FREIGHT COMMANDER

Disclaimer: This game, and by extension this manual, are early versions and are far from complete. Please view progress on this project's GitHub page at https://github.com/tbdye/FreightCommander

INTRODUCTION

The goal of Freight Commander is to simulate the railroad industry as depicted on model railroading layouts by dispatching and tracking freight rolling stock to and from modeled industries. Crew (players) receive dispatch orders as well as check in at various locations in via a smartphone application. Players are faced with a complex puzzle of how to move specific freight cars from the middle of a train into a railroad siding using only push and pull movements in a finite space, in a finite time. The frontend application is connected to a database tracking all train movements during a game session and issues dispatch orders to crew. This manual describes entities in use that allow for gameplay and further development and customization by individuals or groups.

LICENSE

The author of this program, Thomas Dye, may be reached through email at thomas@tfenet.com.

Table of Contents

INTRODUCTION 1

LICENSE 1

GAME OVERVIEW 3

PLAYER ROLES 4

SHIPPING ORDERS, WAYBILLS, AND TRAINS 4

FREIGHT CAR LIFECYCLE 4

ROLLING STOCK, INDUSTRIES, AND PRODUCTS 4

SERVICING INDUSTRIES 5

END OF GAME PARAMETERS 5

POPULATING A LAYOUT 5
SECTION A: INITIALIZE GAME PARAMETERS 5
SECTION B: BUILD LAYOUT 5
SECTION C: INITIALIZE GAME SESSION 5
SECTION D: GAME SESSION 6
GAMEPLAY 6
DATABASE DESIGN 6
DATABASE TABLES 7
LAYOUT 7
Modules 7
MainLines 7
Junctions 7
Industries 8
IndustryActivities 8
IndustryProducts 8
IndustrySidings 8
SidingAssignments 9
Yards 9
GAME 9
ModulesAvailable 9
Regions 9
IndustriesAvailable 10
RollingStockTypes 10
ProductTypes 10
PLAYER 10
RollingStockCars 10
RollingStockAtYards 11
RollingStockAtIndustries 11
Shipments 11
ShipmentsLoaded 11
ShipmentsUnloaded 12
Waybills 12
Trains 12

TrainLocations 12
ConsistedCars 13
TrainCrews 13
FUNCTIONS 13
ufnGetProductType 13
ufnGetProducingIndustry 13
ufnGetConsumingIndustry 13
ufnGetIndustrySiding 14
ufnGetCarModuleName 14
ufnCheckServiceableCarSiding 14
STORED PROCEDURES 14
uspAddTrain 14
uspRemoveTrain 14
uspModifyTrain 14
uspAddCarToGame 15
uspRemoveCarFromGame 15
uspAddCarToService 15
uspRemoveCarFromService 16
uspModifyCarInService 16
uspAddCrewToTrain 16
uspRemoveCrewFromTrain 16
uspMoveTrain 16
uspMoveCarToTrain 16
uspMoveCarFromTrainToYard 16
uspMoveCarFromTrainToIndustry 16
uspServiceIndustry 16
VIEWS 16

GAME OVERVIEW

Freight train operations are a great way to interact with a layout, treating it as a linear puzzle game. The goal of the game is to take trains to local industries to pick up and deliver goods. The challenge is getting a freight car out from the middle of your train into the siding ahead of it. How do you do this with only using push and pull movements? Now how do you do it in a finite space without blocking the main line with the rest of your train?

This system was first created for the Pacific Northwest Region 4dNtrak NRMA model railroading group to provide a means of performing structured switching operations on a modular and dynamically changing layout during exhibition shows. It has since been adapted for wider use by popular request. While designed for modular layouts, non-modular layouts should be able to treat module game entities as a specific geographic areas for track, yards, and industries.

PLAYER ROLES

- Dispatcher
    - Coordinator of layout staffing operations.
    - Assists Yard Masters with Car Card management.
- Yard Master
    - Assigns rolling stock to handle selected waybills.
    - Blocks together trains for Train Operators.
- Train Operator
    - Assigned a DCC throttle and transfers equipment or passengers from module to module.

