Cosmos Cove Control Panel

for SpacetimeDB Version 0.5.8 - Online Documentation Trello Roadmap

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.

This asset focuses on all developers using Unity for game or application development and with either Docker (Windows, Linux or MacOS) or WSL (Windows) as a local SpacetimeDB CLI and server. This 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). Get your server running in an easy setup process where you are guided on the pre-requisites necessary to get SpacetimeDB running.

Pre-requisites

Windows, Linux or macOS	**Docker Desktop 4.11+** *or*	**WSL with Debian** (Windows)
Unity 6000.2.10f1 or Unity 2021.2+	Docker Desktop SpacetimeDB Docker Image Volume Bindings (auto) **Setup instructions provided (~5 min setup)**	Debian 13 Trixie Curl SpacetimeDB SpacetimeDB service (standard systemd) Rust (optional, recommended) Wasm-Opt (optional, recommended) **Setup instructions provided (~15min setup)**

Cosmos Cove Control Panel features a **Setup Window** that should help you get everything needed set up. Select it from the Welcome screen or through Unity´s toolbar **SpacetimeDB Server Manager > Setup Window** to get started. Use the **Local Docker Setup** for the quickest way to get started. There is also a **Local WSL** and **Remote Custom Setup** with more manual steps. The Github version features a scripted installer for each step, but the Asset Store version needs manual install steps to comply with Asset Store terms and conditions. Advanced users can check the Github repository and copy the GitHub.cs and ServerInstallProcess.cs scripts to the /Editor folder of the package to access the scripted installer for a quicker Local WSL and Remote Custom Setup.

Quick Start with Docker

1. A Welcome window will appear. Click on **Main Window**. In **Pre-Requisites** enter your settings for the **Local Docker** CLI provider and server.
   - **Client Language** is recommended to be C# for Unity.
   - **Client Path** is where SpacetimeDB will autogenerate local binding scripts in the Client Language. You can create or choose any directory in your Assets folder. You won't edit these directly, but will use the methods in your scripts.
   - **Module Language** is recommended as Rust, but C# or TS is also supported.
   - **Module New Entry** is where you enter the name of your new SpacetimeDB module and press **Add**. A module is self-contained 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**. You have to **wait to Init New Module** before having Docker and the SpacetimeDB image installed which is step 2. You will be asked to Init New Module when possible.
   - **URL** is the URL of your local machine. The default setting is usually fine.
   - **Auth Token** is only required after step 3 to get full access to it.

2. Open **Setup Window** and start from the top to **Setup** each item. This will setup Docker and all necessary software to be able to run SpacetimeDB.
   - **Setup Docker Desktop** opens the official Docker homepage where you can download and install Docker Desktop for your chosen OS.
   - **Setup SpacetimeDB Docker Image** provides you with a corrected Docker run command for Docker Desktop terminal accessible at the >_ symbol and a SpacetimeDB homepage link where the official command can be seen. Don't include the last "start" word as you see in the corrected command below.

     ```
     docker run --rm --pull always -p 3011:3000 clockworklabs/spacetime
     ```

     Paste it into the Docker Desktop >_ terminal to download the image and press **Refresh** in the Setup Window to detect it. **Setup Docker Container Volume Mounts** runs automatically after and configures the Docker container.

   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 SSO authentication and recoverable if you forget to backup the authentication key.

3. **Start SpacetimeDB Docker** 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 **Run Reducers** 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 only run a **Custom Remote** or **Maincloud** server you need to **also** install a **Local Docker or WSL** SpacetimeDB CLI to be able to publish and generate and control the server.

Quick Start with WSL

1. A Welcome window will appear. Click on **Main Window**.
   In **Pre-Requisites** enter your settings for the **Local WSL** CLI provider and server which you need to create in order to have a local SpacetimeDB CLI.
   - **Client Language** is recommended to be C# for Unity.
   - **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**. You have to **wait to Init New Module** before having the WSL Local server installed which is the next step.You will be asked to Init New Module when possible.
   - **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 usually fine.
   - **Auth Token** is only required after server installation to get full access to it.

