Vehicle GRF Tutorials/ Examples -

Common GRF Coding Challenges

Richard Wheeler (Zephyris) 2008

FAQs

Can I copy and paste the code into my GRFs?

Yes, and no credit required.

Can I copy and paste the code into a wiki?

Yes, but reference this document and let me know about any major corrections.

Can you add this example?

Probably, just let me know...

I think I have found a mistake...

Almost certainly, let me know and I will correct it!

Can I contribute?

Of course! This is a google docs document, so collaboration is simple.

Action0s

I want to change a vehicle's properties.

This is a basic action0 summary on how to change vehicle properties.

Nfo Code Structure

```
 -1 * 0    00 // Action0 (so changes vehicle properties)
           00 // Feature 0 - trains (change to match your vehicle type)
           NN // Number of properties to alter
           01 // Alter properties of one vehicle (if more than one it alters this
vehicle and the ones with successive vehicle IDs)
           ** // Vehicle ID (a hexadecimal code for the vehicle to alter, change this
to match your target vehicle)
              //(
           XX // Property XX
           YY // Value YY ) - repeat NN times
```

Look up the properties and the values they can take in the TTDPatch wiki, for example:

```
 -1 * 0    00 // Action0
           00 // Feature 0 - trains
           04 // Alter 04 properties
           01 // Alter properties of one vehicle
           ** // Vehicle ID
           05 // Property 05 - track type
           01 // Value 01 - monorail
           0B // Property 0B - power
           10 0E // Value 10 0E - 3600 HP
           12 // Property 12 - sprites
           FD // Value FD - new graphics
           27 // Property 27 - miscellaneous flags bitmask
           03 // Value 03 = 01 + 02 - tilting train and two company colours
```

Notice the way each property works:

Property 05:
Described by a byte, so the value uses one byte. 3 possible values, 0, 1 or 2. Each value corresponds to a certain outcome - in this case track type.

Property 0B:
Described by a word value, so the value uses two bytes. The value is directly converted to a numerical outcome - in this case train power.

Property 12:
Described by a byte, so the value uses one byte. Many possible values (to use existing game sprites) or FD for new graphics described by action 3 and action2.

Property 27:
An 8 bit (one byte) bit mask. Different properties depending on which bits are set.

How does a bitmask work?

{{not finished}}

I want to do more than just modify trains in a climate, I want more trains!

Basic action0 summary on changing vehicle climate availability, and its drawbacks if the grf is loaded while the player is in the wrong climate.

{{not finished}}

I want to make some trams!

Basic action0 summary on the road vehicle miscellaneous flag for trams. Does not include how to make articulated road vehicles.

{{not finished}}

Action4s

I want to change a vehicle's name.

Basic action4 summary on changing vehicle names.

{{not finished}}

Action1s and Normal Action2 and 3s

Normal action2s and action3s are used to define new graphics for a vehicle, which will be used if the vehicle has "use new sprites" set in its action0.

I want a train which uses custom graphics.

This is a basic example of how to use an action2 and action3 to give a vehicle non-cargo specific graphics. It also covers how to assign new graphics to a vehicle via action0. Note you cannot "make new trains", only modify existing ones.

Nfo "translation"

There is a vehicle with no cargo specific graphics (the train).
These are the graphics for the train.
These are the real sprites to use.

This corresponds to:

- Action3
- Action2
- Action1/Real sprites

However, because nfo code can only refer to an Action2 that has been previously defined, this logical chain must go in reverse.

Nfo Code

```
 -1 * 0    00 // Action0
           00 // Feature 0 - trains
           01 // Number of properties to alter
           01 // Alter properties of one vehicle
           ** // Vehicle ID
           12 // Property 12 - sprites for vehicle
           FD // Value FD - use new sprites as defined by action3
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           01 // One graphics sets
           08 // 8 sprites per graphics set
// 8 real sprites - one sets of 8 - set ID 00 00 (train graphics)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for train graphics
           00 00 // Action0 sprite set for train graphics
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to
           00 // Number of cargo specific graphics sets 00 - none
           AA 00 // Default action2 ID
```

Notice how the action2 ID may be chosen (in this case the action2 has an ID of "AA"), but the real sprite set IDs from the action1 cannot. Real sprite sets from an action1 are always numbered counting up from zero, and referenced as a word. Note additional properties can also be altered in the same action0, including speed, cargo capacity, etc.

I want a wagon which uses custom graphics, with different graphics depending on how full the vehicle is.

This is a more advanced example of how to use an action2 and action3 to give a vehicle non-cargo specific graphics with different loading stage graphics. It also covers how to assign new graphics to a vehicle via action0.

Nfo "translation"

There is a vehicle with no cargo specific graphics (the wagon).
These are the graphics for the wagon, there are different sets depending on how full it is.
These are the real sprites to use.

This corresponds to:

- Action3
- Action2
- Action1/Real sprites

However, because nfo code can only refer to an Action2 that has been previously defined, this logical chain must go in reverse.

Nfo Code

```
 -1 * 0    00 // Action0
           00 // Feature 0 - trains
           00 // Number of properties to alter
           01 // Alter properties of one vehicle
           ** // Vehicle ID
           12 // Property 12 - sprites for vehicle
           FD // Value FD - use new sprites as defined by action3
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           02 // Two graphics sets
           04 // 4 sprites per graphics set
// 8 real sprites - two sets of 4 - set IDs 00 00 (empty) and 01 00 (full)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           02 // 2 loaded graphic sets
           02 // 2 loading graphic sets
           00 00 // Action0 sprite set for empty graphics (when moving)
           01 00 // Action0 sprite set for full graphics (when moving)
           00 00 // Action0 sprite set for empty graphics (when loading)
           01 00 // Action0 sprite set for full graphics (when loading)
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to
           00 // Number of cargo specific graphics sets 00 - none
           AA 00 // Default action2 ID
```

Notice how the two sets of real sprites from the action1 take IDs "00 00" and "01 00". Sprite sets in action1 cannot be assigned IDs, and are always numbered from zero going down the real sprites, then referenced as a word. Note additional properties can also be altered in the same action0, including speed, cargo capacity, etc.

But I want 3 stages of loading, then when the vehicle is moving show the wagon covered by a tarpaulin.

The code structure is the same as the above, but extra real sprite sets are required (one for half full and one for covered). The action2 can then be adapted to have only one loaded state (covered, for when the vehicle is moving), and three loading stages (empty, half full and full, for when it is loading in a station.)

