Cosmos Cove Control Panel

for SpacetimeDB Version 0.4.2 - Online Documetation

Thank you for your interest in using Cosmos Cove Control Panel for SpacetimeDB!

My wish is for this addon is to simplify anyones journey into creating their own MMO using the new combined database and server system developed by Clockwork Labs. Gone is the necessity of having to setup individual SQL services or to construct complicated server logic relationships across a dozen different systems to be able to run your game or app.
With SpacetimeDB you write your multiuser logic with both database and server logic in the same module, and then publish it to your server. This means fantastic performance, prototyping speed and even allows live hot updates while users are connected - seeing the new changes in realtime.

Initially this asset focuses on all developers using Windows for Unity game or application development and with Windows Subsystem for Linux (WSL) running Debian as a local SpacetimeDB server and CLI. This then allows you also to publish to a custom remote Debian server via the Custom Remote mode, or to use Maincloud which is the official paid hosting solution by Clockworklabs (it's free to use for the time being). This asset allows you to install everything needed to get your server running in an easy and intuitive installation process - whether locally or remotely.

Pre-requisites

Windows 10 or 11	**WSL with Debian** *(auto installer)*
Unity 6000.2.1f1 (any render pipeline) or Unity 2021.2+ (any render pipeline)	*Debian 13 Trixie* *Curl* *SpacetimeDB* *SpacetimeDB service (standard systemd)* *Rust (optional, recommended)* *Wasm-Opt (optional, recommended)*

Note! Use version control and notify me if you find any bugs. All important actions like modifying any data will display confirmation dialogs before proceeding.

Cosmos Cove Control Panel features an automatic **Installer Window** that should get everything needed installed on your Windows PC. Select it from the Welcome screen or through Unity´s toolbar **SpacetimeDB > Installer Window** to get started.

In case the automatic **Installer Window** won't work for your configuration, you can try the manual steps below to get your SpacetimeDB server running in a fresh Unity project.

Quick Start

1. A Welcome window will appear. Click on **Main Window**.
   In **Pre-Requisites** enter your settings for the **WSL Local** server which you need to create in order to have a local SpacetimeDB CLI.
   - **Client Language** is recommended to be C# for Untiy.
   - **Client Path** is where SpacetimeDB will autogenerate local bindings scripts in the Client Language. You can create or choose any directory in your Assets folder. You won't edit these directly, but will refer to the results in your scripts.
   - **Module Language** is recommended to be Rust but C# is also supported.
   - **Module New Entry** is where you enter the name of your new SpacetimeDB module and press **Add**. A module is a self-contained bundle of server logic that can be the entire server or you can have several, for example one per responsibility. Start with creating just one. If you already have a module since before you can just enter the name and location of it when you press **Add**. **Don't Init New Module** before having the WSL Local server installed which is the next step.
   - **Debian Username** is the same user name you are going to use after installing Debian in the next step.
   - **URL** is the URL of your local machine. The default setting is fine.
   - **Auth Token** is only required after server installation to get full access to it.

2. Open **Installer Window** (or follow the Manual Install Process below) and start from the top to **Install** each item. This will install WSL and all necessary software on your WSL instance to be able to run SpacetimeDB.
   - Command line windows will appear showing the progress and when finished requires a key input to close.
   - **Install WSL with Debian** asks for a UNIX username and password when it is successfully finished. Trixie is now included by default in the Debian install.
   - **Either Rust or .NET SDK (C#)** depending on your chosen server language.
   - Install all available software for WSL Local.

   Click **Check Pre-Requisites to Start SpacetimeDB** and you will be asked to **Init Module and Refresh Login** so that your module is set to its default configuration and your default offline identity login and token are replaced with SpacetimeDB's official authentication and recoverable if you forget to backup the key.

3. **Start SpacetimeDB WSL** and **Publish** your default module. The first publish takes longer since it needs to compile dependencies. After a successful Publish you can now **Edit Module** and see the results in the **Logs**, **Database** and **Reducer** windows after every new Publish.
   Best of luck with your MMO development!

Tip! Even if you wish to only run a **Custom Remote** or **Maincloud** server you still need to first install a **WSL Local** server. You need a local SpacetimeDB CLI to be able to publish and generate SpacetimeDB scripts and push them to the server.

Manual Install Process

For clarity and in case the automatic installer doesn't work for you, here are the manual steps of what the **Installer Window** accomplishes behind the scenes.

