G.D.G. Software

EXEOUTPUT FOR PHP

PHP Compiler for Windows

- GUI Apps for Windows
- Console Apps
- Server Apps

USER GUIDE

Table of contents

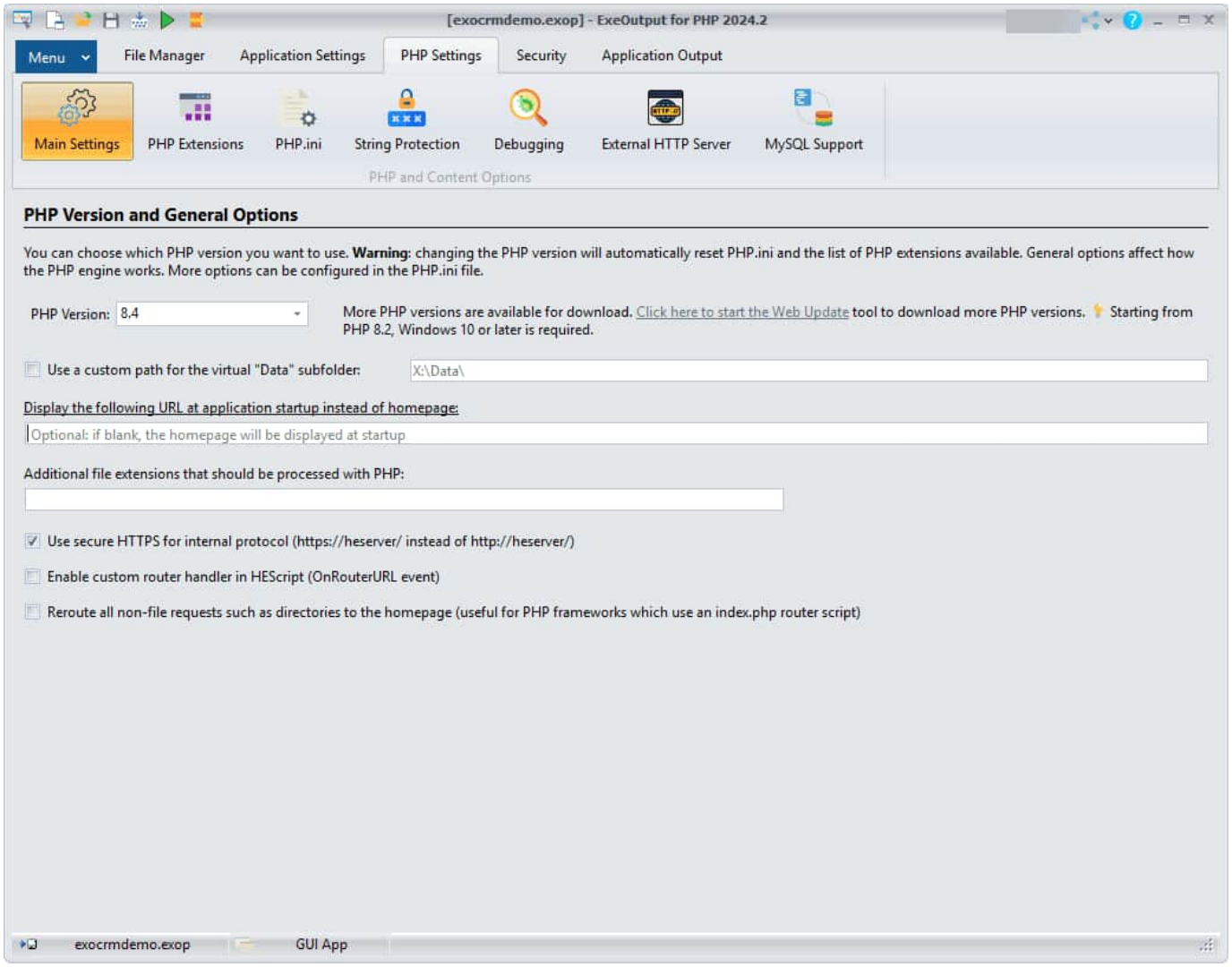
1. Welcome

1.1 Welcome to ExeOutput for PHP

This is the official documentation for ExeOutput for PHP.

1.1.1 Description

ExeOutput for PHP™ is a **compiler that transforms PHP scripts and websites into stand-alone Windows desktop and console applications**.



1.1.2 What's New in ExeOutput for PHP 2025

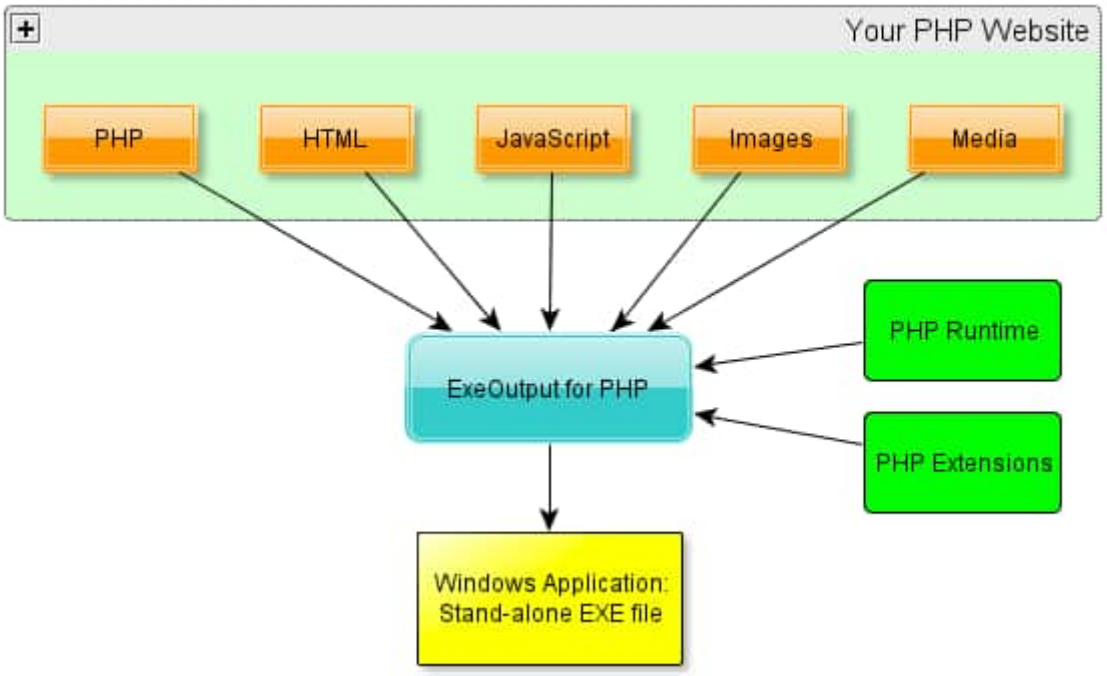
ExeOutput for PHP 2025 introduces major new features and improvements:

- **New Rendering Engine: Microsoft Edge WebView2**. A new option to use the modern WebView2 runtime, resulting in a drastic reduction in EXE file size (around 20 MB vs. 150 MB with CEF). You can easily switch between WebView2 and CEF with a single click in the application settings. See the updated topic about rendering engines.
- **Upgraded Chromium Engine (CEF)**: The CEF engine has been updated to version 139.0.38.
- **Updated PHP Runtimes**: Initial support for PHP 8.5 (beta) and updated versions for PHP 8.4, 8.3, 8.2, and 8.1. See available PHP versions.
- **New Compression Engine**: Faster and more efficient multi-threaded file compression with Zstandard (zstd).
- **And many other improvements**: including enhanced security, bug fixes for popups and multi-instance execution, and UI enhancements. A full list of changes is available in the release notes.

With ExeOutput for PHP

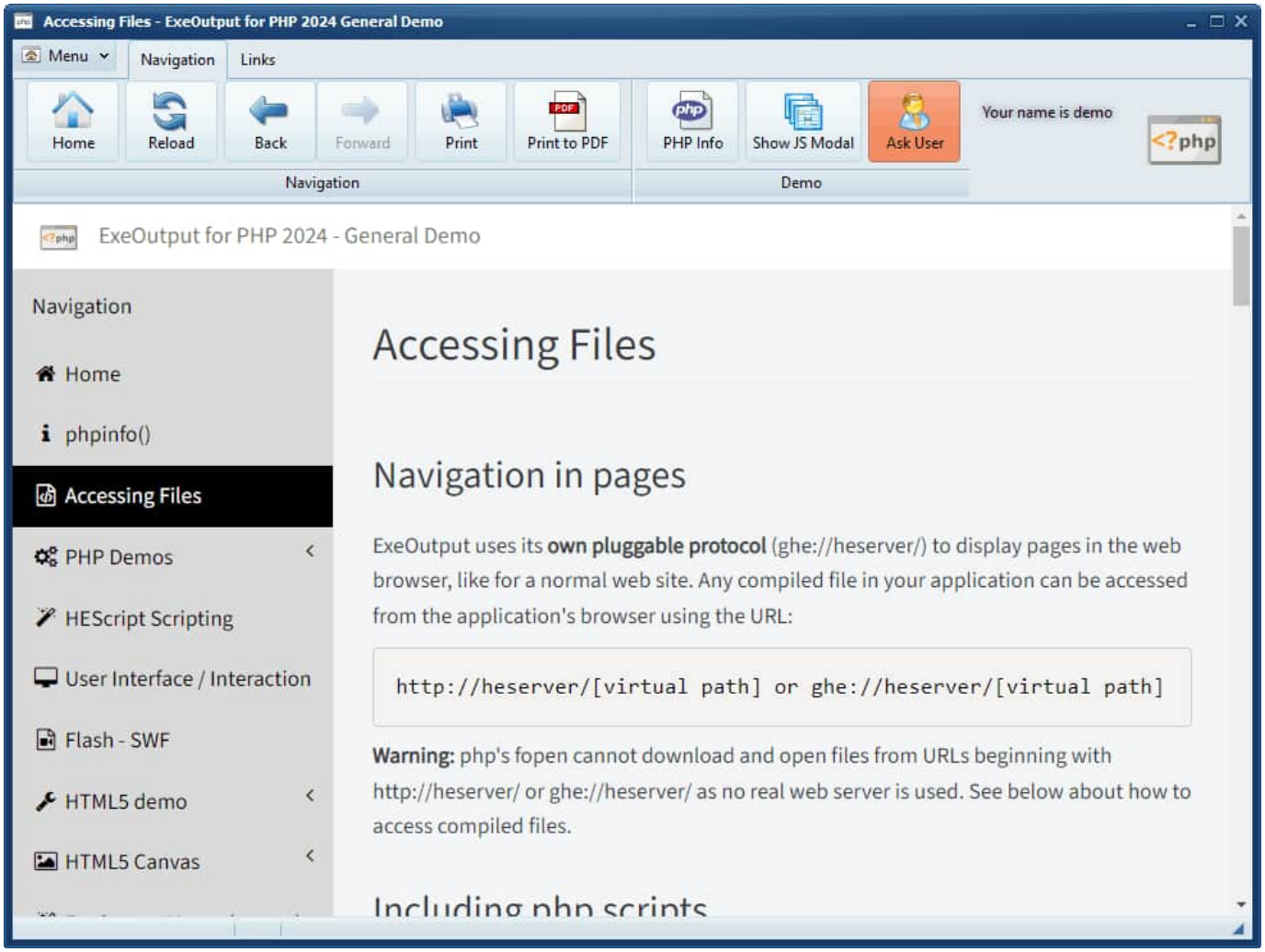
Convert PHP scripts, websites, JavaScript, HTML, and databases into stand-alone Windows applications (single EXE files) that do not require a web server or a separate PHP distribution. Build fully-customizable desktop applications with PHP, HTML, and JavaScript, with no additional skills required.

- **Create a single EXE for Windows**: ExeOutput for PHP packages the PHP runtime and all project files (PHP, HTML, JavaScript, assets, databases, etc.) into one stand-alone EXE. The files are compressed and protected, ensuring they cannot be extracted later with a decompiler or an archive tool like 7-Zip.



- The executable is **stand-alone**: the PHP runtime is embedded within the EXE and is never unpacked to the hard disk.
- You can choose the **PHP version you want to use**, from PHP 7.3 to PHP 8.5.
- PHP scripts are **executed directly from memory** and are never unpacked to the hard disk.
- You can create **console (non-GUI) and command-line applications** from PHP scripts. PHP runs in PHP CLI mode.
- Applications are rendered using the **Chromium Embedded Framework (CEF) v3 engine**. No third-party software is required, making the applications you build **fully stand-alone**.
- Basic **PHP extensions are supported** and can be embedded within the EXE. Database engines like SQLite are handled seamlessly by the PHP engine. Additionally, your applications can **connect to remote database servers such as MySQL**.
- PHP pages and websites are displayed in a **custom, secure browser environment**. Standard navigation tools and menu commands are available and can be customized to your needs.
- Design **unique GUIs for your applications**, including skins, ribbons, toolbars with custom buttons, menus, tray icons, alpha-blended splash screens (32-bit PNGs), and more.
- Use **existing PHP functions and extensions** in your applications, such as cURL, graphics, XML, PDF, Zip, databases, and more.
- AJAX functionality, secondary windows, and pop-ups behave as expected.
- Optionally, let users access your application through their standard web browser (`localhost:port`) using the built-in HTTP server.
- **HTML5 and CSS3 are fully supported**.
- Debugging tools like the CEF Developer Tools and XDebug are included.
- Support for open audio and video formats and WebRTC is included.
- Optimized for high-DPI and 4K screens, as well as multi-monitor setups.
- Features for printing, kiosk printing, and exporting to PDF are available.
- **No web server required**: no port or firewall conflicts to worry about.
- Optionally, you can include an external HTTP server to allow external web browsers to access your application via `http://localhost:port`.
- Include a portable MySQL (MariaDB) server with your PHP application that starts and stops automatically with it.

- Supports various database servers.
- EXE customization allows you to use your own icon and version information.
- Applications can be digitally signed with a code signing certificate (Authenticode with SHA-256), avoiding security warnings in Windows.
- ExeOutput includes its own scripting engine (HEScript), which can be combined with PHP and JavaScript.
- Create **portable applications**.
- Add optional licensing features to sell licenses for your PHP apps with Obsidium.
- Disable printing functionality across the entire application, preventing users from printing any content.
- Full **Unicode support** is provided for the application, though PHP itself is not Unicode-compatible.
- Compatible with various PHP frameworks.
- Project settings are managed with an intuitive GUI featuring a Windows Ribbon-style interface.
- Easily deploy applications with integrated Zip archive or installer creation.
- Compatible with Windows 7 through Windows 11.



1.1.3 How to Use This Documentation

- **Browse** topics with the **table of contents** on the left to discover ExeOutput for PHP's features.
- **Press F1** in the application to open the help topic related to the current page.
- Use the **search engine** to find information on a specific topic.

- If you have **questions**, feedback, or bugs to report, visit our **forum** at gdgsoft.info or contact us by **email** at `info@exeoutput.com` (remove NOSPAM).

2. Getting Started

2.1 Getting Started

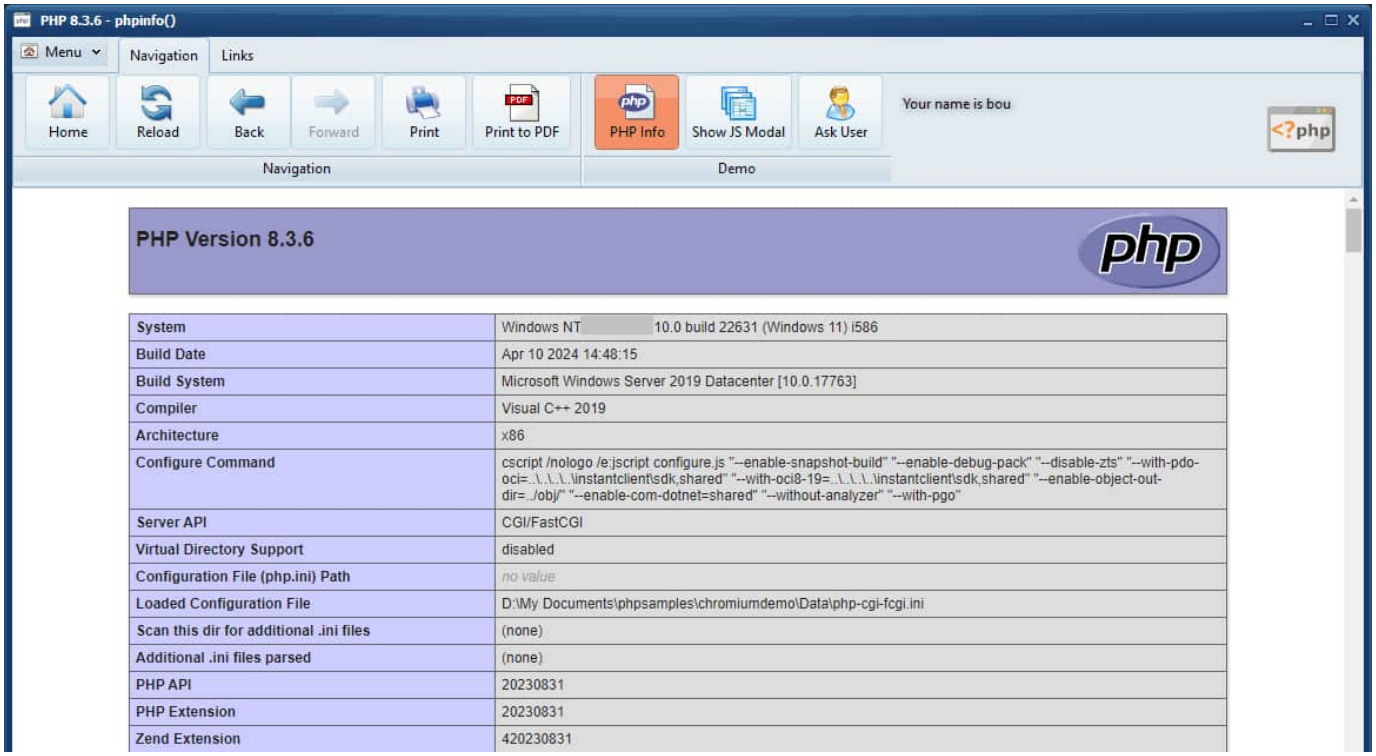
ExeOutput for PHP™ is a **visual PHP to EXE compiler** that converts PHP scripts, PHP applications, and PHP-powered websites into standalone desktop and console applications for Microsoft Windows.

Create eBooks, custom web browsers, database front-ends, games, interactive catalogs, daemons, console programs, HTML5 applications, and much more by combining the power of PHP, HTML, and JavaScript into desktop applications.

2.1.1 Creating GUI and Console Applications for Windows

Applications built with ExeOutput for PHP can feature a GUI and function as a **standard web browser**. Users can navigate your PHP pages as if they were using a web browser, without needing an internet connection, PHP installation, or a remote web server. PHP scripts are executed by the built-in PHP runtime, and results are immediately displayed to the user. Additionally, companion files such as HTML, images, JavaScript, XML, CSS, and more are seamlessly handled.

The screenshot below shows a simple application displaying the output of the `phpinfo()` function:



You can customize the user interface to fit your needs. For example, the PHP Info button in the screenshot above is not included by default.

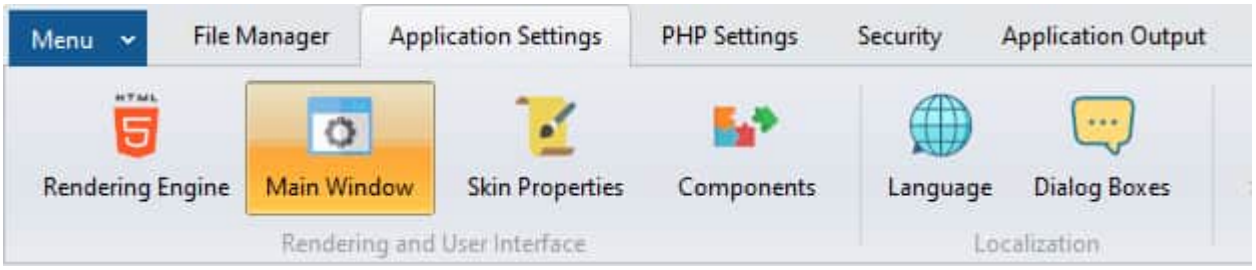
You can also generate PHP console applications (CLI) that run without a GUI, executing PHP in CLI mode.

If needed, you can create an external HTTP server to allow your GUI application to be accessed from external web browsers using `http://localhost:port` URLs.

2.1.2 Navigating the Interface

☑ Use the table of contents on the left to browse different help topics and explore the features of ExeOutput for PHP. You can also **press F1** or click the Help button in ExeOutput for PHP to directly access the relevant help topic.

☑ ExeOutput utilizes **ribbons**, grouping all related features into pages under one of the following five tabs: File Manager, Application Settings, PHP Settings, Security, and Output.



- **File Manager**: Manage files that will be compiled into your application. It functions like Windows Explorer, with a tree view of folders on the left and the files within the selected folder on the right. You can add, remove, edit files, and configure their properties.
- **Application Settings, PHP Settings, Security**: These ribbons are the core of ExeOutput for PHP, containing all the features that allow you to customize the behavior and appearance of your application.
- **Application Output**: Access options related to how ExeOutput for PHP will generate the application's `.exe` file.

> **Tip**
>
> To switch between ribbons, click the desired tab at the top of the main window. Standard and additional commands, such as Load/Save Project and Build Options, are available by clicking the **Application Menu** button:
>
> 
>

☑ Shortcuts are available in the window's toolbar:

- New Project
- Load Project
- Save Project
- Compile Application
- Run Application

2.1.3 Starting Your First Project

Each time you open ExeOutput for PHP, the welcome page is displayed, allowing you to create a new project or load an existing one.

👉 Advice for Getting Started with PHP Applications in ExeOutput for PHP

👉 Creating a Project

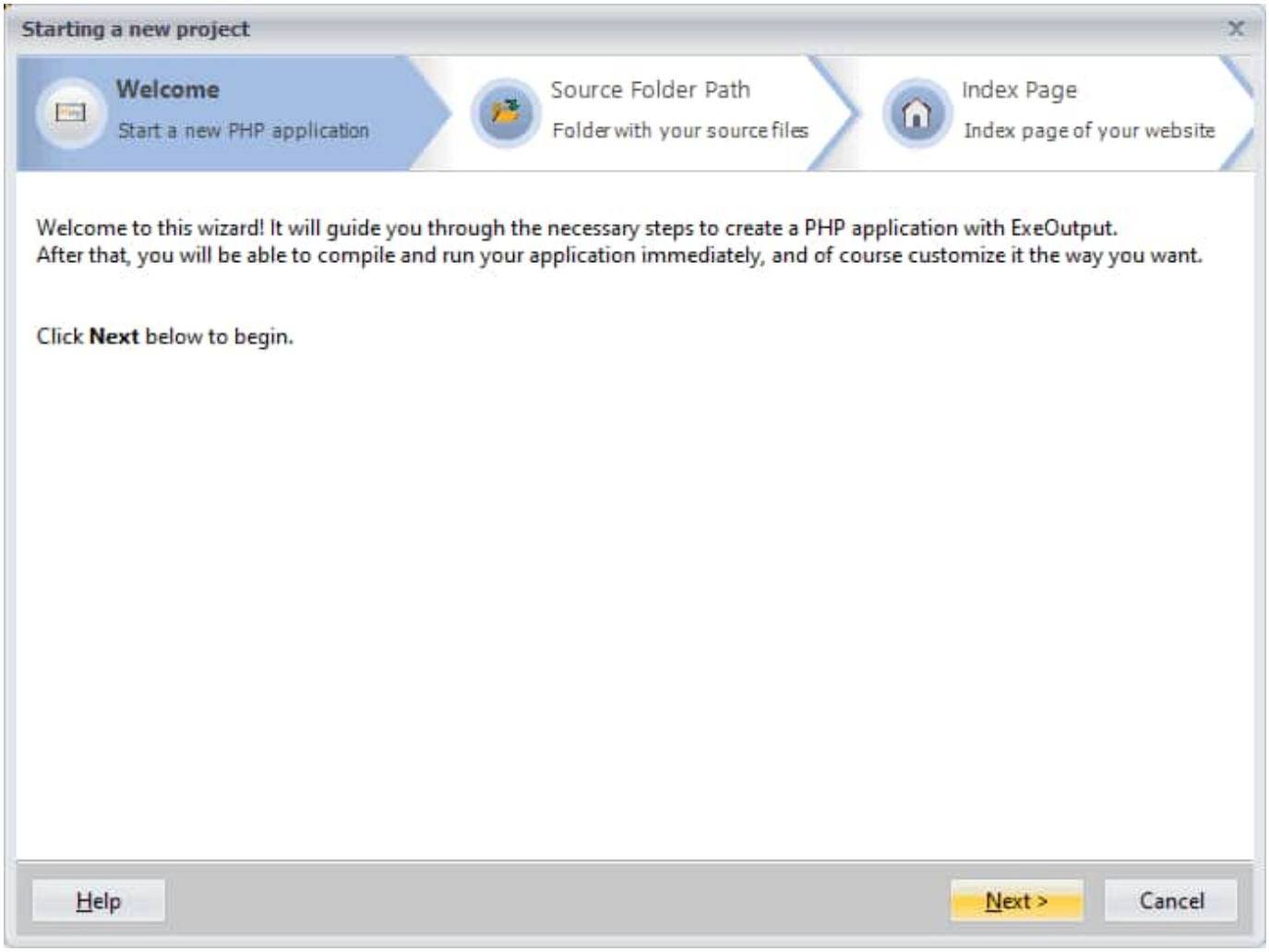
👉 How Compiled GUI PHP Applications Work

👉 How Console PHP Applications (CLI) Work

2.2 Starting a New Project

ExeOutput for PHP stores all application settings in a **project file** (extension: `.exop`). You can open and save project files at any time using the **Load** and **Save** buttons in the toolbar. But first, let's create a new project.

To begin, select **Start a New Project** at startup or click **New** in the toolbar. This will open the **New Application Wizard**:



The wizard will guide you through the initial project setup.

2.2.1 Step 1: Select the Source Folder

The **source folder** should contain all of your website's files: PHP scripts, HTML pages, images, CSS, JavaScript, etc. ExeOutput for PHP treats this folder as the application's **root directory**. All files within this folder and its subdirectories are automatically added to the project. You can add or remove files later using the File Manager.

Directory Structure is Preserved

Your website's directory structure is preserved within the application. The source folder becomes the application's root, and relative paths between your files will continue to work just as they do on a web server. Think of the compiled application as a self-contained web server.

2.2.2 Step 2: Choose the Index Page

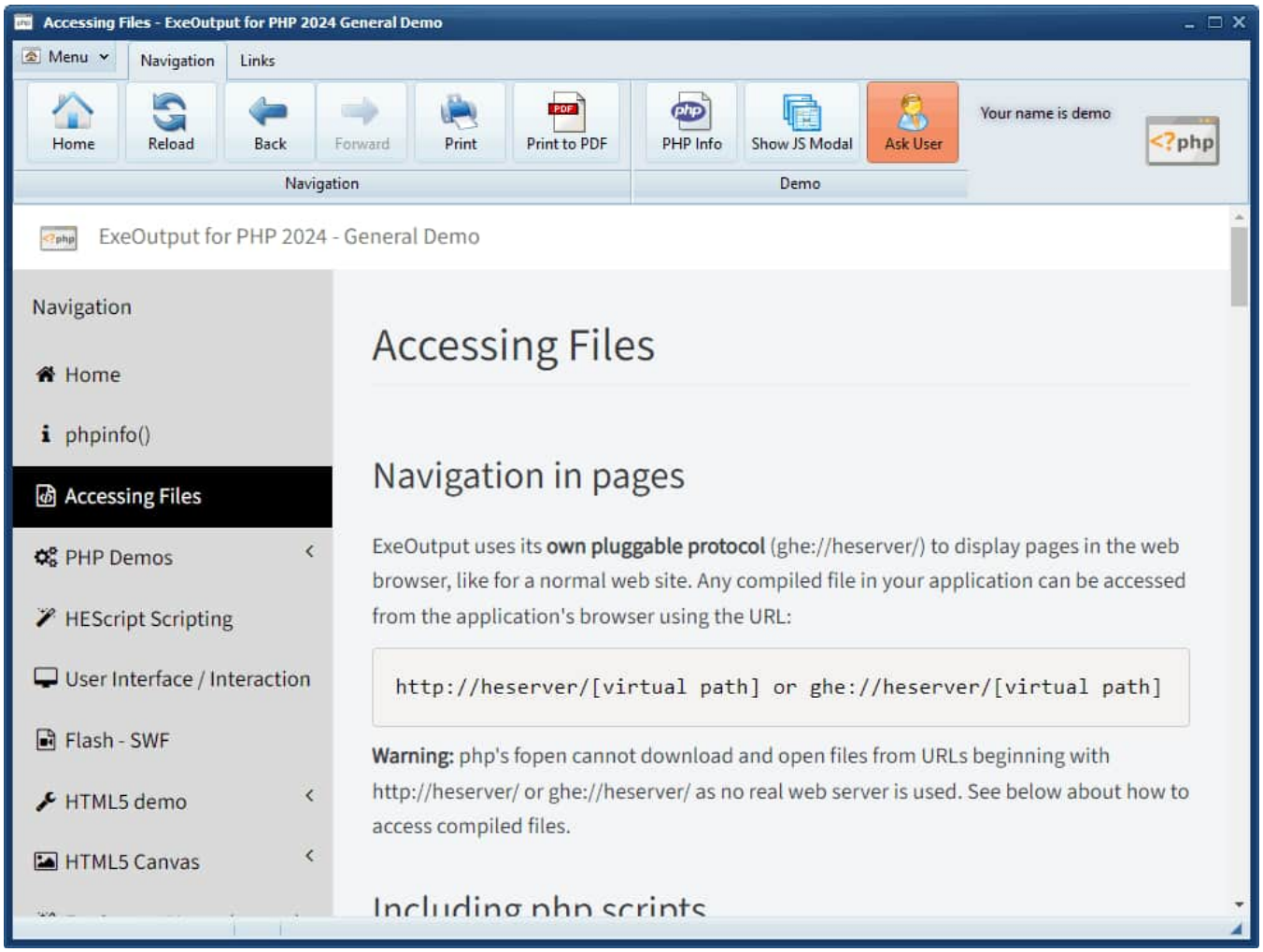
The **index page** is the first page users see when they launch your application; it acts as the home page. The wizard lists all files from your source folder, allowing you to select the desired index page. You can select a file within a subfolder.

Click **Continue** to proceed.

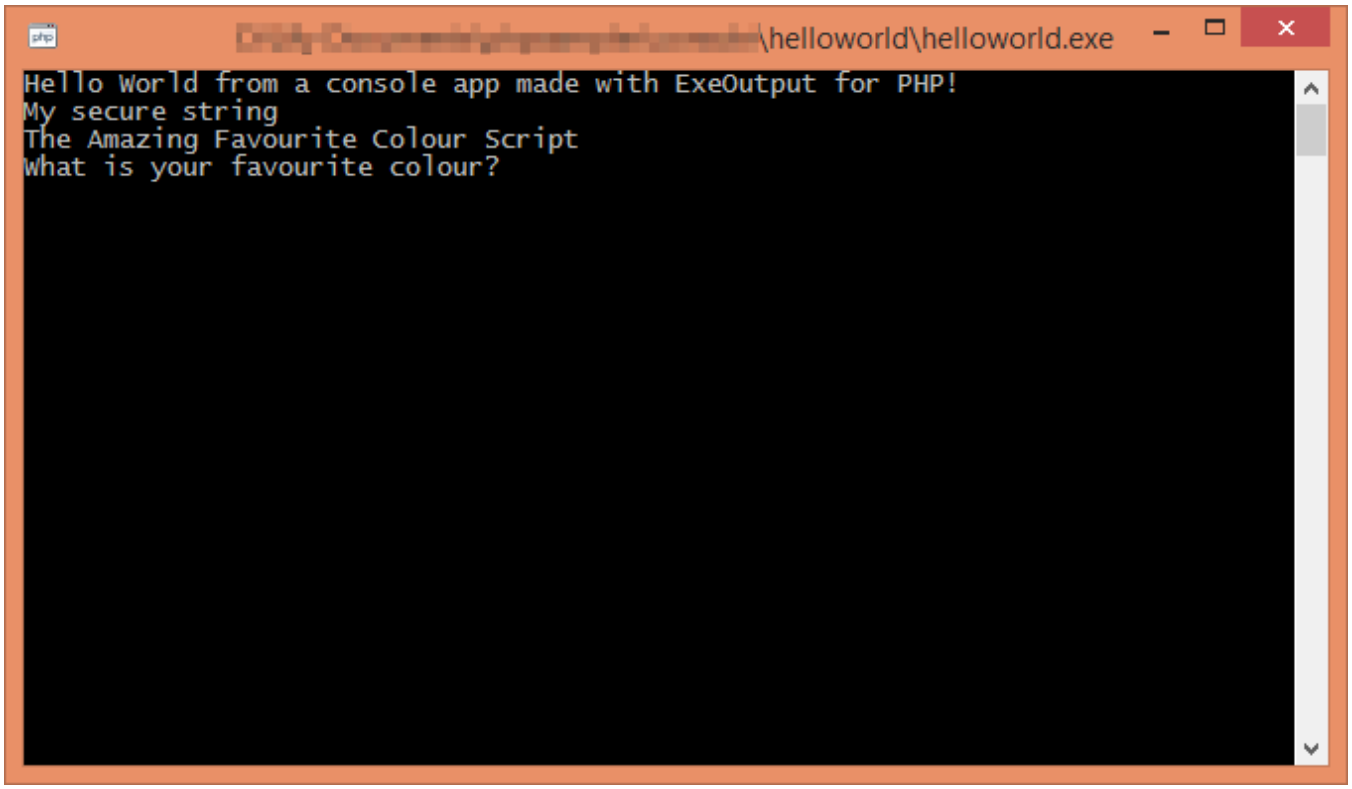
2.2.3 Step 3: Select the Application Type

ExeOutput for PHP can create two types of applications:

- **Graphical User Interface (GUI):** A standard windowed application. You will need to select a default theme for your GUI, which you can change later.



- **Command-Line Interface (CLI):** A console application that runs without a graphical interface. When launched, it operates within the standard Windows console and executes the PHP script you selected as the index page.