Note: Action0 must be used to enable new graphics for the vehicle involved.

```
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           04 // Four graphics sets
           04 // 4 sprites per graphics set
// 16 real sprites - four sets of 4 - set IDs 00 00 (empty), 01 00 (half full), 02 00
(full) and 03 00 (covered)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 2 loaded graphic sets
           03 // 2 loading graphic sets
           03 00 // Action0 sprite set for covered graphics (when moving)
           00 00 // Action0 sprite set for empty graphics (when loading)
           01 00 // Action0 sprite set for half full graphics (when loading)
     &nbsp;     02 00 // Action0 sprite set for full graphics (when loading)
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to
           00 // Number of cargo specific graphics sets 00 - none
           AA 00 // Default action2 ID
```

I want a wagon which uses custom graphics, with 2 cargoes (coal and iron ore) both with different graphics.

This is a basic example of how to use an action3 to give a vehicle which can carry multiple cargoes cargo specific graphics according to which cargo it is currently carrying.

Nfo "translation"

There is a vehicle with cargo specific graphics (the wagon), it uses one set of graphics for coal and one for iron ore.
These are the graphics for the wagon when carrying coal, there are different sets depending on how full it is.
These are the graphics for the wagon when carrying iron ore, there are different sets depending on how full it is.
These are the real sprites to use.

This corresponds to:

- Action3
- Action2
- Action2

- Action1/Real sprites

However, because nfo code can only refer to an Action2 that has been previously defined, this logical chain must go in reverse.

Nfo Code

Note: Action0 must be used to enable new graphics for the vehicle involved.

```
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           03 // Two graphics sets
           04 // 4 sprites per graphics set
// 12 real sprites - three sets of 4 - set IDs 01 00 (empty), 01 00 (full [coal]) and
02 00 (full [iron ore])
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           02 // 2 loaded graphic sets
           02 // 2 loading graphic sets
           00 00 // Action0 sprite set for empty graphics (when moving)
           01 00 // Action0 sprite set for full [coal] graphics (when moving)
           00 00 // Action0 sprite set for empty graphics (when loading)
           01 00 // Action0 sprite set for full [coal] graphics (when loading)
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           BB // Action2 ID "BB"
           02 // 2 loaded graphic sets
           02 // 2 loading graphic sets
           00 00 // Action0 sprite set for empty graphics (when moving)
           02 00 // Action0 sprite set for full [iron ore] graphics (when moving)
           00 00 // Action0 sprite set for empty graphics (when loading)
           02 00 // Action0 sprite set for full [iron ore] graphics (when loading)
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to
           01 // Number of cargo specific graphics sets 01 - one
           01 // Cargo type 01 - coal
           AA 00 // Action2 ID for cargo specific graphic set one (coal graphics)
           BB 00 // Default action2 ID (iron ore graphics)
```

The vehicle can be refitted to any bulk cargo, how do I give some generic graphics when the wagon is not carrying coal or iron ore?

Using generic graphics (a covered wagon) for the default action2 ID in action3 lets the wagon be refitted to any bulk cargo and carry it without looking wrong. There can still be specific graphics for when carrying coal or iron ore.

```
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           04 // Two graphics sets
           04 // 4 sprites per graphics set
// 16 real sprites - four sets of 4 - set IDs 01 00 (empty), 01 00 (full [coal]), 02 00
(full [iron ore]) and 03 00 (covered)
// ...
// ...
```

```
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           02 // 2 loaded graphic sets
           02 // 2 loading graphic sets
           00 00 // Action0 sprite set for empty graphics (when moving)
           01 00 // Action0 sprite set for full [coal] graphics (when moving)
           00 00 // Action0 sprite set for empty graphics (when loading)
           01 00 // Action0 sprite set for full [coal] graphics (when loading)
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           BB // Action2 ID "BB"
           02 // 2 loaded graphic sets
           02 // 2 loading graphic sets
           00 00 // Action0 sprite set for empty graphics (when moving)
           02 00 // Action0 sprite set for full [iron ore] graphics (when moving)
           00 00 // Action0 sprite set for empty graphics (when loading)
           02 00 // Action0 sprite set for full [iron ore] graphics (when loading)
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           BB // Action2 ID "BB"
           02 // 2 loaded graphic sets
           02 // 2 loading graphic sets
           00 00 // Action0 sprite set for empty graphics (when moving)
           03 00 // Action0 sprite set for covered graphics (when moving)
           00 00 // Action0 sprite set for empty graphics (when loading)
           03 00 // Action0 sprite set for covered graphics (when loading)
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to
           02 // Number of cargo specific graphics sets 02 - two
           01 // Cargo type 01 - coal
           AA 00 // Action2 ID for cargo specific graphic set one (coal graphics)
           08 // Cargo type 08 - iron ore
           BB 00 // Action2 ID for cargo specific graphic set one (iron ore graphics)
           CC 00 // Default action2 ID, ie. when carrying anything other than coal or
iron ore (covered graphics)
```

I want the vehicle to appear differently in the buy menu.

A basic action2 and action3 example to demonstrate how to use cargo ID FF to make different graphics appear in the buy menu.

Nfo "translation"

There is a vehicle non-cargo specific graphics, it uses one set of graphics for use in-game and one for in the buy menu.
These are the graphics for the train when in the buy menu.
These are the graphics for the wagon when in-game.
These are the real sprites to use.

This corresponds to:

- Action3
- Action2
- Action2
- Action1/Real sprites

However, because nfo code can only refer to an Action2 that has been previously defined, this logical chain must go in reverse.

Note: Action0 must be used to enable new graphics for the vehicle involved.

Nfo Code

```
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           02 // Two graphics sets
           08 // 8 sprites per graphics set
// 16 real sprites - two sets of 8 - set ID 00 00 (normal graphics) and 01 00 (buy menu
graphics)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for normal graphics
           00 00 // Action0 sprite set for normal graphics
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           BB // Action2 ID "BB"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           01 00 // Action0 sprite set for buy menu graphics
           01 00 // Action0 sprite set for buy menu graphics
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to
           01 // Number of cargo specific graphics sets 01 - one
           FF // cargo type FF - special cargo type for buy menu graphics
           BB 00 //  Action2 ID for cargo specific graphic set one (buy menu graphics)
           AA 00 // Default action2 ID
```

Variational Action2s

Variational action2s apply different graphics to vehicles or objects according to a logical test against a property of the vehicle or game.

I want to have a train passenger carriage that gets dirty/rusted as it gets older.

