Arozos Cloud Desktop Operating System Documentation

Written by tobychui, 8-5-2021, for reference only

REMARKS

The previous release of the arozos is named “ArOZ Online System” or “ArOZ Online Beta” (AOB). For information and documentation for the beta version of this system, please read the “ArOZ Online Distributed Cloud System Documentation” instead.

You can classify the difference between these two systems by identifying the programming language of the system.

Version	Programming Language	File Extension
ArOZ Online Beta	PHP5 / PHP7	.php
ArOZ OS (ArOZ Online 1.0)	Go	.go

ArOZ OS is written as “arozos” in the current documentation for simplicity and easy to read purposes.

Table of Contents

Introduction

Aim and Objective of the project

Many off-the-shelf cloud services and infrastructure are only designed for commercial or business purposes. When it comes to non-profiting oriented or general purpose cloud platforms, there are only limited choices for easy plug & play development.

The beta phrase of this project aims to provide a low cost, personal and private cloud architecture in both software and hardware aspects that is distributed with high scalability and reliability for deploying critical systems.

In the 1.0 version of this system, the system architecture is being redesigned to fit more general purpose cloud computing, including quick system deployment, service binding and allowing general users to engage in cloud computing technology through a user friendly, web desktop interface.

Naming Scheme

Stage Naming

ArOZ Online System (Originally named Automated Remote Operating Zigzagger, which Zigzagger implies stitching things together in a good way) was a platform that is designed to store multimedia files with an external hard disk on Raspberry Pi Model B, providing “media center” like experience to users and allows for media consumption in local area network environment. Later, more and more cloud related functions are added into the system for better enhancing the usability of the system including but not limited to web desktop environment, cluster setups and communication pipelines, distributed file systems and in system programmable modules. These modules and subsystems add more functionality to the cloud platform.

The Beta system (ArOZ Online Beta) is a system that provides all of the functions mentioned above and delivers a web-desktop-like environment by bridging to the underlying Linux file system. Providing a powerful cloud desktop environment for users to be used with any mobile or desktop devices without the constraints of a designated terminal device(s) like personal smartphones or laptops.

The 1.0 version (arozos) is a system that is completely rewritten based on the Beta phrase requirement discovery process. Most of the well known functions on the beta are being rewritten into a more efficiency algorithm with Go instead of PHP. Providing much more performance boost out of linux SBCs like the Raspberry Pis without the need for upgrading the hardware.

Version Naming

The version system of the ArOZ Series software is as follows.

Version Name	Development Date	Systems (Language)
ArOZ (Alpha)	Early 2014	Windows 7 (VB.net)
ArOZ Beta	Late 2014	Windows 7 (VB.net)
ArOZ Omega	Early 2015	Windows 7(VB.net)
ArOZ Online Alpha	Late 2015 - Early 2016	Windows 7 (WAMP + PHP 5)
ArOZ Online Beta	2016 - 2020	Windows / Linux (Apache + PHP 7)
ArOZ OS	2020 - Now	Windows / Linux / Mac OS (Go 1.14+)

Version Code

The ArozOS Version code is formatted as follows.
[unique identification number of branch].[major version number].[minor version number]

The Unique Identification Number of Branch, or the UINB is the number where if a developer fork the ArozOS and release it as his own project for alternative audience or organization, this number has to be changed to prevent the user from mixing the alternative branch from the main branch. In the main branch (source), the UINB will always be 0 (Source of all sub-branches).

Examples version code in the main branches are:
v0.1.110 , v0.1.111 and v0.1.112

Release Code

The ArozOS release code follows the same version naming scheme as the Version Code but without the UINB number. Examples are
v1.110, v1.111 and v1.112

System Usage

ArOZ Online System can be run on a portable device, miniature NAS system or server grade computers. Hence, the system will suit the needs for many different usages, including portable workstation, media conversion or consumption, data backups and restore etc.

The arozos provides even more general purpose for business sector, including fully featured authentication system, user permission and grouping systems, internal reverse proxy services and storage pool management. Allowing the system to be used in deploying web platforms that requires user management as well as user permission management. Modular WebApp system design also strengthened the permission system and allowed more secure cloud computing process and web services.

Terminology

In this documentation, the following terminology will be used to describe the content and functions provided in this system.

Term	Explanation
Arozos / ArOZ OS	The cloud system that is being discussed in this document.
WebApps	Web application that is installed on the arozos as a folder in the /web directory.
Subservice	Web application that is installed on the arozos as a separated binary located in its folder under /subservice directory.
Web Desktop Interface / VDI mode	Virtual Desktop Interface (Mode). The mode where the user is interacting with the arozos through its web based desktop user interface.
List Menu	The application startup menu that used to start an application on both VDI and mobile interface
FloatWindow (fw)	Window-like iframe that is located on the web based desktop interface that the user can drag, resize and hide.
Functio(nal) Bar	The bottom bar of the web desktop interface that used to display the web app opened by the user OR The sidebar of the mobile desktop mode where the webapp opened by the user will be shown.
Status Bar	The top bar on the web desktop interface that is used to show the hostname, current date time and a content button for showing shortcuts.
Arozfs / aroz virtual file system	The virtualized file system that is emulated by arozos.

System Requirement

Hardware

Minimal
1Ghz CPU (ARMv6/7, ARM64 or AMD64), 512MB RAM, 8GB Storage

Recommended
2+Ghz CPU (ARM7, ARM64 or AMD64), 2GB RAM, 32GB Storage

System Tested on:

- Raspberry Pi 3B+ w/ Raspberry Pi OS
- Raspberry Pi Zero W w/ Raspberry Pi OS

- Raspberry Pi 4B+ (1GB / 2GB / 4GB version) w/ Raspberry Pi OS
- Orange Pi Zero Plug (H5 CPU edition) w/ Armbian Buster
- AMD64 ThinClient w/ Debian Buster
- Ryzen 5 w/ Windows 10
- Intel Pentium w/ Windows 7

Software

Operating System:
Raspberry Pi OS / Debian Buster, (Limited functionality on Windows 7+ and macOS Catalina +)

Package (Required):
Wpa_supplicant[1] or nmcli[2], net-tools (ifconfig[3]), FFmpeg[4]

Clients / Browsers

Any modern browsers (Latest version of Chrome / Firefox / Safari (Not Tested) / Edge) on macOS High Sierra or above / Windows 7 or above

Network

Network Environment:
Network speed of Minimum 10Mbps (100Mbps+ Recommended), WiFi 2.4 / 5 Ghz or Ethernet

[1] wpa_supplicant is a cross-platform supplicant with support for WEP, WPA and WPA2 (IEEE 802.11i)
[2] nmcli is a command-line tool for controlling NetworkManager and reporting network status.
[3] ifconfig is a system administration utility in Unix-like operating systems for network interface configuration.
[4] FFmpeg is a large suite of libraries and programs for handling video, audio, and other multimedia files and streams.

System Overview

Summary

Folder Structure

Arozos consists of three major components in its folder structure. The folder structure is listed as follows.

Structure Name	Location	Purpose
Web	./web	Directory for storing WebApp scripts (including System GUI elements)
System	./system	Directory for storing system folders. Databases, templates and other important files are stored here. This should not be exposed via the storage pool handler.
Subservice	./subservice	Subservice that mount to arozos. Allowing arozos to perform reverse proxy access to these web services binaries.
Arozos Binary	./arozos (or arozos.exe)	The executable of the main logic of arozos system

Data Structure

All the data stored in the ArozOS system are located in the following files

File Name	Location	Purpose
ao.db	/system/	Main database for the ArozOS
storage.json	/system/	Storage & Virtual File System configuration of ArozOS system permission group
*.json	/system/storage/	Storage & Virtual File System configuration of non-system permission groups
cron.json	/system/	Scheduler task configuration file
*.log	/system/aecron/	Log file for system scheduler
authlog.db	/system/auth/	Log file for authentication

Application Structures

As arozos move away from PHP, there is no way to dynamically add modules or plugins into a pre-compiled binary. Hence, there are two new methods to add plugins into the system.

1. WebApps - Basic WebApps where main logic is handled by JavaScript and RESTFUL
2. Subservices - Advance WebApps where the application require complex access to the underlying operating system

In the following sections, the structure of the methods will be introduced. For details about developing a plugin for arozos, please see the WebApp and Subservice sections.

WebApp Structures

Arozos Web Applications (or WebApps) are stored under the ./web folder. Each folder contain a list of html, JavaScript and CSS files that can be served via the arozos internal web server. A generic folder structure of a WebApp should consist of the following files.

Filename	Purpose	Mandatory
init.agi	Define the WebApp startup properties	Yes
index.html	Index of WebApp to serve	Yes
*/icon.png	The icon for this module	Yes[5]
*/desktop_icon.png	The icon for showing as a desktop shortcut	No
embedded.html	The embedded mode UI	Depends[6]
floatWindow.html	The float window mode UI	Depends[7]
manifest.json	The required manifest for supporting PWA	Recommend

Notes that the init.agi is mandatory and it must be placed under your WebApp roots (i.e. ./web/{webapp_name}/init.agi). Or otherwise, your WebApp will not be scanned by the arozos startup process.

Subservice Structures

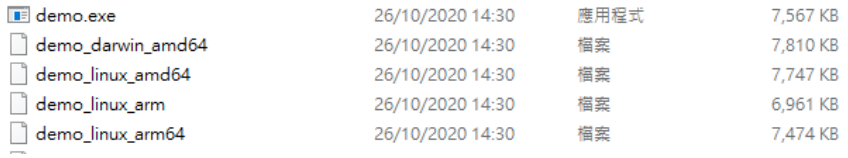
Subservice are web server binaries that are stored under ./subservice folder and provide services that require much higher levels of complexity. A basic subservices contain the following file structures.

Filename	Purpose	Mandatory
{subservice_name}_{platform}_{architecture}	The binary of the subservice	Yes

[5] The image name can be defined in init.agi. See AGI section for more information.
[6] Mandatory when embedded mode is set to true in init.agi
[7] Mandatory when floatWindow mode is set to true in init.agi

.disabled	Flags to disable this subservice on startup	No
.startscript	Flags to load start script for registration instead of the binary itself	No
moduleInfo.json	The module information JSON	Depends[8]
start.sh (or start.bat)	The startup script that replace the -info flag in subservice startup parameter	Depends[9]

Depending on your platform and subservice name the binary name may differ according. For example, here is a subservice named “demo” with support linux (arm, arm64 and amd64), MacOS(darwin) and Windows. Its binary files will look like this under Windows’ File Explorer.

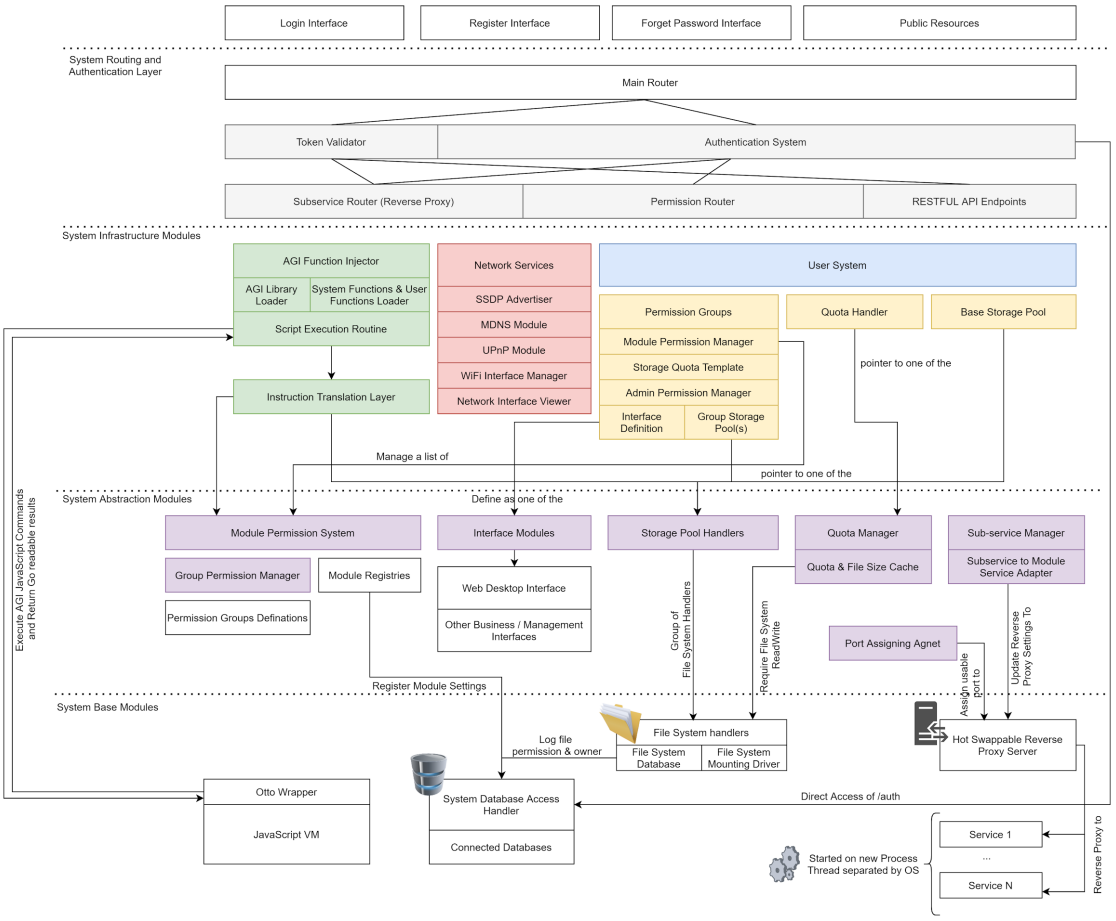


In some cases where glueing scripts are used, there might be a few more bash files or extra binaries. In this case, you will need to assign suitable permission to these files before starting arozos core. Permission denied of executing files inside subservice will lead to failure of starting of arozos system.

[8] If moduleinfo.json exists, the information will be loaded from file instead of the binary itself
[9] Start script is required when .startscrpt flag file exists

Abstraction Structure

The arozos system consists of many layers of complex abstraction for emulating an operating system on any cloud platform or host devices. The following diagram provides an abstract view of the system abstraction structure.



10

In simple words, the arozos structure is mainly consists of the following modules

1. Authentication System
2. Permission Router
3. Reverse Proxy Server
4. ArOZ Gateway Interface (AGI) JavaScript Interpreter
5. Network Services (SSDP / MDNS / UPNP)
6. Permission Group System
7. Storage Quota Management System
8. Storage Pool Management System (Including Path Virtualization)

[10] This diagram is drawn with arozos version 1.105 as base. This diagram might not be up-to-date with the latest arozos system.

User Interface Structure

In arozos, the original “Grid Interface” introduced in Beta has been removed due to most users will just directly launch into their Desktop interface. For that, a new interface is introduced to replace the grid interface as the default interface module for mobile devices. In the arozos UI implementation, the system default supports three kinds of UI.

1. Web Desktop Interface
2. Mobile Desktop Interface
3. Interface Module Interface

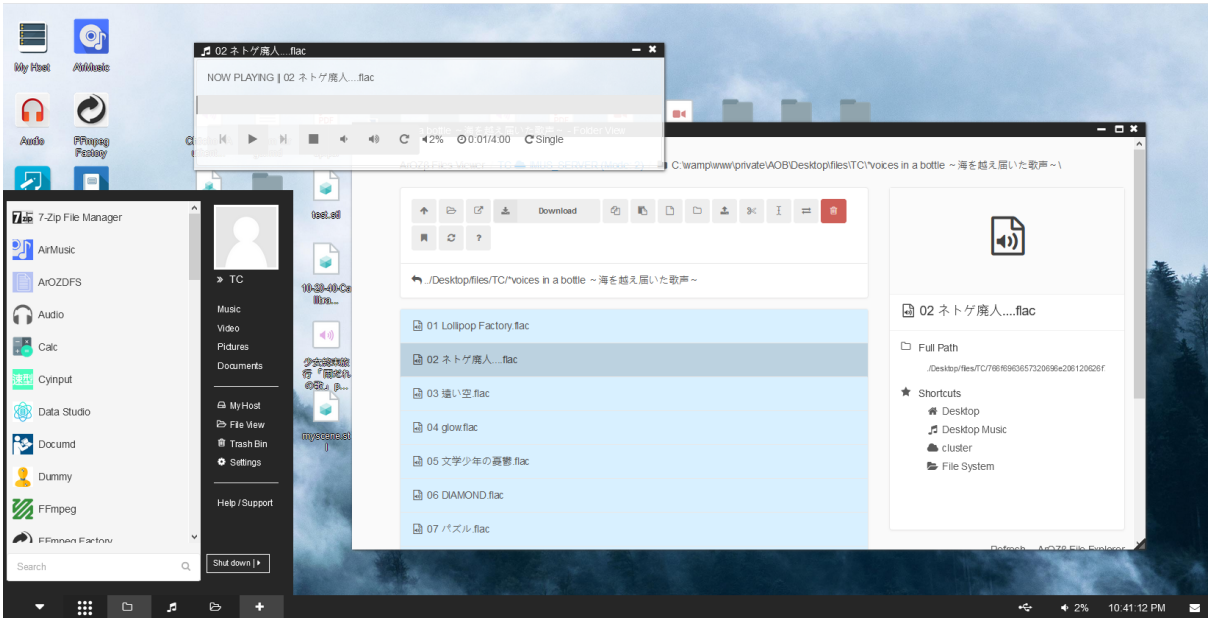
For beta version of aroz, the interface includes

1. Grid Menu Interface
2. Web Desktop Interface
3. Multi System Booting Interface (MSBI)

Web Desktop Interface

The arozos Web Desktop Interface is a complete rewritten of the original desktop interface and provides better user experience compared to the Beta Desktop.

Arozos Web Desktop Interface preview, captured on v1.109 with 21:9 display



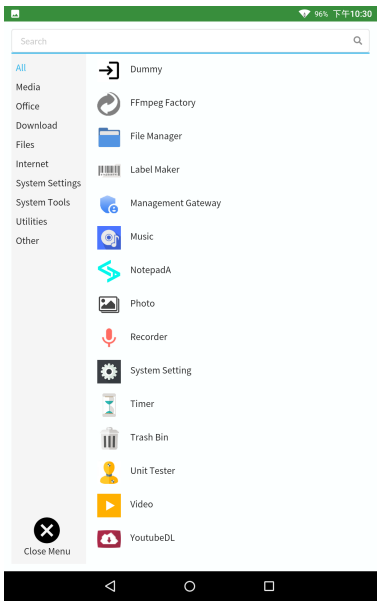
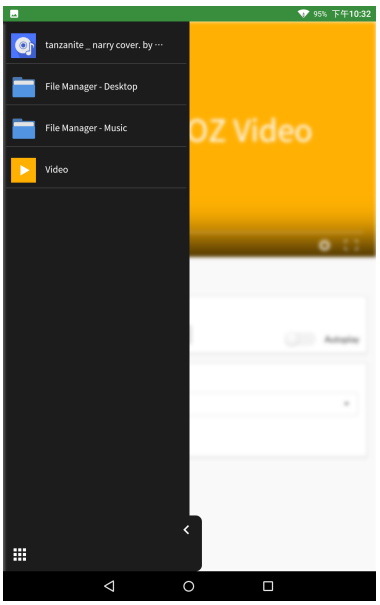
ArOZ Online Beta Web Desktop Interface, capture on Beta LTS

For basic usage, the Web Desktop supports creating new folders, uploading Files by drag drop, double click opening files, folder or application shortcuts etc. See Desktop for more information.

In arozos 1.0, a top menu is added to show time, volume info and Ubuntu 20.04, also provide a notification bar as well as a quick dropdown function menu for access to quick functions including full screen toggle, system settings and user logout.

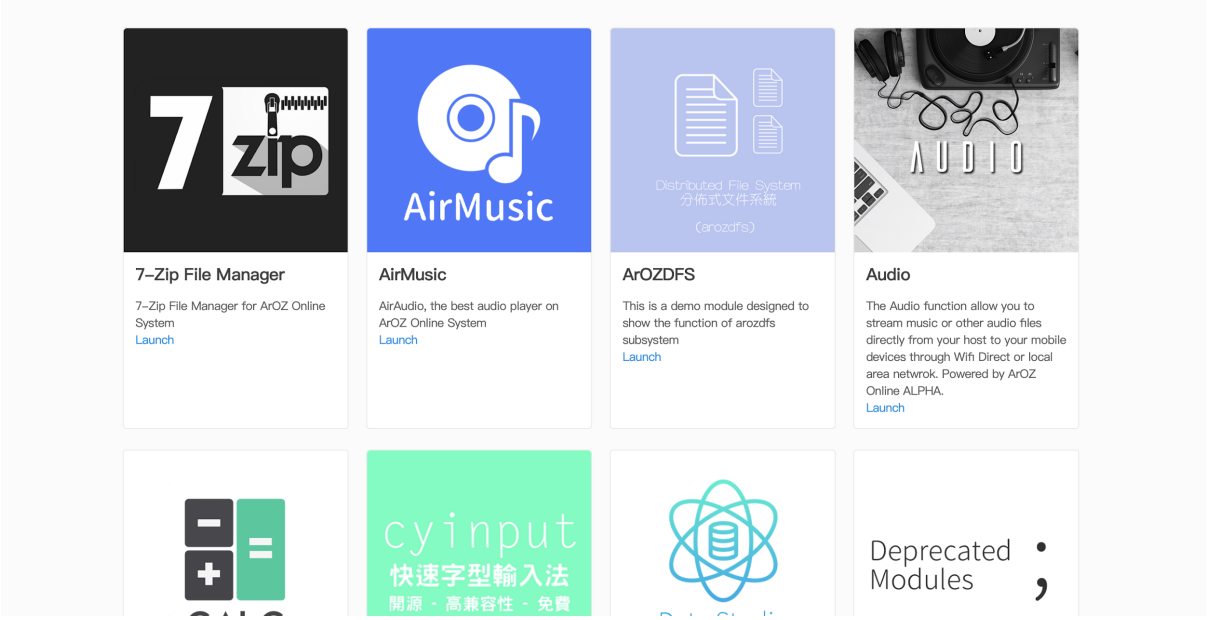
Mobile Desktop Interface

Mobile Desktop Interface was first introduce in arozos 1.105 for supporting vertical screens (Mostly mobile devices). In this mode, the floatWindow is still supported with limited functions. The list menu and toolbar are also replaced with a sidebar instead. However, this interface did allow multi-processing just like the standard Desktop interface.