2. Open **Setup Window** and either follow the Manual Install Process WSL below (Asset Store version) and refresh the Setup Window after every install. Or start from the top auto installing each item (Github version). This will install WSL and all necessary software on your WSL instance to be able to run SpacetimeDB.
   - Install all available software for the **Local WSL Setup**.
   - **Install WSL with Debian** asks for a UNIX username and password when it is successfully finished. Remember these credentials and enter the same Debian username in the Pre-requisites.
   - **Either Rust or .NET SDK (C#)** depending on your chosen server language.

   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 SSO 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 **Run Reducers** 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 only run a **Custom Remote** or **Maincloud** server you need to **also** install a **Local Docker or WSL** CLI to be able to publish and generate and control the server.

Manual Install Process WSL

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 those or any other virtualization on your Windows, please be sure to check your compatibility with WSL1 and WSL2 before continuing.

The **Setup Window** of Cosmos Cove Control Panel features a quick compatibility report that checks if your PC has the necessary features to enable WSL2. You can launch that tool by selecting "Check Compability" (Asset Store build) or ""Install WSL with Debian" (Github build) and see the result.

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 **Windows PowerShell** to install WSL2 with Debian. Then on step 2 and onwards you will enter them one by one in a **Debian CMD window** which you launch from the Debian shortcut in your Windows start menu. The Command foldout in the Main Window also has a button **Open Debian Window**.

1. **Install WSL with Debian Trixie (13) in Windows 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 Setup 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!

Manual Install Process Custom Remote Server

Follow these steps to install your SpacetimeDB Custom Remote server over SSH and press **Refresh** in **Setup Window** after every step to check if the install was successful. Simply paste them in your **CMD terminal** after having logged in via SSH on your remote server.

Installing SpacetimeDB on a remote Linux Debian server via SSH is ideal for dedicated servers, cloud VPS instances or production environments.

A simpler Docker based way to install custom servers will soon be added here. It trades some convenience for slightly worse performance. So if you want full performance this is yet relevant.

Prerequisites

Before starting, you'll need:

- A remote Linux server running Debian (Bookworm 12 or Trixie 13)
- SSH client installed on your local machine (built-in SSH for Windows/macOS/Linux)
- Your server's IP address or hostname
- Root access to the server terminal (via your hosting provider's console/VNC)
- An SSH key pair (we'll generate this in the next step)

Important! For security reasons, it's recommended to create a dedicated non-root user with sudo privileges for running SpacetimeDB rather than using the root account directly.

Important! The Cosmos Cove Control Panel uses SSH key-based authentication (not passwords) for secure connections. This guide will show you how to generate and configure SSH keys.

Initial Setup - Generate SSH Keys

If you don't already have an SSH key pair, you need to generate one on your local machine (the PC where Unity/CCCP runs).
Either press **Generate SSH Key Pair** in the Pre-requisite section of Main Window and skip to **Step 0.1** OR do the SSH keygen process manually:

On Windows (PowerShell) Generate a new SSH key pair**:**

ssh-keygen -t rsa -b 4096 -f "$env:USERPROFILE\.ssh\spacetimedb_key"

# When prompted for a passphrase, you can press Enter for no passphrase
# (required for automated SSH connections from CCCP)

This creates two files:
- `C:\Users\YourUsername\.ssh\spacetimedb_key` (private key - keep this safe!)
- `C:\Users\YourUsername\.ssh\spacetimedb_key.pub` (public key - this goes on the server)

### On macOS/Linux:
In bash terminal
# Generate a new SSH key pair
ssh-keygen -t rsa -b 4096 -f ~/.ssh/spacetimedb_key

# When prompted for a passphrase, press Enter for no passphrase

This creates:
- `~/.ssh/spacetimedb_key` (private key)
- `~/.ssh/spacetimedb_key.pub` (public key)

Note: You'll need to enter the private key path in CCCP's Custom Server settings later.

Step 0.1: Configure SSH Access on Fresh Server

On a completely fresh Debian 13 server, you need to set up SSH access using your hosting provider's console/terminal.

Log into your server through your hosting provider's web console, VNC, or KVM access. You should see a terminal prompt.

