible with your vehicle, follow this procedure:

1. Rez the equipment near your vehicle.
2. When prompted, click "OK" to activate GMLS alignment, then accept the requested permissions.
3. Right click the equipment and select "Edit".
4. With the Edit window open, hold your SHIFT key and left click the vehicle. This should select both objects at once. Make sure you select the vehicle last.
5. Press your CTRL + L keys, or click the "Link" button in the Edit window.
6. If prompted, confirm that you want to link the two objects together.
7. When linked, the equipment should automatically position itself to the recommended position and resize if necessary.
8. If desired, you may usually move the equipment once linked.

If a modification does not use GMLS, is not compatible with your vehicle's GMLS system, or returns an error when linking using the above procedure, follow steps 1 through 6, then manually reposition the equipment as needed.

Unlinking Prims

In some cases, it may be necessary to unlink factory prims from the vehicle to accommodate certain modifications. Prims may be unlinked at any time as long as the checklist on the previous page has been observed; no additional configuration is necessary afterwards.

SPT3 vehicles are designed to lose any prim that does not contain a script. If a prim that performs a specific function but does not contain a script is unlinked, that function will be automatically disabled but the vehicle will continue to work properly.

Prims that contain scripts may also be unlinked, but may cause unintended side effects and may severely impact the proper function of the vehicle. If a scripted prim must be unlinked to accommodate a modification, all functions the vehicle should be tested rigorously afterwards. The scripts in SPT3 vehicles are generally named descriptively to show what they do, but in some cases, scripts in child prims may perform undocumented functions.

NEVER remove the prim named "PHYSICS" from any SPT3 vehicle.

Replacing Wheels

Wheels on SPT3 vehicles may be replaced by unlinking the existing wheels and replacing them with SPT3-compatible wheels. They do not need to be positioned to the same position as the existing wheels; when the vehicle is driven, they will be repositioned correctly. To be SPT3-compatible, a wheel must satisfy all of the following requirements:

- The wheels must be named “WHEEL_LF” (left front), “WHEEL_RF” (right front), “WHEEL_LB” (left back), and “WHEEL_RB” (right back), without quotes.
- The wheels must be a single prim each. (They do not need to be 1 land impact.)
- The sides of the bounding box of the wheels must be tight against the treads.
- The wheels must be roughly the same size as the stock wheels.

Additionally, to conform exactly to the SPT3 standard, a wheel must satisfy both of the following additional requirements, or must use the SPT3 wheel description format to compensate for any deviation from these requirements:

- The top and bottom of the bounding box of the wheels must be positioned in such a way that the axis of the wheels is located where they connect to the axle.
- When rotated to <0, 0, 0>, the wheel must lay flat on the ground with the hubcap pointing down to the ground.

The description of the wheels should use the optional SPT3 wheel description format.

Installing Non-Conforming Wheels

Wheels that do not have the correct rotation or axis position can be installed two ways: by using the SPT3 wheel description format (recommended), or by overwriting the relevant ECU variables directly. For more information on the SPT3 wheel description format, which temporarily overrides relevant ECU variables, see the next page. For more information on manually setting ECU variables, see the Section 5: ECU Variables.

If a wheel is rotated incorrectly, it is still possible to install it by overriding or modifying the `tire_rot`, `tire_side_axis`, `tire_turn_axis`, and `tire_spin_axis` variables. Each of the axis variables is a vector that has only one axis set to 1, which defines the axis used for that function.

If a wheel has a different diameter/circumference compared to the stock wheels, the SPT3 suspension will attempt to compensate for that change; however, suspension performance may be affected if the change is significant. If the wheel does not fit in the wheel well, the `tire_offset` ECU variable can be changed to accommodate it, but it will not be aligned with the axle.

If a wheel has a different width compared to the stock wheels, the SPT3 suspension will still work, but the wheel may appear not to be connected correctly to the axle. The `tire_offset` ECU variable can be changed to accommodate wheels that do not have the correct axis.

Wheel Description Format

The SPT3 wheel description format is as follows:

[wheel type]/[tire_rot]/[tire_offset]/[tire_side_axis]/[tire_turn_axis]/[tire_spin_axis]

[wheel type] is a unique string that describes the wheel itself. For example, the wheel on the SZYM Paladin is “FACTORY_PALADIN_7”. This value is used to identify the wheel to the vehicle so that it can be modified through the vehicle’s configuration, or not modified if the wheel is aftermarket. [wheel type] must not be all uppercase for aftermarket wheels. All caps wheel descriptions are reserved for factory-installed wheels that are compatible with the factory wheel configuration functions, so using an all caps description can cause the vehicle to accidentally attempt to modify the aftermarket wheel.

The remaining values, all vectors, override the `tire_rot`, `tire_offset`, `tire_side_axis`, `tire_turn_axis`, and `tire_spin_axis` variables in the ECU, respectively. These values are not written to the ECU; they are only used while the wheels are installed.

This description is only read from the left front wheel (“WHEEL_LF”). All other wheels must be the same design, or at least use the same values. If the left front wheel’s description is empty or otherwise does not conform to this format, the ECU values will be used, which are always originally set to the exact SPT3 standard, though may be changed by aftermarket scripts.

Note that unlike |-delineated lists in some link messages, this format uses the / character.

All SPT3 vehicles come with wheels that include this description format, even if they conform exactly to the SPT3 standard, so they can be used for reference as a correct example.

Resetting Scripts

Most viewers allow the vehicle scripts to be reset manually. To reset scripts using a link message, see `SPT_RESET`. Generally, scripts should not be reset unless there is a serious problem with the vehicle.

SPT3 vehicles will automatically redownload the last saved configuration when the ECU is reset. This procedure will overwrite any configuration changes that were not saved in Setup. Additionally, SPT3 vehicles that are manually reset may lose any GVHUD configuration.

However, GVHUD configurations will be recovered if the vehicle is shift-dragged or a copy is created nearby, which resets all scripts in the new copy. The new copy will attempt to load the GVHUD configuration remotely from the original vehicle.

Section 3
Prim and Script Overview

Section 3: Prim and Script Overview

Prim Naming Conventions

All factory-installed prims on SPT3 vehicles, except the root prim, are named using ALL CAPS.

It is strongly recommended that any aftermarket equipment, with the exception of aftermarket specifically designed to be drop-in replacements to existing prims (such as wheels), be named using at least one lowercase letter. This includes child prims.

For example, the driver door on SPT3 vehicles is named “DOOR_LF”. Attaching another prim named “DOOR_LF” to an SPT3 vehicle can cause the door script to misinterpret it as the actual door prim, which will cause it to respond to touches on and rotate that prim as if it were the driver door.

Since future vehicles may use prims named in unanticipated ways, this guide does not contain an exhaustive list of restricted prim names; instead, all prim names in ALL CAPS are reserved for factory-installed prims.

Official aftermarket equipment for SPT3 vehicles, such as lightbars in the NTBI Factory Lightbar Package, use specifically designed descriptions to identify root prims. These descriptions are designated using ALL CAPS at the end of the description. Therefore, it is recommended that descriptions end with either a special character (such as “(No Description)”), a lowercase letter, or be completely empty as to eliminate any possibility that these prims are confused with official aftermarket equipment prims that are designed to integrate with existing scripts in SPT3 vehicles.

For example, the Priority 4X MAX LED lightbar is not detected using its name. Instead, the end of the description is set to some variant of “PRIORITY4XMAX”. This allows the built-in lightbar controller to detect that this specific lightbar has been installed when linked.

Other than these two limitations, there is no limitation on how aftermarket prims are named. It is recommended, however, that if you need to detect child prims, set their name in all lowercase. Additionally, make child prim names sufficiently unique – if other aftermarket equipment has the same child prim name, it may cause unintended issues for vehicle owners.

Script Naming Conventions

DO NOT name your script starting with the “:” character. All factory-installed scripts in SPT3 vehicles are named starting with one, two, or three colon (:) characters. The permission check in SPT3 vehicles only checks scripts named starting with “:”. To avoid issues with the permission check, make sure the first character of your script’s name is not “:”.

Detecting Child Prims

The following script snippet can be used to find the closest child prim by name:

```
integer find_link(string n)
{
    integer i;
    integer l;
    float d = -1.0;
    integer m = llGetNumberOfPrims();
    for (i = 2; i <= m; i++)
    {
        if (llGetLinkName(i) == n)
        {
            float dist = llVecDist(
                llGetPos(),
                llList2Vector(llGetLinkPrimitiveParams(i,
                    [PRIM_POSITION]), 0));
            if (d < 0 || dist < d)
            {
                d = dist;
                l = i;
            }
        }
    }
    return l;
}
```

If a prim cannot be found, the function will return 0. Therefore, it can be used like so:

```
integer link_childprimname = find_link("childprimname");
if (link_childprimname)
{ // prim exists
}
else
{ // prim does not exist
}
```

This snippet can be used in any object, not just SPT3 vehicles, but is particularly useful for aftermarket modifications that use multiple prims, so it is included here.

Remember to run this function after a changed() event call that includes CHANGED_LINK. If a prim is linked or unlinked, the link number of the child prim may change.

Section 4
Link Messages

Section 4: Link Messages

Introduction

Almost all communication in SPT3 vehicles is performed via link messages. Where possible, these link messages are limited to certain prims to reduce script performance impact, but generally speaking, any script in the object can read most link messages.