		
List Menu	Tool Sidebar	Opening Applications

Grid Menu Interface, Deprecated

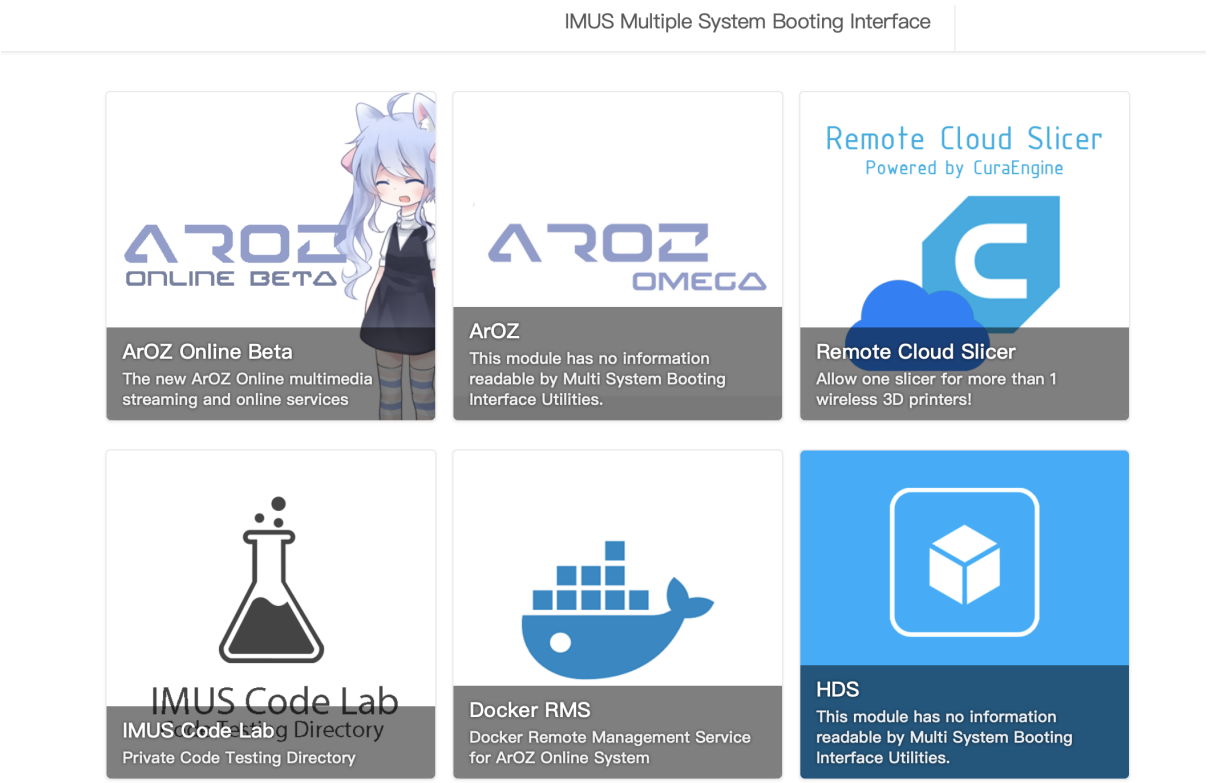
The Grid Menu interface is a deprecated interface designed for ArOZ Online Beta (AOB) users for using the system on mobile devices. It is also the first interface of the ArOZ Online System that provides access to all system modules within one interface.



This interface is deprecated and no longer available on the arozos 1.0. This function has been replaced by the List Men on the mobile desktop interface, and there is no method to enable this feature in arozos 1.0.

IMUS Multi System Booting Interface (MSBI / IMSB), Deprecated

MSBI is the original method for binding service together using a bootloader like system for aroz online beta. It provides a very basic portal for redirecting users to different network services within the same host environment.



For example, multiple of ArOZ Online Beta can be installed on the same machine using the MSBI tool as the main router.

This interface has been deprecated and replaced by the subservice module (functional wise) and permission group interface module settings (Selection wise). For more detail, see the “Interface Module'' section.

System Modules

ArOZ Gateway Interface (AGI)

ArOZ Gateway Interface, or AGI, is an application programming interface that allows access to arozos core system through a ECMA 5.1 scripting interface.

The interface can handle scripting files in file extension .js or .agi. See more application and programming guidelines in the ArOZ Gateway Interface Programming section.

Advance Package Tool (APT)

Advance Package Tool, also known as apt, is a module that helps with installing packages automatically on Linux based host systems that allow access to apt packages under sudo mode. To use this module, one must request package access using agi script and run the arozos in sudo mode.

Authentication (Auth)

System Authentication Module handles all the system authentications, including user creation, login and removal, auto login token management and sessions etc. This module also handles the external API authentication used in AGI interface.

Console

The STDIN / OUT console for arozos debug purpose. To enable the console, use ./arozos -console=true flag during startup and after the web server is started, you can type debug command into the STDIN of the arozos and allow real time access of internal runtime data for debug purposes.

DO NOT ENABLE THIS FLAG WHILE STARTING WITH SYSTEMCTL

Database

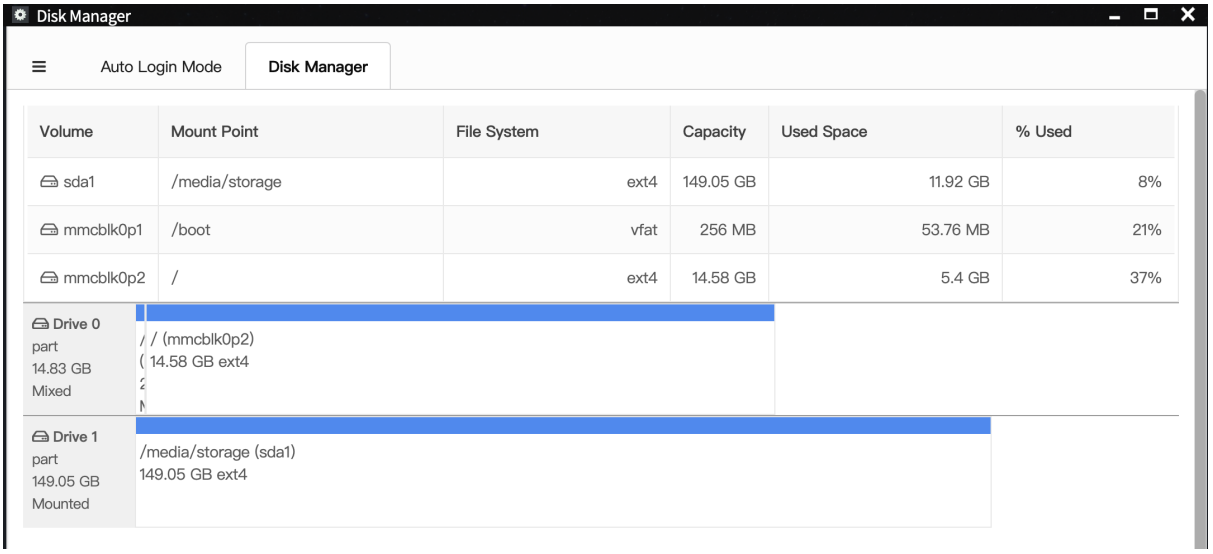
The database handler for the arozos system. The database used in arozos is some kind of key-value database that allow “bucket” creation. Just like a normal database, this module provides a basic interface for reading, writing and checking if a bucket and key exists.

Disk

Diskmg

Diskmg provides the basic interface for handling disk management tools that has been migrated from the aroz online beta to the latest arozos. This tool allows real time in-system mount, unmount and format of external storage devices (or internal, if it is not the primary disk).

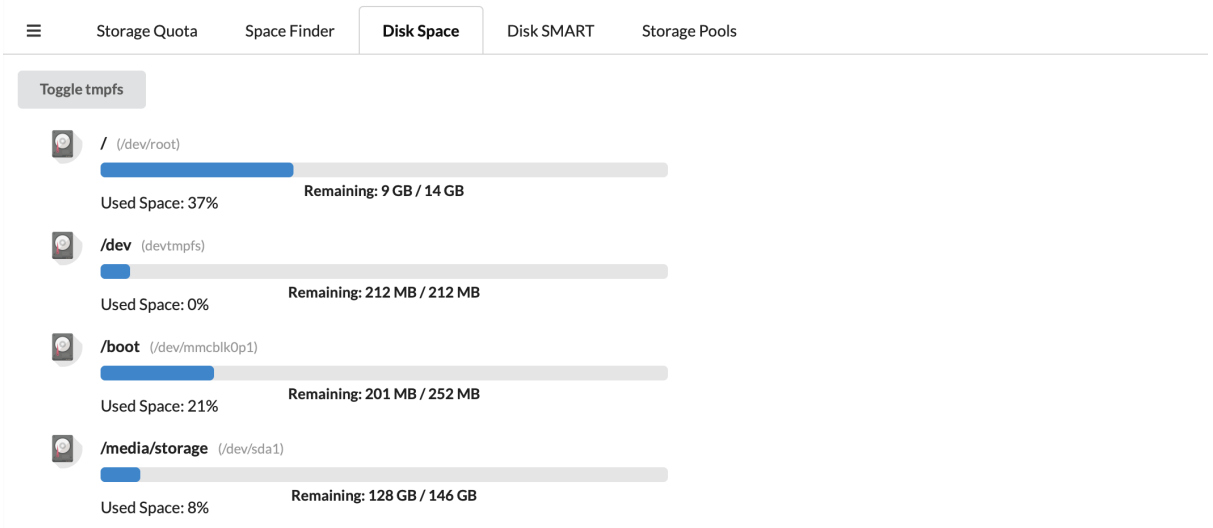
For the disk manager function itself, you can access the function via System Setting > Advance > Disk Manager



Diskspace

DiskSpace is a disk scanning utility that helps you identify the information of the disk remaining space. Similar to the Android space utility, you can see how much storage space you have on your host devices.

This function can be accessed by System Setting > Disk and Storage > Disk Space

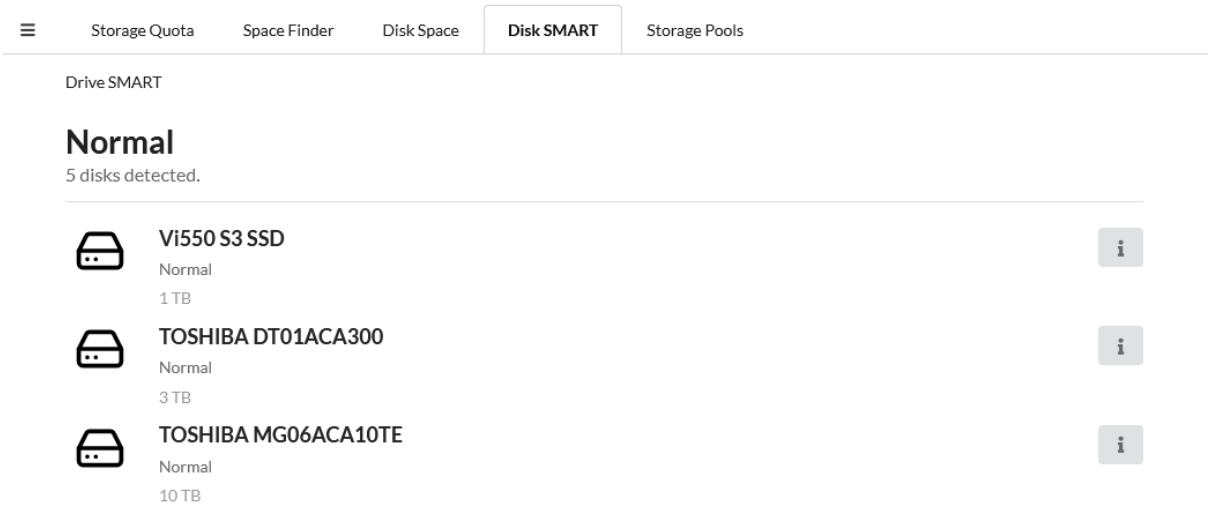


In theory, this utility works on all platforms that support the df command. On windows, disk information is requested from WMIC.

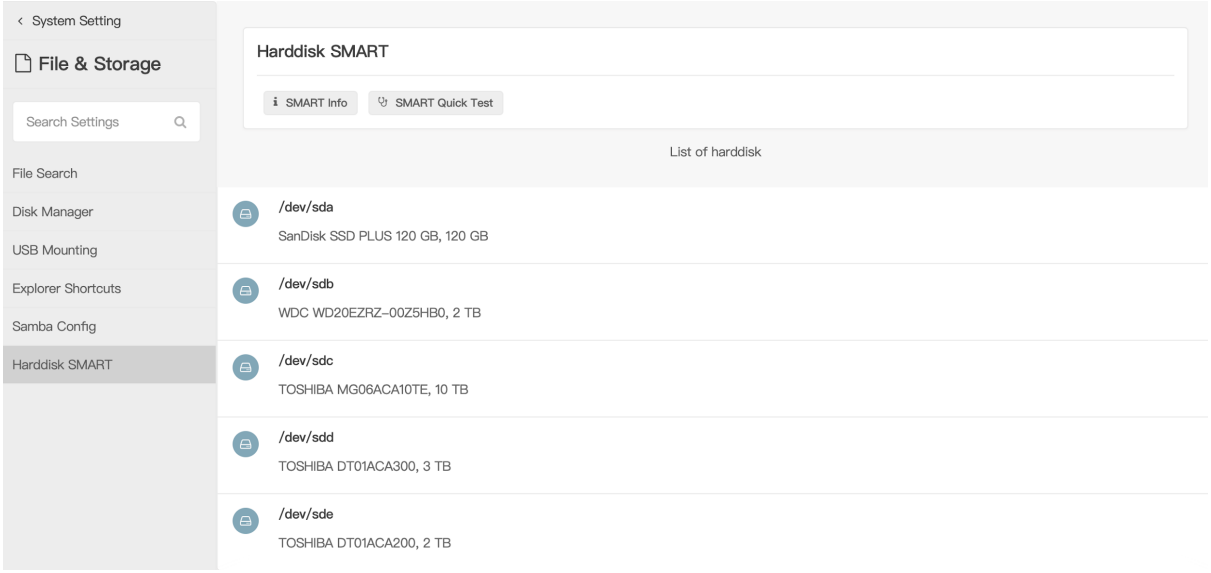
DiskSmart

DiskSMART is a third party developed module that handles the status of the disk, including disk health, disk information and more. This module is migrated from the previous ArOZ Online Beta as one the few module that migrated to arozos during the early stage of development.

The Disk SMART utility can also be found under the Disk and Storage category of the System Settings.



The BETA version of the Disk SMART can be find in System Setting > File & Storage > Disk SMART



SortFile

SortFile is a newly added module that handles large file discovery throughout the arozos mounted virtual storage devices. This module will scan all files within the user's access range and return a file list sorted by the largest (in storage size) to the smallest. The interface can be accessed via System Setting > Disk & Storage as "Space Finder".

Example screenshot of the space finder results

Remarks

The Disk module currently does not have any code that is written directly under the module Disk (imuslab.com/arozos/mod/disk).

File System

The File System module handles major file operations provided by the file_system.go in the main programming scope. Including file operations like copy and paste, move, buffered file copying, zipping etc.

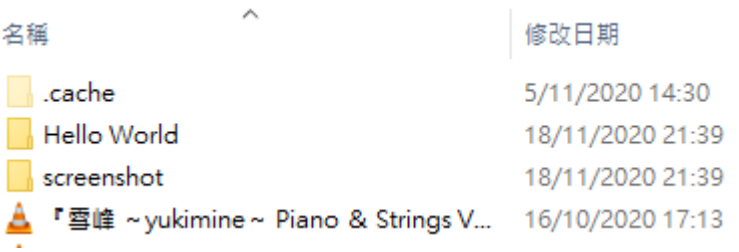
The File System also contains code related to files like metadata extraction, thumbnail generators and other sub-modules that help enhance the user experience in using arozos file explorer. See the details of each module in the subsections below.

Hidden

This is a module that handles file / folder hiding under Linux or Windows. If the current system is operating under a Linux -like environment, for a given path, it will add a “.” prefix to the file to make it hidden. If the system is hosted under Windows, the file will be requested to be hidden using WIN32 API.

This module is mainly used to hide the .cache and .trash folder under Linux / Windows environment.

The example below shows a hidden cache folder rendered under Windows File Explorer



Metadata

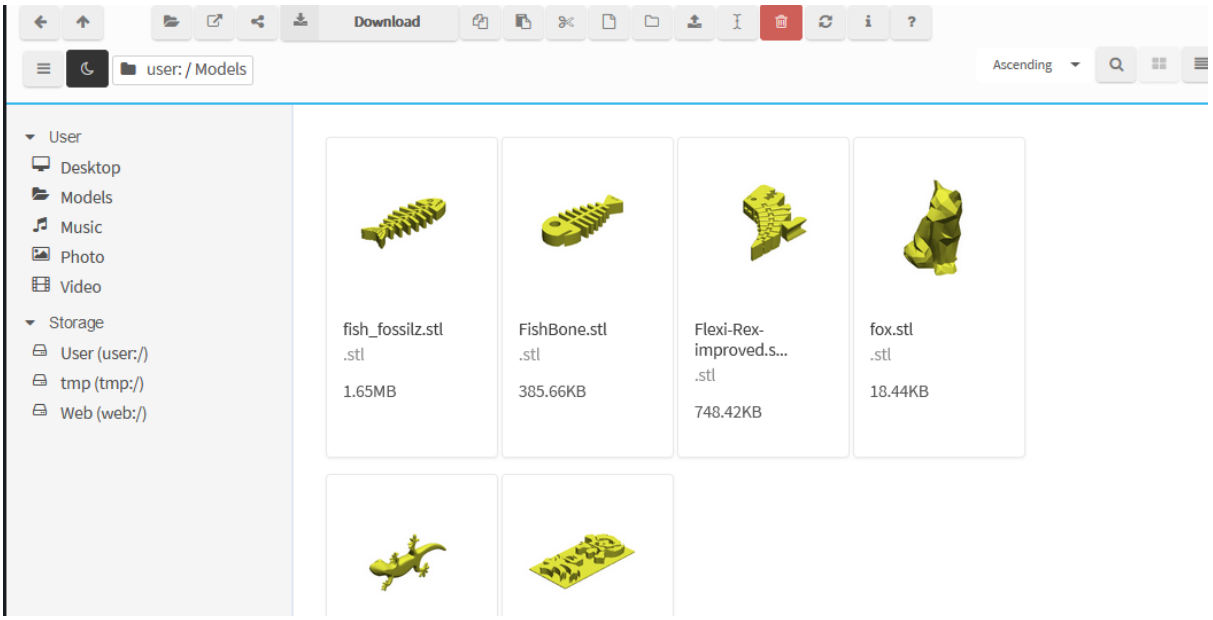
Metadata helps extract the thumbnail from a given file. This module binds different services together and extracts cover from music files, ffmpeg to extract thumbnails from video files and more. **Each thumbnail should be in .JPG format and in 480 x 480 pixel.**

The thumbnail of the extracted files will be stored inside the .cache folder which will then be hidden by the hidden module using local file system methods.

Renderer

Renderer is a 3D rendering module that helps render .stl and .obj file's thumbnail. Mainly used in the metadata module as a rendering assistant.

The followings are some example renders generated by the file system renderer



File System (Core)

Virtual File System Handler

The file system also provides a very important function that creates a FileSystemHandler object from a mounted device. A file system handler is a Go struct that contain the following information (Captured from arozos v1.107)

```
type FileSystemHandler struct {
    Name                string
    UUID                string
    Path                string
    Hierarchy           string
    ReadOnly            bool
    InitiationTime      int64
    FilesystemDatabase *db.Database
    Filesystem          string
    Closed              bool
}
```

The FileSystemHandler will handle the routing of arozos virtual file system translation mechanisms and allow the abstraction above to access only virtualized resources within the scope of the whole host OS. For more information, see ArOZ Virtual File System section.

The virtual file system is created by mounting or real devices from /dev/sd* or any directories within the host OS file system. To see how to create a virtual file system from storage.json config file, see “Base Storage Management Architecture” section.

File Ownership Tracker

File ownership is tracked using the File System database. The database is an aroz key-value database named “aofs.db” located at the root of the virtual file system mount point. To access the file ownership from a given file path (absolute path), these functions can be called:

```
func (fsh *FileSystemHandler) GetFileRecord(realpath string) (string, error)
func (fsh *FileSystemHandler) DeleteFileRecord(realpath string) error
```

The function above will return the owner of the given filepath as string.

Device Mounting Handler

The file system module also provides a handy MountDevice function that can be used to mount storage devices located under /dev/. To use this function, you can simply call to the following function under the file system module:

```
MountDevice(mountpt string, mountdev string, filesystem string)
```

Modules

Modules module handles module registration and launching. This module object keeps a list of running modules and allows modules to be sorted, registered, listed, access its launching parameters and more.