To install Windows Subsystem for Linux (WSL) is the first and most important step, because it enables Virtual Machine Platform hypervisor, which can be incompatible with WMWare and VirtualBox. If you use any other virtualization on your Windows, please be sure to check your compatibility with WSL1 and WSL2 before continuing.

The **Installer Window** of Cosmos Cove Control Panel features a quick compatibility report that checks if your PC has the necessary features to enable WSL2. Even if you are planning to install everything manually, you can still launch only that tool by selecting "Install WSL with Debian" and see the result. You have to confirm before the actual install begins.

> **Note!** It is recommended to install WSL2 for your system for near native Linux performance. Regardless of chosen WSL, remember to always backup anything important on your PC before continuing. While being an official Microsoft feature it changes system settings that may require some time to restore on an incompatible PC. Check the next chapter **WSL** for common fixes.

Enter the step 1 command into your Powershell to install WSL2 with Debian. Then on step 2 and onwards you will enter them one by one in the Debian CMD window which you launch from the Debian shortcut in your Windows start menu. The "Settings and Debian" foldout in Server Management Panel also has a button to launch your Debian CMD window.

1. **Install WSL with Debian Trixie (13) in Powershell**
   wsl --install -d Debian
   (above is for WSL2, check the WSL chapter below for WSL1 instructions)

   > **Info!** Afterwards launch Debian from your start menu to create your Debian credentials if you weren't asked during the WSL install process. Write down your username so you can enter it into CCCP when prompted.

2. **Optional: (in case the new Trixie version wasn't installed directly)**
   Launch Debian and update Debian Bookworm (12) to Trixie (13)
   sudo apt update && sudo apt upgrade -y
   sudo apt install -y update-manager-core
   sudo sed -i 's/bookworm/trixie/g' /etc/apt/sources.list
   sudo apt update
   sudo apt full-upgrade -y
   wsl.exe --shutdown

   > **Info!** If you attempted to install Trixie with the Installer Window and it didn't finish, you can often finish the install manually with the commands:
   > sudo dpkg --configure -a
   > sudo apt --fix-broken install
   > sudo apt update

```
sudo apt full-upgrade -y
```

3. **Start Debian again and Verify that it´s Trixie**
   cat /etc/os-release

4. **Install cURL**
   sudo apt update
   sudo apt install curl

5. **Install SpacetimeDB with default settings**
   curl -sSf https://install.spacetimedb.com | sh

6. **Add SpacetimeDB to Debian PATH**
   nano ~/.bashrc
   Add this line at the bottom
   export PATH="/home/m/.local/bin:$PATH" (check what it tells you to add at install)
   Restart to use spacetime command

7. **Create SpacetimeDB Services**
   nano /etc/systemd/system/spacetimedb.service
   Add this to the newly created spacetimedb.service file (replace yourusername)
   [Unit]
   Description=SpacetimeDB Server
   After=network.target
   [Service]
   User=yourusername
   Environment=HOME=/home/yourusername
   ExecStart=/home/yourusername/.local/bin/spacetime
   --root-dir=/home/yourusername/.local/share/spacetime start
   --listen-addr=0.0.0.0:3000
   Restart=always
   WorkingDirectory=/home/yourusername
   [Install]
   WantedBy=multi-user.target
   Ctrl+O and Enter to save. Then Ctrl+X to exit.
   nano /etc/systemd/system/spacetimedb-logs.service
   [Unit]
   Description=SpacetimeDB Database Logs
   After=spacetimedb.service
   Requires=spacetimedb.service
   [Service]
   User=yourusername
   Environment=HOME=/home/yourusername
   Type=simple
   Restart=always
   RestartSec=5
   ExecStart=/home/yourusername/.local/bin/spacetime logs magical -f

WorkingDirectory=/home/yourusername
[Install]
WantedBy=multi-user.target
Ctrl+O and Enter to save. Then Ctrl+X to exit.

sudo systemctl daemon-reload
sudo systemctl enable spacetimedb.service
sudo systemctl start spacetimedb.service
sudo systemctl enable spacetimedb-logs.service
sudo systemctl start spacetimedb-logs.service

8. **Install Rust (If using Rust for server development) Install default if asked**
   sudo apt install build-essential
   curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
   . "$HOME/.cargo/env"

9. **Install .NET SDK (If using C# for server development)**
   sudo apt update
   sudo apt install -y wget
   wget https://packages.microsoft.com/config/debian/12/packages-microsoft-prod.deb -O packages-microsoft-prod.deb
   sudo dpkg -i packages-microsoft-prod.deb && rm packages-microsoft-prod.deb
   sudo apt install -y dotnet-sdk-8.0
   sudo dotnet workload install wasi-experimental