Step 0.2: Install and Enable SSH Server
# Update package lists
apt update

# Install OpenSSH server if not already installed
apt install -y openssh-server

# Enable and start SSH service
systemctl enable ssh
systemctl start ssh

# Check SSH is running
systemctl status ssh

Step 0.3: Add Your Public Key to Root
Now copy the **public key** you generated earlier (the `.pub` file) and add it to the server:

```
# Create SSH directory for root
mkdir -p /root/.ssh
chmod 700 /root/.ssh

# Create/edit the authorized_keys file
nano /root/.ssh/authorized_keys

# Paste your public key (the ENTIRE content of spacetimedb_key.pub)
# It should look like: ssh-rsa AAAAB3NzaC1yc2EAAAA...very long string...
# Save with Ctrl+O, Enter, then Ctrl+X to exit

# Set correct permissions
chmod 600 /root/.ssh/authorized_keys
```

Windows users: To view your public key, run in PowerShell:

```
Get-Content "$env:USERPROFILE\.ssh\spacetimedb_key.pub"
```

macOS/Linux users: To view your public key:

```
cat ~/.ssh/spacetimedb_key.pub
```

Step 0.4: Configure SSH Settings (Optional but Recommended)

```
# Edit SSH configuration
nano /etc/ssh/sshd_config

# Ensure these settings are enabled (uncomment if needed):
# PermitRootLogin prohibit-password
# PubkeyAuthentication yes
# PasswordAuthentication no  # Disable password auth for security

# Save and restart SSH
systemctl restart ssh
```

Step 0.5: Test SSH Connection

From your local machine, test the connection:

Windows (PowerShell):

```
ssh -i "$env:USERPROFILE\.ssh\spacetimedb_key" root@your-server-ip
```

macOS/Linux:

```
ssh -i ~/.ssh/spacetimedb_key root@your-server-ip
```

If successful, you should connect without being asked for a password!

Step 1: Enter Credentials in CCCP

Now that SSH is configured, you can set up the connection in Cosmos Cove Control Panel:

1. Open Unity and go to **Window > SpacetimeDB Server Manager > 1. Main Window**

2. Change Server Mode to **Custom Server**
3. Enter the following information:
   - **SSH Username**: the username you used when creating the server's Debian install
   - **Server URL**: Your server's IP address or hostname with 3000 as default port
     example: http://192.168.0.247:3000
   - **SSH Private Key Path** (if you wish to set it to another location than the following default):
     - Windows: `C:\Users\YourUsername\.ssh\spacetimedb_key`
     - macOS/Linux: `/home/yourusername/.ssh/spacetimedb_key`

Click **Check Pre-requisites and Connect** to connect the SSH session.

If the connection is successful, you'll see a green **Connected** status!

Step 2: Create a Dedicated User (Recommended)

It's best practice to create a dedicated user for running SpacetimeDB services:

```
# Install sudo if not already installed
apt install -y sudo

# Create new user (you'll be prompted to set a password)
adduser yourusername

# Add user to sudo group
usermod -aG sudo yourusername

# Create SSH directory for the new user
mkdir -p /home/yourusername/.ssh
chmod 700 /home/yourusername/.ssh

# Copy authorized_keys from root to new user (if exists)
if [ -f /root/.ssh/authorized_keys ]; then
  cp /root/.ssh/authorized_keys /home/yourusername/.ssh/
fi

# Set correct ownership and permissions
chown -R yourusername:yourusername /home/yourusername/.ssh
if [ -f /home/yourusername/.ssh/authorized_keys ]; then
  chmod 600 /home/yourusername/.ssh/authorized_keys
fi

# Allow passwordless sudo (optional, for convenience and replace yourusername with your
username)
echo "yourusername ALL=(ALL) NOPASSWD: ALL" > /etc/sudoers.d/yourusername
chmod 0440 /etc/sudoers.d/yourusername
```

Info! After creating the new user, disconnect and reconnect using the new username instead of root.