2.2.4 Step 4: Define Output Settings

This is the final step:

1. **Application Path:** Specify the full path for the output `.exe` file, including the filename. It is strongly recommended to save the `.exe` file in a directory separate from your source folder.
2. **Application Title:** Enter the title for your application. This will appear in window title bars, dialog boxes, and the Windows taskbar.

> **Warning**
>
> If your title contains a double-quote (`"`), you must escape it by doubling it (`""`) to prevent script compilation errors.

Click **Finish** to create the project. ExeOutput for PHP will then create the project file, configure default settings, and import all of your source files. This process may take a few moments, depending on the number of files.

> **Project Created**
>
> Once complete, you will be taken to the Main Window settings. Your project is now ready to be configured further or compiled immediately.

Next Steps

- How to compile your project
- Advice for getting started
- Working with PHP Frameworks

2.3 How to Compile Your Project

Once your project is configured, you can compile it into a single EXE file.

To compile your project, click the **Compile** button on the toolbar. Alternatively, go to **File** => **Compile**.

The compilation process may take some time, depending on your project's size and the number of files to be compiled. During compilation, ExeOutput for PHP will display a progress bar and a log window. The log window will show the compilation progress and any errors or warnings that may occur.

If compilation is successful, ExeOutput for PHP will display a message confirming that the project has been compiled. The compiled EXE file will be created in the Output Path you specified on the Output Settings page.

If any errors occur during compilation, ExeOutput for PHP will display an error message, and the compilation process will stop. You will need to fix the errors and then recompile your project.

2.4 Advice for Getting Started with PHP Applications

☑ Please note that applications created with ExeOutput for PHP are **not intended to operate exactly as they would on a web server**. You may need to adjust your application and PHP code accordingly. We recommend reviewing the General Demonstration included with ExeOutput for PHP by clicking the Application Menu button 

and selecting **Start General Demo**.

> **Note**
>
> As an exception, it is still possible to create an HTTP server application: your compiled application can simulate an HTTP server using the External HTTP Server option.

☑ By default, PHP error and warning messages are displayed. Your PHP website may not function correctly in a compiled state initially, so these messages can help you debug issues. You can disable them on the PHP Settings => Debugging page by unchecking **Display PHP error messages at runtime** and **Enable PHP error logging**. Errors will be saved in the `php_errors.log` file located in the same folder as the EXE.

☑ **Fatal PHP errors** cannot be suppressed; an error page will be shown (Fatal PHP Error). You can customize the error message using the `SPHPFatalErrorMsg` resource string.

☑ If you encounter "file missing" errors or "php failed to open stream" warnings, refer to accessing files in compiled applications.

☑ If your PHP website requires **extensions**, configure them first on the PHP Extensions page.

☑ If your PHP website needs **MySQL**, ExeOutput for PHP can include a portable MySQL (MariaDB) server with your PHP application and manage it automatically when your application starts or stops.

☑ If your application requires redirections to a script router, ensure they are configured using the redirection feature. **Note that `.htaccess` files for Apache are not supported.**

☑ You can adjust PHP parameters using the built-in PHP.INI editor.

☑ If your PHP code uses `getcwd()` to retrieve the current directory, consider replacing it. The current working directory (when launching the EXE) is available in the global variable named `HEStartCurrentDirectory`.

☑ For security reasons, the "**View source code**" option in the final browser is disabled by default. You can enable it for debugging HTML code generated by your PHP scripts on the Components => Context Menu page by selecting **EnableShowSourceCode**.

☑ If you are using a PHP framework, refer to the dedicated page.

☑ Finally, do not hesitate to post your questions or problems in our forum; solutions are generally available.

See also: How Compiled PHP Applications Work

2.5 How Compiled PHP GUI Applications Work

GUI applications created with ExeOutput for PHP are feature-rich, combining technologies such as PHP, JavaScript, the Chromium rendering (CEF3) engine, HEScript, and the Windows API.

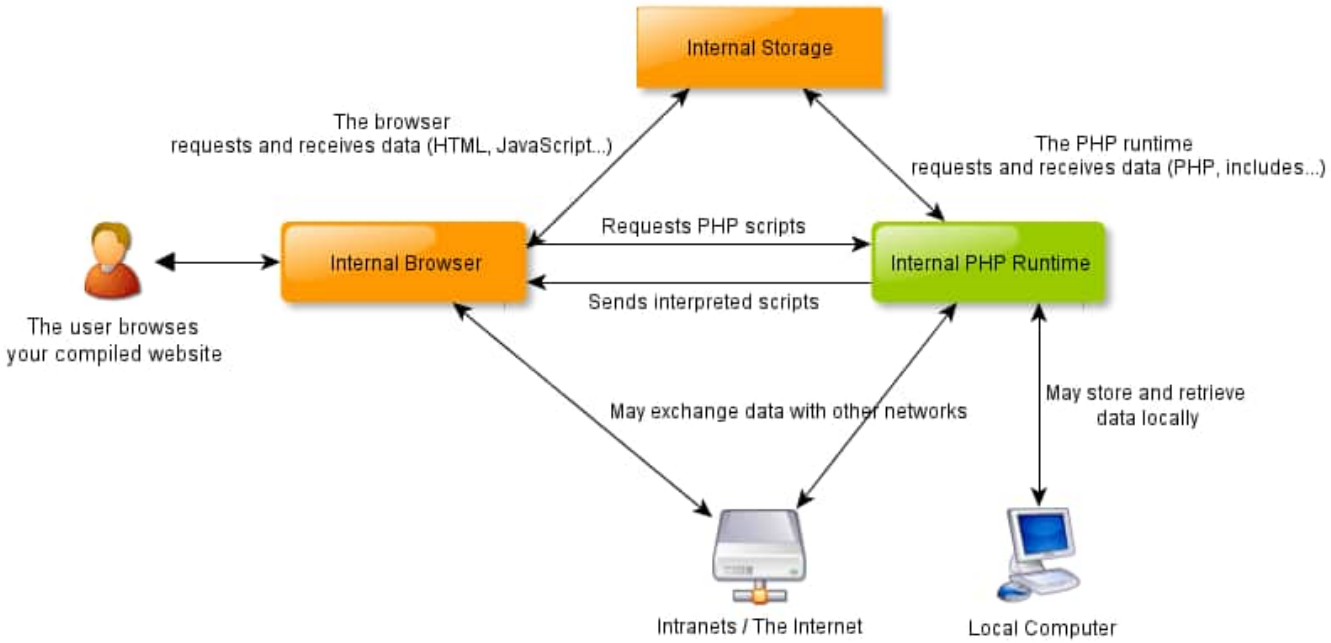
2.5.1 Applications with Custom Interfaces

☑ ExeOutput for PHP **generates a single executable file**; users simply double-click it to launch the application. A **main window that mimics a web browser** appears, displaying your PHP pages. Users navigate PHP pages as if they were hosted on a remote server.

Additionally, you can design your own user interface style for your compiled application, whether it's a browser-style interface or a modern Windows application. This flexibility is made possible by the integrated user interface designer.

2.5.2 Application Components

Your compiled application consists of three main components: the **internal browser**, **internal storage**, and the **PHP engine**.



Internal Browser

The **internal browser** allows users to navigate through your compiled website. Nearly every aspect of the browser (interface, navigation menus, features) can be customized. The **HTML rendering engine** is Chromium, utilizing Chromium Embedded Framework version 3. You can customize the rendering engine here.

Any HTML page compatible with Google Chrome should render successfully in the internal browser while maintaining full functionality. However, the key difference from Chrome is that **your website's source HTML and PHP files are never directly accessible to the user**. As illustrated in the flowchart above, users have no access to the internal storage. Moreover, several options are available to **protect your HTML source code**. For example, the context menu with the "Show Source Code" command is disabled by default, without relying on any JavaScript workarounds.

> **Note**
>
> Console (CLI) applications do not include an internal browser; PHP operates in CLI mode.

Internal Storage

The **internal storage** holds source files that have been compressed and encoded by ExeOutput for PHP. It also manages a virtual storage in memory where files can be unpacked as needed. The internal storage is only accessible by the internal browser and the PHP engine.

PHP Runtime

The **PHP runtime** is invoked by the browser each time a PHP script needs to be executed. The PHP runtime interprets the PHP code and typically returns HTML data that is displayed within the browser. It retrieves all PHP files and other necessary files from the internal storage.

All **standard PHP functions are available**, and you can even access remote web servers (such as database servers) or **local files**. ExeOutput for PHP provides several functions to load and save local files:

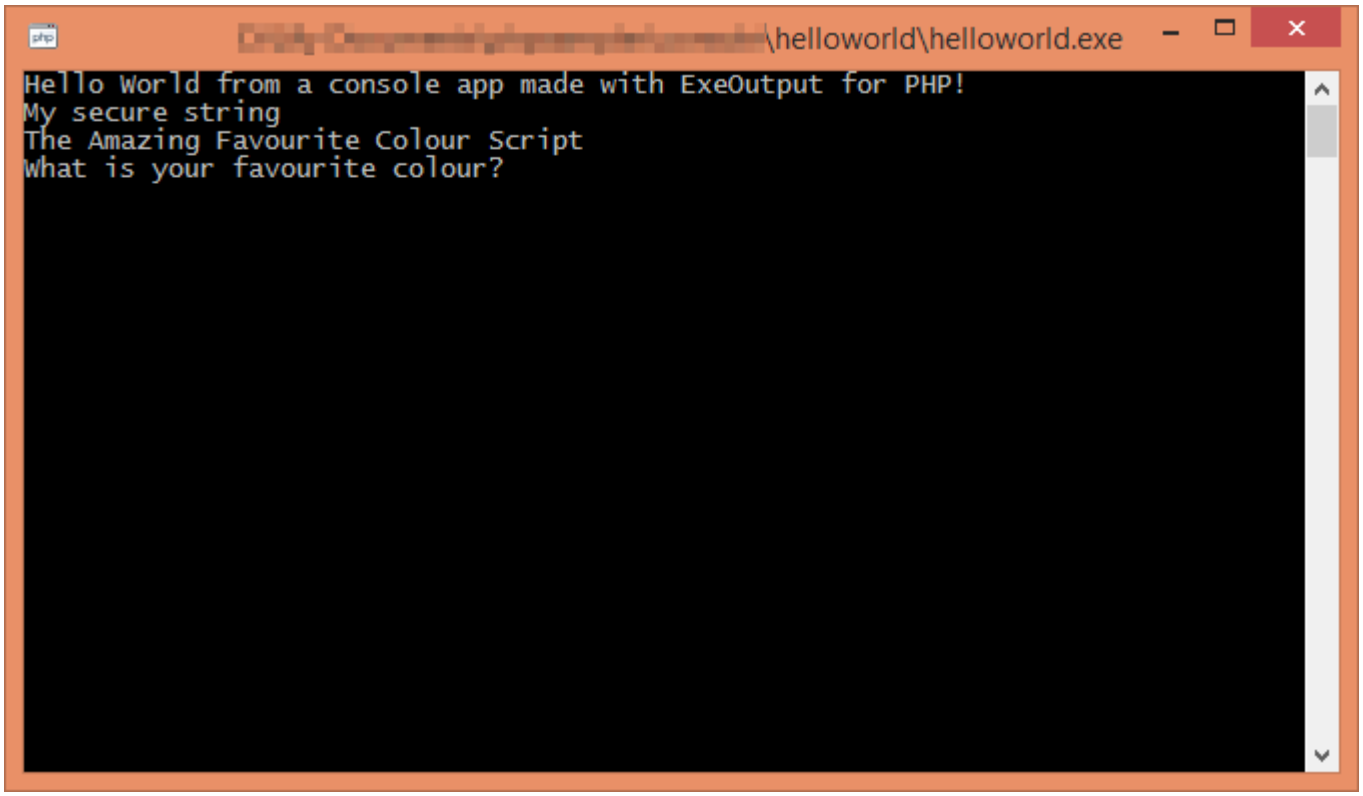
👉 Accessing Files in Compiled PHP Applications

👉 How to Create and Save Files with PHP

👉 Advice for Getting Started with PHP Applications in ExeOutput for PHP

2.6 How Compiled PHP Console Applications (CLI) Work

ExeOutput for PHP can create **console applications without a graphical user interface (GUI)**. These applications operate as if PHP were executed in CLI mode.



You decide whether to create a GUI or a console application when starting a new project. **This choice is final.** The ExeOutput for PHP status bar indicates whether you are working on a GUI or a console application:

👉 ExeOutput for PHP **creates a single, stand-alone executable file**. Users simply launch the resulting EXE file. **No PHP installation is necessary**. All compiled files are automatically loaded into the PHP runtime upon request.

👉 By default, the index page script runs at startup. You can specify a different command line on the PHP Settings => Main Settings page.

> ⚠ **Warning**
>
> Console applications cannot prompt for elevated rights. Use an administrator command prompt to perform administrative tasks.

👉 The `exo_get_resstring` function can be used to protect resource strings.

> **Warning**
>
> Many ExeOutput for PHP features are **not supported by console applications**, including global variables, global protection (security), and HEScript scripting.

2.6.1 Accessing the Folder Containing the .EXE File

Do not use `$_SERVER['DOCUMENT_ROOT']` in a console application, as the `$_SERVER` variable is empty in this context (a console app does not run on a server or handle HTTP requests). Instead, you can retrieve the full path to the directory containing the EXE file by using the `EXOPHPEXEPATH` environment variable, as shown below:

```
<?php
$exoexepath = getenv('EXOPHPEXEPATH');
print($exoexepath);
?>
```

This prints the full path of the EXE's directory. For example:

```
<?php
print(getenv('EXOPHPEXEPATH'));
?>
```

Additionally, you can access the virtual "Data" subfolder using the `EXOPHPDATAPATH` environment variable. For instance:

```
<?php
print(getenv('EXOPHPDATAPATH'));
?>
```

2.6.2 Script Example

The following code was used to generate the console application screenshot shown above.

```
<?php
print("Hello World from a console app made with ExeOutput for PHP!\n");
print(exo_get_protstring('str1'));
fputs(STDOUT, "\nThe Amazing Favorite Color Script\n");
fputs(STDOUT, "What is your favorite color? ");
$sometext = strtolower(trim(fgets(STDIN, 256)));
fputs(STDOUT, "Your favorite color is $sometext!\n\n");
?>
```

2.7 Visual C++ Redistributable Requirement

2.7.1 Troubleshooting: Visual C++ Redistributable Requirement

When running a PHP application compiled with ExeOutput for PHP, you or your users might encounter an error message indicating that a specific version of the Visual C++ Redistributable is missing.

This error occurs because the PHP runtime and the CEF engine rely on the Visual C++ Redistributable to function correctly. Without the appropriate runtime, your application cannot execute.

Required Versions

The required version of the Visual C++ Redistributable depends on the components you use in your application.

- **For applications using CEF version 139 or later:** A minimum of **Visual C++ 2022 Redistributable** is now required. Higher versions are backward compatible.
- **For older versions or other components:** The **Visual C++ Redistributable for Visual Studio 2015-2019** might be required.

ExeOutput for PHP 2025 and later use CEF 139+.

Download Links

You can find the necessary installers in the `Redist` subfolder of the ExeOutput for PHP installation directory. The installers are also available for download from the official Microsoft website:

- Download the latest supported Visual C++ Redistributable

Ensure that you download the **x86** version for 32-bit applications created with ExeOutput for PHP.

Including the Runtime in Your Installer

To provide a seamless installation experience for your users, you can bundle the Visual C++ Redistributable runtime with your application's installer (for example, one created with **Paquet Builder**). This ensures the necessary runtime components are installed automatically with your application, eliminating the need for users to install them manually.

For detailed instructions, please refer to our Creating an Installer with Paquet Builder guide.

2.8 Application Command Line Switches

GUI applications built with ExeOutput for PHP support several **command line arguments** (or switches). These allow you to control the application at launch, for example, to display a specific page when calling it from another program or from the Windows Run dialog.

> **GUI Applications Only**
>
> These switches apply only to GUI applications. For console applications, use PHP's standard `$argc` and `$argv` variables to access command line arguments.

Command line switches are especially powerful when combined with the **"Allow only one instance of the application"** option (available in `Security` > `Global Options`). For example, if a single-instance application is already running, launching it again with a new `page` argument will not start a second instance. Instead, the existing instance will be brought to the foreground and will navigate to the new page specified in the command line.

2.8.1 Built-in Switches

ExeOutput for PHP provides two built-in switches.

`page`

Specifies the virtual path to the page that should be displayed when the application starts.

Syntax: `MyApp.exe page [path_to_page]`

Examples: `MyApplication.exe page products.php` `MyApplication.exe page "customer data/profile.html"`

If the path contains spaces, you must enclose it in double quotes (`"`).

`ignoreuserpos`

Forces the application to open centered on the primary monitor, ignoring any previously saved window position. This is useful in situations where an application might launch off-screen. For instance, if a user moves the application to a secondary monitor that is later disconnected, the OS may try to launch the window in that "lost" position.

Syntax: `MyApp.exe ignoreuserpos`

2.8.2 Creating Custom Switches

You can create your own custom command line switches by reading and parsing the launch parameters with HEScript. This is accomplished using the `ParamStr` and `ParamCount` HEScript functions, which work similarly to their Delphi/Pascal counterparts.

3. Working With PHP

3.1 Working with PHP

ExeOutput for PHP™ integrates the entire PHP runtime into your application, allowing for complete customization.

3.1.1 About PHP Implementation in ExeOutput for PHP

Since version 2, ExeOutput for PHP has utilized PHP CGI as an external process. This approach offers several benefits:

- **Enhanced Stability and Responsiveness:** Isolating PHP from the UI process makes your application more stable and responsive.
- **Automatic Recovery:** If the PHP process crashes, it automatically restarts when the webpage is refreshed, ensuring your application continues to run smoothly.
- **Flexible PHP Version Support:** ExeOutput for PHP supports a wide range of PHP versions, from PHP 5.6 to the latest 8.x releases. You can select your preferred PHP version.
- **Seamless Chromium Integration:** Enjoy full integration between PHP and the Chromium rendering engine, with support for AJAX, cookies, file upload/download dialogs, custom HTTP request and response headers, and Developer Tools.

> **Note**
>
> For console applications, the PHP CLI is used instead of PHP CGI.

3.1.2 Including PHP Extensions in Your Application

If your PHP application requires **extensions**, you must first configure them on the PHP Extensions page.

> **Tip**
>
> ExeOutput for PHP can embed PHP extensions directly into the EXE file, eliminating the need for manual deployment.

3.1.3 Where Are PHP Files Located at Runtime?

ExeOutput for PHP uses virtualization: all PHP files are stored in memory (and can be optionally encoded to prevent memory dumps), while the PHP runtime is led to believe they are located on the hard disk.

Essentially, your PHP application uses a `Data` subfolder to store its PHP files. By default, this "Data" virtual subfolder is in the same directory as the EXE file. For example, if your EXE is at `E:\my folder\myprogram.exe`, the path to the `Data` subfolder will be `E:\my folder\data`. You can even specify a custom virtual folder that does not physically exist on the disk.

You can also place physical files into this `Data` subfolder. For instance, if you choose to keep some files outside your EXE (as External Files), your PHP application can still access them.

If the EXE is in a writable directory (such as **My Documents** or the user's local **AppData** folder), you can write files to the `Data` subfolder.

To retrieve the path to the `Data` subfolder, use `$_SERVER['DOCUMENT_ROOT']`.

For example, you can use the following code to read a file in the `Data` subfolder:

```
<?php
$cont = file_get_contents($_SERVER['DOCUMENT_ROOT'] . '\\include1.txt', FILE_USE_INCLUDE_PATH);
print($cont);
?>
```

3.1.4 Custom PHP.INI

You can adjust PHP parameters directly with the built-in PHP.INI editor.

> **Warning**
>
> Do not place a custom `php.ini` file in the root folder; use the ExeOutput for PHP editor instead.

Advice for Getting Started with PHP Applications in ExeOutput for PHP

How Compiled PHP Applications Work

3.2 Choose a PHP Version

ExeOutput for PHP supports various PHP versions, from PHP 5.6 to the latest 8.x releases. You can choose which PHP version your application will use directly in the ExeOutput for PHP interface.

> **Note**
>
> If you want PHP 5.x, download and install ExeOutput for PHP 2.

3.2.1 Choosing a PHP Version

In ExeOutput for PHP, go to **PHP Settings** => **Main Settings** and use the combo box to select your desired PHP version.

3.2.2 Available PHP Runtimes

ExeOutput for PHP 2025 includes the following PHP runtimes: * PHP 8.5 (beta 2) * PHP 8.4.12 * PHP 8.3.25 * PHP 8.2.29 * PHP 8.1.33

Older versions are also available.

3.2.3 Application System Requirements

Applications built with ExeOutput for PHP are **truly stand-alone**. They do not require any additional components, as the necessary Microsoft Visual C++ Libraries for PHP are built-in.

Windows 7 or higher is required. If you need your app to run on Windows XP or Vista, download and install ExeOutput for PHP 2, which creates compatible applications.

> **Warning**
>
> Recent PHP versions, such as 8.2, 8.3, 8.4, and 8.5, require Windows 10 or higher. If you choose one of these versions, ExeOutput for PHP will configure your application to require a minimum of Windows 10.

See also: Working with PHP

3.3 Accessing Files in Compiled PHP Applications

Applications created with ExeOutput for PHP are **not designed to function like a web server**. You may need to adjust your PHP code to **create or access files** on the user's computer, or files compiled into your application's EXE file, which are listed in the File Manager.

> **Tip**
>
> 👉 Please refer to the Accessing Files topic of the General Demonstration for **live demonstrations and further explanations** about working with local files from your ExeOutput for PHP apps.

3.3.1 Accessing Local Files

By default, PHP scripts in your application can **access any local file** on the user's computer. Absolute paths should be used. See below for how to access files shipped with your application or compiled into it.

3.3.2 Accessing Source Files from the Internal Browser

Your application functions as if it were server software, serving webpages and related files through the HTTP protocol. When the application runs, it creates a custom pluggable protocol (similar to HTTP, FTP, etc.) to communicate with the Chromium rendering engine. In other words, the application works like a small server combined with a client (the main window that allows users to navigate HTML and PHP pages). Your website is then available as if it were on a server, but without requiring an internet connection or a physical server.

> **Tip**
>
> The internal browser can read files from the storage location only if URLs begin with the base URL followed by the virtual path to the file.

The base URL depends on the selected rendering engine:

- For the **CEF engine**, the base URL is `https://heserver/`.
- For the **WebView2 engine**, the base URL is `https://heserver.example/` to ensure proper cookie handling.

For instance, to access a page named `index.html` with the CEF engine, you would use the URL `https://heserver/index.html`.

When your application starts, the browser automatically navigates to your index page.

3.3.3 Accessing External Files from the Internal Browser

Your application can automatically load **external resource files** (not compiled into the EXE). For instance, you can **keep image and media files outside the EXE file** (in the same folder or a subfolder).

Examples:

- If an HTML file references `image1.png`, the application will look for the `image1.png` file in its compiled data. If not found, it will try to locate and load it from the same folder as the EXE file (depending on the URI).
- If you have an image file in a subfolder, e.g., `<img src="myfolder/my image.png" />`, the application will expect the "my image.png" file to be in a subfolder named "myfolder" (if you leave the file external).

> **Note**
>
> External files must be deployed with the application's EXE file in their respective folders.

3.3.4 Accessing Source Files from PHP

ExeOutput for PHP leads the PHP runtime to believe that PHP scripts and other compiled files are on the hard disk in a `Data` subfolder, when they are actually held in virtual memory.

For instance, if the path to your EXE file is `E:\my folder\myprogram.exe`, the path to the "Data" subfolder will be `E:\my folder\data`.

You can think of the `Data` subfolder as playing the same role as the `www` folder in Apache.

Tip

The `DOCUMENT_ROOT` server variable contains the full path to the virtual `Data` folder.

Info

You can also keep PHP files outside the EXE and place them directly into a physical `Data` subfolder (see below).

For security reasons, you may want to choose a custom name for the virtual `Data` folder. The ExeOutput for PHP virtualization engine allows you to choose any virtual folder, even one on a non-existent drive. To activate this feature, go to "PHP Settings" => "Main Settings" and enable "**Use an absolute path for the virtual "Data" subfolder**". Then, enter the path of your choice, for instance, `X:\Data`.

3.3.5 Troubleshooting "file missing" Errors

Generally, ExeOutput for PHP intercepts all files requested by the PHP runtime and makes them available. For instance, ExeOutput for PHP supports `include`, `require`, `include_once`, and `require_once`.

Warning

PHP's `fopen` function cannot download and open files from URLs beginning with `http://heserver/`, as no real web server is used.

If PHP displays "missing file" errors or warnings such as "**php failed to open stream**", you may need to mark files in the File Manager (select your files, click File Properties, and turn on the **Unpack the file(s) to virtual memory at startup** option) or use the ExeOutput-specific PHP function named `exeoutput_unpackvirtualfile`.

```
string exo_unpackvirtualfile ( string $sourcepath , string $optionaldestpath )
```

- `$sourcepath` is the virtual source path to the compiled file you want to unpack to memory.
- `$optionaldestpath` is the absolute path to the destination virtual file. If you leave it blank, the path is constructed from `$_SERVER['DOCUMENT_ROOT']` and the virtual path.

The function returns the absolute path to the virtual file in UTF-8 format. This path can be used with PHP functions like `fopen`, `file_exists`, and `file_get_contents`.

Example: The following script makes `demo1.pdf` available and displays its full (virtual) path and size:

```
<?php
$filename = exo_unpackvirtualfile('res\demo1.pdf', '');
echo $filename . ': ' . filesize($filename) . ' bytes';
?>
```

3.3.6 Accessing External Files from PHP Scripts

You can create a physical `Data` subfolder to place any external or resource files that your application or PHP scripts require, such as database files, XML, etc. Even PHP files can be placed in this folder and used in conjunction with the PHP scripts compiled into your application.

We recommend using the "External Files" feature of ExeOutput for PHP to manage external files automatically.

Important

External files must be deployed with the application's EXE file in the `Data` subfolder.

Warning

If the "**Use an absolute path for the virtual "Data" subfolder**" option ("PHP Settings" => "Main Settings") is enabled, you will not be able to access real files that you may have placed in a physical `Data` subfolder.

For instance, if you have a file named `include1.txt` in the `Data` folder, the following script will display its contents:

```
<?php
$cont = file_get_contents($_SERVER['DOCUMENT_ROOT'].'\\include1.txt', FILE_USE_INCLUDE_PATH);
print($cont);
?>
```

3.3.7 About is_file and is_dir

The standard PHP functions `is_file` and `is_dir` have been modified to handle virtual files and folders. They will return `true` if you check for the existence of a virtual file (even if it is not yet in memory) or a virtual folder.

How to create and save files with PHP

How compiled PHP applications work

3.4 Saving Files with PHP in Desktop Applications

When run as a desktop application, PHP code **can access local files** on the user's computer, making it possible to save and modify files locally. To learn how to load compiled files, please see this page.

> **Tip**
>
> 👉 Please refer to the Saving Files topic of the General Demonstration for **live demonstrations and further explanations** about saving local files with your ExeOutput for PHP apps.

3.4.1 Saving and Storing Files Locally

PHP provides several ways to load and save files. To save files correctly, you must ensure that the user has **write permissions** for the desired location.

For instance, a portable application is often started from a USB stick, and the folder containing the EXE has write permissions. In this case, your application will be able to save files in that folder. Conversely, if your application is installed in the `Program Files` directory, it is generally not allowed to save files in its own folder due to the Windows User Account Control (UAC) feature.

To solve this, ExeOutput for PHP provides a **dedicated storage folder for your application**. The absolute path to this folder can be retrieved with the following PHP command:

```php
<?php
$storagelocation = exo_getglobalvariable('HEPubStorageLocation', '');
echo $storagelocation;
?>
```

`HEPubStorageLocation` is a global variable defined by ExeOutput for PHP at runtime. The `exo_getglobalvariable` function is a built-in PHP command provided by ExeOutput for PHP.

> **Tip**
>
> This folder is always available and can be used to store your application's settings. You can customize the name of this storage folder on the **Output** -> **Output Settings** page.

3.4.2 PHP Code: Saving a File with fopen / fwrite

In compiled applications, you must pass absolute paths to the `fopen` PHP function.

```php
<?php
$storagelocation = exo_getglobalvariable('HEPubStorageLocation', '');
$file = $storagelocation . 'myfile1.txt';
$fp = fopen($file, "w") or die("Couldn't open $file for writing!");
fwrite($fp, $data) or die("Couldn't write values to file!");
fclose($fp);
echo "Saved to $file successfully!";
?>
```

> **Tip**
>
> If you want to let users choose the save path themselves, you can display a "Save As" dialog box.

👉 Accessing Files in Compiled PHP Applications

👉 How compiled PHP applications work

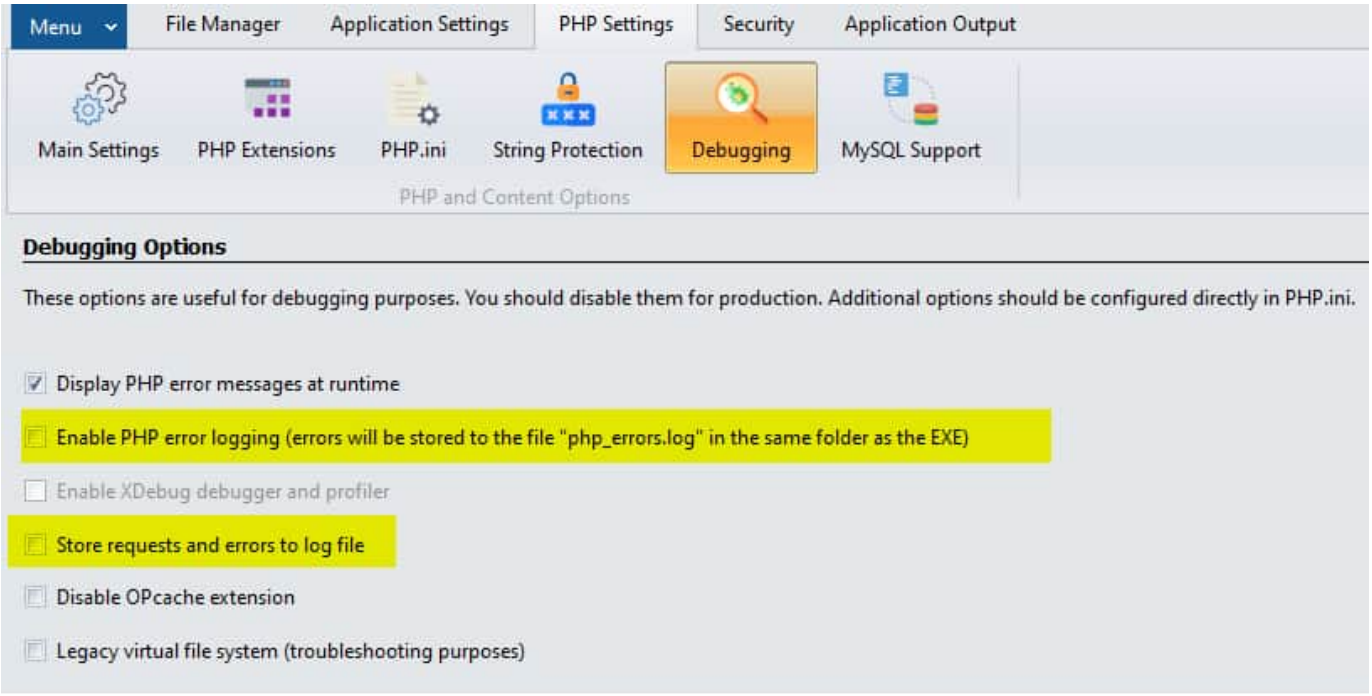
👉 Working with PHP

3.5 Solving PHP Errors

If your PHP scripts contain errors, your application compiled with ExeOutput for PHP can display them.

ExeOutput for PHP provides several options to help you debug your app and resolve potential errors. These are accessible on the PHP Debugging page.

First, we strongly recommend activating these two options when you start a project:



Enable PHP Error Logging

This option automatically configures the error reporting settings in `PHP.INI` so that any PHP error is stored in a file named `php_errors.log`, located in the same folder as the application's EXE file. For instance, if your EXE path is `C:\My Documents\Output\myprogram.exe`, the full path to the log file will be `C:\My Documents\Output\php_errors.log`.

> **Note**
>
> The `error_reporting` level can be configured in the `php.ini` file.

Store Requests and Errors in Log File

When this option is enabled, the application writes all accessed URLs and any errors (such as PHP errors, warnings, or 404 "resource not found" errors) to a log file. This is similar to web server logs.

This **log file is also stored in the same folder** as the application's EXE file.

> **Warning**
>
> ExeOutput for PHP does not automatically delete log files.

See Also: Other Available Debugging Options

3.6 Using the Save As Dialog Box in PHP Applications

Since your ExeOutput for PHP application runs on Windows, you may want to save files locally. In some cases, this involves asking the user for a path and filename.

The "Save As" dialog box lets the user specify the drive, directory, and name for the file to be saved.

It can be invoked with the `SaveFileDialog` HEScript command. This topic explains the steps to implement a "Save As" dialog box in your PHP application.

3.6.1 Step 1: Create the HEScript

1. Open the UserMain script in ExeOutput for PHP.
2. Paste the following code into it:

```
function SaveDlgFile: String;
begin
  // Ask for filename.
  Result := SaveFileDialog("Save Text As", "*.txt", "*.txt", "Text Files (*.txt)|*.txt|All files (*.*)|*.*", "");
  // Returns a blank string if the user cancels.
end;
```

This code calls the built-in `SaveFileDialog` HEScript command, which is described in the Script Reference topic.

```
function SaveFileDialog(const aTitle, aFilename, aDefaultExt, aFilter, aInitialDir: String): String;
```

- **aTitle**: Title of the dialog box.
- **aFilename**: Default filename.
- **aDefaultExt**: Default extension.
- **aFilter**: File extension filter.
- **aInitialDir**: Initial directory.

Result: Returns the full path to the selected file, or an empty string if the dialog is canceled.