10. **Install wasm-opt (binaryen) (Recommended)**
    wget
    https://github.com/WebAssembly/binaryen/releases/download/version_123/binaryen-version_123-x86_64-linux.tar.gz
    tar -xzf binaryen-version_123-x86_64-linux.tar.gz
    sudo cp binaryen-version_123/bin/* /usr/local/bin/
    Then delete the temporary install files
    ls -la (to list files and) rm -r <file or folder name> (to delete the files)

11. **Install Git (Needed by certain spacetime commands)**
    sudo apt update
    sudo apt install -y git

12. It is good practice to first do a **Logout** command and then **Login** command again after your first install, so that your default offline identity and token are replaced with SpacetimeDB's official authentication and recoverable if you forget to backup the key.

13. You can now do the last step (3) in the Quick Start above to Initialize, Start and Publish your new server module.
    Best of luck with your MMO development!

WSL

If your system is incompatible with WSL2, reported by the compatibility tool in Server Installer Window or by your own research - you can install WSL1 for similar functionality by entering these commands in Powershell (Administrator).

1.

```
dism.exe /online /enable-feature
/featurename:Microsoft-Windows-Subsystem-Linux /all /norestart;
```

2.

```
wsl --set-default-version 1 && wsl --install -d Debian
```

If WSL1 reports as incompatible with messages like:
“WSL1 is not supported with your current machine configuration.”
Restart your PC and try again. It should now work.

> Note! If WSL was installed previously and Windows Subsystem for Linux is still active, a Windows popup may appear when opening the Server Installer Window prompting to install WSL. Disregard it and use the install button to install it.

About WSL2 vs WSL1

WSL2 is the most performant and accurate way to run Debian Linux on your new Windows PC. This alternative is better for new PCs. Memory usage: 1-2 GB	WSL1 is more compatible with different Windows PCs since it doesn’t use virtualization. This alternative is better for older PCs. Memory usage: A few mb

> Note! In case you tried to install WSL2 on an incompatible PC in rare cases it can’t boot normally. A common fix for this is to force-power-off your PC (hold the power button) three times in a row during the logo splash and on the third boot Windows will enter the blue Windows recovery tool called WinRE. Then Troubleshoot → Enable advanced startup. Then on the next boot you select Safe Mode (4). Windows will in most cases fix itself and you can reboot normally.
> You can also enter command prompt in WinRE by going to Troubleshoot → Advanced options → Command Prompt and entering “bcdedit /set hypervisorlaunchtype off”.
> Then you can start normally and install WSL1.

Useful Powershell commands for WSL

Enter your Debian
wsl -d Debian
List the current active distributions on your WSL
wsl --list --verbose

Update WSL
wsl --update

To export a hdd image of your entire WSL Debian
wsl --export Debian C:\Debian.vhdx --vhd

To import a hdd image of your entire WSL Debian
wsl --unregister Debian (this deletes your current Debian)
wsl --import Debian "C:\path\to\install\location" "C:\path\to\your\debian.vhdx"

To export a tar of your entire WSL Debian
wsl --export Debian C:\Debian.tar

Remove Debian
wsl --unregister Debian

Uninstall WSL (first unregister any OSes)
wsl --uninstall

SpacetimeDB as a Service

Services in the Linux systemd are automatically configured if you run the automatic **Server Installer** window. Both WSL Local mode and Custom Remote mode works this way by default.

It is possible to just run SpacetimeDB in the foreground with the **spacetime start** command, but Cosmos Cove Control Panel now only supports running SpacetimeDB as a service. You can always manually do the foreground command to quickly test that it runs provided the Spacetime CLI and WSL are installed.

In all other cases it is recommended to run SpacetimeDB as a service. And this is what the asset does by default. The reasons are: stability, logging and performance.

When we run SpacetimeDB as a service it saves all logs to the journalctl logs of the Debian OS. This generally simplifies things because we can just periodically read the journalctl logs behind the scenes and output them into the Logs window.

Running as a service also allows proper start and stop commands. Otherwise we´d have to kill the process to stop if, if it is running in foreground and not as a service. This improves stability and prevents any data loss.