Step 3: Update to Debian Trixie (13) if not already installed

SpacetimeDB requires Debian Trixie (13) for optimal compatibility.

```
# Check current Debian version
cat /etc/os-release

# If you're on Bookworm (12), upgrade to Trixie (13):

# Step 1: Update package lists
sudo apt update && sudo apt upgrade -y

# Step 2: Install update manager
sudo apt install -y update-manager-core

# Step 3: Change sources to Trixie
sudo sed -i 's/bookworm/trixie/g' /etc/apt/sources.list

# Step 4: Update package lists for Trixie
sudo apt update

# Step 5: Perform full upgrade to Trixie
sudo apt full-upgrade -y
```

Info! If the upgrade process encounters any issues, you can often fix them with:

```
sudo dpkg --configure -a
sudo apt --fix-broken install
sudo apt full-upgrade -y
```

After the upgrade completes, verify the installation:

```
cat /etc/os-release
```

Step 4: Install cURL

cURL is required to download the SpacetimeDB installer:

```
sudo apt update
sudo apt install -y curl

# Verify installation
curl --version
```

Step 5: Install SpacetimeDB Server

Now install SpacetimeDB using the official installer:

```
# Run the SpacetimeDB installer
curl -sSf https://install.spacetimedb.com | sh

# Verify installation
~/.local/bin/spacetime --version
```

Step 6: Add SpacetimeDB to PATH

Make the `spacetime` command available from anywhere:

```
# Edit your .bashrc file
nano ~/.bashrc

# Add this line at the bottom (or use the path shown by the installer)
export PATH="$HOME/.spacetime/bin:$HOME/.local/bin:$PATH"

# Save (Ctrl+O, Enter) and exit (Ctrl+X)

# Reload the configuration
source ~/.bashrc

# Verify the command is now available
spacetime --version
```

Step 7: Create SpacetimeDB SystemD Services

Create systemd services to run SpacetimeDB automatically in the background:

Create the Main SpacetimeDB Service

```
# Create the service file
sudo nano /etc/systemd/system/spacetimedb.service
```

Add the following content (replace `yourusername` with your actual username):

```
[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
```

Save (Ctrl+O, Enter) and exit (Ctrl+X).

Create the Database Logs Service with Dynamic Module Switching

First, create the wrapper script:

```
# Create the wrapper script
sudo nano /home/yourusername/.local/bin/spacetime-database-logs-switching.sh
```

Add the following content

```
#!/bin/bash
# SpacetimeDB Database Logs Service Wrapper
# Dynamically reads current module from config file

set -u  # Exit on undefined variables

CURRENT_MODULE_FILE="/home/yourusername/.local/current_spacetime_module"
SPACETIME_PATH="/home/yourusername/.local/bin/spacetime"
LOG_FILE="/tmp/spacetime-logs-wrapper.log"

# Function to log with timestamp
log_debug() {
   echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" >> "$LOG_FILE"
}

# Function to handle script termination gracefully
cleanup() {
   log_debug "Wrapper script received termination signal"
   exit 0
}

# Set up signal handlers
trap cleanup SIGTERM SIGINT

# Validate config file exists and is readable
if [ ! -f "$CURRENT_MODULE_FILE" ]; then
   log_debug "ERROR: Module config file does not exist: $CURRENT_MODULE_FILE"
   sleep 30
   exit 1
fi

# Read and validate module name
```

```
MODULE=$(cat "$CURRENT_MODULE_FILE" | tr -d '\n\r' | xargs)

if [ -z "$MODULE" ]; then
   log_debug "ERROR: Module config file is empty: $CURRENT_MODULE_FILE"
   sleep 30
   exit 1
fi

# Log start
log_debug "Starting database logs for module: $MODULE"

# Start logging - use exec to replace shell process
exec "$SPACETIME_PATH" logs "$MODULE" -f
```

Save and exit, then make it executable

```
sudo chmod +x /home/yourusername/.local/bin/spacetime-database-logs-switching.sh
```

Create the initial module config file (replace `yourmodulename` with your actual module name):

```
mkdir -p /home/yourusername/.local
echo 'yourmodulename' > /home/yourusername/.local/current_spacetime_module
chmod 644 /home/yourusername/.local/current_spacetime_module
```