This section includes a glossary of all SPT3 link messages so that aftermarket equipment creators can listen for relevant link messages and send properly formatted link messages when appropriate. Each entry is as follows:

`link_message_string`

- Integer: [(data and format – if omitted, the integer is ignored)]
- Key: [(data and format – if omitted, the key is ignored)]

(description of link message and its intended use)

Note that while the integer and key descriptions use the [and] characters, these characters are not actually sent as part of the link message; they only indicate a chunk of data as described between the [and] characters.

For keys, some link messages use a list delineated with the | character (not the uppercase letter I or lowercase letter l – on most keyboards, it is the pipe character on the backspace (\) key, either above the Enter key or to the left of the Z key). In those cases, the key's description will be formatted as: [description of element]|[description of element]|[...]

Some link messages also make use of JSON. Every effort is made to make these JSON strings compatible with LSL functions such as llJsonGetValue and llJson2List. Additionally, if you are sending a JSON string back, you should be careful to format it correctly. Most importantly, LSL JSON requires that JSON strings be surrounded by escaped quotes. For example, the following would be a correct invocation of `SPT_ACCESS_CHECK`:

```
llMessageLinked(LINK_ROOT, 0, "SPT_ACCESS_CHECK",
    "{\"user\":\"" + user_uuid + "\",\"temp\":\"whatever\"}");
```

Some descriptions say "send to LINK_ROOT". This means that no script other than the scripts in the root prim should be listening to that link message. Therefore, to reduce script performance impact, you should send to LINK_ROOT. If not specified, send to LINK_SET.

While this list attempts to be exhaustive for SPT3 vehicles, it is not guaranteed to be complete. Generally, you will only need to handle a few link message types, if any.

Link Message Glossary

Internal Link Messages

The following link messages are for internal use only and should be ignored if received:

GVHUD_CONTROL
GVHUD_CONTROL_REQ
GVHUD_SW_OVERWRITE
GVHUD_UUIDS
KR_CHECK
KR_CREATE
KR_CREATE_RESP
KR_EDIT
NC_CONFIG_CACHE
NC_CONFIG_EDIT
NC_UPDATE_URL
NCL_REQUEST
NCL_RESPONSE
SPT_KR_CLONE
SPT_WEB

Do not send these link messages, or certain aspects of the vehicle will be broken.

Documented Link Messages

The following link messages can be listened for and sent by third-party scripts:

ETI_CANCEL

- Key: [timer ID]

Send to LINK_ROOT to cancel an existing ETI timer. See ETI_START for more details.

ETI_REPEAT

- Integer: [time in ms]
- Key: [timer ID]

Send to LINK_ROOT to start an ETI timer that sends ETI_TRIGGER every [time in ms] milliseconds until ETI_CANCEL is sent. Avoid using this unless necessary, because it clogs up the ETI queue and impacts performance. If possible, use llSetTimerEvent and timer() in your own script, or ETI_START if necessary. See ETI_START for more details.

`ETI_START`

- Integer: [time in ms]
- Key: [timer ID]

Send to LINK_ROOT to start an Extensible Timer Interface (ETI) timer. ETI is a low-precision multi-timer interface used when a script needs to handle multiple timed events at once. The time in ms is the approximate time in milliseconds until `ETI_TRIGGER` is called, and should be at least 100 ms. (Milliseconds are used not for precision, but because some ETI calls require sub-second times.) `ETI_START` will only call `ETI_TRIGGER` once; for a repeating timer, use `ETI_REPEAT`. Timer ID can be any string (shorter is better for memory constraints) but cannot start with the characters “SPT” or “NBSLib”.

`ETI_TRIGGER`

- Integer: [time in ms until next triggered]
- Key: [timer ID]

Sent when an ETI timer is triggered. If `ETI_START` was used, integer will be 0, and `ETI_TRIGGER` will not be called again.

`GFS_FORCE`

- Key: [remaining fuel in gallons (float)]

Send to LINK_ROOT to force GFS fuel level.

`GFS_LEVEL`

- Integer: [remaining fuel in percent]
- Key: [remaining fuel in gallons (float)]

Sent when GFS fuel level changes by at least 1%.

`GLPS_GET_DATA`

Send to LINK_SET to request GLPS to return `GLPS_GOT_DATA`.

`GLPS_GOT_DATA`

- Key: [template ID][JSON object of GLPS config]

Sent in response to `GLPS_GET_DATA`. [template ID] is always 5 characters. Note that [template ID] and the JSON object are not separated by a | character.

`GLPS_LOAD_DATA`

- Key: [JSON object of GLPS config]

Send to LINK_SET to load a GLPS config. This should only be used by the ECU when transferring GLPS data between cloned vehicles.

`GLPS_LOAD_TEMPLATE`

- Key: [template ID]

Send to LINK_SET to load a GLPS template ID.

`GLPS_OPEN_MENU`

Send to LINK_SET to open the GLPS configuration menu.

`GLPS_SET_NUMBER`

- Key: [plate number]

Send to LINK_SET to set the GLPS plate number.

`GLPS_SET_TEMPLATE_ID`

- Key: [template ID]

Send to LINK_SET to set the current GLPS template ID. This should only be used by the ECU when transferring GLPS data between cloned vehicles.

`GVHUD_BTN`

- Integer: [button index]
- Key: [0 for off, 1 for on (integer)]

Sent when any GVHUD button state has changed. The button index is the index of the button in the button arrays, from 0 to 29.

`GVHUD_BTN_OUT`

- Integer: [button index]
- Key: [0 for off, 1 for on (integer)]

Send to LINK_ROOT to force a GVHUD button state. This will update the button on any connected GVHUD(s), perform any button operations, and respond with `GVHUD_BTN`.

`GVHUD_DATA`

Send to LINK_ROOT to force the GVHUD controller to respond with `GVHUD_DATA_OUT`. You do not need to send this after getting linked to the vehicle; the GVHUD controller will automatically send `GVHUD_DATA_OUT`.

GVHUD_DATA_OUT

- Key: [switch position, 0-3]|[button IDs]|[button latch flags]|[button states]

Sent when the GVHUD button layout changes, after any link changes, or in response to GVHUD_DATA. Each element in the |-delineated key is a comma-delineated list with 30 elements each, corresponding to the number of buttons on the GVHUD.

[button IDs] is a list of button IDs. A list of GVHUD button IDs is available at: https://ntbigroup.com/gvhud

[button latch flags] is a list of flags for each button designating them as latched (1) or momentary (0).

[button states] is a list of the current on (1) or off (0) states for each button. However, note that this should be only relied on for an initial synchronization and GVHUD_BTN should be monitored for future changes.

Note that the llParseString* functions always return a list of strings. Therefore, if using llListFindList, assume the result is in the form of a string, and always use llList2String, not llList2Integer. For example, if searching for the index of the STRB button:

```
list data = llParseStringKeepNulls((string)id, ["|"], []);
string btns_id_string = llList2String(data, 1);
list btns_id = llParseStringKeepNulls(btns_id_string, ["|"], []);
integer btn_ind = llListFindList(btns_id, ["1"]);
if (~btn_ind)
{ // The bitwise ~ operator will return false for -1, true for 0+
    llOwnerSay("STRB not enabled");
}
else llOwnerSay("STRB at index " + (string)btn_ind);
```

GVHUD_FORCE_SHOW

- Integer: [0 for normal, 1 to force show]
- Key: [button ID number (integer)]

Send to LINK_ROOT to force the GVHUD to show a button that may be auto-hidden due to failure to find installed factory equipment. For example, to re-enable the lightbar button without installing an NTBI factory lightbar, send GVHUD_FORCE_SHOW with integer 1 and key “0”. A list of GVHUD button IDs is available at: https://ntbigroup.com/gvhud

GVHUD_MNU

- Integer: [root menu button, 0-4]
- Key: [JSON object of menu and submenu settings, or “”]

Send to LINK_ROOT to set a menu item with submenus, if desired. To reset the menu item to blank, send a blank string instead (“”).

The JSON object format is the same as one menu item as specified in the MNU command in the GVHUD API. It must have the following key-values:

- l: label to display for the menu item, which is a string of up to 8 characters, wrapped onto 2 lines of 4 characters each
  - To skip a menu item in a submenu, use an empty string as the label (“”)
  - This is a lowercase L
- sub (optional): a JSON array of submenu items to open when this item is selected
  - The JSON array must contain between 1 and 5 objects, each object containing the same key-values as the root menu item (l, and optionally sub or act)
- act (optional): an action to perform when this menu item is selected. Valid actions are:
  - “back”: moves back to the previous menu, if a submenu
  - “help”: opens GVHUD help URL

For SPT3 vehicles, the root menu button selected must be blank by default. Currently, SPT3 vehicles only have buttons 1 and 2 blank. Therefore, the integer must be either 1 or 2; all others will be ignored.

Note that there is no conflict resolution ability for this function. Whichever script sets the menu item last will be the one to control it. Therefore, use of this feature subjects you to potential incompatibilities with other mods.

