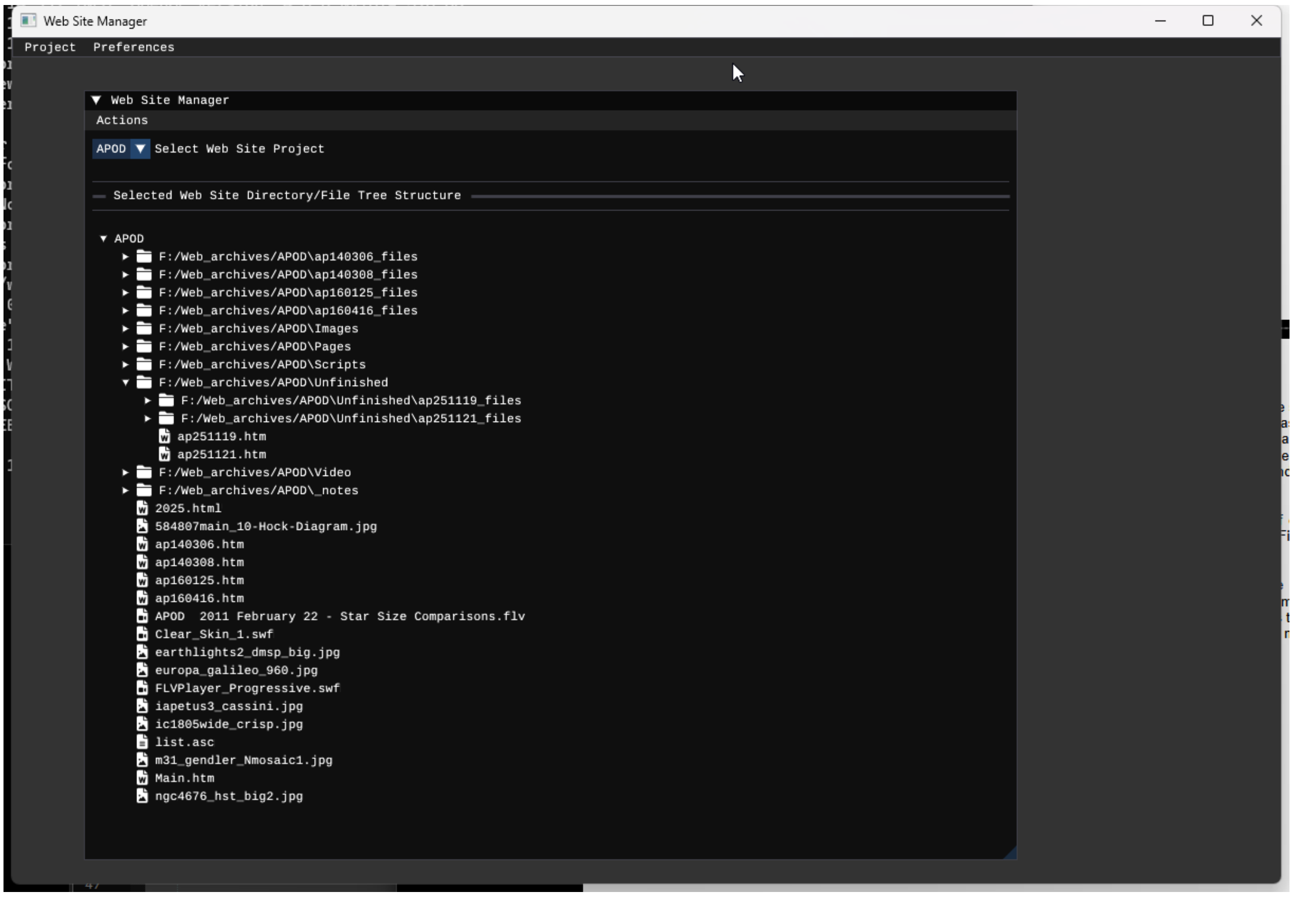
Web Site Manager (WSM) Design and Source Code Notes

Introduction

Web Site Manager (WSM) is an application that is designed specifically to manage the files that are stored in a web site project directory. A web site directory structure will have web page related files such as image, video, sound, html page and css files in a structured form of specific directories. Web site manager is designed not only to manage the simple tasks of any file manager software such as move, delete and copying of files, but it also performs the tasks of tracking and updating the html links and source dependencies that exist with the html web page files of the project.

The Web Site Manager application is not a conventional file manager with two or more lists of directory contents in a panel, but is a file manager that displays the file structure of a web project as a tree. (Fig 01)



```
— Selected Web Site Directory/File Tree Structure

▼ APOD
    ▶ F:/Web_archives/APOD\ap140306_files
    ▶ F:/Web_archives/APOD\ap140308_files
    ▶ F:/Web_archives/APOD\ap160125_files
    ▶ F:/Web_archives/APOD\ap160416_files
    ▶ F:/Web_archives/APOD\Images
    ▶ F:/Web_archives/APOD\Pages
    ▶ F:/Web_archives/APOD\Scripts
    ▼ F:/Web_archives/APOD\Unfinished
        ▶ F:/Web_archives/APOD\Unfinished\ap251119_files
        ▶ F:/Web_archives/APOD\Unfinished\ap251121_files
          ap251119.htm
          ap251121.htm
    ▶ F:/Web_archives/APOD\Video
    ▶ F:/Web_archives/APOD\_notes
      2025.html
      584807main_10-Hock-Diagram.jpg
      ap140306.htm
      ap140308.htm
      ap160125.htm
      ap160416.htm
      APOD  2011 February 22 - Star Size Comparisons.flv
      Clear_Skin_1.swf
      earthlights2_dmsp_big.jpg
      europa_galileo_960.jpg
      FLVPlayer_Progressive.swf
      iapetus3_cassini.jpg
      ic1805wide_crisp.jpg
      list.asc
      m31_gendler_Nmosaic1.jpg
      Main.htm
      ngc4676_hst_big2.jpg
```

Fig 01

Inspiration for creating web site manager came from the Dreamweaver web site design application which has these capabilities making managing and updating html links far easier that otherwise would be the case of manualy tracking and editing such hyperlinks, However, Adobe asks too high a price and has all its applications based upon a monthly subscription such that for a part time user of such software it is not economical to use the Dreamweaver application. A personal need was required to edit and manage html pages and their linked files downloaded and archived from the internet into a custom directory structure such that these links are no broken when doing so. The feature of Dreamweaver to move, rename and edit files that keep track of and update html links could not be found in any other application that deals with web site management,.