SHIPPING ORDERS, WAYBILLS, AND TRAINS

1. Shipping orders are received at a yard for a product delivery.
2. Waybills are used to associate freight cars to shipping orders.
3. Trains are blocked together in "consists" of freight cars.

Note: These are all things that a game master might set up for a player. For regular players, they're the ones that handle delivering the goods.

FREIGHT CAR LIFECYCLE

- The 1st operator is assigned a locomotive and a train consisted with cars. An associated freight car is picked up from a local yard and is delivered to the waybill's "load" point. When delivered, the relevant freight car remains at the industry's siding and the shipping order is considered to be "picked up" and the freight car "loaded".
- The 2nd operator servicing an industry collects the car from the industry siding, adds the rolling stock to their train, and delivers it to the waybill's "unload" point. The car card is again deposited in the industry's siding and the shipping order is considered to be "delivered" and the freight car "unloaded".
- The 3rd operator servicing an industry collects the car again, and this time delivers the "empty" rolling stock to the specified return yard. The car is then given to the yard master for reclassification.

ROLLING STOCK, INDUSTRIES, AND PRODUCTS

TODO: Write info about car identification and types. Describe car lengths, industry sidings, and product associations. Industry matching. Producers and Consumers.

SERVICING INDUSTRIES

The goal of the operator is to deliver all assigned rolling stock, return all empty cars to the assigned yards, and return the locomotive to its assigned yard. Operators can elect to pick up existing rolling stock at industries currently being serviced to deliver the cars to the next destination. If the loading track at an industry is full, the operator must collect all waiting rolling stock, but not to exceed the maximum train length.

END OF GAME PARAMETERS

The operator has delivered all rolling stock in their assigned train, possesses no freight cars, and has returned to their assigned yard. Alternately, the operator has only “empty” rolling stock and has delivered it to the destination yard.

POPULATING A LAYOUT

TODO: Write this section.

SECTION A: INITIALIZE GAME PARAMETERS

Expected Insertion Order:

1. Define all RollingStockTypes
2. Define all ProductTypes.

Note: For each category of rolling stock in use, declare a car type and car length.

Note: For each category of rolling stock car type, multiple product types can be carried. Declare all product types and rolling stock car associations.

SECTION B: BUILD LAYOUT

Expected Insertion Order (for each Module):

1. Declare Modules
2. Declare MainLines for Modules
3. Declare Junctions on MainLines
4. Declare Yards on MainLines
5. Declare Industries on MainLines
6. Declare IndustriesAvailable for Industries
7. Declare IndustryActivities for Industries
8. Declare IndustryProducts for Industries
9. Declare IndustrySidings for Industries
10. Declare SidingAssignments for IndustrySidings

Note: Data declared in this section is likely to remain persistent and will be used across multiple game sessions.

SECTION C: INITIALIZE GAME SESSION

Expected Insertion Order:

1. Declare all ModulesAvailable for Modules.

2. Declare all Regions for Modules.

Note: This section contains non-persistent data which typically only exists for a single game session. A game session is considered active when trains, rolling stock cars, and crews are in existence. To start a game, select which modules are present from the library of previously defined modules and declare which region they are to be in to determine layout shape.

SECTION D: GAME SESSION

Expected Insertion Order:

1. Declare Trains
2. Declare TrainLocations for Trains
3. Declare RollingStockCars
4. Declare Shipments
5. Declare Waybills for Shipments on RollingStockCars
6. Declare ConsistedCars for Trains using RollingStockCars
7. Declare TrainCrews for Trains

Note: A locomotive is selected to declare a new train will exist, originating on a particular module. Available rolling stock for trains are identified and must be assigned to an originating yard or industry. Shipment orders are created on demand based off of current layout and industry parameters. For each rolling stock car, waybills must be associated with a shipping order before that car is consisted to a train. For each train, crews are declared and then assigned to a train. The train and player are then considered to be ready for game play.

GAMEPLAY

TODO: Write this section.