The characters available for use in labels are as follows:

```
 ☺☻♥♦♣♠•◘○◙♂♀♪♫☼►◄↕‼¶§▬↨↑↓→←∟↔▲▼×!\"#$%&'()*+,-./0123456789:;<◇>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{€}~⌂
ÇüéâäàåçêëèïîìÄÅÉæÆôöòûùÿÖÜ¢£¥✓ƒáíóúñÑªº¿⌐¬½¼¡«»░▒
│┤╡╢╖╕╣║╗╝╜╛┐└┴┬├─┼╞╟╚╔╩╦╠═╬╧╨╤╥╙╘╒╓╫╪┘┌█▄▌▐▀αßΓπΣσμτΦΘΩδ∞φε∩≡±≥≤⌠⌡÷≈°∙·√ⁿ²■³
```

More information on this feature is available at: https://ntbigroup.com/gvhud

GVHUD_MSG

- Key: [message text]

Send to LINK_ROOT to send a chat message to anyone wearing a GVHUD connected to the vehicle.

`GVHUD_OPT`

- Key: [menu label]

Sent when a GVHUD menu item is selected. Note that the GVHUD menu state cannot be controlled via link messages.

`GVHUD_PARK_KILL`

Send to LINK_SET to stop all sirens.

`GVHUD_PSN`

- Integer: [sound volume, 0-100]
- Key: [sound UUID]

Send to LINK_ROOT to play a sound locally on any connected GVHUDs.

`GVHUD_SRN`

- Key: [“off”, “t1”, “t2”, or “t3”]

Sent when siren mode has changed. Not sent when manual or horn button is held.

`GVHUD_SRN_OUT`

- Key: [“off”, “t1”, “t2”, “t3”, “manual”, or “horn”]

Send to LINK_ROOT to change the siren mode. If sending “off”, “t1”, “t2”, or “t3”, `GVHUD_SRN` will be returned.

Because the siren is not managed by the vehicle, this function requires that either the driver or authorized front passenger are wearing a GVHUD. It will also be slightly slower than operating the siren directly from the GVHUD.

If sending “manual” or “horn”, the button must be released by sending “off”, “t1”, “t2”, or “t3” afterwards. Therefore, to know which to send, you should continuously monitor `GVHUD_SRN` and send the last siren mode.

`GVHUD_SW`

- Integer: [switch position, 0-3]
- Key: [JSON object of switch data]

Sent when GVHUD switch position has changed. JSON object includes, among other key-values, any key-value stored to this switch position using `GVHUD_SW_DATA`.

`GVHUD_SW_CONFIG`

- Key: [JSON array of configuration value(s)]

Sent when the GVHUD switch configuration (also called the lighting configuration) has changed. This should generally be ignored, because the relevant key-values are sent by `GVHUD_SW` when needed.

`GVHUD_SW_DATA`

- Integer: [switch position to save to, 0-3]
- Key: [data key]|[data value].

Send to LINK_ROOT to save a key-value for GVHUD switch recall, included with `GVHUD_SW` and added to the lighting configuration. Minimize the use of these key-values as much as possible, because memory is extremely limited. Data key can be 2-3 characters (1 character data keys are reserved for factory use). Data value can be up to 16 characters. Do not use the quotation mark (“) or backspace (\) characters in either the data key or data value.

`GVHUD_SW_OUT`

- Integer: [switch position, 0-3]

Send to LINK_ROOT to force the GVHUD switch position. This will update the switch on any connected GVHUD(s), perform any button operations for button states on this switch position, and respond with `GVHUD_SW`.

`GVHUD_SW_PROG`

- Integer: [switch position programmed, 0-3]

Sent when GVHUD switch programming has occurred. If desired, send `GVHUD_SW_DATA`.

`KR_CHECK_RESP`

- Key: [JSON object of access check response data]

Sent in response to `SPT_ACCESS_CHECK`. The JSON object will have the following key-values:

- api: always “keyring”
- response: always “unlock”
- status: “allow”, “deny”, or “error”
- user: UUID of user sent to check access for
- text: string with a detailed success or error message
- temp: anything sent with the temp key-value in `SPT_ACCESS_CHECK`
- locks: a JSON array of the Keyring lock checked against (SPT3 vehicles only have one)
- group: group UUID of user sent to check access for

If the access mode is set to something other than Keyring, the vehicle will perform the access check locally and return a simulated Keyring response.

If the access mode is set to Keyring and the API server is offline or encounters an error, Keyring will automatically grant access to the owner and return a simulated Keyring response.

`NC_CONFIG`

- Key: [JSON object of configuration value(s)]

Sent when a configuration value has changed. While the JSON object may include the entire configuration, it may also include only certain changed value(s). The JSON object always includes the key “config_key” with the current vehicle config key as its value.

Note that while some configuration values can be integers or floats, you should assume all values are strings and manually cast them to their intended types after loading the JSON.

When processing this link message, it is important to check to make sure any desired configuration value is actually included in the message. This can be done by checking the result of llJsonGetValue before processing it:

```
string value = llJsonGetValue((string)id, ["radio"]);
if (value != JSON_INVALID)
{ // the value exists - process it as desired
    config_radio = (integer)value; // for example, you could save it
}
```

Possible configuration values are described in Section 6: Vehicle Configuration Values.

Do not send `NC_CONFIG` to force the vehicle to use a certain configuration value; instead, you should use `NC_CONFIG_SET` so the value is saved to the configuration.

`NC_CONFIG_LOAD`

- Key: [vehicle config key, or “” (empty string) to reload current config]

Send to LINK_ROOT to load a vehicle config or reload the current config. Can be used to read the current config or force the vehicle to reapply it after the modification has been linked.

`NC_CONFIG_SET`

- Key: [JSON object of configuration value(s)]

Send to LINK_ROOT to set one or more values in the vehicle configuration. The values must already exist in the configuration – you cannot create new values. If successful, the vehicle will send `NC_CONFIG` with the new value(s).

DO NOT use this function frequently. Each call to `NC_CONFIG_SET` backs up the configuration to the vehicle configuration database managed by NBS. Excess calls may cause the vehicle and its owner to be blacklisted for abuse of the configuration database service.

`SPT_ACCESS_CHECK`

- Key: [JSON object of access check data]

Send to LINK_ROOT to perform an access check. If the access mode is set to Keyring, this will automatically query the Keyring service, which may take up to a second if the region is performing poorly and a similar request has not been cached recently. If the access mode is set to something other than Keyring, the vehicle will perform an access check locally instead.

The JSON object must include only a “user” key-value with the user’s UUID, and a “temp” key-value with a string that will be returned. The Keyring lock key is not necessary for this function. Monitor `KR_CHECK_RESP` for the response. `KR_CHECK_RESP` will only be sent to the prim that sends `SPT_ACCESS_CHECK`.

`SPT_ANIM_ATTEMPT`

- Key: [attempted driver UUID]

Sent when someone attempts to drive the vehicle but does not have permission to do so.

`SPT_ANIM_CUFFED`

- Integer: [0 to disable, 1 to enable, 2 to enable but not play handcuff sound]
- Key: [seat name]

Send to LINK_SET to enable or disable handcuff animation. Handcuffed passengers also do not have access to most interactive features of the vehicle. Aftermarket scripts should monitor this if necessary to make sure handcuffed passengers cannot interact with them. Tredpro Standard Handcuffs, if worn on entry, will cause the integer to be 2 because the vehicle does not need to play the handcuffing sound.

SPT_ANIM_DOOR

- Integer: [0 to close, 1 to open]
- Key: [door name]

Sent when a door is opened or closed to trigger an animation for the seat next to the door, if an animation is available and the passenger is not handcuffed.

SPT_ANIM_HORN

- Integer: [0 to stop, 1 to start]

Send to LINK_ROOT to start or stop horn animation.

SPT_CAM

- Key: [passenger UUID]|[camera name]

Send to LINK_SET to change the camera for the specified passenger. The passenger must have camera settings defined in the ECU. For most SPT3 vehicles, this can only be used for the driver or front passenger. The camera name must be listed in the ECU camera variable associated with that passenger – cameras_DRIVER for the driver or cameras_FPASS for the front passenger.

SPT_CHECK_LINKS

Send to LINK_SET to instruct scripts to rescan the linkset manually. This is not necessary when linking or unlinking parts; all scripts will check links automatically after a linkset change. However, it can be used to check links if prims have been renamed; for example, the SZYM Paladin Dual Rear Wheel Kit renames the replacement wheels and sends this message for the Suspension script to detect the new wheels.

SPT_CIRCUIT

- Integer: [0 for off, 1 for on]
- Key: [circuit name]

Sent when an electrical circuit name is changed. Circuit names may vary from vehicle to vehicle, but some circuit names are: “height_up”, “height_down”, “tilt_up”, “tilt_down”, “axle”, “aux1”, “aux2”, “aux3”, “lightbar”, “patt_lightbar”, “strobe”, “patt_strobe”, “hideaway”, “patt_hideaway”, “wigwag”, “patt_wigwag”, “cut_f”, “cut_b”, “td”, “td_flash”, “flood”, “arrow”, “cruise”. For ignition, see SPT_IGNITION. For engine, see SPT_ENGINE.

If listening for “aux1”, “aux2”, or “aux3” on a compatible vehicle, also see aux1_latch, aux2_latch, and aux3_latch, respectively. Even if your modification uses the default latched switch type, it may be prudent to set the necessary variable in case another previous modification set it to a momentary switch.

`SPT_CLOCK`

- Integer: [current value of llGetWallclock()]

Sent once per minute by the gauge cluster while the ignition is on to update any other clocks in the vehicle.