Thus it was decided to try and emulate as best that could be done to an acceptable level this ability to manage and update html hyperlinks.

Once the thought process was put to task about how to approach such a problem and then sitting down to work and test a few things through code, it only took only a few hours to figure out the basics of how to do this from scratch. The basics and design of how to do this left an impression of amazement of why no other software that edits and manages html web pages and files has not this featureor has not been developed.

Described below is the design and coding of the web site manager tool to perform such a task.

The Web Site Manager Application

WSM can in no way be considered as a complete application. This is because It does not perform all of the tasks the a web site manager and design application is expected to perform such as the Adobe Dreamweaver or other web site application software. WSM is more of a local personal web site file manager application that has at its core the task to manage and update the html hyperlinks of html files when the files that are linked to the html files are moved, renamed, or deleted, or the html files themselves are edited.

Principle Concept of Design

The basic design concept of WSM is very similar to that utilised in Dreamweaver. The user selects a web site project that exists, and which is in essence just the root directory of a file structure of a web site. The file structure of this web site is then read and loaded to be displayed as a tree file structure as illustrated in **Fig 01**. While loading this web site, a series of functions determines which text files (ie html files) have links defined within them and then reads those files, finds the defined links and stores those links into a data structure to define which files have links from a html file, and which files a html file links to.

Once this data structure of links is defined and stored in computer memory, then it is a matter for the application code to query this data structure whenever a html linked file, or the html file itself is moved, renamed, or modified so as to update the html link definition within the html file that is associated with that change.

Thus the WSM consists of two basic principle parts.

1: A GUI of the web site file structure displayed as a tree structure which the user selects the files to edit or perform file manager tasks.
2: A data structure associated with the linked files that stores the links associated with that file and updates the links of the html file if/when the file has its status changed.

Application Coding

To achieve the application goals, the WSM coding is in C++ 20 standard utilising the GLFW windows and the ImGui tools to define the application interface and display. It was decided not to utilise GLFW as a C++ library and to directly compile GLFW from source since the overhead of such is not that great and solves a potential problem encountered of library incompatibility due to choice of C++ standard and compiler to generate the GLFW library.

A developed coding framework is used as a basis for the application operation and implementation.

Additional dependency code for glm and tinyFileDialog are also included as part of the application for data structure and file selection.

Application Directory Structure

The directory structure of the Virtual Worlds project as currently defined in August 2025 consists as follows.

```
Bin              (compilation binary and run time application data files)
Documentation    (Application documentation )
FrameWork        (Application Framework code that is designed to be shared between  applications)
Source           (Application  Source code)
thirdparty       (Source code of dependent third party support applications)

ws_manager.sln
ws_manager.vcxproj
ws_manager.vcxproj.filters
ws_manager.vcxproj.user
```

Listing 01 : Virtual Worlds root directory structure

Sub directory structure

```
Bin → Intermediate        (compiler intermediate code)
      x64 →Debug          (Application debug executable and data files)
           Release        (Application release executable and data files)

FrameWork → AFW           (Application framework code that can be used in any application)
            GUI           (Framework code for basis of an ImGui tree )
            OGLFW         (Framework code that can be used for general OpenGL graphics applications)


Source →Application       (WSM application source files)
        File_Manager      (Tree and html file manager source files)
        Tree_Framework    (WSM Tree base and link data structure source files)
        main.cpp          (C++ WSM application entry file)

thirdparty →  glew
              glfw
              glm-1.0.1
              ImGui
              tinyFileDialog
```

Listing 02 : Virtual Worlds sub directory structure

Dependant third party version of software is given in the third party source documentation.
For all intents and purposes, the third party directory can be considered as part of the applications framework and may be moved at some later stage to the FrameWork directory.

Code conventions

The C++ code conventions currently adopted is to have all the C++ code, unless it cannot be avoided to exist in .h header files only. This is to make it easier and quicker to edit and maintain a single file rather than having to set up separate header files and then having to create an associated ..cpp source file. This allows it to be easier to read and track such things as variables, and virtual functions. Often when reading and tracking code written with separate header and source files, it was found that one needs two files open to determine such things like a class variable that is used within a class or even whether a certain function is inherited or not.

To make the code as readable as possible, abbreviations or condensing of class, variable, and function names are not performed widely, though is sometimes used. The naming convention is such that there is no upper case variables used and each word of a name is separated by an underscore to make reading easier and more natural. User defined macro variable names follow this rule but have all characters in upper case so as to distinguish them from a normal variable or class.

To make it as plain and as descriptive as possible for the user reading the code to understand the purpose or functionality of the code, the naming of all variables, data structure types , classes, functions etc is such as to describe as precise as possible what that variable, data structure, class, function represents or functionality is. By doing so then allows the user reading the code to better understand and interpret correctly the code written.

So as to make it easier to define when reading the C++ code what user created data type a certain variable name refers to, a convention is adopted as follows

All class types have the word class at the end.
All data structure types have the words struct_type at the end.
All enumeration types has the word enum at the end.

