

**ALP Documentation and Guide**

Contents

[First Run: **Servers Startup.bat** 5](#_Toc207721113)

[Add your own server to the suite: **Add Server.py** 6](#_Toc207721114)

[Add a new Download to ALP’s internet page for clients: **Add Download.py** 7](#_Toc207721115)

[Add a game server status checker to the website: **Add Server Status checker to site.py** 8](#_Toc207721116)

[Game Server Setup Wizard - **Server\_selector.py** 10](#_Toc207721117)

[Overview 10](#_Toc207721118)

[Main Functional Components 10](#_Toc207721119)

[Constants and Folder Setup 10](#_Toc207721120)

[Key Functions 10](#_Toc207721121)

[GUI Class: ALPGameWizard(tk.Tk) 11](#_Toc207721122)

[How This Fits Into Automated LAN Party (ALP) 12](#_Toc207721123)

[Requirements and Dependencies 12](#_Toc207721124)

[Running the Script 12](#_Toc207721125)

[Launcher for the Automated LAN Party environment: **launch\_all\_servers.bat** 13](#_Toc207721126)

[Purpose 13](#_Toc207721127)

[Script Flow and Key Sections 13](#_Toc207721128)

[Integration with Automated LAN Party Suite 14](#_Toc207721129)

[Team Fortress 2 example: **Launch\_TF2\_Server.bat** 15](#_Toc207721130)

[Purpose 15](#_Toc207721131)

[Detailed Breakdown 15](#_Toc207721132)

[Integration Notes 16](#_Toc207721133)

[ALP Game Server Setup Wizard: **Add Server.py** 17](#_Toc207721134)

[Purpose 17](#_Toc207721135)

[Key Features 17](#_Toc207721136)

[How to Use 17](#_Toc207721137)

[Detailed Operation Description: **Add Download.py** 19](#_Toc207721138)

[Initialization and Setup 19](#_Toc207721139)

[Step 0 — Welcome Screen 19](#_Toc207721140)

[Step 1 — Download Details 20](#_Toc207721141)

[Step 2 — Select Files 20](#_Toc207721142)

[Step 3 — Summary & Submit 21](#_Toc207721143)

[Submission Process (submit method) 21](#_Toc207721144)

[Additional Methods 22](#_Toc207721145)

[Execution 22](#_Toc207721146)

[Overall Structure 23](#_Toc207721147)

[Key Components and Workflow 23](#_Toc207721148)

[Initialization and Setup 23](#_Toc207721149)

[Scrollable Container 24](#_Toc207721150)

[Input Fields 24](#_Toc207721151)

[Dynamic UI Behavior 24](#_Toc207721152)

[Database Interaction 24](#_Toc207721153)

[Icon Extraction and Processing 24](#_Toc207721154)

[Input Validation 25](#_Toc207721155)

[Script Generation 25](#_Toc207721156)

[Overwrite Handling 25](#_Toc207721157)

[Post-Generation Actions 25](#_Toc207721158)

[Status Feedback 25](#_Toc207721159)

[Server side web development: **PHP\_Template.php** 27](#_Toc207721160)

[Purpose: 27](#_Toc207721161)

[Configuration: 27](#_Toc207721162)

[Database Interaction: 27](#_Toc207721163)

[Logic: 27](#_Toc207721164)

[Output: 28](#_Toc207721165)

[Icon and Image Paths: 28](#_Toc207721166)

[Usage Notes: 28](#_Toc207721167)

[Error Handling: 28](#_Toc207721168)

[Dependencies: 29](#_Toc207721169)

[Game Servers Detection: **PY\_Broadcast\_Template.py** 30](#_Toc207721170)

[Purpose: 30](#_Toc207721171)

[Configuration: 30](#_Toc207721172)

[Core Functionality: 30](#_Toc207721173)

[Error Handling: 32](#_Toc207721174)

[Output: 32](#_Toc207721175)

[Dependencies: 32](#_Toc207721176)

[Usage Notes: 32](#_Toc207721177)

[Game Server Status Monitoring: **PY\_EXE\_Template.py** 33](#_Toc207721178)

[Purpose: 33](#_Toc207721179)

[Configuration: 33](#_Toc207721180)

[Core Functionality: 33](#_Toc207721181)

[Database Schema: 34](#_Toc207721182)

[Error Handling: 35](#_Toc207721183)

[Output: 35](#_Toc207721184)

[Dependencies: 35](#_Toc207721185)

[Tunez Project Documentation 36](#_Toc207721186)

First Run: **Servers Startup.bat**  
The servers startup folder (Shell:startup) has a .bat file that will check if it’s the first time it’s been ran, and if so launch the server renaming and selection python GUI full screen as well as the mySQL server. Within this screen you can click any games you wish to enable this .bat file to start at server startup, as well as what you want to name this server so the intranet page, and game services will match, and the discord URL of the group using it which will change that URL on the intranet page hosted by the server.

**Note**: this can be reopened later after reboot to change these settings under

**C:\Servers\Initial Server Setup.lnk**

or the actual file location:

**C:\Games\Batch\_Files\ Startup\Server\_selector.py**

Add your own server to the suite: **Add Server.py**  
This program will guide you through the process of adding a new game server, or service to ALP including making shortcuts in a new folder within C:\Servers, putting the correct batch file in C:\Games\Batch\_Files which link to that new shortcut and adding a line to call that batch file in the startup batch file.

Add a new game server doing the following steps:

1. Put your working validated dedicated server software folder within C:\Games
2. Follow the prompts to add the server’s exe, or .bat
3. Add any supporting configuration files you want easy access to make shortcuts for  
   A screenshot of a computer

   AI-generated content may be incorrect. A screenshot of a computer

   AI-generated content may be incorrect.  
   A screenshot of a computer

   AI-generated content may be incorrect. A screenshot of a computer

   AI-generated content may be incorrect.