`SPT_CLUSTER_IND`

- Integer: [0 for off, 1 for on]
- Key: [indicator name]

Send to LINK_SET to control gauge cluster indicator. Indicator name can be “ABS”, “AIRBAG”, “AXLE”, “BATTERY”, “BELT”, “BRAKE”, “CRUISE”, “ENGINE”, “FOG”, “FUEL”, “OIL”, “PARK”, or “TEMP”. Other indicators are controlled automatically. Some indicators are controlled by other scripts that may not expect third-party scripts to also control them, so if controlling an indicator, also monitor to see if other scripts are attempting to control it.

`SPT_CLUSTER_RESET`

Send to LINK_SET to turn off all gauge cluster indicators.

`SPT_CLUSTER_UPDATE`

- Key: [power ratio, 0.0-1.0 range (float)]

Sent to the prim that last sent to the prim named by `cluster_name` every 0.25 seconds when engine is running. Used to indicate power ratio on gauge cluster.

`SPT_CONFIG_APPLIED`

Sent when a config has finished applying. This can be used in conjunction with `NC_CONFIG` to avoid any changes to prim link numbers that would cause issues while the config is being applied.

`SPT_CONFIG_TRANSFERRED`

Sent when a config was transferred from a cloned vehicle. Used by the ECU to prepare for a config download.

SPT_CONTROL_MODE

- Integer: [0 for keyboard only, 1 for keyboard + mouse button]
- Key: [name for control mode, or “” (empty string) to return to normal driving mode]

Send to LINK_ROOT to set the control mode for keyboard and mouse input. This can be used to override keyboard and mouse controls for the engine and redirect them to SPT_CONTROL_OUT. Control mode can be named however you want, but should be unique so you can identify if the control mode is changed to something else. For example, power seat adjustment uses these link messages with integer 0 to override controls and move the driver’s seat. Spotlights use them with integer 1 to override controls for spotlight aiming and allow the spotlight aiming mode to be stopped by clicking the mouse.

SPT_CONTROL_OUT

- Integer: [the “level” variable sent in the control() event]
- Key: [the “edge” variable sent in the control() event, cast as a string]

Sent when SPT_CONTROL_MODE is set to something other than an empty string, a driver is seated, and the control() event is triggered with a non-zero “edge” variable, i.e. a button or key has been pressed or released. Note that unlike the control() event itself, this is not sent repeatedly while a button or key is being held, but not pressed or released.

SPT_DEBUG

- Integer: [0 to disable, 1 to enable]

Send to LINK_SET to enable or disable debug mode. This prints all link messages and some vehicle debug information to local chat. This should never be used unless diagnosing a problem, because it causes significant viewer and script performance impacts.

SPT_DOOR_FORCE

- Integer: [0 to close, 1 to open]
- Key: [name of door prim]

Send to LINK_ROOT to force door open or closed. For example, you can force the driver door open by sending this link message with integer 1 and key “DOOR_LF”.

SPT_DOOR_STATUS

- Key: [JSON object of door statuses]

Sent when a door starts opening or finishes closing. The JSON object includes a key-value pair for each door, with its value set as either 0 for closed or 1 for opened (either partially or fully).

`SPT_DOOR_STATUS_ALT`

- Key: [JSON object of door status]

Sent when a door finishes opening or starts closing. The JSON object includes a key-value pair only for the door that is moving, with its value set as either 0 for starting to close or 1 for finished opening. This can be used in lieu of, or in conjunction with, `SPT_DOOR_STATUS` for scripts that need to monitor both ends of the door movement cycle.

`SPT_DRIVER_READY`

Sent when driver is seated and ready to control the vehicle if the engine is already running.

`SPT_DYN_CONTACT`

- Integer: [contact state (see below)]
- Key: [contact type, either “NONE”, “PAVEMENT”, or “TERRAIN”]

Sent when tire contact state changes. Contact state is either -1 (scraping on side/roof), 0 (no tire contact), or 1-15 (some/all tire contact). If some or all tires are in contact, the contact state is effectively a bitwise integer, calculated as WHEEL_LF * 1 + WHEEL_RF * 2 + WHEEL_LB * 4 + WHEEL_RB * 8. You can determine whether a specific wheel is in contact by using the & operator.

`SPT_DYN_HANDBRAKE`

Send to LINK_ROOT to trigger handbrake until accelerator is released.

`SPT_DYN_SLIP`

- Integer: [wheelslip percentage, 0-100]

Sent when wheelslip changes. Used to trigger skid.

`SPT_DYN_TRACTION`

- Integer: [traction percentage, 0-100]

Send to LINK_ROOT to set the base traction for the wheels. Base traction should always be kept at 100. If base traction is reduced, the vehicle will experience wheelspin on all wheels and will behave unpredictability. (This is an experimental legacy compatibility feature and should be avoided.)

`SPT_ECU_LOADED`

- Key: [JSON object of ECU variable(s)]

Sent when an ECU variable is requested via `SPT_ECU_READ` or written via `SPT_ECU_WRITE`, or the ECU has been reset. Like `NC_CONFIG`, the JSON object may include only certain changed value(s), and you should assume the JSON object contains string values even if the values are defined as integers or floats. Refer to NC_CONFIG for instructions on how to verify that a desired ECU variable is included in the JSON object.

Do not send `SPT_ECU_LOADED` to write ECU variables. The ECU will warn the user and revert any changes. See `SPT_ECU_WRITE`.

`SPT_ECU_READ`

- Key: [ECU variable name]

Send to LINK_ROOT to request the ECU to respond with the value of the requested variable via `SPT_ECU_LOADED`.

`SPT_ECU_READY`

- Integer: [0 for not ready, 1 for ready]
- Key: [GMLS ID]

Sent when the ECU readiness state changes.

`SPT_ECU_RESET`

- Integer: [0 for normal mode, 1 for debug mode]

Send to LINK_ROOT to reset the ECU. If integer is 1, the ECU will enable debug mode before resetting; otherwise, it will disable debug mode.

`SPT_ECU_SUPPRESS`

- Integer: [0 to show change notifications, 1 to suppress change notifications]

Send to LINK_ROOT to optionally suppress ECU change notifications to the vehicle owner. This should only be used if you are changing an ECU variable regularly and do not want to annoy the vehicle owner. The vehicle owner will still be warned about the first ECU change, as well as when any script sets the ECU to suppress change notifications.

`SPT_ECU_WRITE`

- Key: [ECU variable name]|[ECU variable value]

Alternatively:

- Key: [JSON object of ECU variables]

Send to LINK_ROOT to write a value to the specified ECU variable(s). If any of the variable keys exist and the write is successful, the ECU will respond with `SPT_ECU_LOADED` with any new values. If using JSON, multiple variables can be provided at once; otherwise, they must be written one at a time. The write is temporary only – when the ECU is reset, the change will be reverted.

`SPT_EJECT`

- Integer: [0 to turn engine off, 1 to leave engine on (driver only)]
- Key: [seat name]

Send to LINK_SET to eject the passenger sitting in the defined seat. The seat name is the name of the seat, not the prim the seat script is in. Generally, seat names are “DRIVER”, “FPASS”, “LPASS”, “CPASS”, and “RPASS”. If ejecting the driver, use integer 1 to leave engine on. Otherwise, integer is ignored.

`SPT_ENGINE`

- Integer: [0 for off, 1 for on]

Sent when the engine power circuit state changes. Do not send `SPT_ENGINE` – it may cause issues if the engine is started or stopped without the key.

`SPT_FAN`

- Integer: [0 for off, 1 for on]

Sent when the engine fan power circuit state changes. Care should be taken if `SPT_FAN` is sent by a third-party script – it is sent by the engine based on coolant temperature. Consider using `SPT_HVAC` if creating an HVAC unit replacement.

`SPT_GEAR`

- Key: [“P”, “R”, “N”, or “D”]

Sent when gear changes.

`SPT_GEARSHIFT`

- Integer: [0 for released, 1 for held]
- Key: [“P”, “R”, “N”, or “D”]

Sent to LINK_ROOT when gearshift is manually held or released. When gearshift is moved manually to a different position but is continuously held, only `SPT_GEAR` is sent.

`SPT_HORN`

- Integer: [0 for off, 1 for on]

Send to LINK_SET to control the horn.

`SPT_HORN_TAP`

Send to LINK_SET to honk the horn briefly.

`SPT_HVAC`

- Integer: [0 for off, 1 for on]
- Key: [0 for low, 1 for high (certain vehicles only)]

Sent when HVAC controls change HVAC fan speed to activate the engine fan. If replacing HVAC unit, send to LINK_SET to control the engine fan. All HVAC sounds come from the HVAC control unit. In certain vehicles that do not have a standalone HVAC control unit, HVAC sounds come from another vehicle component and the key provides the blower fan speed. In all other vehicles, the key can be ignored.

`SPT_IGNITION`

- Integer: [0 for off, 1 for on]

Sent when ignition power circuit state changes. Do not send `SPT_IGNITION` – it may cause issues if the ignition is started or stopped without the key.

`SPT_KEY`

- Integer: [0 for out, 1 for in, 2 for on, 3 for start]

Sent when key state changes.

`SPT_KEY_SWITCH`

Send to LINK_ROOT to turn the ignition key.