Visual Studio Code Compilation

The visual studio C++ code was compiled as ISO C++20 Standard using the Visual studio 2022(v143) toolset and Windows SDK 10.0.22621.0 or the Visual studio 2026(v145)

Advised not to use earlier then C++20 Standard to maintain compatibility.

Code Documentation

Documentation and explanation of source code exists within the source code itself as much as possible and naming of variables and functions, classes etc is kept to a standard of readability as much as possible. However, this is often not enough to explain the many nuances and work runarounds that have been needed to be implemented to get things to work as intended and to comply with compiler demands.

Thus over time as code is finalised, documentation of code may be performed in a more formal manner of being documented in a pdf file to explain and visualise what the function of the code is for and does. This documentation is not just for any member of the public, but is also for to remind the author of what was coded and the relationships and links between all the classes, functions, structures etc. that may have been forgotten over a long period of time.

Coding Design Principles

Global Variables (Singleton Classes)

A form of a global function that was common and needed to be accessed or referenced across the entire application was required for the application logger class. The solution found was to create a singleton class that is visible and accessible through out the entire application. This singleton class defines a global logger class function to display and log error and other messages to a terminal and/or to a log file to inform the user or software developer of the operation of the application..

One such singleton class exist and must be defined as an included file within the main application .cpp file before all other include file declarations that may use them, otherwise the global variable data that exists within them are not set correctly, are cannot be accessed.

The singleton class called afw_globalc (application framework) is very widely used to convey such logger messages via a defined logger class called afw_Logger_class.

For more information on how singleton class are defined and how they work, an online search will give adequate information on this topic better than can be described here.

01 : Application operation

Application base Class afw_app_base_class

The execution and running of the application is implemented and managed using an abstract application base class afw_app_base_class.

afw_app_base_class is designed to define and handle the execution and function of any user application that incorporates as part of it function a user input and interaction of use of a keyboard and mouse. Within this class is a virtual function called setup is called to define the application setup routines and code and then a another virtual function run_loop is called that defines the application run time execution loop. Once the execution run loop is exited due to a user interaction, a final application virtual close function is called, from which any clean up operations are to be performed.

afw_app_base_class could also act as a base class from which multiple applications could be defined and run from a single application entry point.

The afw_app_base_class is inherited by the main application class from which the developer specifies the application code for each of the virtual functions to handle mouse, keyboard interactions and the application startup, run loop and close functions.

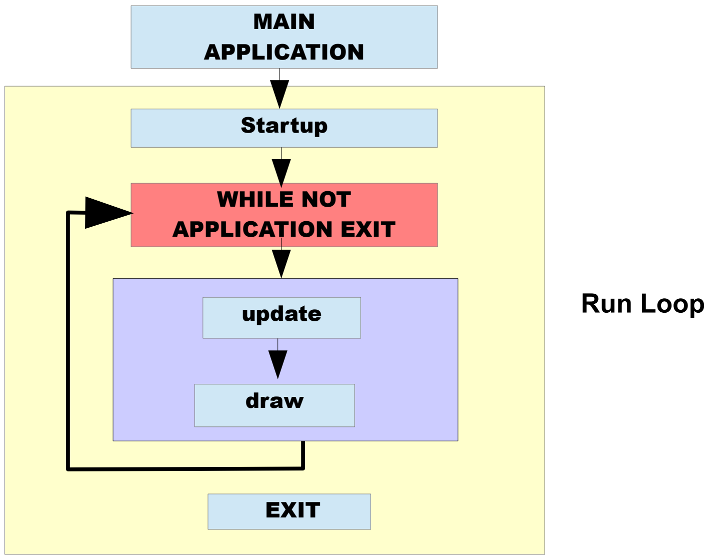


Fig 01.01 Schematic of Application base Class afw_app_base_class application run operation

The application main() function entry point can be as simple as

```
int main() {
   app_main_class* app = new app_main_class;
   app->run();
   exit(0);
}
```

02 : Loop Cycle

The WSM application tasks to be performed per loop cycle as illustrated in **Fig 01.01** can be summarised for each loop cycle function update, draw and exit in table 02.01 as

startup()

ImGUI Initialisation
WSM initialisation

update()

draw()

ImGui render setup
Display WSM interactive GUI
ImGui render exit

exit()

table 02.01

03 : WSM File Tree Manager

03:01 The WSM file tree structure

The WSM application centres around the concept of a computer file structure being displayed as a tree as illustrated in **Fig 01**. ImGui has a functional ability to display this file tree structure, interact and perform tasks on it, and thus is used to do so. Such a tree implementation consists of two basic attributes that make up this tree structure. A branch which is a root directory and alll of its subdirectories, and a leaf which is a file of any type. This concept should be familiar to any programmer.By use of this definition for tree branches and leaves, a WSM tree file manager can be defined and implemented to perform all the tasks that a file manager is required to do. Upon performing these file manager tasks, an additional task to update the links within a html file can be also performed.

Such a tree structure to perform the core file manager tasks is defined in a C++ class ws_tree_class. It can be observed that such a file tree structure is fractal in nature and any branch of the WSM in isolation can be considered on its own as being a whole tree in itself. This leads to the ability to use recursion to perform the tasks of creating and managing the WSM file structure as it also reflects the real data structure of the computer file system itself.

With the establishment that each tree branch can be considered as a file system directory that is a fractal component of the whole, then it can be established that each tree branch is of the same data type structure as the whole tree itself and can be used in recursive algorithms, then each tree branch is of the data type ws_tree_class.

