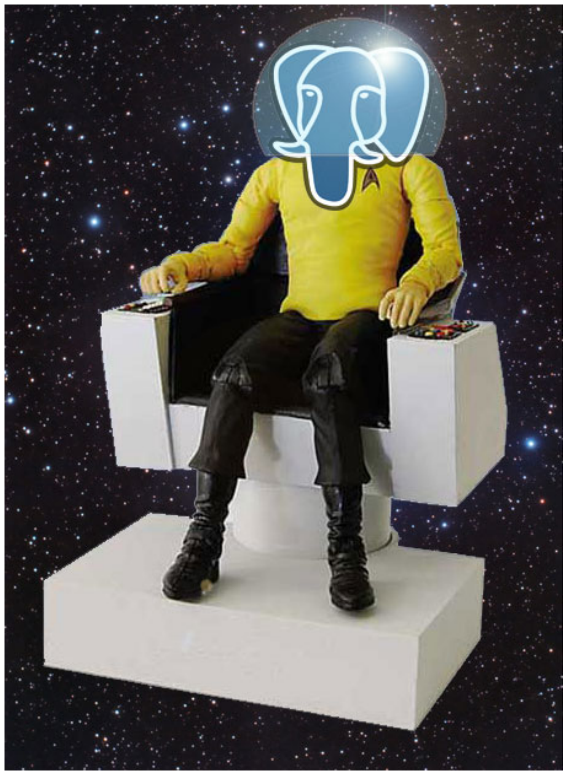


How the hell do you play this game?

SCHEMAVERSE
DEFCON

An Introduction

Welcome to the Schemaverse! Your mission in this game is to fly around the universe and conquer more planets than any of the other players. To accomplish this mission, you will need to give your fleets strategies to follow, build new ships, and upgrade your ships' statistics so that they can attack, defend, repair, and mine resources from the planets you come across.

It's a pretty standard space battle game really. But, there is one minor difference. There is **no** pretty interface. Unless you write one, this universe exists purely in the form of data.

To join the battle, head over to the DEFCON Contest Area and register.

A Thank You

To all my friends that helped put this document together, gave input on the presentation and spent countless hours testing The Schemaverse, I really can't thank you enough. Especially **Tigereye**, **appl**, **rick, Saint** and **Netlag,** this would not have happened without all your help.

Our sponsors below also deserve a big thank you. Their fantastic contributions for prizes have helped to legitimize the tournament in its first year and enhance the level of competition.

-Abstrct

PostgreSQL — The world's most advanced open source database.

Navicat™ — The gateway to your database !

CMD — The PostgreSQL Company

PostgreSQL Conference West 2011 — September 27-30 — San Jose Convention Center

Getting Started

What does the universe look like?

So where is it best to begin? Well first off you should take a look around. Run the following SELECT statement to see what planets fill up the universe:

```
SELECT * FROM planets;
```

Since you are just starting out, seeing only the closest planets would probably be more helpful. SQL is your friend here! Just change the statement, as you would expect:

```
SELECT * FROM planets WHERE
    location_x BETWEEN -5000 AND 5000
    AND location_y BETWEEN -5000 AND 5000;
```

Every player is made the conqueror of a random planet at the time of registration. Look for your player_id in the conqueror_id column of the planets view and start here!

Creating some Ships

Seeing as this game is about flying space ships around, you'll probably want at least one of those. You can create a ship at any time for the cost of 1000 credits.

```
INSERT INTO my_ships(name) VALUES('Shipington');
```

There are some values of the ship that you can change right off the bat. These values are the ship's Attack, Defense, Engineering (repair), and Prospecting (mining) abilities. So long as these values add up to 20, you can distribute them as you see fit. The default value for each is 5.

For example, if you wanted to build a ship meant for killing, you may want to create a ship like so:

```
INSERT INTO my_ships(name, attack, defense, engineering, prospecting)
    VALUES('My First Attacker',15,5,0,0);
```

All these skills can be upgraded; these initial values are just a starting point for your ship.

You can now take a look at your huge fleet of 1 ship by checking out your my_ships view:

```
SELECT * FROM my_ships;
```

Do you want to see if there are any ships around you currently? You can use the ships_in_range view for that:

```
SELECT * FROM ships_in_range;
```

You can also specify the starting location of your ships during an insert like so:

```
INSERT INTO my_ships(name, location_x, location_y)
    VALUES(“My Strategically Placed Ship”, 100, 100);
```

There is a catch though! You can only create a ship where one of the following is true