`SPT_LIGHTS_BRAKE_SET`

- Integer: [0 for off, 1 for on]

Sent when brake lights should be turned on or off.

`SPT_LIGHTS_BRIGHT_SET`

- Integer: [0 for off, 1 for on]

Sent when high beam switch is turned on or off. High beams are only activated while headlights are on, but the setting will remain set after turning headlights off.

`SPT_LIGHTS_DRL_SET`

- Integer: [0 to disable, 1 to enable]

Sent when DRLs are enabled or disabled. Does not need to be sent when the DRLs are turned on or off – the lighting script controls that automatically based on the ignition state.

`SPT_LIGHTS_HEAD_SET`

- Integer: [0 for off, 1 for parking lamps, 2 for headlights]
- Key: [“0” for manual headlights, “1” for auto headlights]

Send to LINK_ROOT to control parking lamps and headlights. If auto headlights are used, integer is ignored. For high beams, see `SPT_LIGHTS_BRIGHT_SET`. Do not monitor `SPT_LIGHTS_HEAD_SET` for third-party lights – monitor `SPT_LIGHTS_HEAD_STATE` instead.

`SPT_LIGHTS_HEAD_STATE`

- Integer: [0 for off, 1 for parking lamps, 2 for headlights]
- Key: [“0” for manual headlights, “1” for auto headlights]

Sent when headlight state changes. If auto headlights are used, also sent when headlights are turned on or off automatically. Do not send `SPT_LIGHTS_HEAD_STATE` to control headlights – send `SPT_LIGHTS_HEAD_SET` instead.

`SPT_LIGHTS_LOCK_SET`

- Integer: [0 to disable, 1 to enable]

Sent when light lock button is pressed. When the lock is enabled, the exterior light shut-off timer is disabled and the headlights will remain on permanently with the key removed.

`SPT_LIGHTS_REVERSE_SET`

- Integer: [0 for off, 1 for on]

Sent when reverse lights should be turned on or off.

`SPT_LIGHTS_SIG_SET`

- Key: [“” (empty string) for off, “L” for left turn, “R” for right turn, “H” for hazards]

Sent when turn signal state changes. Send to LINK_SET to control turn signals or hazards.

SPT_LOAD_SCRIPT

- Integer: [0 to reset, 1 to generate a pin]

Send to LINK_ROOT to generate a pin to use with llRemoteLoadScriptPin to load a script into the root. For security, this can only be sent by scripts already linked to the vehicle. The pin will automatically reset once a script is loaded. If requesting a pin or waiting for the pin to reset, monitor SPT_LOAD_SCRIPT_PIN for the response.

SPT_LOAD_SCRIPT_PIN

- Integer: [current pin]
- Key: [root prim UUID]

Sent in response to SPT_LOAD_SCRIPT or when the pin is automatically reset by the root's inventory changing. Note that if [current pin] is 0, scripts cannot be loaded, as llRemoteLoadScriptPin will not accept a pin of 0.

SPT_MEMORY

Send to LINK_SET force all scripts to output memory usage information to local chat.

If desired, you can listen for this link message and output memory usage information.

SPT_MOTOR_MAX

- Key: [circuit name]

Sent when motor connected to circuit is at its limit so that sound is changed. Eligible cirucits are "height_up", "height_down", "tilt_up", and "tilt_down", though more circuits may be added later.

SPT_NOISE

- Key: ["NONE", "PAVEMENT", or "TERRAIN"]

Sent to change road noise type. See also SPT_DYN_CONTACT.

SPT_OSD_SEND

- Key: [OSD tool UUID]

Sent when OSD tool is connected. Used to send data to OSD tool.

SPT_PERMS

- Key: [prim name]|[inventory name]|[inventory type]|[perm mask]|[next perm mask]

Sent by one script in each factory prim for each item in that prim's inventory when the script resets. Used to check permissions on inventory items prior to release. Do not send SPT_PERMS – it can trigger a permissions violation.

SPT_RESET

- Key: [script name, or “ALL”]

Send to LINK_SET to reset any factory script with the defined script name. If key is “ALL”, all scripts are reset. The ECU always ignores this link message – use SPT_ECU_RESET instead. Aftermarket scripts do not need to monitor for SPT_RESET, but can do so.

SPT_SEATED

- Key: [seat name]|[passenger UUID or NULL_KEY]

Sent when a passenger sits or unsits. The seat name is the name of the seat, not the prim the seat script is in. Generally, seat names are “DRIVER”, “FPASS”, “LPASS”, “CPASS”, and “RPASS”. If the seat now holds a passenger, their UUID is included; otherwise, NULL_KEY is sent ("00000000-0000-0000-0000-000000000000"). If monitoring SPT_SEATED to grant access for some aftermarket function to a seated passenger, SPT_ANIM_CUFFED should also be monitored to ensure handcuffed passengers do not have access.

SPT_SOUND

- Key: [local position vector]|[sound UUID]|[volume, 0.0-1.0 (float)]

Send to LINK_SET to trigger llPlaySound at the specified local position. This uses a sound emitter prim, which moves to that position and plays the sound. The start of the sound may play at the last location of the emitter prim due to latency. This is used for doors and some interior components to reduce the number of sound emitter scripts in the vehicle.

SPT_SOUND_POS_RESET

Send to LINK_SET to reset the sound emitter prim to its default position. See SPT_SOUND.

SPT_SOUND_RELEASE

- Integer: [rev state to play release sound from, 1-10]

Sent when engine plays release to idle sound.

SPT_SOUND_REV

- Integer: [rev state to play rev sound to, 1-10, or 0 to play redline]

Sent when engine plays rev sound.

SPT_SOUND_START

Sent when engine plays start sound.

SPT_SOUND_STALL

Sent when engine plays stalled start sound.

SPT_SOUND_STOP

Sent when engine plays stop sound.

SPT_SPOT_ACTIVE

- Integer: [0 for inactive, 1 for active]
- Key: [seat name]

Sent when spotlight aiming is activated or deactivated. Currently only sent for FPASS (SPT_CONTROL_MODE is used for DRIVER), but may be sent for other seats in the future.

SPT_SPOT_ANGLE

- Integer: [90, 45, 0, -45, or -90, indicating angle to twist arm for animation]
- Key: [“d” for driver, or “p” for front passenger]

Sent when the spotlight handle angle changes enough to trigger a different spotlight aiming animation.

SPT_STEER

- Integer: [-1 for left, 0 for straight, 1 for right]

Sent when wheel is turned or released using turning keys to trigger driver turning animations.

SPT_TEMP

- Integer: [coolant temperature percent, 0-100]

Sent when engine coolant temperature changes by at least 1%.

SPT_THROTTLE

- Integer: [target speed in miles per hour]
- Key: [throttle index]|[number of throttle indices]

Sent when engine starts or throttle target speed is changed. Throttle index starts at 1.

SPT_VISOR

- Integer: [0 for closed, 1 for open]
- Key: [visor prim name]

Sent when a sun visor is open or closed.

`SPT_WHEEL_ANGLE`

- Key: [wheel angle to turn to, range from -1.0 (left) to 1.0 (right) (float)]

Sent when wheel angle needs to be changed due to a turn input. Engine generally always sends either -1.0, 0.0, or 1.0, though the suspension can accommodate smaller values by turning wheels more slowly. Suspension smooths out wheel turns – `SPT_WHEEL_ANGLE` only indicates which direction the wheels are turning towards but does not indicate the exact angle when sent.

Section 5
ECU Variables

Section 5: ECU Variables

Introduction

The SPT3 ECU is a script that maintains, among other ancillary functions, a large collection of variables used to modify a variety of vehicle parameters. These variables control engine, transmission, turning, and suspension behavior, door and seat layouts, cameras, tire information, and certain variables for standardized scripts that can change on a per-vehicle or per-variant basis so that scripts can be standardized as much as possible.

When the ECU is reset, either manually or by shift-dragging the vehicle, it prompts all other scripts in the vehicle to reset using `SPT_RESET`. Then, it performs a self-check, connects to NBS Keyring, sends out a list of default ECU variables, and triggers a vehicle config download. Once complete, the ECU is "ready" and the vehicle can be operated normally.

ECU variables can be changed in the field to allow certain modifications to work better, or to change certain features not accessible via Setup, using `SPT_ECU_WRITE`.

ECU Variable Glossary

Be aware that this glossary is not exhaustive. We attempt to document variables that modders may want to use here, but all vehicles have additional variables, or may not have all of the variables listed here. To confirm whether your vehicle utilizes a specific variable, use `SPT_ECU_READ`. Care should be taken when writing to the ECU – it is strongly recommended that scripts first read the value first to get a baseline value for any future changes.

`ac_run`

Sound UUID for HVAC blower run.

`ac_start`

Sound UUID for HVAC blower start.

`ac_start_time`

Time in seconds for HVAC blower start sound.

`ac_stop`

Sound UUID for HVAC blower stop.

`ac_vol`

Volume for HVAC blower sounds.

`anim_prefix`

Prefix for animation names.

`aux1_latch`, `aux2_latch`, `aux3_latch`

Whether each aux switch is latched or momentary.