The following is a module's information structure. For how to create a new module registry, see ArOZ Gateway Interface Programming section.

```
type ModuleInfo struct{
    Name string                     //Name of this module. e.g. "Audio"
    Desc string                     //Description for this module
    Group string                //Group of the module, e.g. "system" /
"media" etc
    IconPath string             //Module icon image path e.g.
"Audio/img/function_icon.png"
    Version string              //Version of the module. Format:
[0-9]*.[0-9][0-9].[0-9]
    StartDir string      //Default starting dir, e.g. "Audio/index.html"
    SupportFW bool              //Support floatWindow. If yes, floatWindow
dir will be loaded
    LaunchFWDir string          //This link will be launched instead of
'StartDir' if fw mode
    SupportEmb bool             //Support embedded mode
    LaunchEmb string            //This link will be launched instead of
StartDir / Fw if a file is opened with this module
    InitFWSize []int            //Floatwindow init size. [0] => Width, [1]
=> Height
    InitEmbSize []int           //Embedded mode init size. [0] => Width,
[1] => Height
    SupportedExt []string     //Supported File Extensions. e.g. ".mp3",
".flac", ".wav"
}
```

Network

The network module handles all network related functions within the arozos system. Its subservice are listed belows.

MDNS

MDNS provides software discovery of arozos using MDNS protocol. Allowing Mac devices to find the hosting server.

SSDP

SSDP provides an advertising broadcast to nearby Windows hosts so the device can be scanned via the Network Discovery tab.

UPNP

UPnP provides a wrapper of the zero config settings that allow the host to request UPnP port forward from the NAT router (if any). In the default scenario , the port that hosts the web interface of the arozos will be forward when UPnP (allow_upnp flag) is set to true.

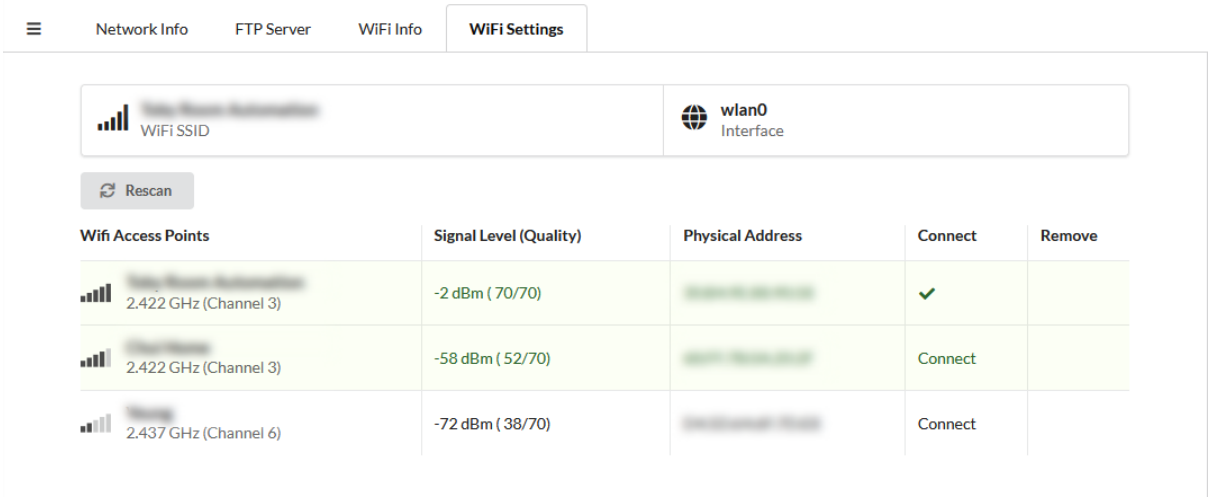
WiFi

The WiFi Module handles WiFI hardware management based on wpa_supplicant and nmcli (Linux) or WMIC (Windows, Read Only).

Under Linux environment, this module provide access to

- WiFi Switching
- Connecting to new WiFi Ap
- Editing Connection Information

The setting interface can be see on the System Setting > Network tab



Reverse Proxy

The Reverse Proxy module is a clone of the ReverPxoy library from github.com/cssivision/reverseproxy with modified error handler that allow arozos to restart freezed subservice during a reverse proxy handling error .See subservice section for more information.

Permission

Permission is a module that handles a user's permission groups and its access permissions to virtual file systems. A permission group consists of the following properties (under arozos v1.107)

Properties Name	Type	Usage
Name	String	The name of this permission group
IsAdmin	Boolean	Do this group has admin privileges

DefaultInterfaceModule	String	The default interface module for all the users in this group.
DefaultStorageQuota	Int64	The default storage quota for user initialize in this group
AccessibleModules	String Slice	All the module that is accessible by this user
StoragePool	*storage.StoragePool	The storage pool that this permission group is assigned to
parent	*PermissionHandler	The parent handler for this permission group

The permission module mainly used to handle user access to modules, control permissions on reading and writing paths and more. The wrapper of this module can be found in User > permissionHandler.go script.

Permission Router (Prouter)

Permission router is a simple to use interface to replace HTTP.HandleFunc with permission control built in. Here is a quick example extracted from file_system.go showcasing how it works.

```
router := prout.NewModuleRouter(prout.RouterOption{
       ModuleName:  "File Manager",
       AdminOnly:   false,
       UserHandler: userHandler,
       DeniedHandler: func(w http.ResponseWriter, r *http.Request) {
             sendErrorResponse(w, "Permission Denied")
       },
    })
```

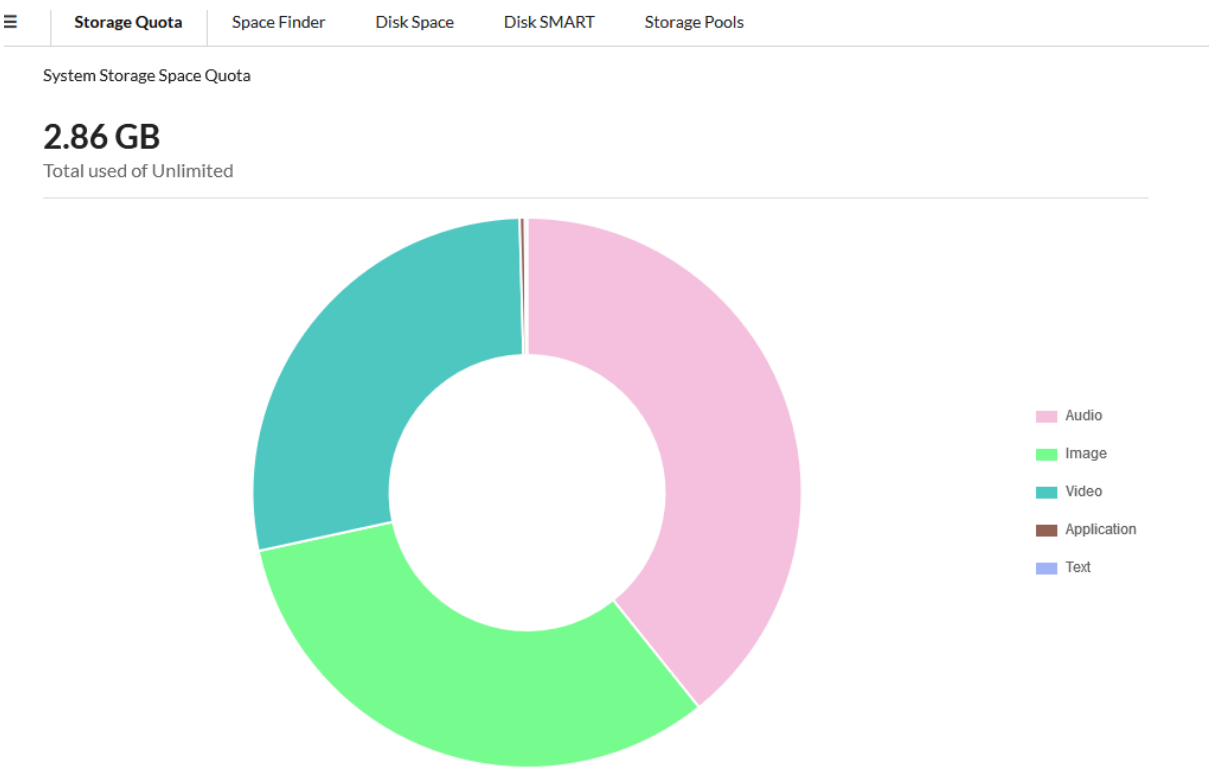
Example handle functions

```
router.HandleFunc("/system/file_system/validateFileOpr",
system_fs_validateFileOpr)
router.HandleFunc("/system/file_system/fileOpr", system_fs_handleOpr)
```

Quota Manager (Quota)

The quota manager is a module that manuage a user storage quota. It keeps track of all file operations within the system (excluding files accessed by subservices). The storage manager can be accessed via the user editing / group editing tool.

The quota manager also provides a graphical user interface for displaying a given user's storage quota and its file type classifications.



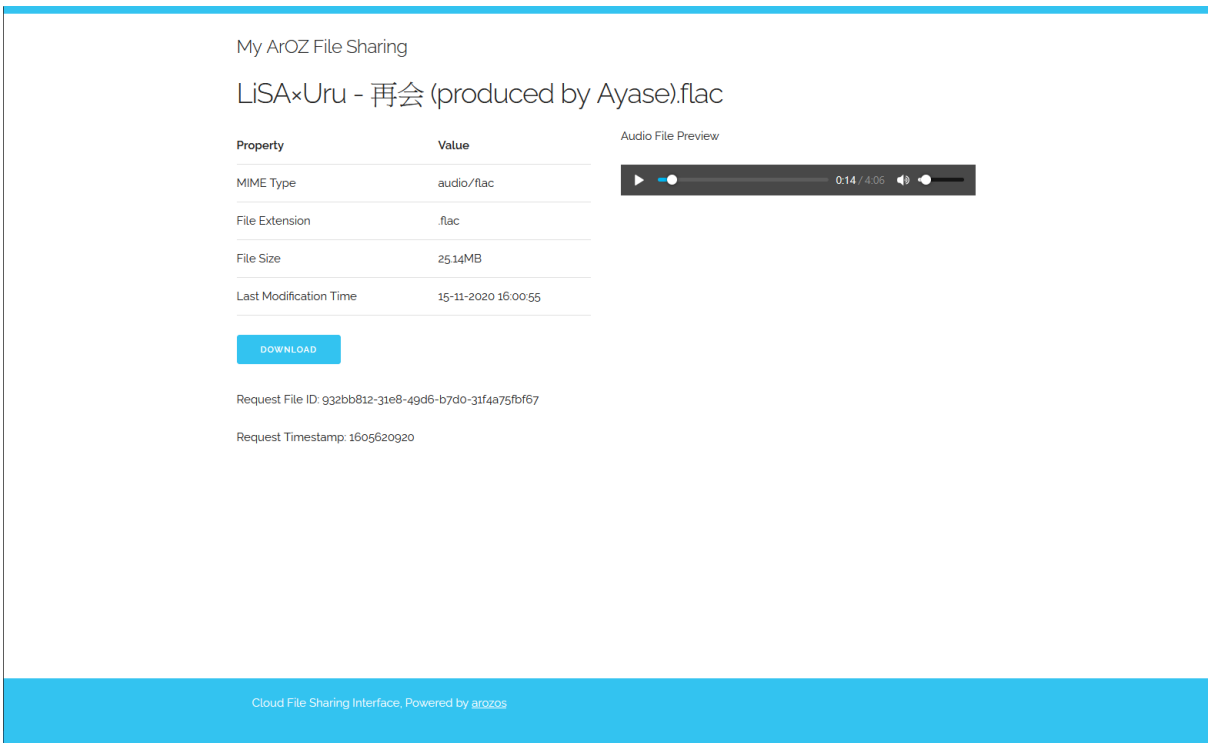
The diagram might take time to load due to its heavy disk IO request.

Share

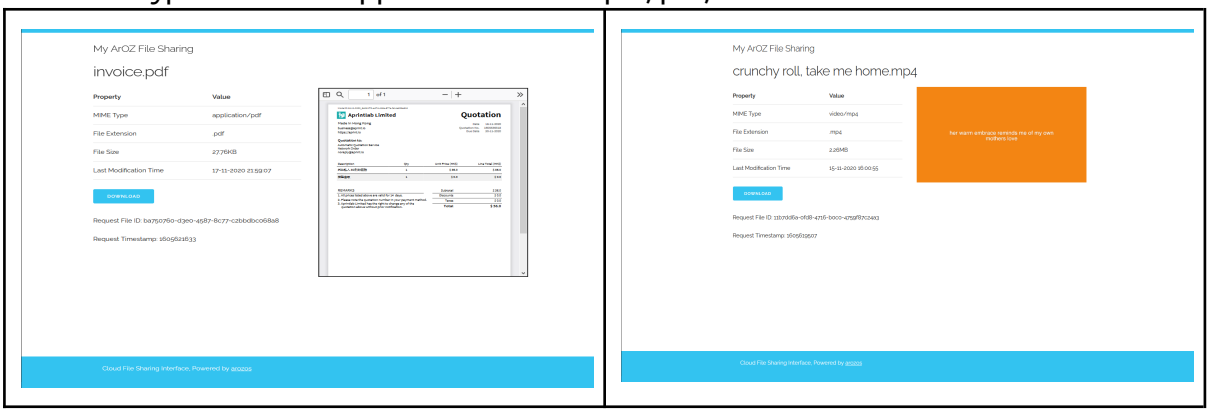
The share module handles all share request and file access request for shared file objects. This module handles

- Share file creation
- Edit shared files
- Remove file share
- Serving shared files
- Keep track of all the shared item UUID and its real filepath

This module is also a special module that would handle its own UI file serving. The following is an example of a shared file download interface for non-logged in users.



Other file types are also supported. For example, pdf, video and others



This module also handles share cleaning by removing the share UUID which the file that the UUID points to no longer exists. When this event occurs, the File Not Found UI will be served instead.

Storage

The Storage module handles most of the tasks related to storage pools and user storage allocation functions.

Storage Pools

Storage Pools are a group directory on the host operation system that is assigned to a certain user or permission group.

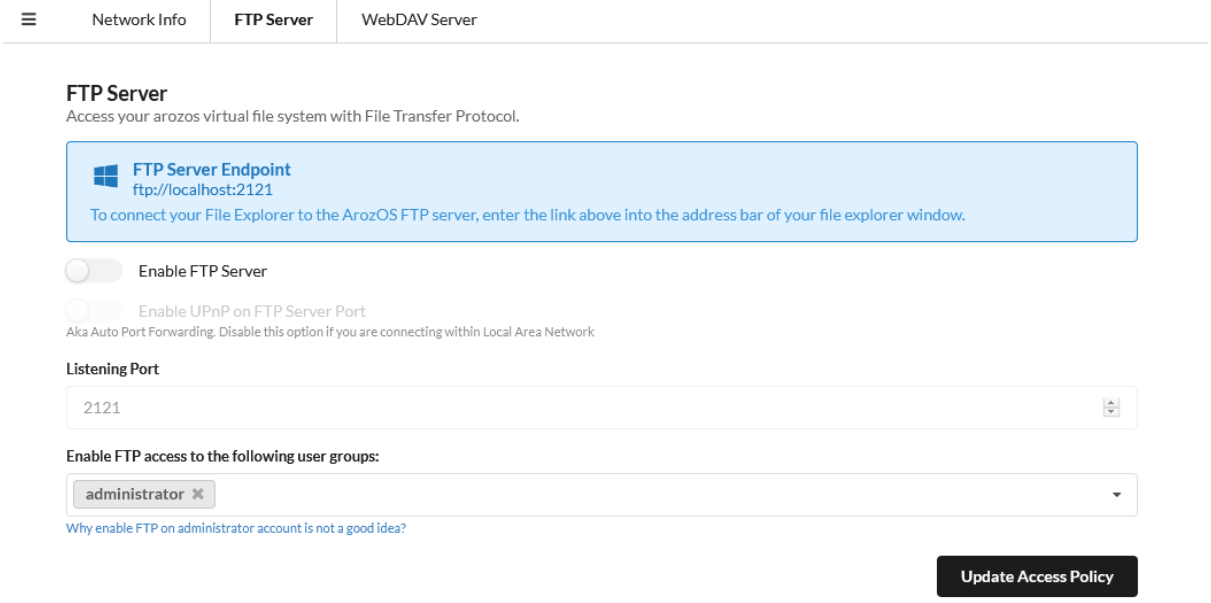
A storage pool is consists of a basic structure as follows

Key	Type	Usage
Owner	String	Owner of the storage pool, either a username or group name
OtherPermission	string	Permission on non-owner user / user group. Using the same keywords as file system handlers for permission management.
Storages	[]*fs.FileSystemHandler	A list of usable file system handler under this storage pool permission

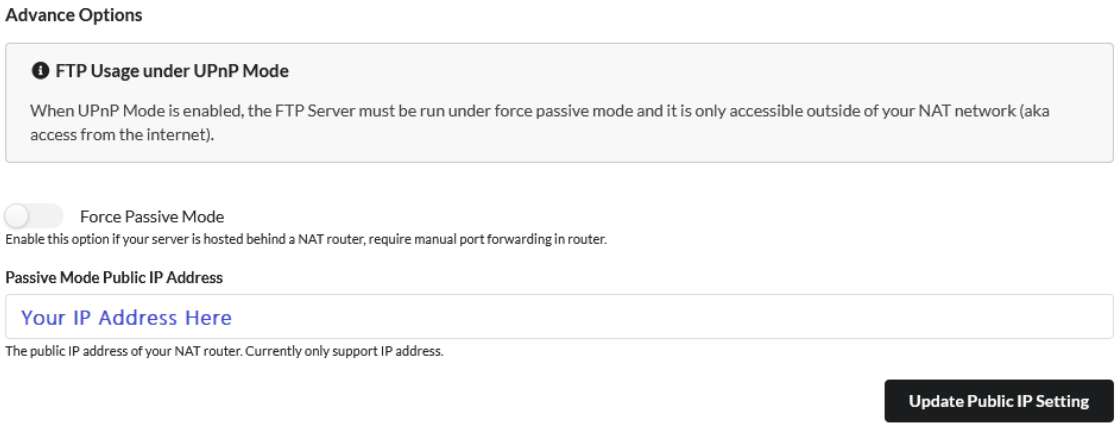
FTP Storage

FTP Storage module provides access to the user roots and all his virtual storage root through File Transfer Protocol. This module will create a folder inside the user root named “ftpbuf” when FTP storage is enabled and the user tries to access his storage root using FTP clients like Windows File Explorer or Filezilla.

The FTP Server related settings can be found inside the Network > FTP Server tab inside system settings.

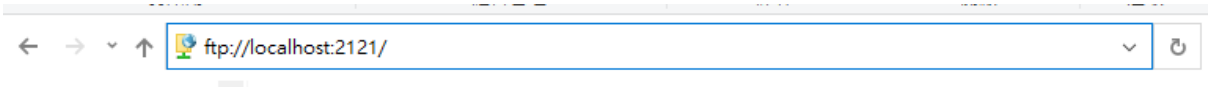


If your ArozOS host is hosted behind a NAT router, you might also need to set up the passive mode for your FTP server. In the lower part of the setting interface, enable Force Passive Mode and enter your public IP address into the input box below the checkbox. This will be automatically filled up if you have enabled ArozOS with UPnP.

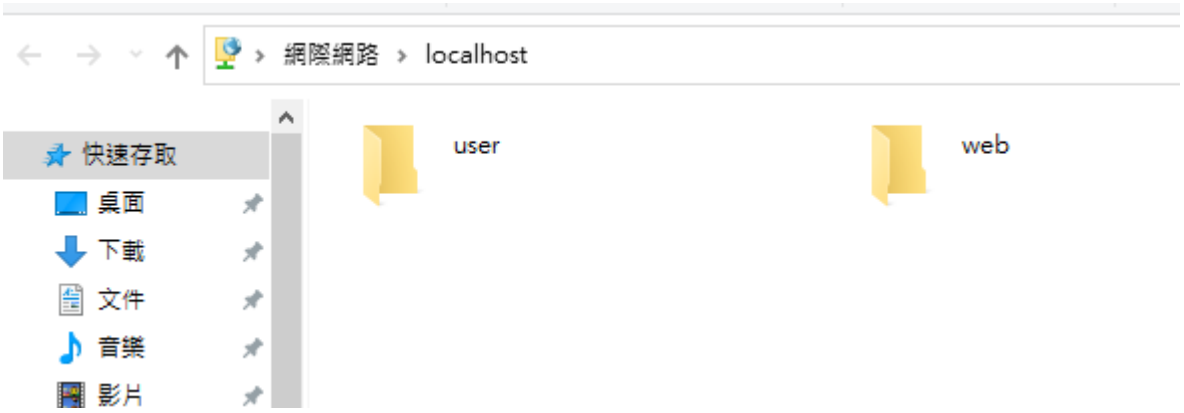


Connecting FTP under Windows

To connect to the FTP server under Windows, open File Explorer and enter the ftp address copied from the setting page into the address field. In this example, “ftp://localhost:2121” was used.



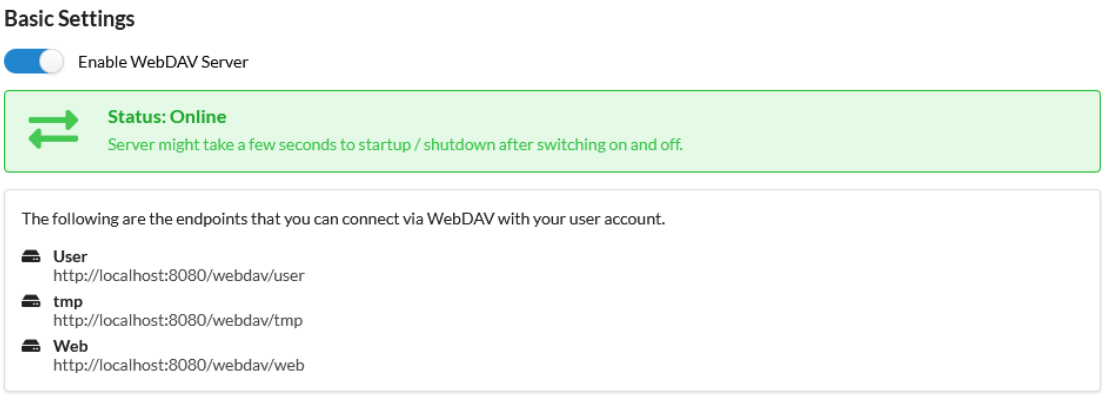
Enter your aroz username and password to continue. After connection is established, you will see your vroots shown in the file explorer. Note that the tmp folder will not be shown under FTP server mode.