This is a basic example of a variational action2. It uses a single variational action2 to check to see if the age of the vehicle is over a critical value, and changes the graphics accordingly.

Nfo "translation"

- There is a vehicle with no cargo specific graphics (the train carriage).
- When younger than 1000 days it should use "clean" graphics, if not younger than 1000 days it should use "dirty" graphics.
- These are the graphics for the clean carriage.
- These are the graphics for the dirty carriage.
- These are the real sprites to use.

This corresponds to:

- Action3
- Variational Action2
- Action2
- Action2
- Action1/Real sprites

However, because nfo code can only refer to an Action2 that has been previously defined, this logical chain must go in reverse.

Nfo Code

Note: Action0 must be used to enable new graphics for the vehicle involved.

```
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           02 // Two graphics sets
           04 // 4 sprites per graphics set
// 8 real sprites - two sets of 4 - set IDs 00 00 (dirty) and 01 00 (clean)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for dirty graphics
           00 00 // Action0 sprite set for dirty graphics
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           BB // Action2 ID "BB"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           01 00 // Action0 sprite set for clean graphics
           01 00 // Action0 sprite set for clean graphics
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
```

```
          CC // Action2 ID "CC"
          85 // Variable type - Query value for current object
             //                 Return a word
          C0 // Variable C0 - vehicle age in days
          00 // Bit shift 00 - do not shift variable
          FF FF // AND mask FF FF - use all bits
          01 // Check variable against one range
          BB 00 // Action2 ID to use if result is in the first range
            00 00 // Range 1 lower bound 00 00 - 0 days
            E8 03 // Range 1 upper bound E8 03 - 1000 days
          AA 00 // Action2 ID to use if result is not in any of the ranges
 -1 * 0   03 // Action3
          00 // Feature 0 - Trains
          01 // Apply graphics to one vehicle
          ** // Vehicle ID to apply graphics to
          00 // Number of cargo specific graphics sets 00 - none
          CC 00 // Default action2 ID
```

Whats all the bit shifting and and masking in the variational action2 about?

The value of the variable "40" (vehicle age) returns a word which is the vehicle age in days

```
    BB BB
```

A typical value of this may be:

```
    A3 32
```

Variable type is 85, so the single lowest word from the variable (ie. the whole variable) is considered.

1. Shift the bits:

```
   |      |
    BB BB
```

The word "BB BB" must line up with the single word "window" we are looking at the variable with. Because the variable is also a word we do not have to shift the bytes to align the variable with the window. Therefore the bit shift is 00.

2. AND mask
An AND mask is used to remove unwanted bits from the variable. We are interested in all the bits of the word, so do not want to remove any bits. An AND mask works by only taking bits from the variable which match up with a one from the AND mask, the others are set to zero. Therefore we must use an and mask of 11111111 11111111 (FF FF hexadecimal):

```
    FF FF   11111111 11111111    AND mask
    A3 32   10100011 00110010    Variable value from the single word window
    A3 32   10100011 00110010    Result from AND masking
```

Actually, I want the graphics to change according to the age of the engine pulling the carriage.

The example above checks variable "C0" (the vehicle age) for variable type "85" (the vehicle itself, return a word value). To check the age of the vehicle pulling the wagon the variable type needs to be "86". This checks the "related vehicle"'s age - the head of the consist. The variational action2 would then need to be:

```
 -1 * 0   02 // Action2
          00 // Feature 0 - trains
          CC // Action2 ID "CC"
          86 // Variable type - Query value for the related object
             //                 Return a word
          C0 // Variable C0 - vehicle age in days
          00 // Bit shift 00 - do not shift variable
          FF FF // AND mask FF FF - use all bits
          01 // Check variable against one range
          BB 00 // Action2 ID to use if result is in the first range
            00 00 // Range 1 lower bound 00 00 - 0 days
            E8 03 // Range 1 upper bound E8 03 - 1000 days
          AA 00 // Action2 ID to use if result is not in any of the ranges
```

I want a tilting carriage for my tilting train.

This is a more complex example of a variational action2. It uses a single variational action2 to check to see if the curvature of the vehicle is zero, is "positive" or is "negative", and changes the graphics accordingly.

Nfo "translation"

- There is a vehicle with no cargo specific graphics (the train carriage).
- When the vehicle is curved to the left the "left tilt" graphics should be used, when curved to the right the "right tilt" graphics should be used, otherwise the "untilted" graphics should be used.
- These are the graphics for the "left tilt" carriage.
- These are the graphics for the "right tilt" carriage.
- These are the graphics for the "untiltied" carriage.
- These are the real sprites to use.

This corresponds to:

- Action3
- Variational Action2
- Action2
- Action2
- Action2
- Action1/Real sprites

However, because nfo code can only refer to an Action2 that has been previously defined, this logical chain must go in reverse.

Nfo Code

Note: Action0 must be used to enable new graphics for the vehicle involved.

```
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           02 // Two graphics sets
           08 // 8 sprites per graphics set
// 24 real sprites - three sets of 8 - set IDs 00 00 (untilted), 01 00 (right tilt) and
02 00 (left tilt)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for untilted graphics
           00 00 // Action0 sprite set for untilted graphics
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           BB // Action2 ID "BB"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
```

```
          01 00 // Action0 sprite set for right tilt graphics
          01 00 // Action0 sprite set for right tilt graphics
 -1 * 0   02 // Action2
          00 // Feature 0 - trains
          CC // Action2 ID "CC"
          01 // 1 loaded graphic set
          01 // 1 loading graphic set
          02 00 // Action0 sprite set for left tilt graphics
          02 00 // Action0 sprite set for left tilt graphics
 -1 * 0   02 // Action2
          00 // Feature 0 - trains
          DD // Action2 ID "DD"
          83 // Variable type - Query value for current object
             //                 Return a byte
          45 // Variable 45 - vehicle curvature
          10 // Shift variable value 10 bits - shift variable 2 bytes to the right
          0F // AND mask 0F - only use last 4 bits of the value
          02 // Check variable against two ranges
          BB 00 // Action2 ID to use if result is in the first range
            01 // Range 1 lower bound 01 - right curve 45 degrees
            04 // Range 1 upper bound 04 - right curve 180 degrees
          CC 00 // Action2 ID to use if result is in the second range
            0C // Range 1 lower bound 00 00 - left curve 180 degrees
            0F // Range 1 upper bound E8 03 - right curve 45 degrees
          AA 00 // Action2 ID to use if result is not in any of the ranges
 -1 * 0   03 // Action3
          00 // Feature 0 - Trains
          01 // Apply graphics to one vehicle
          ** // Vehicle ID to apply graphics to
          00 // Number of cargo specific graphics sets 00 - none
          DD 00 // Default action2 ID
```

Whats all the bit shifting and and masking in the variational action2 about?