1. Click **Save Script**. The HEScript function is now ready to be called from PHP or JavaScript.

3.6.2 Step 2: Call from PHP

```
<?php
echo "Asking for filename...<br>";

// Executes HEScript to call the system "Save As" dialog...
$filename = exo_return_hescriptcom("UserMain.SaveDlgFile", "Error");

echo "Filename: " . $filename;

if (empty($filename)) {
  echo "Operation canceled.<br>";
  return;
}

// Example of saving a PDF file from a library like FPDF.
$pdf->Output($filename, "F");

echo "Done.";
?>
```

> **Warning**
>
> Virtual files (in the `Data` subfolder) will **not** appear in the "Save As" dialog box unless you activate the Do not hide virtual files in open/save dialog boxes security option.

- How to Create and Save Files with PHP
- Selecting Files with PHP in Local Applications (Upload Replacement)
- Accessing Files in Compiled PHP Applications

3.7 Selecting Local Files with PHP (File Upload Replacement)

Your ExeOutput for PHP application may need to **work with local files**. Since you can access local files directly with PHP, traditional file upload functions are often unnecessary. Your application is **not designed to work like a web server**; users do not have to upload local files to a remote server for your application's PHP code to process them.

This topic shows you **how to prompt a user to select a file and then process it with PHP**.

> **Tip**
>
> Of course, ExeOutput for PHP also supports traditional file uploads. You can see the General Demonstration for **live demonstrations and further explanations** about uploading files with your ExeOutput for PHP apps.

3.7.1 Prompting the User with an "Open File" Dialog Box

On the web, when a user needs to upload a file, the browser prompts them to select it with an "Open" dialog box. You can achieve the same result in an ExeOutput for PHP application.

The **Open** dialog box lets the user specify the drive, directory, and name of a file (or files) to open.

It can be invoked with the `OpenFileDialog` HEScript command.

3.7.2 Step 1: Create the HEScript

1. Open the UserMain script in ExeOutput for PHP.
2. Paste the following code into it:

```
function OpenDlgFile: String;
begin
  Result := OpenFileDialog("Select a File to open", "*.*", "*.*", "All files (*.*)|*.*", "");
end;
```

This code calls the built-in `OpenFileDialog` HEScript command, which is described in the Script Reference topic.

```
function OpenFileDialog(const aTitle, aFilename, aDefaultExt, aFilter, aInitialDir: String): String;
```

- **aTitle**: Title of the dialog box.
- **aFilename**: Default filename.
- **aDefaultExt**: Default extension.
- **aFilter**: File extension filter.
- **aInitialDir**: Initial directory.

Result: Returns the full path to the selected file, or an empty string if the dialog is canceled.

1. Click **Save Script**. The HEScript function is now ready to be called from PHP or JavaScript.

3.7.3 Step 2: Call from PHP

```
<?php
echo "Asking for filename...<br>";
// Executes HEScript to call the system "Open" dialog...
$filename = exo_return_hescriptcom("UserMain.OpenDlgFile", "Error");

// The $filename variable contains the full path to the file selected by the user.
// It can then be passed to functions like `fopen`.
if (empty($filename)) {
  echo "You have not selected a file. Operation canceled.<br>";
  return;
}
```

```
// Example: load the selected file into a PDF library.
$pdf->load($filename);
?>
```

Warning

Virtual files (in the `Data` subfolder) will **not** appear in the "Open" dialog box unless you activate the Do not hide virtual files in open/save dialog boxes security option.

Accessing Files in Compiled PHP Applications

How Compiled PHP Applications Work

3.8 Built-In ExeOutput for PHP Functions

ExeOutput for PHP provides several built-in PHP functions that can be called from your PHP scripts.

All strings used and returned by these PHP functions are in **UTF-8 format**.

3.8.1 Available PHP Functions

exo_unpackvirtualfile

```
string exo_unpackvirtualfile ( string $sourcepath , string $optionaldestpath )
```

Unpacks a compiled file from internal storage to a specified virtual folder. Note that the file is unpacked into memory, not to the hard disk.

- `$sourcepath` is the virtual source path to the compiled file you want to unpack to memory.
- `$optionaldestpath` is the absolute path to the destination virtual file. If left blank, the path is constructed from `$_SERVER['DOCUMENT_ROOT']` and the virtual path.

The function returns the absolute path to the virtual file in UTF-8 format. This path can be used with PHP functions such as `fopen`, `file_exists`, and `file_get_contents`. See more about accessing compiled files.

> **Info**
>
> As an alternative, you can also use the "Unpack the file(s) to virtual memory at startup" option.

exo_removevirtualfile

```
void exo_removevirtualfile(string $destpath);
```

Removes the specified file from internal storage.

- `$destpath` must contain the full path to the file.

exo_getglobalvariable

```
string exo_getglobalvariable (string $name, string $default);
```

Returns the value of the global variable specified by `$name`. If the global variable does not exist, the `$default` value is returned. This is similar to the GetGlobalVar HEScript function.

exo_setglobalvariable

```
void exo_setglobalvariable (string $name, string $value, bool $isstored);
```

Sets the value of a global variable specified by `$name`. If `$isstored` is `true`, the global variable is made persistent, meaning its value will be stored and restored the next time the application runs. This is similar to the SetGlobalVar HEScript function.

exo_runhescriptcom

```
void exo_runhescriptcom(string $comline);
```

This is a commonly used function that lets you execute an HEScript function or procedure. It is similar to the `hescript:` protocol, except that you specify the command line via the `$comline` parameter.

Syntax for `$comline`: `[HEScript script name].[procedure/function name]|Param1|Param2|....|ParamN`

Please see the "How to Call a Script" help topic.

exo_return_hescriptcom

```
string exo_return_hescriptcom(string $comline, string $defvalue)
```

This function is similar to the previous one, except it calls an HEScript function that returns a string, and it returns that function's result. It is similar to the `hescript:` protocol, except that you specify the command line via `$comline` and a default value in `$defvalue` to be used if the function is not found.

Syntax for `$comline`: `[HEScript script name].[procedure/function name]|Param1|Param2|....|ParamN.`

Please see the "How to Call a Script" help topic for an example.

exo_get_resstring

```
string exo_get_resstring(string $ID)
```

Returns the resource string specified by `$ID`.

3.8.2 Example

Calling an HEScript function and returning its result:

```
<?php
echo exo_return_hescriptcom("UserMain.ReturnDate", "Error");
?>
```

Introduction to Scripting

How to Call HEScript Procedures/Functions

3.9 Global Variables

Global variables can be used by HEScript, PHP, and JavaScript to **share data** and **store values** within the application. Each HEScript script runs in a separate engine, so they cannot share variables directly. To share data between two scripts, for instance, you must use global variables.

You can manage global variables with the following functions:

- `SetGlobalVar` and `GetGlobalVar` in HEScript.
- `SetGlobalVariable` and `GetGlobalVariable` in JavaScript.
- `exo_setglobalvariable` and `exo_getglobalvariable` in PHP.

> **Info**
>
> Global variables can be used as an alternative to cookies in your JavaScript or PHP code. See the cookies topic for more information.

3.9.1 Properties

- **Global variable names** must be unique and contain only alphanumeric characters (no spaces). As a general rule, your global variable names should not begin with "HE", as these are reserved for internal use by ExeOutput for PHP.
- Global variables may be **persistent or temporary**. A persistent variable is stored in the application's state file, meaning its value is saved and then loaded the next time the application runs. A temporary variable exists only for the current session. You determine whether a variable is persistent when you set its value with the `SetGlobalVar` method.
- Global variables are not shared between multiple running instances of an application. However, you can force all instances to have the same global variable values by calling the `SynchronizeGlobalVar` HEScript function.

3.9.2 Predefined Global Variables

> **Warning**
>
> Not all global variables are available in console applications; only `HEPublicationFile` and `HEPublicationPath` are used. It is also not possible to change global variable values in console applications.

Global Variable Name	Description
`HEPublicationFile`	The full path to the application's .exe file (including filename).
`HEPublicationPath`	The full path to the folder that contains the application's .exe file (no filename). It will always include a trailing backslash (e.g., `C:\MyPath\`).
`HELasterrormessage`	When an error occurs, this variable contains the error message.
`DefWinTitle`	The title of the main window.
`HomePage`	The index page's filename.
`HEPubStorageLocation`	The path where the application stores its data. You can customize it.
`HEPubTempPath`	The path to a temporary location where the application stores its external files. This path changes each time the application is run.
`HEPHPDataPath`	The path to the virtual cache folder used for PHP files (includes a trailing backslash).
`HEPublicationDiskInfo`	Contains the device ID (returned by Windows) of the USB disk on which the application EXE lies.

Global Variable Name	Description
`CurPageTitle`	The title of the page which is currently displayed.
`FwdButtonEnabled`	Indicates whether the forward button is enabled (`true` / `false`).
`BackButtonEnabled`	Indicates whether the back button is enabled (`true` / `false`).
`HEMyDocDirectory`	Returns the path to the current user's "My Documents" folder. Does **not** include a trailing backslash.
`HEStartCurrentDirectory`	The working directory when the EXE was launched. Read this global variable instead of calling `getcwd()` in PHP.
`HEPublicationOnUSB`	A boolean (`0` or `1`) that indicates whether the application is on a USB drive. To use this, `GetManualHardwareID(1)` must be invoked first.
`isappterminated`	Only accessible from PHP. This is set to `1` when the application is terminated. It is useful for breaking infinite loops in PHP.
`hesuppressjsdlg`	A boolean (`0` or `1`) that lets you disable all dialog boxes displayed by JavaScript functions such as `alert()` .

3.9.3 PHP Example: Reading a Global Variable

```
<?php
$storagelocation = exo_getglobalvariable('HEPubStorageLocation', '');
$file = $storagelocation . 'myfile1.txt';
$fp = fopen($file, "w") or die("Couldn't open $file for writing!");
fclose($fp);
?>
```

3.9.4 HEScript Example

```
procedure OnStartMainWindow;
begin
  // When the main window is going to be displayed (just before the homepage is shown).
  SetGlobalVar("hesuppressjsdlg", "1", false);
end;
```

3.9.5 PHP Loop Example

For example, you can use this PHP code inside your loops to safely exit:

```
if (exo_getglobalvariable("isappterminated", "0") == "1") {
  break;
}
```

👉 JavaScript Sample with Global Variables

👉 How to Call HEScript Procedures/Functions

3.10 About PHP Sessions and Cookies

PHP sessions and cookies are supported in applications built with ExeOutput for PHP.

By default, PHP stores session files in the local user's temporary folder.

If you prefer, you can set **the session file path to be virtual** (i.e., in memory only, not written to the hard disk). To do this, go to the PHP.INI editor page in ExeOutput for PHP and change the `session.save_path` directive to:

```
session.save_path = "%EXOPHPVIRTUALTEMP%"
```

For example:

```
[Session]
; Handler used to store and retrieve data.
; http://php.net/session.save-handler
session.save_handler = files

; Argument passed to save_handler. For files, this is the path
; where data files are stored. Note: Windows users must change this
; variable to use PHP's session functions.
;
; The path can be defined as:
;
session.save_path = "%EXOPHPVIRTUALTEMP%"
```

Tip

👉 Please refer to the Session and Cookies topic of the General Demonstration for **demonstrations and more explanations** about integrating PHP sessions and HTTP cookies within your ExeOutput for PHP apps.

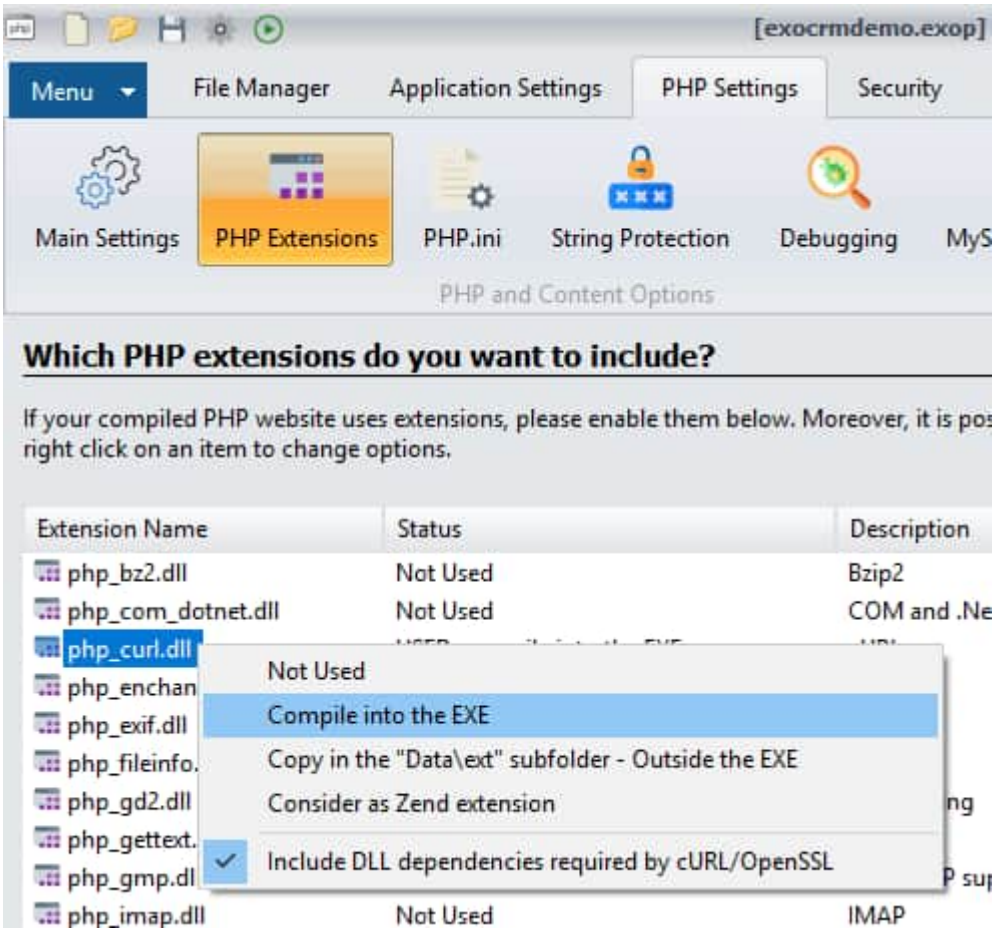
👉 See also: Global Variables

3.11 Using the cURL Extension

3.11.1 Activating cURL Support in Your Apps

ExeOutput for PHP allows you to access the cURL library that ships with PHP, enabling you to make requests to URLs using various protocols such as HTTP, HTTPS, FTP, and more.

To use cURL in your app, you must first enable the cURL PHP extension in ExeOutput for PHP. In ExeOutput for PHP, navigate to **PHP Settings** -> **PHP Extensions**. Select `php_curl.dll`, right-click it, and choose **Compile into the EXE** from the context menu.



After this, cURL functions will be accessible.

> **Tip**
>
> 👉 Please refer to the cURL topic of the General Demonstration for a **working cURL demo with HTTPS support**.

> **Note**
>
> The cURL extension requires additional dependencies (such as `libeay32.dll`, `libssh2.dll`, `ssleay32.dll`, and `nghttp2.dll`). ExeOutput for PHP automatically compiles these dependencies into the EXE if you choose "Compile into the EXE" for cURL. Conversely, if you choose "Copy...outside the EXE", these DLLs will be copied to the same folder as your EXE file.

3.11.2 Accessing Secure Websites with HTTPS

Without the correct configuration, attempts to access an HTTPS (SSL/TLS-protected) resource in PHP using cURL will likely fail.

The proper way to handle this is to include the `cacert.pem` file with your app. `cacert.pem` is a bundle of CA certificates that cURL uses to verify the identity of the remote server.

You can download an up-to-date `cacert.pem` file here.

Then, place the `cacert.pem` file in your project's Source folder (for instance, the root folder, which is "Application Root" in the File Manager).

Finally, provide cURL with the full path to the `cacert.pem` file by using the following PHP code:

```
<?php
// ...
curl_setopt($ch, CURLOPT_CAINFO, getcwd() . "\\cacert.pem");
// ...
?>
```

This code assumes that `cacert.pem` is in the same folder as the executing PHP script.

> **Tip**
>
> 👉 Please refer to the cURL topic of the General Demonstration for a **working cURL demo with HTTPS support**.

👉 See also: PHP Extensions

3.12 Using exec(), system() in Applications

PHP's `exec()`, `shell_exec()`, and `system()` commands are supported in applications built with ExeOutput for PHP.

The following PHP code illustrates how to use these commands to run external EXE files, BAT (batch) files, and more. By default, a console window is displayed.

☑ To run a batch file located in the same folder as the application's EXE file:

In this sample, the `exo_getglobalvariable` PHP function returns the path to the folder where the EXE is located.

```
<?php
$mypath = exo_getglobalvariable('HEPublicationPath', '') . 'sample.bat';
echo exec($mypath);
?>
```

For a console application, use the following:

```
<?php
system('cmd /c "'.dirname(__DIR__) . '\\test.bat"');
?>
```

☑ To run system commands:

```
<?php
echo system('echo | C:\\WINDOWS\\System32\\wbem\\wmic.exe path win32_computersystemproduct get uuid');
?>
```

☑ To run an EXE program in the background, you can use the following function:

```
<?php
function execInBackground($cmd) {
    pclose(popen("start /B " . $cmd, "r"));
}
?>
```

☑ To display the contents of the current folder:

```
<?php
echo system("dir");
?>
```

4. Databases

4.1 Using Databases in Applications

Applications compiled with ExeOutput for PHP can **work with databases** thanks to PHP and its extensions, such as MySQL, SQLite, MariaDB, PostgreSQL, etc.

Since MySQL is the most commonly used database software with PHP projects, ExeOutput for PHP can include a portable MySQL (MariaDB) server with your PHP application and start/stop it automatically when your application starts or stops.

Tip

👉 Please refer to the Databases topic of the General Demonstration for **demonstrations and more explanations** about integrating databases with your ExeOutput for PHP applications.

Warning

Database support requires the corresponding PHP extension to be enabled. Go to the **PHP Settings** -> **PHP Extensions** page in ExeOutput for PHP.

4.2 Using a Portable MySQL (MariaDB) Server

Many PHP projects rely on a MySQL or MariaDB database. ExeOutput for PHP allows you to bundle a portable MariaDB server directly with your compiled application. This server is considered "portable" because it runs as a standard process and does not require installation as a Windows service. This is similar to a WAMP stack, but ExeOutput for PHP's integrated browser and PHP runtime replace the need for a separate Apache web server.

Tip

Watch our video tutorial on configuring, using, and distributing the portable MySQL server with your app.

Separate Download Required

The portable MariaDB server is **not included in the main ExeOutput for PHP installer**. You must download it separately using the Web Update utility. In ExeOutput for PHP, click **Check for updates** to launch the utility, then select and install the **"MySQL package for ExeOutput for PHP"**.

After installation, the core server files are located in the `MySQL\Source\mysql` subfolder of your ExeOutput for PHP installation directory. You can update these files manually if needed.

4.2.1 Initial Server Setup

Before setting up the server, you must first define your project's Output Path.

ExeOutput for PHP will then copy the necessary server files into a `MySQL` subfolder within your specified output directory. During this setup, you will be prompted to configure the **root password** and the **connection port** (default is `3306`).

After you provide these details, the server files are copied, and the server is started temporarily to apply the configuration. Once configured, the server is shut down and is ready to be used by your application.

Distribution

The `MySQL` folder is required for the application to function and must be distributed alongside your `.exe` file. When creating an installer for your application (for example, with Paquet Builder), ensure you include this entire folder.

Changing the Password

Do not change the root password by manually editing configuration files. ExeOutput for PHP stores this password internally to manage the server shutdown process. To change the password, you must repeat the initial setup process.

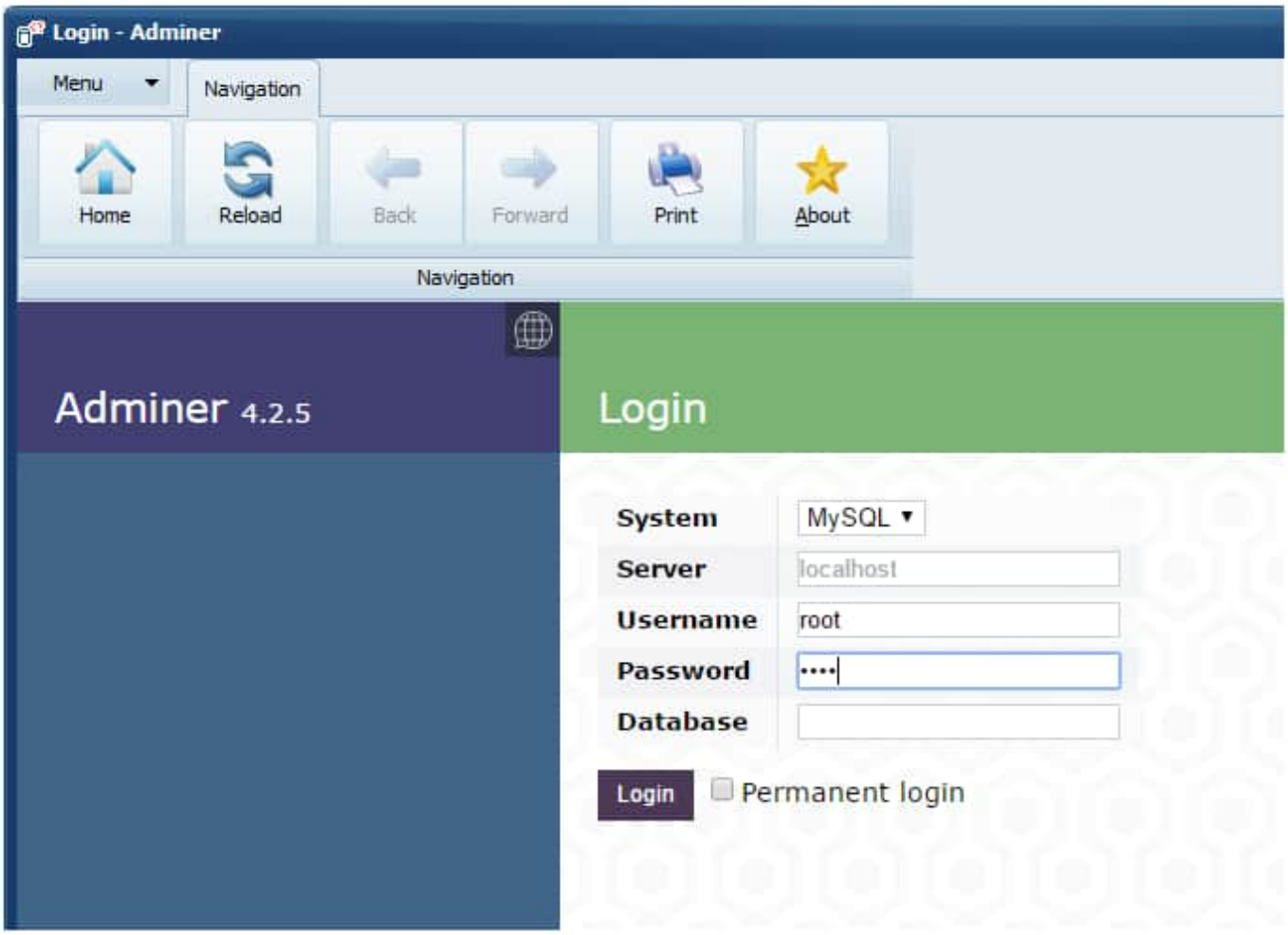
You can view the configured credentials and port at any time by clicking the **Show Configured Username and Password** and **Show Configured Port** links in the ExeOutput for PHP interface.

4.2.2 Database Management with AdminNeo

To manage your databases, you need a MySQL client. ExeOutput for PHP offers integration with AdminNeo, a popular database management tool.

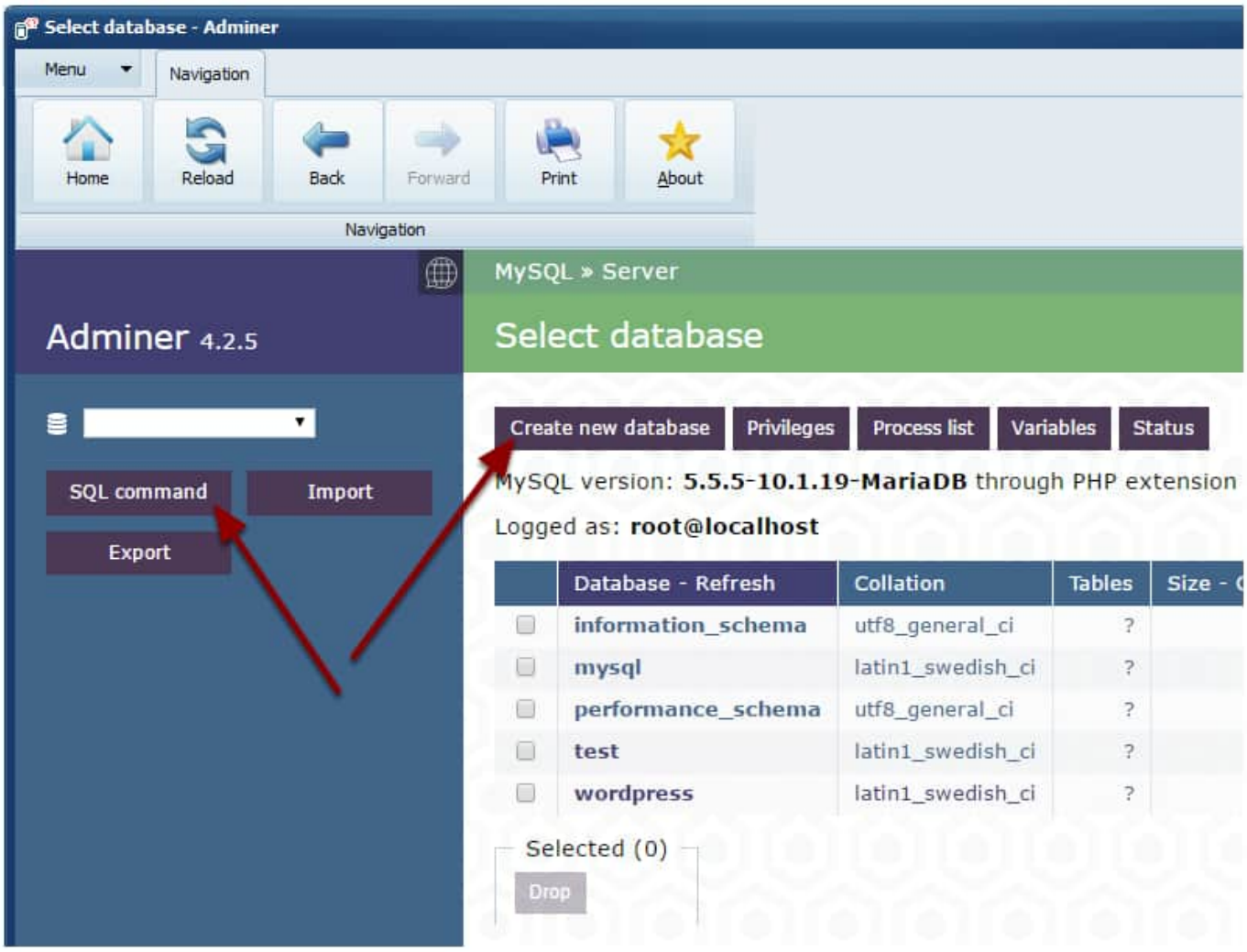
ExeOutput for PHP provides a pre-compiled, standalone version of AdminNeo called **AdminNeo EXE**. From the ExeOutput for PHP interface, you can launch AdminNeo EXE to manage your project's portable database. This action temporarily places AdminNeo EXE next to your application's output `.exe`, starts your portable MySQL server, and then runs AdminNeo. When you close AdminNeo, the MySQL server is automatically shut down.

Connecting with AdminNeo



- **Server**: `localhost`. If you used a custom port, enter it in the format `localhost:PORT` (e.g., `localhost:3307`).
- **Username**: `root`
- **Password**: The password you configured during the initial setup.

Once connected, you can create new databases, execute SQL commands, import data, and perform other management tasks.



AdminNeo EXE is a development tool and should not be distributed with your final application.

> **AdminNeo EXE Download**
>
> Like the MariaDB package, AdminNeo EXE must be downloaded separately via the Web Update utility.

4.2.3 Application Integration

To connect your PHP code to the database, follow these steps:

1. Enable the `MySQLi` PHP extension on the PHP Extensions page.
2. On the MySQL Server page, check the option **Start and stop the MySQL server automatically**. This ensures the database server runs alongside your application.
3. The MySQL server may take a few seconds to initialize. If your application tries to connect immediately at startup, it may fail. To prevent this, you can configure a startup delay using the **Delay to wait for the server to start (in seconds)** option. This forces your application to wait before displaying its main window, giving the server time to become ready.

If the server fails to start, a customizable error message is displayed. You can edit this message on the Localization page.

> **Single Instance**
>
> Only one instance of the portable server can run at a time. It is highly recommended to also limit your application to a single instance.

4.2.4 Removing the MySQL Server

To decouple the server from your application, uncheck **Start and stop the MySQL server automatically** in your project settings. You can then safely delete the `MySQL` subfolder from your application's output directory.

4.2.5 Viewing Server Logs

The MySQL server maintains a detailed log file, which is essential for troubleshooting. To view it, click the **Open server log** link in the ExeOutput for PHP interface.

4.2.6 WordPress Demo

To see a complex, real-world example, check out our demonstration project where we compiled the entire WordPress platform into a portable desktop application. The sample includes the full source code and a step-by-step tutorial.

4.3 How to Check MySQL Server Connection

To check if a MySQL server is connected in an ExeOutput for PHP application, you can use PHP's MySQLi extension.

First, enable the `MySQLi` PHP extension to allow your PHP application to interact with MySQL. To do this, go to the **PHP Settings** -> **PHP Extensions** page in ExeOutput for PHP.

Then, use the following PHP code to check the connection and display an alert if it fails:

```php
<?php
// Database connection parameters
$host = 'localhost'; // MySQL server host
$username = 'your_username'; // MySQL username
$password = 'your_password'; // MySQL password
$database = 'your_database'; // MySQL database name

// Create a new MySQLi connection
$connection = new mysqli($host, $username, $password, $database);

// Check the connection
if ($connection->connect_error) {
    // Connection failed; display an alert
    echo "<script>alert('MySQL connection failed: " . $connection->connect_error . "');</script>";
} else {
    // Connection successful
    echo "Successfully connected to the MySQL server.";
}

// Close the database connection
$connection->close();
?>
```

5. PHP Frameworks

5.1 Using PHP Frameworks

ExeOutput for PHP is designed to work seamlessly with PHP frameworks. Since these frameworks typically involve numerous PHP `include` statements, ExeOutput for PHP intercepts all file requests from the PHP runtime and makes the files available (see Accessing Files in PHP).

However, this interception process can slow down your application. To address this, ExeOutput for PHP offers the External Files feature, which allows you to keep the framework's PHP files outside the application (usually in a `vendor` subdirectory). This ensures your application remains responsive, even if it requires tens of thousands of files.

If your preferred PHP framework does not function correctly, please refer to the Advice for Getting Started with PHP Applications section. For example, if PHP displays "missing file" errors or warnings like "**php failed to open stream**", you can try marking files in the File Manager (select your files, click File Properties, and enable the **Unpack the file(s) to virtual memory at startup** option) or, preferably, keep them external.

5.1.1 Redirecting Directory Requests to the Router Script

Some PHP frameworks support pretty URLs, similar to Apache's `mod_rewrite` in certain cases.

> **Note**
>
> Ensure that you enable the option to redirect all non-file requests to your framework's `index.php` router script.

5.1.2 Using Custom Redirection Rules

If you prefer to define your own redirection rules, you can use the Redirection Rules feature.

5.1.3 Setting the Initial Page / Base URL

If your `index.php` homepage should not be the first URL displayed, you can specify a custom URL for the application bootstrap. Refer to the Application Startup URL settings.

Base URLs should start with `https://heserver/` for the CEF engine, or `https://heserver.example/` for the WebView2 engine (see Accessing Files in Compiled PHP Applications).

5.1.4 Compatible PHP Frameworks

The following PHP frameworks have been successfully tested with ExeOutput for PHP:

Laravel

> **Tip**
>
> Check out our Laravel sample with step-by-step instructions.

To use Laravel, you must configure the `storage` and `bootstrap/cache` folders to remain outside the EXE so that files can be written to them. We also recommend keeping the `laravel` (and `vendor`, if applicable) subdirectories external.

In ExeOutput for PHP, navigate to the File Manager, select each of the folders mentioned above, click Properties, and enable the option: **Keep the selected file(s) external and copy them to the "Data" subfolder**.

👉 For more details, refer to the External Files topic.

CodeIgniter

> **Tip**
>
> 👉 Check out our CodeIgniter sample with step-by-step instructions.

To use CodeIgniter, you need to configure the `application/cache` and `application/logs` folders to remain outside the EXE so that files can be written to them.

In ExeOutput for PHP, navigate to the File Manager, select each of the folders, click Properties, and enable the option: **Keep the selected file(s) external and copy them to the "Data" subfolder**.

CakePHP

To use CakePHP, you need to configure the `logs` and `tmp` folders to remain outside the EXE so that files can be written to them.

In ExeOutput for PHP, navigate to the File Manager, select each of the folders, click Properties, and enable the option: **Keep the selected file(s) external and copy them to the "Data" subfolder**.

Finally, edit `www/app/Config/core.php` and add the following line:

```
Configure::write('App.baseUrl', env('SCRIPT_NAME'));
```

Fat-Free Framework (F3)

No specific configuration is required.

Other Frameworks

This list is not exhaustive. The absence of a framework does not mean that a PHP application based on it will not work when compiled with ExeOutput for PHP.

👉 Accessing Files in Compiled PHP Applications

👉 How Compiled PHP Applications Work

6. JavaScript And Browser

6.1 JavaScript and the Chromium Browser

6.1.1 The Chromium Browser Engine

ExeOutput for PHP applications use an internal browser to display pages and provide user interaction. This browser is built upon the powerful Chromium/Blink rendering engine, integrated via the Chromium Embedded Framework (CEF).

👉 Learn more about the rendering engine and configurable Chromium options.

6.1.2 Developer Tools

ExeOutput for PHP provides access to Developer Tools within your application, allowing for easier debugging and inspection of your web content.

6.1.3 JavaScript API

ExeOutput for PHP provides a custom JavaScript API through the global `exeoutput` object. This allows for deep integration between your web content and the application itself.

👉 See the `exeoutput` API documentation for more details.