# Add a new Download to ALP’s internet page for clients: **Add Download.py**

C:\Servers\Website\Add Download.lnk

or

C:\wamp64\www\Python\_Scripts\Templates\Add Download.py

This GUI will aid you in putting a new download within your intranet downloads page for a game or program. The .zip you intend to share, as well as a header image if desired must be first placed within C:\wamp64\www\wp-content\uploads as directed by the program before continuing:

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.  
  
After doing so checking the games download we see now:  
A screenshot of a computer

AI-generated content may be incorrect.

# Add a game server status checker to the website: **Add Server Status checker to site.py**

C:\Servers\Website\Add Server Status checker to site.lnk

or

C:\wamp64\www\Python\_Scripts\Templates\Add Server Status checker to site.py

This GUI will give you a step-by-step walkthrough of how to update the server status page, and provde you with 2 files at the end, a .php document you can copy into the wordpress code snippets, and then apply that WP code snippit ID to the bottom of your server status page, as well as the .py which will do the scanning of the network / local EXE to populate the database with the server status.  
  
A screenshot of a computer

AI-generated content may be incorrect.   
  
The server query hexes were found within wireshark to see what a real client sends to a real server when asking for details for LAN servers on the network. If these are unknown you can simply use the EXE checker, where if the server EXE is running, we mark the game as online.

**TECHNICAL OUTLOOKS ON HOW THESE FILES WORK:**

# Game Server Setup Wizard - **Server\_selector.py**

## Overview

**This Python script provides a GUI-based wizard to assist users in registering dedicated game servers into the Automated LAN Party (ALP) environment on Windows. It helps users specify server details, create shortcuts, batch launch scripts, and organize server-related files in a standardized directory structure.**

**The tool also manages folder icons and launch batch files to streamline launching dedicated servers during LAN parties.**

## Main Functional Components

### Constants and Folder Setup

* **BATCH\_DIR: "C:\\Games\\Batch\_Files" — the directory where batch files for launching servers are stored.**
* **STARTUP\_BATCH: The main batch file launch\_all\_servers.bat located inside the Startup subfolder under BATCH\_DIR.**
* **The script ensures these directories exist, creating them if necessary.**

### Key Functions

**create\_shortcut(target\_path, shortcut\_path, arguments="", icon\_path=None)**

**Creates a Windows shortcut (.lnk) at shortcut\_path pointing to target\_path with optional command line arguments and icon.**

* **Uses Windows Script Host COM (WScript.Shell).**
* **Sets the working directory to the folder containing the target.**

**extract\_icon\_from\_exe(exe\_path, save\_to)**

**Attempts to extract the first icon from an .exe file and save it as a bitmap at save\_to.**

* **Uses Windows API (win32gui, win32ui, win32api).**
* **Returns True on success, False otherwise.**

**set\_folder\_icon(folder\_path, icon\_path)**

**Sets a custom icon for a folder by writing a desktop.ini file with the icon resource and setting proper attributes (+h, +s on desktop.ini, and read-only on folder).**

* **Optionally refreshes the folder icon in Explorer (may be unreliable or omitted).**

**create\_game\_entry(game\_name, ram, vram, server\_path, launch\_args, icon\_path, supporting\_files)**

**The core function that sets up a dedicated game server entry, doing the following:**

1. **Create Server Directory:  
   Under C:\Servers\<game\_name\_no\_spaces>\.**
2. **Handle Icon:**
   * **If the server executable (.exe) or batch file (.bat) has no icon specified, attempts to extract from the executable or copy the .ico file.**
   * **Sets folder icon using set\_folder\_icon.**
3. **Create Server Shortcut:  
   Shortcut named <game\_name> Server.lnk in the server directory, pointing to the server executable with any launch arguments.**
4. **Create Shortcuts for Supporting Files:  
   Additional shortcuts for any user-selected supporting files.**
5. **Create Launch Batch File:  
   A .bat file in BATCH\_DIR to launch the dedicated server, including logic to skip launching if already running.**
6. **Update Startup Batch File (launch\_all\_servers.bat):  
   Inserts a call to the new game's launch batch in a marked user section, preventing duplicates.**
7. **Prints confirmation when setup is complete.**

## GUI Class: ALPGameWizard(tk.Tk)

**Implements a multi-step wizard UI using Tkinter, guiding the user through:**

**Steps (Frames)**

1. **Welcome Screen:  
   Explains purpose and points user to where server files should be placed (C:\Games).**
2. **Game Information:  
   Input fields for:**
   * **Game Name**
   * **Expected RAM (MB)**
   * **Expected VRAM (MB)  
     Validates RAM/VRAM inputs to only allow digits.**
3. **Dedicated Server Path:**
   * **File picker for .exe or .bat server executable.**
   * **Depending on file type, enables or disables:**
     + **Launch arguments input (for .exe only).**
     + **Icon selection (for .bat only).**
4. **Supporting Files:  
   Optionally add any number of supporting files (e.g., config, scripts), showing them in a list.**
5. **Summary & Finish:  
   Shows all entered data and supporting files for confirmation.  
   Clicking Finish calls create\_game\_entry() with collected data, shows a confirmation popup, and closes the wizard.**

**Navigation**

* **Back and Next buttons navigate between steps.**
* **Internal management of frame packing/unpacking to show only one step at a time.**

## How This Fits Into Automated LAN Party (ALP)

* **Automates setting up game servers for LAN parties by:**
  + **Organizing server files in C:\Servers\GameName**
  + **Creating shortcuts with proper icons**
  + **Managing batch scripts to launch servers and prevent duplicates**
  + **Integrating servers into a master startup batch to launch all servers easily**
* **Provides a user-friendly GUI so users don't have to manually create shortcuts or batch scripts.**
* **Centralizes server management to a known folder structure and launch system.**