The value of the variable "45" (vehicle curvature) does not return a single number, it returns three in the form of a dword:

```
    XX XT XB XF
```

A typical value of this may be:

```
    A3 43 51 D2
```

T, B and F give different numbers which carry information about the vehicle curvature. X may have any number - it is "junk". In this case the value of T is what we are interested in.
Variable type is 81, so only the single lowest byte from the variable is considered. The value of the variable may have to be recalculated to get the desired value in that single byte.

To analyse T we need to:

1. Shift the bits:

```
            |  |
    XX XT XB XF
 ---->    XX XT XB XF
```

The byte "XT" must line up with the single byte "window" we are looking at the variable with. The variable must be shifted 2 bytes, 16 bits (10 bits in hexadecimal) to the right. Therefore the bit shift is 10.

2. Get rid of the junk "X"

```
    XT ----> 0T
```

An AND mask is used to remove junk from bits in the byte. We are only interested in the lowest four bits - the range 00 to 0F (0 to 15 decimal). An AND mask works by only taking bits from the variable which match up with a one from the AND mask, the others are set to zero. Therefore we must use an and mask of 00001111 (0F hexadecimal):

```
    0F    00001111    AND mask
    B3    10110011    Variable value from the single byte window
    03    00000011    Result from AND masking
```

The result of the bit shift and AND mask is the variable now takes the value 0T, not XX XT XB XF, which is much easier to work with.

Livery Override Action3

Livery overrides provide a simple way to force a vehicle (for example a passenger car) to use different graphics when pulled by a particular engine.

I want my new double headed train graphics to have matching passenger cars.

This is a basic example of a livery override action3, which changes the graphics which appear for a passenger car when pulled by a particular engine.

Nfo "translation"

- There is a vehicle with no cargo specific graphics (the train engine).
- There is another vehicle which should change its graphics to these non-cargo specific graphics when pulled by a particular engine.
- These are the graphics for the carriage.
- These are the graphics for the engine.
- These are the real sprites to use.

This corresponds to:

- Action3
- Livery Override Action3
- Action2
- Action2
- Action1/Real sprites

However, because nfo code can only refer to an Action2 that has been previously defined, this logical chain must go in reverse, with the exception of the livery override action3 which must follow the normal action3 for the train engine.

Nfo Code

Note: Action0 must be used to enable new graphics for the vehicles involved.

```
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           02 // Two graphics sets
           08 // 8 sprites per graphics set
// 16 real sprites - two sets of 8 - set IDs 00 00 (engine) and 01 00 (carriage)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for engine graphics
           00 00 // Action0 sprite set for engine graphics
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           BB // Action2 ID "BB"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           01 00 // Action0 sprite set for carriage graphics
           01 00 // Action0 sprite set for carriage graphics
-1 * 0     03 // Action3
```

```
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to
           00 // Number of cargo specific graphics sets 00 - none
           AA 00 // Default action2 ID
-1 * 0     03 // Action3
           00 // Feature 0 - Trains
           81 // Apply graphics to one vehicle, bit 80 set so graphics override for
when attached to the vehicle mentioned in the previous action3
           ** // Vehicle ID to apply graphics to
           00 // Number of cargo specific graphics sets 00 - none
           BB 00 // Default action2 ID
```

But I want to have different graphics for if the passenger is immediately behind or in front of the double headed engine.

Action3 livery overrides can target variational action2s. This means they livery override can be aware of the position of the carriage in the consist, and change the graphics appropriately appropriately.

```
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           04 // Four graphics sets
           08 // 8 sprites per graphics set
// 32 real sprites - four sets of 8 - set IDs 00 00 (engine), 01 00 (normal carriage),
02 00 (front carriage) and 01 00 (rear carriage)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for engine graphics
           00 00 // Action0 sprite set for engine graphics
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           BB // Action2 ID "BB"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           01 00 // Action0 sprite set for normal carriage graphics
           01 00 // Action0 sprite set for normal carriage graphics
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           CC // Action2 ID "CC"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           02 00 // Action0 sprite set for front carriage graphics
           02 00 // Action0 sprite set for front carriage graphics
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           DD // Action2 ID "DD"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           03 00 // Action0 sprite set for rear carriage graphics
           03 00 // Action0 sprite set for rear carriage graphics
-1 * 0     02 // Action2
           00 // Feature 0 - trains
           EE // Action2 ID "EE"
           81 // Variable type - Query value for current object
```

```
              //                 Return a byte
           40 // variable 40 - position in and length of consist
           00 // Shift variable value 0 - do not shift variable
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           CC 00 // Action2 ID to use if result is in the first range
             01 // Range 1 lower bound 01 - vehicle is foremost excluding double headed
engine
             01 // Range 1 upper bound 01 - vehicle is foremost excluding double headed
engine
           BB 00 // Action2 ID to use if result is not in any of the ranges
-1 * 0     02 // Action2
           00 // Feature 0 - trains
           FF // Action2 ID "FF"
           81 // Variable type - Query value for current object
              //                 Return a byte
           40 // variable 40 - position in and length of consist
           08 // Shift variable value 8 - shift variable 1 byte to the right
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           DD 00 // Action2 ID to use if result is in the first range
             01 // Range 1 lower bound 01 - vehicle is rearmost excluding double headed
engine
             01 // Range 1 upper bound 01 - vehicle is rearmost excluding double headed
engine
           EE 00 // Action2 ID to use if result is not in any of the range
-1 * 0     03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to (train engine ID)
           00 // Number of cargo specific graphics sets 00 - none
           AA 00 // Default action2 ID
-1 * 0     03 // Action3
           00 // Feature 0 - Trains
           81 // Apply graphics to one vehicle, bit 80 set so graphics override for
when attached to the vehicle mentioned in the previous action3
           ** // Vehicle ID to apply graphics to (carriage ID)
           00 // Number of cargo specific graphics sets 00 - none
           FF 00 // Default action2 ID
```

Whats all the bit shifting and and masking in the variational action2 about?

