Updated: **Jan. 7th 2025** by *glitchdetector*

User Applications

This system allows loading any web app into the game, the web app is then fed various information automatically from the game client. The web app is shown directly in the game window, and can be interacted with by the player directly within the game.

See the example app at https://cdn.tycoon.community/dev/userapp/sample.html. It contains the basic code to listen to incoming data and send requests back to the game client.

You can also load the application `monitor.html` in-game to see all data values in real time.

Interface

By default, the F1 key opens the User Applications interface. The user is requested to input a Web URL for the web app they want to load.

The F1 key is used to regain focus to the web app and display the web app if it was hidden previously.

You can load more than one application at once, by clicking "New Tab" you create an additional application slot. You may currently have up to 5 applications loaded at once.

Tabs are not hidden by default, and will all stay visible at the same time. Apps may have specific behavior based on if they are the active tab or not. You are unable to physically interact with anything not in the current tab.

Pro tip: When you open the interface, but before interacting with anything, you may use the `TAB` key to switch between tabs.

Pro tip: When you open the interface, but before interacting with anything, you may use the `ESC` key to return control to the game (pin the applications).

Commands

You can send commands to the game from your web app.

The command is sent as a JSON object with `type` set to the command name, and each argument as their own properties. Eg `{type: "setWaypoint", x: 500, y: 250}`

Command	Parameters	Description
setWaypoint	x : number, y : number	Sets the in-game waypoint
sendCommand	command : string	Sends a console command to the game client
notification	text : string	Shows a notification over the map
info	text : string, time : number	Shows a lingering info message on the bottom right, time is in seconds
getData	No arguments	Forces the game client to send the entire data cache
getNamedData	keys : array	Requests only the named data keys from the cache
close	No arguments	Gives focus back to the game and hides the web app
pin	No arguments	Gives focus back to the game, but keeps the web app on screen
sfx	sfx : number	Plays a SFX, see list below for sfx indexes
popup	title : string, text : string	Shows a full-screen text message
oneliner	text : string	Shows a black mid-screen box with text
message	text : string	Shows a mid-screen text message
shareLocalData	key : string, value : string	Adds a shared key/value pair accessible by all loaded applications (see below)
shareServerData	key : string, value : string	Adds a server-shared key/value pair that is shared with all players (see below)

Data

This is a (not complete) list of data that the game may provide. The data is generally provided when updated in-game.

Key	Type	Description
user_id	number	The user ID of the player
source	number	The player index of the player
name	string	The player's name
job	string	The current job the player has
wallet	number	Current wallet balance
bank	number	Current bank balance
vehicle	string	model name for current occupied vehicle (onFoot when on foot)
vehicleClass	number	class id for vehicle
vehicleName	string	display name for vehicle
vehicleMake	string	vehicle brand name
vehicleClassName	string	Class name for vehicle (not always based on vehicleClass)
rpm	number	Vehicle engine RPM
engine	string	Vehicle engine state, either `on` or `off`
fuel	number	Remaining fuel in vehicle
honk	boolean	Is vehicle horn honking
car	string	owned spawned car model name
cab	string	owned spawned cab model name
trailer	string	owned spawned trailer model name
aircraft	string	owned spawned aircraft model name
helicopter	string	owned spawned helicopter model name
boat	string	owned spawned boat model name
notification	string	Last shown notification
pos_x	number	Player Position X component
pos_y	number	Player Position Y component
pos_z	number	Player Position Z component
pos_h	number	Player rotation (heading)
zone	string	Map area ID
zoneName	string	Name of map area
street	string	Current street name
discord	string	Discord identifier for user, if present (in raw form `discord:<id>`)
runway_[RUNWAY_ID]	string	Contains state of the runway with specified id, either `free` (yellow), `occupied` (red) or `reserved` (green). `reserved` being the local player having reserved it.
inventory	string	JSON of the current player inventory
weight	number	Current weight of inventory