For every industry, industries have product types that are produced or consumed. Each product type is associated with a specific type of rail car. Shipping orders are generated to pick up a “produced” product at one industry and deliver to another “consumed” industry. Every “produced” industry interacts with every other matching “consumed” industry by product type, except itself. Shipping orders only exist for types of rail cars in use for the current game session.

For each player, a train is assigned. For each train, rail cars are added. For each rail car, a shipment order is created, assigned to a train via a waybill, and the car is added to the train. Finally, players are assigned to trains as crew. Players will be able to deliver rail cars to industries on the layout.

DATABASE DESIGN

This freight train operations simulator is a database driven game with most game logic held within the database itself. This design breaks the database into three groups of relations: Layout, Game, and Player

Layout: This data defines game parameters and relationships, such as properties of rolling stock, product types, and industries.

Game: This data defines the "state of the world" for a game session. It includes active crews, available rolling stock, available industries, and industry occupancy.

Player: This data tracks shipping orders, active waybills (destination orders), and industry interaction.

TODO: Add more to this. Consider just Layout and Session instead.

DATABASE TABLES

LAYOUT

Modules

Defines properties of a physical space containing industries and railroad.

Attributes:

- ModuleName: Unique name for a module
- ModuleOwner: Owner name for a module
- ModuleType: Arbitrary description of module type
- ModuleShape: Arbitrary description of module shape
- Description: Other module notes

MainLines

Defines a railroad track on a module that allows passage from at least one side of a module. A main line is contiguous if it allows through traffic to pass over the entirety of a module. Main lines are used by a routing engine and will indicate a path for a train to follow (version 2.0).

Pre-conditions: A valid Modules entity must exist.

Attributes:

- LineName: Name for a module train track
- ModuleName: References Modules relation for identity
- IsContiguous: Determines full transit authority

Junctions

Defines a connection point between two main lines on a module where trains may cross from one main line to another. Physically, junctions are represented by turnouts, and are currently non-directional in nature. Junctions are used by a routing engine and will indicate a path for a train to follow (version 2.0).

Pre-conditions: A valid Modules entity and two MainLines entities must exist.

Input Constraint: FromLine <> ToLine

Attributes:

- JunctionID: Arbitrary identifier
- ModuleName: References Modules relation for identity
- FromLine References MainLines LineName
- ToLine References MainLines LineName

Industries

Defines a business, accessible by rail, producing or consuming goods, and transports those goods via rolling stock cars.

Pre-conditions: A valid Modules entity and a MainLines entity must exist.

Attributes:

- IndustryName: Name for an industry on a module
- ModuleName: References Modules relation for identity
- MainLine: References MainLines LineName

IndustryActivities

Defines the overall frequency of an existing industry to be considered for new shipping orders.

Pre-conditions: A valid Industries entity must exist.

Input Constraint: ActivityLevel = {1, 2, 3} where 1 is lowest and 3 is highest.

Attributes:

- IndustryName: References Industries IndustryName
- ActivityLevel: Setting to control automatic shipping order matching frequency

IndustryProducts

Defines which product types an industry produces or consumes. An industry product entity must be created for each product type served. If this entity exists and IsProducer is FALSE, then the industry is a consumer for that product type.

Pre-conditions: A valid Industries entity and a ProductTypes entity must exist.

Attributes:

- IndustryName: References Industries IndustryName
- ProductTypeName: References ProductTypes ProductTypeName
- IsProducer: Determines if an industry produces or consumes a product

IndustrySidings

Defines which track siding an industry uses for rolling stock cars. Declares the remaining length available for an industry siding when occupied by rolling stock cars. AvailableLength is used to determine if industries should be considered for new shipping orders. If AvailableLength is less than the length of an incoming rolling stock car, the siding is considered to be "full" and should not be considered for new shipping orders. An industry must have at least one track siding to be accessible.

Pre-conditions: A valid Industries entity must exist.