Variable 40 does not return a single number, it returns 3 in the form of a dword:

```
    00 NN BB FF
```

A typical value may be:

```
    00 06 02 03
```

Where NN is the number of vehicles in the consist (counting from zero), BB is the position in the consist (counting from zero) from the rear, and FF is the position in consist (counting from zero) from the front. In the example there are 7 vehicles in the consist, and the one in question lies 3 from the rear (4 from the front).
Variable type is 81, so only the single lowest byte from the variable is considered. The value of the variable may have to be recalculated to get the desired value in that single byte.

To check FF:
Want to look at the single lowest byte:

1. Do not shift bits - FF is in line with the single byte "window"

```
            |  |
    00 NN BB FF
```

2. AND mask with 11111111 (FF), as all bytes want to be accessed.

The result of the bit shift and AND mask is the variable now takes the value FF, not 00 NN BB FF, which is much easier to work with.

To check BB:
Want to look at the the single second lowest byte:

1. Shift bits 8 (1 byte) to the right to align BB with the single byte "window"

```
         |  |
 00 NN BB FF
 -> 00 NN BB FF
```

2. AND mask with 11111111 (FF), as all bytes want to be accessed.

The result of the bit shift and AND mask is the variable now takes the value BB, not 00 NN BB FF, which is much easier to work with.

Callback Action2s

Callbacks allow the modification of vehicle properties, like in an action0, but with the control of logical tests possible with variational action2s.

I want to make a steam train with a tender.

Callback to build an articulated vehicle. Adds a vehicle of different vehicle ID (typically with capacity, power, etc. of zero), for use as a tender or dummy trailer.

Nfo "translation"

- There is a vehicle with no cargo specific graphics (the tender).
- These are the graphics to use for the tender.
- These are the real sprites to use.
- There is a vehicle with no cargo specific graphics (the train engine).
- The train engine has a callback which makes it into an articulated vehicle.
- The callback should add the tender to follow the engine.
- These are the graphics to use for the engine.
- These are the real sprites to use.

This corresponds to:

- Action3 (tender)
- Action2 (tender)
- Action1/Real sprites (tender)
- Action3 (engine)
- Variational action2 (check engine for callbacks)
- Callback action2 (add vehicles for articulation)
- Action2 (engine)
- Action1/Real sprites (engine)

However, because nfo code can only refer to an Action2 that has been previously defined, this logical chain must go in reverse.

Nfo Code

Note: Action0 must also be used to set the other properties for the engine and tender, weight (for tender and engine), and power, speed, cargo capacity, etc. (for engine).

```
 -1 * 0    00 // Action0
           00 // Feature 0 - trains
           02 // Number of properties to alter
           01 // Alter properties of one vehicle
           $$ // Vehicle ID (engine)
           12 // Property 12 - sprites for vehicle
           FD // Value FD - use new sprites as defined by action3
           1E // Property 1E - callbacks bitmask
           10 // Value 10 - bit 4 set, so use articulated vehicle callback for this
vehicle
 -1 * 0    00 // Action0
           00 // Feature 0 - trains
           03 // Number of properties to alter
           01 // Alter properties of one vehicle
           ** // Vehicle ID (tender)
           12 // Property 12 - sprites for vehicle
           FD // Value FD - use new sprites as defined by action3
           14 // Property 14 - power
           00 // Value 00 - power of zero, therefore a wagon
           0B // Property 0B - cargo capacity
           00 // Value 00 - capacity of zero

 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           01 // One graphics sets
           08 // 8 sprites per graphics set
// 8 real sprites - one sets of 8 - set ID 00 00 (engine)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for engine graphics
           00 00 // Action0 sprite set for engine graphics
 -1 * 0    02 // Action2 (sets callback properties for callback ID 16)
           00 // Feature 0 - Trains
           BB // Action2 ID "BB"
           81 // Variable type - Query value for current object
              //                 Return a byte
           10 // Variable 10 - extra callback information (number of "trailers" added
to articulated vehicle so far)
           00 // Shift variable value 0 - no shift
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           ** 80 // Bit 15 set, so not action2 ID but callback result. "**" - vehicle
ID to add as trailer to articulated vehicle
             01 // Range 1 lower bound 1 - number of "trailers" added to articulated
vehicle so far is one
             01 // Range 1 upper bound 1 - number of "trailers" added to articulated
vehicle so far is one
           FF FF // Bit 15 set, so not action2 ID set but callback result. Value "FF
FF" so termination of callback chain.
 -1 * 0    02 // Action2 (checks for callbacks)
           00 // Feature 0 - Trains
           CC // Action2 ID "CC"
           81 // Variable type - Query value for current object
              //                 Return a byte
           0C // Variable 0C - current callback ID
           00 // Shift variable value 0 - no shift
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           BB 00 // Action2 ID to use if result is in the first range (go to callback
action2)
             16 // Range 1 lower bound 16 - callback ID is 16
             16 // Range 1 upper bound 16 - callback ID is 16
           AA 00 // Action2 ID to use if result is not in any of the ranges (go to
graphics action2)
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           $$ // Vehicle ID to apply graphics to (engine)
           00 // Number of cargo specific graphics sets 00 - none
           CC 00 // Default action2 ID

 -1 * 0    01 // Action1
```

```
           00 // Feature 0 - trains
           01 // One graphics sets
           08 // 8 sprites per graphics set
// 8 real sprites - one sets of 8 - set ID 00 00 (tender)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for tender graphics
           00 00 // Action0 sprite set for tender graphics
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to (tender)
           00 // Number of cargo specific graphics sets 00 - none
           AA 00 // Default action2 ID
```

Ok, run that whole variational action2/callback action2 past me again!

Notice how the "check for callbacks" action2 is just a variational action2 which checks the value of variable 0C - the callback ID. It then points to action2 ID BB 00 (to define the properties of the callback) for callback ID 16 (articulated vehicles), and action2 ID AA 00 for all other callback values. Multiple callbacks IDs can be checked for at this point.

The "set callback properties" action2 is also just a variational action2 which checks the value of a variable (in this case variable 10 - properties of the current callback). Instead of pointing to action2 IDs (words with bits 9 to 15 always set to 0) it gives a callback result, indicated by having bit 15 set to 1. These give a result which is interpreted in relation to the current callback ID (in this case the vehicle ID to add to the articulated vehicle). FF FF is used to indicate the callback has done all it needs to do, so to stop. This must always be the default callback result pointed to in the last variational action2 in a callback chain.