`cameras_DRIVER`

!-separated strided list of camera names, CAMERA_ACTIVE, CAMERA_BEHINDNESS_ANGLE, CAMERA_BEHINDNESS_LAG, CAMERA_DISTANCE, CAMERA_FOCUS_OFFSET (if not cinematic), CAMERA_FOCUS_LAG, CAMERA_FOCUS_LOCKED, CAMERA_FOCUS_THRESHOLD, CAMERA_PITCH, CAMERA_POSITION (if not cinematic), CAMERA_POSITION_LAG, CAMERA_POSITION_LOCKED, CAMERA_POSITION_THRESHOLD, CAMERA_POSITION offset for cinematic mode (set to <0,0,0> for not cinematic). Used for driver.

`cameras_*PASS`

See cameras_DRIVER for format. `*` can be any character, which in the form `*PASS` is the seat name. Common seat names are FPASS, LPASS, RPASS, et cetera.

`cluster_name`

Name of prim to send `SPT_CLUSTER_UPDATE` to.

`demo`

Whether the vehicle is in demo mode.

`dome_light_doors`

Comma-separated list of door names that can trigger the dome light.

`door_ajar_doors`

Comma-separated list of door names that can trigger the door ajar warning message and chime.

`doortag`

UUID of the build tag texture, usually located inside the driver door.

`driveshaft_rot`

Default rotation (in vector degrees) for driveshaft.

`dyn_acc_lean`

Acceleration lean torque power.

`dyn_accel_rate`

Target power change rate when speed is below target speed.

`dyn_ADE`

VEHICLE_ANGULAR_DEFLECTION_EFFICIENCY vehicle parameter.

`dyn_angle_start`

Forward angle in degrees to start degrading engine power.

`dyn_angle_end`

Forward angle in degrees for maximum engine power loss.

`dyn_angle_exp`

Exponential engine power degradation for forward angle.

`dyn_change_rate`

Power change rate when power is below target power.

`dyn_downforce_gravity`

Vertical force to apply when in the air.

`dyn_downshift_decel_rate`

Power change rate when power is above target power.

`dyn_eff_air`

Efficiency when in the air.

`dyn_eff_brake`

Efficiency when braking.

`dyn_eff_coast`

Efficiency when coasting.

`dyn_eff_handbrake`

Multiplier for rear traction when handbrake is enabled.

`dyn_eff_scrape`

Efficiency when scraping.

`dyn_LDE`

VEHICLE_LINEAR_DEFLECTION_EFFICIENCY vehicle parameter.

`dyn_LDT`

VEHICLE_LINEAR_DEFLECTION_TIMESCALE vehicle parameter.

`dyn_LMDT`

VEHICLE_LINEAR_MOTOR_DECAY_TIMESCALE vehicle parameter.

`dyn_LMT`

VEHICLE_LINEAR_MOTOR_TIMESCALE vehicle parameter.

`dyn_max_acc`

Maximum change in engine power per engine cycle loop.

`dyn_pwr_brake`

Braking inertia (inverse modifier for braking friction).

`dyn_pwr_scrape`

Scraping inertia (inverse modifier for scraping friction).

`dyn_reartrac_exp`

Rear traction exponential effect on power.

`dyn_recovery_handbrake`

Recovery value for rear traction when handbrake is disabled. Doubled when there is no acceleration or braking.

`dyn_reverse_mod_threshold`

Inverse modifier for setting target speed when in reverse.

`dyn_reverse_switch_spd`

Speed in m/s when reverse can be automatically enabled/disabled.

`dyn_stop_brake_spd`

Speed in m/s when to set engine stop brake (non-physical).

`dyn_throt`

A comma-separated list of throttle target speeds in miles per hour.

`dyn_throt_default`

Default throttle index when engine started. Counted starting at 0.

`dyn_throt_r`

Throtle target speed for reverse in miles per hour. Modified by `dyn_reverse_mod_threshold` to increase when throttle is set high.

`dyn_turn_balance`

Speed in m/s when to switch to high speed turn effect.

`dyn_turn_dampen`

Number of engine cycle loops of holding turn until maximum turn power.

`dyn_turn_lean`

Turn lean torque power after being clamped to 1.0.

`dyn_turn_lean_prefactor`

Turn lean torque power before being clamped to 1.0.

`dyn_turn_lean_max_angle`

Maximum vertical roll angle in radians for turn lean cutout.

`dyn_turn_power_base`

Linear turn power at high speed.

`dyn_turn_power_speed`

Exponential turn power at high speed.

`dyn_VAEN`

VEHICLE_VERTICAL_ATTRACTION_EFFICIENCY vehicle parameter.

`dyn_VATN`

VEHICLE_VERTICAL_ATTRACTION_TIMESCALE vehicle parameter.

`dyn_VAEU`

VEHICLE_VERTICAL_ATTRACTION_EFFICIENCY vehicle parameter when unflipping.

`dyn_VATU`

VEHICLE_VERTICAL_ATTRACTION_TIMESCALE vehicle parameter when unflipping.

`dyn_ZLFT`

Z value for VEHICLE_LINEAR_FRICTION_TIMESCALE vehicle parameter.

`eng_idle`

Sound UUID for engine idle. If blank, use default sound.

`eng_rev_limit`

Sound UUID for engine revving up to the limiter. If blank, use default sound.

`eng_run_limit`

Sound UUID for engine running at the limiter. If blank, use default sound.

`eng_rel_low`

Sound UUID for engine throttle being released while at low RPM. If blank, use default sound.

`eng_rel_high`

Sound UUID for engine throttle being released while at high RPM. If blank, use default sound.

`eng_run_low`

Sound UUID for engine running at throttle positions 1-3. If blank, use default sound.

`eng_run_med`

Sound UUID for engine running at throttle positions 4-7. If blank, use default sound.

`eng_run_high`

Sound UUID for engine running at throttle positions 8-10. If blank, use default sound.

`eng_run_max`

Sound UUID for engine running at max throttle. If blank, use default sound.

`eng_start`

Sound UUID for engine starting. If blank, use default sound.

`eng_stall`

Sound UUID for engine failing to start (no fuel). If blank, use default sound.

`eng_stop`

Sound UUID for engine stopping. If blank, use default sound.

`eng_rel_low_time`

Time in seconds for `eng_rel_low` to transition into `eng_idle`. If blank, use default time.

`eng_rel_high_time`

Time in seconds for `eng_rel_high` to transition into `eng_idle`. If blank, use default time.

`eng_rev_limit_time`

Time in seconds for `eng_rev_limit` to transition into `eng_run_limit`. If blank, use default time.

`eng_start_time`

Time in seconds for `eng_start` to transition into `eng_idle`. If blank, use default time.

`exhaust_factor`

Adjustment factor for exhaust particles' opacity and size.

`getout_pos`

!-separated strided list of seat names and seat "get out" local positions.

`getout_rot`

"Get out" local rotation.

`fan_run`

Sound UUID for fan run.

`fan_start`

Sound UUID for fan start.

`fan_start_time`

Time in seconds for fan start sound.

`fan_stop`

Sound UUID for fan stop.

`fan_vol`

Volume for fan sounds.

`gfs_diesel`

Use of diesel fuel for GFS.

`gfs_efr`

Engine fuel rate adjustment for GFS.

`gfs_manual`

Use of manual fuel selection for GFS. Note that currently, the type of fuel used when filling is ignored for SPT3 vehicles, so changing this variable and the user selecting the wrong fuel type would not cause any problems.

`gfs_max`

Maximum GFS fuel capacity in US gallons.

`gfs_mpg`

Average GFS fuel usage in miles per US gallon.

`gfs_pos`

Local position of GFS nozzle.

`gfs_rot`

Local rotation of GFS nozzle.

`gfs_start`

Starting GFS fuel amount in US gallons.

`gfs_warn`

Ratio (from 0 to 1) of when to warn the driver that the GFS fuel level is low.

`hood_ajar_doors`

Comma-separated list of door names that can trigger the hood ajar warning.

`horn_run`

Sound UUID for horn run.

`horn_stop`

Sound UUID for horn stop.

`man_url`

Vehicle owner's guide URL.

`menu_ignore`

Comma-separated list of prim names that should be ignored if clicked (vehicle menu will not be sent).

`nozzle_pos`

The nozzle_pos variable provided by the GFS Fuel Nozzle Positioner tool.

`nozzle_rot`

The nozzle_rot variable provided by the GFS Fuel Nozzle Positioner tool.

`phys_max_z`

Local Z-position maximum of PHYSICS prim at maximum ride height.

`phys_min_z`

Local Z-position minimum of PHYSICS prim at minimum ride height.

`phys_pos`

Local position of PHYSICS prim. When changing this for the first time, it is a good idea to move the vehicle up or down based on the current value of phys_pos compared to the new value; otherwise, the vehicle will jump or fall when next driven. This value will also automatically offset `tire_top_lb`, `tire_top_rb`, `tire_top_lf`, `tire_top_rf` and the corresponding raycast positions for the suspension.

`phys_size`

Size of PHYSICS prim. This should not be changed, as it may affect stability.

`prod_name`

Vehicle product name.

`seats_doors`

Comma-separated strided list of seat names and door prim names.

`seats_prims`

Comma-separated strided list of seat prim names and seat names.

`seats_targets`

!-separated strided list of seat names, seat local positions, and seat local rotations.

`sig_auto_off_start`

How many engine cycles until the turn signal auto off timer starts.

`sig_auto_off_timer`

Time in seconds for the turn signal auto off timer.

`sig_rate`

Rate in flashes per second for turn signal flasher.

`sig_sound`

Sound UUID for turn signal.