The location of the service files are here
\\wsl.localhost\Debian\etc\systemd\system

You are supposed to have two SpacetimeDB services
spacetimedb.service (Main service which runs the server and saves module logs)
spacetimedb-logs.service (Database service which only outputs database logs)

Files Created by Cosmos Cove Control Panel

After a successful install these directories and files have been created in your project.

1. **Asset Directory** /Packages/Cosmos Cove Control Panel for SpacetimeDB/
   This directory contains the main scripts that makes the asset function.
2. **Settings file** /Assets/Cosmos Cove Control Panel Settings/CCCPSettings.asset
   This scriptable object gets updated with the settings of your asset. You may move it's location by using the Browse button in **Project Settings**/Cosmos Cove Control Panel.

After an uninstall these files may remain in your project.

1. **Settings file** /Assets/Cosmos Cove Control Panel Settings/CCCPSettings.asset
   If you later want to install this asset again you can backup or keep the settings asset so the settings are applied on the new install.

Workflow

Once you have entered all pre-requisites, installed all software and the server is running - here is a brief overview of a workflow suggestion.

1. **Code your server**'s lib.rs (for Rust) or lib.cs (for C#) inside your module directory by pressing **Edit Module** in the **Main Window**.
   Note: Much of same rules of any programming applies to writing server code. You may very soon need to create individual scripts for every responsibility of the server such as chat.rs, authentication.rs, movement.rs, etc. Your lib.rs will then refer to these scripts which can be placed in subfolders in the servers /src directory. And you make changes to the individual scripts instead of lib.rs.

2. **Save your edits** in your IDE of choice (such as VS Code) and return to Unity. Cosmos Code Control Panel will detected if any server code file was changed and will notify you that you should **Publish** to see your changes. By default its set to also auto **Generate** the local client code that interface with the server code, but you can choose to have separate Publish and Generate for more control in Settings. There is also an auto-publish mode that automatically Publishes/Generates on detected server changes if you wish to automate it when you return to Unity. Keep in mind that Generate needs to call an editor compile when finishes for the scripts be available.

3. **The changes are live** on a successful Publish of your server and you can try it out by playing your game or by running a reducer (server method) in the **Run Reducer** window.

   Repeat :)

This is the magic of SpacetimeDB. Your entire server becomes updated by unified Rust scripts which both determines the server logic through reducers and the server databases through tables. You can even do many Publishes while the server is running and players are connected to see the results live.

It is recommended to use a Rust analyzer in your IDE to catch any errors before attempting to Publish. In case your edits resulted in an error the Command Output will display the reason of the error when it tries to compile the Rust code. You may get assistance using your AI coding tool of choice to query the error message and your code to get a solution.

Tip! Read the SpacetimeDB official documentation and check the example projects and tutorial videos for more details. Links are in the Useful Links chapter below.

Export Database

Currently Cosmos Cove Control Panel supports exporting **CSV** or **JSON** database files in the **Database Window**. This will give you .csv or .json text files for every public table in your database including all the database fields in its current state.

When exporting CSV files you get a choice of delimiter. A delimiter is by which text character the data is separated in the CSV database representation. Excel of different regions support different delimiters, so check which delimiter your regions Excel (or other CSV app) support in order for the data to be correctly separated when opening the text documents.

JSON is rarely used but is included as a backup format since it is the source file format that SpacetimeDB can output. CCCP converts the JSON to CSV when choosing to export CSV.

Import Database

Note! Make sure you have backed up your current database before import.

Currently Cosmos Cove Control Panel only supports importing CSV or JSON database files in the **Database Window** through a custom reducer in lib.rs. The reducer needs to have matching entries for the tables that are in your database.

Here is an example reducer that works for my project:
https://github.com/strosz/com.northernrogue.cccp.spacetimedbserver/blob/main/Samples/import_reducer_example.rs

Integrate a similar server code into your project keeping the reducer name with expected variables since this is what the client code expects.

```
pub fn import_table_data(ctx: &ReducerContext, table_name: String, json_data: String) -> Result<(), String>
```

Ask your LLM of choice to transform the import_reducer_example.rs to support all your project´s tables and features after you have pasted it into your own lib.rs, or rust module.

Any new database changes you make to tables after that will also need an updated reducer for the import to work. It's quick and easy to ask your LLM to update it.

You can then use the "Import from Folder…" for an entire folder of tables and "Import Single File…" for a single table in the Database Window.

The Database Window and the import process only supports public tables.

This feature is subject to change and may be improved in the future.