WebDAV Storage

The WebDAV storage is an alternative network access method for ArozOS. WebDAV is recommended to be used with TLS mode enabled. If TLS mode is not enabled, users can still use WebDAV but some extra steps will be required on Windows.

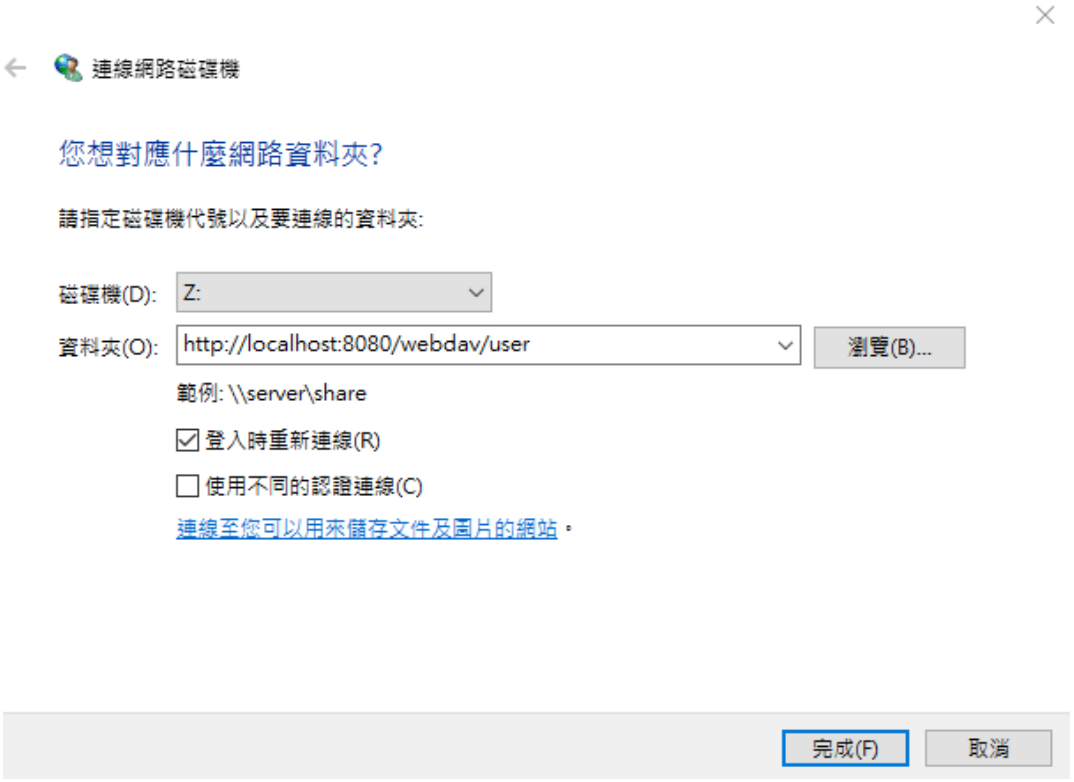
To enable WebDAV server, visit System Settings > Network > WebDAV Server, scroll to Basic Settings and enable the WebDAV Server via switching the button on.



The endpoint for your user account has been listed in the message box below. In the default case, you should see a user root, a tmp folder and web if you are an administrator.

Setup WebDAV On Windows (Non TLS Mode)

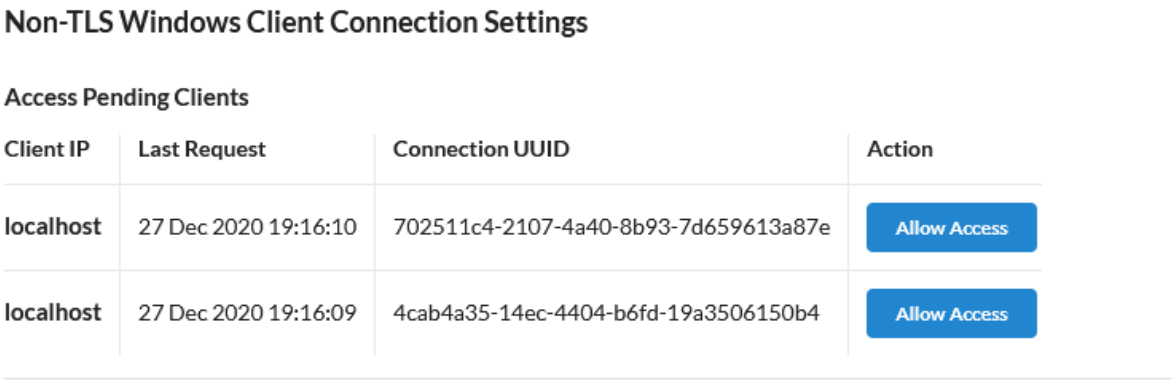
Setting up WebDAV On Windows is a bit tricky for non TLS mode ArozOS system. First, open “My Computer” > “Map Network Drive”
For the folder, enter the URL shown on the vroot you want to mount. In this example, “http://localhost:8080/webdav/user” will be used.



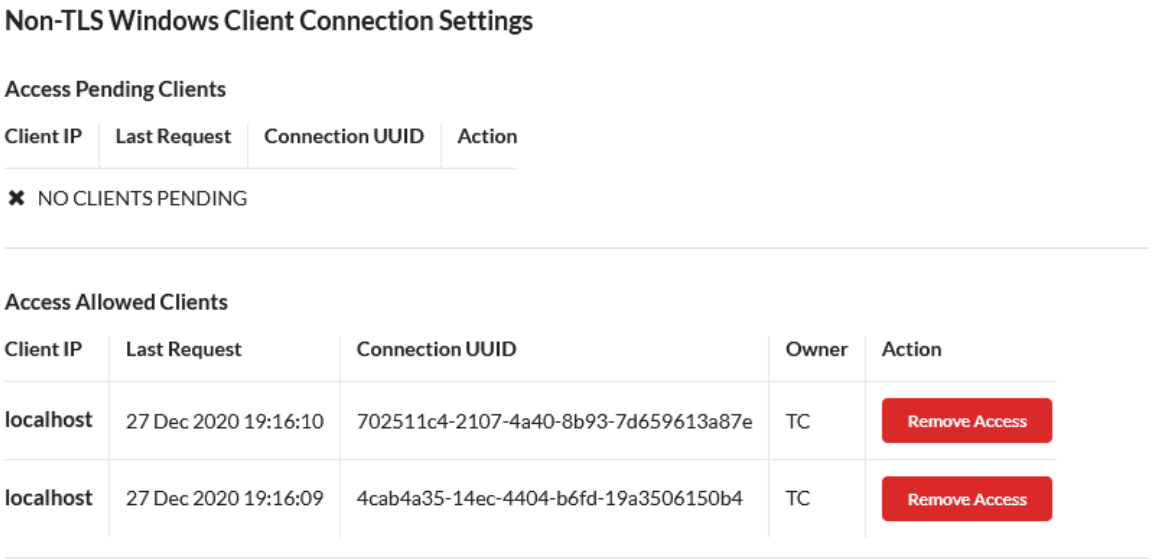
DO NOT ENTER PASSWORD DURING THE FIRST CONNECTION

After the drive has been connected,you will see a blank, read only folder mounted to your device.

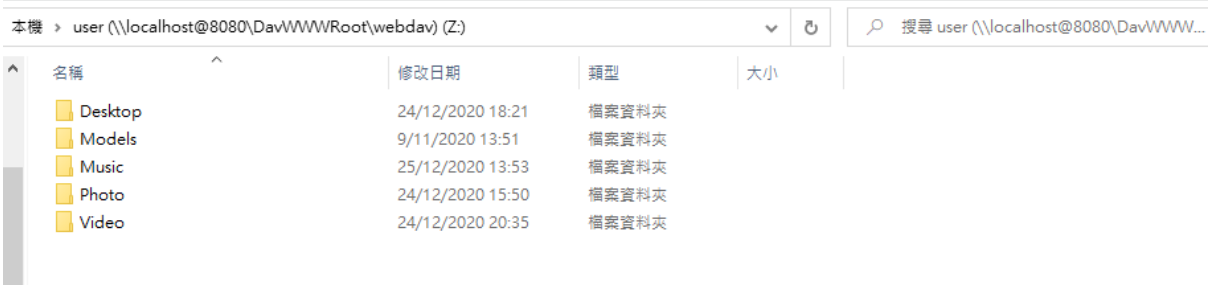
Now, move back to your System Setting Interface. Refresh the “WebDAV Server” tab or click on the “Refresh List” button. You will see some new connection appears on your list of “Access Pending Clients”



Click “Allow Access”. The one you clicked will be moved to the table below showing that you now grant access to these clients (aka File Explorer instants)



Finally, get back to your File Explorer and click on the refresh button. All your folder inside your user root should be shown as follows.

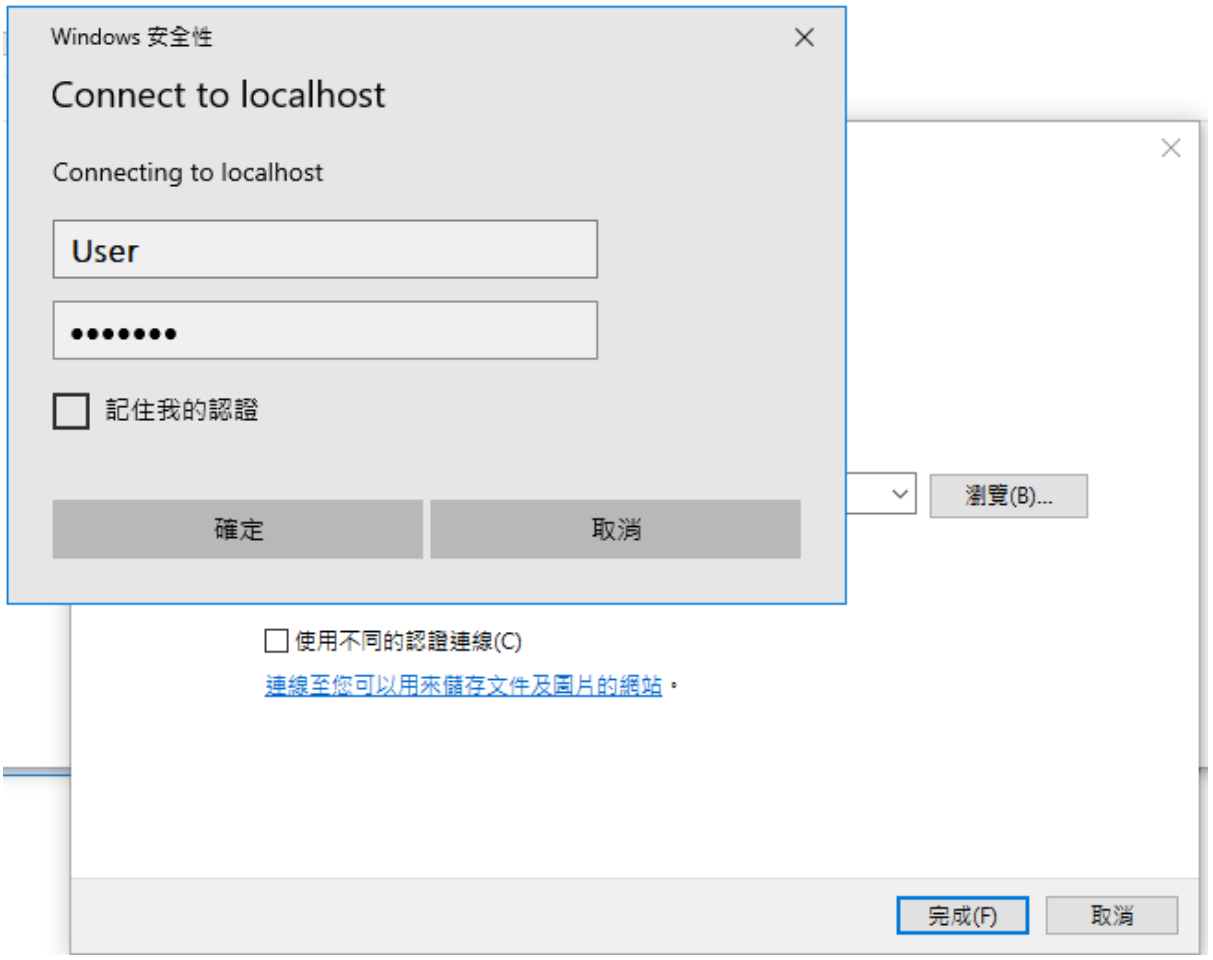


Setup WebDAV On Windows (TLS mode)

Setting up WebDAV On Windows is a bit tricky for non TLS mode ArozOS system. First, open “My Computer” > “Map Network Drive”

For the folder, enter the URL shown on the vroot you want to mount. In this example, “http**s**://localhost:8080/webdav/user” will be used.

Next, a login interface will pop up and ask for your username and password. Enter your aroz username and password to proceed.

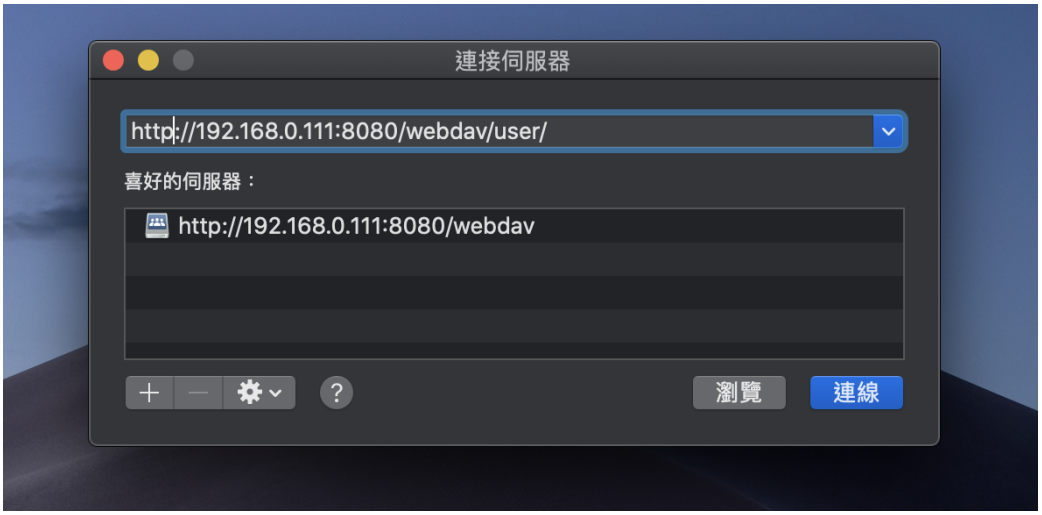


After connection is established, you will see your vroot folders inside the mounted drive.

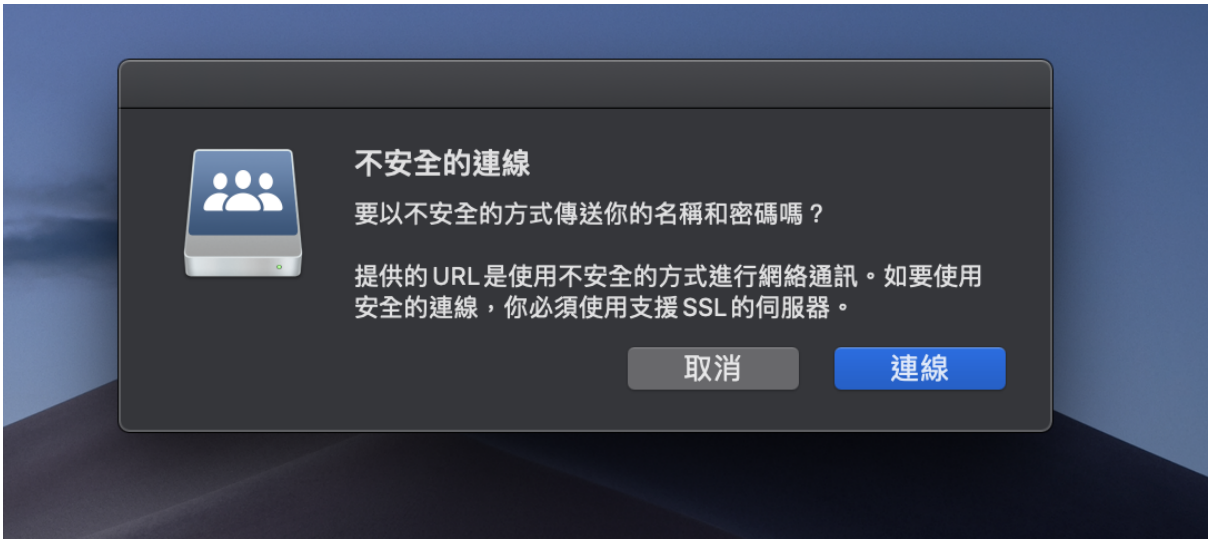
Setup WebDAV On MacOS (TLS and non TLS)

To connect WebDAV server on MacOS, start Finder, click on “Go” → “Connect to Server”

Next, enter the endpoint shown in the system setting page. In this example, "http://localhost:8080/webdav/user" will be used.

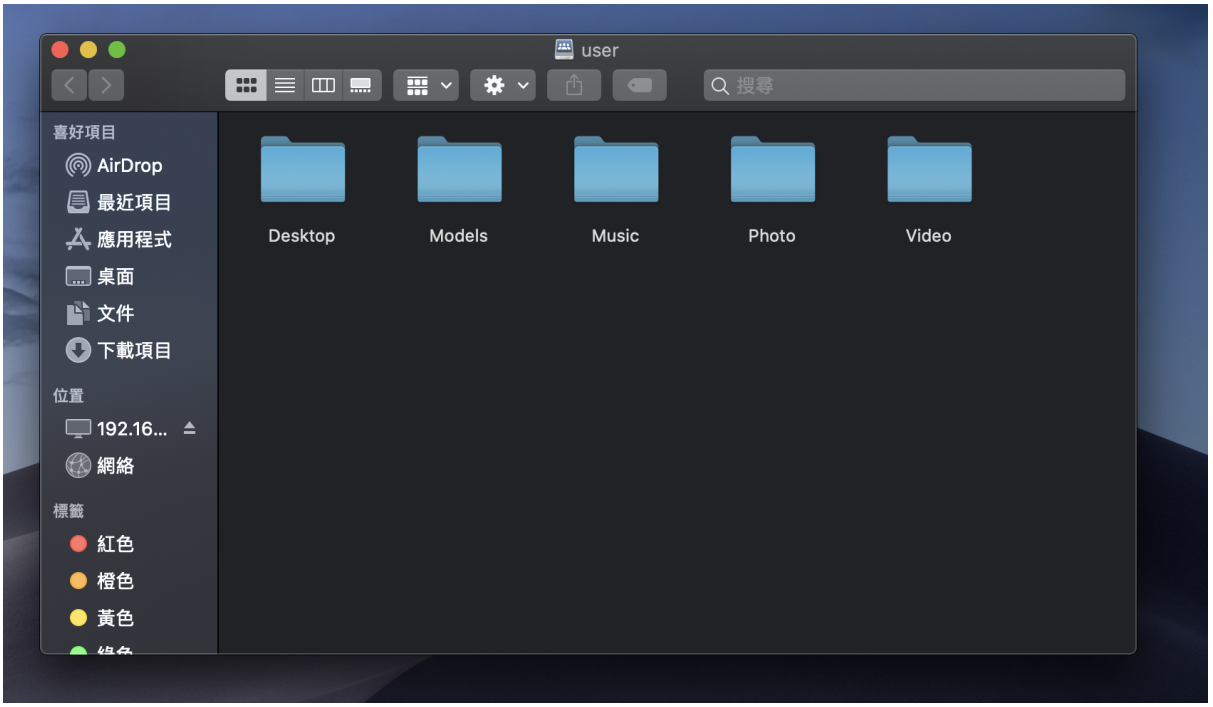


If a security warning pops up in the case where you are using non-TLS mode, click accept and continue.



Then, login with your aroz username and password.

After login, you will see a new mounted drive on your desktop and a Finder window shows up with your vroot contents.



Subservices

Subservice is one of the methods of expanding the arozos cloud system with external software. This module handles the mounting and reverse proxy automation for subservice within the same host. For more details on how to program and set up a subservice, see “Subservice Programming '' section.

User

The user module handles user permissions, storage pools and user account resolution services. The userHandler object is the core of the whole arozos system and can be requested to return the system database, authentication agent, permission manager and other system components binded to this handler through public interfaces (As described in the Object Orientation Programming architecture, in Golang’s term: Public Access Functions (or even simpler, the functions start with capital letters)).