## Requirements and Dependencies

* **Python packages:**
  + **tkinter (built-in)**
  + **pywin32 (for COM and Windows API calls: win32com.client, win32gui, win32ui, win32api)**
* **Windows environment (for shortcuts, folder icons, batch files)**
* **The folder C:\Games and C:\Servers should exist or be writable**

## Running the Script

* **Run python <script\_name>.py to launch the wizard.**
* **Follow the steps to input game server info and supporting files.**
* **Finish to generate shortcuts and batch files.**
* **Use the launch\_all\_servers.bat in C:\Games\Batch\_Files\Startup\ to launch all registered servers at once.**

# Launcher for the Automated LAN Party environment: **launch\_all\_servers.bat**

## Purpose

**This batch script acts as the main launcher for the Automated LAN Party environment, sequentially starting required services, essential programs, and all registered game servers.**

**It supports a “first boot” logic to run database and server selector initialization on first execution and then proceeds to launch all servers and services on subsequent runs.**

## Script Flow and Key Sections

**1. FirstBoot Check**

* **Variable FirstBoot controls whether this is the initial launch.**
  + **If set to "1", the script launches the MySQL database server and the Python-based Server\_selector.py script, both essential startup components.**
  + **If set to "0", skips these and proceeds directly to launching servers.**

**2. Launching Database and Server Selector (If First Boot)**

* **Starts MySQL daemon (mysqld.exe) minimized.**
* **Waits 10 seconds for database to start.**
* **Calls the Python server selector script (Server\_selector.py), presumably for initial server setup or configuration.**
* **Then proceeds to the next steps.**

**3. Main Server and Service Launch Section**

* **Displays messages to the user about the launching process.**
* **Calls a series of batch files and Python scripts, each responsible for starting specific game servers or services.**
* **Each line follows the pattern:**

**pgsql**

**CopyEdit**

**REM Name="<Game or Service Name>" Ram="<RAM Allocation>" VRAM="<VRAM Allocation>" RG="<Optional Game Group>"**

**call "<Batch File or Python Script>"**

* **Examples include launching:**
  + **Web service (launch\_wamp.bat)**
  + **TeamSpeak 3 (launch\_teamspeak.bat)**
  + **Various Call of Duty variants (launch\_cod4 - No Bots.bat, etc.)**
  + **Popular games like Minecraft, Halo CE, Valheim, etc.**
  + **Custom tools and servers (launch\_PVPgn.bat, launch\_STBC.bat, etc.)**
* **Also includes a line to open a web admin shortcut (ALP Admin.url).**

**4. User-Added Games Section**

* **Marked by these comments:**

**ini**

**CopyEdit**

**REM =================USER ADDED GAMES START=========================**

**REM =================USER ADDED GAMES FINISH=========================**

* **The Python wizard you showed earlier inserts calls to new game launch batch files here dynamically.**
* **Ensures new servers registered by users are included without overwriting the standard servers.**

**5. Final Messages and Exit**

* **After launching all servers and services, prints confirmation that all servers have been launched.**
* **Waits 10 seconds before closing, allowing users to read any messages.**
* **Ends the script cleanly.**

## Integration with Automated LAN Party Suite

* **The Python wizard’s create\_game\_entry function modifies this file by inserting calls to user-added games between the “USER ADDED GAMES” markers.**
* **This keeps the list of launched servers dynamic and extensible without manual batch editing.**
* **The FirstBoot logic supports initial environment setup (like database start) only once, preventing repeated launches of those services.**
* **The batch files called (launch\_<game>.bat) are generated or maintained separately, and are responsible for launching the actual dedicated servers.**

**Notes**

* **The script uses timeout /t <seconds> /nobreak > nul for delays between launches to allow processes to initialize properly.**
* **The use of call ensures each batch file completes before moving on.**
* **The Python script is called directly via call (which works if .py is associated with Python on the system).**
* **The script assumes all paths and batch files exist as specified; missing files may cause errors.**

# Team Fortress 2 example: **Launch\_TF2\_Server.bat**

## Purpose

**This batch script launches a Team Fortress 2 (TF2) dedicated server with the following features:**

* **Automatically finds an available UDP port starting from 27015 to avoid conflicts.**
* **Ensures Steam client is running and up-to-date by launching it minimized if necessary.**
* **Detects the appropriate local IP address to bind the server.**
* **Launches the TF2 server executable with correct parameters for network and game settings.**
* **Provides user feedback about ports, IPs, and status.**

## Detailed Breakdown

**1. Script Directory Setup**

* **Uses CD /D "%~dp0" to ensure all relative commands run from the script’s own directory.**

**2. Port Detection**

* **Starts checking from UDP port 27015 upward.**
* **Uses PowerShell to query currently open UDP ports via Get-NetUDPEndpoint.**
* **If a port is already in use:**
  + **Lists the PIDs using it via netstat.**
  + **Retrieves and displays the corresponding process names.**
  + **Increments the port and checks again.**
* **Stops once an available UDP port is found, saving it in %PORT%.**

**3. Steam Launch & Update Check**

* **Queries Windows Registry to find Steam’s install path.**
* **If Steam is not found, skips launching it.**
* **Checks if Steam is already running:**
  + **If running, skips launch.**
  + **If not running, launches Steam minimized with silent parameters to force updates.**
* **Waits for steamwebhelper.exe process to appear (indicating Steam is loaded).**
* **After detecting Steam WebHelper, waits a few seconds then forcibly closes Steam to avoid having it running unnecessarily.**