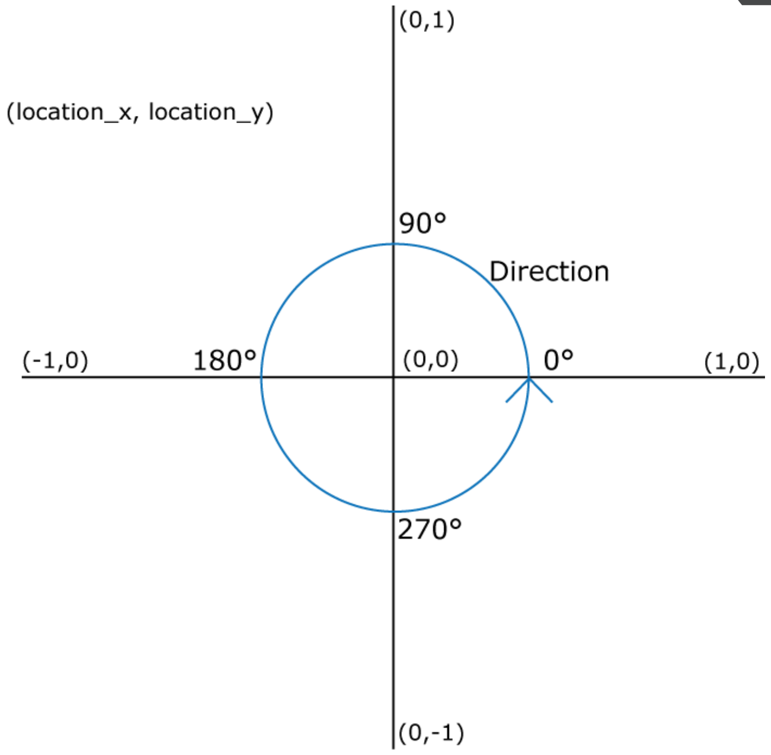
- location_x and location_y is between -3000 and 3000
- location_x and location_y are the same coordinates as a planet you are the current conqueror of

Moving around

To move around, you can use the command aptly named Move() which is defined like this:

```
Move(Ship ID, Speed, Direction, Destination X, Destination Y)
```

Ship ID should be an ID of one of your own ships, speed is the distance you will travel in one tic and direction is a value between 0 and 360 (in this game, space is 2D). I’m not hardcore enough for a 3D SQL based space game. That would just be weird. If you want to calculate the direction automatically based on your specified destination_x and destination_y coordinates, simply set direction as NULL and it will be filled in for you.



Now, assuming you want to move all your ships at the same time, there is nothing stopping you from doing the following:

```
SELECT
   MOVE(id,100, 200, destination_x, destination_y),
   id, name, location_x, location_y
   FROM my_ships;
```

When using the MOVE() function, you can call it in two ways:

"I want to go there"
If you know your destination coordinates, you can specify these as the last two parameters in the MOVE() function. MOVE() will calculate how much fuel it would take to accelerate you to the specified speed, and decelerate you when you reach that destination. If your ship has enough fuel, the MOVE() command will return true ('t') and you will be on your way.

If you don't have enough fuel, your error channel will report this. See "What the hell is going on" below for more information.

"I want to go that way"
If you don't specify a destination manually, you must specify a manual direction. This will set your ship on course and no fuel calculations will be made to ensure you are able to stop since you don't have a destination.

Keep in mind that that each ship can only move once per game **tic**. So, if you use Move() on a ship, you will not be able to run Move() on that ship again until tic.pl is run.

At the end of each tic, every ship (which has not had the MOVE command run manually on it) will progress in the direction specified in the ship's control information. If the ship has reached its destination (if one exists), the ship will try to stop (if there's enough fuel). You can see this information for all your ships with

SELECT direction, speed, current_fuel from my_ships;

To update this data for all your ships, you could run:

UPDATE my_ships SET
direction=180, speed=20 WHERE 1=1;

If you wanted only a single ship, the command:

UPDATE my_ships SET direction=90, speed=10 WHERE name='Shipington'

If your ships run out of fuel, you can fill them up with the fuel in your my_player.fuel_reserve. This command would refuel all your ships at once:

SELECT REFUEL_SHIP(id), id FROM my_ships;

Actions

Outside of moving around, there are three main actions that a ship can perform once per tic. These actions must be performed on ships and/or planets that are within range of the ship. If a ship is down to 0 health it will not be able to perform any of them until it is repaired. These actions are as follows:

- Attack(AttackerShip, EnemyShip)

  SELECT Attack(ship_in_range_of, id), name FROM ships_in_range;

  This would cause all of your ships to attempt to attack any ship that is in range.

- Repair(RepairShip, DamagedShip)

  SELECT Repair(10, id) FROM my_ships ORDER BY current_health ASC;

  This would use ship with ID 10 to repair the most damaged ship you own.