Input Constraints: SidingLength > 0, AvailableLength > 0, AvailableLength <= SidingLength

Attributes:

- IndustryName: References Industries IndustryName

- SidingNumber: Arbitrary siding ID
- SidingLength: Arbitrary siding length, in feet
- AvailableLength: Stored current value for available siding length remaining

SidingAssignments
Defines preferences for specific product types going to only specific industry sidings. An industry siding will accept all product types for an industry by default. Siding assignments will restrict a siding to only authorized product types. Siding assignments should not be used unless at least two industry sidings exist.

Pre-conditions: A valid Industries entity must exist more than one declared IndustrySidings entities and at least one ProductTypes entity.

Input Constraint: The count for total industry sidings at an industry must be at least 2.

Attributes:

- IndustryName: References Industries IndustryName
- SidingNumber: References IndustrySidings SidingNumber
- ProductTypeName: References ProductTypes ProductTypeName

Yards
Defines a collection of tracks used to store trains and rolling stock when not actively being used by crew in a game session. Yards are the typical origination point of empty rolling stock cars and the eventual destination for empty rolling stock cars after shipments have been completed.

Pre-conditions: A valid Modules entity and a MainLines entity must exist.

Attributes:

- YardName: Name for a train yard on a module
- ModuleName: References Modules ModuleName
- MainLine: References MainLines LineName

GAME

ModulesAvailable
Defines the availability of a previously created module. A module is available if it is part of a layout, but can remain defined and not be available.

Pre-conditions: A valid Modules entity must exist.

Attributes:

- ModuleName: References Modules relation for identity
- IsAvailable: Allows a module to be disabled but not deleted

Regions
Defines groupings of modules and assigns a common name for the purpose of layout maps. Regions are used by a routing engine to indicate a path for a train to follow (version 2.0). They also provide

information to the shipping order generator to allow for physical layout spacing between load and unload points (version 2.0).

Pre-conditions: A valid Modules entity must exist.

Attributes:

- RegionName: Name for a collection of modules
- ModuleName: References Modules relation for identity

IndustriesAvailable
Defines the availability of an existing industry to be used in creating new shipping orders. Industries have the ability to be disabled during a game session to prevent new shipping orders from being created.

Pre-conditions: A valid Industries entity must exist.

Attributes:

- IndustryName: References Industries IndustryName
- IsAvailable: Allows an industry to be disabled but not deleted

RollingStockTypes
Define a rolling stock car by name and length.

Input Constraint: CarLength > 0

Attributes:

- CarTypeName: Name of a type of rolling stock car
- CarLength: Arbitrary average length used for computing available space on sidings

ProductTypes
Define a product type and which rolling stock car type it is carried by.

Pre-conditions: A valid RollingStockTypes entity must exist.

Attributes:

- ProductTypeName: Name of a product to be carried by a car
- CarTypeName: References RollingStockTypes CarTypeName

PLAYER

RollingStockCars
Represents a physical train car used by players in a game session.

Pre-conditions: A valid RollingStockCarTypes entity must exist.

Attributes:

- CarID: User defined unique ID for freight cars
- CarTypeName: References RollingStockTypes CarTypeName

RollingStockAtYards

Declares the identies of rolling stock cars currently at a specific train yard. Rolling stock cars not consisted to a train or reported at an industry must report at a yard.

Pre-conditions: A valid RollingStockCars entity and Yards entity must exist.

Attributes:

- CarID: References RollingStockCars CarID
- YardName: References Yards YardName
- TimeArrived: Time stamp

RollingStockAtIndustries

Declares the identities of rolling stock cars current at a specific industry. Rolling stock cars not consisted to a train or reported at a yard must report at an industry.

Pre-conditions: A valid RollingStockCars entity and Industries entity must exist.

Attributes:

- CarID: References RollingStockCars CarID
- IndustryName: References Industries IndustryName
- TimeArrived: Time stamp

Shipments

Declares a shipping order of a product type for pickup at one industry and delivery to another industry. Shipping orders are created on demand when rolling stock cars are added to a game session (version 2.0).

Pre-conditions: A valid ProductTypes entity and two Industries entities must exist. For each industry, one IndustrySidings entity must exist.