**4. Local IP Address Detection**

* **Parses output of ipconfig to find all IPv4 addresses and their associated default gateways.**
* **Chooses the first IPv4 address that:**
  + **Has an associated gateway.**
  + **Is not in APIPA range (169.254.\*.\*).**
  + **Is not in private 172.16.0.0/12 subnets reserved for other purposes (skips 172.16.\*.\* to 172.31.\*.\*).**
* **If none found, defaults to binding to all interfaces (i.e., no specific IP).**

**5. TF2 Dedicated Server Launch**

* **Runs srcds2.exe (the TF2 dedicated server executable) with these parameters:**
  + **-console and -usercon to enable console input and user console.**
  + **-game tf specifies Team Fortress 2.**
  + **+sv\_pure 1 enforces pure server rules.**
  + **+map ctf\_2fort sets the starting map.**
  + **+con\_logfile "console.log" to log server console output.**
  + **+maxplayers 32 to limit max players.**
  + **-port <detected\_port> the first available UDP port found.**
  + **+ip <detected\_ip> if a suitable IP was detected.**
* **Launches the server process and detaches from the batch script.**

**6. Script End**

* **Displays a message indicating the server will close the console window in 5 seconds.**
* **Waits 5 seconds (timeout) before script exits.**
* **Uses endlocal to clean environment variable changes.**

## Integration Notes

* **This script is designed to be called by the main launch\_all\_servers.bat or run standalone.**
* **Port detection prevents conflicts when running multiple servers on the same machine.**
* **Steam launch ensures server files and Steam environment are up-to-date before starting.**
* **IP detection helps bind the server to the correct local network interface, which is important for multi-NIC or VPN setups.**
* **The use of native Windows tools (PowerShell, netstat, tasklist, Registry queries) makes it robust without external dependencies.**

# ALP Game Server Setup Wizard: **Add Server.py**

## Purpose

**This Python Tkinter GUI wizard helps users register new dedicated game servers into the ALP (Automated LAN Party) system by:**

* **Collecting game server info (name, RAM/VRAM).**
* **Selecting the server executable or batch file.**
* **Adding optional supporting files.**
* **Creating shortcuts and folder icons.**
* **Generating a launch batch script to start the server.**
* **Automatically adding the server to the master launch\_all\_servers.bat.**

## Key Features

* **Folder Setup: Creates C:\Games\Batch\_Files and C:\Games\Batch\_Files\Startup if missing.**
* **Icon Extraction: Extracts .ico icons from .exe files using Windows API for a nice folder and shortcut icon.**
* **Shortcut Creation: Creates shortcuts (.lnk) for the server executable and supporting files.**
* **Folder Icon: Writes a desktop.ini file to assign the folder icon in Explorer.**
* **Batch Script: Writes a launch\_<gamename>.bat script to launch the server only if not already running.**
* **Master Batch Update: Inserts/removes calls in the master launch\_all\_servers.bat between user-added markers.**
* **Wizard GUI:**
  + **Step 0: Welcome screen.**
  + **Step 1: Enter game name, RAM, VRAM.**
  + **Step 2: Select server .exe or .bat, set launch arguments for .exe, or icon for .bat.**
  + **Step 3: Add supporting files (optional).**
  + **Step 4: Review summary and finish setup.**

## How to Use

1. **Run Add Server.py.**
2. **Follow the wizard steps:**
   * **Enter your game server info.**
   * **Locate the dedicated server executable or batch file.**
   * **Optionally add supporting files.**
   * **Review summary.**
3. **Click Finish Setup to:**
   * **Create shortcuts.**
   * **Extract icons and set folder icon.**
   * **Generate launch batch script.**
   * **Add the game to the master server launcher batch file.**
4. **The new server is ready to launch via your ALP master script.**

# Detailed Operation Description: **Add Download.py**

**This Python script implements a multi-step graphical wizard for adding a new downloadable item to the ALP WordPress Download Manager plugin database via direct MySQL access.**

## Initialization and Setup

* **Class DownloadWizard inherits from tk.Tk:  
  The main window is a fixed-size 700x530 px Tkinter window titled "Add New Download to ALP WordPress."**
* **Data Storage:  
  Uses tk.StringVar() variables stored in a dictionary self.data to hold user input fields:**
  + **'name' (download title),**
  + **'category' (dropdown: default "Game"),**
  + **'version' (version string),**
  + **'description' (HTML allowed),**
  + **'zip\_path' (full path to .zip file),**
  + **'header\_image\_path' (optional header image path),**
  + **'zip\_size' (display string for zip file size).**
* **Database Configuration:  
  Contains connection parameters for the local MySQL database:**
  + **host: 'localhost'**
  + **user: 'root'**
  + **password: 'LANpass1337'**
  + **database: 'alp'**
* **Step Frames:  
  The wizard is divided into 4 main frames stored in self.frames, corresponding to each step:**
  + **Welcome**
  + **Download Details**
  + **Select Files**
  + **Review & Submit Summary**

**The current step is tracked by self.current\_step and self.current\_frame.**

## Step 0 — Welcome Screen

* **Shows instructions on where the user should place the download .zip file and optional header image before proceeding.**
* **Two folder paths are displayed as clickable, underlined blue labels:**
  + **.zip files expected in:  
    C:/wamp64/www/wp-content/uploads/download-manager-files/**
  + **Header images expected in:  
    C:/wamp64/www/wp-content/uploads/2023/08/**
* **Clicking the folder paths attempts to open the respective folder in the OS file explorer.**
* **A Next button advances the wizard to Step 1.**