Considering that each leaf of the tree structure is a file of any type, then the leaves in code must be of a specific data type structure or class that can define any and all computer file types. However, the file data type of any leaf may be different and is defined by its file extension. The type of file that the file falls into within a common category type can be defined. By segregating the file data types into categories rather than individual data types can make managing them easier and more efficient.

This can be done for the tree branches to be classed as a directory category and for each tree structure type of branch and leaf to come under a single category as being referred to as a tree node. Each tree node has a classification or category indicating what type of tree node it is. Since the WSM tree structure is based solely upon the file structure of computer storage memory, such a tree node category classification is defined in WSM as an enumeration data type afw_tree_node_item_type_enum.

```
enum class afw_tree_node_item_type_enum {root_tree_node,tree_node, tree_leaf,undefined};
```

root_tree_node	: indicates the tree node that is the top most or root directory of the website file structure
tree_node	: indicates a tree node that is a directory of the website file structure.
tree_leaf	: indicates a tree node that is a data file of the website file structure.
undefined	: indicates a tree node that has not be assigned

Each tree node that is a tree leaf thus needs to be then defined into a category type to designate what type of file it is as discussed above. Such tree node leaf category classification is defined in WSM as an enumeration data type afw_file_item_category_type_enum.

```
enum  class afw_file_item_category_type_enum { directory, page, pageformat, image, script,media, audio,
                                                pdf, code, undefined};
```

To display an ImGui tree node of any category, a requirement of a unique id for the tree node is required for ImGui to process its GUI functionality of highlighting and selection of tree nodes or other desired functions. A label to display the information of what the tree node represents is required, as well as a flag to indicate if the tree node is selected or not. By applying some clever text to the label, an icon can be displayed as part of the label.

Thus a C++ data structure for a tree node to define a tree node can be defined to be used to construct an ImGui tree incorporating the above discussed elements as given for the C++ structure afw_tree_node_item_base_struct_type in **Listing 03:01**.

```
struct afw_tree_node_item_base_struct_type {
    afw_tree_node_item_type_enumtree_node_item_type = afw_tree_node_item_type_enum::undefined;

    std::string label   = " Item label";
    std::string icon_id = "";

   id_type tree_node_id = INVALID_ID;

   bool selected = false;
```

};

Listing 03:01

This structure forms the basis of all ImGui tree node elements and can be found in the C++ file

FrameWork→GUI→afw_tree_node.h

03:01 WSM html linking

Any ImGui tree can be constructed using the structure type afw_tree_node_item_base_struct_type, but for the purposes of the WSM, to store, keep track of and update the html links to files and the hyperlinks within the html files, an additional data structure is required to store and process the link data that each tree node leaf may or may not have associated with it.

The html links that any files may have can be considered as links to that file from a html file, and links from a html file to another file. Thus any html files can have links to it from one or more other html file hyperlinks, and links from that html file to one or more another html files. A html files may even have hyperlinks to locations within itself. This can be illustrated in **Fig 03.01**.

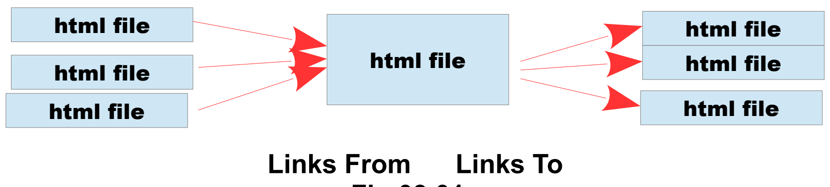


Fig 03.01

Similarly, any html file can have a link to a source file that is used to display images, video or other types of data, but never can anything other than another html or web page hyperlink link to it. **Fig 03.02.**

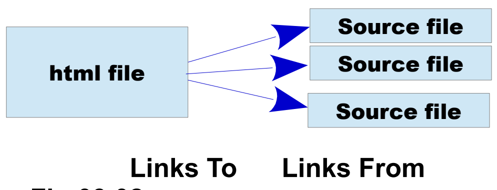


Fig 03.02

Any non html or web page file that is a source file to any html file can have one or more html files linked to it. Such files are as mentioned above as image, video or audio files that can only have a link to it, and not link from that source file.**Fig 03.03.**

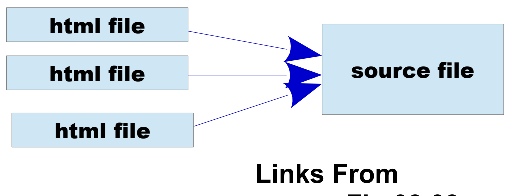


Fig 03.03

A final consideration of the links as illustrated in **Fig 03.01**, **03.02** is that some of the links from a html file may be links to an external html or source file outside the bounds of the web site root parent directory that defines the web site file structure. If such links exist, they need to be resolved by incorporating them into the project and to be updated, either manually or by use of the WSM application, or they can be ignored with the intention of deleting these links which will be designated as broken links.

Each file and directory that the WSM file tree represents has a path name associated with it, and it is also a path name to a file that a html has as its designation of files to link to as either a hyperlink to open another file, or to use that linked file to be incorporated as part of its functioning or display. Each link to and from a file as illustrated in **Fig 03.01 to Fig 03.02** is a link to a file that has a path name to that file.