- Mine(MinerShip, Planet)

  SELECT mine(9, 1);

In this example, my ship with ID 9 would try to mine planet 1. This adds the ship to the planet_miners table and at the end of a tic, the system will decide who in the table is awarded fuel from the planet.

What the hell is going on

As you play the game, you may want to keep track of what is actually happening (or you may not…). To do so, you can watch the my_events view. To see it ordered with the latest events at the top you could do the following:

*SELECT * FROM my_events ORDER BY toc DESC;*

If you would like a more readable version of events, use the read_event() function within the select statement like so:

SELECT READ_EVENT(event_id) FROM my_events ORDER BY toc DESC;

There will also be times where things just don't seem to be working right. Originally, this game had an error log table, but it just grew out of control constantly and was pretty much useless. So, the solution to this was to utilize the NOTIFY and LISTEN commands to create an error channel that you can listen on.

Check your my_players view to find your error channel and if your PostgreSQL client allows it, you can use:

LISTEN <channel name>;

With every next query you make (until UNLISTEN), the response will include any new messages to your channel.

If your client doesn't support it or it just doesn't seem that convenient, fear not! If you can get python working on your system then you use the Schemaverse SOS client, SchemaverseOutputStream.py, from our GitHub repository (https://github.com/Abstrct/Schemaverse/tree/master/clients/SchemaverseOutputStream).

Buying Upgrades

To upgrade your ship use the function: UPGRADE (Ship ID, Code, Quantity)

The following is the price list at the time of publishing:

code	cost	description
MAX_HEALTH	50	Increases a ships MAX_HEALTH by one
MAX_FUEL	1	Increases a ships MAX_FUEL by one
MAX_SPEED	1	Increases a ships MAX_SPEED by one
RANGE	25	Increases a ships RANGE by one
ATTACK	25	Increases a ships ATTACK by one
DEFENSE	25	Increases a ships DEFENSE by one
ENGINEERING	25	Increases a ships ENGINEERING by one
PROSPECTING	25	Increases a ships PROSPECTING by one

There are certain limits regarding how much you can upgrade your ships. Those values can all be found in the public_variable view. At the time of publishing, they were:

Ability	Max Value
MAX_SHIP_SKILL	500
MAX_SHIP_RANGE	2000
MAX_SHIP_FUEL	5000
MAX_SHIP_SPEED	2000
MAX_SHIP_HEALTH	1000

The Tic (or flow of game)

A tic is a unit of time in the Schemaverse. Tics occur approximately every minute, but they can vary depending on how long it takes to execute fleet scripts. There is a cron job that executes TIC.PL, which drives the universe forward by moving ships, awarding fuel for planets that are currently being mined, and executing fleet scripts.

The order of events in tic.pl is as follows:

- Every ship moves based on the ships direction, speed and destination coordinates currently stored in my_ships
- All fleets run their fleet_script_#() function if they have a runtime of at least 1 minute and are enabled
- Mining happens for all ships who ran the mine() command that tic
- Some planets randomly have their fuel increased
- Any damage/repair that occurred during the tic is committed to the ship table
- Any ships that have been damaged to zero health for the same amount of tics as the EXPLODED variable is set to (currently 60 tics or approximately 1 hour) are set to destroyed
- tic_seq is incremented

Every tic is numbered sequentially for the lifetime of the Schemaverse. As mentioned earlier, ships can only perform one action per tic. Every time a ship performs an action its *LAST_ACTION* column is updated. You can see the current tic number by executing the following SELECT statement:

SELECT last_value FROM tic_seq;

To execute commands automatically every tic, see *Fleets* below.

Fleets

Fleets are essentially groups of ships, but with a twist. You can attach PL/pgSQL code to be executed each tic, along with variables to track values during the script's execution.

When your script is executed each tic, the TIC.PL script logs into the Schemaverse with your user account and executes the contents of every activated fleet script you have. Your script can include any SQL commands you can think of and act upon any ships you choose – not just the ships within that fleet.

Using scripts, you can tell your ships to execute the Mine() action repeatedly to earn money, Move() commands to travel to other planets, as well as Attack(), Repair(), Convert_resource(), and any other SQL you can think of.

Each fleet needs three things in order to be active: the ENABLED field to be true (or 't'), some execution time purchased (using the UPGRADE() function), and some valid PL/pgSQL code to execute. Here are examples of how you can accomplish this:

```
INSERT INTO my_fleets (name) VALUES ('My First Script');

UPDATE my_fleets SET script = 'PERFORM Mine(id, 1) ON my_ships;' WHERE
   name = 'My First Script';

SELECT UPGRADE(id, 'FLEET_RUNTIME', 1);

UPDATE my_fleets SET enabled = 't' WHERE name = 'My First Script';
```

Keep in mind that upgrading the runtime of a fleet costs 10000000 per 1 new minute of time added.