## Step 1 — Download Details

* **Contains form inputs for:**
  + **Download Name: text entry.**
  + **Category: dropdown selection with options "Games" or "Programs", defaulting to "Games".**
  + **File Version: text entry, free form.**
  + **Description: a multi-line text widget with vertical scrollbar that supports HTML content.**
* **The description text widget syncs its contents to the self.data['description'] variable on focus loss (<FocusOut> event).**
* **Navigation buttons:**
  + **Back: returns to Step 0.**
  + **Next: saves description and advances to Step 2.**

## Step 2 — Select Files

* **Allows users to select:**
  + **The Zip File:**
    - **Text entry for the full path (editable).**
    - **Browse button opens a file dialog starting at the expected uploads folder for zip files.**
    - **Upon selecting a zip file, the script calculates and displays the file size in MB below the path entry.**
  + **The Header Image (optional):**
    - **Text entry for image path.**
    - **Browse button opens a file dialog starting at the expected header images folder, filtered for common image formats.**
* **Navigation buttons:**
  + **Back: returns to Step 1.**
  + **Next: validates zip file presence (shows error if missing), then advances to Step 3.**

## Step 3 — Summary & Submit

* **Displays a read-only summary text box that shows all the gathered input fields formatted with labels:**
  + **Download Name**
  + **Category**
  + **File Version**
  + **Description (raw HTML)**
  + **Zip File Path**
  + **Header Image Path or "None"**
* **Navigation buttons:**
  + **Back: returns to Step 2.**
  + **Submit: triggers the database insertion.**

## Submission Process (submit method)

* **Data Preparation:  
  Trims whitespace and standardizes file paths by replacing backslashes with forward slashes.**
* **Validation:  
  Checks for required fields:**
  + **Name must not be empty.**
  + **Zip file path must be selected.**
  + **Version defaults to "1.0" if left blank.**
* **Slug Generation:  
  Creates a slug from the name by converting to lowercase and replacing spaces with hyphens.**
* **Content Composition:  
  Constructs the post content HTML string:**
  + **If a header image path is provided, prepends an <img> tag referencing the relative URL path (stripping C:/wamp64/www prefix).**
  + **Appends the description HTML.**
* **Database Insertions:**
  + **Connect to MySQL database using credentials in self.db\_config.**
  + **Insert into wp\_posts table a new post with:**
    - **post\_author = 1**
    - **post\_date and post\_date\_gmt = current timestamp**
    - **post\_content = constructed content**
    - **post\_title = download name**
    - **post\_status = 'publish'**
    - **comment\_status and ping\_status = 'closed'**
    - **post\_name = slug**
    - **post\_type = 'wpdmpro' (Download Manager post type)**
  + **Retrieve the newly inserted post\_id.**
  + **Query taxonomy IDs for the selected category from wp\_terms joined to wp\_term\_taxonomy where taxonomy is 'wpdmcategory'.**
  + **If found, insert relationship in wp\_term\_relationships linking the new post with that taxonomy.**
  + **Insert meta data into wp\_postmeta for the post:**
    - **\_\_wpdm\_files: serialized array containing the zip file path.**
    - **\_\_wpdm\_version: file version.**
    - **\_\_wpdm\_package\_size: calculated size of zip file in MB with two decimals.**
    - **\_\_wpdm\_access: serialized access string (guest).**
    - **\_\_wpdm\_view\_count and \_\_wpdm\_download\_count: initialized to '0'.**
    - **\_\_wpdm\_page\_template: fixed page template string.**
  + **Commit all changes and close the connection.**
* **User Feedback:**
  + **Shows a success message box with the new post ID.**
  + **Closes the wizard window.**
* **Error Handling:**
  + **Any MySQL errors display an error message box.**
  + **Missing required fields cause an error dialog and redirect to appropriate step.**

## Additional Methods

* **open\_folder(folder\_path)  
  Attempts to open a given folder path in the OS file explorer. Shows error dialogs if the folder doesn’t exist or cannot be opened.**
* **validate\_files\_step()  
  Checks that the zip file path is selected before moving to summary step; shows error if not.**
* **show\_summary()  
  Formats all entered data into a readable summary for review before submission.**

## Execution

* **The script runs the wizard GUI loop when executed as a main program:**

**python**

**CopyEdit**

**if \_\_name\_\_ == "\_\_main\_\_":**

**app = DownloadWizard()**

**app.mainloop()**

**This comprehensive explanation covers the structure, UI flow, data handling, database operations, and error handling of the Add Download.py script.**

**Add Server Status checker to site.py — Detailed Operation Description**

**This script provides a graphical user interface (GUI) to help users generate two files: a Python server query script and a PHP snippet for embedding a game server status checker into a WordPress site. It automates the creation of these files based on user inputs and prepares them for deployment.**

## Overall Structure

* **The script is built on tkinter and creates a main window titled "Game Server Query Script Generator" sized 600x480 pixels, non-resizable.**
* **It uses a scrollable frame to accommodate all input widgets vertically.**
* **The interface allows users to specify details for either Local EXE detection or Network Scan detection methods.**
* **Once all inputs are validated, the script generates:**
  + **A Python script to query the game server(s) based on chosen detection method.**
  + **A PHP snippet file to embed the server status output on a WordPress site.**
* **The script also extracts or converts icons, manages file output locations, and guides the user on next steps via a popup with clickable instructions.**

## Key Components and Workflow

### Initialization and Setup

* **Variables are stored as tk.StringVar() for dynamic binding with input widgets.**
* **Folder constants define where Python scripts, icons, and PHP files will be saved:**
  + **OUTPUT\_FOLDER for generated Python scripts.**
  + **ICON\_SAVE\_FOLDER for storing 25x25 BMP icons.**
  + **PHP\_OUTPUT\_FOLDER defaults to the user's Desktop.**
* **A PHP template file (PHP\_Template.php) is loaded from the script directory once at startup for later use.**