The easiest way to explain what is happening is by some pseudo-basic programing language. Logic is shown in CAPITALS, variables in *italics*, [tests in square brackets] and actions in **bold**. This code is written in reverse order to the actual nfo code, to make the logic make a bit more sense.

```
ID   Action         Logic
// Vehicle ID **
--   Action3        GOTO AA
CC   Action2        apply tender graphics

// Vehicle ID $$
--   Action3        GOTO CC
CC   Var Action2    IF [callback ID = 16] GOTO BB
                    ELSEIF GOTO AA
BB   CB Action2     IF [number of vehicles in consist = 1] add vehicle with ID **
                    ELSEIF STOP
AA   Action2        apply engine graphics
```

The tender doesn't appear in the buy menu, how do I make it appear there?

Only the first vehicle in articulated vehicle appears in the buy menu. To make the tender appear you have to make an additional set of real sprites for the buy menu (only one horizontal view is required), give it an appropriate action2, and use those graphics for cargo type "FF" - buy menu graphics.

Note action0s are required as before to set the engine and tender properties, but these are the same as in the previous example.

```
 -1 * 0    01 // Action1
           00 // Feature 0 - trains
           02 // One graphics sets
           08 // 8 sprites per graphics set
// 16 real sprites - two sets of 8 - set IDs 00 00 (engine) and 01 00 (buy menu engine
and tender)
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for engine graphics
           00 00 // Action0 sprite set for engine graphics
 -1 * 0    02 // Action2 (sets callback properties for callback ID 16)
           00 // Feature 0 - Trains
           BB // Action2 ID "BB"
           81 // Variable type - Query value for current object
              //                 Return a byte
           10 // Variable 10 - extra callback information (number of "trailers" added
to articulated vehicle so far)
           00 // Shift variable value 0 - no shift
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           ** 80 // Bit 15 set, so not action2 ID but callback result. "**" - vehicle
ID to add as trailer to articulated vehicle
             01 // Range 1 lower bound 1 - number of "trailers" added to articulated
vehicle so far is one
             01 // Range 1 upper bound 1 - number of "trailers" added to articulated
vehicle so far is one
           FF FF // Bit 15 set, so not action2 ID set but callback result. Value "FF
FF" so termination of callback chain.
 -1 * 0    02 // Action2 (checks for callbacks)
           00 // Feature 0 - Trains
           CC // Action2 ID "CC"
           81 // Variable type - Query value for current object
              //                 Return a byte
           0C // Variable 0C - current callback ID
           00 // Shift variable value 0 - no shift
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           BB 00 // Action2 ID to use if result is in the first range (go to callback
action2)
             16 // Range 1 lower bound 16 - callback ID is 16
             16 // Range 1 upper bound 16 - callback ID is 16
           AA 00 // Action2 ID to use if result is not in any of the ranges (go to
graphics action2)
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           DD // Action2 ID "DD"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for buy menu engine and tender graphics
           00 00 // Action0 sprite set for buy menu engine and tender graphics
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           $$ // Vehicle ID to apply graphics to (engine)
           00 // Number of cargo specific graphics sets 01 - one
           FF // cargo type FF - special cargo type for buy menu graphics
           DD 00 //  Action2 ID for cargo specific graphic set one (buy menu graphics)

           CC 00 // Default action2 ID
```

Note the action1, action2 and action3 are still required for the tender, but these are the same as in the previous example.

The articulated vehicle callback is not used for the buy menu graphics, so only the properties of the engine will be shown. The additional weight due to the tender, etc. will not be shown.

This is how the logic has changed:

```
ID   Action        Logic
// Vehicle ID $$
--   Action3       IF [cargo ID = FF] GOTO DD
                   ELSEIF GOTO CC
DD   Action2       apply buy menu graphics
CC   Var Action2   IF [callback ID = 16] GOTO BB
                   ELSEIF GOTO AA
BB   CB Action2    IF [number of vehicles in consist = 1] add vehicle with ID **
                   ELSEIF STOP
AA   Action2       apply engine graphics
```

But I don't want the tender to appear if a train's major cargo is coal.

The "set callback properties" action2 does not have to point to callback results (bit 15 set to 1), but can also point to other action2s, so long as the eventual outcome is always a callback result. The variational action2s need to be altered and an additional "set callback properties" action2 needs to be added.

```
 -1 * 0    02 // Action2 (sets callback properties for callback ID 16)
           00 // Feature 0 - Trains
           BB // Action2 ID "BB"
           81 // Variable type - Query value for current object
              //         &nbsp;       Return a byte
           10 // Variable 10 - extra callback information (number of "trailers" added
to articulated vehicle so far)
           00 // Shift variable value 0 - no shift
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           ** 80 // Bit 15 set, so not action2 ID but callback result. "**" - vehicle
ID to add as trailer to articulated vehicle
             01 // Range 1 lower bound 1 - number of "trailers" added to articulated
vehicle so far is one
             01 // Range 1 upper bound 1 - number of "trailers" added to articulated
vehicle so far is one
           FF FF // Bit 15 set, so not action2 ID set but callback result. Value "FF
FF" so termination of callback chain.
-1 * 0     02 // Action2
           00 // Feature 0 - trains
           CC // Action2 ID "CC"
           81 // Variable type - Query value for current object
              //                Return a byte
           42 // variable 42 - cargoes transported by consist
           08 // Shift variable value 8 - shift variable 1 byte to the right
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           FF FF // Action2 ID to use if result is in the first range
             01 // Range 1 lower bound 01 - vehicles major cargo is coal
             01 // Range 1 upper bound 01 - vehicles major cargo is coal
           BB 00 // Action2 ID to use if result is not in any of the range
 -1 * 0    02 // Action2 (checks for callbacks)
           00 // Feature 0 - Trains
           DD // Action2 ID "DD"
           81 // Variable type - Query value for current object
              //                 Return a byte
           0C // Variable 0C - current callback ID
           00 // Shift variable value 0 - no shift
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           CC 00 // Action2 ID to use if result is in the first range (go to callback
action2)
             16 // Range 1 lower bound 16 - callback ID is 16
             16 // Range 1 upper bound 16 - callback ID is 16
           AA 00 // Action2 ID to use if result is not in any of the ranges (go to
graphics action2)
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           $$ // Vehicle ID to apply graphics to (engine)
           00 // Number of cargo specific graphics sets 00 - none
           CC 00 // Default action2 ID
```

Ok, run that whole variational action2/callback action2 past me again!