Fleet Programming Tips

- To escape quotes when updating your scripts, use two single quotes in your PL/pgSQL (eg: ''a string'')

- Keep your scripts organized by using comments within them

- Call your script directly to test it for runtime errors. All Fleet Scripts can be called by using the following syntax:

  ```
  SELECT FLEET_SCRIPT_#();
  ```

  Where # is the Fleet ID of the fleet you want to run.

- Monitor your error channel to see if fleets are running each tic as you expect

For more examples of scripts, please visit the wiki at
https://github.com/Abstrct/Schemaverse/wiki/Fleet-Scripts

Random Details

Planets

Planets can run out of fuel. The actual amount of fuel a planet has remaining is hidden from players but if mining keeps failing, you should take that as a hint. Each tic 5000 planets have their fuel replenished. If a planet is empty during the current turn, it may have more next tic.

If you conquer a planet, you can name it with an UPDATE statement on the planets view.

3 Really Useful Functions

GET_PLAYER_ID(username);
GET_PLAYER_NAME(player_id);

Use these two to convert back and forth from the username and player id. This is mostly just to make it so that it feels like you are actually playing against other people, rather than against some numbers. Some examples of its use include:

```
SELECT id, get_player_id(username), username, get_player_username(id) FROM my_player;

SELECT get_player_username(player_id) FROM ships_in_range;
```

Finally, you will need to take note of the function called CONVERT_RESOURCE(StartResource, Quantity).

This function will allow you to sell your Reserve_Fuel for more money (or the other way around) to help build up your forces.

```
SELECT convert_resource('FUEL',500);

SELECT convert_resource('MONEY',500);
```

You can also specify the starting location of your ships during an insert like so:

```
INSERT INTO my_ships(name, location_x, location_y)

VALUES("My Strategically Placed Ship", 100, 100);
```

There is a catch though! You can only create a ship where one of the following is true

- location_x and location_y is between -3000 and 3000
- location_x and location_y are the same coordinates as a planet you are the current conqueror of

Quick Start Steps

These are the first five queries you should run to start making money in the game.

Step 1 - Create a ship at the centre of the universe (where planet 1 is)

```
INSERT INTO my_ships(name) values ('My First Ship');
```

Step 2 - Upgrade the ships mining ability

```
SELECT UPGRADE(id, 'PROSPECTING', 200) from my_ships;
```

Step 3 - Create a fleet that will run while you're not paying attention

```
INSERT INTO my_fleets(name) VALUES('My First Fleet');
```

Step 4 - Update the fleet to do something

```
UPDATE my_fleets
   SET
   script_declarations= 'miners RECORD; ',
   script='
   FOR miners IN  SELECT id FROM my_ships
   LOOP
   --Since I know that 1 is the center planet I am just hardcoding that in
   PERFORM MINE(miners.id, 1);
   END LOOP;
   ',
   enabled='t'
   WHERE
   name = 'My First Fleet'
```

Step 5 - Buy processing time for the fleet to use every tic. This will buy one minute (it's expensive!)

```
SELECT  UPGRADE(id, 'FLEET_RUNTIME', 1),  id,   name, enabled
  FROM
  my_fleets;
```

Whats next?

Convert Fuel - As you mine, this increases the value in your my_player.fuel_reserve column. You can use this fuel to fly around but you can also convert fuel to money to buy all sorts of great things like new ships, upgrades and fleet runtime.

This is a statement that would convert all your fuel to money:

SELECT convert_resource('FUEL', fuel_reserve) from my_player;

Buy more ships (Step 1) Upgrade more ships (Step 2) Change on your fleet script so that it mines, repairs, attacks, creates, and travels (Step 4).

Check the event log with:

*SELECT READ_EVENT(id), * from my_events;*

There is also an error stream the Schemaverse sends out. It uses the Postgresql NOTIFY command, but it is a bit involved to describe. Check out the "*What The Hell Is Going On?"* section for more details.

Tables

While browsing these tables you will notice that for many of them, a player does not even have the ability to SELECT from. This is because this information is hidden behind views to control what a player can see about others in the game.