Therefore the data type to specify a link that a tree leaf or file links to, or has a link from a html file is one that gives best the path name to a file, which in the case of C++ can be the C++ std::filesystem::path data type. Thus any WSM tree node data structure needs to contain a std::filesystem::path data type specifying the path name to that file. In addition, a value giving the file category needs to be present so as to indicate what type of file the WSM tree node represents. This data type incorporates and has added to it the afw_tree_node_item_base_struct_type defined in **Listing 03:01.** This structure can be considered as a treee node item describing the file that the tree node represents. To complete the representation and description of a WSM tree node item, the C++ data structure afw_tree_node_item_base_struct_type is inherited by this datatype and is defined as ws_tree_node_item_struct_type given in **Listing 03:02**.

```
struct ws_tree_node_item_struct_type : public afw_tree_node_item_base_struct_type {
	std::filesystem::path item_path;
	std::filesystem::path previous_item_path = "";

	ws_tree_class *node_item_tree_branch = nullptr;

	afw_file_item_category_type_enum afw_file_item_category_type;

	int tree_item_type;
};
```

Listing 03:02

To update a link, a storage of the old or previous link value before any update needs to be present and is the reason that a previous_item_path storage variable is present.

With consideration of how a web page html file has links to and from it, a data structure of links can be defined that all WSM tree node leaves need to have. Given that **Fig 03.01 to Fig 03.02** indicates that such a relationship of links is a many to one, and one to many, such a structure would incorporate a vector of these link data type relationships. Such a tree node leaf structure type would have internal links that exist within the WSM project, and external links that exist outside the WSM project. Through observation and design, only external links from any html file in the WSM project to a file external to the project can be defined. All of these external links will be considered as irrelevant and only stored for reference to retrieve files and data to be included into the WSM project if needed. Thus the importance of internal WSM project links from a source and to a source for each file in the project are defined and needed for updating. The links to stored as a vector array to be read and updated are stored in the data structure ws_tree_node_item_struct_type defined in **Listing 03:02**.

Since every WSM tree node represents a file that potentially has links to and from another file of the data structure of ws_tree_node_item_struct_type in **Listing 03:02,** each tree node needs to have as part of a final tree leaf data type structure these vector arrays of ws_tree_node_item_struct_type. However, each tree leaf data type structure also needs as its core definition, the tree ws_tree_node_item_struct_type to define the file the tree node represents.

Such a C++ structure that exists in the WSM application is given as ws_tree_node_leaf_struct_type in **Listing 03:03**.

```
struct ws_tree_node_leaf_struct_type : public ws_tree_node_item_struct_type {
	std::vector <ws_tree_node_item_struct_type *> internal_links_from = {};
	std::vector <ws_tree_node_item_struct_type *> internal_links_to   = {};
	std::vector<std::filesystem::path>                    external_links_to   = {};

	bool update_link_path = false;
};
```

Listing 03:03

The data structures that forms the basis of all ImGui tree node elements and can be found in the C++ file

Source→Tree_Framework→ws_tree.h

With the data structures defined by **Listing 03:01** to **Listing 03:03** the C++ coding to create, display and define the functions to allow user interactin of the WSM tree to manage the links and files of a defined web site can begin.

04 : Web Site Manager Tree Class

The web site tree class is the C++ class object file that forms the basis of the definition of a WSM project and perform all the WSM tree functionality and processes to create, display, modify the WSM tree nodes and update the links to the files of the html files present. The following text can be best understood by viewing the C++ source code file

Source→Tree_Framework→ws_tree.h

and locating the C++ class ws_tree_class .

04:01 ws_tree_class as the basis of the WSM tree

The data structure ws_tree_node_item_struct_type given in **Listing 03:03** gives the basis of an individual tree node, but does not define the entire tree itself. The C++ class ws_tree_class does this.

Each tree branch has child tree branches that reflect the directory file structure of the web site project that the ws_tree_class is modelling. Each tree branch has branch leaves that reflect the files that exist within the directory that the tree branch is modelling. Thus from this point on, when speaking of a WSM tree, one can consider that it is the file structure of the web site that is spoken of where branches are directories, and leaves are files of that web site project.

Because more than one child branch or leaves can exists within each branch of a tree, then a C++ vector data type can be used to store a data value representing these child branches and leaves for any particular tree branch. Thus a child branch is also of the same data type as the parent branch.

The root branch of the tree structure is the ws_tree_class itself, and since all child branches of the ws_tree_class are also branches, then their data type is also ws_tree_class. Each branch of a WSM tree as explained in section 03 is a tree node item, and each tree node item has a structure of ws_tree_node_item_struct_type as given in **Listing 03:02**. Therefore the ws_tree_class inherits this structure.

Each tree leaf is defined as a data structure defined by ws_tree_node_leaf_struct_type as given in **Listing 03:03**.

Thus as a start to the ws_tree_class its structure is defined as in **Listing 04:01**

```
class ws_tree_class : public  ws_tree_node_leaf_struct_type{

        std::filesystem::path parent_path = INVALID_PATH;
        ws_tree_class* parent_branch     = nullptr;

        std::vector<ws_tree_node_leaf_struct_type*>  leaves = {};
        std::vector<ws_tree_class*>                  branches = {};

};
```

Listing 04:01

For the purpose of easier tree traversal and management, the ws_tree_class leaves and branches are stored as pointers. To allow traversal and other purposes, a pointer and path name to reference the parent branch is required

Listing 04:01 with the data structures of **Listing 03:01** to **Listing 03:03** in section 3 forms the basis of all the functions and operations dealing with the management of files, directories and links of the web site project performed by the WSM application.

04:02 Create Tree of Web Site Project

To create a tree representation of a web site project, there needs to be given a path name to the top or root directory of the web site file structure. Then from this information a tree root node of the data type ws_tree_class is created that defines the entry point as it were to the web site project file structure. A test is performed to validate that the directory exists on the computer before the root tree node is created, and if valid, the ws_tree_class is initialised to give information that this tree node is a root tree node and all the relevant information to represent the web site root directory so the application can find the actual root directory file on the computers storage hardware and the user can identify the project root tree node on their GUI. ie directory path name and label of the root tree label. Other relevant information for processing and tracking of link and directory file data are also defined and stored in the data structure of the ws_tree_class as described in section 05:01.