6.1.4 Proxy Settings

You can configure the internal browser to use a proxy for internet connections.

👉 Learn how to modify proxy settings at runtime.

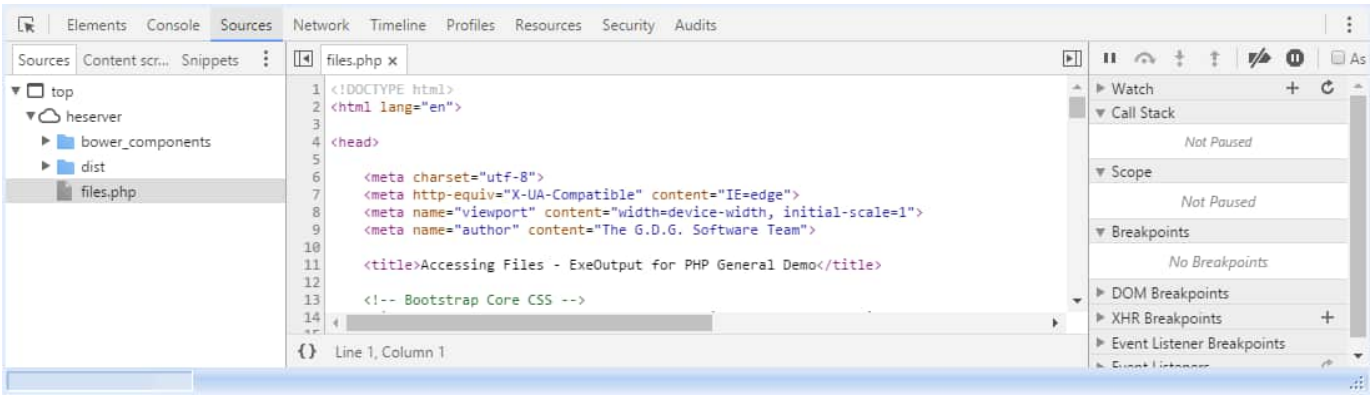
6.1.5 Authentication

The browser engine supports HTTP Basic Authentication, allowing you to create applications that require users to log in.

6.2 Developer Tools in ExeOutput Applications

6.2.1 Enabling Developer Tools in Your Application

In ExeOutput for PHP, navigate to **Application Settings** => **Rendering Engine**. Developer Tools are accessible if you enable the **DeveloperTools** option on the rendering engine settings page for CEF or WebView2:

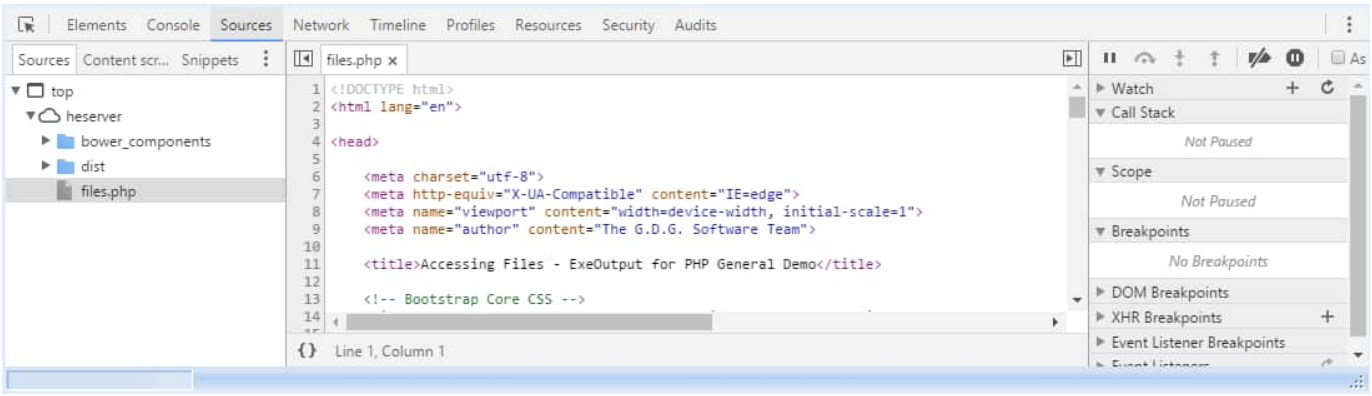


Enabling Developer Tools in Pop-up or Secondary Windows

1. Set the `DevToolsInPopup` property to `True` on the rendering engine settings page for CEF or WebView2.
2. In secondary windows, Developer Tools will function slightly differently. Instead of opening as a section within the window, they will launch as an additional, independent window.

6.2.2 Accessing Developer Tools in Your Application

If enabled, Developer Tools can be accessed through the application's context menu: simply right-click to display the context menu and select "Developer Tools".



You can also access Developer Tools in Google Chrome (or any other web browser) by navigating to `http://localhost:9000` when your application is running.

> **Note**
>
> Use the `DevToolsPort` property, available for both CEF and WebView2, to change the listening port for accessing Chromium Developer Tools. `9000` is the default value.

> **Warning**
>
> Ensure that Developer Tools are disabled before distributing your application!

6.3 The exeoutput JavaScript Object API

In addition to its own HEScript scripting language, ExeOutput for PHP extends the browser's environment with a special `exeoutput` JavaScript object. This API allows you to interact with and control the application directly from your HTML pages.

For instance, the following JavaScript code will close the application:

```
exeoutput.RunHEScriptCode('ExitPublication;');
```

> **Asynchronous Operations**
>
> The `exeoutput` methods operate **asynchronously**. They do not return values directly. Instead, results are passed to a callback function that you specify in the method call. See the examples below for details.

6.3.1 `exeoutput` Methods

Method	Prototype	Description
`CloseCurrentWindow`	`exeoutput.CloseCurrentWindow()`	Closes the current window. May prompt the user before closing.
`exportPDF`	`exeoutput.exportPDF(filename)`	Exports the current page to a PDF file specified by `filename`. **WebView2 only.** For CEF, use `window.exportPDF()`.
`moveTo`	`exeoutput.moveTo(x, y)`	Moves the current window to the specified `x` and `y` coordinates. **WebView2 only.** For CEF, use `window.moveTo()`.
`resizeTo`	`exeoutput.resizeTo(width, height)`	Resizes the current window to the specified `width` and `height`. **WebView2 only.** For CEF, use `window.resizeTo()`.
`GetGlobalVariable`	`exeoutput.GetGlobalVariable(Name, DefaultValue, Callback)`	Retrieves the value of the global variable `Name`. If not found, `DefaultValue` is returned. The result is passed to the `Callback` function. See GetGlobalVar.
`SetGlobalVariable`	`exeoutput.SetGlobalVariable(Name, Value, IsStored)`	Sets the global variable `Name` to `Value`. If `IsStored` is true, the variable is persistent (saved and reloaded on restart). See SetGlobalVar.
`RunHEScriptCom`	`exeoutput.RunHEScriptCom(ComLine)`	Executes an HEScript procedure or a function without a return value. Similar to the hescript:// protocol.
`RunHEScriptCode`	`exeoutput.RunHEScriptCode(Code)`	Compiles and executes the given string of HEScript `Code`.
`GetHEScriptCom`	`exeoutput.GetHEScriptCom(ComLine, Callback)`	Executes an HEScript function and passes its string result to the `Callback` function. See usage example below.
`GoToPage`	`exeoutput.GoToPage(URL, Window)`	Navigates to the specified `URL`. The `Window` parameter is not used and should be an empty string.
`GetString`	`exeoutput.GetString(ID, Callback)`	Retrieves a resource string by its `ID` and passes it to the `Callback` function.

6.3.2 Examples

Calling an HEScript Procedure

To call an HEScript procedure (which does not return a value), use `RunHEScriptCom`.

For example, to call the HEScript procedure `UserMain.MyProcedure`:

```
exeoutput.RunHEScriptCom("UserMain.MyProcedure");
```

Getting a Global Variable's Value

This example retrieves the value of the `HEPHPPath` variable and displays it in an alert.

```
<script language="JavaScript">
  function DemoCallback(content) {
    alert("PHP Path is: " + content);
  }

  // The default value "not found" will be used if the variable doesn't exist.
  exeoutput.GetGlobalVariable('HEPHPPath', 'not found', DemoCallback);
</script>
```

See the list of pre-defined global variables.

Calling an HEScript Function and Getting a Result

Use `GetHEScriptCom` to execute an HEScript function and receive its return value in a callback.

JavaScript:

```
<script language="javascript">
  // This callback function will receive the result from HEScript.
  function DemoCallback(content) {
    alert("The MD5 hash is: " + content);
  }

  // Calls the HEScript function "usermain.getit" and passes "password123" as a parameter.
  exeoutput.GetHEScriptCom('usermain.getit|password123', DemoCallback);
</script>
```

HEScript (`UserMain` script): This script defines the `getit` function that will be called from JavaScript.

```
function getit(what: String): String;
begin
  Result := MD5OfAString(what);
end;
```

Displaying a Resource String

This example retrieves a localized resource string and inserts it into a `div` element.

```
<div id="demo"></div>

<script language="JavaScript">
  function DemoCallback(content) {
    // Injects the localized string into the "demo" div.
    document.getElementById('demo').innerHTML = '<p>' + content + '</p>';
  }

  // Replace "MyStringID" with the name of your resource string.
  exeoutput.GetString("MyStringID", DemoCallback);
</script>
```

Closing the Application

```
// Executes the HEScript command to exit the application.
exeoutput.RunHEScriptCode('ExitPublication;');
```

6.3.3 See Also

- JavaScript `window` Object Extensions
- Introduction to Scripting
- How to Call HEScript from HTML/JS

6.4 JavaScript window extension

In addition to the "exeoutput" JavaScript object available in ExeOutput applications, ExeOutput for PHP extends the `window` object, particularly to manage secondary windows or pop-ups.

> **CEF Engine Only**
>
> These `window` object extensions are only available with the CEF rendering engine. They are not supported by the WebView2 engine. For WebView2, please use the methods of the `exeoutput` JavaScript object.

> **Tip**
>
> The following examples can be tested in the **Secondary Windows and Popups** topic of the **General Demonstration**.

The following commands are available:

6.4.1 window.open

`window.open()` is fully supported. You can also name your pop-ups to identify or manipulate them with JavaScript:

```
<script>
  $(function() {

    $("#btn").click( function()
    {
        window.open('samples/testpopup.php', 'Popup');
    }
 });
</script>
```

6.4.2 window.exportPDF

`window.exportPDF(pdffilename)` is a JavaScript extension available in ExeOutput applications. This function allows you to export the window's content as a PDF file.

`pdffilename` allows you to specify the full path and filename for the PDF export.

For example:

```
<script language="javascript">
   // Callback for the save dialog box: `content` contains the full path to the future PDF file.

    function DemoSavePDF(content) {
            if (content === "") return;
            window.exportPDF(content);
        }

    function exportpdf() {
        // Run the HEScript script `SavePDFDlgFile` (defined in `UserMain`) to obtain the PDF filename.
        exeoutput.GetHEScriptCom('hescript://UserMain.SavePDFDlgFile', DemoSavePDF);
        }

</script>
```

6.4.3 window.moveTo and window.resizeTo

`window.moveTo()` and `window.resizeTo()` are also supported.

Here is an example of displaying a fullscreen window:

```
function showmaxpop()
     {
     var w = window.open('about:blank', 'Report', "fullscreen");
     w.document.open();
```

```
        w.document.write('Hello large popup');
        w.document.close();
        w.moveTo(0,0);
        w.resizeTo(screen.availWidth, screen.availHeight);
        }
```

See these topics also:

👉 The JavaScript `exeoutput` Object

👉 How to Call HEScript Procedures/Functions?

6.5 Special Protocols for Links

ExeOutput for PHP provides **pre-defined protocols** for hyperlinks within your applications, which can also be utilized for menu items.

Syntax:

```
<a href="[protocol][destination URI]">Your link</a>
```

6.5.1 List of predefined protocols

- `heopenit://` + virtual path to the file you want to open with the external application associated with its file type (for example, `heopenit://mydoc.doc` will extract and open `mydoc.doc` in Microsoft Word).
- `heopenext://` + path to an external file you want to open with the external application associated with its type (a file that is not compiled into the application but resides in the same folder as the EXE file). It can also be located in a subfolder.
- `heexternal://` + a URL or a path to an application: this protocol allows you to launch an external application or open a URL in the end user's default web browser.

6.5.2 Examples

☑ `heopenit://` can be used for **any document file** (e.g., executable program files, text files, Microsoft Office® files) that is **compiled into the EXE**. These files are listed in the File Manager.

```
<a href="heopenit://mydocument.docx">Open this Word document</a>
```

☑ `heopenext://` can be used for **any document file** (e.g., executable program files, text files, Microsoft Office® files). These files are **available outside the application**, residing in the same folder as the EXE file.

```
<a href="heopenext://myprogram2.exe">Start another application</a>
```

Example of opening a file in a subfolder named `subdir`:

```
<a href="heopenext://Subdir/powerpoint%20demo.pptx">Open PowerPoint in subdir</a>
```

☑ `heexternal://` is used to open URLs in the user's default browser or to launch external applications.

Open a website:

```
<a href="heexternal://https://www.exeoutput.com">Visit our website</a>
```

Launch an application (e.g., Calculator):

```
<a href="heexternal://calc.exe">Open Calculator</a>
```

It also supports custom protocol URLs, like WhatsApp:

```
<a href="heexternal://whatsapp://send?text=Hello">Send a WhatsApp message</a>
```

> **Tip**
>
> If a filename contains spaces, encode it properly (e.g., `my%20program.exe`).

6.6 HTML5 and CSS3 Support

Applications created with ExeOutput for PHP use the Chromium rendering engine, which provides native support for modern web standards. Features such as HTML5 Canvas, Web Storage, WebRTC, and CSS3 animations are fully supported.

WebRTC Requirements

Using WebRTC requires a secure context. Therefore, you must meet the following conditions:

1. Ensure that the **"Use secure HTTPS for internal protocol" option is enabled**.
2. Activate the `EnableMediaStream` property in the Chromium engine options.

6.6.1 Live Demonstrations

The General Demonstration included with ExeOutput for PHP provides several working examples:

- HTML5 Notifications
- HTML5 Web Storage
- HTML5 Canvas
- Embedded YouTube Video
- WebRTC Audio Recording

6.7 Using HTML5 Video and Audio

Applications created with ExeOutput for PHP can play video and audio directly on webpages using the HTML5 `<video>` and `<audio>` tags. The `autoplay` attribute is fully supported.

For example, this HTML code will play a video with standard browser controls:

```
<video id="video" loop autoplay controls>
  <source src="/assets/sunpeek.webm" type="video/webm" />
</video>
```

6.7.1 File Management

You have two options for including media files in your application:

- **Internal (Compiled):** You can compile media files directly into your application's `.EXE`. This method is convenient and recommended for smaller files (e.g., under 5 MB).
- **External:** For larger files, it is best to keep them as external files. Place them in the same folder as your application's `.EXE`, or in a subfolder. The application will automatically locate and use these external files if they are not found inside the `.EXE`.

Supported Media Formats

Due to licensing restrictions, the built-in Chromium engine only supports open-source audio and video formats and their associated codecs, such as: * WEBM * WEBA * OGG / OGV * MP3

Proprietary formats like MP4 (H.264) and AAC are **not supported** out-of-the-box.

To use your existing media, you must convert it to a supported format. Free encoding software, such as the open-source Miro Video Converter, can help with this process.

Tip

See **working samples** in the HTML5 video topic and HTML5 audio topic of the General Demonstration.

6.8 Using Flash Objects (SWF) in Compiled Applications

Adobe Flash® was a popular format for delivering rich web content over the internet via the Adobe® Flash Player. ExeOutput for PHP applications **can still display Flash content** (in SWF format) under certain conditions.

> **Warning**
>
> Adobe Flash has been discontinued, and HTML5 is recommended as an alternative. We do not provide technical support or assume liability for Flash content. ExeOutput for PHP does not include a Flash player. However, users who wish to play Flash content within their PHP application can follow the instructions below.

Two methods are available for playing Flash content: using the Flash Player emulator Ruffle, or employing an older version of Chromium that still supports the legacy Flash Player.

6.8.1 Using Ruffle

You can use the Flash Player emulator named Ruffle.

Ruffle is a Flash Player emulator that runs natively in Chromium via WebAssembly, making it compatible with ExeOutput applications. Download Ruffle, follow its documentation to integrate the required script into your webpages, and Ruffle will enable your Flash content.

> **Tip**
>
> 👉 See **working samples** in the Flash topic of the General Demonstration.

> **Note**
>
> Ruffle is still in development; not all Flash movies may work, especially those requiring ActionScript 3.

6.8.2 Using an Older Chromium Version

The Chromium Embedded Framework (CEF) used by ExeOutput for PHP can still play Flash content if you choose CEF version 87 as the rendering engine and if the `Pepper Flash` player is available.

👉 An **external requirement exists**: the Pepper Flash DLL file must be placed in the same folder as your application's EXE file, and it must be redistributed to your users.

ExeOutput for PHP does not provide the Pepper Flash DLL file due to legal reasons.

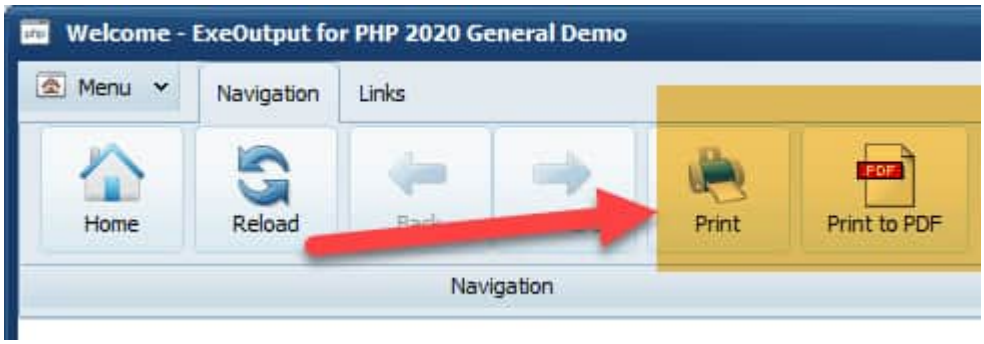
The DLL file begins with `pepflashplayer32`. For instance, place the Pepper Flash DLL `pepflashplayer32_32_0_0_238.dll` into the EXE folder, and the Chromium engine will recognize it for playing Flash files.

Finally, use the standard `EMBED` HTML tag to display Flash movies. Example:

```
<embed align=""
   type="application/x-shockwave-flash"
   name="myMovieName2"
   bgcolor="#FFFFFF"
   quality="high"
   src="assets/ball.swf"
   height="400" width="160"
   href="assets/ball.swf"/>
```

6.9 Print, Kiosk Printing, and PDF

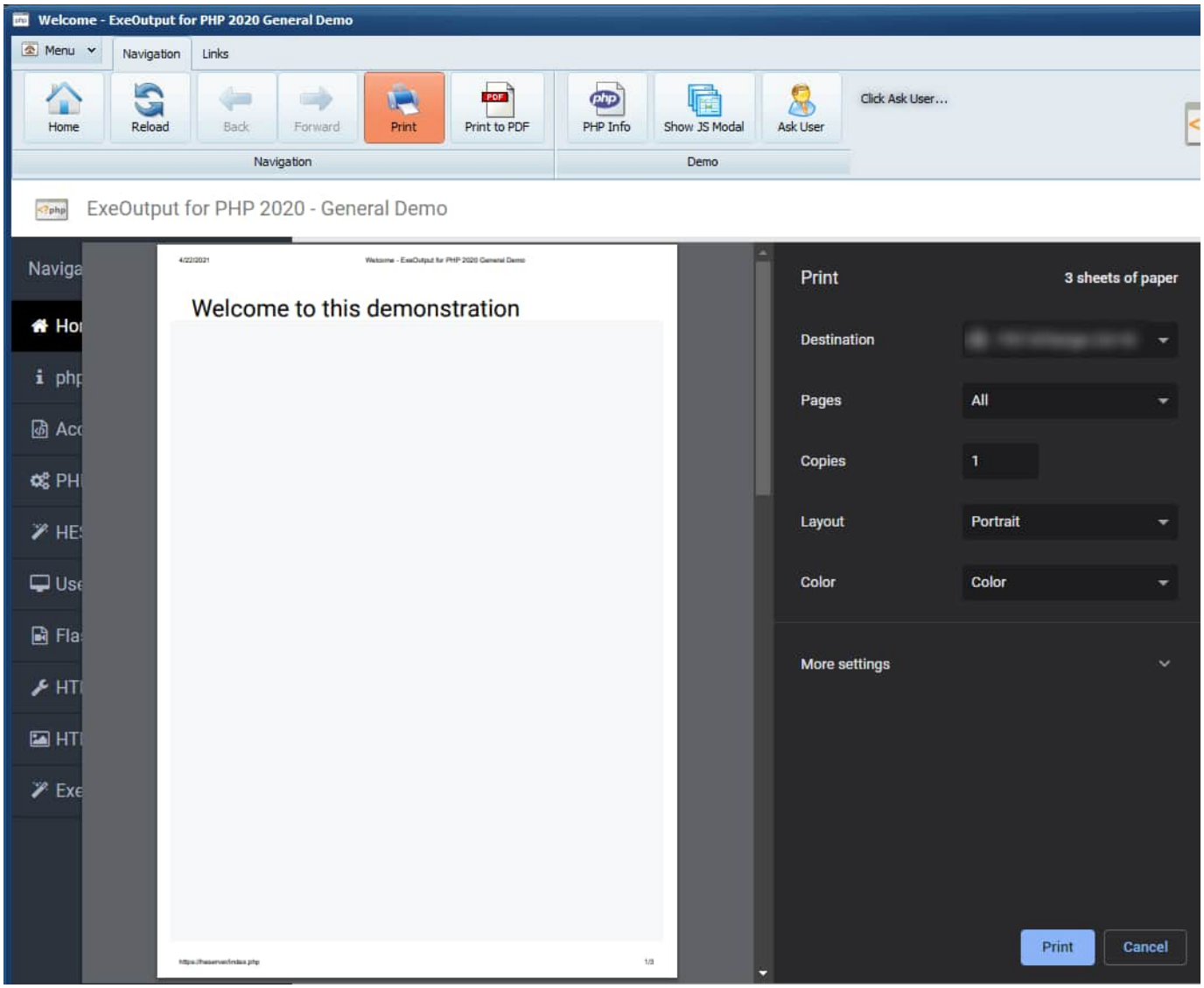
Applications created with ExeOutput for PHP can print webpages directly to a printer or save them as PDF files.



👉 You can customize various properties of the printing process using the "Printer" component.

6.9.1 Print Preview

By default, print preview is enabled. This means users will see the following dialog box when they select "Print":



> **Tip**
>
> You can disable Print Preview by setting `EnablePrintPreview` to `false` in the "Printer" component.

6.9.2 Direct or Kiosk Printing

ExeOutput for PHP allows you to print directly to the default printer without prompting the user—a feature known as kiosk printing.

To **enable kiosk printing**, edit the "Printer" component and set `EnablePrintPreview` to `false` and `EnableKioskPrint` to `true`.

Programmatically Toggling Kiosk Printing / Print Preview

You can enable or disable kiosk printing and/or print preview at runtime using JavaScript, PHP, or HEScript.

The following global variables are used internally to control kiosk printing and print preview. You can modify their values to change the application's behavior.

Variable Name	Description
PrintPreviewEnabled	`1` (enabled) or `0` (disabled).
KioskPrintEnabled	`1` (enabled) or `0` (disabled).

For example, the following HEScript code enables kiosk printing at runtime. Simply add this code to the `UserMain` script:

```
procedure EnableKiosk;
begin
 SetGlobalVar("KioskPrintEnabled", "1", false);
 SetGlobalVar("KioskPrintCopies", "1", false);
 SetGlobalVar("PrintPreviewEnabled", "0", false);
end;
```

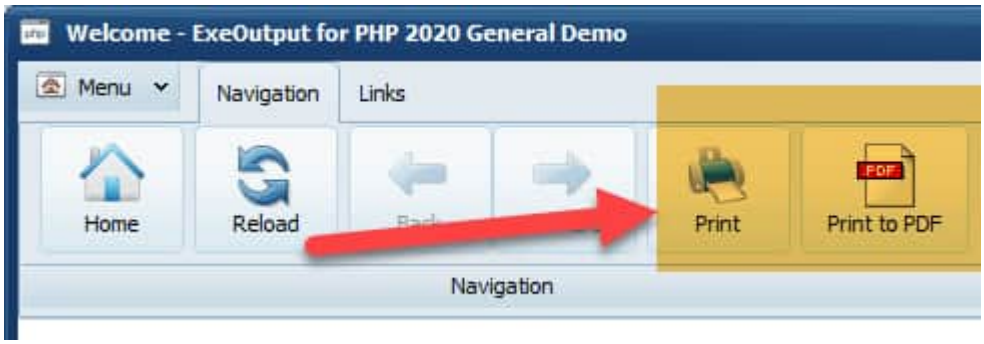
This procedure can then be invoked from JavaScript or PHP or associated with an event.

Here is the equivalent code in JavaScript:

```
<script language="JavaScript">
exeoutput.SetGlobalVariable('KioskPrintEnabled', '1', false);
exeoutput.SetGlobalVariable('KioskPrintCopies', '1', false);
exeoutput.SetGlobalVariable('PrintPreviewEnabled', '0', false);
</script>
```

6.9.3 Print to PDF

You can allow users to export the current webpage as a PDF. For example, clicking the "Print to PDF" button prompts the user for a filename, and the PDF is then saved to the specified location.



Saving Directly to a PDF File Programmatically

You can use the `PrintPDF` and `PrintPDFFile` HEScript functions.

- `PrintPDF` performs the same action as clicking the "Print to PDF" button.
- `PrintPDFFile` exports the PDF directly to a file you specify.

For example, the following HEScript code exports the current webpage to a PDF file named "mydoc.pdf" in the user's "My Documents" folder. Add this code to the UserMain script:

```
procedure ExportPDFSample;
var
 S: String;
begin
 S := GetGlobalVar("HEMyDocDirectory", "C:") + "\mydoc.pdf";
 PrintPDFFile(S);
end;
```

You can then invoke this function from JavaScript or PHP or associate it with an event or a UI control.

Alternatively, ExeOutput apps provide a JavaScript extension, `window.exportPDF(pdffilename)`, which allows you to export the window's content as a PDF file. The `pdffilename` parameter specifies the full path and filename for the exported PDF.

For example:

```
<script language="javascript">
   // Callback for the save dialog box.
   // The 'content' parameter contains the full path for the new PDF file.

    function DemoSavePDF(content) {
            if (content === "") return;
            window.exportPDF(content);
        }

    function exportpdf() {
        // Run the HEScript function SavePDFDlgFile (defined in UserMain) to get the PDF filename.
        exeoutput.GetHEScriptCom('hescript://UserMain.SavePDFDlgFile', DemoSavePDF);
        }

</script>
```

See Also: "Printer" component

6.10 Adding Custom Headers to Requests

Some applications built with ExeOutput for PHP function as custom browsers for navigating websites. You can add custom headers to HTTP requests, allowing you to identify the user by passing a browser/machine ID and take appropriate actions.

> **Note**
>
> Headers are name/value pairs that appear in both request and response messages. The header name is separated from its value by a single colon.

Insert the following HEScript code into your `UserMain` script (see how to do this) to create a custom HTTP header. Modify the `My Header Name` and `My Header Value` fields as needed.

```
procedure OnStartMainWindow;
begin
  // When the main window is going to be displayed (just before the homepage is shown).
  SetUIProp("crm", "CustomHeaderName", """My Header Name""");
  SetUIProp("crm", "CustomHeaderValue", """My Header Value""");
end;
```

This can be done in the `OnStartMainWindow` HEScript event.

> **Warning**
>
> In this specific case, the third parameter should use triple double quotes for string values (e.g., `"""String Value"""`) and single quotes for functions (to be interpreted by the script engine, e.g., `"MD5OfAString(...)"`).

6.10.1 Tutorial: How to Send a Unique Computer ID to a Remote Server

1. Go to the Scripting page and double-click `UserMain`.
2. Locate the `OnStartMainWindow` event and replace it with the following code:

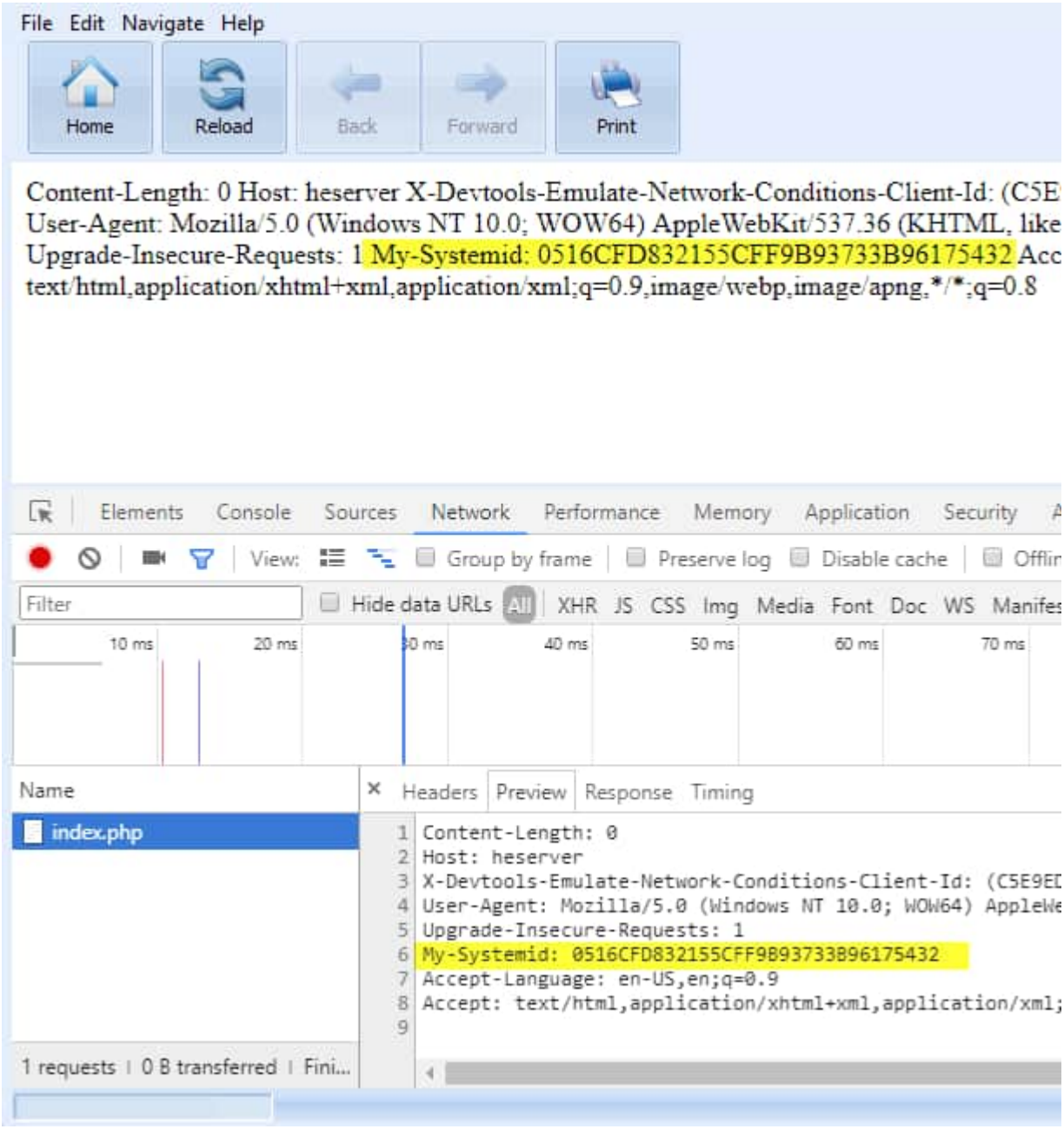
```
procedure OnStartMainWindow;
begin
  // When the main window is going to be displayed (just before the homepage is shown).
  SetUIProp("crm", "CustomHeaderName", """My-SystemID""");
  SetUIProp("crm", "CustomHeaderValue", "MD5OfAString(""MySystemID"" + GetManualHardwareID(2))");
end;
```

`GetManualHardwareID` is an HEScript function that returns a unique system ID based on hardware specifications. The `2` argument indicates that the unique ID will be based on CPU specifications (e.g., CPU ID). We also generate an MD5 hash sum of the system ID.

Now, when you navigate to any remote URL (e.g., `http://www.yoursite.com/...`), a custom HTTP header named `My-SystemID` is added to the request. You can read this header with a PHP script on your remote server. For instance, the following PHP script prints all HTTP request headers:

```
<?php
foreach (getallheaders() as $name => $value) {
  echo "$name: $value\n";
}
?>
```

The following screenshot shows the result of the code above:



As you can see, there is a custom HTTP header named `My-SystemID` with the appropriate MD5 hash sum.

👉 Scripting with HEScript

6.11 Opening New Windows

ExeOutput for PHP effectively handles pop-ups and secondary windows, allowing you to open them using the standard `_blank` target in your links.

To control the size of a new window, use the `window.open` JavaScript method:

```
var myWindow = window.open("mypage.php", "", "width=200,height=100");
```

> **Tip**
>
> The `window.moveTo()` and `window.resizeTo()` JavaScript methods are also supported. Additionally, you can directly export a pop-up's contents to a PDF.

6.11.1 Fullscreen Mode

Just like the main application window, pop-up windows support fullscreen mode. Users can press the **F11** key to toggle fullscreen on and off for the active pop-up window.

6.12 How to Configure Proxy for Your App

You can configure your application to use a proxy server for connecting to the Internet, rather than connecting directly.

> **Info**
>
> The following proxy server instructions apply only to the Chromium rendering engine, not to PHP.

Place the following code in the `UserMain` HEScript script:

```
procedure OnStartMainWindow;
begin
// This code runs just before the homepage is shown, as the main window is about to be displayed.
SetUIProp("crm", "ProxyServer", """122.133.144.155""");
SetUIProp("crm", "ProxyUsername", """user""");
SetUIProp("crm", "ProxyPassword", """pass""");
SetUIProp("crm", "ProxyPort", "3456");
SetUIProp("crm", "ProxyType", "3");
SetUIProp("crm", "ProxyScheme", "0");
NavigateCommand(13); // Update preferences
end;
```

> **Warning**
>
> Triple quotes are required for the `ProxyServer`, `ProxyUsername`, and `ProxyPassword` fields.