You may still find this information interesting though because if you plan to create an item or trophy you can access any and all information you see below within the item/trophy script.

action

Column	**Type**	**Player Permissions**	Extra Details
action	character(20)	Select, Insert, Update	
string	text	Select, Insert, Update	

If you have added an item into the item table then you have the ability to insert/update an action here so long as action.name is the same as item.system_name. This allows for the item to add custom event logs when run.

event

Column	**Type**	**Player Permissions**	Extra Details
Id	integer		Sequence:event_id_seq
Action	character(20)		
player_id_1	integer		FK:player(id)
ship_id_1	integer		FK:ship(id)
player_id_2	integer		FK:player(id)
ship_id_2	integer		FK:ship(id)
referencing_id	integer		
descriptor_numeric	numeric		
descriptor_string	character varying		
Location_x	integer		

Location_y	integer		
Public	boolean		
Tic	integer		
Toc	timestamp without time zone		Default:NOW()

fleet

Column	**Type**	**Player Permissions**	Extra Details
Id	integer		Sequence:fleet_id_seq
Player_id	integer		FK:player(id)
Name	character varying(50)		
Script	text		The PL/pgSQL commands that make the body of your function
script_declarations	text		The PL/pgSQL definitions that make up the DECLARE section of your function
last_script_update_tic	integer		
enabled	boolean		
runtime	interval		How many minutes of execution are allowed before the script is forcefully aborted

item

Column	**Type**	**Player Permissions**	Extra Details
System_name	character varying	Select, Insert, Update	
Name	character varying	Select, Insert, Update	
description	text	Select, Insert, Update	
Howto	text	Select, Insert, Update	
persistent	boolean	Select, Insert, Update	Default:FALSE
Script	text	Select, Insert, Update	
creator	integer	Select	FK:player(id)
approved	boolean	Select	Default:FALSE
round_started	integer	Select	

item_location

Column	**Type**	**Player Permissions**	Extra Details
system_name	character varying		
location_x	integer		
location_y	integer		

planet

Column	**Type**	**Player Permissions**	Extra Details
id	integer		Sequence:planet_id_seq
name	character varying(50)		
fuel	integer		This is hidden from players!
mine_limit	integer		

location_x	integer		
location_y	integer		
conqueror_id	integer		FK:player(id)

planet_miners

Column	**Type**	**Player Permissions**	Extra Details
planet_id	integer		FK:planet(id)
ship_id	integer		FK:ship(id)

player

Column	**Type**	**Player Permissions**	Extra Details
id	integer		Sequence:player_id_seq
username	character varying		Unique
created	timestamp without timezone		
balance	integer		
fuel_reserve	integer		
password	character(40)		
error_channel	character(10)		
starting_fleet	integer		FK:fleet(id)

player_inventory

Column	**Type**	**Player Permissions**	Extra Details
id	integer		Sequence:player_inventory_id_seq
player_id	integer		FK:player(id); Default:GET_PLAYER_ID(SESSION_USER)
item	character varying		FK:item(system_name)
quantity	integer		Default:1

player_trophy

Column	**Type**	**Player Permissions**	Extra Details
round	integer	Select	
trophy_id	integer	Select	FK:trophy(id)
player_id	integer	Select	FK:player(id)

price_list

Column	**Type**	**Player Permissions**	Extra Details
code	character varying	Select	
cost	integer	Select	
description	text	Select	

ship

Column	**Type**	**Player Permissions**	Extra Details
id	integer		Sequence:ship_id_seq

fleet_id	integer		
player_id	integer		FK:player(id)
name	character varying		
last_action_tic	integer		
last_move_tic	integer		
last_living_tic	integer		
current_health	integer		
max_health	integer		Default:100
current_fuel	integer		
max_fuel	integer		Default:1100
max_speed	integer		Default:1000
range	integer		Default:300
attack	integer		Default:5
defense	integer		Default:5
engineering	integer		Default:5
prospecting	integer		Default:5
location_x	integer		Default:0
location_y	integer		Default:0
destroyed	boolean		Default:FALSE

ship_control

Column	**Type**	**Player Permissions**	Extra Details
ship_id	integer		FK:ship(id)
direction	integer		
speed	integer		

destination_x	integer		
destination_y	integer		
repair_priority	integer		

ship_flight_recorder

Column	**Type**	**Player Permissions**	Extra Details
ship_id	integer		FK:ship(id)
tic	integer		
location_x	integer		
location_y	integer		

stat_log

Column	**Type**	**Player Permissions**	Extra Details
current_tic	integer	Select	
total_players	integer	Select	
online_players	integer	Select	
total_ships	integer	Select	
avg_ships	integer	Select	
total_trades	integer	Select	
active_trades	integer	Select	
total_fuel_reserve	integer	Select	
avg_fuel_reserve	integer	Select	
total_currency	integer	Select	

avg_balance	integer	Select	

trade

Column	Type	Player Permissions	Extra Details
id	integer		Sequence:trade_id_seq
player_id_1	integer		FK:player(id)
player_id_2	integer		FK:player(id)
confirmation_1	integer		
confirmation_2	integer		
complete	integer		