The function that performs this task is part of the ws_tree_class class and is called create_root_tree.

From this root directory that exists on the computer storage, the application queries each entry that exists in the root directory and determines if it a child directory or a file. The function that performs this task is called create_ws_contents and is given a parameter of the directory path name that it is to process data. The very first directory is the web site root directory.

If it is a file, then the a tree leaf is created and the data entries defining that tree leaf to represent that file is stored within a ws_tree_node_leaf_struct_type, and a pointer to that ws_tree_node_leaf_struct_type is then stored in the leaves vector data type for the root tree branch.

If is a directory, then a child branch data type that is also of the data type ws_tree_class is created and the relevant data assigned to the variables to define and represent that ws_tree_class as a file directory in the computers memory storage. The function create_tree_branch performs this task. A pointer to this child branch is added to the tree branch branches storage vector data type.

For each branch or directory that exists, the above process of the function create_ws_contents is repeated in the form of a recursion. This works because the directory structure of any computer can be considered as a fractal and fractals have a recursion of structure as a core attribute.

This act of recursion of reading directories into a ws_tree_class structure eventually builds up a tree data structure that directly represents the directory structure of the web site project that is to be managed. **Fig 04:01** gives an illustration of this recursive process.

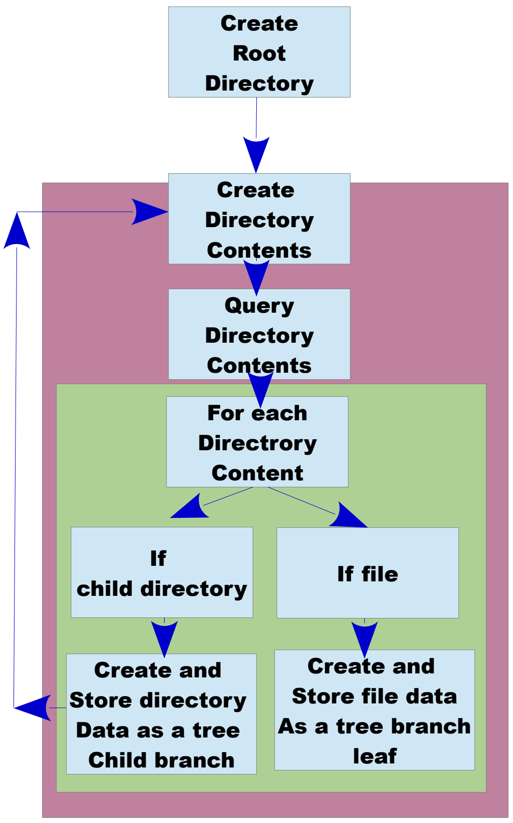

Create Root Directory
Create Directory Contents
Query Directory Contents
For each Directrory Content
If child directory
If file
Create and Store directory Data as a tree Child branch
Create and Store file data As a tree branch leaf

Fig 04:01 Schematic of creating WSM web site tree

When creating and storing the details of a file that exists in the web site project, a designation of a category of file type is assigned to that tree leaf data structure type dependent upon the file extension of the file being examined. These file extensions are stored in different C++ vectors as a string within the ws_tree_class class where each vector is used to define a category file type. Such C++ vectors are defined in the ws_tree_class as

```
std::vector<std::string> image_extensions = { "jpg" ,"jpeg","gif","tif","png","webp","svg","bmp" };
std::vector<std::string> html_extensions = { "htm" ,"html", "xhtm", "dhtml","php" };
std::vector<std::string> text_extensions = { "asc","txt","csv" };
std::vector<std::string> script_extensions = { "css" ,"js"};
std::vector<std::string> media_extensions = { "mpeg","avi","webm","mp4","flv","swf"};
std::vector<std::string> audio_extensions = { "weba","wav" };
std::vector<std::string> pdf_extensions = { "pdf" };
```

where the name of the vector is of the format <category_type>_extensions that identifies the category type that the given file extension belongs to.

The source code of of the ws_tree_class to create the representation of the directory structure of a web site project as a C++ tree structure exists within the file

Source→Tree_framwork→ws_tree.h

05 : HTML Manager Tree Class

The html manager class is a C++ class that performs all the tasks of reading and updating the html file links and link data of the html files that exist within the web site project. The C++ class that performs all these and related tasks exists in the file

Source→File_manager→html_manager.h

with the C++ class name ws_html_manager_class.

The storage of the link data for each html file is defined by the method of linking that each html file uses as its base, which is a file path name to a file that the html has a hyperlink to another html file, or a link to a file that is a resource to be used to display or load data, or perform some user interaction. Using the same arguments and reasoning as in section **03:01** to form a method of storage for the link data, the C++ method of storing the link data of any html file is a similar use of the C++ vector array to store file path names of the links to a file, and links from a file to the html file. Such a vector storage of link data is defined within the ws_html_manager_class as

```
std::vector<std::filesystem::path> internal_links_from = {};
std::vector<std::filesystem::path> internal_links_to     = {};
std::vector<std::filesystem::path> external_links_to    = {};
```

Take Note : To avoid confusion that the convention used in the WSM application are that links to are links that the html file points to, and links from are links from another html file. This is important to understand when reading the C++ code.

Also the file path name that is stored in this link data is a relative link, and if any absolute link path names are given in the html file they need to be converted to a relative link before being stored.

05:01 Find and Create html link data

To crate the link data of a web site project, the links only exist from and to the html files of that web site project. Thus only the html files need to be examined or read to extract the link data of any web site project. Since html files are basically text files, and that the link data is defined consistently in a narrow band with of format that can be considered as a form of coding, this examination and extraction of link data should not be difficult to perform.