Input Constraint: FromIndustry <> ToIndustry

Attributes:

- ShipmentID: Arbitrary unique ID.
- ProductType: References ProductTypes ProductTypeName
- FromIndustry: References Industries IndustryName
- FromSiding: References IndustrySiding SidingNumber
- ToIndustry: References Industries IndustryName
- ToSiding: References IndustrySiding SidingNumber
- TimeCreated: Time stamp

ShipmentsLoaded

Declares if a shipping order has been picked up from an Industry producing a certain product type. If this entity exists, the product has been loaded onto a rolling stock car.

Pre-conditions: A valid Shipments entity must exist.

Attributes:

- ShipmentID: References Shipments ShipmentID
- TimePickedUp: Time stamp

ShipmentsUnloaded
Declares if a shipping order has been delivered to an Industry consuming a certain product type. If this entity exists, the product has been unloaded from a rolling stock car.

Pre-conditions: A valid Shipments entity must exist.

Attributes:

- ShipmentID: References Shipments ShipmentID
- TimeDelivered: Time stamp

Waybills
Declares the association of a shipping order and a specific rolling stock car. YardName determines the location empty rolling stock will be sent to after the shipping order is complete.

Pre-conditions: At least one valid entity for RollingStockCars, Shipments, and Yards must exist.

Attributes:

- CarID: References RollingStockCars CarID
- ShipmentID: References Shipments ShipmentID
- YardName: References Yards YardName

Trains
Represents a physical locomotive (or locomotive group) for a player to control in a game session. TimeCreated is for reference only and should not be updated.

Pre-conditions: A Modules entity must exist for player origination.

Attributes:

- TrainNumber: Arbitrary unique ID
- LeadPower: Number board identifier for lead locomotive
- DCCAddress: Digital Command Controller ID number
- TimeCreated: Time stamp

TrainLocations
Represents the location of a train on a layout. TimeUpdated will apply the current timestamp automatically with any activity.

Pre-conditions: A valid Trains entity and Modules entity must exist.

Attributes:

- TrainNumber: References Trains TrainNumber
- ModuleName: References Modules ModuleName

- TimeUpdated: Time stamp for last action

ConsistedCars

Represents the association of individual rolling stock cars attached to trains.

Pre-conditions: A valid Trains entity must exist, and for each car added, a RollingStockCars entity must exist.

Attributes:

- TrainNumber: References Trains TrainNumber
- CarID: References RollingStockCars CarID
- TimeAdded: Time stamp

TrainCrews

Declares the association of a crew with a train.

Pre-conditions: A valid Trains entity and valid Crews entity must exist.

Attributes:

- TrainNumber: References Trains TrainNumber
- CrewName: Player name
- TimeJoined: Time stamp

FUNCTIONS

TODO: Talk about what these are generally for and their usage.

ufnGetProductType

For a given car type, return a random product type. The product type is guaranteed to have a producer and consumer on the layout.

Usage: ufnGetProductType('CarTypeName')

ufnGetProducingIndustry

For a given product type, return the name of a random industry that produces that product and ensure that the transporting railcar will fit at that industry siding. Selection of industries are weighted so those with higher activity levels are more likely to be chosen. Industries become ineligible for shipments if the available length reported for their sidings is less than the servicing car's reported CarLength.

Usage: ufnGetProducingIndustry(CarLength, 'ProductTypeName')

ufnGetConsumingIndustry

For a given product type, return the name of a random industry that consumes that product and ensure that the transporting railcar will fit at that industry siding. Selection of industries are weighted so those with higher activity levels are more likely to be chosen. Industries become ineligible for shipments if the available length reported for their sidings is less than the servicing car's reported CarLength.

Usage: ufnGetConsumingIndustry(CarLength, 'ProductTypeName')

ufnGetIndustrySiding

For a given industry and product type, return a siding number capable of servicing the given product type which has the maximum amount of unoccupied space available for rolling stock.

Usage: ufnGetIndustrySiding('IndustryName', 'ProductTypeName')

ufnGetCarModuleName

For a given rolling stock Car ID, return the name of the module that car is reported to be on.