trade_item

Column	Type	Player Permissions	Extra Details
id	integer		Sequence:trade_item_id_seq
trade_id	integer		FK:trade(id)
player_id	integer		FK:player(id)
description_code	character varying		
quantity	integer		
descriptor	character varying		

trophy

Column	Type	Player Permissions	Extra Details
id	integer	Select	Sequence:trophy_id_seq

name	character varying	Select, Insert, Update	
description	text	Select, Insert, Update	
picture_link	text	Select, Insert, Update	
script	text	Select, Insert, Update	
script_declarations	text	Select, Insert, Update	
creator	integer	Select	FK:player(id)
approved	boolean	Select	Default:FALSE
round_started	integer	Select	

variable

Column	**Type**	**Player Permissions**	Extra Details
name	character varying		
private	boolean		
numeric_value	integer		
char_value	character varying		
description	text		

Views

As a player, the views are where you are likely to spend most of your time querying around in.

my_events

Column	Type	Player Permissions
event_id	integer	Select
action	character(20)	Select
player_id_1	integer	Select
ship_id_1	integer	Select
player_id_2	integer	Select
ship_id_2	integer	Select
referencing_id	integer	Select
descriptor_numeric	numeric	Select
descriptor_string	character varying	Select
location_x	integer	Select
location_y	integer	Select
tic	integer	Select
toc	timestamp without time zone	Select

my_fleets

Column	Type	Player Permissions
id	integer	Select
name	character varying(50)	Select, Insert, Update
script	text	Select, Update

script_declarations	text	Select, Update
last_script_update_tic	integer	Select
enabled	boolean	Select, Update
runtime	interval	Select

my_player

Column	**Type**	**Player Permissions**
id	integer	Select
username	character varying	Select
created	timestamp without timezone	Select
balance	integer	Select
fuel_reserve	integer	Select
password	character(40)	Select
error_channel	character(10)	Select
starting_fleet	integer	Select, Update

my_player_inventory

Column	**Type**	**Player Permissions**
id	integer	Select
player_id	integer	Select
item	character varying	Select
quantity	integer	Select

my_ships

Column	Type	Player Permissions
id	integer	Select
fleet_id	integer	Select, Update
player_id	integer	Select
name	character varying	Select, Insert, Update
last_action_tic	integer	Select
last_move_tic	integer	Select
current_health	integer	Select
max_health	integer	Select
current_fuel	integer	Select
max_fuel	integer	Select
max_speed	integer	Select
range	integer	Select
attack	integer	Select, Insert
defense	integer	Select, Insert
engineering	integer	Select, Insert
prospecting	integer	Select, Insert
location_x	integer	Select, Insert
location_y	integer	Select, Insert
direction	integer	Select, Update
speed	integer	Select, Update
destination_x	integer	Select, Update
destination_y	integer	Select, Update

repair_priority	integer	Select, Update

When performing an INSERT on the my_ships view, the fields *attack*, *defense*, *engineering*, and *prospecting* can equal anything so long as their combined total is not greater then 20. Also during INSERT, you can use any location_x, location_y coordinates that fall on planets you currently have conquered.

my_ships_flight_recorder

Column	**Type**	**Player Permissions**
ship_id	integer	Select
tic	integer	Select
location_x	integer	Select
location_y	integer	Select

ships_in_range

Column	**Type**	**Player Permissions**
id	integer	Select
ship_in_range_of	integer	Select
player_id	integer	Select
name	character varying	Select
health	integer	Select
location_x	integer	Select
location_y	integer	Select

planets

Column	Type	Player Permissions
id	integer	Select
name	character varying(50)	Select, Update
mine_limit	integer	Select
location_x	integer	Select
location_y	integer	Select
conqueror_id	integer	Select

my_trades

Column	Type	Player Permissions
id	integer	Select
player_id_1	integer	Select, Insert
player_id_2	integer	Select, Insert
confirmation_1	integer	Select, Insert, Update
confirmation_2	integer	Select, Insert, Update
complete	integer	Select

trade_items

Column	Type	Player Permissions

id	integer	Select, Delete
trade_id	integer	Select, Insert
player_id	integer	Select
description_code	character varying	Select, Insert
quantity	integer	Select, Insert
descriptor	character varying	Select, Insert

trade_ship_stats

Column	Type	Player Permissions
trade_id	integer	Select
player_id	integer	Select
description_code	character varying	Select
quantity	integer	Select
descriptor	character varying	Select
ship_id	integer	Select
ship_name	character varying	Select
ship_current_health	integer	Select
ship_max_health	integer	Select
ship_current_fueld	integer	Select
ship_max_fuel	integer	Select
ship_max_speed	integer	Select
ship_range	integer	Select
ship_attack	integer	Select
ship_defense	integer	Select
ship_engineering	integer	Select