The user handler provide four main category of services

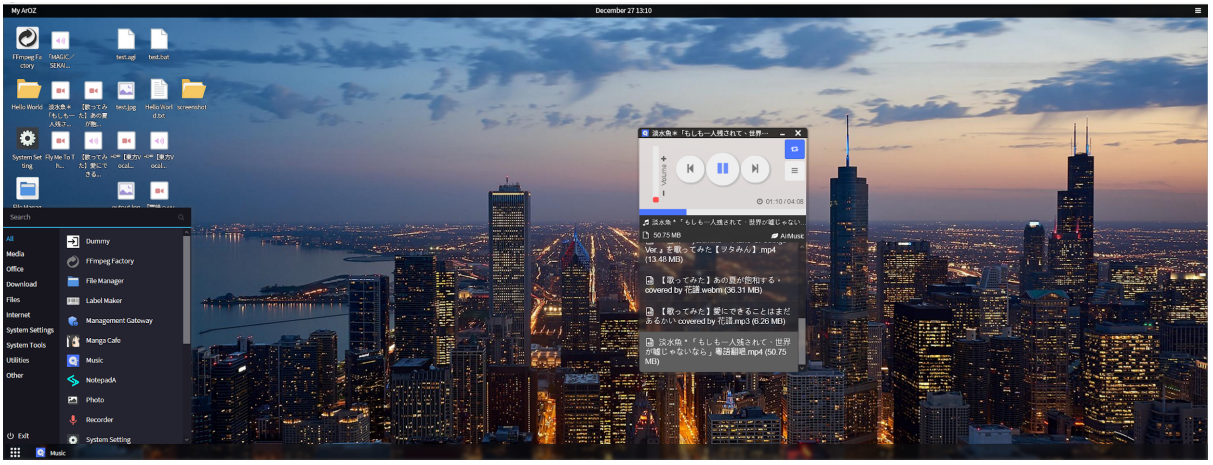
1. Directory Handlers (e.g. Handle virtual path to real path translation)
2. Permission Handler (e.g. Handle user permission checking)
3. Quota Handler (e.g. Handle user storage quota)
4. User Info Resolver (e.g. Handler get user information from request or username)

Note that no parts of this module is handling the user authentication, user creation or permission group assignment. This is the top level module that binds everything together.

System Components

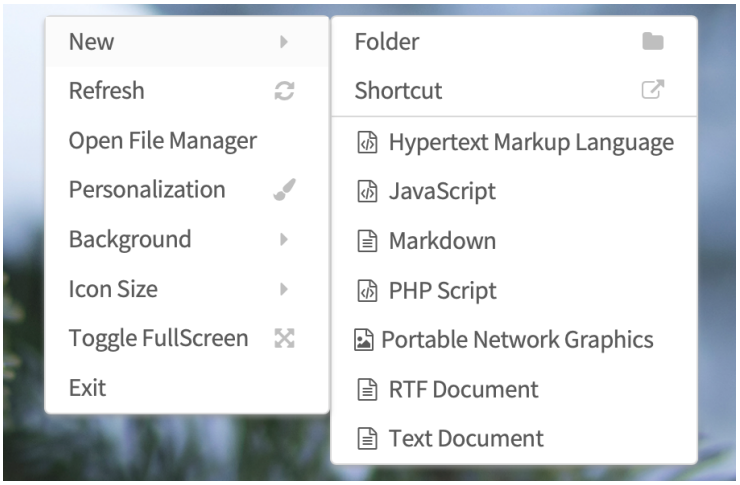
Desktop

The Web Desktop system is one of the most iconic functions of the ArozOS system. Featuring a full fledged Web Desktop experience with drag drop supports, you can do most of your desktop routine on your ArozOS host just like what you would normally do on your PC.



Creating New Files / Folder

To create a new file or folder on the desktop, right click the desktop on any place that has no file. A context menu should show up afterwards. Select “New” → “Folder” or any new type of files you want to create by clicking the target format types.



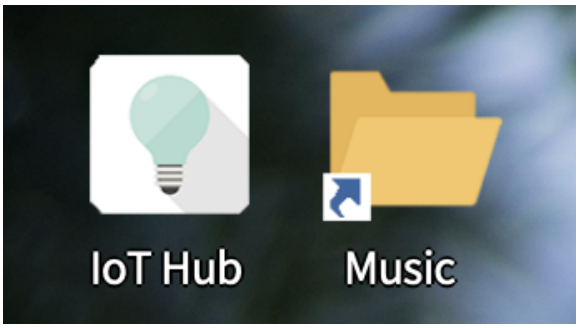
Creating Shortcuts

To create a shortcut, click New → Shortcut. A shortcut creation assistant should pop up and you can choose to create a shortcut of one of the three type of objects in the system.

1. WebApp modules, any kind of modules or subversive installed on the system. Will open that module once double clicked
2. Folder / Directory, any virtual path within the system that is accessible by the user can be opened one double clicked on the shortcut icon.

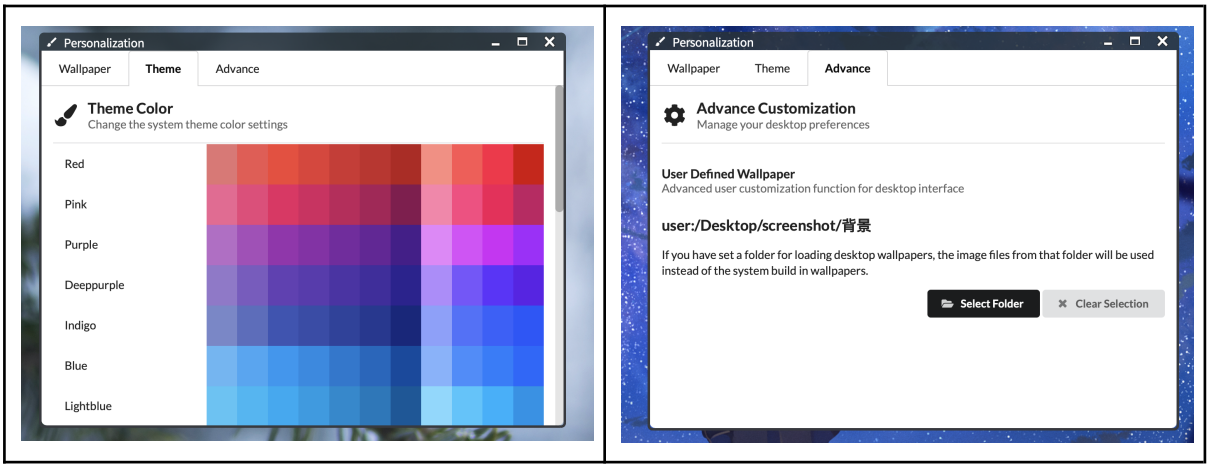
3. Website URL, any link will be open with a new popup window when double clicking the shortcut icon

Once the shortcut is created, you will see a new icon popup on your desktop. The following shows the shortcut of the IoT Hub and the Music folder shortcut.

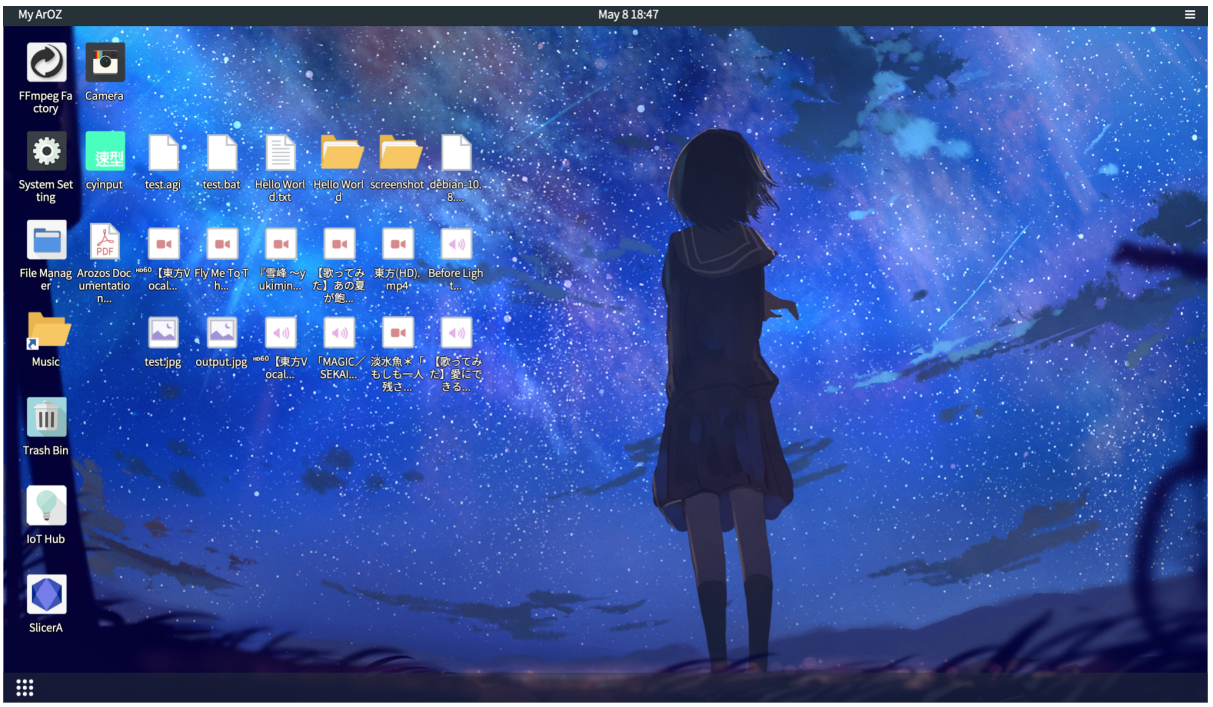


Customizing Desktop

You can customize your desktop using the personalization option in the context menu or use the “background” option of the context menu for a quick background swap. To change the background wallpaper and the theme color, open the “Personalization” utility and set the theme color and customize a folder for wallpaper image under the “Theme” and “Advance” tab.



The following examples show a desktop with customized theme color and wallpaper

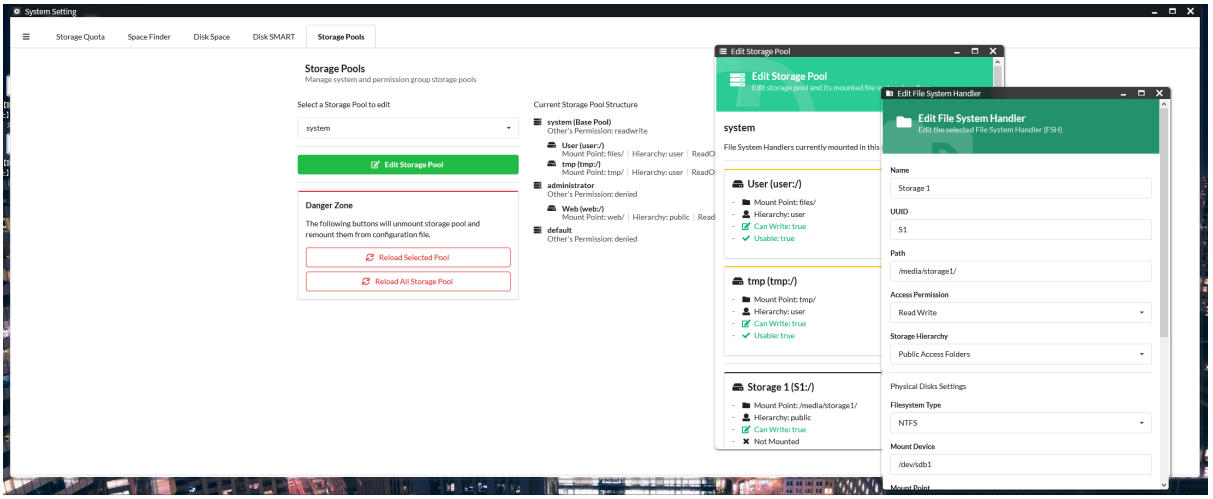


Uploading File

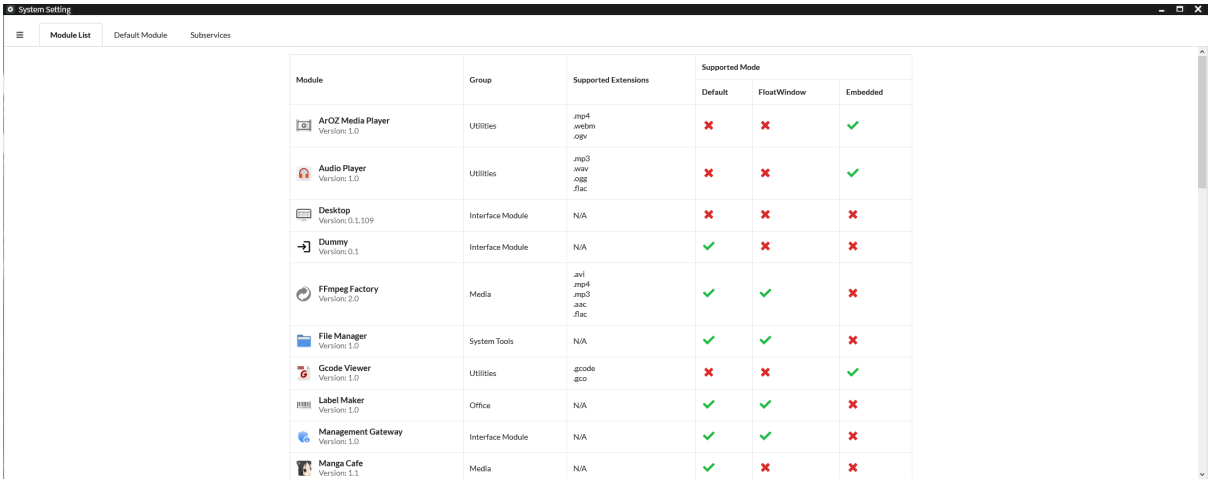
To upload files to the desktop, simply drag and drop them to the location you want on the desktop. It is also possible to download files from the desktop to local devices using the drag and drop method. But this method is only supported on some versions of Chrome on Windows.

System Settings

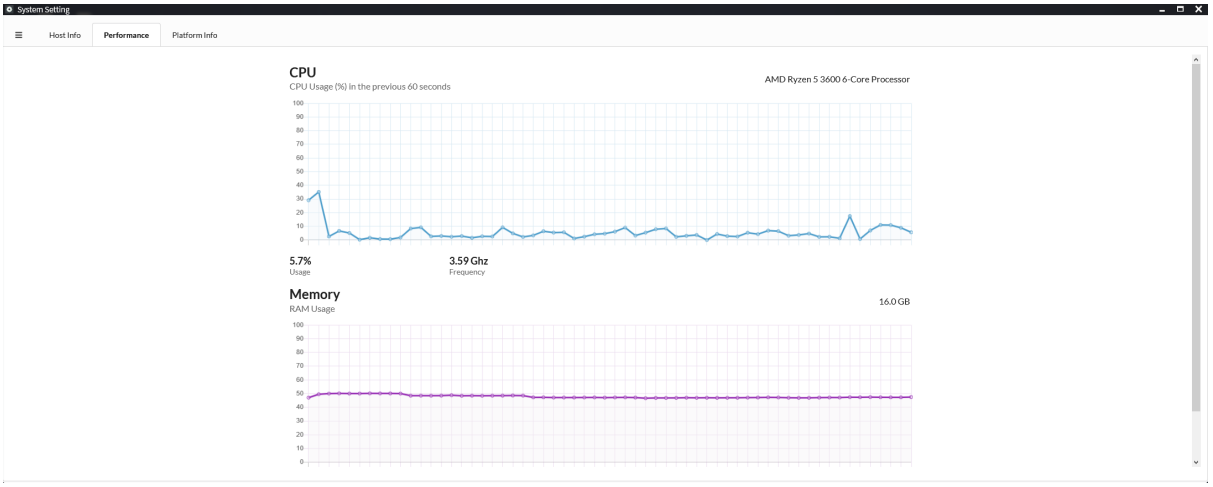
System Settings is a built- in module that allows users to adjust their system settings through the Web UI without the need to enter the ssh terminal. You can easily change settings on users, network services and even disk management in this easy to use interface.



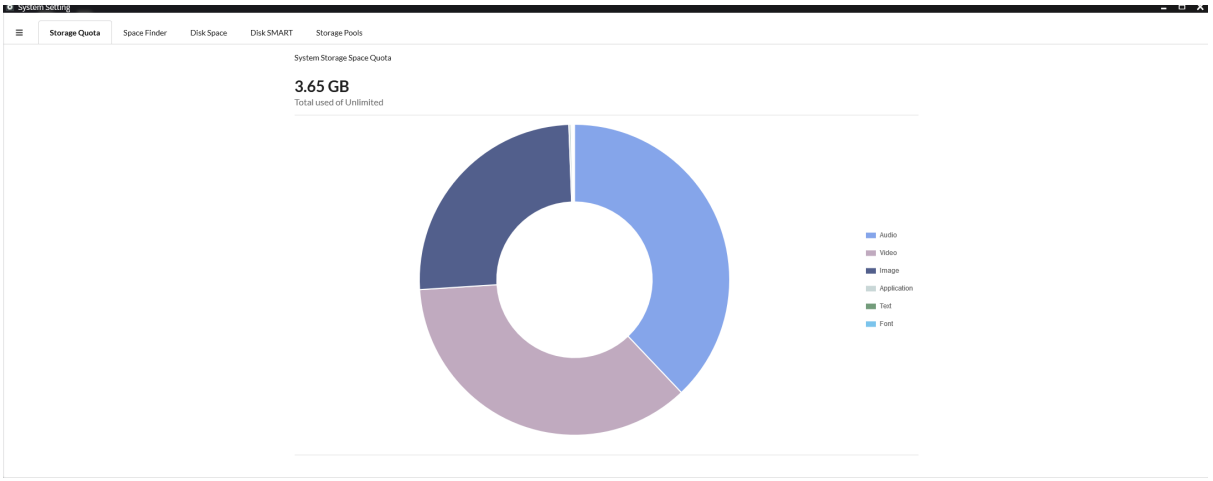
Storage Pool Management Interfaces / Windows



Module Management Interface



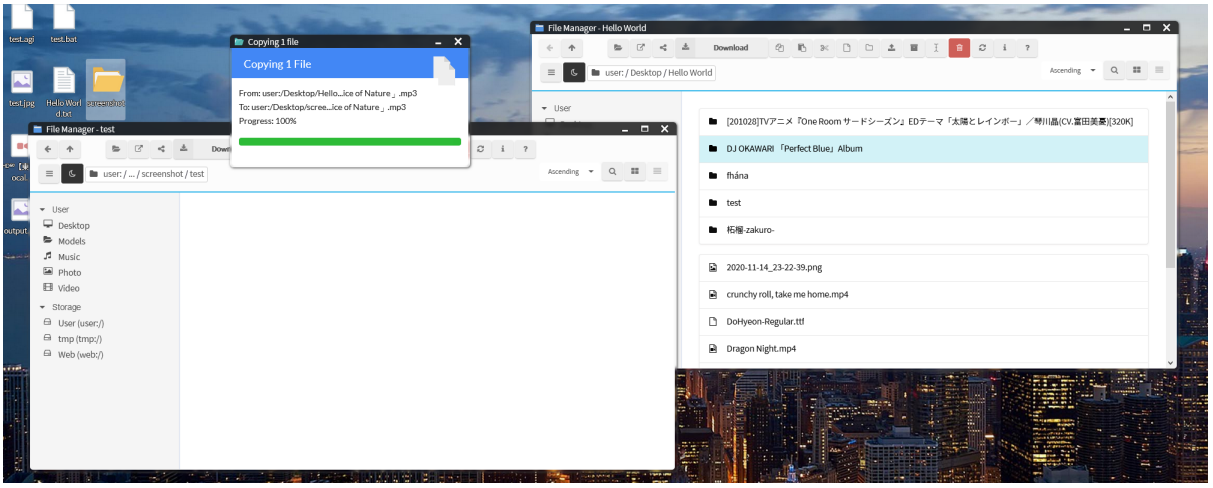
Performance & Resources Monitor



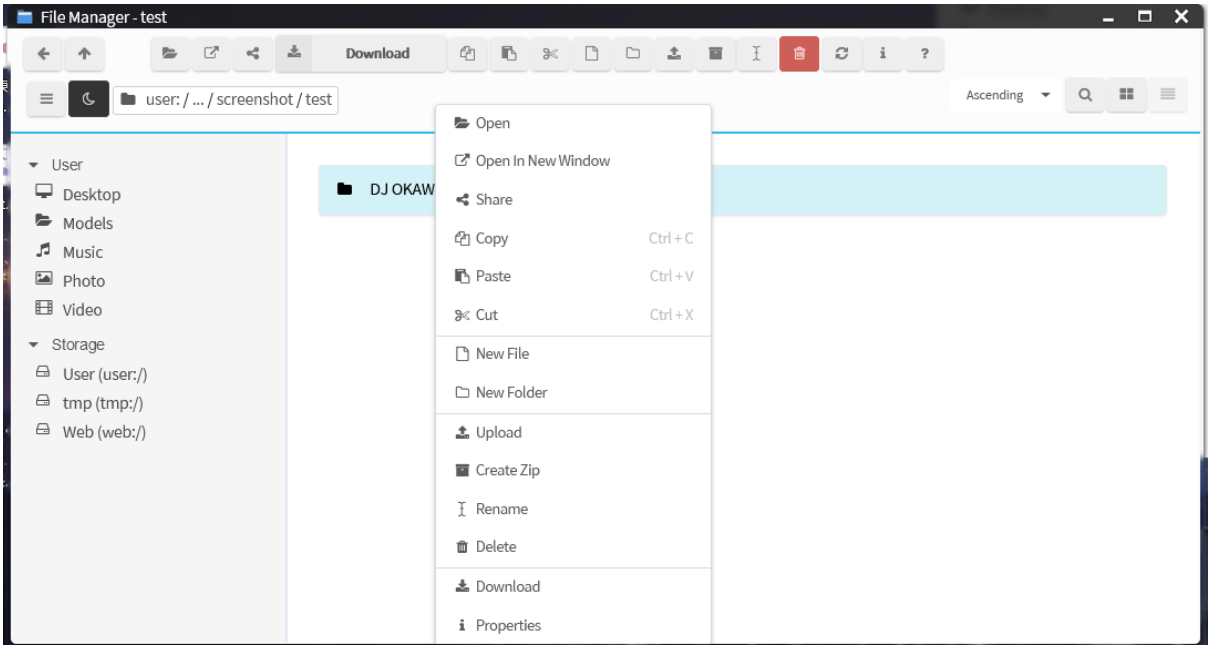
Storage Management

File Manager (File Explorer)

File Explorer is another key element of ArozOS that allows easy and quick access to files stored on your ArozOS host devices. File Manager supports drag drop upload, multi thread upload, file operations liek copy, paste, move, zip, search and more.