Usage: ufnGetCarModuleName('CarID')

ufnCheckServiceableCarSiding

For a given industry and product type, return a bool {TRUE, FALSE} if a rolling stock car is on a siding which is capable of loading or unloading goods for that product type.

Usage: ufnCheckServiceableCarSiding('ProductTypeName', 'IndustryName', SidingNumber)

STORED PROCEDURES

TODO: Talk about what these are generally for and their usage.

uspAddTrain

Add a player train to a game session. The train must originate on a module that has been declared available.

Pre-conditions:

- The given TrainNumber must be unique.
- Valid Modules and ModulesAvailable entities must exist and match the given ModuleName input.

Post-conditions: A Trains entity and TrainLocations entity is created.

Usage: uspAddTrain(TrainNumber, LeadPower, DCCAddress, 'ModuleName')

uspRemoveTrain

Remove a player train from a game session. A player train cannot be removed if it has consisted cars.

Pre-conditions:

- A valid Trains entity must exist and match the given TrainNumber input.
- The given TrainNumber attribute must not exist in any ConsistedCars entities.

Post-conditions: The supplied Trains and TrainLocations entities are deleted.

Usage: uspRemoveTrain(TrainNumber)

uspModifyTrain

Update the LeadPower (locomotive number) and DCCAddress descriptions for a player train.

Pre-conditions: A valid Trains entity must exist.

Post-conditions: The LeadPower and DCCAddress attributes are modified on the Trains entity.

Usage: uspModifyTrain(TrainNumber, LeadPower, DCCAddress)

uspAddCarToGame

Add a rolling stock car to a game session for player use. Declare an identifying name and valid car type for each rolling stock car. Cars are to be added to yards for initial classification.

Pre-conditions:

- The given CarID must be unique.
- A valid RollingStockTypes entity must exist for the given CarTypeName.
- A valid Yards entity must exist for the given YardName.

Post-conditions: A RollingStockCars and RollingStockAtYards entity is created.

Usage: uspAddCarToGame('CarID', 'CarTypeName', 'YardName')

uspRemoveCarFromGame

Remove a rolling stock car from a game session. Cars cannot be removed if they are consisted to a train or have a waybill and are in service.

Pre-conditions:

- A valid RollingStockCars entity must exist.
- Associated ConsistedCars and Waybills entities must not exist.

Post-conditions: The supplied RollingStockCars, RollingStockAtIndustries, and RollingStockAtYards entites are deleted.

Usage: uspRemoveCarFromGame('CarID')

uspAddCarToService

Generate a shipping order and associate a waybill to a rolling stock car that is active (added) in a game session. To be added to service, the rolling stock car must not currently be in service. A car will not be able to be added to service if its associated product type does not have a producer and consumer industry on the current layout, or if industries report that they are at production capacity and no goods are available to ship. An industry will become available again after it is serviced by players.

Pre-conditions:

- A valid RollingStockTypes entity must exist.
- An associated Waybills entity must not exist.
- For a given car type, the product type must be allowable on the current layout.
- For a given product type, an associated producing industry must have available length greater than the given rolling stock car's length on an industry siding.
- For a given product type, an associated consuming industry must have available length greater than the given rolling stock car's length on an industry siding.

Post-conditions:

- A Shipments entity is created.
- A Waybills entity is created and associated with the given RollingStockCars entity.
- AvailableLength is reduced by the rolling stock car's length on an assigned producing industry siding.

Usage: uspAddCarToService('CarID')

uspRemoveCarFromService

TODO: Add comments.

uspModifyCarInService

TODO: Add comments.

uspAddCrewToTrain

TODO: Add comments.

uspRemoveCrewFromTrain

TODO: Add comments.

uspMoveTrain

TODO: Add comments.

uspMoveCarToTrain

TODO: Add comments.

uspMoveCarFromTrainToYard

TODO: Add comments.

uspMoveCarFromTrainToIndustry

TODO: Add comments.

uspServiceIndustry

TODO: Add comments.

VIEWS

TODO: Talk about what these are generally for and their usage.