Now create the logs service:

```
# Create the database logs service file
sudo nano /etc/systemd/system/spacetimedb-logs.service
```

Add the following content:

```
[Unit]
Description=SpacetimeDB Database Logs
After=spacetimedb.service
Requires=spacetimedb.service

[Service]
User=yourusername
Environment=HOME=/home/yourusername
Type=simple
Restart=on-failure
RestartSec=30
StartLimitIntervalSec=300
StartLimitBurst=3
ExecStart=/home/yourusername/.local/bin/spacetime-database-logs-switching.sh
```

```
WorkingDirectory=/home/yourusername

[Install]
WantedBy=multi-user.target
```

Save (Ctrl+O, Enter) and exit (Ctrl+X).

Enable and Start the Services

```
# Reload systemd to recognize new services
sudo systemctl daemon-reload

# Enable and start the main SpacetimeDB service
sudo systemctl enable spacetimedb.service
sudo systemctl start spacetimedb.service

# Enable and start the database logs service
sudo systemctl enable spacetimedb-logs.service
sudo systemctl start spacetimedb-logs.service

# Check service status
sudo systemctl status spacetimedb.service
sudo systemctl status spacetimedb-logs.service
```

Info! To change which module the logs service follows, simply update the config file:

```
echo 'newmodulename' > ~/.local/current_spacetime_module
sudo systemctl restart spacetimedb-logs.service
```

Install Git (Required for certain SpacetimeDB commands)

```
sudo apt update
sudo apt install -y git

# Verify installation
git --version
```

Final Steps

After installation, it's good practice to authenticate with the official SpacetimeDB authentication server even if you run your own server. This is so you can more easily recover the identity if needed.

Logout and login again to replace offline identity with authenticated token

Info! Your SpacetimeDB server is now running on port 3000. Make sure to configure your firewall to allow connections to this port if you want to access it from external networks.

You can now proceed to initialize, start, and publish your custom remote server module using the Cosmos Cove Control Panel in Unity!

Best of luck with your MMO development!

WSL Info

If your system is incompatible with WSL2, reported by the compatibility tool in Server Setup 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 Setup 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 in WSL

Services in the Linux systemd are automatically configured if you run the automatic **Server Setup Window** (Github build). 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 in WSL. You can always manually do the foreground command to quickly test that it runs provided the SpacetimeDB package 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)

Docker Info

Docker is the quickest way to get SpacetimeDB running on your computer offering both a Command Line Interface (CLI) to manage SpacetimeDB servers and a Local Docker Server. Docker gets you the entire environment necessary and includes everything in the WSL install process in one image.

One common question about Docker containers is persistence. When an image is used to create a container in Docker, that container is ephemeral and if that container for example needs to be reconfigured or updated the container along with all its data is destroyed and created again.

Cosmos Cove Control Panel counters this by automatically setting up Docker to have two persistent volumes of data that will be there regardless of the container state.

These are found in the Docker Desktop **Volumes** tab after a successful container start.

spacetimedb-data
This volume contains all the modules and their databases that you have published to the local SpacetimeDB server.

spacetimedb-auth
This volume contains the active authentication session that was used to access those modules and their databases.

As soon as those volumes have been created on your Docker, that means you are safe and can recreate the container from the spacetime image however you like and your data will be safe.

The Backup and Restore commands in Docker CLI mode found in the Commands section of the main window will tar backup the data volume and restore it, but auth is currently not in the backup for security reasons. If you restore data you may have to re-authenticate with the official SpacetimeDB SSO by logging out and logging in again.

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 often even Publish 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.

Dev Mode Database

This asset supports to set the database in a dev-mode state by going to the Database window and switching the Dev Mode toggle on.

This will make all private tables public for easier development access. With dev-mode on you will be able to browse all private tables in the Database window as any normal public table.