`sig_vol`

Volume for turn signal sound.

`sit_DRIVER_base`

Base position of driver seat.

`sit_DRIVER_upper`

Maximum position of driver seat. Y-value is ignored.

`sit_DRIVER_lower`

Minimum position of driver seat. Y-value is ignored.

`sit_DRIVER_pos`

Sit target position for driver.

`sit_DRIVER_rot`

Sit target rotation for driver.

`sit_DRIVER_custom`

Custom position movement offset for driver.

`steer_range`

Rotational range for steering wheel when turning.

`steer_release`

Speed factor for steering wheel to return to center when no longer turning.

`steer_turn`

Speed factor for steering wheel to turn to limit when turning.

`tire_offset`

Local positional offset of all tires, based on the offset for WHEEL_LF. This is rotated automatically on other tires. Note that this variable is automatically overridden if WHEEL_LF uses the SPT3 wheel description format.

`tire_range_vert`

Vertical range of tire movement in meters.

`tire_rot`

Rotation (in vector degrees) for WHEEL_LF. This is rotated automatically on other tires. Note that this variable is automatically overridden if WHEEL_LF uses the SPT3 wheel description format.

`tire_side_axis`

Unit vector for side axis of tire. Note that this variable is automatically overridden if WHEEL_LF uses the SPT3 wheel description format.

`tire_spin_axis`

Unit vector for spin axis of tire. Note that this variable is automatically overridden if WHEEL_LF uses the SPT3 wheel description format.

`tire_top_lb`, `tire_top_rb`, `tire_top_lf`, `tire_top_rf`

Local position of top of tire for WHEEL_LB, WHEEL_RB, WHEEL_LF, and WHEEL_RF. This should not be changed when lifting the vehicle using `phys_pos`; they should be changed only in reference to the default `phys_pos` value.

`tire_tread_offset`

Horizontal offset of tire tread from the tire's axis for skid particle emitter.

`tire_turn_axis`

Unit vector for turn axis of tire. Note that this variable is automatically overridden if WHEEL_LF uses the SPT3 wheel description format.

`ver`

Vehicle version number.

`wheel_range`

Rotational range for front wheels when turning.

`wheel_release`

Speed factor for front wheels to return to center when no longer turning.

`wheel_turn`

Speed factor for front wheels to turn to limit when turning.

`wheelspin_pavement`

Comma-separated list of sound UUIDs for wheelspin when on pavement.

wheelspin_terrain

Comma-separated list of sound UUIDs for wheelspin when on terrain.

Section 6
Vehicle Configuration Values

Section 6: Vehicle Configuration Values

Introduction

SPT3 vehicles use a JSON-based configuration, edited using a Web Setup service provided by NBS. Configuration values can be modified using `NC_CONFIG_SET`. When changed, either via link message, by loading defaults, or by manual changes in Web Setup, scripts will receive `NC_CONFIG`. All configuration values are JSON object key-values.

Since each vehicle has a different configuration layout, this section is split up by vehicle type.

Note that this section lists the configuration variables' types. Some variables are listed as "Boolean" type. These are NOT JSON true/false Booleans – they are integers that can only be 0 or 1. This is due to LSL's JSON handling requiring the use of JSON_TRUE and JSON_FALSE constants for JSON Booleans, which is annoying enough that we elected to just use integers.

Some variables are listed as "bitwise" type. These variables use bitwise flags defined in the variable's description. Unless stated otherwise, these variables can be written by simply adding up the individual flag(s) desired, each multiplied by their flag's value, and can be deciphered by using the & operator. For example:

```
integer config_ant_fin = 1;
integer config_ant_radio = 0;
integer config_ant_puck = 1;
integer config_ant_short = 0;
integer config_ant_long = 0;
integer set_ant = config_ant_fin * 1 + config_ant_radio * 2
                  + config_ant_puck * 4 + config_ant_short * 8
                  + config_ant_long * 16;
string json_config = "{\"ant\":" + (string)set_ant + "}";
// Then, to read the value...
string value = llJsonGetValue(json_config, ["ant"]);
if (value != JSON_INVALID)
{
    if ((integer)value & 1) llOwnerSay("Fin antenna enabled");
    else llOwnerSay("Fin antenna disabled");
    if ((integer)value & 2) llOwnerSay("Radio antenna enabled");
    else llOwnerSay("Radio antenna disabled");
    if ((integer)value & 4) llOwnerSay("Puck antenna enabled");
    else llOwnerSay("Puck antenna disabled");
    if ((integer)value & 8) llOwnerSay("Short antenna enabled");
    else llOwnerSay("Short antenna disabled");
    if ((integer)value & 16) llOwnerSay("Long antenna enabled");
    else llOwnerSay("Long antenna disabled");
}
```

SZYM Paladin Configuration Values

Note that this vehicle has five trim levels – Base, Utility, Police, HX, and HXL. For some configuration values, the defaults change for each trim level.

Note also that some of these configuration values are ignored for certain vehicle varieties. For example, for 2DR versions, the “mold” option ignores the flag for rear side door moldings.

`am`

String. Access mode. Can be one of the following:

- “k” – Keyring
- “o” – owner only
- “g” – same group only
- “e” – everyone

`ant`

Bitwise. Antenna options. Includes the following flags:

- 1 – fin antenna
- 2 – front fender radio antenna
- 4 – puck antenna
- 8 – short roof antenna
- 16 – long roof antenna

`aux`

Boolean. Auxiliary switches.

`bed`

Integer. Bed color. Can be one of the following:

- 0 – color-matched painted
- 1 – black plastic

`beep`

Integer. Reversing beeper. Can be one of the following:

- 0 – not installed
- 1 – beeping tone
- 2 – white noise

`bpri`

Boolean. Priority 4X MAX lightbar brake light mode.

`bump`

Integer. Bumper material. Can be one of the following:

- 0 – color-matched painted
- 1 – black plastic
- 2 – chrome

`cam`

Boolean. Dash camera.

`cap`

Boolean. Factory bed cap.

`chrome`

Boolean. Grille chrome trim.

`chid`

String. Colors for corner hideaway flashers. Must be 6 characters (characters referring to fog lamp hideaways are ignored if fog lamps are disabled). Each character must be R (Red), A (Amber), G (Green), B (Blue), P (Purple), or W (White).

`cpri`

String. Colors for Priority 4X MAX and Agent 4X MAX, if installed. Must be 16 characters (some characters are skipped for Agent, and characters for non-installed half of Agent are ignored). Each character must be R (Red), A (Amber), G (Green), B (Blue), P (Purple), or W (White).

`cstr`

String. Colors for strobes. Must be 8 characters (characters referring to non-installed strobes are ignored). Each character must be R (Red), A (Amber), G (Green), B (Blue), P (Purple), or W (White).

`cver`

String. Colors for Vertex 4, if installed. Must be 5 characters (counted from left to center, then skipping to far right pod). Each character must be R (Red), A (Amber), G (Green), B (Blue), P (Purple), or W (White).

`cup`

Bitwise. Cupholder junk. Includes the following flags:

- 1 – coffee mug
- 2 – water bottle

`ddc`

Boolean. Disables door chime.

`deflect`

Boolean. Hood bug & stone deflector.

`diesel`

Boolean. Diesel engine.

`dome`

Integer. Dome light type. Can be one of the following:

- 0 – factory
- 1 – aftermarket

`drl`

Boolean. “Halo” DRLs.

`dwheel`

Integer. Wheel rim type (Dual Rear Wheel Kit wheels only). Can be one of the following:

- 0 – extended chrome
- 1 – extended black

If the Dual Rear Wheel Kit is not installed, this option is ignored; see `wheel`.

`dws`

Boolean. Disable rear window switches. Note that doors can still be opened unless passengers are handcuffed manually.

`ext0d`, `ext0n`, `ext0s`

Strings. Front exterior texture UUIDs for diffuse, normal, and specular textures.

`ext1d`, `ext1n`, `ext1s`

Strings. Rear exterior texture UUIDs for diffuse, normal, and specular textures.

`fog`

Boolean. Fog lamps.

`gfs`

Integer. GFS fuel mode. Can be one of the following:

- 0 – realistic fuel consumption
- 1 – infinite fuel
- 2 – 5X fuel consumption

`grille`

Integer. Grille bars. Can be one of the following:

- 0 – honeycomb only
- 1 – honeycomb + side bars
- 2 – honeycomb + side bars + upper/lower bars
- 3 – honeycomb + side bars + upper/lower bars + extra lower bar

Note that side bars are forced on if grille strobes are installed.

`ha`

Boolean. Corner hideaway flashers.

`hubcap`

Boolean. Hubcaps (moderate duty and heavy duty factory wheels only).

`idr1`, `idr2`, `idr3`

Strings. IntelliDoor Remote passwords for buttons 1, 2 and 3.

`int0d`, `int0n`, `int0s`

Strings. Interior A texture UUIDs for diffuse, normal, and specular textures.

`int1d`, `int1n`, `int1s`

Strings. Interior B texture UUIDs for diffuse, normal, and specular textures.

`lt`

Boolean. Locks theme from being changed. When enabled, the vehicle script will not apply any theme textures, and Setup will hide the exterior theme selections. This is set by the theme applier tool.

`marker`

Boolean. Roof marker lights.