- `ProxyScheme`:
- 0 (HTTP)
- 1 (SOCKS4)
- 2 (SOCKS5)
- `ProxyType`:
- 0 (direct, no proxy)
- 1 (auto-detect)
- 2 (system-defined)
- 3 (fixed servers)
- 4 (pac script)

For a PAC script, you must also specify the script URL using `ProxyScriptURL`.

6.13 HTTP Basic Authentication in Applications with Integrated Browser

When accessing a webpage protected by **HTTP Basic Authentication** (e.g., via `.htaccess`), a dialog box prompts for username and password.

ExeOutput for PHP can display a **"Remember my credentials"** checkbox, allowing users to save credentials. The checkbox text can be customized using the `SRememberCredentials` resource string.

6.13.1 Managing Stored Credentials

Saved credentials are stored in **Windows Credential Manager** (Windows Vault).

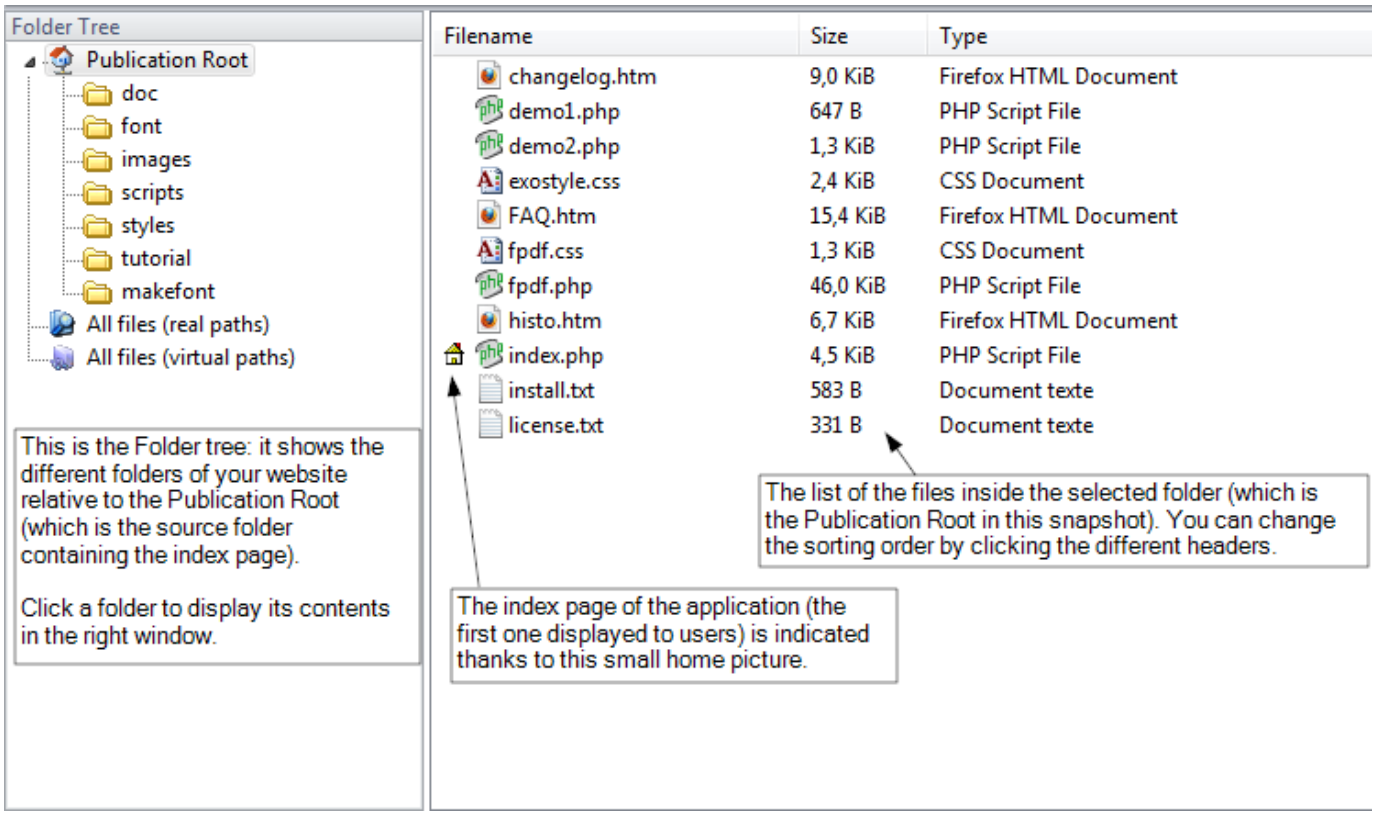
To remove them:

1. Open **Credential Manager** (press **Windows Key**, type *Credential Manager*, press **Enter**, or open via **Control Panel**).
2. Select **Windows Credentials**.
3. Scroll to the **Generic Credentials** section.
4. Locate the entry named **CompanyName – ApplicationName**.
5. Expand the entry to view details (UserName, etc.).
6. Click **Remove** to delete it.

7. File Manager

7.1 File Manager

You can access the **File Manager** by clicking its ribbon at the top. It allows you to manage all files to be **compiled** into your application and **behaves like Windows Explorer**.



☑ You get a **global view** of your application's contents. The Folder tree displays the available subfolders, similar to using an **FTP program** to manage files on a server.

☑ To **view a folder's contents**, simply select it in the Folder tree, and its contents will be listed. Note that ExeOutput for PHP may take a few seconds to load the entire file list when displayed for the first time, especially if the subfolder contains a large number of files.

☑ The **index page** is indicated by a small house icon (as shown in the screenshot). **Excluded files** are marked with a red cross.

Selecting the **All Files (real paths)** node in the Folder tree will list all files to be compiled with their full path. Selecting the **All Files (virtual paths)** node will list all files with their virtual path (i.e., the path that follows the server or domain name in a URL).

> **Warning**
>
> ExeOutput for PHP stores file relative paths in projects. Once a project is started, you may need to move source files to another directory. In some cases, ExeOutput for PHP may prompt you to select the new source folder, and the program will attempt to correct all file path references. It will refuse to compile a project if any files are missing.

7.1.1 About Virtual Paths

Virtual paths are paths that follow the server or domain name in a URL. We use virtual paths in applications because they function like servers: to access a compiled file, you need to use its URL. For instance, if you have the URL: `ghe://heserver/images/picture1.gif`

- `https://heserver/` (for CEF) or `https://heserver.example/` (for WebView2) refer to the custom internal protocol and namespaces used by ExeOutput for PHP to display files in the custom browser. The `ghe://` protocol is also an alias.
- `images` is the **virtual path**.
- `picture1.gif` is the filename.

Note that ExeOutput for PHP automatically manages virtual paths (see below); all features related to virtual paths are designed only for advanced users.

7.1.2 Adding Virtual Folders

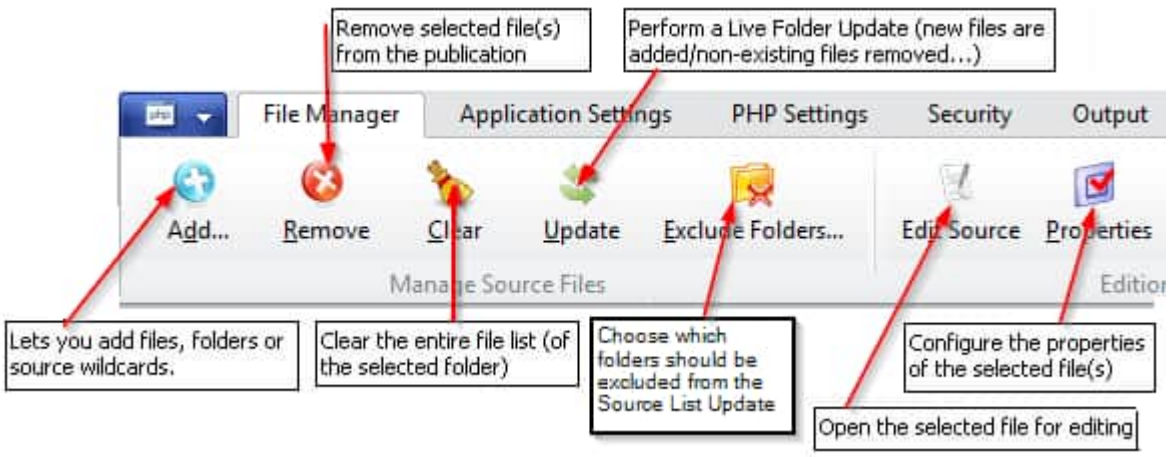
To add empty virtual folders to your application, click **Add** and then **Add Virtual Folder**. You will be prompted to enter its name, and it will then be created as a **child of the selected folder** (if no folder is available or selected, it will be created as a child of the application root). You can then move files to this folder using drag-and-drop operations.

> **Warning**
>
> **Important:** If you do not place any files inside an empty virtual folder, it will be automatically removed the next time your project is loaded.

7.1.3 Managing Files

All source files are generally added automatically when ExeOutput for PHP creates your project. However, you can manually add new files, folders, or source wildcards to your list at any time. Alternatively, let ExeOutput for PHP perform this task automatically using the powerful **Source File List Update** option.

You can manage your files using the button bar below or the **right-click context menu** on the file list:

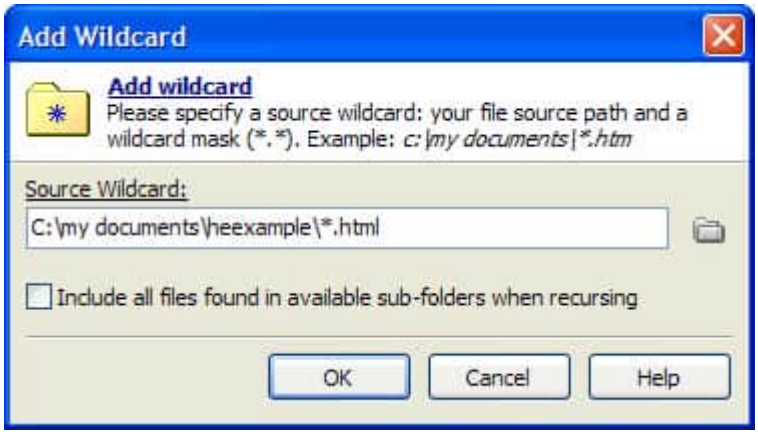


Adding Files

To add files to the list, click the **Add** button and select an option. You can add single or multiple files, entire folders, virtual folders, and custom wildcards. Just select the appropriate action after clicking the **Add** button:

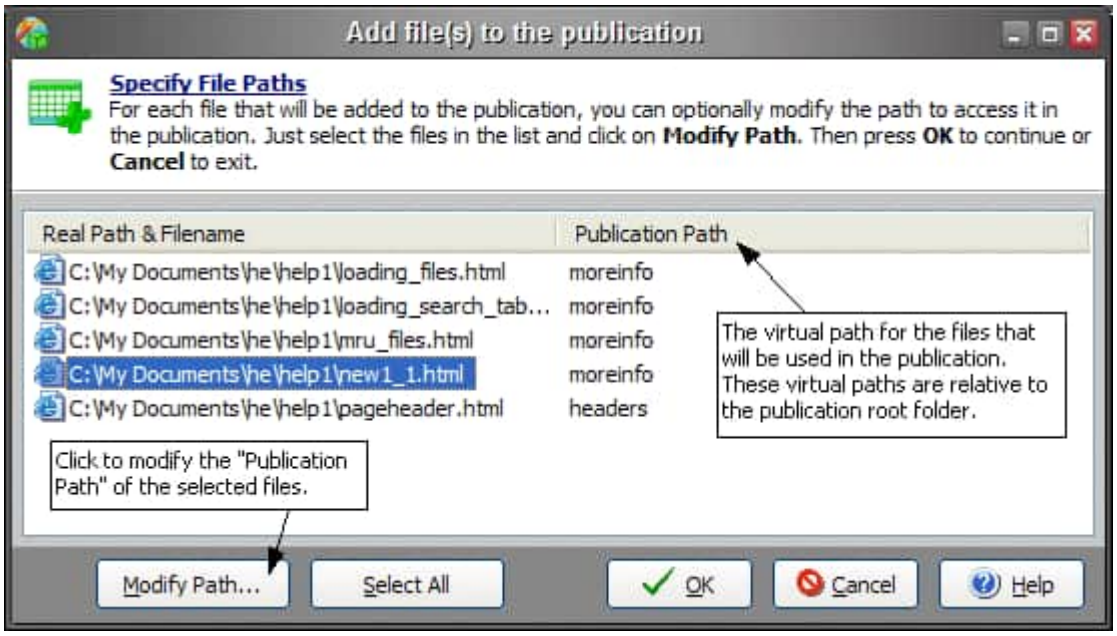
- Add Files
- Add Folder
- Add Source Wildcard
- Add Virtual Folder

For example, selecting "Add Wildcard" will open this window:



You can enter a path and specify which files should be added using a mask. In the situation shown in the screenshot, only `.html` files from `C:\my documents\heexample` will be added. Alternatively, you can add entire folders (including their subfolders or not) using the "Add Folder" command.

☑ When added files are not in the source folder or one of its children, you need to specify the **application path** (also known as the virtual path) that should be used to access these files once they are compiled into the application. ExeOutput for PHP lets you determine these virtual directories using the following window (which is automatically shown when adding files):



The real paths appear on the left; on the right, you can determine the related virtual path (also known as the application path). By default, ExeOutput for PHP attempts to find the best virtual paths, but you can change this yourself. Just select one or more files, then click **Modify Path**. You will be prompted to enter the new virtual path and confirm your changes.

Once you click **OK**, ExeOutput for PHP adds them to the application's file list, and the result is:

You can see that the virtual paths `moreinfo` and `headers` appear as children of the application root, even though the added files do not belong to the source folder.

Success

This option is powerful because you can add files from any folder and determine virtual paths yourself. The main advantage is that you do not need to copy the source files to the source folder before compilation.

Drag-and-Drop Operations

ExeOutput for PHP supports drag-and-drop operations:

- From **Windows Explorer** and other shell windows: Select all the files in Explorer, drag them onto the manager's file list, and they are automatically added. You may be prompted to enter the virtual path for the dropped files, as explained above. You can launch Windows Explorer by clicking the **Explorer** button.
- Within the **File Manager**: You can move files from one virtual folder to another, thereby changing their virtual paths as desired. Highlight the file(s) you want to move in the list, then drag and drop them onto the destination folder in the Folder Tree. Note that real paths are not modified (source files are not physically moved).

Removing Files

☑ To remove one or more files from the list, select them and click the **Remove** button. You can select multiple files by pressing **Ctrl+Click** or **Shift+Click**. You may also use the **Select All** menu command from the context menu.

☑ Finally, the **Clear** button lets you **remove the entire folder currently selected in the Folder Tree**. This includes all files and subfolders within that selected folder. This operation cannot be undone, so proceed with caution. It is also useful for **removing empty folders**.

Notes:

- You cannot remove the index page from the application.
- You cannot clear the application root; use the **Remove** operation instead.
- Removing files does **not** delete them from your hard disk; it only removes them from the application.

Sorting Files

ExeOutput for PHP displays information about each file, including filename (or file path/virtual path), file type, and size. If you need to sort your files according to a specific criterion (for example, file types), click the associated column header. You can also resize the columns.

Source File List Update Operation

Generally, source files need to be updated. You can add new files to the source folder, update existing ones, or remove unused ones. In other words, the contents of the source folder may vary, and consequently, the file list maintained by ExeOutput for PHP may become outdated.

You can, therefore, use the **File List Update** action. This action forces ExeOutput for PHP to scan the source folder (and its subfolders) and detect all changes. It then compares the results and determines the appropriate actions:

- If new files are found (not in the file lists), they are added. You can optionally be prompted to specify the new virtual paths (see the Environment Options).
- If source files are newer, ExeOutput for PHP updates its file lists.
- If source files are not found (e.g., they were deleted), they are removed from the file lists. You can also be prompted to confirm the operation (see the Environment Options).

Consequently, you do not need to manage the file lists manually.

There are several ways to trigger a File List update:

- Click the **Update** button in the File Manager.
- Configure automatic File List updates using the Environment Options.
- You can manually indicate which subfolders should be **excluded** from the File List update by clicking **Exclude Folders** and adding full paths to the folders you want to exclude to the dedicated list. You can also exclude some files from being added to the file list based on their extension. Go to the Environment Options.

Notes: * File List Update only accounts for files in the source folder (or subfolders). If you have files from other folders, the operation will ignore them. * **Hidden files are automatically excluded**: Files with the "hidden" attribute are not included during a file list update.

7.1.4 Editing Files

Select a file in the list and click the **Edit** button.

- If the selected file is a PHP or HTML-compatible page, the internal PHP/HTML editor will be displayed.
- Otherwise, the program registered with the file will be opened to edit it (this is equivalent to double-clicking the file in Windows Explorer).

☑ More information about the internal PHP/HTML editor

7.1.5 Setting Folder, File, and PHP/HTML Page Properties

You can configure options regarding compression and security for source files (and even entire folders). Select a folder or file in the list and click the **Properties** button (or double-click it). The file properties editor will then be displayed.

☑ More information about the file properties.

7.1.6 Using the Context Menu

All of the previous commands are also available from the context menu. Right-click the file list to display it. It contains additional commands not directly available, such as "Select All" or "File Information". The last command provides global information about the application files in the selected folder (total size, number of files, etc.). Finally, the "Shell Properties" command will display the "Properties" dialog available in Windows Explorer (allowing you to access the properties of the selected file).

7.2 File Properties Editor

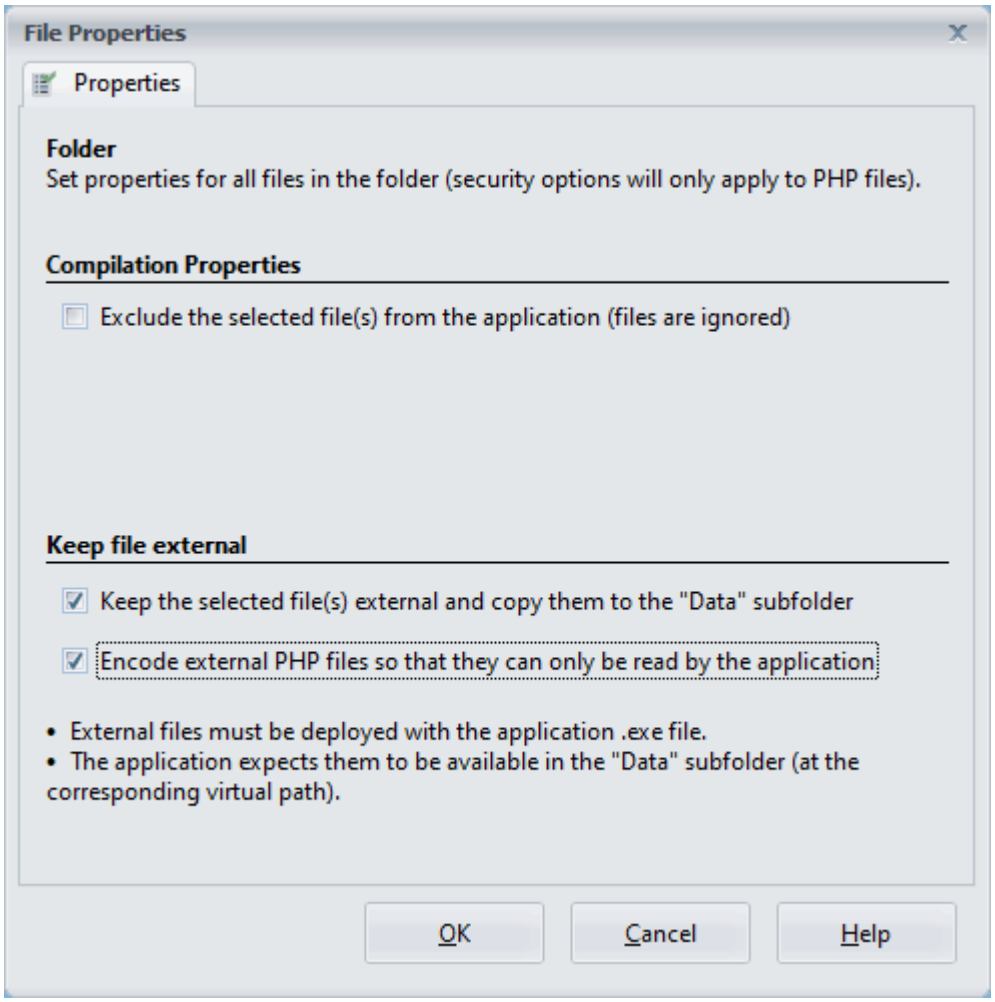
File compression and security properties can be edited via the File Manager. To edit properties, select one or more files, then click **Properties** (or press **Ctrl+P**). The following window will appear:

If the selected file is a PHP script, you will see two tabs: **Properties** and **Security**. For other file types, only the **Properties** tab is available.

The name and type of the selected file will be displayed, or "Multiple files" if several are selected. Hovering over the filename will show the **full path** to the file in a tooltip.

7.2.1 Folder Properties

You can also set properties for an entire folder: select a folder in the **Folder Tree** and click **Properties**. The "Folder" window will appear:



For folders, you can apply property values to all files and subfolders within. For example, excluding a folder will automatically exclude all contained files and subfolders.

> **Warning**
>
> Folder properties override individual file properties. If a folder is set to be kept external, individual file settings within that folder cannot be altered. In such cases, ExeOutput will disable the corresponding checkbox.

7.2.2 File Properties

Compilation Properties

- **Exclude the selected file(s)**: If enabled, ExeOutput for PHP will exclude the file from compilation, making it completely unavailable. Excluded files are marked with a red cross in the File Manager.
- **Do not compress the selected file**: If this option is enabled, ExeOutput for PHP will compile the file but not compress it. The file is stored uncompressed in the application's data folder. This option is beneficial for including files that are already compressed.
- **Unpack the file(s) to virtual memory at startup**: This option unpacks the file to virtual storage (the virtual `Data` subfolder within the folder where the EXE is located) when the application starts. The file is unpacked to memory, not to the hard disk, and can be accessed by PHP as if it were a regular file on the disk. This is akin to the internal `exo_unpackvirtualfile` PHP function. This setting helps prevent users from substituting PHP files with altered versions in the `Data` subfolder.

Keep File External and Copy to the "Data" Subfolder

To avoid the unpacking process, large files or folders with many files can be kept outside the application's .EXE file. These files are then loaded directly from the "Data" subfolder, which significantly reduces loading times.

If this option is enabled, ExeOutput for PHP will automatically copy the external files to the designated location in the `Data` subfolder.

> **Important**
>
> Existing external files in the destination folder will be **overwritten** by ExeOutput for PHP if the source file is more recent than their copies in the destination folder.
>
> External files must be deployed alongside the application's .exe file. Installers generated by ExeOutput for PHP can include external files in the `Data` subfolder.

Encoding External PHP Files for Enhanced Security

ExeOutput for PHP allows you to encrypt external PHP files, ensuring that their contents can only be accessed by the application and remain hidden from users.

The downside is that external files will be encrypted during each application compilation, extending the compilation time when encryption is used.

> **Note**
>
> This encryption feature is specific to PHP files; non-PHP files are not affected.

See also: File Properties Editor

7.2.3 Security

This tab offers **two security features specific to PHP**.

Protection Marks

You can mark selected PHP files for additional protections, such as internal encoding. These protections must be enabled in the PHP settings ribbon to take effect.

- **Encode PHP file with internal protection**: This option encrypts the PHP file's content at compilation, ensuring it remains encrypted in memory even at runtime. This is beneficial for protecting your PHP source code from being exposed during a memory dump.
- The "Do not cache PHP source file in memory" option is incompatible with the **Unpack the file(s) to virtual memory at startup** setting.

7.3 About External Files

Large source files, such as videos and other media resources, can be **kept outside of the application** to significantly reduce loading times. Initially, all files compiled into the application's .exe must be unpacked into memory to be accessible. This unpacking step requires time, which can be substantial for very large files.

To minimize loading time, you can store your files externally to the application's EXE file. This way, the application only needs to load them, not unpack them.

> **Warning**
>
> External files **must be deployed** alongside the application's .exe file. Therefore, using an installer for your application is the best deployment method. Installers generated by ExeOutput for PHP can include external files in the `Data` subfolder.

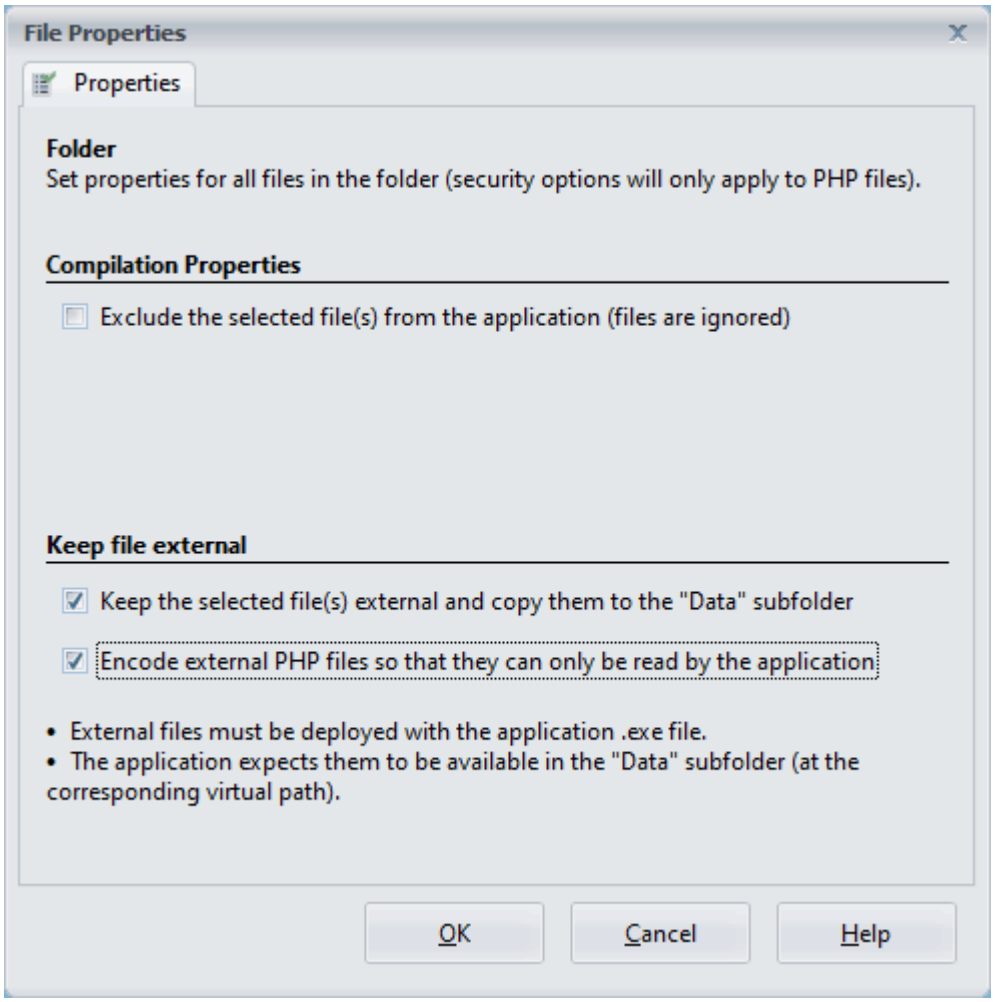
7.3.1 Keeping Large Amounts of Files External

Some PHP applications may contain tens of thousands of files. Compiling these directly into your application can significantly slow it down. Keeping your files external, however, ensures that your application's performance remains high, even when compiled with ExeOutput for PHP, compared to a server-hosted environment.

> **Important**
>
> We recommend keeping file-heavy and non-essential directories external to the application's EXE file. Directories such as `vendor` or framework dependencies, which are often open-source, do not need protection and can thus be stored externally.

7.3.2 Setting a File as External

To designate a file as external, select it in the File Manager and click Properties. The "File Properties" window will appear; enable the **Keep the selected file(s) external** option. ExeOutput for PHP will automatically copy the external files to the appropriate location in the `Data` subfolder.



Important

If external files already exist in the destination folders, they will be overwritten by ExeOutput for PHP if the source file is more recent than the one in the destination folder.

7.3.3 Encoding External PHP Files for Enhanced Security

ExeOutput for PHP allows you to encrypt external PHP files, ensuring that their contents can only be accessed by the application and remain hidden from users.

The downside is that external files will be encrypted during each application compilation, extending the compilation time when encryption is used. It also slightly reduces the application's performance, as it introduces an additional decryption step that must be executed at runtime when running the external encrypted PHP scripts.

Note

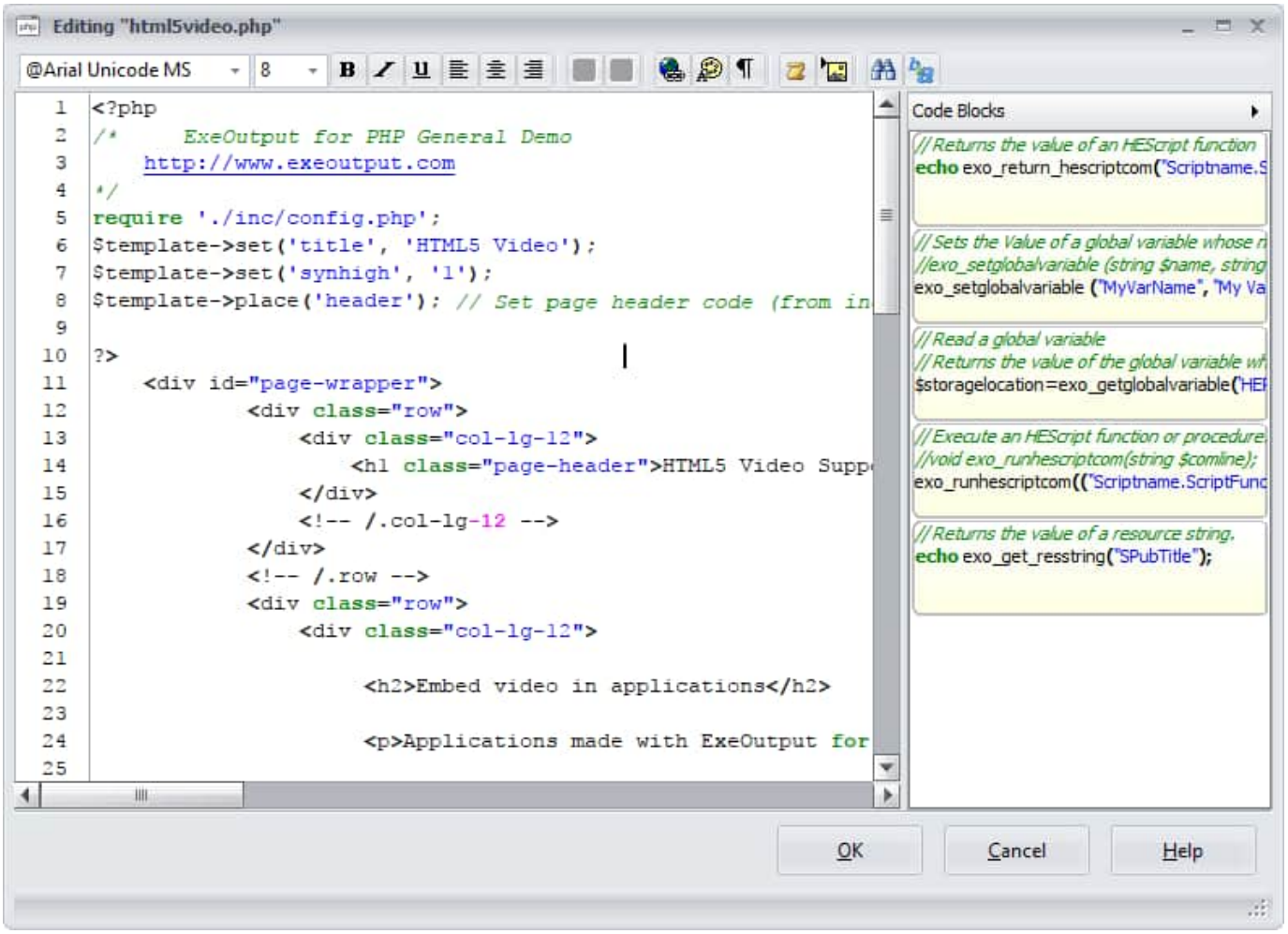
A new encryption key is generated with each build of your application, preventing external encrypted PHP files from being shared between different applications.

See also: File Properties Editor

7.4 Internal Code Editor

ExeOutput for PHP includes a built-in code editor that allows you to modify your PHP, JavaScript, and HTML files directly within the interface. The editor provides syntax highlighting for these languages.

To open a file in the editor, go to the File Manager, select a PHP, HTML, or JavaScript file, and click **Edit**.



The editor provides a simple text area with syntax highlighting to make code easier to read. The toolbar at the top provides basic formatting options for HTML content, similar to a standard rich text editor. It also includes buttons for special commands and search functionality. Hover your mouse over a button to see a tooltip describing its function.

7.4.1 Saving Changes

Make your changes directly in the text area. When you are finished, click **OK** to save your changes back to the file.

A backup file (`.~*`) can be automatically created upon saving, depending on your Environment Options.

7.4.2 Special Toolbar Commands

Insert Image

This command helps you insert `<img>` tags into your HTML. It displays a list of all image files in your project, and upon selection, it automatically inserts the required HTML code into your document.

Insert HEScript Call

This command is for advanced users working with HEScript. It inserts an HTML hyperlink (`<a>`) that calls an HEScript function. A dialog will show all available script functions in your project. After you choose one, the editor inserts a hyperlink that will execute the function when clicked by the user.