ship_prospecting	integer	Select
ship_location_x	integer	Select
ship_location_y	integer	Select

online_players

Column	Type	Player Permissions
id	integer	Select
username	character varying	Select

current_stats

Column	Type	Player Permissions
current_tic	integer	Select
total_players	integer	Select
online_players	integer	Select
total_ships	integer	Select
avg_ships	integer	Select
total_trades	integer	Select
active_trades	integer	Select
total_fuel_reserve	integer	Select
avg_fuel_reserve	integer	Select

total_currency	integer	Select
avg_balance	integer	Select

public_variable

Column	Type	Player Permissions
name	character varying	Select
private	boolean	Select
numeric_value	integer	Select
char_value	character varying	Select
description	text	Select

trophy_case

Column	Type	Player Permissions
player_id	integer	Select
username	character varying	Select
tropy	character varying	Select
times_awarded	integer	Select

Functions

Getting around

move(Ship ID, Speed, Direction, Destination X, Destination Y)

Use this function to move ships around the map. Each ship can execute the MOVE command once per tic. At the end of a tic, if the ship has not moved, but it has values in my_ships.speed and my_ships.direction then the MOVE command will be executed automatically for it.

It is also important to note that moving will decrease the ship's fuel supply when accelerating and when decelerating. Whether you travel 100m away or 1,000,000m away, the fuel cost will be 2x your speed: once to get up to speed, then once to stop at your destination. In addition, fuel is deducted when changing headings mid-flight at a cost of 1 fuel unit per degree changed mid-flight.

Any errors that occur during this function will be piped through the player's error_channel.

Parameters

Name	**Type**	**Description**
Ship ID	integer	
Speed	integer	Cannot be greater then my_ships.max_speed
Direction	integer	Leave this as NULL to have your ship automatically go in the direction required to get to your destination. A destination is required if this is set to NULL.
Destination X	integer	Use Destination X and Destination Y to tell the system to clear values my_ships.speed and my_ships.direction once the destination is in range. This will stop the ship from moving automatically next turn away from the destination
Destination Y	integer	Leave Destination X and Destination Y as NULL if you don't want to stop. You must specify a non-NULL direction if these are NULL.

Returns

Type	**Description**

boolean	Returns TRUE (t) if the ships move is successful and FALSE (f) if it is not.

refuel_ship(Ship ID)

Using this function will take fuel from your players fuel reserve (my_player.fuel_reserve) and add it to the fuel of the specified ship ID. It will always fill up the ship to the level of max_fuel.

This does not count as a ship action.

Errors that occur during this function are piped through the player's error_channel.

Parameters

Name	Type	Description
Ship ID	integer	

Returns

Type	Description
integer	Returns amount of fuel added to the ship.

Actions

attack(Attacking Ship ID, Enemy Ship ID)

Use this function to attack other ships. Be careful though, friendly fire is possible!

When the attack is executed successfully, an event will be added to the *my_events* view for both players involved. Any errors that occur during this function will be piped through the player's error_channel.

Using this function will act as an Action for the ship. The ship will not be able to perform another action until the game tic increases.

Parameters

Name	Type	Description
Attacking Ship ID	integer	
Enemy Ship ID	integer	

Returns

Type	Description
integer	Damage done in attack to enemy ship

mine(Mining Ship ID, Planet ID)

Use this function to mine planets that are in range. Mining is important, because it allows you to acquire fuel that can power your fleets or be converted to cash, in order to purchase upgrades.

When a ship starts mining by calling this command, the ship is added to one of the hidden Schemaverse system tables. At the end of each system tic, the Schemaverse tic.pl script executes a function called mine_planets(). For each planet currently being mined this tic, the system takes a look at each ships prospecting abilities and the amount of mining that can occur on a planet and calculates which ship(s) have successfully mined the planet. Once the actual mining takes place, the information will be added to the *my_events* view for all involved players.

At some point I will write a separate wiki page to describe the mining process in a bit more detail.

Any errors that occur during mining will be piped through the player's error_channel.

Using this function will act as an Action for the ship. The ship will not be able to perform another action until the game tic increases.

Parameters

Name	Type	Description
Mining Ship ID	integer	
Planet ID	integer	The Planet must be in range of the ship attempting to mine it

Returns

Type	Description
boolean	Returns TRUE (t) if the ship was successfully added to the current mining table for this tic. Returns FALSE (f) if the ship is out of range and could not be added

repair(Repair Ship ID, Damaged Ship ID)

Use this function to repair other ships. A ship with zero health cannot perform actions.

When the repair is executed successfully, the RepairShip's Engineering value will be added to the DamagedShip's health, and an event will be added to the *my_events* view for the player involved. Any errors that occur during this function will be piped through the player's error_channel.