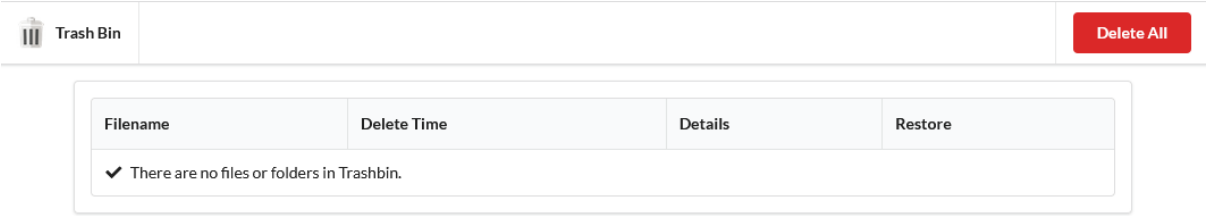
File Operations between multiple windows



Context Menu of the ArozOS File Manager

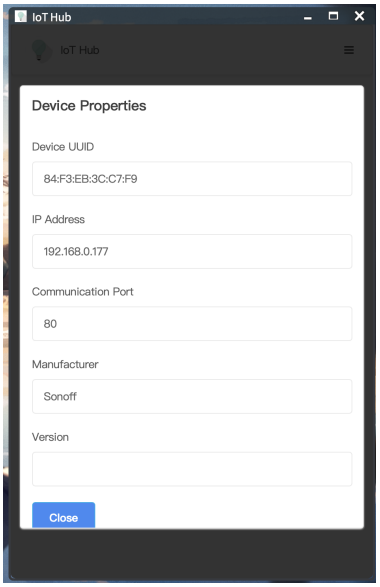
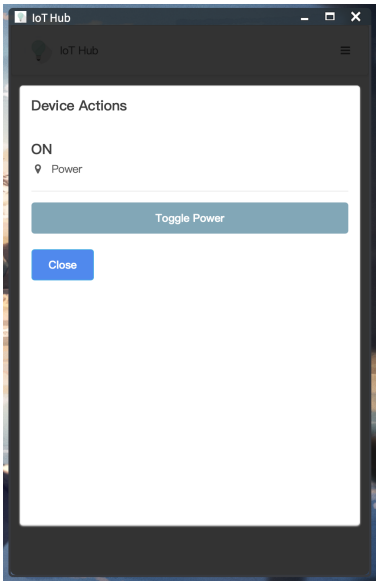
Trash Bin

The trash bin works by creating a hidden trash folder within the trash root folder. Under the root operation system, you can access the trash folder by <current_folder>/.trash/. You might need to enable the “Show hidden folder” option on Windows hosts.

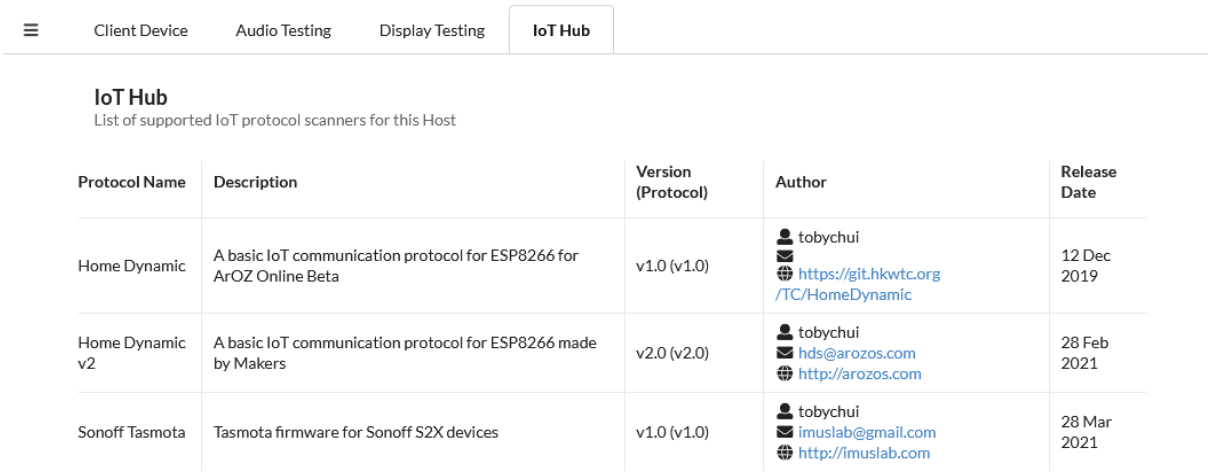


IoT Hub

The IoT Hub is a built- in WebApp that supports IoT Scanning ability through defined IoT Scanners. The IoT Scanner is expandable in the system core and allows users to add in any IoT device scanner if they know how it works. In the default situation, only the Home Dynamic protocol and SonOff S2X scanner is included in the main system core.

		
Main Control Interface. Names can be customized using the edit button.	Device Properties tab, showing the devices IP address and UUID.	Device Action tab. This is defined by the device manager and the iot device.

To check which type or brand of IoT devices that your host system is supporting, you can visit the system settings → Device and IoT → IoT Hub for a list of installed IoT device managers.



IoT devices can be accessed and controlled by agi script using JavaScript as the main programming language. See the system example code and ArOZ JavaScript Gateway Interface section for more information.

Home Dynamic v2 Protocol

The Home Dynamic v2 is the default IoT protocol for the ArozOS IoT Hub. It supports ESP8266 based devices and allows it to be controlled over the local area network with no security settings. To allow your ESP8266 device to be scannable by the ArozOS HDSv2 scanner, make sure to broadcast the required mDNS information using Dynamic Service TXT Callback function in your ESP8266 code. Here is an example of such broadcast callback.

```
void MDNSDynamicServiceTxtCallback(const MDNSResponder::hMDNSService
p_hService) {
      //Define the domain of the HDSv2 devices
      MDNS.addDynamicServiceTxt(p_hService, "domain","hds.arozos.com");
      MDNS.addDynamicServiceTxt(p_hService, "protocol","hdsv2");

      //Define the OEM written values
      MDNS.addDynamicServiceTxt(p_hService, "uuid",getMacAddress());
      MDNS.addDynamicServiceTxt(p_hService, "model","Generic");
      MDNS.addDynamicServiceTxt(p_hService, "vendor","HomeDynamic
Project");
      MDNS.addDynamicServiceTxt(p_hService, "version_minor","0.00");
      MDNS.addDynamicServiceTxt(p_hService, "version_build","0");
}
```

For a full example, visit the Home Dynamic System repository at https://github.com/tobychui/arozos.

Cache Renderer

The cache renderer is a system component used by the ArozOS File Manager for rendering preview for the following file mime types

- Video (e.g. mp4, avi, webm)
- Audio (e.g. mp3, flac)
- Photo (e.g. png, jpg, gif)

The Cache Renderer outputs a cache image file located relative to the file being cached will the path ".cache/{filename}.jpg" with a size of 480 x 480 pixel.

The Video cache rendering is based on FFmpeg and will extract the frame on the 5 second mark as a preview image. If the video is shorter than 5 seconds, no preview will be generated. This function requires FFmpeg to be installed on the Host System.

The Audio cache rendering is based on extracting the audio meta info and generating the album art using the Golang jpeg library. The album art caching only works when there is a valid image meta data inside the audio file. See the following library for more information.

- "github.com/dhowden/tag"

The Photo cache rendering is based on shrinking the original image to the desired cache size. See the following library for further development

- "github.com/nfnt/resize"
- "github.com/oliamb/cutter"

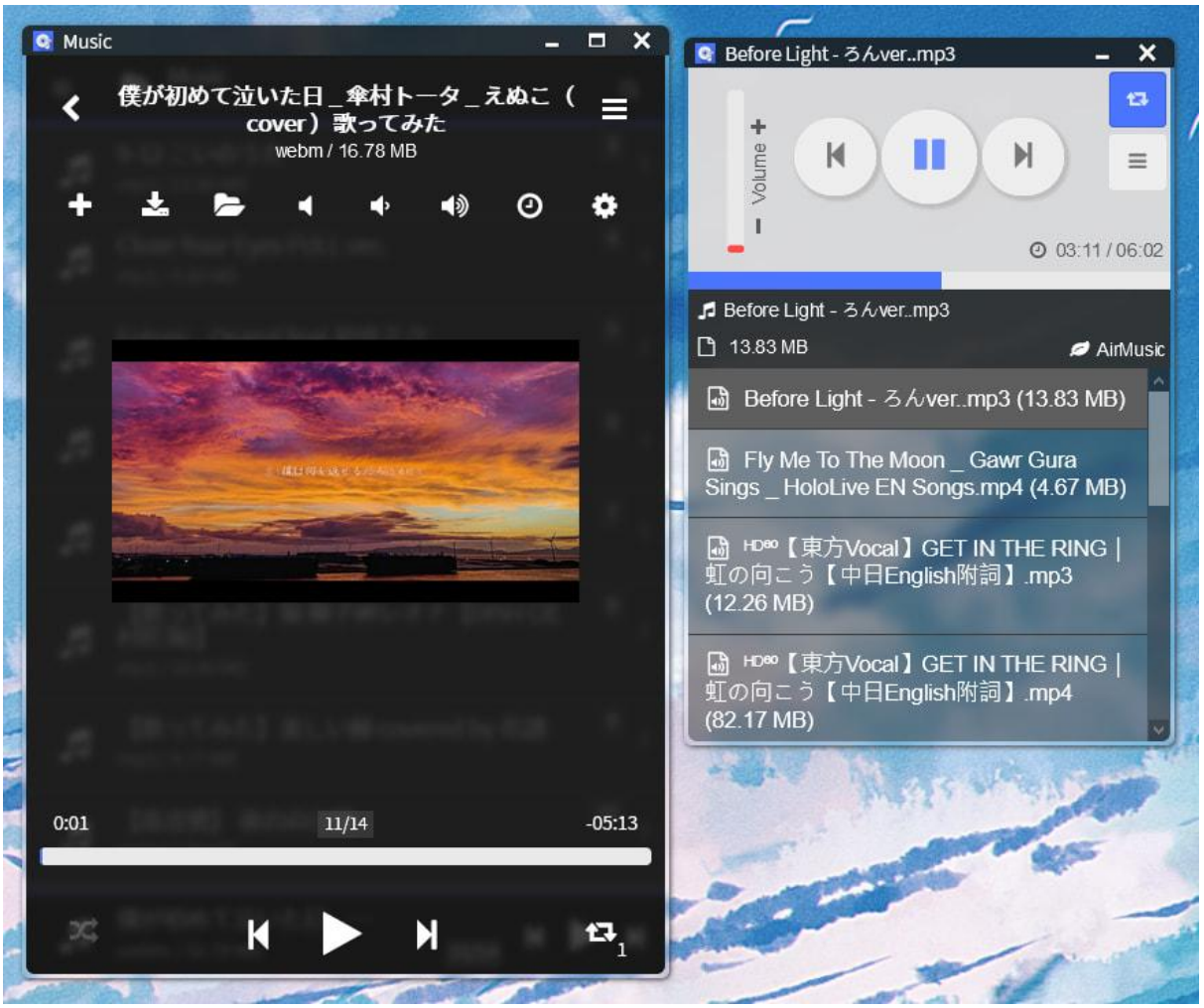
Utilities

Utilities are modules that are hard coded into the system core. These utilities only exist here for most basic file viewing purposes. Examples are Audio / Video file player etc.

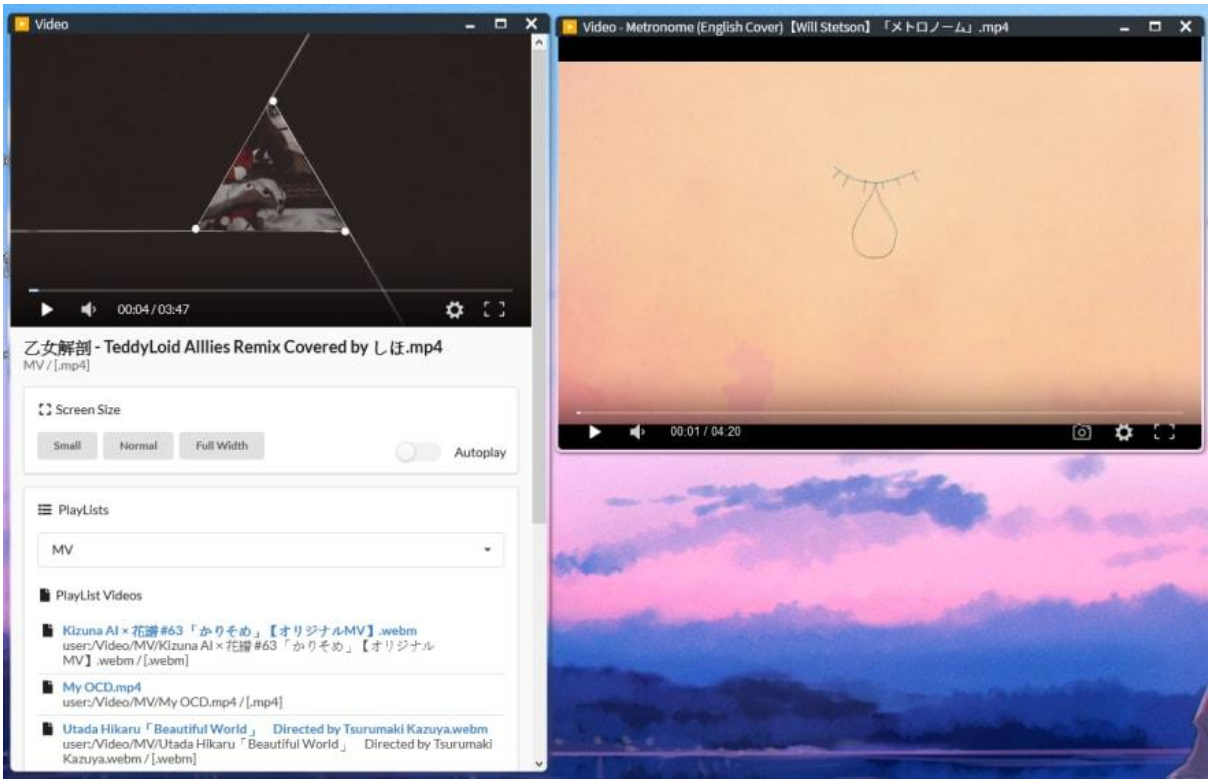
Build in WebApps

The following showcase some preview of the build in WebApps

Music


Music
僕が初めて泣いた日 _ 傘村トータ _ えぬこ（cover）歌ってみた
webm / 16.78 MB
0:01
11/14
-05:13
Before Light - ろんver..mp3
Volume
03:11 / 06:02
Before Light - ろんver..mp3
13.83 MB
AirMusic
Before Light - ろんver..mp3 (13.83 MB)
Fly Me To The Moon _ Gawr Gura Sings _ HoloLive EN Songs.mp4 (4.67 MB)
【東方Vocal】GET IN THE RING | 虹の向こう【中日English附詞】.mp3 (12.26 MB)
【東方Vocal】GET IN THE RING | 虹の向こう【中日English附詞】.mp4 (82.17 MB)

Video


Video
00:04 / 03:47
乙女解剖 - TeddyLoid Alllies Remix Covered by しほ.mp4
MV / [.mp4]
Screen Size
Small
Normal
Full Width
Autoplay
PlayLists
MV
PlayList Videos
Kizuna AI × 花譜 #63「かりそめ」【オリジナルMV】.webm
user:/Video/MV/Kizuna AI × 花譜 #63「かりそめ」【オリジナル MV】.webm / [.webm]
My OCD.mp4
user:/Video/MV/My OCD.mp4 / [.mp4]
Utada Hikaru「Beautiful World」 Directed by Tsurumaki Kazuya.webm
user:/Video/MV/Utada Hikaru「Beautiful World」 Directed by Tsurumaki Kazuya.webm / [.webm]
Video - Metronome (English Cover) 【Will Stetson】「メトロノーム」.mp4
00:01 / 04:20

Photo

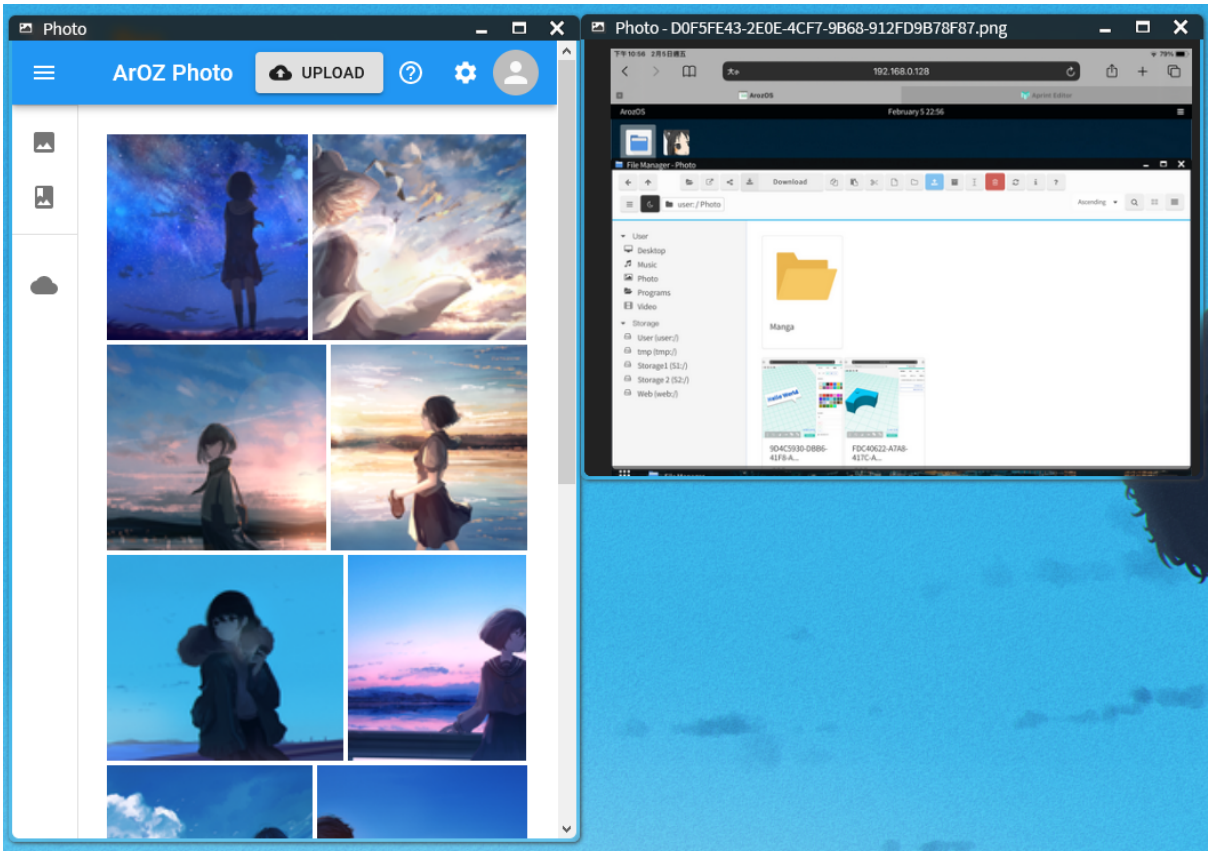
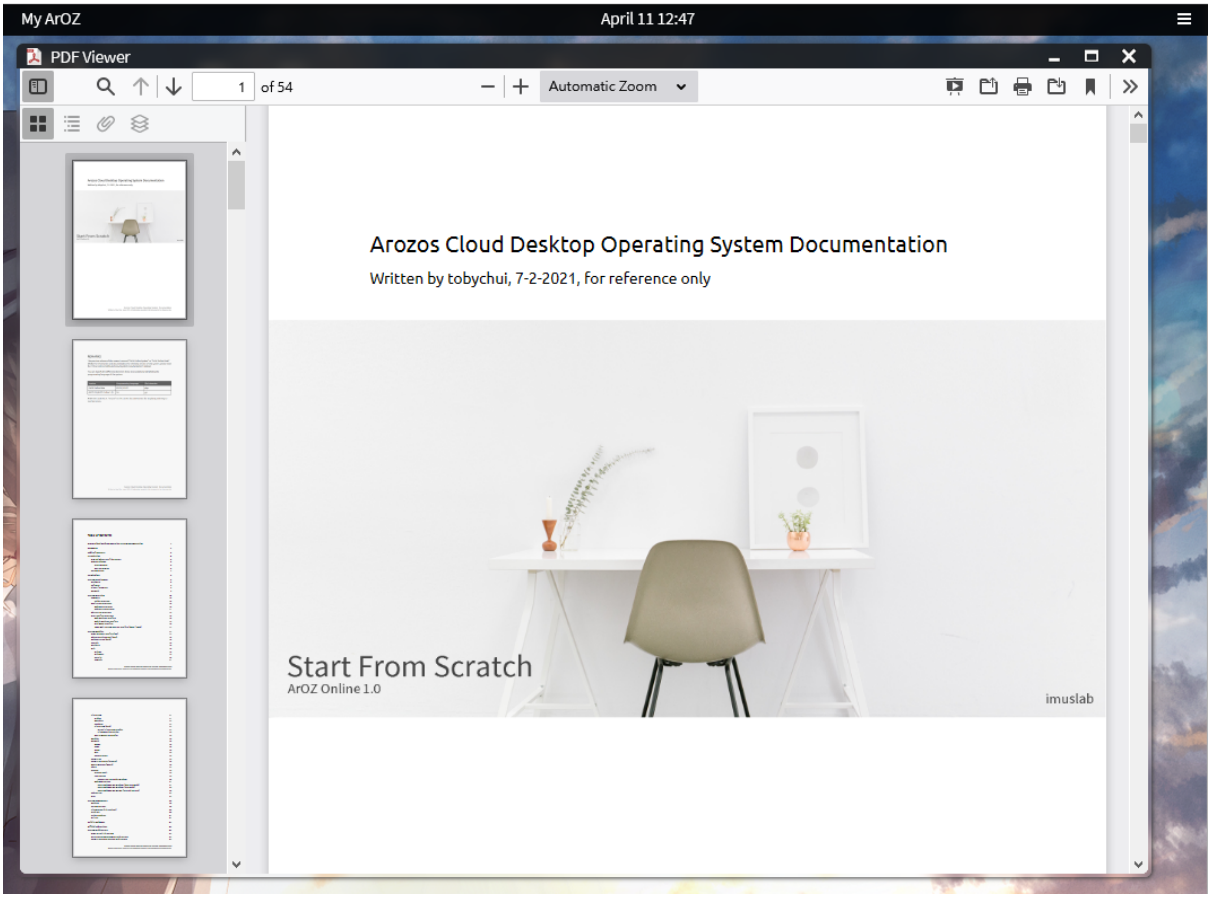
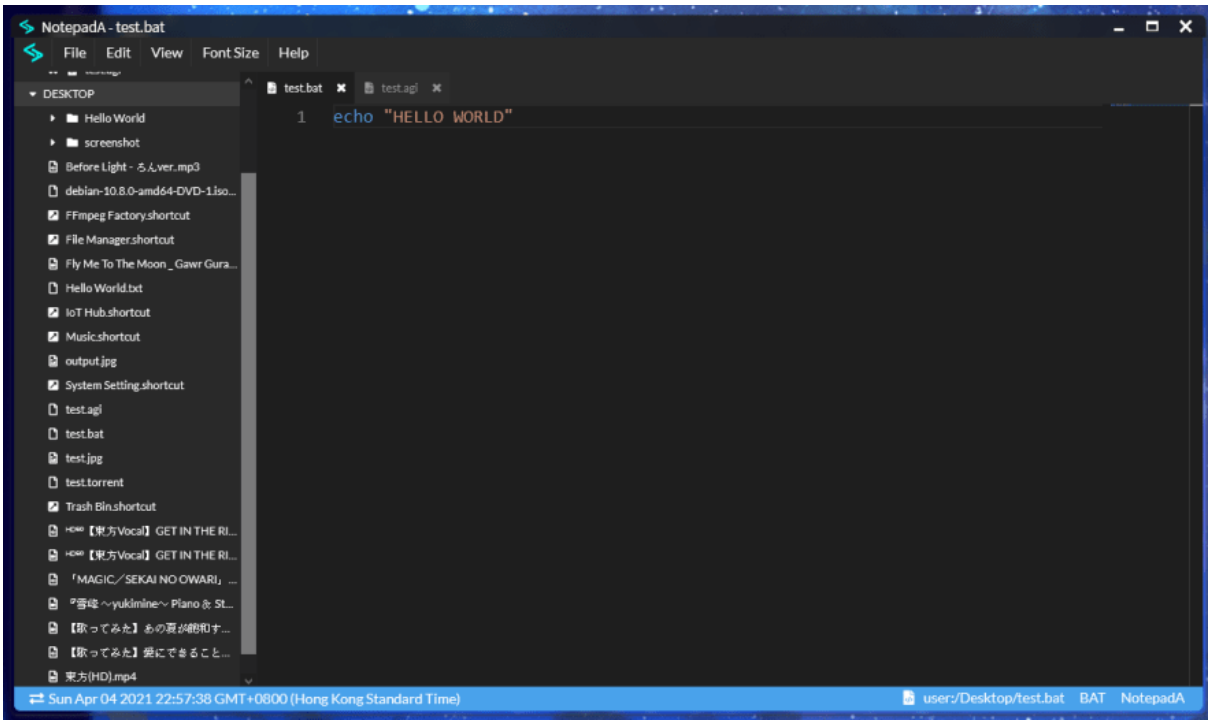