7.4.3 See Also

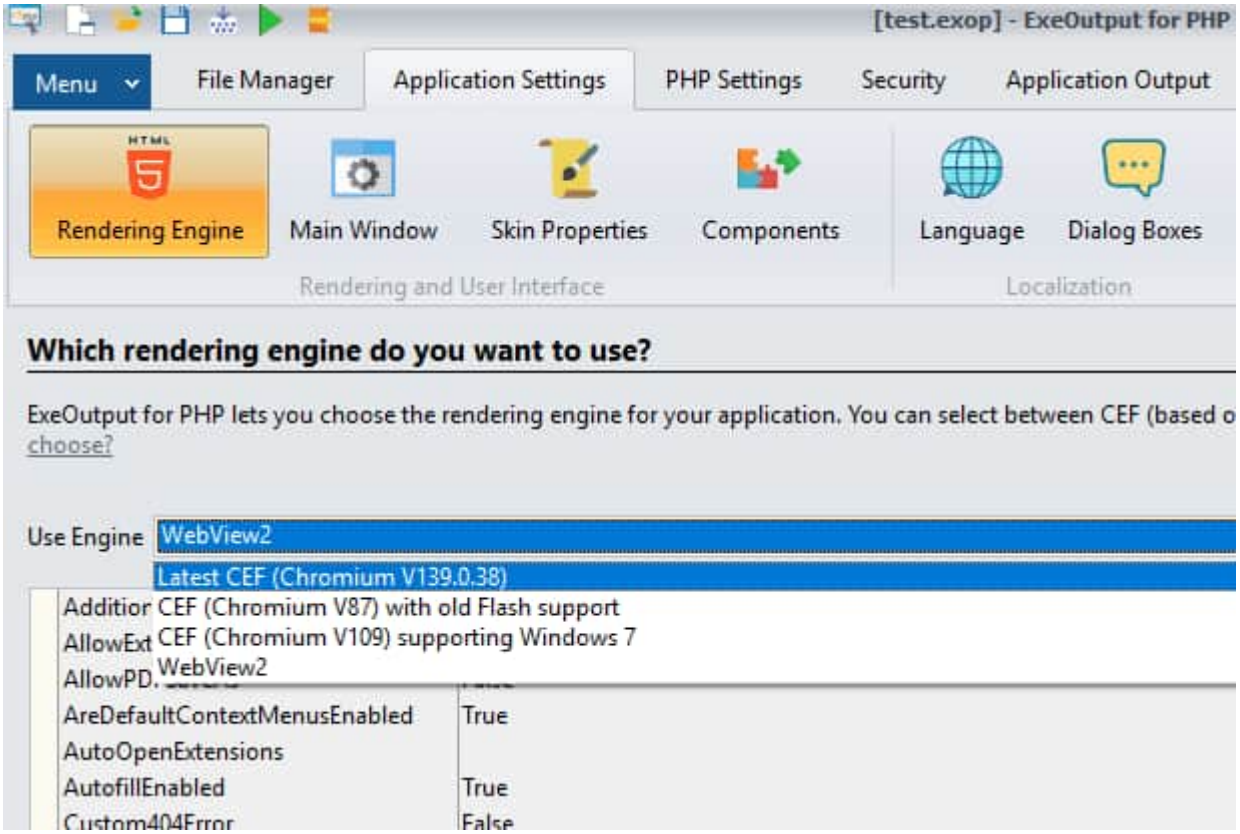
- Introduction to Scripting
- Calling HEScript Functions

8. Application Settings

8.1 Choosing a Rendering Engine

ExeOutput for PHP gives you the flexibility to choose between two powerful HTML rendering engines for your application: **Microsoft Edge WebView2** and the **Chromium Embedded Framework (CEF)**. Both are based on the Chromium project, ensuring modern web standards compatibility, but they differ significantly in how they are integrated and distributed.

Your choice of engine impacts your application's final size, its dependencies, and some of its features, such as media playback. To select an engine, navigate to **Application Settings → Rendering Engine**.



8.1.1 Comparison: WebView2 vs. CEF

Here is a summary of the key differences to help you decide which engine is best for your project:

Feature	Microsoft Edge WebView2	Chromium Embedded Framework (CEF)
Core Technology	Microsoft Edge (Chromium)	Embedded Chromium (Version 139.0.38 in ExeOutput 2025)
Distribution	Uses the WebView2 runtime installed on the system.	All framework files are bundled inside your EXE.
Application Size	**Very small** (approx. 20 MB).	**Large** (approx. 150 MB).
Dependencies	Requires the WebView2 runtime on the end-user's PC.	**None**. Creates a fully self-contained, portable application.
Updates	"Evergreen": automatically updated by Windows Update.	Bundled version is updated only when you recompile with a new ExeOutput version.
Media Codecs (MP4/H.264)	**Supported by default.**	**Not supported by default** due to licensing. Requires a special licensed build.

8.1.2 When to Choose WebView2

We recommend **WebView2 for most new projects**. Choose it if: * You want the **smallest possible application size** for faster downloads and installation. * You need to support modern web features and want your app's rendering engine to be **automatically updated with the latest security patches** by the operating system. * Your application requires **MP4/H.264 video playback** without needing special licenses.

!!! warning Known Issue: File Uploads in Forms There is currently a bug in the Microsoft Edge WebView2 engine that may prevent file uploads (`<input type="file">`) from working correctly with PHP (see bug report here).

```
As a workaround, you can either use the CEF engine (which does not have this issue) or implement a native file selection dialog as explained in the topic on [how to select local files with PHP](choosingfilesupload.md). This requires some additional code but provides a reliable solution.
```

8.1.3 When to Choose CEF (Chromium Embedded Framework)

CEF is an excellent choice when portability is your top priority. Choose it if: * You need to create a **fully self-contained and portable application** with zero external dependencies. * You require precise control over the exact version of the Chromium engine used by your application. * You are targeting environments where you cannot guarantee the presence of the WebView2 runtime and prefer not to handle its installation.

8.1.4 Engine Configuration

Both engines offer a wide range of properties that can be configured in the **Application Settings → Rendering Engine** section of ExeOutput for PHP.

Configuring WebView2

The WebView2 engine provides several options to control its behavior, such as context menus, autofill, and developer tools.

👉 Learn more about configuring the WebView2 engine

Configuring CEF

The CEF engine offers extensive customization, from command-line arguments to fine-grained control over caching, security, and JavaScript.

- **MP4/H.264 Support:** The default CEF build only supports open-source codecs like WebM and Ogg. Playback of licensed formats like MP4/H.264 requires a special build. If you need this, we recommend using WebView2 or contacting us for a licensed CEF build (requires a valid MPEG license from Via Licensing Alliance).
- **Flash Support:** For legacy projects requiring Adobe Flash, an older CEF version with Flash support is available. However, we recommend using the modern Ruffle.rs alternative.

Learn more about configuring the CEF engine

8.1.5 Distributing Applications with WebView2

Because WebView2 relies on a runtime component, you need to be aware of its distribution model.

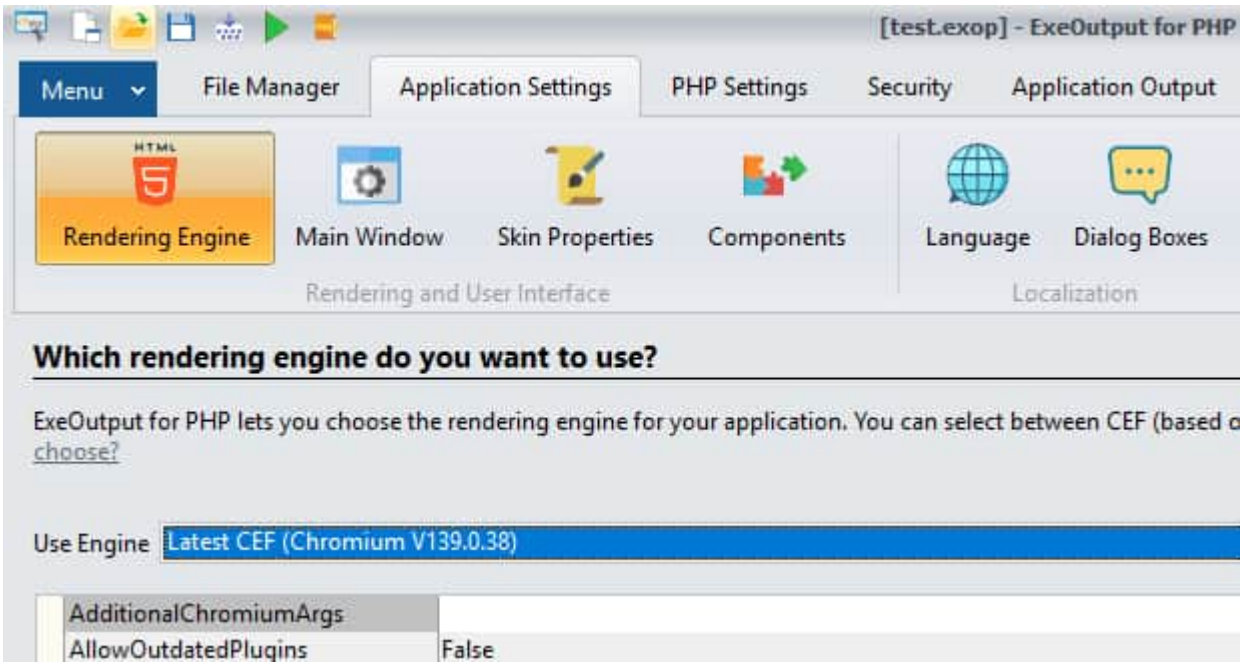
- The WebView2 runtime is **included by default on Windows 11** and most recent versions of Windows 10.
- For older systems (Windows 7, 8.1, or older Windows 10), the runtime may need to be installed.
- ExeOutput for PHP includes the official "Evergreen Bootstrapper" installer (`MicrosoftEdgeWebview2Setup.exe`) in its `Redist` sub-directory. You should **bundle this installer with your application's setup program** to ensure a seamless experience for all users.

> **Note**
>
> For more details, please refer to the official Microsoft documentation: Distribute the WebView2 Runtime

8.2 Chromium Embedded Framework (CEF)

ExeOutput for PHP includes the **Chromium Embedded Framework (CEF)**, an open-source project that brings the power of the Chromium browser engine directly into your applications. This allows you to build rich user interfaces with **HTML, CSS, and JavaScript**.

In ExeOutput for PHP, you can choose **Latest CEF** here: **Application Settings → Rendering Engine**.



8.2.1 Overview

CEF is a good choice if your application requires:

- **Self-contained distribution**: CEF is bundled inside your application. No external runtime (like WebView2) is needed.
- **Fine-grained customization**: ExeOutput exposes a wide range of properties to control caching, rendering, JavaScript behavior, and more.
- **Stable, well-tested technology**: CEF is cross-platform and widely used, making it reliable for embedded browsers.

8.2.2 CEF Properties

In ExeOutput for PHP, you can configure many Chromium options under **Application Settings → Rendering Engine → Latest CEF (Chromium)**.

AdditionalChromiumArgs

Add custom command-line arguments for Chromium. Example:

```
--disable-accelerated-video
```

Use spaces to separate multiple arguments.

AllowOutdatedPlugins

Disables the Chromium security feature that blocks outdated plugins.

> **Warning**
>
> Allowing outdated plugins exposes users to vulnerabilities.

ApplicationCache

Controls whether the Application Cache can be used (`STATE_ENABLED` , `STATE_DISABLED`).

AutoOpenExtensions

Lets compiled files open in their native applications when accessed by users.

- Extensions must be listed, separated by semicolons (e.g., `.docx;.xlsx`).
- Files are extracted temporarily to disk (potential security risk).
- Ignored for **PDF** if the built-in PDF Viewer is enabled, and for **DOCX** if the built-in Word Viewer is enabled.

BackgroundColor

Sets the browser background color before content loads. Only the RGB components are used.

CaretBrowsing

Controls whether the caret (text cursor) is displayed and can be used for navigation.

Custom404Error

Displays the custom 404 error page defined in *Dialog Boxes*.

- `True` : Returns `200 OK` with your custom error page.
- `False` : Returns standard `404 Not Found` .

CustomUserAgent

Defines a custom User-Agent string.

Databases

Enables or disables the use of web databases.

DefLocale

Sets the Chromium locale (e.g., `fr` , `en-US`). Locales are in the `CEFRuntime\locales` folder.

DeveloperTools

Enables/disables Chromium DevTools. End users can open them via the browser's context menu.

DevToolsPopup / DevToolsPort

- **DevToolsPopup**: Allows DevTools in a separate popup window.
- **DevToolsPort**: Sets the port for remote debugging (e.g., `9000`). Open `http://localhost:9000` in Chrome/Edge.

DisableAccelerated2DCanvas

Disables GPU acceleration for 2D canvas rendering.

DisableAlertDialogWorkaround

Disables the workaround that prevents status bar flicker with JavaScript dialogs.

DisableBuiltInKeyHandling

Disables built-in Chromium key handling. Can be useful for full control of keyboard input.

DisableDragDrop

Blocks drag-and-drop of resources.

DisableFindText

Disables the “Find Text” command.

DisableImgDragDrop

Prevents images from being dragged out of the browser (avoiding leaks of embedded images).

DisableLocalCache

Controls whether caching and temporary files are allowed. By default, they are stored in the **User Data** directory.

DisableSafeBrowsing

Turns off Google Safe Browsing.

DoNotTrack

Enables or disables the “Do Not Track” feature.

EnableGPUPlugin

Controls GPU-related plugins. Should normally stay disabled.

EnableMediaStream

Enables WebRTC (audio/video streaming).

> **Warning**
>
> HTTPS is recommended for proper support.

EnableSpeechInput

Controls whether the Speech Input API is allowed.

FileAccessFromFileUrls

Controls whether local file URLs can access other file URLs.

ForbidDownloadMimeType

Blocks downloading of specific MIME types (still playable). Example:

```
application/pdf;audio/mp3;audio/ogg;video/mp4
```

ImageLoading

Allows or blocks image loading.

ImageShrinkStandaloneToFit

Defines how standalone images are resized to fit the window.

Javascript

Controls whether JavaScript execution is allowed.

JavascriptAccessClipboard

Enables scripts to access the clipboard.

JavascriptCloseWindows

Allows or blocks `window.close()` in scripts.

JavascriptDomPaste

Allows pasting DOM content via JavaScript.

JavascriptOpenWindows

Controls whether scripts can open new windows.

LocalStorage

Enables LocalStorage.

LogSeverity

Sets Chromium logging level:

- `LOGSEVERITY_DEFAULT`
- `LOGSEVERITY_VERBOSE`
- `LOGSEVERITY_INFO`
- `LOGSEVERITY_WARNING`
- `LOGSEVERITY_ERROR`
- `LOGSEVERITY_DISABLE`

If enabled, logs are written to `cefdebug.log` in the EXE folder.

MultiThreadingMode

Configures CEF multithreading for file reading. Disabling may improve compatibility but reduce responsiveness.

MuteAudio

Mutes all audio output.

PersistSessionCookies

Stores session cookies across app sessions.

Plugins

Enables or disables plugins.

SendReferrer

Controls whether the HTTP Referrer header is sent.

SmoothScrolling

Enables smooth scrolling.

TabToLinks

Allows the Tab key to navigate between links.

TextAreaResize

Enables resizing of `<textarea>` elements.

UniversalAccessFromFileUrls

Grants file URLs access to other URLs.

WebGL

Enables/disables WebGL rendering.

WebSecurity

Controls enforcement of the Same-Origin Policy.

> **Warning**
>
> Disabling removes security restrictions and is unsafe.

WindowlessFrameRate

Ignored by ExeOutput.

> **Notes**
>
> • Most settings are either boolean (`True` / `False`) or enumerations (`STATE_ENABLED` , `STATE_DISABLED` , `STATE_DEFAULT`).
>
> • Default values may vary depending on Chromium builds used in ExeOutput.

8.3 WebView2 Rendering Engine

Your app made with ExeOutput for PHP can utilize the Microsoft Edge WebView2 control to display application content. WebView2 is powered by the same Chromium engine as the Microsoft Edge browser, ensuring your application supports modern web standards and technologies.

8.3.1 Overview

Using WebView2 offers several advantages:

- **Modern & Evergreen**: Your application will benefit from an up-to-date version of Chromium with regular platform updates and security patches. This ensures compatibility with the latest HTML5, CSS3, and JavaScript features.
- **Enhanced Performance**: WebView2 provides robust performance and speed for complex web applications.
- **Access to Native Features**: It allows for deep integration between your web code and the native capabilities of the Windows operating system.

To use WebView2, your end-users must have the **WebView2 runtime** installed. If it is missing, your application will not start and will display an error message.

- The WebView2 runtime is included **by default** on **Windows 11** and recent versions of **Windows 10**.
- On **Windows 7**, the runtime is automatically installed if Microsoft Edge is installed and Windows Update runs regularly.
- On **older versions of Windows** (such as Windows 8.1), the runtime will likely need to be installed manually using the evergreen installer.

The official evergreen installer, `MicrosoftEdgeWebview2Setup.exe`, is available in the `Redist` sub-directory of your ExeOutput for PHP installation. You should bundle this installer with your application's own setup program to ensure a smooth user experience.

> **Note**
>
> For more details on distributing the WebView2 runtime, please refer to the official Microsoft documentation: Distribute the WebView2 Runtime

8.3.2 WebView2 and Cookie Storage

When using the WebView2 engine, ExeOutput for PHP sets the application's internal domain to `https://heserver.example/`. This is different from the `https://heserver/` domain used by the CEF engine.

This change is necessary due to the stricter security policies of the underlying Microsoft Edge browser. Modern browsers, including Edge, are increasingly limiting cookie functionality for non-standard domains. Using a domain that resembles a real-world domain, like `.example`, ensures that cookies, sessions, and other local storage mechanisms function correctly and reliably within your application.

If you were to use a simple hostname like `heserver` with WebView2, you might encounter issues with session persistence and cookie-based authentication. The `.example` TLD is reserved for documentation and testing purposes, making it a safe and appropriate choice for this internal application domain.

8.3.3 WebView2 Properties

You can configure how the WebView2 engine behaves in your ExeOutput application using the following settings.

AdditionalChromiumArgs

Add custom Chromium command-line switches. Example to disable GPU-accelerated video:

```
--disable-accelerated-video
```

Separate multiple switches with spaces.

AllowExternalDrop

If enabled, allows users to drag items **from outside** (e.g., Windows Explorer) into the WebView.

> **Note**
>
> If `DisableDragDrop` is enabled, it overrides this and blocks drag/drop entirely.

AllowPDFSaveAs

Controls whether the **Save As** command is available in the built-in PDF viewer.

AreDefaultContextMenusEnabled

Enables the default WebView2 context menus (right-click menus). Disable to remove them and rely on your own context menu.

AutoOpenExtensions

Allows compiled files to open in their native applications when users click links to them.

> **Note**
>
> • Provide a semicolon-separated list of file extensions (e.g., `.docx;.xlsx`).
> • Files are temporarily extracted to disk, which may allow copying—review security implications.
> • Ignored for **PDF** when the built-in PDF Viewer is enabled

AutofillEnabled

Enables or disables form autofill.

> **Warning**
>
> There is no support for storing passwords securely.

Custom404Error

Uses the **custom 404 page** defined in *Dialog Boxes* for missing pages.

- If `True`: a `200 OK` status is returned and your custom page is shown.
- If `False`: a standard `404 Not Found` is returned and the browser's default error page appears.

CustomUserAgent

Sets a custom User-Agent string.

DefLocale

Specifies the Chromium locale (language). Locales are in `CEFRuntime\locales`. Example:

```
fr
```

DeveloperTools

Enables or disables Chromium DevTools. Users can access them from the internal browser's context menu.

DisableDragDrop

Blocks **all** drag-and-drop operations in the WebView (both internal and external). Use to prevent accidental extraction/leaks of embedded resources.

DisableFindText

Disables the **Find Text** command.

DisableImgDragDrop

Prevents dragging images out of the WebView (which otherwise saves the image to a folder and can leak compiled resources).

DisableLocalCache

Prevents the engine from writing cache and temporary files to disk. By default (when not disabled), a subfolder inside the **User Data** directory is used. See Deployment options.

ForbidDownloadMimeType

Blocks downloads—via context menu or default handlers—for specific MIME types while still allowing playback in media elements. Example:

```
application/pdf;audio/ogg;audio/x-mpg;audio/mp3;audio/mpeg;video/ogg;video/mp4;audio/webm;video/webm;audio/wav
```

IsBuiltInErrorPageEnabled

If enabled, WebView2's built-in **network/navigation error pages** are displayed on failures. Can be combined with `Custom404Error` depending on how you want 404/other errors handled.

Muted

Mutes all audio output from the WebView.

PDFDisableMoreOptionsMenu

Disables the "More options" (⋯) menu in the built-in PDF viewer to reduce access to actions such as download/print (depending on viewer UI).

PinchZoomEnabled

Enables pinch-to-zoom.

PrivateMode

Enables private/incognito mode for the WebView instance.

ShowDownloadDialog

Shows the default **download dialog** when a file download begins. Disable to suppress the dialog (downloads may be blocked unless your app handles them). Useful with the dedicated HEScript UserMain event OnDownloadComplete.

SingleSignOn

Enables single sign-on (SSO) where supported.

SwipeNavigationEnabled

Enables swipe gestures for back/forward navigation (touch devices/trackpads).

TrackingPrevention

Enables tracking prevention features.

8.3.4 Troubleshooting CORS Problems

If the DevTools console shows errors like:

```
Uncaught DOMException: Blocked a frame with origin "null" from accessing a cross-origin frame.
```

you can temporarily disable CORS checks by adding this switch to **AdditionalChromiumArgs**:

```
--disable-web-security
```

> **Info**
>
> Use only for debugging; do not ship with web security disabled.

8.4 Main Window Settings

The main window is the primary interface for your application. It displays your HTML pages and allows users to navigate your content. This page explains the options available for customizing its appearance and behavior.

8.4.1 General Properties

Window Caption

Sets the text that appears in the window's title bar. This text is distinct from the application title used by the Windows taskbar and Task Manager.

You can dynamically insert the current HTML document's title using the `%DOCTITLE%` variable. For example: `My Application - %DOCTITLE%`.

8.4.2 Window State and Size

Initial Window State

Determines how the window first appears at launch.

- **Standard Window**: Opens the window with default dimensions.
- **Maximized Window**: Opens the window maximized to fill the entire screen.
- **Custom-Sized Window**: Lets you specify a custom width and height for the window on startup.
- The **Auto-Sizer** button provides a visual way to set these dimensions. It opens a sample window that you can resize; ExeOutput for PHP then automatically populates the width and height fields based on your chosen size.

> **Note**
>
> If "Save Window Position and Size" is enabled, the application saves the window's last state (maximized or normal). You may need to reset your saved application settings for changes to this option to take effect.

Minimum Window Size

Sets the minimum allowed height and width for the main window, preventing users from resizing it smaller than these dimensions. Set to `0` to disable constraints.

8.4.3 Window Behavior

Allow User Resizing

Determines whether users can resize the main window. Disabling this is not recommended, as it can negatively impact user experience.

Show Maximize Button

Shows or hides the window's standard "Maximize" button. This is useful for kiosk-style applications or to prevent a maximized window from being restored to its normal state.

Window Stays on Top

If enabled, the main window will remain on top of all other application windows.

Pop-ups Stay on Top

Ensures that pop-up windows always appear on top of the main application window. If disabled, pop-ups may appear behind the main window and will get their own taskbar button.

Save Window Position and Size

When enabled, the application saves the main window's size and position when the user closes it. On the next launch, the window is restored to its last known state. This is highly recommended for a better user experience.

- If disabled, the application will always open with the initial state and dimensions defined here.
- To save only the size but not the position, enable this option along with "Always Center Window".

Always Center Window

Forces the window to be centered on the primary screen at startup. This overrides any saved position from a previous session.

> **Warning**
>
> This option is not recommended for users with multiple monitors.

8.5 UI Skin Properties

8.5.1 Application Skins

A key feature of ExeOutput for PHP is its support for **skins**. Skins allow you to change the entire look and feel of your application's windows and controls with a single click. This is similar to Windows themes, but your application will use its own built-in skinning engine, ensuring a consistent appearance regardless of the user's system settings.

To apply a skin, simply select one from the list. A preview will be displayed.

> **Skins are Compiled In**
>
> ExeOutput for PHP includes several ready-to-use skins. The selected skin file is compiled directly into your application, so you do not need to distribute it separately.

For advanced customization, a **Skin Editor** is available as a separate download (via the Web Update utility). With the Skin Editor, you can modify existing skins or create new ones from scratch. Click **Open Skin Editor** to launch it if you have it installed.

8.5.2 Window Chrome Properties

These options modify the window's frame, also known as the "non-client area" or "chrome".

Borderless Window : Removes the window's title bar and border entirely. If you enable this, you must provide a custom UI element (e.g., a button) for users to close the application, as they will have to rely on `Alt+F4` otherwise.

Hide All Caption Buttons : Removes the minimize, maximize, and close buttons from the title bar, while keeping the title bar itself visible. You should provide an alternative way for users to close the application.

Hide System Menu Icon : Hides the application icon that appears in the top-left corner of the title bar. The minimize, maximize, and close buttons remain visible.

8.5.3 Skinning Behavior

Disable Window Skinning : Disables skinning for the main window frame only, causing it to use the standard Windows appearance. Skinned controls inside the window (buttons, etc.) are not affected.

Disable Skinning for Dialogs : Prevents skins from being applied to standard system dialogs, such as message boxes or the open/save file dialogs.

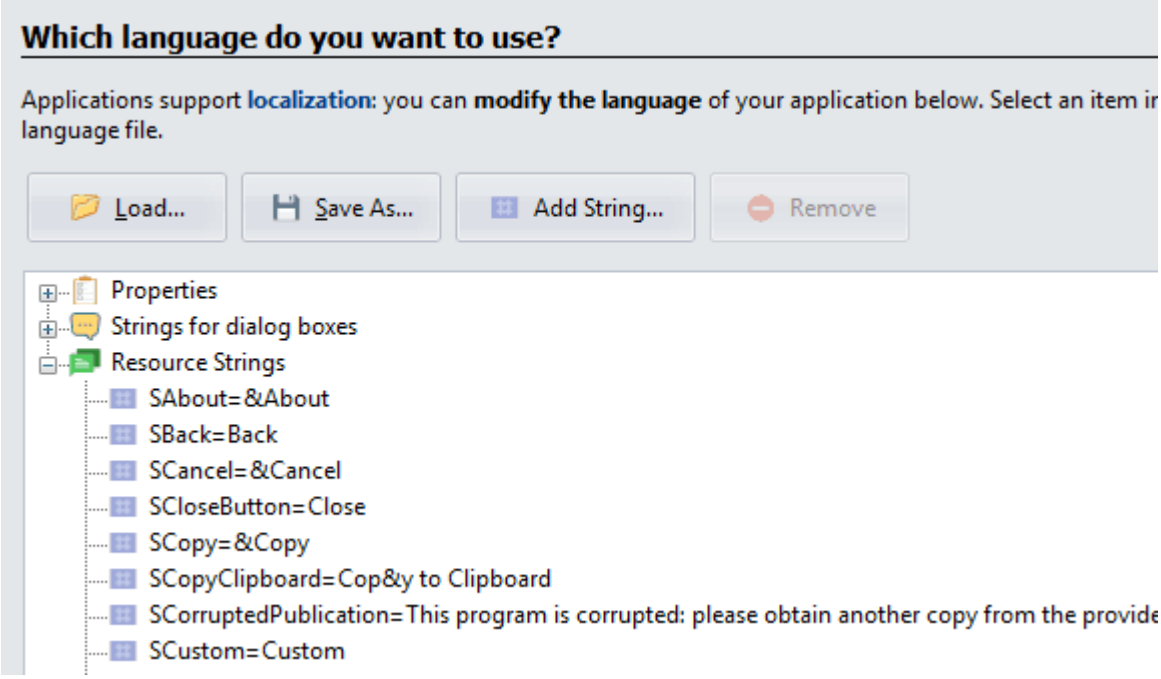
8.6 Application Components

This page in ExeOutput for PHP allows you to edit your application's user interface using **Components**.

👉 Please refer to the section dedicated to User Interface and Components.

8.7 Language and Localization

ExeOutput for PHP allows you to **localize** your application, translating built-in text and creating multilingual interfaces. This means that dialog captions, button labels, and other interface elements can be displayed in any language you choose. This section explains how to manage your application's text strings and use language files.



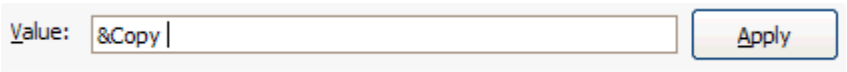
8.7.1 Resource Strings

Resource strings are named text constants used to translate UI elements like messages, menus, buttons, and window titles. For example, to create a French and English version of your application, you don't need to create two separate versions. Instead, you can define a list of resource strings for each language.

In the ExeOutput for PHP user interface (for example, in button captions), you can reference a resource string by wrapping its name in percent signs: `%SAbout%`. At runtime, the application replaces these identifiers with the string value for the currently selected language.

Managing Resource Strings

- To **edit a string**, select it from the list to open the editor. Modify the **Value** field and click **Apply** to save your changes.
- To **add a new string**, click **Add String**. You will be prompted for a name, which must be unique, start with an `S`, and contain only alphanumeric characters.
- To **remove a string**, select it and click **Remove**. You can only remove custom strings that you have added.



Using Resource Strings in Code

You can access resource strings directly from your code:

- **PHP:** Use the built-in `exo_get_resstring` function:

```
<?php echo exo_get_resstring('SPubCopyright'); ?>
```

- **HTML/JavaScript:** Use the asynchronous `exeoutput.GetString` function, which returns the value to a callback:

```
<div id="demo"></div>
<script>
  function displayString(content) {
    document.getElementById('demo').innerHTML = '<p>' + content + '</p>';
  }
  exeoutput.GetString("SPubCopyright", displayString);
</script>
```

8.7.2 System Page Strings

These special strings are used exclusively for translating the content of built-in system dialog boxes.

They are referenced using the syntax `[#ID]`, where the `ID` typically starts with a `Y` (for example, `[#YAbout]`). Unlike resource strings, these are **not available at runtime**. They are compile-time constants; ExeOutput for PHP replaces the `[#ID]` placeholders with their values when the application is built.

You can edit these strings in the same way as regular resource strings.

8.7.3 Managing Language Files

You can import and export all of your project's strings to and from **language files** (`.exol` extension) using the **Load** and **Save** buttons. This allows you to reuse translations across multiple projects or share them with other developers.

- All strings are saved directly within the project file (`.exop`), so you do not need to keep a separate language file for each project.
- If you plan to distribute a language file, be sure to fill out its **properties**. The **locale** is the numeric Windows Language Code Identifier (LCID). You can find a list of these identifiers on the Microsoft website.
- You can set a default language file in the Environment Options to have it automatically loaded every time you create a new project.

8.7.4 Automatic Resource Strings

ExeOutput for PHP automatically creates several resource strings at compile time based on your project settings. These can be used in your code just like any other resource string.

- `SPubTitle`: The title of your application.
- `SPubHomepage`: The default homepage URL.
- `SPubAuthor`: The author's name.
- `SPubCopyright`: The copyright notice for your application.
- `SPubEMail`: The author's email address.
- `SPubVersion`: The application's file version.
- `SPubProductVer`: The application's product version.

8.8 Application Settings - Dialog Boxes

All dialog boxes displayed by applications are HTML pages that function as custom HTML dialogs, such as the About dialog box.

You can edit the HTML code of these dialog boxes to customize or translate their contents. However, modifying scripts or forms within these HTML pages is not recommended. These pages are managed by internal scripts that govern application functionality.

☑ To edit a dialog box or related resource, select it from the list and click **Edit** (or double-click it). The internal HTML editor will appear. Press **OK** to save your changes. It is your responsibility to ensure the pages function correctly after modifications.

To import or export a dialog box or related resource, select it from the list and click **XML**. Choose **Import/Export**, and you will be prompted to enter the filename for the XML file. Pages are stored as CDATA items in XML format, allowing editing with any XML editor (or even Notepad).

When you import a page, the existing one is overwritten without a prompt.

☑ Dialog boxes utilize resource strings, termed "**Strings for Dialog Boxes**," for easier localization. All references to these strings are directly replaced when ExeOutput for PHP compiles the application. To **insert a reference** to a string for a dialog box, use this syntax: `[#ID]`, where `ID` is the string's ID (IDs always begin with `Y`). For example, `[#YAbout]` will be replaced with the value of `YAbout`, which is "About".

☑ By default, dialog boxes can be **resized by users**. To prevent this, add the following meta tag to the `<head>` section of the system HTML page's code:

```
<meta name="WindowNoResize" content="1">
```

8.9 Startup and Exit Messages

This page allows you to configure message boxes that are displayed to the user at specific times. You can configure two distinct messages: one that appears when the application starts, and one that appears when it closes.

8.9.1 Startup Message

This message is displayed immediately after the application is launched. It is a confirmation prompt with "Yes" and "No" buttons that asks the user if they wish to proceed with running the application. If the user clicks "No", the application will exit.

Example: `Welcome! This application requires a minimum screen resolution of 1024x768. Do you want to continue?`

8.9.2 Exit Message

This message is displayed immediately after the user closes the application, just before the process terminates. It is a simple informational message with an "OK" button.

Example: `Thank you for using My Application! Visit us at www.example.com.`

8.9.3 Configuration Notes

- **Disabling Messages:** To disable either message, leave its corresponding text field blank.
- **Line Breaks:** To insert a line break within a message, use the special character sequence `/§`. For example: `First line/§Second line`.
- **Appearance:** These messages are standard system dialogs and are not affected by the application's selected skin. This is because they are displayed before the skin engine is loaded (for the startup message) or after it has been unloaded (for the exit message).

9. PHP Settings

9.1 PHP Settings - Main Settings

This page lets you configure the main settings for the PHP runtime.

9.1.1 PHP Version

Use the combo box to select your desired PHP version.

👉 More information is available about the available PHP versions and their system requirements.

> **Warning**
>
> Changing the PHP version will reset the `PHP.INI` file and its associated PHP extensions.

9.1.2 Custom Virtual "Data" Subfolder Path

By default, the "Data" virtual subfolder is located in the same folder as the application's EXE. For example, if your EXE file is at `E:\my folder\myprogram.exe`, the path to the "Data" subfolder will be `E:\my folder\data`.

You can use the `%EXEDIR%` placeholder to include the path to your application's executable file directory. The resulting path does not include a trailing backslash (). For example, you can enter `%EXEDIR%\MySubfolder` or `%EXEDIR%\wwwroot`.