### Scrollable Container

* **A Canvas widget with a vertical scrollbar hosts the entire form content inside a ttk.Frame, allowing vertical scrolling if the window content exceeds the view.**

### Input Fields

* **Server Name: Single-line text entry for the server’s display name.**
* **Detection Method: Dropdown with two options:**
  + ***Local EXE*: Detect server by querying a local executable file.**
  + ***Network Scan*: Detect servers via network broadcast scanning.**
* **Local EXE Inputs:**
  + **Path to server executable file with a browse button.**
* **Network Scan Inputs:**
  + **Start and end port numbers for scanning.**
  + **Broadcast IP address to send queries to.**
  + **Broadcast hex payload string (raw hex data for broadcast).**
  + **Icon file selector with browse button.**
  + **Multiple optional handshake payload hex strings that can be added dynamically with an "Add Payload" button.**
* **Max Players: Numeric input specifying the max players for the server.**
* **Download Location: Dropdown populated dynamically from the WordPress database listing published downloadable posts (wpdmpro post type).**

### Dynamic UI Behavior

* **Switching between detection methods toggles visibility of the appropriate input sections (Local EXE fields vs Network Scan fields).**
* **Initial state defaults to *Local EXE*.**

### Database Interaction

* **On startup, the script queries the local MySQL WordPress database (alp database) to fetch all published download slugs, and populates the "Download Location" dropdown with URLs formatted as /download/{post\_name}/.**
* **Errors during DB access update the status label with an error message and disable the download links.**

### Icon Extraction and Processing

* **For *Local EXE* method:**
  + **Attempts to extract the first icon embedded in the EXE file using Windows API calls (win32gui, win32ui).**
  + **Saves extracted icon as a temporary large bitmap.**
  + **Then resizes and converts the icon to a 25x25 BMP, saving it permanently.**
* **For *Network Scan* method:**
  + **Converts the user-selected icon image file to a 25x25 BMP using PIL (Pillow).**

### Input Validation

* **Validates required fields such as Server Name and Max Players.**
* **For *Local EXE* method, checks EXE path validity and icon extraction success.**
* **For *Network Scan* method, verifies:**
  + **Start and End ports are numeric and in the valid port range (1-65535).**
  + **Broadcast IP is a valid IPv4 address.**
  + **Broadcast hex payload and handshake payloads contain only valid hex characters.**
  + **Icon file exists and can be converted.**

### Script Generation

* **Depending on detection method, the script content is generated by loading and formatting one of two Python script templates:**
  + **PY\_EXE\_Template.py for Local EXE method.**
  + **PY\_Broadcast\_Template.py for Network Scan method.**
* **The generated Python script is saved to OUTPUT\_FOLDER with a filename based on the server name.**
* **The PHP snippet is created by replacing placeholders in the loaded PHP template with the server’s name and download path, then saved to the Desktop folder.**

### Overwrite Handling

* **If files with the target names already exist, the user is prompted to confirm overwriting.**
* **Cancelling stops the generation and updates the status label accordingly.**

### Post-Generation Actions

* **Automatically attempts to open the generated PHP snippet in Notepad for easy copying.**
* **Opens a popup window providing detailed next steps on how to use the generated PHP snippet in WordPress:**
  + **Copy the PHP code into the WPCode plugin.**
  + **Activate the snippet.**
  + **Copy the shortcode to the Server Status page.**
* **The popup includes clickable links to:**
  + **The WPCode plugin admin page.**
  + **The Server Status page edit screen.**

### Status Feedback

* **The bottom label shows success or error messages for the user at all stages (e.g., loading downloads, icon extraction, file generation).**

**Summary – Server Status Search**

**This script automates the creation of customized server status checking scripts and WordPress embed code with a user-friendly GUI. It supports two detection methods—local executable or network scanning—and integrates with a WordPress site's downloads database for convenience. It handles icon extraction/conversion, input validation, file writing, and user guidance for deployment, making it easy for non-technical users to add live game server status displays to their WordPress sites.**

# Server side web development: **PHP\_Template.php**

## Purpose:

**This PHP script connects to a MySQL database to retrieve and display the status of a specific game server. It dynamically generates an HTML table row summarizing the server information, status icon, and a download link.**

## Configuration:

**- MySQL Connection Parameters:**

**- Host: 127.0.0.1**

**- Username: root**

**- Password: LANpass1337**

**- Database: server\_status**

## Database Interaction:

**- Queries the table named `%Gamename%\_game\_server` (where `%Gamename%` is replaced by the game name) to fetch the first row containing server status data.**

