WAPP  
Windows App Framework

WAPP is a C++ class library to help organize and simplify the creation of semi-sophisticated Microsoft Windows desktop applications. It provides a series of classes that provide simplified access to various Windows functionalities.

It layers on top of Direct2D for its graphics package, and so it requires a system that supports Direct2D, which should be most Windows 11 and beyond systems.

WAPP eliminates or simplifies a lot of the boilerplate code that often clutters up Windows source code, and abstracts away many of the mistakes beginning Windows programmers fall into.

# Creating Your First Application

## Parts of a **WAPP** Application

### **WAPP** Headers and Libraries

**WAPP** is a static link C++ library. The definitions and some inline functionality is provided in C++ header files.

### Resources

Resources are not a **WAPP** feature, but a Windows feature that **WAPP** builds on top of. The intent is to simplify international releases of applications, where everything that should be translated in a foreign language version of the application is in one place.

Resources should only contain items that need to be modified by localizers/translators. And anything that needs to be translated should be in the resources. This means most strings, error messages, dialog boxes, and menus will exist in the resources. Often graphical images will be here, too.

### The **WAPP** Object

The root object created by every application will usually be a **WAPP** object, or the “Windows Application”.

### The **Run** Entry Point

Every application gets control from **WAPP** from the **Run** entry point. Typically this is a very simple function, which creates the **WAPP** and then calls the message pump which runs until the application terminates.

### Message Pump

Windows applications use what we call a ‘message pump’, which is an idle loop that monitors the system for user input and then directs that input to the appropriate place. Windows has a standard protocols and conventions for directing messages to the appropriate places. The standard **WAPP** message pump includes a feature for overriding default processing by filtering certain events and handling them in a non-standard manner.

### Commands

Commands are how your end user tells your program what to do. Menus launch commands, and buttons launch commands. **WAPP** provides a flexible command framework that should permit applications with complex command processing, including undo/redo and command recorders.

## Setting Up the Project and Build

The WAPP header files must be added to the include path for your C++ compiles and your resource compiler. The location of the WAPP library needs to be added to the linker library path.

The compilation process will automatically add a linker directive to pull in the necessary library files, which includes various system libraries and the **WAPP** library itself. As long as the include and library search paths are correctly set up, the rest of the build should work seamlessly.

## Adding Standard Resources

At the very minimum, applications should have a menu resource, string resources to name their application, an application icon, and an About dialog box to tell the user a little about your application.

Adding a resource requires two steps: (1) adding the identifier of the resource in **resource.h**, and (2) entering the resource in your applications **.rc** file.

## Writing your **WAPP** Object

## Writing Simple Commands

# Drawing and Coordinates

Drawing is performed on a **DC** object, or a drawing context.

## Points, Sizes, and Rectangles

## Brushes

## Shapes

## Text

## Images

# Windows Objects

A particularly powerful feature of the **WAPP** library is **WN** object.

**WAPP** allows you to create a child **WN** of any **WN**. The child **WN** is owned by the parent **WN** and is drawn entirely within the parent’s bounding box.

Every **WN** is a drawing context or a **DC**, so all graphical drawing operations are available in a **WN**. However, if you want to draw your WN outside the normal Draw notification, you must call **BeginDraw** and **EndDraw**. In general, we recommend calling Redraw

# Handling User Input

## Keyboard

## Mouse

## Commands

# Controls

## Layout

Controls can include information for automatic layout, which is especially helpful in dialog boxes. A layout engine object is typically used to aid in control sizing and positioning. As long as certain attributes are set, most layout can be accomplished automatically.

### Box Model

The box model we use for controls is similar to that used in HTML and CSS. Controls live within a box, which is the interior of the control’s WN. Inside the interior, controls have a **border** and a **padding**, and the **content** of the control is inside the **padding**.

The padding area is typically filled with just the background color of the control, and the outline is typically a thin line drawn in the text color. The control’s Draw method is responsible for filling in the content area.

Exterior to the control is a **margin**, which is used to position controls relative to its sibling controls and its parent. The margin is only used as part of the automatic layout process, and is not enforced by the control itself. The margin between two adjacent controls will be the maximum of the two control’s margins – they do not add.

Content

Border

Margin

Padding

Interior



Many controls have an option to display attached labels

Need terminology for the different rectangles.

Global

Bounds

Non-client area.

Interior

Local

Client

Content