Specification: Primary Game Data File

Purpose

This JavaScript file serves as the primary data store for a client-server strategy web application. It provides a centralized source of game state and configuration information, which is:

1. **Read and written on the server** via PHP using `json_decode()` and `json_encode()`.
2. **Read directly by the client** in JavaScript to render UI elements and execute game logic.

The file contains structured JSON-like objects representing game entities such as colonies, fleets, events, units, maps, and player/empires. It is intended to be authoritative for game state.

Data Structures

1. Colonies

Variable: `colonies`
Type: Array of objects

Each colony object contains the following properties:

Property	Type	Description
`name`	string	Name of the colony.
`capacity`	integer	Maximum statistics of the colony
`fort`	integer	Level of fortification.
`intel`	integer	Intelligence coverage.
`morale`	integer	Morale of the colony population.
`owner`	string	Owning empire, empty if unowned.
`population`	integer	Current population of the colony.
`raw`	integer	Resource richness index.
`type`	string	Environmental type (e.g., "Barren", "Adaptable", "Homeworld", "Dead", "Extreme", "Garden").
`notes`	string	Special features or historical notes about the colony.
`fixed`	array	Array of fixed structures or units that cannot move on their own, at the colony.

2. Empire

Variable: `empire`
Type: Object

Represents the player's empire.

Property	Type	Description
`empire`	string	Empire identifier.
`maintExpense`	integer	Maintenance expense.
`miscExpense`	integer	Miscellaneous expenses.
`miscIncome`	integer	Miscellaneous income.
`name`	string	Full name of the empire.
`systemIncome`	integer	Income from all controlled systems.
`previousEP`	integer	Economic points from previous turn.
`techYear`	integer	Current technological year.
`tradeIncome`	integer	Income from trade.
`researchInvested`	integer	Resources invested in research.

3. Events

Variable: `events`
Type: Array of objects

Each event object contains:

Property	Type	Description
`event`	string	Short event description.
`time`	string	Turn of event.
`text`	string	Detailed narrative or calculation for the event.

4. Fleets

Variable: `fleets`
Type: Array of objects

Each fleet object contains:

Property	Type	Description
`name`	string	Fleet identifier.
`location`	string	Current colony or sector.
`units`	array	List of ship/unit names in the fleet.
`notes`	string	Optional notes regarding fleet.

5. Game State

Variable: `game`
Type: Object

Property	Type	Description
`game`	string	Name of the game.
`turn`	integer	Current turn number.
`monthsPerYear`	integer	Game months per year.
`blankOrders`	integer	Number of blank orders available.

Property	Type	Description
`turnSegment`	string	Current phase of turn (e.g., "pre", “post”).
`nextDoc`	string	Identifier of next saved game file.
`previousDoc`	string	Identifier of previous saved game file.

6. Map Data

- **Map Points (`mapPoints`)**: Array of `[x, y, owner, name]` coordinates representing colonies on the map.
- **Map Connections (`mapConnections`)**: Array of `[from, to, status]` representing known or unexplored routes between colonies. `Status` is “Unexplored”, “Restricted”, “Minor”, “Major”

7. Orders, Treaties, and Projects

- **Orders (`orders`)**: List of player-issued actions in the format of the following:

Property	Type	Description
`type`	string	Order Type
`receiver`	string	Thing being ordered
`target`	string	Where to perform order
`note`	string	User-entered value
`perm`	boolean	If 0, show order in drop-downs if 1, no drop down. Show text

- **Offered Treaties (`offeredTreaties`)**: Array of treaties offered to other empires. Format is `[ Empire, Treaty Type ]`
- **Treaties (`treaties`)**: List of treaties with other empires

Property	Type	Description
`cooldown`	int	Diplomatic cooldown value
`type`	string	Political state: War, Hostilities, Neutral, Non-Aggression, Trade, Mutual Defense, Alliance
`income`	int	Known income of other power
`navy`	int	Known naval construction value of other power

- **Intel Projects (`intelProjects`)**: List of ongoing espionage or sabotage projects in the format of

Property	Type	Description
`Type`	string	Type of mission
`Target`	string	What system the mission is affecting
`Location`	string	Where the mission is sourced

8. Units and Purchases

- **Unit List (`unitList`)**: Contains ship/unit types with `ship` name, `yis` year introduced, `design` type, `cost`, and `notes`.

Property	Type	Description
`Ship`	string	Ship designator or class (e.g. “FFE”, “BCH”)
`YIS`	int	The year that the ship becomes available
`Design`	string	The ship hull designation (e.g. “CA”, “AB”)
`Cost`	int	The cost of the unit, in EPs
`Notes`	string	Any notes on the units of this class. Comma deliminated. Some notes are followed by a number in parenthesis.

- **Purchases (`purchases`)**: List of purchasable units with `name` and `cost`.
- **Under Construction (`underConstruction`)**: Tracks units currently being built at specific colonies.
- **Units in Mothballs (`unitsInMothballs`)**: Same format as fleets, but always named "Mothballs".
- **Units Needing Repair (`unitsNeedingRepair`)**: Array of unit identifiers who are crippled. Format of unit identifiers are `[unit designator]+” w/ “+[fleet name]`.
- **States of Units (`unitStates`)**: Array of unit identifiers who are have other states. The format is [`[unit designator]+” w/ “+[fleet name], “unit state” ]`. States can be “Out of Supply", "Exhausted", and other strings, but not “Crippled”. (“Crippled” states are noted in `unitsNeedingRepair`.)

9. Other Data

- **Other Empires (`otherEmpires`)**: Array of names of other player empires.
- **Unknown Movement Places (`unknownMovementPlaces`)**: Array of colony names where the statistics are not yet known. These names may not be the actual name of the colony at that location.

Server-Client Considerations

1. **Server-side (PHP):**
    - Use `json_decode()` to read the JS file into PHP associative arrays or objects.
    - Use `json_encode()` to write updates back to this file.
2. **Client-side (JavaScript):**
    - Directly reads the file to display game state and update the UI.
    - All data must be kept in valid JSON-compatible format.

Notes

- All arrays and objects should remain consistent in key naming and type to ensure compatibility between server and client.
- This file serves both as the authoritative game state and configuration reference for gameplay mechanics.
- Colony and unit names must be unique identifiers.
- The key names may be in any order, by it is customarily sorted alphabetically except that `unitList` is placed at the end. This is because `unitList` is generally the largest array and would make the file less readable if other keys are placed after it.
- Standard JSON_PRETTY_PRINT routines are space wasteful. A limited pretty-print algorithm is used: newlines are inserted after closing semicolons and after commas that separate array elements. The exception is that newlines are not inserted into arrays that are themselves nested inside a JSON object.