Key	Type	Description
max_weight	number	Capacity of inventory
waypoint	boolean	Is a waypoint set?
waypoint_x	number	Waypoint Position X component
waypoint_y	number	Waypoing Position Y component
menu	string	Current open vRP menu title
menu_choice	string	Last vRP menu button choice
chest	string	Internal ID of the current open chest (storage, trunk etc)
chest_[chest_id]	string	JSON of the current inventory in the specified chest, only updates when chest is opened.
faction_id	number	Faction ID
faction_name	string	Name of the current faction
faction_tag	string	Faction chat tag
faction_president	boolean	Is the player the president of the faction?
pkey	string	The public API key for this user, if one is generated.
health	number	Player's current health
armor	number	Player's current armor
landing_gear	string	State of the landing gear. `deployed`, `retracting`, `deploying`, `retracted` or `broken`.
altitude	number	Vehicle altitude over terrain
hidden	boolean	Is the web app hidden (closed)?
pinned	boolean	Is the web app pinned (shown but not in focus)?
focused	boolean	Is the web app in focus?
tabbed	boolean	Is this web app the current tab?
players	string	JSON of online players with server id as string keys, each player is an object and should contain the name property
players_[key]	string	JSON of shared server data from web apps (see below)
local_[key]	string	Shared data between web apps (see below)
weather	string	Current weather type (or the one we're transitioning to)
weather_forecast	string	The next expected weather type
weather_frozen	boolean	Is the weather frozen? (not going to change)
weather_snow	boolean	Is there snow on the ground?

Player identifiers (like `steam`, `license` etc) are also provided when available, with the type as the key and the raw identifier as the value. Eg `steam: steam:12345678`

Runways

Runways will provide their state when they update, the different states are `free` (yellow), `occupied` (red) or `reserved` (green). `reserved` being when the local player called ATC for the runway.

The keys for runways always begin with `runway_`, and they generally follow the format `[airport]_[designation]` eg. `LSIA_MAIN`, `MGA_SIDE` or `SSIA_JET`.

Cache Behavior

Most key/value pairs are stored locally by the player. The cache will contain the value so it may be requested at any time. Keys with the following prefixes are not stored in cache and cannot be requested by `getData` or `getNamedData` :

- `temp_`
- `trigger_`
- `chest_`

Triggers

Triggers work just like data, and are sent as data. The key is always prefixed with `trigger_` . Triggers are not cached and will not be returned by `getData` . The value provided is the game client timer value, which is in milliseconds.

There are 4 built-in trigger binds that can be configured in `Settings > Keybinds > FiveM` . **Square**, **Triangle**, **Circle** and **Cross**.

These send `trigger_square` , `trigger_triangle` , `trigger_circle` and `trigger_cross` respectively. The command `userapp_trigger <key>` can send custom triggers, eg `userapp_trigger accept` would send `trigger_accept` to the web app.

Sound Effects

Using the `sfx` command, you can play a curated set of sound effects:

sfx id	Name
1	CHECKPOINT_MISSED
2	FLIGHT_SCHOOL_LESSON_PASSED
3	TIMER_STOP
4	Bed
5	MEDAL_UP
6	CHALLENGE_UNLOCKED
7	ScreenFlash
8	On_Call_Player_Join
9	Out_Of_Bounds_Timer
10	ROUND_ENDING_STINGER_CUSTOM
11	DELETE
12	OTHER_TEXT
13	GOLF_NEW_RECORD
14	GOLF_BIRDIE
15	GOLF_EAGLE
16	MP_RANK_UP
17	MP_WAVE_COMPLETE

Local Shared data

You can send data that is shared between all the applications loaded on the client using the `shareLocalData` command.

This data is fed back as a key/value pair with the following format: `local_[key] = [value]` .

Server Shared data

There's a built in protocol to share data between players on the current server, this allows you to communicate certain information between several players at once.

By invoking the command `shareServerData`, you can set a `key` and a `value` parameter that is synced across all players. The resulting synced value is stored in a single JSON field, containing all player's values for said key, and a server time value (in milliseconds, not system time) for when the value was last updated. The TTL (time-to-live) for the values stored is 2 minutes (120 seconds), in which new updates no longer include the expired values.

An example:

You send the server data of key `checkpoint` with a value of `5` from a player with server id `6`, the resulting value sent to all clients is then `players_checkpoint = {"6": [5, 987654321]}` (`987654321` being the server time).

If you now send `checkpoint` with value `4` from another player with server id `12`, the value is updated to `players_checkpoint = {"6": [5, 987654321], "12": [4, 987655557]}`.

By providing an empty string for `value`, you will remove the entry for the player.