There are two "set callback properties" action2s, the later one (ID "CC") first checks to see if the trains major cargo is coal. If it is coal then it terminates the callback chain with FF FF - no articulated vehicle is added. If it is not coal then the callback chain is sent to earlier "set callback properties" action2 (ID "BB") which adds vehicles until the articulated vehicle length is 2.

```
ID   Action         Logic
// Vehicle ID **
--   Action3        GOTO AA
CC   Action2        apply tender graphics

// Vehicle ID $$
--   Action3        GOTO DD
DD   Var Action2    IF [callback ID = 16] GOTO CC
                    ELSEIF GOTO AA
CC   CB Action2     IF [major cargo type = coal] STOP
                    ELSEIF GOTO BB
BB   CB Action2     IF [number of vehicles in consist = 1] add vehicle with ID **
                    ELSEIF STOP
AA   Action2        apply engine graphics
```

I want to make an articulated road vehicle (a tram), where each trailer carries cargo.

Callback to build an articulated vehicle. Adds a vehicle of the same vehicle ID (so with the same capacity, power, etc. as the head of the consist), typically used for DMUs/EMUs and trams. Includes altering graphics according to position in consist.

Nfo "translation"

- There is a tram with no cargo specific graphics.
- The tram has a callback which makes it into an articulated vehicle.
- The callback should add another two copies of the tram to follow as powered and cargo carrying trailers.
- The tram should take different graphics according to where it is in the consist.
- These are the graphics to use for the front tram car.
- These are the graphics to use for the middle tram car.
- These are the graphics to use for the rear tram car.
- These are the real sprites to use.

This corresponds to:
- Action3
- Variational action2 (check for callbacks)
- Callback action2 (add vehicles for articulation)
- Variational action2 (change graphics according to consist position)
- Action2 (rear)
- Action2 (middle)
- Action2 (front)
- Action1/Real sprites

However, because nfo code can only refer to an Action2 that has been previously defined, this logical chain must go in reverse.

Nfo Code

Note: Action0 must also be used to set the other properties for the tram; weight, power, speed, cargo capacity, etc..

```
 -1 * 0    00 // Action0
           01 // Feature 0 - trains
           03 // Number of properties to alter
           01 // Alter properties of one vehicle
           ** // Vehicle ID
           0E // Property 12 - sprites for vehicle
           FD // Value FD - use new sprites as defined by action3
           17 // Property 1E - callbacks bitmask
           10 // Value 10 - bit 4 set, so use articulated vehicle callback for this
vehicle
           1C // Property 1C - miscellaneous flags
           01 // Value 01 - bit 0 set, so this vehicle is a tram

 -1 * 0    01 // Action1
           01 // Feature 0 - road vehicles
           03 // Three graphics sets
           08 // 8 sprites per graphics set
// 24 real sprites - three sets of 8 - set IDs 00 00 (front), 01 00 (middle) and 02 00
(rear)
// ...
// ...
 -1 * 0    02 // Action2
           01 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for front
           00 00 // Action0 sprite set for front
 -1 * 0    02 // Action2
           01 // Feature 0 - trains
           BB // Action2 ID "BB"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           01 00 // Action0 sprite set for middle
           01 00 // Action0 sprite set for middle
 -1 * 0    02 // Action2
           01 // Feature 0 - trains
           CC // Action2 ID "CC"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           02 00 // Action0 sprite set for rear
           02 00 // Action0 sprite set for rear
-1 * 0     02 // Action2
    &nbsp;      01 // Feature 1 - Road vehicles
           DD // Action2 ID "DD"
           81 // Variable type - Query value for current object
              //                  Return a byte
           40 // variable 40 - position in and length of consist
           00 // Shift variable value 0 - do not shift variable
           FF // AND mask FF - use all bits
           02 // Check variable against two ranges
           AA 00 // Action2 ID to use if result is in the first range
             00 // Range 1 lower bound 00 - vehicle is first in consist
             00 // Range 1 upper bound 00 - vehicle is first in consist
           CC 00 // Action2 ID to use if result is in the second range
             02 // Range 2 lower bound 02 - vehicle is third in consist
             02 // Range 2 upper bound 02 - vehicle is third in consist
           BB 00 // Action2 ID to use if result is not in any of the ranges
 -1 * 0    02 // Action2 (sets callback properties for callback ID 16)
           01 // Feature 1 - Road vehicles
           EE // Action2 ID "EE"
           81 // Variable type - Query value for current object
              //                  Return a byte
           10 // Variable 10 - extra callback information (number of "trailers" added
to articulated vehicle so far)
           00 // Shift variable value 0 - no shift
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           ** 80 // Bit 15 set, so not action2 ID but callback result. "**" - vehicle
ID to add as a trailer to articulated vehicle
             01 // Range 1 lower bound 1 - number of "trailers" added to articulated
vehicle so far is one
             02 // Range 1 upper bound 2 - number of "trailers" added to articulated
vehicle so far is two
           FF FF // Bit 15 set, so not action2 ID set but callback result. Value "FF
FF" so termination of callback chain.
 -1 * 0    02 // Action2 (checks for callbacks)
           01 // Feature 1 - Road vehicles
           FF // Action2 ID "FF"
           81 // Variable type - Query value for current object
              //                  Return a byte
           0C // Variable 0C - current callback ID
           00 // Shift variable value 0 - no shift
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           BB 00 // Action2 ID to use if result is in the first range (go to callback
action2)
             16 // Range 1 lower bound 16 - callback ID is 16
             16 // Range 1 upper bound 16 - callback ID is 16
           DD 00 // Action2 ID to use if result is not in any of the ranges (go to
graphics action2)
 -1 * 0    03 // Action3
```

```
           01 // Feature 1 - Road vehicles
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to
           00 // Number of cargo specific graphics sets 00 - none
           CC 00 // Default action2 ID
```

Ok, run that whole variational action2/callback action2 past me again!

The easiest way to explain what is happening is by some pseudo-basic programing language. Logic is shown in CAPITALS, variables in *italics*, [tests in square brackets] and actions in **bold**. This code is written in reverse order to the actual nfo code, to make the logic make a bit more sense.

```
ID   Action         Logic
// Vehicle ID **
--   Action3        GOTO FF
FF   Var Action2    IF [callback ID = 16] GOTO EE
                    ELSEIF GOTO DD
EE   CB Action2     IF [number of vehicles in consist = 1] add vehicle with ID **
                    ELSEIF STOP
DD   Var Action2    IF [vehicle position = 1] GOTO AA
                    IF [vehicle position = 3] GOTO CC
                    ELSEIF GOTO BB
CC   Action2        apply last vehicle graphics
BB   Action2        apply middle vehicle graphics
AA   Action2        apply first vehicle graphics
```

Actually I want 4 vehicles in the articulated consist...