**- Columns expected in the table:**

**- `server\_name`**

**- `map`**

**- `max\_players`**

**- `game\_mode`**

**- `last\_updated` (timestamp)**

## Logic:

**1. Sets the default timezone based on the PHP configuration (`php.ini`).**

**2. Connects to the MySQL database using the specified credentials.**

**3. Executes a query to retrieve the server status record.**

**4. Assigns default values if no record is found (`NA`).**

**5. Calculates the time difference in minutes between the current time and the `last\_updated` timestamp.**

**6. Determines the server status icon based on:**

**- Offline if `max\_players` is '0' or 'NA'.**

**- Online if the last update was within 10 minutes.**

**- Unknown otherwise.**

## Output:

**- Generates an HTML table containing:**

**- Game icon (`%Gamename%\_icon.bmp`) sized 25x25 pixels.**

**- Server status icon (`online.png`, `offline.png`, or `unknown.png`).**

**- Game name.**

**- Server name.**

**- Maximum number of players.**

**- Current map name.**

**- Game mode.**

**- Download link icon linking to `%DownloadPath%`.**

## Icon and Image Paths:

**- Icons are expected in the relative path: `../../../wp-content/uploads/2023/08/`**

**- Server status icons: `online.png`, `offline.png`, `unknown.png`**

**- Game icon file format: BMP, named `%Gamename%\_icon.bmp`**

**- Download icon: `download.png`**

## Usage Notes:

**- `%Gamename%` and `%DownloadPath%` placeholders must be replaced dynamically before deployment.**

**- The database table and columns must exist and be populated with current server data.**

**- The script assumes the images are uploaded and accessible via the specified relative paths.**

**- The PHP script outputs a single table row intended for inclusion in a larger webpage.**

## Error Handling:

**- On database connection failure, the script terminates with an error message.**

**- If no server data is available, default values and icons indicate the status accordingly.**

## Dependencies:

**- PHP with MySQLi extension enabled.**

**- Proper file permissions to access images and database.**

# Game Servers Detection: **PY\_Broadcast\_Template.py**

## Purpose:

**This Python script performs a network broadcast to detect game servers on a local network by sending UDP packets to a specified broadcast address and port range. It listens for server responses, optionally sends additional handshake payloads, and updates a MySQL database with the discovered server status.**

## Configuration:

**- Database connection parameters (MySQL):**

**- Host: localhost**

**- User: root**

**- Password: LANpass1337**

**- Database: server\_status**

**- Script parameters (dynamically inserted):**

**- `game\_name`: Display name of the game server.**

**- `max\_players`: Maximum player count expected.**

**- `gamename`: Used as a prefix for database table naming.**

**- `start\_port`, `end\_port`: Port range for broadcasting.**

**- `broadcast\_ip`: Target broadcast IP address.**

**- `broadcast\_hex`: Hex string representing the UDP broadcast payload.**

**- `additional\_payloads`: List of optional hex string payloads to send to responding servers.**

**- `timeout`: Socket timeout in seconds for listening to responses.**

## Core Functionality:

**1. `get\_short\_hostname()`**

**- Retrieves the local machine's hostname truncated at the first period.**

**2. `broadcast\_and\_listen(start\_port, end\_port, broadcast\_ip, message\_hex, timeout)`**

**- Converts the hex string broadcast message to bytes.**

**- Creates a UDP socket configured for broadcasting.**

**- Sends the broadcast message to all ports within the specified range.**

**- Listens for UDP responses until the timeout expires.**

**- Returns a list of tuples containing received data and sender address.**

**3. `send\_additional\_payloads(additional\_payloads, responses, timeout)`**

**- For each additional hex payload and each responding server, sends the payload.**

**- Waits for optional replies within the timeout period.**

**- Logs activity and errors.**

**- Returns list of received replies.**

**4. `update\_database(gamename, game\_name, server\_name, max\_players, map\_name, game\_mode)`**

**- Connects to the MySQL database.**

**- Creates the server status table `<gamename>\_game\_server` if it does not exist, with columns:**

**- id (primary key)**

**- game\_name**

**- server\_name**

**- max\_players**

**- map**

**- game\_mode**

**- last\_updated (timestamp with auto-update)**

**- Deletes any previous records from this table.**

**- Inserts a new record with the current server information.**

**- Commits changes and closes the connection.**

**- Prints status and error messages.**

**5. `main()`**

**- Initializes parameters from formatted placeholders.**

**- Performs broadcast and listens for responses.**

**- If responses are received:**

**- Optionally sends additional handshake payloads.**

**- Takes the first response, resolves hostname if possible.**

**- Updates the database with server info, using IP:port as map name and "LAN" as game mode.**

**- If no responses received:**

**- Updates the database indicating the server is offline.**

## Error Handling:

**- Socket exceptions during broadcast or payload sending are caught and logged.**

**- Database errors are caught and printed.**

**- Hostname resolution fallback to local machine hostname if reverse DNS fails.**

## Output:

**- Console prints status messages for broadcasting, receiving data, sending additional payloads, and database operations.**

## Dependencies:

**- Python standard libraries: `socket`, `time`, `datetime`**

**- External library: `mysql-connector-python` (mysql.connector)**

**- MySQL database must be accessible with given credentials.**

## Usage Notes:

**- Replace placeholders with actual server info and parameters before running.**

**- Ensure UDP broadcast messages are formatted according to target server protocol.**

**- Additional payloads are optional and used for advanced handshake or query sequences.**

**- The database schema and connectivity must be correctly configured for updates.**

# Game Server Status Monitoring: **PY\_EXE\_Template.py**

## Purpose:

**This Python script monitors the status of a local game server by checking if a specified server executable is running on the machine. It gathers information about the server process, including its listening network ports and IP address, and updates a MySQL database table with the current server status. If the server process is not running, it writes an offline status record.**

## Configuration:

**- Database connection parameters:**

**- Host: localhost**

**- User: root**

**- Password: LANpass1337**

**- Database: server\_status**

**- Script parameters (to be replaced in template):**

**- `exe\_path\_escaped`: Full path to the server executable (escaped backslashes)**

**- `server\_name\_py`: Human-readable server/game name**

**- `max\_players`: Maximum allowed players on the server**

**- `gamename`: Used as a prefix for the database table name**

## Core Functionality:

**1. `get\_short\_hostname()`**

**- Returns the local machine hostname truncated at the first dot.**

**2. `get\_local\_ipv4()`**

**- Attempts to obtain the local IPv4 address by connecting a UDP socket to a public IP (8.8.8.8).**

**- Falls back to localhost (127.0.0.1) if detection fails.**

**3. `is\_process\_running(exe\_name)`**

**- Checks currently running processes (via Windows `tasklist` command) to determine if the specified executable is active.**