ClockworkLabs has announced that they are releasing their own fully featured database tool, and for this reason the development of the Server Database Window won't continue until that tool is released and we can possibly make Cosmos Cove Control Panel have an option to open that database tool.

SpacetimeDB Tips and Tricks

Tables

SpacetimeDB does not support changing existing tables while preserving the data. When you need to change an existing table you should instead use the following workflow.

1. Create a new table with the new row/rows.
2. Write a migration reducer.
3. Run the reducer with **Run Reducer** window which moves the existing data to the new table - handling the new row/rows.

Example of a migration reducer in Rust from the table message to the new table messages where “id” row was added

```rust
// Migration reducer to move messages from old to new table
#[spacetimedb::reducer]
pub fn migrate_messages(ctx: &ReducerContext) -> Result<(), String> {
    log::info!("Starting message table migration...");
    let mut migrated_count = 0;

    // Iterate through all messages in the old table
    for old_message in ctx.db.message().iter() {
        // Insert into new table with auto-incrementing ID
        ctx.db.messages().insert(Messages {
            id: 0, // Will be auto-assigned
            sender: old_message.sender,
            sent: old_message.sent,
            text: old_message.text.clone(),
            sender_name: old_message.sender_name.clone(),
        });
        migrated_count += 1;
    }

    log::info!("Migration complete. Migrated {} messages.", migrated_count);
    Ok(())
}
```

If you wish to delete a table, you have to first make a backup of all data for all tables. You can do this with the export to JSON or CSV command in the **Database Window**. Then Ctrl+Alt publish your module to clear all data. This will update the tables if you removed any, and then you can re-import all data.

You can always Ctrl+Alt publish your module to reset your database and this will make all your breaking changes be accepted by the server.
This will delete all data, so always have a backup before!

Authentication

SpacetimeDB uses **cryptographic identities** where each client has a unique public/private key pair (identity) tied to an authentication token. The server remembers which identity owns each database and enforces access control accordingly.

Local WSL Initial Setup

When you first install SpacetimeDB CLI in WSL, the CLI may create a local / server-issued identity by default; that identity is local unless you explicitly register it with the global auth server via the login flow. This default offline identity:

- Works perfectly for local development and testing
- Can access local SpacetimeDB instances and run reducers
- But, it is not registered with SpacetimeDB's official authentication servers
- Cannot be recovered if your local environment is reinstalled, unless you backup the \\wsl.localhost\Debian\home\<UserName>\.config\spacetime directory.

After installation do a Logout and Login cycle. This registers your identity with SpacetimeDB's official authentication servers, making it recoverable and portable across machines.

Custom Remote Server

When you publish your local module to a remote SpacetimeDB server, the server registers your current local WSL SpacetimeDB identity (not just the token) as an authorized user.

- Your specific identity becomes the "owner" of databases you create.
- The server remembers your identity's public key for future authentication.
- You gain full access to your server.

Important: It's the cryptographic identity, not just the token, that determines access rights.

How to regain access to a Custom Remote server after Local WSL reinstall

If you reinstall your local environment, you get a new cryptographic identity that the remote server doesn't recognize. Here's the recovery process:

Step 1: Re-authenticate Locally
In your local WSL environment, run:

1. Commands>Logout and then Commands>Login
2. This launches your browser to SpacetimeDB's official authentication server
3. Verify your identity (e.g., with GitHub account)
4. Copy the new auth token from the CLI output to the Pre-Requisites WSL mode.

Step 2: Re-authenticate on Remote Server

1. Run the same Logout and Login process as above.
2. Since the remote server likely lacks a browser, copy the authentication URL from the terminal and paste it into your local browser to complete authentication.
3. Copy the new authentication token to the Pre-Requisites Custom Remote mode and verify access.

Both your local and remote environments now have new identities registered with SpacetimeDB's authentication servers. The server will authorize these new identities to access your existing databases.

Summary of SpacetimeDB Authentication

Identity: Cryptographic public/private key pair (permanent)
Auth Token: Authentication credential token tied to that identity (can be regenerated)

Server-Side Permission Management

- Each database remembers which identity owns it
- Multiple identities can be authorized for the same database
- Server administrators can manage identity permissions
- Identity authorization is separate from token management

Backup and Recovery Strategy

- Backup your \\wsl.localhost\Debian\home\<UserName>\.config\spacetime directory to be safe that whatever the identity you have been working with on your local server - it can be recovered if needed.
- It is the cli.toml file that spacetimedb uses to authenticate. You can modify it or replace it while logged in and it will instantly use the new cli.toml.