For security reasons, such as placing virtual files in an inaccessible folder, you may want to **choose a custom name** for the "Data" virtual folder. The ExeOutput for PHP virtualization engine allows you to **choose any virtual folder, even one on a non-existent drive**. To do this, enable the option and enter an absolute path, such as `X:\My Application\123456\Data`.

> **Warning**
>
> Ensure you choose a path that is unique to your application.

9.1.3 Application Startup URL

You can define the full URL that the application should load at startup. If you leave this field blank, the application will use the URL of the homepage selected in the File Manager.

This feature is useful for using frameworks or displaying remote websites. For instance, some PHP frameworks like Kohana use the `PATH_INFO` server variable to display pages. A PHP application can be started with `index.php/Welcome`, where `/Welcome` tells the framework to show the "Welcome" view.

For another example, if you enter `http://www.exeoutput.com`, the application will start and display the ExeOutput website. This is useful for creating a custom web browser.

Homepage Arguments

To customize the homepage at runtime based on command-line arguments, for example, you can use the following code (to be added to the UserMain HEScript script):

```
procedure OnStartMainWindow;
begin
 // When the main window is going to be displayed (just before the homepage is shown).
 if ParamStr(1) = "/mode=debug" then
 begin
   SetGlobalVar("HomePage", "index.php?debugmode=1", false);
```

```
  end;
end;
```

9.1.4 Additional PHP File Extensions

Define additional file extensions that should be treated as PHP scripts and passed to the PHP interpreter. Separate multiple extensions with a semicolon (`;`), for example: `.ex1;.ex2` .

For example, if you enter `.png` , all PNG files will be passed to the PHP interpreter.

9.1.5 Use Secure HTTPS for Internal Protocol

ExeOutput for PHP uses its own internal protocol for displaying web content in the web browser engine. The base URL depends on the selected rendering engine: `https://heserver/` for CEF and `https://heserver.example/` for WebView2. The non-secure `http://` and the alias `ghe://` are also available. By default, the homepage will be opened at startup using the secure protocol unless you deactivate this option.

9.1.6 Enable Custom Router Handler in HEScript (OnRouterURL event)

Enables the router handler script at runtime, allowing you to configure custom redirections for all requests directly in HEScript.

👉 Read more about the custom router handler in HEScript

9.1.7 Reroute All Non-File Requests to Homepage

> **Info**
>
> This option is a simpler alternative to using the custom router handler in HEScript.

This adds support for pretty URLs in your PHP applications, similar to Apache's `mod_rewrite` in certain cases. All URLs that do not point to a file with an extension will be rewritten as if the application's homepage was requested. The homepage should, of course, be the `index.php` router script of your framework.

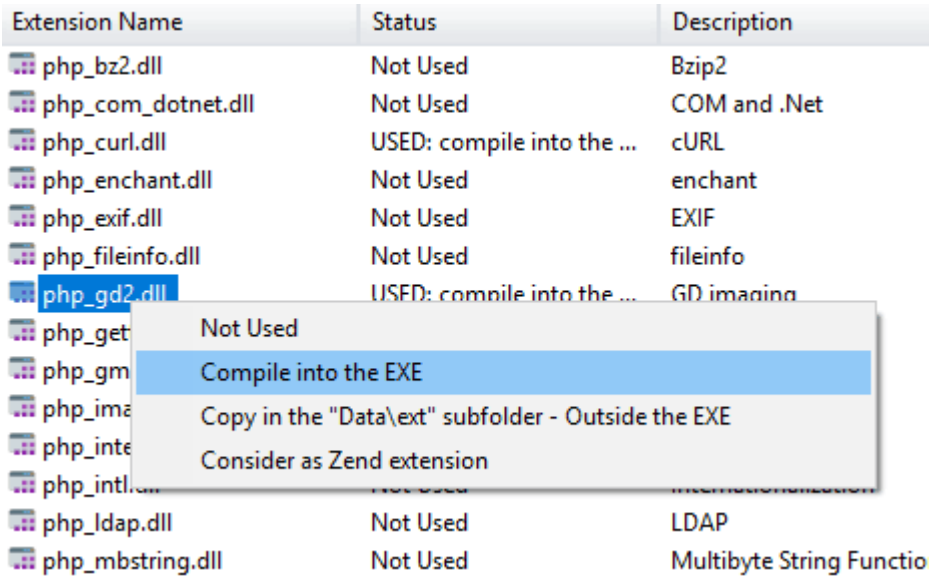
For instance, `https://heserver/Welcome` or `https://heserver.example/myapp/myaccount/1234/` would be rerouted, depending on the selected rendering engine.

👉 See also PHP Debugging Options and PHP.INI.

9.2 PHP Settings - PHP Extensions

PHP includes several extensions (for databases, graphics, XML, libsodium encryption, etc.) that you can use in your PHP applications. ExeOutput for PHP can compile these extensions directly into the EXE, or you can leave them as external files. In the latter case, they must be deployed with your EXE file.

You must specify **which extensions your application requires** in the **extension list**. There is no need to edit the `php.ini` file yourself; ExeOutput for PHP manages the "Extensions" section automatically.



To specify a required extension, right-click it in the list and choose one of the two options:

- **Compile into the EXE**: The extension will be compiled into the EXE file, so there is no need to deploy it separately.
- **Leave outside the EXE**: The extension will be left as an external file. In this case, it must be placed in the `Data\ext` subfolder. For instance, if your EXE is located at `C:\My Documents\`, the subfolder would be `C:\My Documents\Data\ext`. ExeOutput for PHP will automatically copy the extensions to this folder during compilation. When you deploy your application, **this folder structure must be preserved**.

You can add more PHP extensions to the default set provided with the PHP distribution that ships with ExeOutput for PHP. Using Windows Explorer, navigate to the ExeOutput for PHP installation folder and place the new DLL files in the `PHPRuntimeXX\ext` subfolder. Restart ExeOutput for PHP, and the new extensions will be listed automatically.

> **Info**
>
> Some extensions may not be compatible with ExeOutput for PHP. Note that some extensions require additional files, drivers, or separate installations on the user's computer. Please refer to their respective documentation for deployment information.

If an extension is a Zend extension, right-click it and choose **Consider as Zend extension**. ExeOutput for PHP will then load it properly. This option is mandatory for some extensions, such as the ionCube PHP Encoder.

Note

Some extensions have **dependencies**. ExeOutput for PHP automatically detects these DLL dependencies (provided they are in the corresponding `PHPRuntimeXX` subfolder) and will compile them into the EXE if you choose "Compile into the EXE" for the extension. Conversely, if you choose "Copy...outside the EXE", the dependency DLLs will be copied to the same folder as your EXE file.

For instance, the **cURL** extension requires additional dependencies such as `libeay32.dll`, `libssh2.dll`, `ssleay32.dll`, and `nghttp2.dll`. ExeOutput for PHP handles these DLLs for you when you use cURL in your PHP app.

Info

Please see the topic dedicated to cURL.

Tip

👉 Please refer to the sample using the GD2 library available in the General Demonstration, and to the sample using the cURL extension available in the General Demonstration.

9.3 PHP.ini Settings

PHP's initialization file, `php.ini`, is responsible for configuring many aspects of PHP's behavior. ExeOutput for PHP lets you configure the `php.ini` file that will be used by your application.

For more specific information, see the official PHP documentation: http://php.net/configuration.file

> **Warning**
>
> Some sections and values **are automatically managed by ExeOutput for PHP and should not be modified**.

You can search for a specific entry in the `php.ini` file using the **Find in PHP.INI** button.

The `php.ini` file is automatically compiled into the EXE. There is no need to deploy it separately.

The default `php.ini` file used by ExeOutput for PHP is available in its `PHPRuntimeXX` subfolder.

> **Warning**
>
> Do not place a custom `php.ini` file in your source folder.

9.4 PHP Settings - String Protection

Since PHP scripts must be unpacked into memory to be interpreted by the PHP runtime, it may be possible for a skilled hacker to extract portions of compiled PHP files. This could grant them access to any string stored in plain text within the PHP code, such as private passwords, database login info, etc.

To make this task more complicated and time-consuming, ExeOutput for PHP includes a **String Protection** feature.

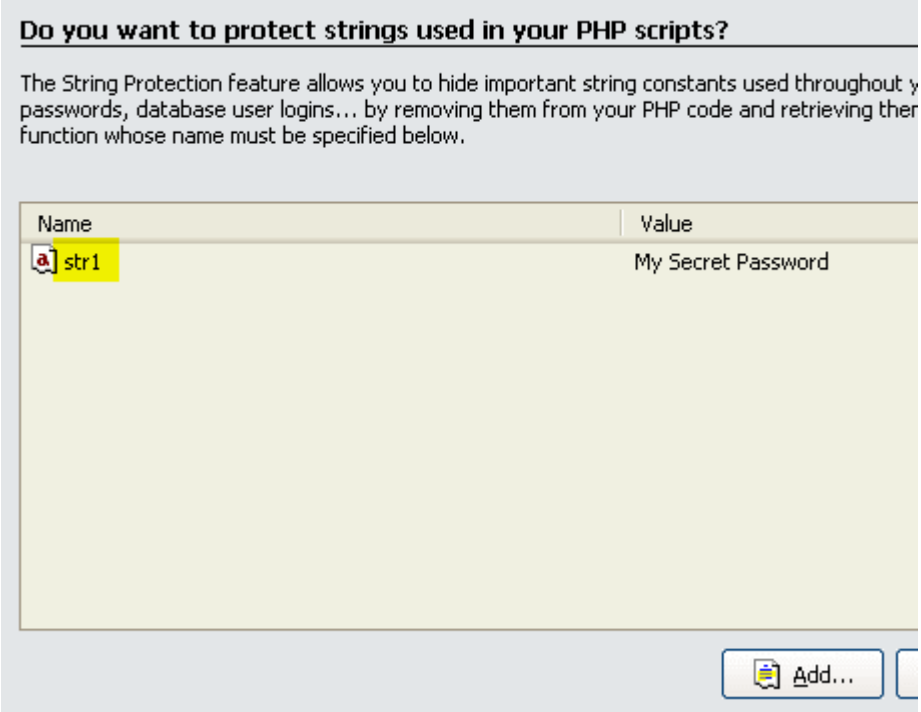
The String Protection feature allows you to **hide string constants** by replacing them in your PHP code with **a call to a special PHP function**. This way, the strings do not appear in plain text in your PHP code.

To use this feature, define the strings you want to protect on the **PHP Settings -> String Protection** page. Each string is assigned a **unique identifier**. Then, replace all instances of those strings in your code with a call to the PHP function, using the corresponding identifier.

The **prototype of the PHP function** is:

```
string exo_get_protstring(string $stringid);
```

For instance, suppose we have a password, "**My Secret Password**", with the identifier "**str1**". The name of the PHP function is `exo_get_protstring`.



Instead of using this PHP code, which contains the password in plain text:

```
<?php
$pass = 'My Secret Password';
echo ("The password is: $pass");
?>
```

...use the following code instead:

```
<?php
$pass = exo_get_protstring('str1');
echo ("The password is: $pass");
?>
```

> **Tip**
>
> **Advice:** You should use non-descriptive identifiers for your strings.

The **Comments** button lets you associate an optional comment with the selected string. This comment is for your reference only and is not compiled into the application.

9.5 PHP Settings - PHP Debugging

9.5.1 Display PHP Error Messages at Runtime

If your PHP scripts contain errors, the PHP runtime may return error and warning messages. If you enable this option, a message box will display errors as they occur. In the case of multiple errors or warnings, the application will ask, "Do you want to disable future error messages?"

Finally, the last error message is displayed in the browser so that users can see and report it. You can also customize the error page that is displayed using the "Error Message" entry in Dialog Boxes.

Notes:

- The `error_reporting` level can be configured in the `php.ini` file.
- You can customize the "Fatal PHP error" message, which is displayed when the PHP runtime fails to execute a script, using the `SPHPFatalErrorMsg` resource string.

9.5.2 Enable PHP Error Logging

This option automatically configures the error reporting settings in `PHP.INI` so that any PHP error is stored in a file named `php_errors.log`, located in the same folder as the application's EXE file.

> **Important**
>
> Enabling this option will override the previous one; PHP error messages will be logged to the file instead of being displayed in message boxes.

9.5.3 Enable Xdebug Debugger and Profiler

ExeOutput for PHP can enable and configure the well-known Xdebug PHP profiler to run with your application. Please refer to the official Xdebug documentation for further explanation on how to use it.

9.5.4 Store Requests and Errors in a Log File

When this option is enabled, the application writes all accessed URLs and any errors (such as PHP errors, warnings, or 404 "resource not found" errors) to a log file. This is similar to standard web server logs.

The **log file is stored in the same folder** as the output EXE file. For instance, if your EXE path is `C:\My Documents\Output\myprogram.exe`, the full path to the log file will be `C:\My Documents\Output\myprogram.log`.

> **Warning**
>
> ExeOutput for PHP does not automatically delete log files.

9.5.5 Disable OPcache Extension

By default, ExeOutput for PHP enables the OPcache extension to make your application run faster. You can disable it with this option if necessary.

9.6 PHP Settings - External HTTP Server

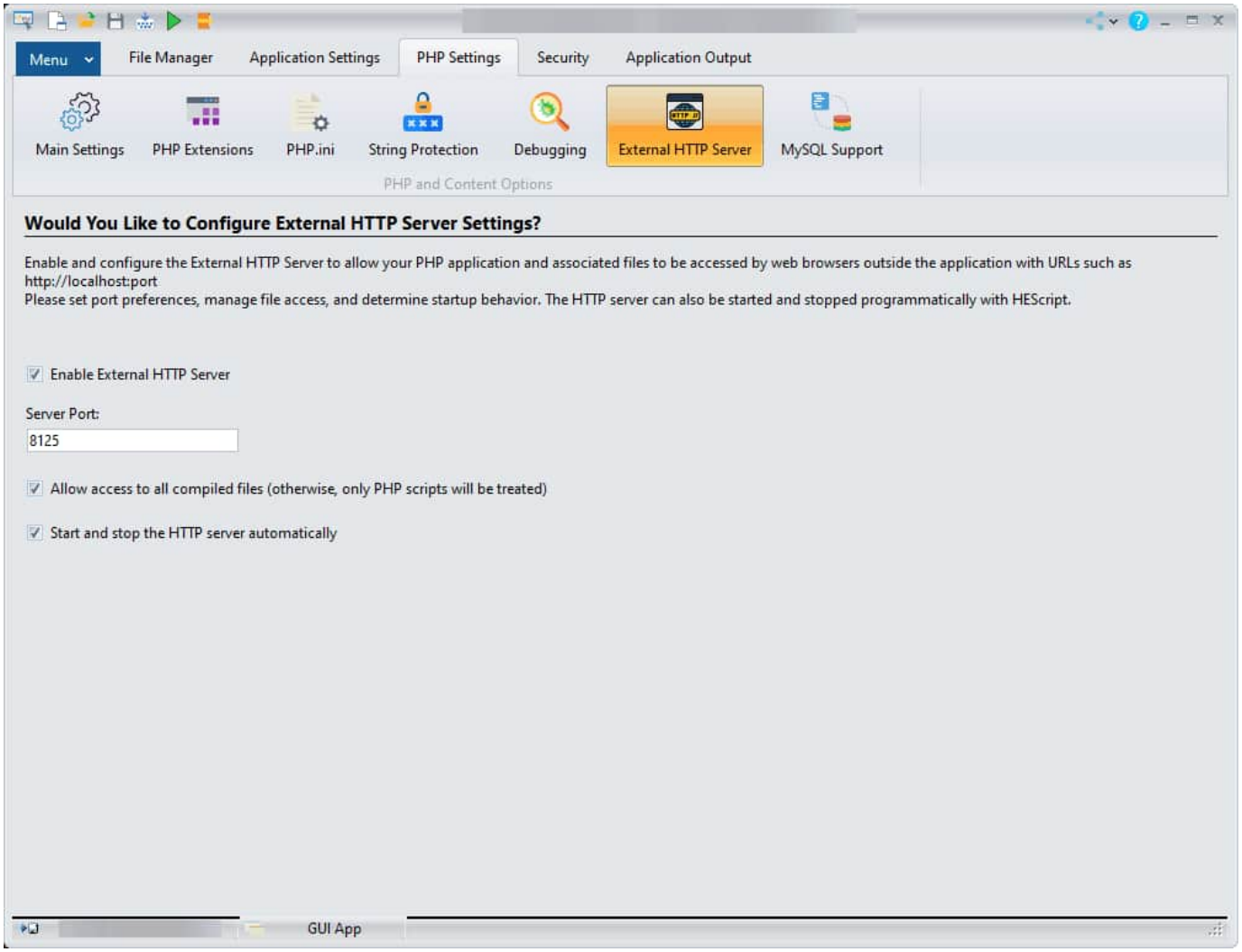
ExeOutput for PHP allows you to configure an external HTTP server, enabling access to your PHP application and its files from a web browser using a URL like `http://localhost:port`.

9.6.1 Why Use an HTTP Server?

This feature is useful for developers who need to test their PHP applications in a browser, or for those who want to let users interact with the application through their own web browsers. It can also allow multiple users to access the application concurrently (depending on your server setup).

> **Info**
>
> You can also start and stop the HTTP server programmatically with HEScript, as explained below.

9.6.2 Configuration Options



The following options are available for configuring the external HTTP server:

1. **Enable External HTTP Server**: Enables the external HTTP server.
2. **Server Port**: The port on which the external HTTP server will listen.

> **Warning**
>
> Do not use common ports like 80 or 8080, as some antivirus software may flag your application as potential malware.

3. **Allow access to all compiled files**: If checked, all compiled files (HTML, images, etc.) will be accessible. Otherwise, only PHP scripts will be served.
4. **Start and stop the HTTP server automatically**: If checked, the HTTP server will start and stop with the application. If unchecked, you must use HEScript to control the server.

> **Warning**
>
> If the specified port is already in use, a "Could not bind the server port" error will be displayed. However, the application will continue to run without the external server.

9.6.3 Starting and Stopping the Server Manually

You can control the HTTP server manually with HEScript commands. The following code, placed in your UserMain script, demonstrates how to do this.

```
procedure StartMyHTTPServer;
begin
  // Starts the HTTP server on the specified port (8142 in this case).
  StartHTTPServer(8142);
end;

procedure StopMyHTTPServer;
begin
  // Stops the HTTP server.
  StopHTTPServer;
end;
```

To start the server manually, call the `UserMain.StartMyHTTPServer` function. To stop it, call the `UserMain.StopMyHTTPServer` procedure.

9.7 MySQL and MariaDB Support

9.7.1 MySQL and MariaDB Support

ExeOutput for PHP provides tools to run a local MySQL or MariaDB database server for your PHP applications.

Please refer to the following sections for detailed instructions and information:

- Using Databases in Applications
- How to run a MySQL server with your PHP application

9.8 Redirection and Routing

9.8.1 How Redirection and Routing Work

The redirection feature in ExeOutput for PHP lets you redirect users from an old page to a new one. For frameworks, it can also redirect all requests to a single router script.

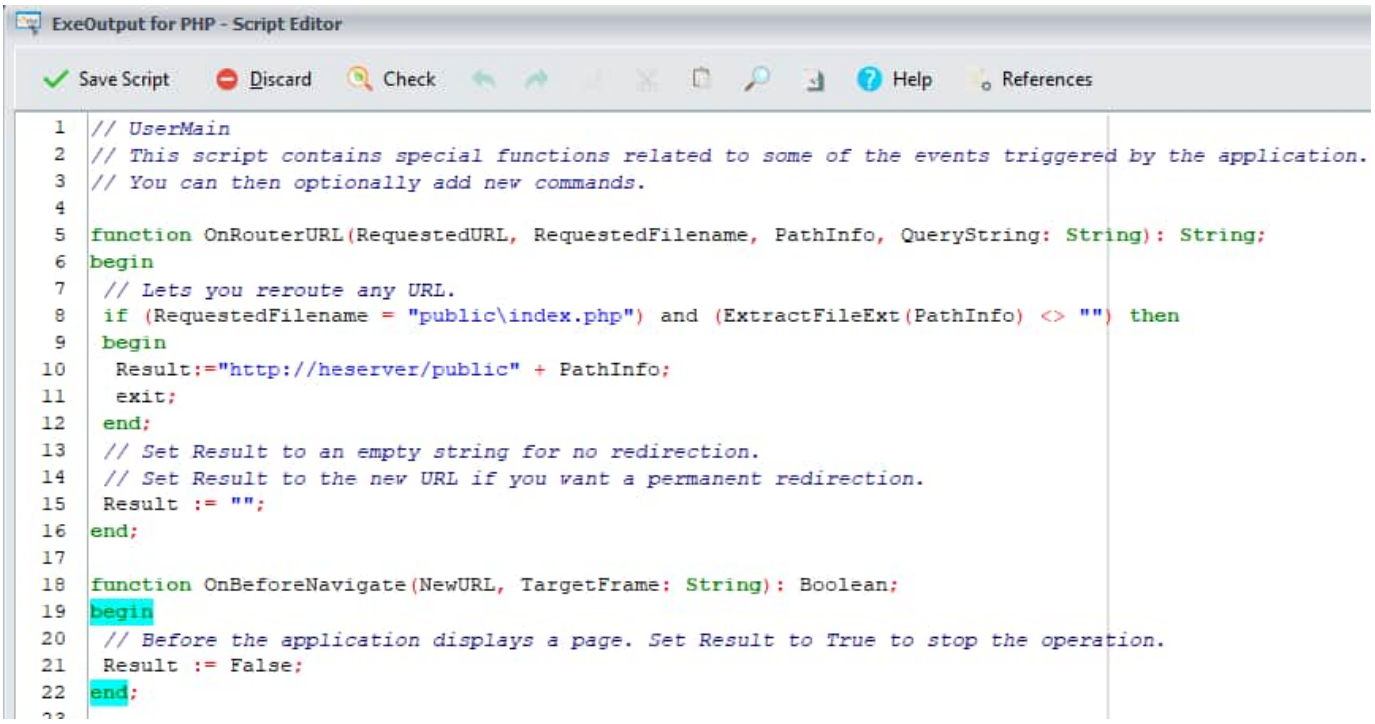
> **Warning**
>
> Redirection support is disabled by default. **You must enable this feature** in ExeOutput for PHP by going to PHP Settings => Main Settings and checking the **Enable custom router handler in HEScript (OnRouterURL event)** option.

> **Info**
>
> If you do not need redirection, it is best to leave this feature disabled, as it can slightly increase the internal server's processing time.

9.8.2 Adding Redirection Rules

Once redirection support is enabled, the `OnRouterURL` HEScript event, available in the UserMain script, is invoked for each server request.

Consider the `UserMain` script below:




```
function OnRouterURL(RequestedURL, RequestedFilename, PathInfo, QueryString: String): String;
```

The following parameters are passed to the event:

- `RequestedURL` : The full request URL.
- `RequestedFilename` : The virtual path to the requested file.
- `PathInfo` : Represents the value of the `PATH_INFO` variable as defined for PHP: any client-provided pathname information that trails the script filename but precedes the query string.
- `QueryString` : The query string, if any (the text after `?` in the URL).

The engine performs a redirection only if the function's `Result` is set to the new URL. To avoid redirection, return an empty string.

9.8.3 Redirection Example

> **Note**
>
> The base URL of your application depends on the rendering engine. This example uses `https://heserver/` (CEF). For WebView2, you would use `https://heserver.example/` .

The router script (which is also the homepage) is `public\index.php` . The framework in this sample generates URLs such as `https://heserver/public/index.php/assets/image.png` . Since the Chromium engine cannot resolve these URLs directly, we must rewrite them:

```
function OnRouterURL(RequestedURL, RequestedFilename, PathInfo, QueryString: String): String;
begin
  // Reroute any URL.
  if (RequestedFilename = "public\index.php") and (ExtractFileExt(PathInfo) <> "") then
  begin
    Result := "https://heserver/public" + PathInfo;
    exit;
  end;
  // Set Result to an empty string for no redirection.
  // Set Result to the new URL if you want a permanent redirection.
  Result := "";
end;
```

For instance, the script above transforms the URL: `https://heserver/public/index.php/assets/image.png` into: `https://heserver/public/assets/image.png`

9.8.4 See Also

- HTTP Basic Authentication

10. User Interface

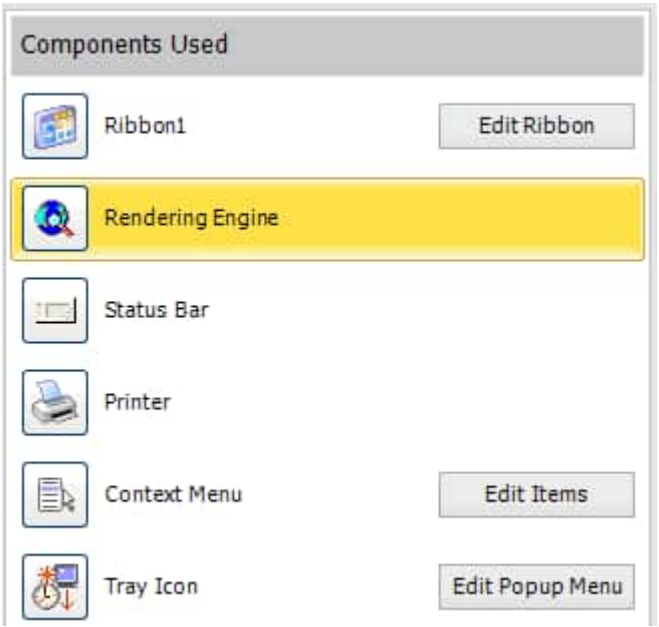
10.1 User Interface Components

ExeOutput for PHP includes a **visual editor that lets you create modern and complex user interfaces for your PHP applications**. To edit your application's interface, go to **Application Settings** => **Components**.

You will see two lists, **Available Components** and **Components Used**, with a **Properties Editor** on the right side.

10.1.1 Components Used

This list contains all of the customizable user interface components:



The following components are available by default and cannot be removed. Please see their corresponding topics for more help:

- Status Bar
- Printer
- Context Menu
- Tray Icon

Other optional components include:

- Menu Bar
- Toolbar
- Ribbon
- Timer (cron job)

You can add multiple toolbars if you wish; there is no hard-coded limit.

Each component has its own properties, which can be configured with the Properties Editor. To edit a component's properties, **select it in the list**.

> **Info**
>
> Some components have an **Edit** button that opens the associated UI editor.

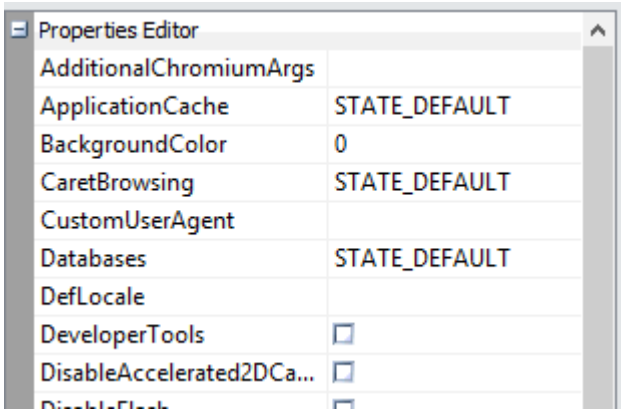
10.1.2 Adding a UI Component

You can add new UI components to your application. These components will make up your application's main window interface.

Available components include the **Menu Bar**, **Toolbar**, and **Ribbon**. Select the desired component type from the **Available Components** list and click **Add Component**.

Each UI component has a **name** defined in the **Components Used** list. When you add a component, you are prompted for a name. This name is an important identifier used for manipulating the UI programmatically.

10.1.3 Editing Component Properties



The **Properties Editor** lets you modify the properties of the selected component. It works similarly to the one found in Paquet Builder. If you are unfamiliar with this grid editor, a brief description of how it works is provided below.

> **Warning**
>
> All changes are saved automatically.

10.1.4 Modifying UI Components and Controls at Runtime

ExeOutput for PHP supports changing control properties at runtime.

👉 Refer to the dedicated topic "How To Modify Controls At Runtime".

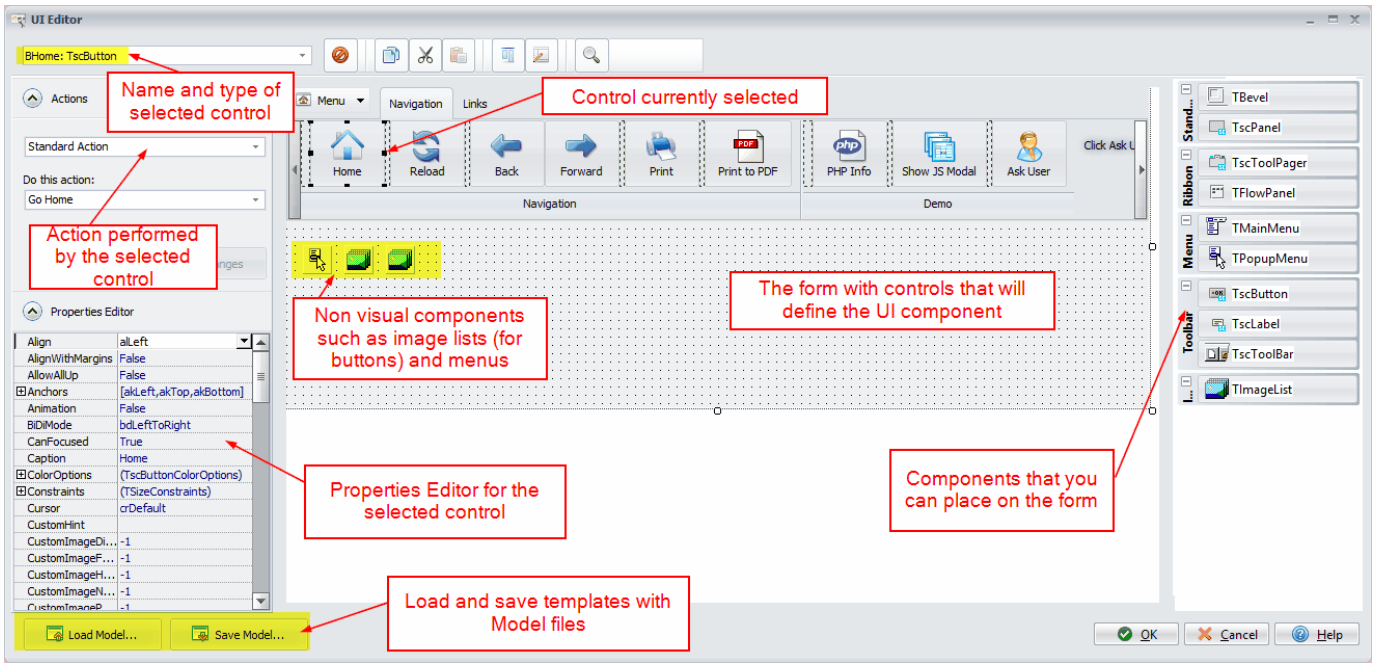
10.1.5 How the Properties Editor Works

The Properties Editor enables you to set properties for the selected component, which defines its state.

- The first column lists property names. A plus sign (+) indicates sub-properties that can be expanded. A minus sign (-) collapses them. You can also use the `+` and `-` keys.
- The second column displays property values. When a property is selected, you can type a new value.
- A small button (...) appears for properties that can be set using a dialog.
- A drop-down arrow appears for properties with a predefined list of values (enumerated types).

10.2 User Interface Editor

When you select a component in the **Components Used** list, you can sometimes click "Edit..." to modify the Windows visual controls that define that UI component. The UI Editor will then be displayed:



10.2.1 The UI Editor

The UI editor is similar to the form designers in Embarcadero Delphi and C++ Builder. On the right side is a palette of **different controls** that you can place on the UI form. On the left, you can **edit the properties** of selected controls with the Properties Editor. In the top left, you can choose a control from the list and **associate actions with it**, if applicable.

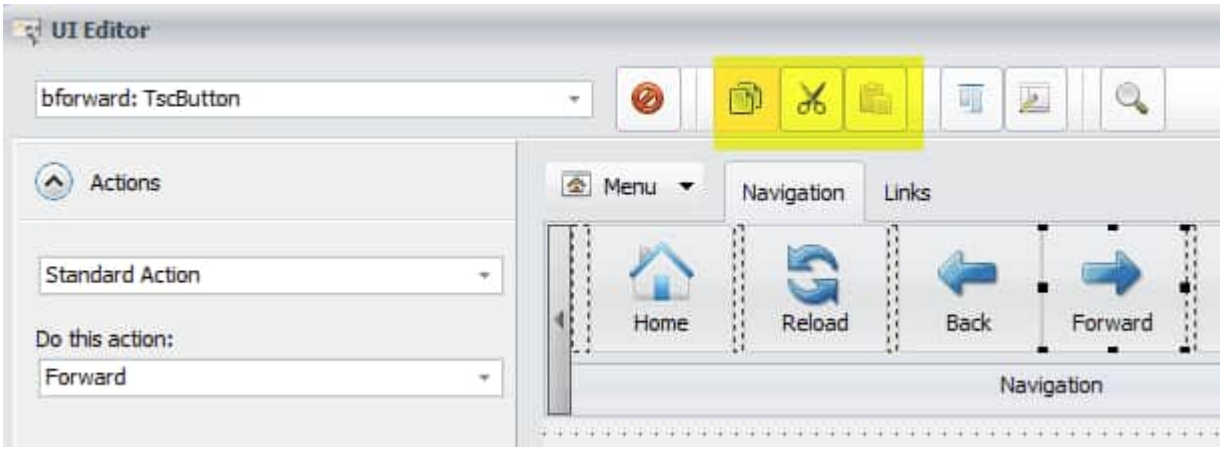
☑ **Each control has its own unique name**. You cannot change the name directly in the Properties Editor (see below). You are **prompted for a name when you add a new control** to the form.

Adding a New Control

Choose the type of control you want and place it on the form (or double-click it). You can resize the control using its grips.

Note that some controls are containers that can own other controls (called children). Available containers include `TscPanel`, `TFlowPanel`, `TscToolbar`, and `TscToolPager`.

☞ The easiest way to create new controls is to copy and paste existing ones. To quickly modify an existing UI model, use the **Copy**, **Cut**, and **Paste** buttons:



See the dedicated topics for these specific controls:

- Menu Bar
- Toolbar
- Ribbon
- Image Logo
- Timer (Cron Jobs)

Editing Properties

The Properties Editor displays all properties for the selected control. It works the same way as the main Properties editor.