Html files have link data in two forms, hyperlinks to another html file, or a link to a source file. Hyperlinks are defined by the text href and source file links by src. By searching for the text href and src within a html file, a candidate for creating a link data can be defined.

To search for such link data, the designated html file is read into a buffer and the text within that buffer is searched to find text that matches either href or src. Within the html manager class, this buffer is a C++ vector that stores a C++ string data type. Each string of this text buffer vector is a line of the html file being examined. Being of a C++ string data type, to perform the tasks of reading, finding and extracting text strings etc can utilise and take advantage of the functions and attributes of the C++ string data type.

The function to read and create such a text buffer is performed in the function define_html_file_links of the html_manager_class. The examination and extraction of html and src link assumes that the entire path name of the linked file is on the same line in which the text html or src was found. By using the C++ string and some custom string functions, the candidate link path name is extracted and examined if it is a local internal link (ie the linked file exists within the web site project file structure under the project root directory) or is an external link (ie the linked file exists outside the web site project file structure or project root directory), or in the case of a hypertext href link, if the hypertext link is a link to a section of the same html file. All linked files are defined between the quote " text character, so searching for the first quote position and next one after it will give the path name of the linked file.

All hypertext links that link to a section of the same html file are ignored.
All href hyperlinks or src source links that link to an external file are stored in the external_links_to vector.
All href hyperlinks or src source links that link to an internal file are stored in the internal_links_to vector.

Because more than one link can exist within any html file that links to the same external or internal file, a test needs to be performed to make sure that the link is not already defined so as to avoid duplicate link data being stored.

Extracting and storing the links to computer system files for a html file that is for a tree leaf data type is thus not so difficult and is perform by the functions scan_for_href_links and scan_for_src_links within the C++ ws_html_manager_class class.

06 : Web Site Tree manager Class

The web site tree manager class is the C++ class that controls and performs the procedures of the application in creating, displaying and managing all the aspects of the application regarding to the creation of the data and interface of the application for the user to interact with the so as to obtain the results of modifying the file structure and updating the html and link data of a web site project.

The C++ web site tree manager class is called ws_tree_manager_class and is defined in the file

Source→File_manager→ws_tree_manager.h

06:01 Create web site tree

One of the main tasks of the ws_tree_manager_class is to create a C++ tree model representation of the directory file structure and links between files of a web site project. This is performed in the function create_ws_tree of the ws_tree_manager_class in the steps.

1: Build a tree model of the directory structure of the web site project. Performed by the function create_root_tree that exists in the ws_tree_class as discussed in Section **04:02 Create Tree of Web Site Project**

2: For each leaf of the web site tree that represents a html file, find and store the files that the html file has links to. Section **05:01 Find and Create html link data**

3: For each leaf of the web site tree that represents a html file, for each link that the html file has in its list of links to as found in step 2, find within the web site tree structure, the tree leaf node that represents that linked to file. Define a link from data entry to indicate that the file that the leaf data node represents has a link from a html file to it.(**03:01 WSM html linking Fig 03.02** and **Fig 03.03)**

To perform step 2 the ws_tree_manager_class calls up the function define_html_tree_leaf_links that exists in the ws_tree_class object, This function in turn calls up the define_html_file_links function of the html_manager_class (Section **05:01 Find and Create html link data**) to find and assign the files that the html has links to. If any links are found, then those links are converted to an absolute file path name and this absolute path name is added to the vector storage data type links_to of the tree node that represents that html file.

The reason for absolute file path names being stored as a link in the web site tree model is so as to locate in the real directory file structure stored on disk the file that has a link to or from it. Also an absolute file path name is used to locate the web site tree node leaf that corresponds to the linked file and that is vital to performing the above step 3.

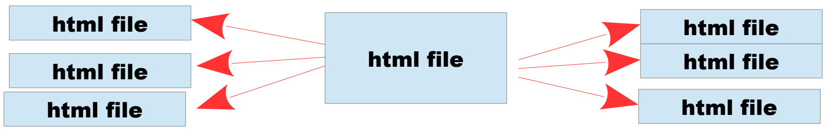
To find the tree node leaf that represents the file that a html link refers to, a traversal is made from the root node of the web site tree based upon the absolute path name of the html file link data. Each tree node has a value defined of the absolute path name of the file that the tree node represents which was assigned to it while the web site tree structure was being created in step 1 above. By querying the path name of the html link, a progression through the tree structure is conducted to find the tree node leaf that represents the linked file is found.

Once found, a pointer to this tree node leaf structure is returned and stored in the tree internal_links_to storage vector data type. Conversly, a pointer of the tree node representing the html file that has a link to the file that this tree node leaf structure represents is added to its internal_links_from storage vector data type. This makes it much easier to change link data should either the html file or the data file have their attributes changed such as names or file locations that are expressed through their file path names.

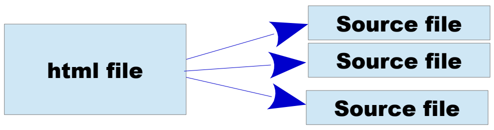
The above end result of building the web site tree data can have each box representing a html or source file being a tree node data structure with the arrows being pointers of the internal_links_to and internal_links_from as in **Fig 06.01** to **Fig 06.03**

What this also does is to make it easier and efficient for the real html files to have the text updated of any link data that exists within them that is changed.

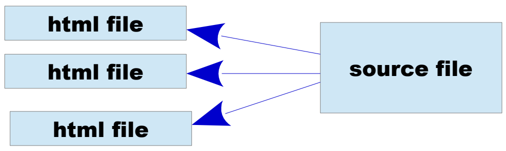
Once step 3 above is completed, the web site tree data structure is initialised and ready for the application and user to use for performing tasks to manage the real directory file structure of the web site stored on computer disc memory and change and update the links within the html files as files are moved, renamed or some other function performed that changes the links of a html file.