Photo
ArOZ Photo
UPLOAD
Photo - D0F5FE43-2E0E-4CF7-9B68-912FD9B78F87.png

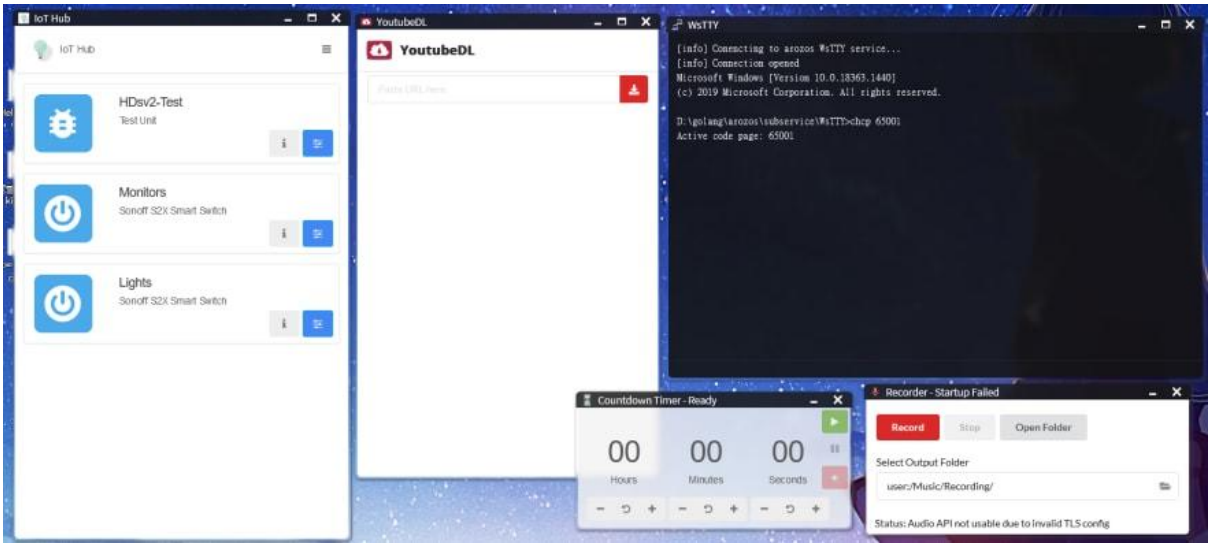
PDF Viewer


My ArOZ
April 11 12:47
PDF Viewer
1
of 54
Automatic Zoom
Arozos Cloud Desktop Operating System Documentation
Written by tobychui, 7-2-2021, for reference only
Start From Scratch
ArOZ Online 1.0
imuslab

NotepadA



Others



System Architecture

ArOZ Virtual File System

Introduction

ArOZ Virtual File System or VFS is a file system abstraction that is designed to add a layer of isolation between the host file system and the user file system that is accessible through the web interface. This file system will not emulate any file or folder. Instead,

encase the scope of a particular folder as virtual root and provide access to the folder's content using virtual path. This ensures the user cannot directly interact with the host Operating System and also reduces the complexity of the system for checking directory escape or other common concern on serving file systems over networks.

ArozOS VFS Functionality

The ArozOS VFS provides file access through virtual paths. Virtual paths in the VFS looks something like this

```
user:/Desktop/hello_world.txt
S1:/Video/test.mp4
Disk:/Files/foobar.md
```

The syntax of a VFS Path is denoted as follows.

```
{Storage ID}:/{Relative path to ID matching virtual root}
```

ArozOS VFS provides two main features in the Golang written module (and the core system) that are included in the user module.

```
func (u *User) VirtualPathToRealPath(vpath string) (string, error)
func (u *User) RealPathToVirtualPath(rpath string) (string, error)
```

Virtual path is an abstraction of file path location within the VFS. Real path is the relative path of that file to the ArozOS root or the host system root. The path returned is either absolute or relative depending on the storage.json configuration of the storage folder.

For example, a file located in "./files/Alice/Desktop/test.txt" can be accessed by Alice using her own account with the File Manager WebApp at "user:/Desktop/test.txt". Under this condition, the above two functions can help translate the path between realpath and virtualpath based on the developer's need.

Example Usage in Golang

Assume you have logged in as "User" and access the following function.

```
func handleGetDesktop(w http.ResponseWriter, r *http.Request){

userinfo, _ := userHandler.GetUserInfoFromRequest(w, r)
desktop, err := userinfo.RealPathToVirtualPath("user:/Desktop/")
log.Println(desktop)

//Should print out "./files/User/Desktop/"
}
```

Upload and Download

File Upload into the System

Problems of Uploading in low memory systems

When uploading to a system with very little RAM, for example 512MB of the Raspberry Pi Zero W, the /tmp folder fills up really quickly and leads to either the arozos process being killed or the whole system freezes and no longer accepts external requests and connections. Hence, an alternative solution for upload has to be developed and bypass the default pipeline that Golang handles uploads by default which is writing to the RAM before writing to disk.

Solutions

File uploading to the system mainly uses two endpoints written in the core of the ArozOS File System handler.

```
/system/file_system/upload
/system/file_system/lowmemUpload
```

In most cases, only one of them will be used in the same period of time through the File Manager webapp. In the current implementation of the File Manager upload protocol, the File Manager will enter low memory mode when the system memory in the Host OS is less than 1.8GB (Corrected for conversion of 2GB memories). However, the backend will not shut down the default upload handling endpoint if the system memory is less than 1.8GB. Uploading to the low memory endpoint is just the decision made in developing in the File Manager to provide a stable and robust file upload experience for the host with ultra low memory.

The standard upload interface uses FORM posts with file type input to handle file upload like a normal online form. It uses XHR API for uploading and real time upload progress can be observed on the user interface. This method will first buffer the file into memory (as well as page if it exceeds the system memory limit of upload, which can be configured using startup flags). This method is suitable for hosts with relatively more memory like a small linux server with 4GB of RAM or a Windows Server with 8GB of RAM or above.

The alternative to the standard upload interface is the low memory upload interface, which utilizes websocket to perform data upload. The process of the low memory upload interface behaves as follows.

1. The file selected is split into 512KB chunks
2. WebSocket connection to server is opened
3. One chunk is written into the socket, and wait for the host to finish writing to disk
4. Repeat step 3 until all the chunks write finished
5. Host merge the chunks into one file and move to the final destination on disk

This method of upload is significantly slower and computational intensive compared to the traditional upload mode. If possible, use the default upload API instead of the low memory upload mode unless strictly necessary.

File Upload using AGI File Write API

An alternative method to upload files into the system is using AGI FIle Write API for writing text based data into the ArozOS File System. Here is an example for creating and writing to a file with given" filepath" and "content" as variables.

```
 //Write to the file
var succ = filelib.writeFile(filepath, content);
if (!succ){
      error("Unable to save file");
      return
}else{
      sendResp("OK");
      return
}
```

This method is only suitable for small, text based files. Developers can also utlize this API to write base64 encoded binary files like images and sound files but it is not recommended.

File Download from the System

FIle downloading from the system all goes through the /media endpoint. To access a particular file with your account, access the following endpoint.

```
/media/?file=user:/Desktop/music.mp3
/media/?file=user:/Desktop/music.mp3&download=true
```

The download parameter tells the host server to serve the file with the "Content-Disposition" header and attach the filename of the target file with the responses.

Get Mime of FIle

To get the mime of the file, you can access the getMime endpoint listed below.

```
/media/getMime/
```

The required parameters are as same as the download endpoint and will return the Mime type of the file in plain/text format.

Downloading via Share Interface

The sharing interface allows public or other users to access a particular file or directory within your file system. The API for this access is located at /share/.

You can pass in the “download” parameter and “serve” parameter for the host to return the target file with a given type of response headers. Examples are as follows.

The following link will serve the file to the client with the “Content-Disposition” header which will set the download filename and pop up a download dialog in the browser.
http://localhost:8080/share?id=21b7cae0-4a5a-4f36-834e-54a02e3ede77&download=true

The following link will start serving the file in streaming mode. The browser can playback the file (if it is supported) or popup a download dialog if the browser do not have a suitable player for the shared file format.
http://localhost:8080/share?id=21b7cae0-4a5a-4f36-834e-54a02e3ede77&serve=true

Bandwidth Compression

ArozOS built in with gzip compression turned on. If you want to disable the compression, set the gzip flag to false.

Programming Interface

ArOZ Gateway Interface Programming

ArOZ JavaScript Gateway Interface, A(J)GI for short, is a programming interface based on the otto engine, allowing ECMAscript v5 to be used to interact with the ArozOS core functions written in Golang. The access methods are separated into two types. The following are the explanations of the access gateway.

Startup Loader

The ArozOS startup script with AGI will load the WebApp startup script from the WebApp root folder. The startup script name must be named “init.agi” that contains a module registration system call within the script in order for the module to be loaded into the ArozOS host successfully. Here is a minimum example for the init startup script extract from the Dummy module.

```
//Define the launchInfo for the module
var moduleLaunchInfo = {
   Name: "Dummy",
   Group: "Interface Module",
   IconPath: "Dummy/img/small_icon.png",
   Version: "0.1",
   StartDir: "Dummy/index.html"
}

//Register the module
```

```
registerModule(JSON.stringify(moduleLaunchInfo));
```

For more examples, see the WebApp programming section.

Internal Access

To access the AGI internally, your application can create an AJAX request to the AGI endpoint located at `"/system/ajgi/interface"`

Executing Gateway Script

To use the gateway, developers can call to the `ao_module_agirun` inside the ao_module.js wrapper. The definition of the function is as follows.

```
function ao_module_agirun(scriptpath, data, callback, failedcallback =
undefined, timeout=0)
```

Where the `scriptpath` is the script file location relative to the web root (aka ./web folder inside the src folder)

Here is an example request of the interface made from the FFmpeg Factory WebApp

```
function readStorage(key, callback = undefined){
    ao_module_agirun("FFmpeg Factory/backend/readConfig.js", {key: key},
callback)
}
```

Passing Parameter to Gateway Script

To pass parameters to the AGI script, you should create an JSON object and put it in the "data" field of the ao_module_agirun function call. Here is an example of sending parameters to the AGI script extracted from FFmpeg Factory WebApp.

```
function saveStorage(key, value, callback = undefined){
      ao_module_agirun("FFmpeg Factory/backend/writeConfig.js", {
            key: key,
            value: value
      }, callback)
}
```

This acts similar to your JavaScript variable definition in the AGI script. Before your AGI script loads, the Otto VM will inject the above two variables to the runtime environment using its key as the variable name. To access the injected variable in your AGI script, you can directly access the "key" and "value" parameters as follows.

```
if (newDBTableIfNotExists("FFmpeg Factory")){
```

```
    if (writeDBItem("FFmpeg Factory",USERNAME + "_" + key,value)){
      //Do something here
    }
}
```

External Access

To access the AGI externally from subservice domains, you need to create a GET request to the `"/api/ajgi/interface"` endpoint.

To authenticate against the ArozOS authentication agent, you also need to pass in a POST parameter with key: `"token"` that is set to the user access token with the user request. During a user request a page from the subservice domain, the request header will be injected with two new parameters

Header Key	Value	Usage
aouser	ArozOS Username	The user authentication username
aotoken	Token	The one time token for subservice to request AGI interface

The subservice should use the token embedded into the request header for accessing the AGI using external requests. Here is an example section of code for getting the username and token from the header in Golang

```
username := r.Header.Get("aouser")
token := r.Header.Get("aotoken")
```

To execute a AGI script remotely, create a FORM POST and request it to the interface as follows.

```
resp, err := http.PostForm(a.restfulEndpoint,
       url.Values{"token":{token}, "script":{script_content}})
      if err != nil {
       // handle error
       return nil, err
    }
```

To process the returned results from AGI, you can read the response body just like retrieving other FORM results as follows.

```
bodyBytes, err := ioutil.ReadAll(resp.Body)
	if err != nil {
		log.Println(err)
		w.Write([]byte(err.Error()))
		return
	}
	resp.Body.Close()

//Do something with bodyBytes
```

For a full example, please see the demo subservice repository.

WebApp Programming

ArozOS used standard HTML5 WebApp structures. Developers no need to be forced to use any programming tools / nodejs library when developing the ArozOS WebApps.

A WebApp is on ArozOS system has the following properties

1. Its root folder is completely inside the./web folder.
2. It use only HTML5, JavaScript (and its framework) and CSS, with backend powered by AGI system written in ECMAscript
3. Included the ao_module wrapper in the head section of the starting HTML file

WebApp Startup Modes

An ArozOS WebApp can be started up in 3 different modes, sometime 4 modes (depending on developer support)

1. Native Mode (Opening the WebApp endpoint directly in a browser window)
2. Float Window Mode (fw mode, opening WebApp on Web Desktop Environment window)
3. Embedded Mode (Opening WebApp on Web Desktop Environment with a file descriptor pointer)
4. (Optional) PWA mode, Android Progressive WebApp support with Google framework

WebApp Registration Script (init.agi)

The init.agi script is used to define the startup parameter of the current WebApp. It must be located at the root path of the module itself. The module register script can be used to define the following module properties:

- WebApp name
- WebApp startup mode (Native, Float Window and Embedded)
- WebApp icon
- WebApp default window size
- WebApp type and meta information
- WebApp supporting file extensions

The definition of init.agi supports a JSON object to be imported for handling the module registration. In the source code, you can find such section of code that define the key value of such JSON object structure

```
type ModuleInfo struct {
    Name          string   //Name of this module. e.g. "Audio"
    Desc          string   //Description for this module
    Group         string   //Group of the module, e.g. "system" /
"media" etc
    IconPath      string   //Module icon image path e.g.
"Audio/img/function_icon.png"
    Version       string   //Version of the module. Format:
[0-9]*.[0-9][0-9].[0-9]
    StartDir      string   //Default starting dir, e.g.
"Audio/index.html"
    SupportFW     bool  //Support floatWindow. If yes, floatWindow dir
will be loaded
    LaunchFWDir  string   //This link will be launched instead of
'StartDir' if fw mode
    SupportEmb   bool   //Support embedded mode
    LaunchEmb     string   //This link will be launched instead of
StartDir / Fw if a file is opened with this module
    InitFWSize   []int  //Floatwindow init size. [0] => Width, [1] =>
Height
    InitEmbSize  []int  //Embedded mode init size. [0] => Width, [1] =>
Height
    SupportedExt []string //Supported File Extensions. e.g. ".mp3",
".flac", ".wav"
}
```

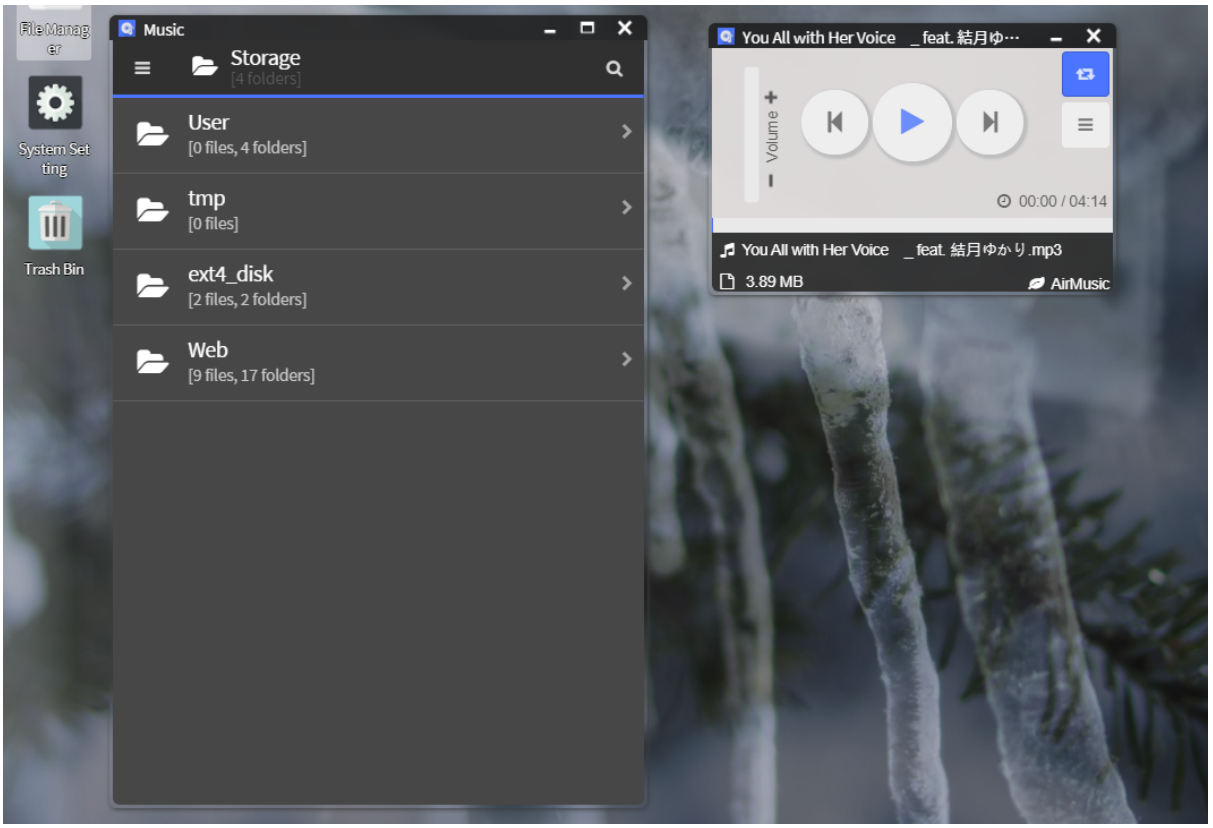
Here is an example extracted from the Music module for how to register a module in init.agi script.

```
var moduleLaunchInfo = {
    Name: "Music",
    Desc: "The best music player in ArOZ Online",
    Group: "Media",
    IconPath: "Music/img/module_icon.png",
    Version: "0.1.0",
    StartDir: "Music/index.html",
    SupportFW: true,
    LaunchFWDir: "Music/index.html",
    SupportEmb: true,
```

```
    LaunchEmb: "Music/embedded.html",
    InitFWSize: [475, 700],
    InitEmbSize: [360, 240],
    SupportedExt: [".mp3",".flac",".wav",".ogg",".aac",".webm",".mp4"]
}


//Register the module
registerModule(JSON.stringify(moduleLaunchInfo));
```

The above example will register one module named “Music” with two mode: FloatWindow mode (defined by “SupportFW” parameter) and Embedded Mode (defined by “SupportEmb” parameter). The results of the size definition are shown in the screenshot below.