`mat`

Integer. Floor mats. Can be one of the following:

- 0 – not installed
- 1 – rubber
- 2 – carpet

`metric`

Boolean. Metric units.

`mold`

Bitwise. Side door molding options. Includes the following flags:

- 1 – front side doors
- 2 – rear side doors

`pb`

Bitwise. Pushbar type. Includes the following flags:

- 1 – base (should always be enabled if pushbar is installed)
- 2 – upper bar
- 4 – wraparound bars
- 8 – headlight protector bars

`rack`

Integer. Rack behind rear window. Factory bed cap must not be installed. Can be one of the following:

- 0 – not installed
- 1 – rack only
- 2 – rack + upper mounting brackets

`radio`

Boolean. Police radio sounds.

`rain`

Boolean. Side window rain shields.

`rha`

Boolean. Ride height adjustment package.

`spotc`

Integer. Spotlight(s) color. Can be one of the following:

- 0 – black plastic
- 1 – chrome

`spotd`

Integer. Driver spotlight type. Can be one of the following:

- 0 – not installed
- 1 – halogen
- 2 – LED

`spotp`

Integer. Passenger spotlight type. Can be one of the following:

- 0 – not installed
- 1 – halogen
- 2 – LED

`srn`

String. Name of installed siren. Can be “GT6500”, “GT20A”, “VR900”, “BF200”, “UG4000”, “SP700”, “GT4500”, “MP224”, “eM”, “PC9”, “MP224E”, or “EURO1”. Can also be “” (empty string) to disable the siren.

`stack`

Boolean. Exhaust stacks. Factory bed cap must not be installed.

`step`

Boolean. Running board steps.

`strobe`

Bitwise. Installed strobe locations. Includes the following flags:

- 1 – grille
- 2 – dash
- 4 – side doors
- 8 – tailgate

`tcap`

Integer. Tailgate aftermarket wing. Can be one of the following:

- 0 – not installed
- 1 – black plastic
- 2 – color-matched painted plastic

`tint`

Integer. Window tint level. Can be one of the following:

- 0 – 5% side/back window tint + 5% windshield tint
- 5 – 5% side/back window tint
- 20 – 20% side/back window tint
- 40 – 40% side/back window tint
- 60 – no window tint

`trim`

String. Trim level. Can be “” (empty string), “U” (Utility), “P” (Police), “HX”, or “HXL”.

`toll`

Boolean. ETC toll tag.

`visor`

Integer. Exterior sun visor. Can be one of the following:

- 0 – not installed
- 1 – black plastic
- 2 – color-matched painted plastic

`wcover`

Boolean. Steering wheel cover.

`wheel`

Integer. Wheel rim type (factory standard wheels only). Can be one of the following:

- 0 – standard chrome
- 1 – performance chrome
- 2 – moderate duty chrome
- 3 – heavy duty chrome
- 4 – standard black
- 5 – performance black
- 6 – moderate duty black
- 7 – heavy duty black

If the Dual Rear Wheel Kit is installed, this option is ignored; see `dwheel`.

`ww`

Boolean. Wig-wag headlight flasher module.

Section 7
Standard Equipment Specifications

Section 7: Standard Equipment Specifications

Introduction

This section describes specifications for certain standard equipment on SPT3 vehicles.

Radio Head Unit

Most SPT3 vehicles come with a 195 x 135 x 70 mm (W x H x D) head unit. This will continue to be the standard size for SPT3 vehicles. For reference, double DIN in real life is 180 x 100 mm. If designing a head unit based on real life sizes, it may be necessary to include an adapter plate to fit the oversized SPT3 head unit opening.

No specific functions are required of the head unit, so it can be removed and replaced if desired with no side effects, and aftermarket equipment manufacturers may design and script it however they see fit.

HVAC Control Unit

Most SPT3 vehicles come with a 190 x 125 x 100 mm (W x H x D) HVAC control unit. This will continue to be the standard size for SPT3 vehicles.

No specific functions are required of the HVAC control unit; however, on SPT3 vehicles, it may include the only in-cabin button for the hazard warning flashers via `SPT_LIGHTS_SIG_SET`. Otherwise, it can be replaced with whatever equipment is desired. If you want to control the engine fan for HVAC purposes, see `SPT_HVAC`. If you want to use the built-in sounds, you must do so in the control unit itself, but can monitor `ac_*` for sounds to use.

Gauge Cluster

Most SPT3 vehicles come with a 400 mm x 170 mm (W x H) gauge cluster. This will continue to be the standard size for SPT3 vehicles.

The gauge cluster is expected to send `SPT_CLOCK` once per minute. The gauge cluster is also expected to monitor `SPT_CLUSTER_IND`, `SPT_CLUSTER_RESET`, `SPT_CLUSTER_UPDATE`, `SPT_DOOR_STATUS`, `SPT_DYN_SLIP`, `SPT_ENGINE`, `SPT_GEAR`, `SPT_IGNITION`, `SPT_LIGHTS_BRIGHT_SET`, `SPT_LIGHTS_HEAD_STATE`, `SPT_LIGHTS_SIG_SET`, `SPT_TEMP`, `SPT_THROTTLE`, and `SPT_UNITS`. Aftermarket gauge clusters should either change `cluster_name` or rename themselves to whatever `cluster_name` is set to to receive `SPT_CLUSTER_UPDATE`.

Section 8
Recommended Integrations

Section 8: Recommended Integrations

Introduction

This section includes a list of free products developed by the NTBI Group that are compatible with SPT3. Aftermarket equipment manufacturers should review these integrations to determine if they could improve their customers' experience.

Global Mod Link System (GMLS)

The Global Mod Link System (GMLS) is an open-source set of scripts used to automatically position, rotate, and rescale aftermarket modifications for modifiable vehicles.

SPT3 vehicles include GMLS compatibility, and aftermarket equipment manufacturers are strongly urged to use GMLS in their products to help eliminate installation mistakes. Setup is simple and takes only a few minutes.

For more information about GMLS, see: https://ntbigroup.com/gmls

Global Vehicle HUD (GVHUD)

The Global Vehicle HUD (GVHUD) is the HUD used for all SPT3 vehicles. It is a self-configurable HUD that is capable of controlling vehicle components.

SPT3 vehicles include GVHUD compatibility by default, so aftermarket equipment should listen for various link messages sent via the GVHUD Controller script, such as `SPT_CIRCUIT`, `GVHUD_SW`, and `SPT_LIGHTS_HEAD_STATE`, instead of attempting to interact with the GVHUD directly. Attempting to connect to the GVHUD will disconnect it from the vehicle.

If your equipment needs to use a GVHUD button that is automatically hidden when factory equipment is not installed, you may need to send `GVHUD_FORCE_SHOW`.

Some link messages require knowledge of the GVHUD button IDs. Each type of button – for example, a red lightbar button – has its own GVHUD button ID, regardless of where it is positioned on the HUD for each vehicle. For a list of GVHUD button IDs, as well as instructions on how to integrate GVHUD with a non-SPT3 vehicle, see: https://ntbigroup.com/gvhud

Some SPT3 vehicles have an auxiliary switch option. If enabled, a certain number of auxiliary switches will be shown on the GVHUD and possibly in the cabin. You can listen for the state of these switches by monitoring `SPT_CIRCUIT`. You can also optionally use `NC_CONFIG_SET` when linked to enable the auxiliary switch option.

Global Fuel System (GFS)

The Global Fuel System (GFS) is a free fuel system maintained by the NTBI Group. SPT3 vehicles support GFS, and anyone can get a free GFS fuel pump and GFS fuel cans.

By default, SPT3 vehicles do not have full GFS simulation – fuel will never run out.

We encourage creative uses of GFS. For more information, see: https://ntbigroup.com/gfs

Global License Plate System (GLPS)

The Global License Plate System (GLPS) is a free, open-source license plate system maintained by the NTBI Group. Anyone can attach a GLPS license plate onto their own vehicle and sell their vehicle with GLPS license plates installed.

SPT3 vehicles include front and rear GLPS plates. They can be moved, and the front plate can be unlinked, but the rear plate should generally remain attached.

The standard dimensions for GLPS plates are 12 x 6 inches for US and Japanese plates, and 520 x 130 mm for EU and Russian plates. However, SPT3 vehicles usually upscale plates to match the vehicle dimensions, so check fitment first.

For more information about GLPS, see: https://ntbigroup.com/glps

NBS Keyring

NBS Keyring is a free access management service that allows users to define access lists for restricted-access objects in-world. SPT3 vehicles support Keyring, though it must be enabled manually by changing the vehicle's access mode to Keyring.

Interactive aftermarket equipment should make use of `SPT_ACCESS_CHECK` to check if a user has access to the vehicle regardless of whether Keyring is enabled or not. If the Keyring access mode is enabled, this function sends the user's UUID and active group UUID, if any, to the Keyring service. Keyring check requests are not saved by NBS. If the same request has been made to the service within the past 60 seconds (subject to change in the future), the vehicle itself will return a cached result. The cache expiry is based on the latest request that was sent to the service and that did not result in a cache hit, so if testing Keyring integration, wait at least one minute after making any changes to the Keyring access list.

Aftermarket equipment should also monitor `SPT_SEATED` and `SPT_ANIM_CUFFED` to bypass the access check for seated, unhandcuffed passengers.

For more information about NBS Keyring, see: https://northbridgesys.com/keyring/support