Links From Pointers Links To Pointers
Fig 06.01



Links To Pointers
Fig 06.02



Links From Pointers
Fig 06.03

07 : WSM Application Class

The WSM application class is the main C++ class that controls and performs the top most functionality procedures of the application in creating, displaying and managing all the aspects of the application to create the data and interface so as to act as the gateway and controller for the user to interact with the application and obtain the results of modifying the file structure and updating the html and link data of a web site project.

The C++ class that is the WSM application class exists in the file

Source→Application→wsm_application.h

07:01 Initialisation of web site manager application

1: Upon starting the web site manager application, the first task that is performed by the wsm_manager is to define an ImGUI window panel to display the web site tree structure within, and Create the application main menus and initialise user preferences to be used in the application.

2: Define the web site GUI tree representation of the web site project directory file structure, and define all of the link data that is applicable to the all of files of the web site project within this tree representation or model.

This is done by building a tree model of the directory structure of a selected web site project as outlined in section **06:01 Create Web Site Tree**. If no web site project root directory is selected then no web site tree is created and displayed until the user defines one.

This web site tree representation model is then displayed on screen utilising the ImGUI tree widget functions, and the user interactions with this ImGUI tree GUI is handled by the ws_tree manager_class class object. **Section 06**.

08 : HTML Link Update

The major reason for this application existing is to be able to manage automatic updates of html file links within a html file whenever a file that has a link to if from one or more html files has its directory path name location changed either by moving it to a directory, renaming it, or deleting it. Similarly if a html file is moved, renamed or deleted, any relevant link data within that or other files are also automatically updated.

To achieve this task is a simple principle. Knowing the absolute path name of the file in which a link is to be updated, and the absolute old path name to the file that is being changed, and the new absolute path name to which the file that the file is being changed to is a matter of comparing those absolute path names and changing the link path name based upon that comparison.

More precisely, the absolute path name of the html file or source of the link to another target file of that link would share a common path name to a directory that is common to both of them. Subtract this common path name from both absolute path names of the source and target, a difference in the path to each file is then given. From this difference, it can be deduced if the target file is in the same directory as the source, or in a sub-directory of the source, or is in a directory path above that of the source.

From this difference, a new relative path name of the new target path name from the source can be deduced to replace the old link path name that exists within the source html file.This is why in the C++ data structure ws_tree_node_item_struct_type given in **Listing 03:02** there is a previous_item_path variable. This stores the old or previous absolute path name of a linked file to search for within a html file which is to be replaced by its new absolute path name that is converted to and searched for as a relative path name.

This update task is performed by the functions update_html_file_links and update_html_file_from_links that exist within the htm_manager_class. Both of these functions call a function get_relative_pathname that is a static function defined in the C++ class file_path_tools_class. This get_relative_pathname function performs the task of getting a relative path name from a source file to a target file as described above.

The C++ class file_path_tools_class exists in the file

Source→File_manager→file_path_tools.h

Two issues occur in performing this and other related tasks concerning file path names and C++ std::strings in general.

1 : The Microsft Windows operating system uses forward slash '\' as well as backslash characters within a path name. This interferes with many C++ functions and other computer system file path name conventions such that this forward slash is interpreted to mean that the character following it designates a special command or formatting feature is to be performed.

2: The C++ standard string data type when initialised for the first time to a given string of characters is given a fixed length or number of characters that corresponds to the length of this initial string. This length cannot be changed by reassignment to any new string of characters which can result in a truncation of a string of longer length, or a series of trailing nulls or undefined character values after a terminating end of string null value.

Both of these issues can cause enormous problems in dealing with the updating of html links.

The forward slash problem1 was simple to solve by just replacing any '\' characters found in a path name with a '/'.

The constant string length problem when reassigning file path names of a different number of characters 2 was solved by using the C++ std::string data type resizing function. However when performing this task, it is the reassignment of a path name using a string c_str() function that must be used in resizing a string A to that of another string B, and never using the size of the string data type B itself as the size of string B will give the number of characters in the string it was originally initialised to.

To do this the following type of code needs to be implemented

```
std::string string_B = “string”;

std::string string_A = item_path.string().c_str();
int string_A_size = string_A.size();
string_B.resize(string_A_size);

string_B = string_A; or string_B = item_path.string().c_str();
```

09 : Web Site Tree Node Selection tree_selection_struct_type

The user of the web site application will need for the application to function, to perform a selection of one or more tree nodes representing either a directory or files to be moved or to be selected for other functions. The ImGUI toolkit has the functionality for the C++ code to permit the user to be able to select tree nodes to then have function tasks performed on the basis of that selection. Multiple selections can be made of the web site tree nodes, and a storage of these tree nodes can thus be stored within a C++ vector data type. Each tree node is of the data structure type ws_tree_node_item_struct_type as described in section **03:01 Listing 03:02**, and since the web site tree is made up of pointers to this data type, then so this tree node selection vector stores pointers of the data structure type ws_tree_node_item_struct_type. ie

```
std::vector<ws_tree_node_item_struct_type*> tree_node_selection = {};
```

Such a storage vector is defined in a C++ data structure called tree_selection_struct_type that exists within the C++ header file ws_tree.h. tree_selection_struct_type could be considered as a class, but its primary function is to store user web site tree node selections and perform tasks and manage the selection data, not perform application tasks.

When the user nominates to perform an application task such as to move, rename or other related task on a web site tree node, then a reference to all the entries that exist within the tree_node_selection vector within the tree_selection_struct_type is made.

The C++ class tree_selection_struct_type exists in the file

Source→Tree_Framewok→ws_tree.h