(Left: FloatWindow mode, Right: Embedded Mode)

Note that an invalid init.agi startup script file or in the case where the startup file doesn’t exist, the module will not be loaded and will be ignored during the startup process. However, its content located in the web root folder will still be accessible using direct URL.

User Interface Programming

The interface programming for ArozOS WebApps is standard HTML5 with JS and CSS. A basic WebApp can be a simple HTML5 Website that does not need to interact with the system functions. If specific functions are required like changing Float Window title,

setting resizable or opening a new Float Window within the WebApp, see ao_module.js system wrapper for more information.

Receiving File Descriptor Pointer

The ArozOS system sends file descriptors using URL hash. Here is an example of the hash which one file being sent to another WebApp module using the file descriptor pointer.

```
#%7B%22filename%22%3A%22test.txt%22%2C%22filepath%22%3A%22user%3A%2FDesk
top%2Ftest.txt%22%7D
```

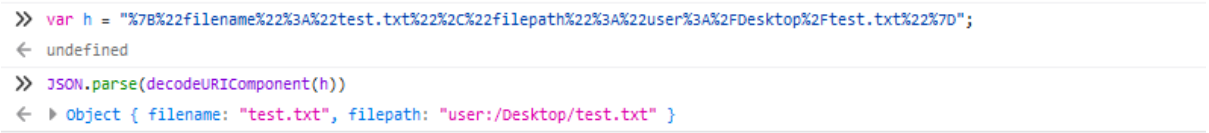
There are two ways of receiving files sent by the ArozOS system services including the Desktop and File Manager. Manual parsing or calling to the ao_module_wrapper.

Manual Parsing of File Descriptor Pointer

To access the original value of the hash, you can perform the following conversion in JavaScript (Front end)

```
JSON.parse(decodeURIComponent(hash))
```

For the example above, you will see the following output.



You can also generate the same hash using the reverse operation as the decode option. For example

```
var a = {
 filename: "test.txt",
 filepath: "user:/Desktop/test.txt"
}

var hash = encodeURIComponent(JSON.stringify(a));
console.log(hash);
```

Will return the hash value of the file descriptor object

```
"%7B%22filename%22%3A%22test.txt%22%2C%22filepath%22%3A%22user%3A%2FDesk
top%2Ftest.txt%22%7D"
```

The hash generated then can be used in calling the new floatWindow API for opening in system components like File Manager.

ao_module Wrapper Parser

The ao_module wrapper also provides a convenience API for parsing the file input. You can call the following functions for returning if there are input files. The returned value is a JavaScript array of file descriptor pointers.

```
var files = ao_module_loadInputFiles();
```

*** The function will return null if there is no input files* **

If your module can only handle one file at most, you can simply take the first file from the returned value.

Here is a short example extracted from the Video module.

```
var playbackFile = ao_module_loadInputFiles();
//Only handle one file
playbackFile = playbackFile[0];
if (playbackFile == null){
   return
}
```

Creating Backend Calls with AGI script

A WebApp on ArozOS can create backend operations through the ArOZ JavaScript Gateway Interface (AGI). For interface structure related documentation, see "ArOZ Gateway Interface Programming" section.

To create a backend request using AGI, call the agi_run function in ao_module as follows.

```
 ao_module_agirun(scriptpath, data, callback, failedcallback =
undefined, timeout=0)
```

The parameter required are as follow

Item Name	Usage	Value Example
scriptpath	The agi / js script location you want to run from the web root	`"demo/backend/hello_world.js"`
data	The data you want to send with the request	`{"name" : "FooBar"}`
callback	The callback when the process is finished on the server side	`function(data){ console.log(data);`

		}

failedcallback	The callback when the request is failed	function(){
		    alert("Oops");
		}

timeout	The timeout for the AJAX request	3000

The above example with the following AGI script will result as follows.

(Content of ./web/demo/backend/hello_world.js)

```
sendResp("Hello World to " + name );
```

(Console Output)

```
"Hello World to FooBar"
```

See Appendix “Aroz Javascript Gateway Interface API” and “Aroz JavaScript Gateway Interface Script Example” for more information.

Calling Other System Services / Functions

If you need to interact with the system / Web Desktop framework, see ao_module.js for all the possible system call functions.

IME Programming

IME (Input Method Editor) is a special type of WebApp that runs on ArozOS that behaves like a native Input Method Editor. IME is a common software installed on computers that its users do not use English as their primary language. IME programming cannot be done using ao_module and it has to be done via direct interaction to the desktop.system.

To register the ime keydown handler, check and set the desktop windows’ window.ime properties as follows.

```
if (ao_module_virtualDesktop){
      if (!parent.window.ime){
            alert("This version of ArozOS does not support IME function!")
      }else{
            parent.window.ime.handler = handleKeydownInput;
      }
}
```

And for the handleKeydownInput function, it is just a generic function that handles keydown events.

```
function handleKeydownInput(e) {
    //handle e.keyCode here
}
```

To send out the processed text, you can send it to the target pointed by the window.ime.focus object. Here is an example of sending text to the focused DOM element.

```
var text = "你";
if (parent.window.ime && parent.window.ime.focus != null){
      insertAtCaret(parent.window.ime.focus, text);
}
```

Where the insertAtCaret function is a generic function for inserting text to an DOM element (e.g. textarea). Here is an example implementation of such a function.

```
function insertAtCaret(target, text) {
    var txtarea = target;
      if (txtarea == undefined){
            return
      }
    var scrollPos = txtarea.scrollTop;
    var caretPos = txtarea.selectionStart;

    var front = (txtarea.value).substring(0, caretPos);
    var back = (txtarea.value).substring(txtarea.selectionEnd,
txtarea.value.length);
    txtarea.value = front + text + back;
    caretPos = caretPos + text.length;
    txtarea.selectionStart = caretPos;
    txtarea.selectionEnd = caretPos;
    txtarea.focus();
    txtarea.scrollTop = scrollPos;
}
```

SubService Programming

ao_module.js Function Wrapper

Ao_module.js is the main front end function wrapper for all the ArozOS Web Apps. Functions include adjustment of floatWindow size, editing of floatWindow properties and interaction with ArozOS custom events handler like the IME and more. These functions can be called within a module after you have included the function wrapper. The function wrapper can usually be found under web/script/ao_module.js.

Example Usage

The following example is extracted from the Dummy Module, whose parent directory is located at the same root with the script folder.

```
<script src="../script/jquery.min.js"></script>
<script src="../script/ao_module.js"></script>
```

Be noted that the ao_module wrapper must be included within the current HTML script using relative path. Absolute path will not work due to some implementation within the ao_module that requires the relative location of the script to the current script to be known.

To execute ao_module functions, simply call the function name. Most ao_module functions come with a “prefix ao_module”. Here are some examples.

```
var flist = ao_module_loadInputFiles();

ao_module_openFileSelector(fileLoader, "user:/Desktop/",type="new",true,
{
        defaultName: "New File.txt"
 })
```

Not all ao_module functions are included in the global scope. For example, some utilities functions are included with their own objects. Here are a few examples of these functions.

```
var fileinfo = ao_module_utils.getDropFileInfo(dropEvent);

ao_module_utils.getRandomUID();
```

See the ao_module function list appendix for a full list of ao_module functions usable in your webapp.

Function Override

It is possible for the user to overwrite some of the ao_module functions in order to implement custom features from the desktop and web app interaction. One of the most commonly used overrides is overriding the ao_module_close() api with a custom one to provide a save check. Here is an example extracted from the Notebook module for showcasing the override implementation.

```
//Overwrite the close sequence
function ao_module_close(){
    if (!isSaved()){
        //Not saved
        if (confirm("Some changes are not saved. Save before exit?")){
            saveText(function(){
                //Exit after save
                closeThisWindow();
            });
        }else{
            //User request not to save
            closeThisWindow();
        }
    }else{
        //Saved. Exit
        closeThisWindow();
    }
}

function closeThisWindow(){
    if (!ao_module_virtualDesktop){
        window.close('','_parent','');
        window.location.href = ao_root +
"SystemAO/closeTabInsturction.html";
        return;
    }
    parent.closeFwProcess(ao_module_windowID);
}
```

You can notice that the override function is directly called to the desktop internal function. This is necessary in this case as function overriding must directly interact with the desktop.system internal function and this should be the only case when developing an ArozOS WebApp that directly calls to the internal funcstion of the desktop interface. Please be noted that the mobile interface contains only a subset of the desktop internal function. Hence, not all desktop internal functions can be used in the mobile interface.

Scope of Application

The ao_module script is designed to be used inside an iframe under virtual desktop mode for desktop or mobile interface. If you try to call it under standard viewing mode (as a normal webpage), most of its functions will not behave as expected to deal with the dependencies of the wrapper to the desktop interface script (desktop.system and mobile.system). This rule also applies to multiple layers of iframes within your web apps module. If your mobile cannot avoid using an iframe, and ao_module is expected to work with an iframe , please consider directly implementing the function in the top most script of your webapp and access it with

```
parent.{function name}()
```

instead of trying to use ao_module within the iframe nor directly interfacing with the desktop.system or mobile.system program sope.

Compatibility and Updates

Using ao_module as the main programming interface will guarantee system compatibility in the future 5 versions of the ArozOS system. Deprecated API will start to log out a deprecated message if the API will be removed 5 or more versions ahead.

DO NOT TRY TO DIRECT INTERACT WITH THE FUNCTIONS IN DESKTOP.SYSTEM AND MOBILE.SYSTEM. ANY COMPATIBILITY ISSUE USING THIS INTERACTION METHOD WILL NOT BE ENTERTAINED THROUGH ISSUES OR SUPPORT.

If you have any specific requirement regarding the API of ao_module, please directly email us for your requested enhancement.

Application Categories

Introduction

WebApps and subservice within ArozOS are categorized according to their type of usage. However, there are a few special categories that do not lie within the standard categories. Here is a list of such categories that the system will treat differently.

Interface Modules

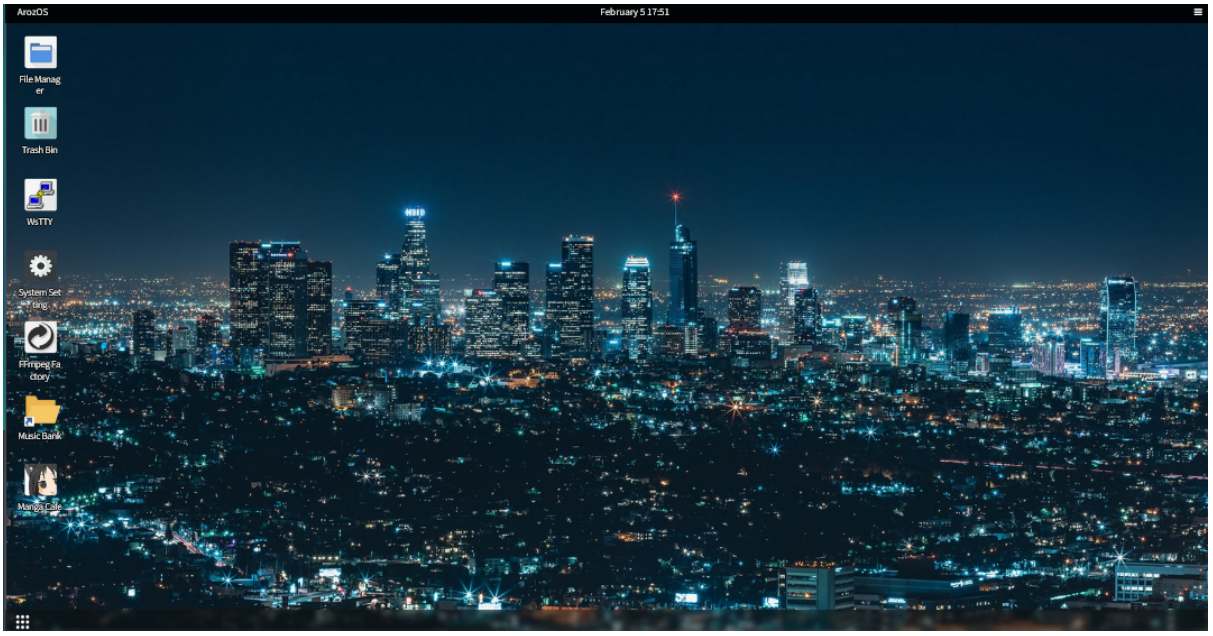
Interface module is a special class of module that can be acted as the first default module to handle user access and also act as something like “Kiosk” mode of Chromium or Firefox.

An interface module can also support other opening methods including floatWindow and embedded mode. Here are two example of Interface Modules

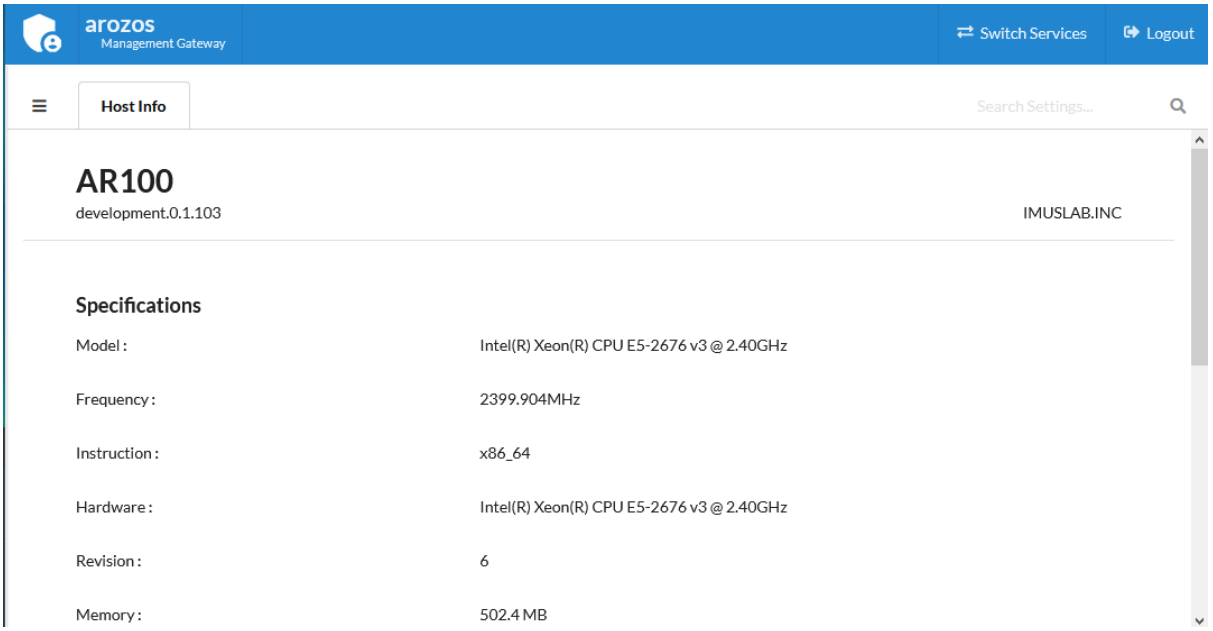
Desktop Version: 0.1.111	Interface Module	N/A	✗	✗	✗
Dummy Version: 0.1	Interface Module	N/A	✓	✗	✗

Each user permission group will have 1 default interface module. By default, Desktop is the default interface module for the administrator group. Once a user is set to use a given interface module, the user will be directed to that interface module once logged in. Making that interface module “full screen”.

Desktop as interface module → User login and will be redirected to Desktop

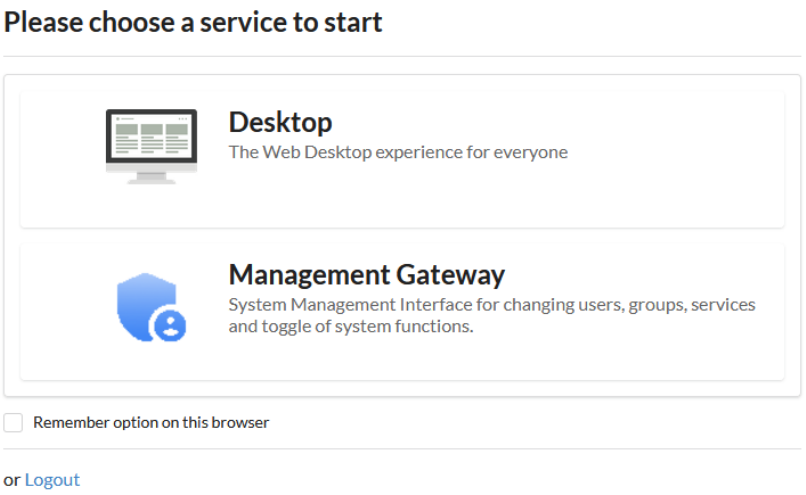


Management Gateway as interface module → User login and will be redirected to Management Gateway WebApp

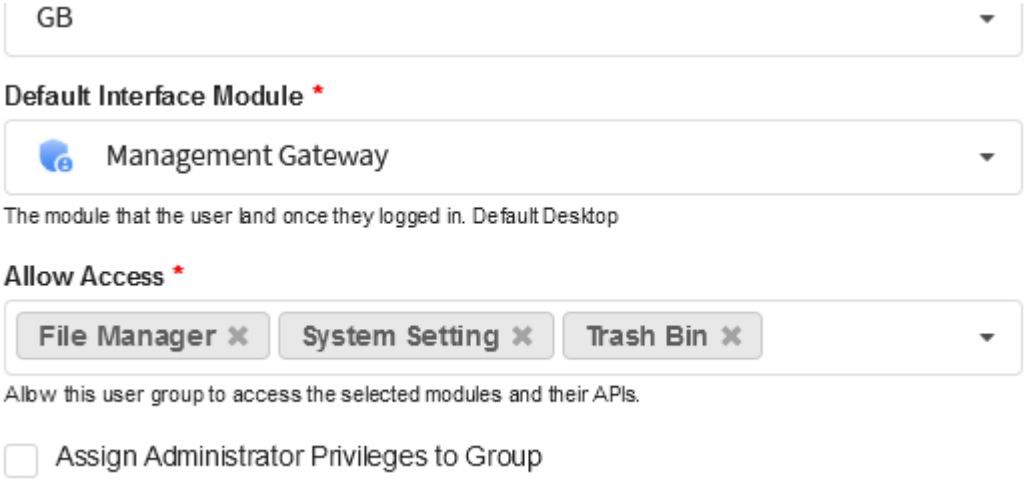


For users that are inside two permission groups with different interface modules, a selection interface will popup and request for choosing the target interface. Here is an example of a user that is inside two permission group that is setup as follow

Permission Group Name	Interface Module	Is Admin
Default	Desktop	false
Manager	Management Gateway	false



To update the interface module setting, go to Permission Group → Edit → Default Interface Module and select the target module from the dropdown list.



IME (Input Method Editor)

IME is another type of special category that they are specialized for handling inputs and modify the output of keystrokes. This feature is added in v1.113 for experimental purposes.

When IME startup, it will hook its input handler to the window.ime object on the desktop.system. This will allow ao_module included webapps to handle input using the handling pipeline and send keydown events to the specific input methods.

Here is an example of the input method Cyinput that uses the pipeline to input Cantonese into the Notebook webapp.

	
IME can also send output strings to desktop input or textarea	IME behaves similarly on shortcut and floatWindows events.

To see how to bind events to support IME, see IME (Programming) section.

ArozOS Culture

Mascot

Original ArOZ Mascot

The ArozOS Mascot is a character reused from the legacy ArOZ Online Beta system which also reused one from the old ArOZ Project from ArOZ Online Omega. The designer of this character is unknown and through remix of the original character's facial expression and clothes, the first error message was designed to provide interesting feedback to user when they see errors in the ArozOS system.

Error Message Icons

Here is a list of in-use error message icon of the ArozOS

	Already logged in warning This error occurred when a user already logged in wanted to login with an auto login token.
	Generic Error Message This error image will shown when there is a generic error situation.

	No Interface Module This error occurred when a user's interface module setting was corrupted and cannot be accessed.
	200 OK
	200 OK Alternative This message will show when the request is processed without error but the user is not supposed to see this page. This page sometimes also shows as invalid configuration's default information page.

ArozOS Mascot

Since ArozOS v1.110 release, a new version of the mascot has been designed and allows a much more grown up version of the fox girl to be displayed throughout the later developed modules and documentation.

The following image shows the full version of the illustration in postcard format, which is designed to celebrate the official release of the first ArozOS stable version.

ArozOS v1.0

https://arozos.com

END OF DOCUMENTATION