In order for your server code to be compatible with dev-mode you have to ensure that all tables that you wish to normally keep private and yet support switching automatically to public in dev-mode, to be declared in the following way:

```
#[cfg_attr(feature = "dev-mode", spacetimedb::table(name = person, public))]
#[cfg_attr(not(feature = "dev-mode"), spacetimedb::table(name = person))]
```

Instead of the default

```
#[spacetimedb::table(name = person)]
```

FAQ

Database

My table isn't listed

This happens by default even with the default Person sample table from the Init New Module process. The reason is that for security reasons the table is private by default. Only public tables can be queried by the HTTP API that the Database Window reads. You can use the dev-mode feature to easily switch tables between private and public without needing to update your server code. Read more in the **Dev Mode** chapter. How to make the default table public in the Rust module language:

```
#[spacetimedb::table(name = person, public)]
pub struct Person {
    name: String
}
```

Removing a table caused an error message on the next publish

This is because Unity may take a moment to recognize the newly generated client bindings files. Switch to a different window and return to Unity and it should update.
A sample of an error message:
"error CS2001: Source file 'C:\.../DevExportAllTables.g.cs' could not be found."

Publish

Errors on Publish and Solutions

```
"Error: error decoding response body
Caused by:
    unknown variant `migrate_plan`, expected `AutoMigrate` or
`ManualMigrate` at line 1 column 15"
```

This is usually seen when the SDK and Local CLI SpacetimeDB version or Remote Custom SpacetimeDB version (if using Remote Custom server) is not matching. Confirm that they are on the same version.

```
"Error: Pre-publish check failed with status 500 Internal Server
Error: abi version 10.2 is not supported (host implements 10.1)"
```

Abi version mismatch errors are a known issue when updating to new SpacetimeDB versions.
Solution: Run Clean Temp Files from the command section of the main window or manually run **cargo clear** on your local CLI Provider and it will remove old dependency build versions that may interfere with the publishing process.

Publish Shows Permission Errors or Most Obscure Errors
Solution: Often solved by running Clean Temp Files from the command section of the main window. You may only run Clean Cargo and it should fix it.

Scripts

SpacetimeDBClient.g.cs shows errors in the Unity console

Run Clean Temp Files from the command section of the main window. Many errors of SpacetimeDBClient.g.cs can be due to new updates introducing new features and we need to clear the client bindings and publish the server again to fix this. Also make sure you are using the latest SpacetimeDB SDK.

Recommended folder structure

/Assets/ClientGeneratedBindings
/Library
/Packages
/Server

Client Directory: /Assets/ClientGeneratedBindings
Here is where Unity can save the generated bindings .cs files that your scripts can reference.

Server Directory: /Server (outside of assets folder)
Here is where we keep the module server code in the chosen language after **Add** and **Init**.
You can also use subfolders like /Server/Module1 etc. if you plan to have several modules.

Log Window

Why doesn’t my Database logs show?

Sometimes after switching server modes another new line of logs needs to load to show the previous line. You can trigger a Debug reducer or similar to get the database to output new content so the Log Window detects it.

If you see the error message “identity … does not own the database” as the status message at the bottom of the Log Window it means that your login and authentication is not valid for the database you’re trying to see the logs of.
Solution: Logout and Login again so that the SSO launches allowing you to login online.

Files Created by Cosmos Cove Control Panel

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

1. **Asset Directory** at /Packages/Cosmos Cove Control Panel for SpacetimeDB/
   This directory contains the main scripts that makes the asset function.
2. **Settings file** at /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.
2. **Any user created folders like client and server folders.**

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 Docker or WSL Initial Setup

When you first install SpacetimeDB CLI in Docker or 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 to access the database if your local environment is reinstalled, unless you backup the \home\<UserName>\.config\spacetime directory.

After installation always 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 \home\<UserName>\.config\spacetime directory in either Docker or WSL to be safe that whatever the identity you have been working with on your local server it can be recovered if needed.
- The cli.toml file is what 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: Logout and Login again

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

- You're not logged into the correct identity for the database.
- Solution: Restore your original cli.toml auth code if you have a backup or try to Logout and Login again

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 Setup 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 Setup 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 Site and Documentation
https://spacetimedb.com/
https://spacetimedb.com/docs

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

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

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

SpacetimeDB Games

Bitcraft
https://bitcraftonline.com/

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