Using this function will act as an Action for the ship. The ship will not be able to perform another action until the game tic increases.

Parameters

Name	**Type**	**Description**
Repair Ship ID	integer	
Damaged Ship ID	integer	

Returns

Type	**Description**
integer	Health regained by the ship

Purchasing and Trading

convert_resource(Current Resource Type, Amount to Convert)

Use this function to convert fuel to currency, or vice versa. The value of the fuel will fluctuate based on levels in the game.

Any errors that occur during this function will be piped through the player's error_channel.

Using this function does not count as an action and can be run as often as you like.

Parameters

Name	Type	Description
Current Resource Type	character varying	What is the player selling for conversion; either the string 'FUEL' or 'MONEY'
Amount to Convert	integer	

Returns

Type	Description
integer	Total resources acquired from the conversion

upgrade(Ship ID | Fleet ID, Product Code, Quantity)

Use this function to upgrade your fleets or your ships. This does not count as a ship action.

To see a list of what is available for upgrade, run a SELECT on the price_list table. Then use the code listed there for the Product Code parameter for this function.

There are a maximum amount of upgrades that can be done to ships. To learn the maximums look to the *public_variable* view.

Any errors that occur during this function will be piped through the player's error_channel.

Parameters

Name	Type	Description
Ship ID | Fleet ID	integer	
Product Code	character varying	See the price_list table for a list of values to use here.
Quantity	integer	

Returns

Type	Description
boolean	Returns TRUE (t) if the purchase was successful and FALSE (f) if there was a problem

Utilities

get_char_variable(Variable Name)

This utility function simply makes it easier to recall character varying values from the *public_variable* view.

Using this function does not count as an action and can be run as often as you like.

Parameters

Name	Type	Description
Variable Name	character varying	The name of the value you wish to return from *public_variable*

Returns

Type	Description
character varying	The matching character varying value from the *public_variable* view

get_numeric_variable(Variable Name)

This utility function simply makes it easier to recall integer values from the *public_variable* view.

Using this function does not count as an action and can be run as often as you like.

Parameters

Name	Type	Description
Variable Name	character varying	The name of the value you wish to return from *public_variable*

Returns

Type	Description
integer	The matching integer value from the *public_variable* view

get_player_id(Player Username)

This utility function performs a lookup of a users player id based on the username given.

Using this function does not count as an action and can be run as often as you like.

Parameters

Name	Type	Description
Player Username	character varying	

Returns

Type	Description
integer	The player id for the username supplied

get_player_username(Player ID)

This utility function performs a lookup of a player's username based on the Player ID given.

Using this function does not count as an action and can be run as often as you like.

Parameters

Name	Type	Description
Player ID	integer	

Returns

Type	Description
character varying	The player username for the Player ID supplied

get_player_error_channel(Player Username [DEFAULT SESSION_USER])

This utility function performs a lookup of a user's error_channel based on the username given. This information is readily available from my_players but this just makes the lookup easier.

Using this function does not count as an action and can be run as often as you like.

Parameters

Name	Type	Description
Player Username	character varying	

Returns

Type	Description
character(10)	The error channel for the username supplied

in_range_planet(Ship ID, Planet ID)

This utility function performs a lookup to see if a ship is within range of a specified planet. It's helpful to find out if a ship is able to mine a planet during this tic.

Using this function does not count as an action and can be run as often as you like.

Parameters

Name	Type	Description
Ship ID	integer	
Planet ID	integer	

Returns

Type	Description
boolean	Returns TRUE (t) if the Planet is within range and FALSE (f) if it is not

in_range_ship(Ship ID, Ship ID)

This utility function performs a lookup to see if a ship is within range of another specified ship. It's helpful to find out if a ship is able to attack or repair the other ship during this tic.

Using this function does not count as an action and can be run as often as you like.

Parameters

Name	Type	Description
Ship ID	integer	
Ship ID	integer	

Returns

Type	Description
boolean	Returns TRUE (t) if the Ships are within range and FALSE (f) if they are not

read_event(Event ID)

This utility uses the available data within a row of the event table to convert the information into a readable string of text. Consider the following entry in my_events:

EventID	Action	player_id_1	ship_id_1	referencing_id	descriptor_numeric	public
171	MINE_SUCCESS	1	1	1	1879	t

SELECT READ_EVENT(171) as string_event;
will return the following:

string_event
(#1)cmdr's ship (#1)dog has successfully mined 1879 fuel from the planet (#1)Torono"

Using this function does not count as an action and can be run as often as you like.

Parameters

Name	Type	Description
Event ID	integer	

Returns

Type	Description
Text	Returns the text based on the type of action being read and event details