To change the number of vehicles the callback properties and variational action2 for graphics need to be changed. Altering the callback properties adds the extra vehicle, and altering the variational action2 corrects the graphics:

```
-1 * 0     02 // Action2
           01 // Feature 1 - Road vehicles
           DD // Action2 ID "DD"
           81 // Variable type - Query value for current object
              //                 Return a byte
           40 // variable 40 - position in and length of consist
           00 // Shift variable value 0 - do not shift variable
           FF // AND mask FF - use all bits
           02 // Check variable against two ranges
           AA 00 // Action2 ID to use if result is in the first range
             00 // Range 1 lower bound 00 - vehicle is first in consist
             00 // Range 1 upper bound 00 - vehicle is first in consist
           CC 00 // Action2 ID to use if result is in the second range
             03 // Range 2 lower bound 03 - vehicle is fourth in consist
             03 // Range 2 upper bound 03 - vehicle is fourth in consist
           BB 00 // Action2 ID to use if result is not in any of the ranges
 -1 * 0    02 // Action2 (sets callback properties for callback ID 16)
           01 // Feature 1 - Road vehicles
           EE // Action2 ID "EE"
           81 // Variable type - Query value for current object
              //                 Return a byte
           10 // Variable 10 - extra callback information (number of "trailers" added
to articulated vehicle so far)
           00 // Shift variable value 0 - no shift
           FF // AND mask FF - use all bits
           01 // Check variable against one range
           ** 80 // Bit 15 set, so not action2 ID but callback result. "**" - vehicle
ID to add as a trailer to articulated vehicle
             01 // Range 1 lower bound 1 - number of "trailers" added to articulated
vehicle so far is one
             03 // Range 1 upper bound 3 - number of "trailers" added to articulated
vehicle so far is three
           FF FF // Bit 15 set, so not action2 ID set but callback result. Value "FF
FF" so termination of callback chain.
```

And this is how the logic changed:

```
ID   Action         Logic
// Vehicle ID **
--   Action3        GOTO FF
FF   Var Action2    IF [callback ID = 16] GOTO EE
                    ELSEIF GOTO DD
EE   CB Action2     IF [1 </= number of vehicles in consist </= 2] add vehicle with ID
** REPEAT
                    ELSEIF STOP
DD   Var Action2    IF [vehicle position = 1] GOTO AA
                    IF [vehicle position = 4] GOTO CC
                    ELSEIF GOTO BB
CC   Action2        apply last vehicle graphics
BB   Action2        apply middle vehicle graphics
AA   Action2        apply first vehicle graphics
```

Players should only be able to pull passenger and mail carriages with this engine.

Callback to allow addition of only select vehicle IDs to the consist. Includes optional text in buy window callback too.

{{not finished}}

I want the passenger carriages added to my train to be 3/4 the length of normal passenger carriages.

Livery override driven callback for vehicle length.

{{not finished}}

Randomized Action2s

I want the box cars added to any train to be random colours.

This is a basic example of how to use a randomized action2 to give a vehicle random non-cargo specific graphics.

Nfo "translation"

There is a vehicle with no cargo specific graphics (the box car).
There are three different graphics sets to choose between.
These are the graphics for the first colour scheme.
These are the graphics for the second colour scheme.
These are the graphics for the third colour scheme.
These are the real sprites to use.

This corresponds to:

- Action3
- Randomized Action2
- Action2
- Action2
- Action2
- Action1/Real sprites

However, because nfo code can only refer to an Action2 that has been previously defined, this logical chain must go in reverse.

Nfo Code

Note: Action0 must be used to enable new graphics for the vehicle involved.

```
-1 * 0   01 // Action1
           00 // Feature 0 - trains
           04 // Three graphics sets
           08 // 8 sprites per graphics set
// 36 real sprites - four sets of 8 - set ID 00 00 (colour scheme 1), 01 00 (colour
scheme 2), 02 00 (colour scheme 3) and 03 00 (colour scheme 4).
// ...
// ...
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           AA // Action2 ID "AA"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           00 00 // Action0 sprite set for colour scheme 1
           00 00 // Action0 sprite set for colour scheme 1
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           BB // Action2 ID "BB"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           01 00 // Action0 sprite set for colour scheme 2
           01 00 // Action0 sprite set for colour scheme 2
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           CC // Action2 ID "CC"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           02 00 // Action0 sprite set for colour scheme 3
           02 00 // Action0 sprite set for colour scheme 3
 -1 * 0    02 // Action2
           00 // Feature 0 - trains
           DD // Action2 ID "DD"
           01 // 1 loaded graphic set
           01 // 1 loading graphic set
           03 00 // Action0 sprite set for colour scheme 4
           03 00 // Action0 sprite set for colour scheme 4
 -1 * 0    02 // Action2
           00 // Feature 0 - Trains
           EE // Action2 ID "EE"
           80 // Randomise vehicle as an individual, not according to head of consist
           01 // Randomisation trigger bit mask - bit 0 set so randomised with new
cargo load
           00 // Randomise all bits
           04 // Number of sets to randomise from
             AA 00 // Randomised set 1
             BB 00 // Randomised set 2
             CC 00 // Randomised set 3
             DD 00 // Randomised set 4
 -1 * 0    03 // Action3
           00 // Feature 0 - Trains
           01 // Apply graphics to one vehicle
           ** // Vehicle ID to apply graphics to
           00 // Number of cargo specific graphics sets 00 - none
           EE 00 // Default action2 ID
```

Actually, I would like all the box cars in the consist to take the same colour scheme as each other, to simulate carrying a particular company's cargo.

Randomisation can either be based on the individual vehicles or the "related object" (the head of the consist), similar to variational action2s. By randomising according to the head of the consist all box cars in that consist will be the same colour. Only the randomised action2 needs altering:

```
 -1 * 0    02 // Action2
           00 // Feature 0 - Trains
           EE // Action2 ID "EE"
           83 // Randomise vehicle according to the head of the consist, not as an
individual
           01 // Randomisation trigger bit mask - bit 0 set so randomised with new
cargo load
           00 // Randomise all bits
           04 // Number of sets to randomise from
             AA 00 // Randomised set 1
             BB 00 // Randomised set 2
             CC 00 // Randomised set 3
             DD 00 // Randomised set 4
```

Action7/9s

I want to have my vehicles only available in certain climates.

Action0 and action7/9s required to use additional vehicles from other climates when in one climate, but return them to the defaults when in another climate.