Common Autentication Issues

Publishing Errors
"Identity not valid for server" Error

- Indicates identity mismatch between client and server
- Verify both local and remote environments are using authenticated identities
- Solution: Use the Logout and Login cycle described above

Token Expiration

- Solution: Use the Logout and Login cycle described above

Database Log Errors
Error: Identity does not own database, expected: c200cddc223d… 13e529b got: c200e65fe56…38b390ca

- You’re not logged into the correct identity for the database.
- Restore your original cli.toml auth code if you have a backup.
- Solution: Use the Logout and Login cycle described above

Settings

When Cosmos Cove Control Panel starts for the first time it attempts to create a **CCCPSettings.asset** scriptable object asset in the **/Assets/Cosmos Cove Control Panel Settings/** folder.

This path is user customizable in **Project Settings** if you wish to move it. In the Project Settings Cosmos Cove Control Panel category you can view your current settings, set all settings to default or migrate from this assets earlier EditorPrefs settings system.

The **CCCPSettings.asset** contains all the settings that you have set up in this asset. It can be shared to another user which can overwrite or place it in their project and browse to it, in order for them to use all your settings. This is also very useful for setting up this asset quicker in your own alternative PCs aside from your main PC. You can simply load your existing settings from your main PC.

All the changes to settings you make in the different windows of this asset are automatically saved to the settings asset. The Project Settings view is simply for reference aside from the previously mentioned extra features.

In earlier versions, **EditorPrefs** were used for a lightweight settings system, but with the downsides of not being sharable and being global - that is all projects you used on the same Unity Editor version shared the same Cosmos Cove Control Panel settings. The current save system is project specific and dependant on the settings asset file. You migrate from the old EditorPrefs system to the new system automatically if you used an earlier version of this asset. There is also a manual migrate button in the Project Settings view.

A few settings are yet dependent on new project specific EditorPrefs. EditorPrefs can be found on your PC in the registry under HKCU\Software\Unity Technologies\Unity Editor 5.x.

More details on what each setting in this asset does is coming in a future version of this document.

Limitations

Purpose of this asset

This asset provides the necessary UI and workflow for you to code on your own. It enables creation. I won't be able to help you with the MMO coding itself.

SpacetimeDB features official example Unity projects, has announced to open source Bitcraft and keeps an active Discord forum with lots of good advice. Helpful links are provided in the documentation.

SpacetimeDB Limitations in order to use it for Free

This asset enables you to start coding your own online game and server logic using SpacetimeDB by Clockwork labs which is free **for one public instance**. That is one instance of the spacetimedb application. You may have several STDB modules on that server and hardware enabling thousands of CCU (concurrent users) as long as it is one SpacetimeDB instance that other players can connect to. There are no limits on concurrent users as long as it is one instance. **For local testing and using Maincloud there are no instance limitations.** If using remote Maincloud hosting, it may soon require a usage fee which should be reasonable for the data usage but is beyond my control. The server module code you create for any mode can easily be published to any other server mode when you want to switch.

If you wish to have several SpacetimeDB application instances on your own servers which the players can connect to you may contact Clockwork Labs to pay for an enterprise on-premises license.

Third party software

I will help as much as possible with everything related to this asset and to fix any bugs. I can't answer questions about the third party software the Installer Window can install. **The third party software required is not created by me and you will have to refer to the support forums of Rust, WSL, Binaryen etc in case of specific questions about them. They luckily have sprawling communities with lots of information already available.** The Installer Window is simply an easy and intelligent agent that downloads and sets up that open source software for you so you don't have to do it manually saving a lot of time. I will of course work to fix any bug which may be introduced between this asset and any new updates of the required third party software.

Useful Links

Documentation

SpacetimeDB Official Discord
https://discord.com/channels/1037340874172014652/

SpacetimeDB Official Site and Documentation
https://spacetimedb.com/
https://spacetimedb.com/docs

SpacetimeDB Rust Documentation
https://docs.rs/spacetimedb/latest/spacetimedb/

Free to Use Game Frameworks

Blackholio - An example beginner friendly game
https://github.com/clockworklabs/Blackholio

Blackholio - Tutorial video
https://www.youtube.com/watch?v=msVwc0IwYI0

Live SpacetimeDB Examples Made by Clockwork Labs

Bitcraft
https://bitcraftonline.com/

Bitcraft Steam Page
https://store.steampowered.com/app/3454650/BitCraft_Online/