**- Returns `True` if running, otherwise `False`.**

**4. `get\_listening\_ports\_for\_process(exe\_name)`**

**- Uses `tasklist` to get process IDs (PIDs) for the target executable.**

**- Uses `netstat -ano` to find TCP or UDP ports associated with those PIDs that are listening.**

**- Returns a sorted list of port numbers as strings.**

**5. `main()`**

**- Extracts executable filename and initializes variables.**

**- Checks if the server process is running:**

**- If running:**

**- Gets local IP and listening ports.**

**- Builds `map\_name` string combining IP and ports.**

**- Updates database with server status, including "Dedicated" as the game mode.**

**- If not running:**

**- Connects to database.**

**- Creates the status table if not exists.**

**- Deletes previous records.**

**- Inserts a record indicating the server executable is not running with max players set to 0.**

**- Handles database connection errors by printing error messages (non-fatal).**

## Database Schema:

**Table `<gamename>\_game\_server` with columns:**

**- `id`: INT, AUTO\_INCREMENT, PRIMARY KEY**

**- `game\_name`: VARCHAR(255), NOT NULL**

**- `server\_name`: VARCHAR(255), NOT NULL**

**- `max\_players`: VARCHAR(255), NOT NULL**

**- `map`: VARCHAR(255), NOT NULL**

**- `game\_mode`: VARCHAR(255), NOT NULL**

**- `last\_updated`: TIMESTAMP, auto-updated on row modification**

## Error Handling:

**- Exceptions in process detection or database operations are caught and logged to console.**

**- Errors during offline info write do not abort the script.**

## Output:

**- Console prints status messages on process detection, IP and ports found, and database update success or failures.**

## Dependencies:

**- Python standard libraries: `os`, `socket`, `subprocess`, `sys`**

**- External library: `mysql-connector-python` (mysql.connector)**

**- Windows environment required for `tasklist` and `netstat` commands.**

**Usage Notes:**

**- Replace template placeholders with actual executable path, server name, max players, and gamename before running.**

**- Ensure MySQL database credentials and permissions are correct.**

**- Script is designed for monitoring dedicated server processes on Windows.**

# Tunez Project Documentation

**What is Tunez?**

**The Tunez project is a music management and playback system. It consists of a web-based interface for uploading and managing music files, a database to store song metadata, and a backend daemon that handles the playback queue. The system is designed to allow multiple users to manage a shared music library and vote on what to play next.**

**How the System Works**

**The system operates using several key components that work together:**

* **Web Interface (PHP): This is the part of the application that users interact with. Files like upload.php, download.php, and admin\_edit\_record.php provide the user with forms to upload new songs, download existing ones, and edit song details. The PHP scripts handle form submissions, validate user permissions, and perform operations on the database.**
* **Database (MySQL): The database serves as the central repository for all song metadata. It stores information such as artist, song title, album, year, and file path. The database.inc.php file provides the functions for interacting with this data. The database schema includes tables for songs, artists, and albums.**
* **Daemon (tunezd.py): The daemon is a Python script that runs continuously in the background. It is responsible for the core logic of the playback system. It interacts with a priority queue to determine the next song to play, often based on user votes.**
* **Playback Queue (PQueue.class.php): The priority queue manages the order of songs to be played. Users can vote for songs, and the daemon uses this information to build a queue. This allows the community to influence the song that is played next.**

**Adding New Songs via the Web Interface**

**To add a new song to the Tunez library, users must access the upload.php form. This form can be found by navigating to /tunez/upload.php or clicking the "Upload a Song" link from the main page.**

**On this form, you will need to provide the following information:**

* **Song File: The MP3 or OGG file you wish to upload.**
* **Artist: The artist's name (this is a mandatory field).**
* **Song Title: The title of the song (this is also mandatory).**
* **Album: The album name (optional).**
* **Year: The year the song was released (optional).**

**When you click "Upload the Song," the upload.php script handles the process. It first checks your permissions and ensures that uploads are enabled on the system. It then attempts to read metadata directly from the song file itself and uses the information you provided in the form to create a new entry in the database. Finally, it moves the uploaded file to the correct location on the server and provides a success message.**

**The Role of the Daemon**

**The daemon is a critical, but largely invisible, part of the system. It is a separate process that runs independently of the web server. As shown in the run\_python\_daemon.vbs file, it's launched as a background process from the command line.**

**The primary responsibilities of the daemon are:**

* **Playback Queue Management: The daemon uses the PQueue class to constantly manage the list of songs waiting to be played.**
* **Next Song Selection: It's the daemon's job to determine which song is next in the queue. It can do this based on a number of factors, such as the number of user votes or a simple randomization.**
* **Database Updates: After a song is played, the daemon updates the database to reflect that the song has been played, which is used for statistics and, in some cases, for weighted random song selection.**

**Daemon Status and State Management**

**The daemon uses a few mechanisms to communicate its status and track the current playback state:**

* **np Database Table: The np (Now Playing) table in the database is used to hold information about the song that is currently being played. The daemon uses the dequeue() function in PQueue.class.php to insert a new record into this table every time a song begins playing. The table stores the song\_id, a play\_id (a unique identifier for the current play session), the timestamp when the song started, and a flag indicating if the song was a random selection. The web interface can read this table to display the currently playing song to users.**
* **status.txt File: The daemon also writes its current status to a file named status.txt in the main Tunez directory. This is a simple plain text file that contains information for external monitoring systems or other scripts. For example, it might contain the ID of the currently playing song, the daemon's process ID, or an indication of whether the daemon is running or idle. This provides a lightweight way for other parts of the system to check on the daemon's health without constantly querying the database.**

**Essentially, while the web pages let you add and manage the music library, the daemon is the "brain" that decides what music gets played and when, with np and status.txt serving as key communication points.**