You do not have to modify all properties. Only some are typically used, such as `Caption`, `Align`, `Name`, `Enabled`, `Visible`, etc.

Some controls, such as `TImageList`, have **specific property editors** that you can open by double-clicking them.

Aligning Controls for Resizing

👉 To create UI controls that resize properly, use the following properties. You can also watch a tutorial at https://www.youtube.com/watch?v=puO2XLR0azI:

- Align
- AlignWithMargins
- Anchors
- Margins
- Padding

Renaming a Control

You cannot rename a control after it has been created. The `Name` property is read-only.

Removing a Control

Select the control or controls you want to delete and click the **Delete** button (or press the `DEL` key).

10.2.2 Importing/Exporting a Model

ExeOutput for PHP ships with several ready-to-use and modifiable models. You can find them in the `UIModels` subfolder of your ExeOutput for PHP installation directory. Model files have the `.exouim` extension.

You can also export your own models.

10.2.3 Control Actions

When you place a menu item or button on the UI form, you must associate an action with it that will be performed when the user clicks it.

👉 See all available control actions

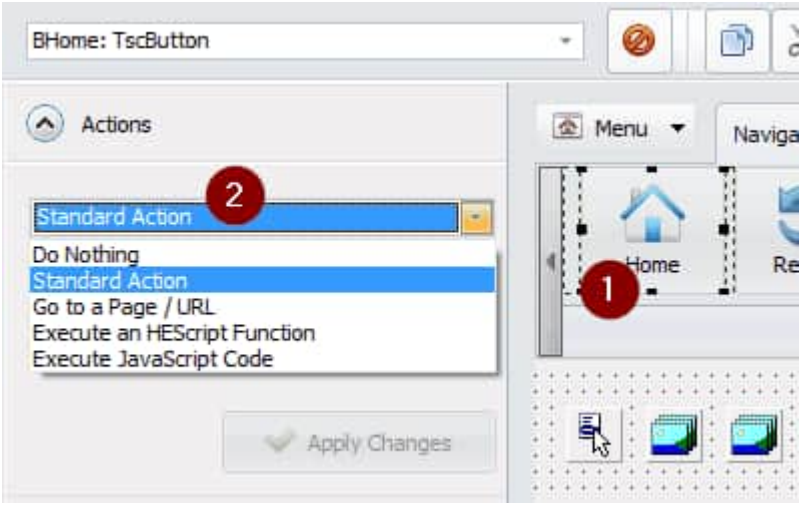
10.2.4 Modifying Controls at Runtime

It is possible to change a control's properties at runtime.

👉 Refer to the dedicated topic "How To Modify Controls At Runtime".

10.3 UI Control Actions

In the UI editor, when you place a menu item, button, or timer on the UI form, you must associate an action with it that will be performed when the user clicks it:



Images can also optionally be associated with a click action.

10.3.1 How to Define an Action

1. Select the component (button, image, or menu item) to activate the **Actions** panel.
2. Choose the desired action: `Standard Action`, `Go to a page/url`, `Execute an HEScript Function`, `Execute PHP Code (async)`, or `Execute JavaScript Code`.
3. Be sure to click **Apply Changes** to save your settings before selecting another control or exiting the UI editor.

10.3.2 Available Actions

Standard Action

Lets you choose a predefined action from the list:

- Go Home
- Back
- Forward
- Refresh
- Print
- Find
- Select All
- Copy
- Cut
- Paste
- Zoom In
- Zoom Out
- Reset Zoom
- About
- Exit Application
- Print to PDF

Standard actions are not available for images.

Go to a page/url

Opens the specified page or URL in the application's browser. You can optionally specify a target:

- `_heexternal` : Opens the URL in the default external web browser.
- `_henewinstance` : Starts a new instance of the application and opens the URL.

Finally, if you are using frames, the destination frame's name can be specified.

> **Important**
>
> This is not the correct way to open a popup. See below for instructions.

> **Warning**
>
> Do not specify a frame if there is no frame on your current webpage; otherwise, the action will fail.

Execute an HEScript Function

Associates an HEScript procedure or function with the button, menu item, or timer.

Syntax: `[scriptname].[functionname]`

Execute PHP Code (async)

This action saves the specified PHP code to a temporary virtual PHP file in the application's root folder and has the PHP engine execute it. When specifying the PHP code, you **must** include the `<?php` and `?>` tags. For instance:

```
<?php include('cron.php'); ?>
```

Execute JavaScript Code

This runs the specified JavaScript code. It must be the actual JS code, not just the name of a function. The code is executed within the context of the current HTML page.

OPENING A PAGE IN A POPUP

To open a new window or popup, you must use JavaScript. Choose the **Execute JavaScript Code** action and type the following, for instance:

```
window.open('https://www.exeoutput.com', 'preview', 'width=1000,height=700');
```

10.4 How to Modify Controls at Runtime

You can modify UI components and controls (especially their properties) at runtime with ExeOutput's HEScript engine.

10.4.1 Setting and Getting Property Values

HEScript provides two functions for this purpose:

- **SetUIProp**: Sets the value of a specified property for a control identified by its ID.
- **GetUIProp**: Gets the value of a specified property for a control identified by its ID.

```
procedure SetUIProp(const id, propname, propval: String);
```

- `id` : The name of the component followed by the name of the control.
- `propname` : The name of the property.
- `propval` : The new value to assign (always a string).

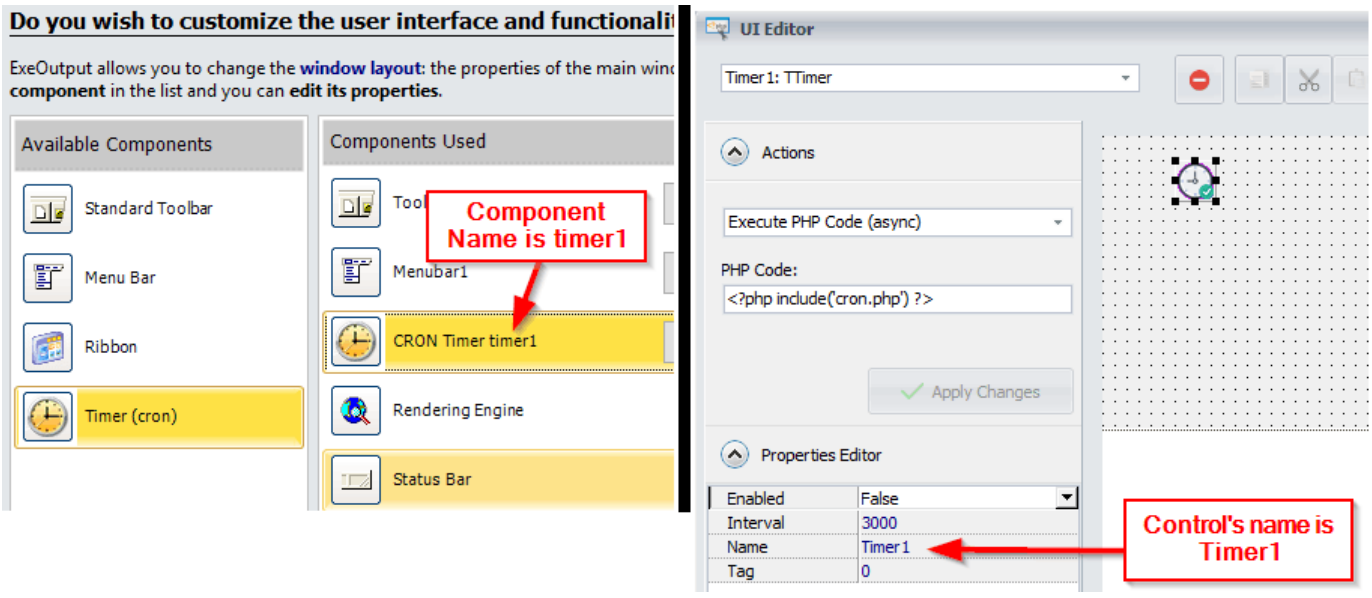
```
function GetUIProp(const id, propname: String): String;
```

- `id` : The name of the component followed by the name of the control.
- `propname` : The name of the property.
- `result` : The value of the property (as a string). An error may occur if the control is not found.

10.4.2 Determining a Control's ID

In ExeOutput, a control's ID is formed by concatenating the parent component's name and the control's name (with no space in between).

For example, for a `Timer1` control that belongs to the `timer1` component, its ID would be `timer1Timer1`.



10.4.3 Changing a Control's Caption

The following HEScript code changes the Home button's caption:

```
procedure Procedure1;
begin
  SetUIProp("ribbon1BHome", "Caption", "TEST!!");
  ShowMessage("This is Procedure 1 from the UserMain script. I changed the Home button caption!");
end;
```

In the code above, `ribbon1BHome` specifies that we are modifying a property of the `BHome` UI control, which belongs to the `ribbon1` UI component. `Caption` is the name of the property, and the last parameter is the new value.

> **Note**
>
> All control properties listed in the Properties Editor can be modified programmatically (e.g., `Caption`, `Visible`, `Enabled`).

10.4.4 Showing or Hiding a Control

The following HEScript code hides the control named `UserLabel`:

```
procedure Procedure2;
begin
  SetUIProp("ribbon1UserLabel", "Visible", "False");
end;
```

To make it visible again:

```
procedure Procedure2;
begin
  SetUIProp("ribbon1UserLabel", "Visible", "True");
end;
```

👉 To execute these HEScript procedures, please refer to the topic on how to call HEScript functions from PHP, HTML (links), and JavaScript.

10.5 Status Bar Properties

The **Status Bar** component displays a panel at the bottom of the main window. It shows information for users, similar to the status bars in web browsers. When a user moves the cursor over a hyperlink, the URL can appear in the status bar. To hide the status bar, set the `Visible` property to `False`.

When you select the **Status Bar** component in the "Components Used" list, the following properties become available:

- **AutoShowURLs**: Determines whether the application displays the URL of a link when the mouse pointer is over it. Possible values are:
- `HideAllURLs`: No URLs are shown. This is useful if you want to keep your URLs secret.
- `HideExternal`: Hides all external URLs (e.g., `http://`, `ftp://`) but not internal ones. Internal URLs begin with `https://heserver/` (for CEF) or `https://heserver.example/` (for WebView2).
- `HideInternal`: Does the opposite of the previous option.
- `ShowAll`: All URLs are shown.
- **DefaultText**: The default text to be displayed when no other information is indicated.
- **HideProgressBar**: If `True`, the progress bar on the status bar will be removed.
- **Visible**: If `False`, the status bar is removed.

10.6 Printer Properties

The **Printer** component configures properties for the printer and manages how HTML documents are printed. It also includes properties related to the Print to PDF feature.

When you select the **Printer** component in the "Components Used" list, the following properties become available:

- **EnablePrintPreview**: If `True`, the Print Preview window will be shown before printing the current webpage.
- **HeaderText, FooterText**: Sets the text to be displayed at the top (Header) and bottom (Footer) of each printed HTML page. The formatting options are the same as those in Internet Explorer.
- **OpenPDFAfterPrint**: If `True`, the application will automatically open the newly created PDF file after a user selects Print to PDF.
- **PrintLandscape**: If `True`, the page will be printed in landscape orientation.
- **SelectionOnly**: If `True`, only the selected portion of the page will be printed.
- **ShowHeaderFooter**: If `True`, the header and footer will be included in the printout.

See Also: Print and Kiosk Printing

10.7 Context Menu Properties

The "Context Menu" component controls the menu that appears when a user right-clicks. It gives users access to the most-used commands.

☑ You can add your own items to the context menu with the UI editor (click **Edit Items**).

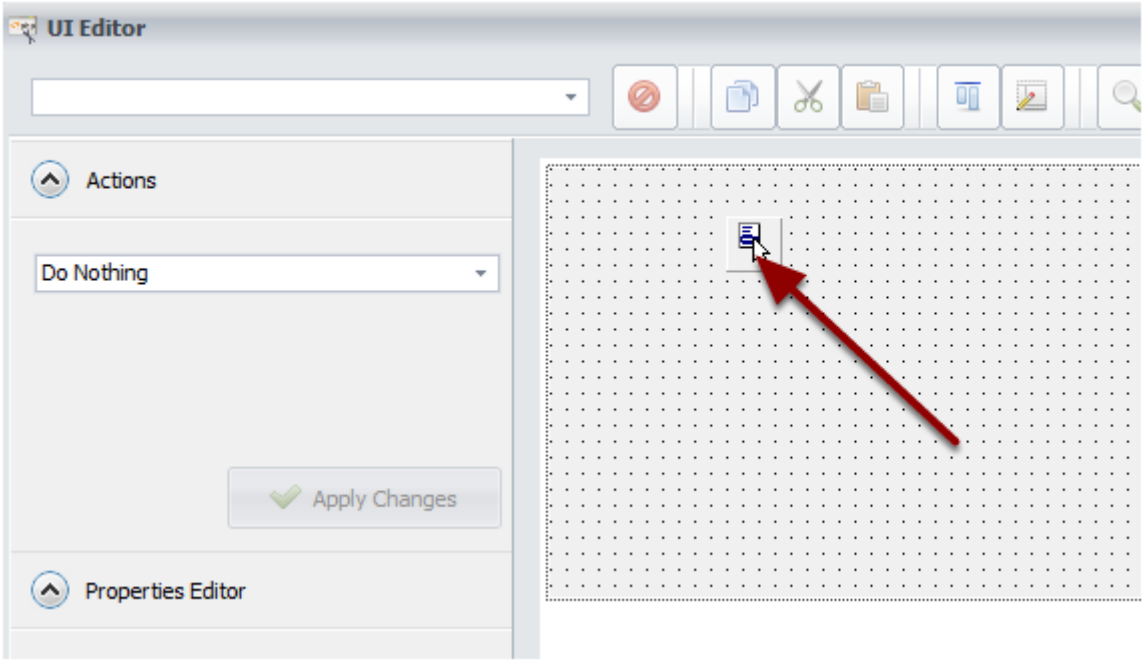
When you select the "Context Menu" component in the "Components Used" list, the following properties become available:

- **DisableContextMenu:** Set to `True` to remove the context menu.
- **DisableOpenLinkNewWindow**: Disables the "Open Link In New Window" menu item that appears when users right-click links on your webpages. ExeOutput for PHP handles secondary windows and pop-ups.
- **EnableShowSourceCode:** This can be useful for debugging. If this option is `True`, the "Show Source Code" menu item will be displayed, allowing you to inspect the full source code of the current HTML page.

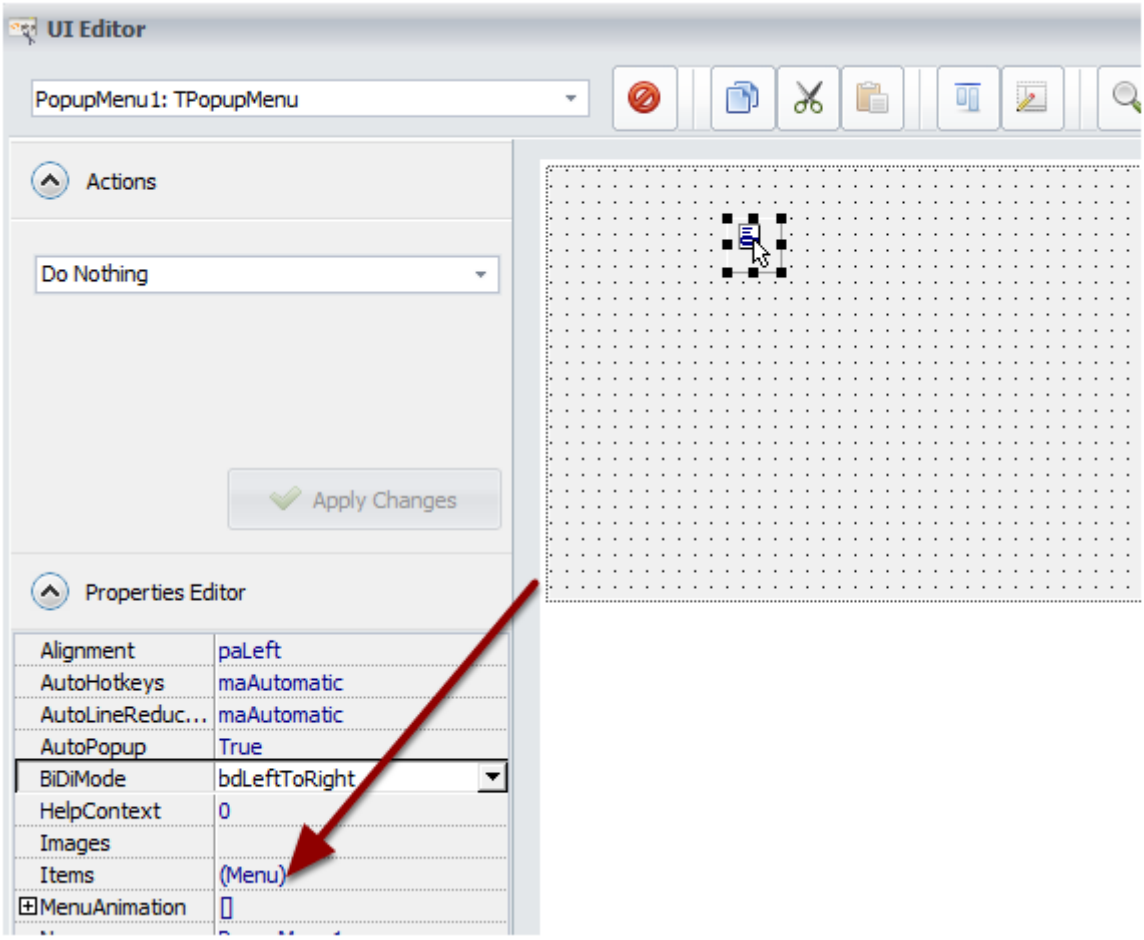
10.7.1 Edit Context Menu Items

Click **Edit Items**, and the UI editor will be shown.

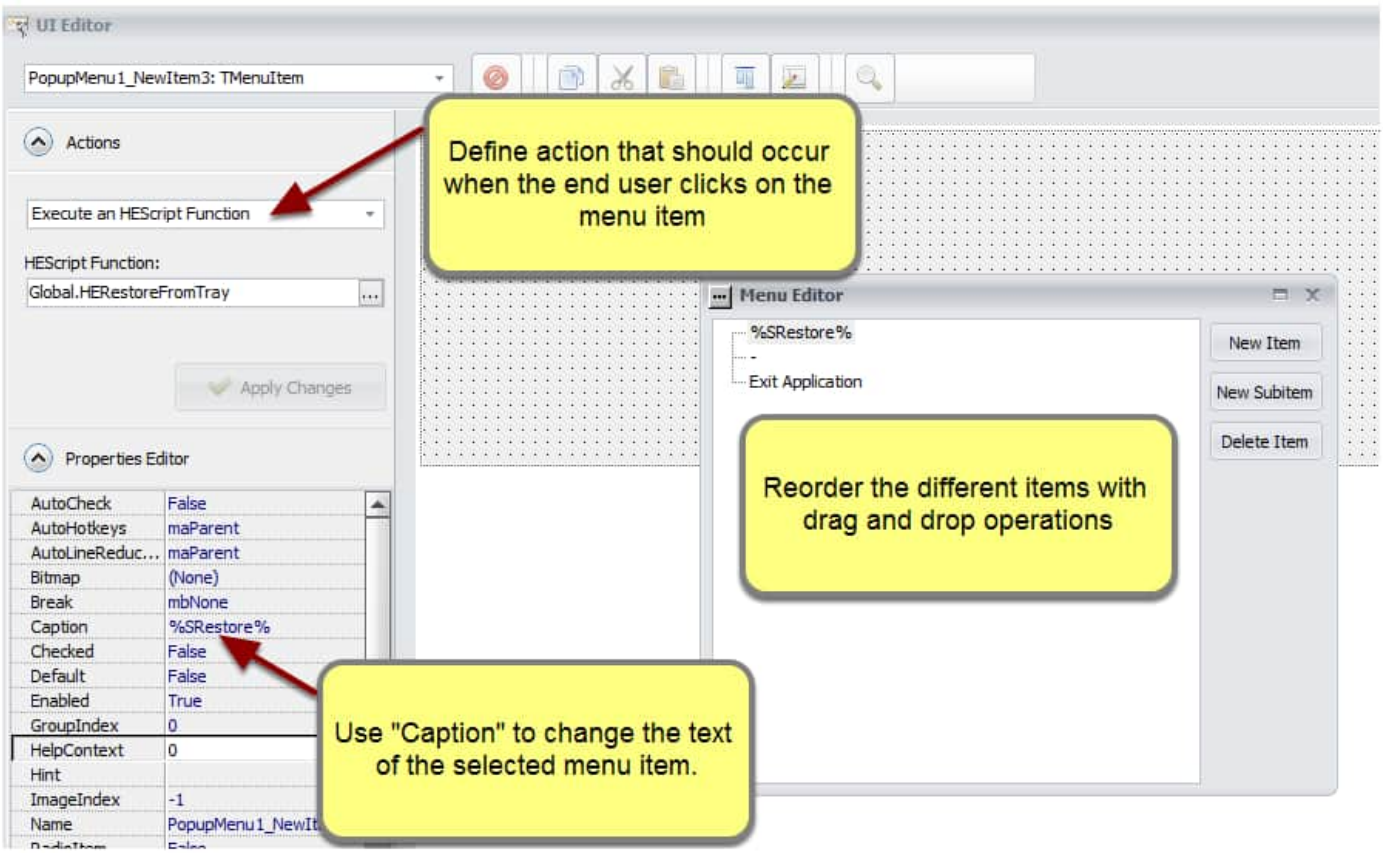
1. Select the TPopupMenu control.



2. Double-click on `(Menu)` in the Properties Editor.



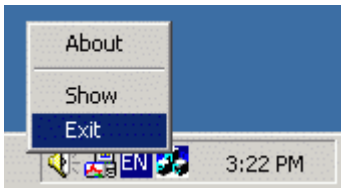
3. Click "New Item" to add a new item, or "New Sub-item" to create a child item.



👉 For the `Caption` property, you can use Resource Strings for easier localization. To insert a resource string, use `%RESID%` and replace `RESID` with the name of the resource string.

10.8 Tray Icon Properties

Instead of a taskbar button, you can have your application **show a small icon in the system tray**:



To do this, turn on the "**Enable Tray Icon**" option. The icon displayed is the application's main EXE file icon.

The tray icon can optionally display a popup menu when a user right-clicks it.

> **Info**
>
> In Windows 7 and later, not all tray icons are always visible by default. This is normal, and users can adjust their system's notification area settings to change this behavior.

10.8.1 Properties

When you select the **Tray Icon** component in the "Components Used" list, the following properties become available:

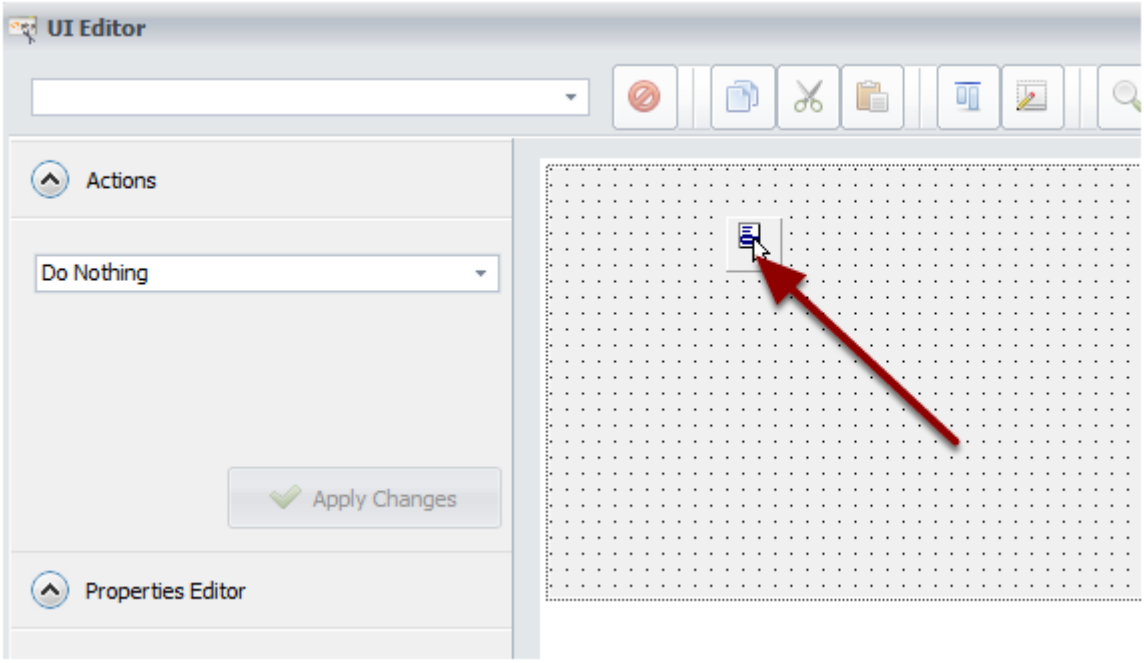
- **Default Hint Text**: The text to display in the tooltip when hovering over the tray icon.
- **Show Minimized on Start**: When this option is enabled, the application starts minimized; the main window is hidden, and only its tray icon is visible.
- **The Close button Minimizes the Main Window**: By default, clicking the Close button on the main window exits the application. To prevent the application from exiting, enable this option. It will force the main window to minimize to the tray instead of closing.

  In that case, ensure you **provide users with another way to exit your application** (such as an "Exit" command in the tray's popup menu). Otherwise, they will have to use the Windows Task Manager to close it.
- **Always minimize to tray**: If enabled, the main window will always minimize to the system tray instead of the taskbar. Otherwise, a taskbar button will appear when the main window is minimized.
- **Edit Popup Menu**: Allows you to manage the tray icon's popup menu items using the UI editor. This menu appears when a user right-clicks the tray icon. It should contain, at a minimum, "Restore" and "Exit" commands. These are created automatically for new projects, and we recommend keeping them.

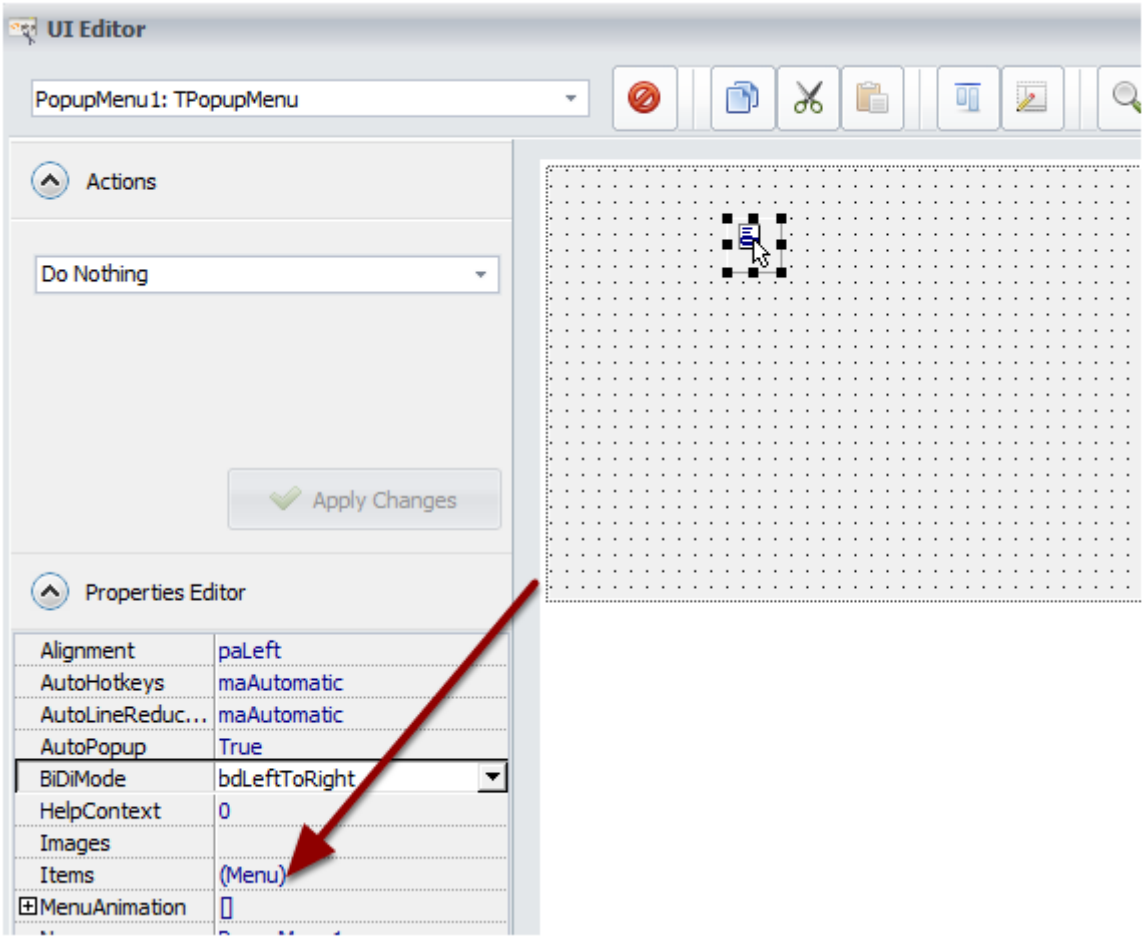
10.8.2 Edit Popup Menu

Click **Edit Popup Menu**, and the UI editor will be shown.

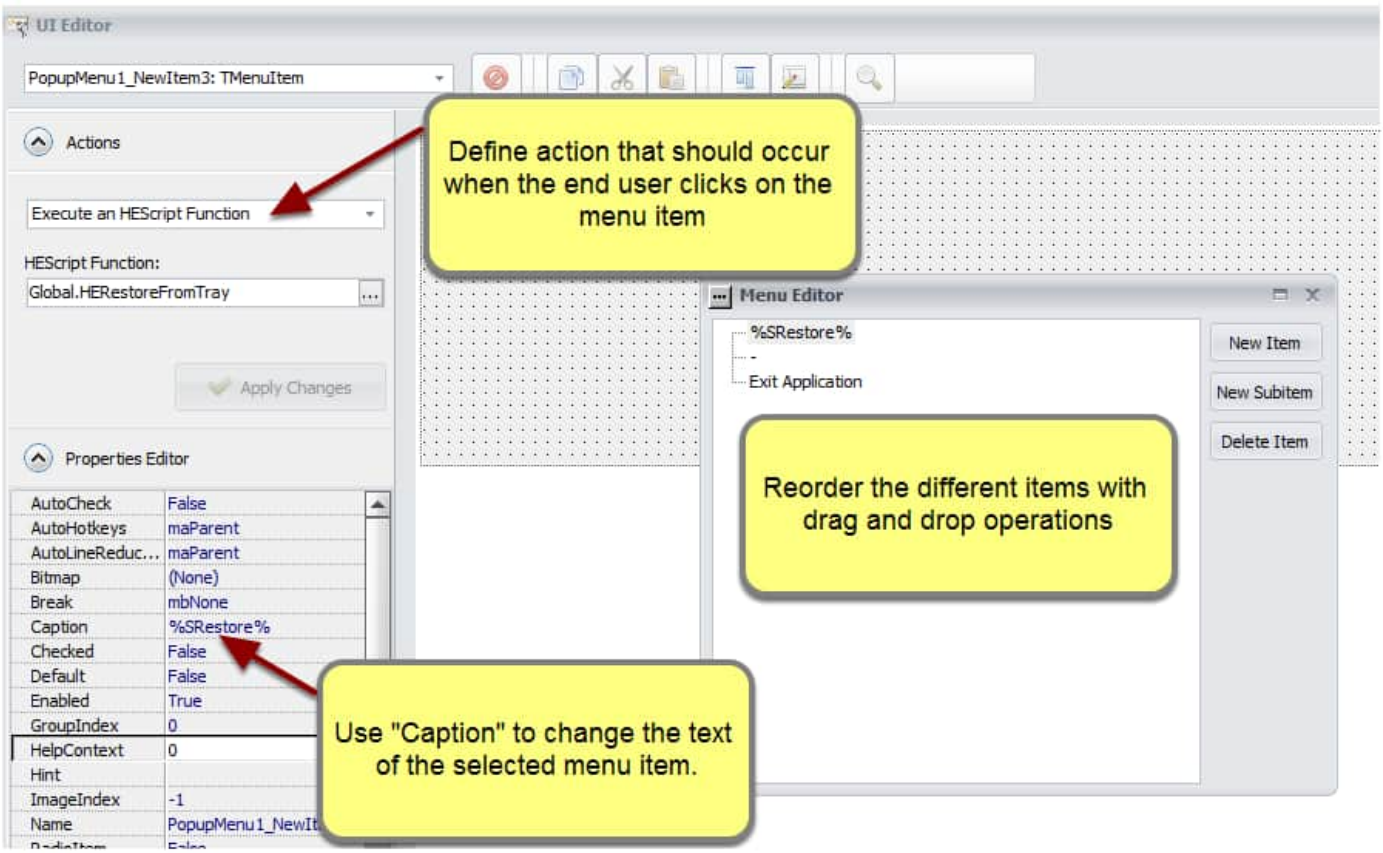
1. Select the TPopupMenu control.



2. Double-click on `(Menu)` in the Properties Editor.



3. Click "New Item" to add a new item or "New Sub-item" to create a child item.



👉 For the `Caption` property, you can use Resource Strings for easier localization. To insert a resource string, use `%RESID%` and replace `RESID` with the name of the resource string.

10.8.3 Changing the Icon Programmatically

You can change the tray icon at runtime using the `TrayChangeIcon` HEScript function. By default, the application's icon is used.

```
procedure TrayChangeIcon(const IconFile: String);
```

`IconFile` must be the full path to an external icon file or the virtual path to an icon file compiled into the application.

> **Warning**
>
> The file must be a valid icon file (`.ico`). To easily create `.ico` files, consider using a tool like GConvert.

Code Example: Modifying the Tray Icon at Startup

Place `sample1.ico` in your application's Source Folder so it is in the root directory.

Then, edit the UserMain script, locate the `OnStartMainWindow` event, and replace it with:

```
procedure OnStartMainWindow;
begin
  TrayChangeIcon("sample1.ico");
end;
```

Now, when the application starts, `sample1.ico` will be used as the tray icon.

10.9 Creating a Ribbon for Your Application

ExeOutput for PHP allows you to create ribbons for your PHP applications.

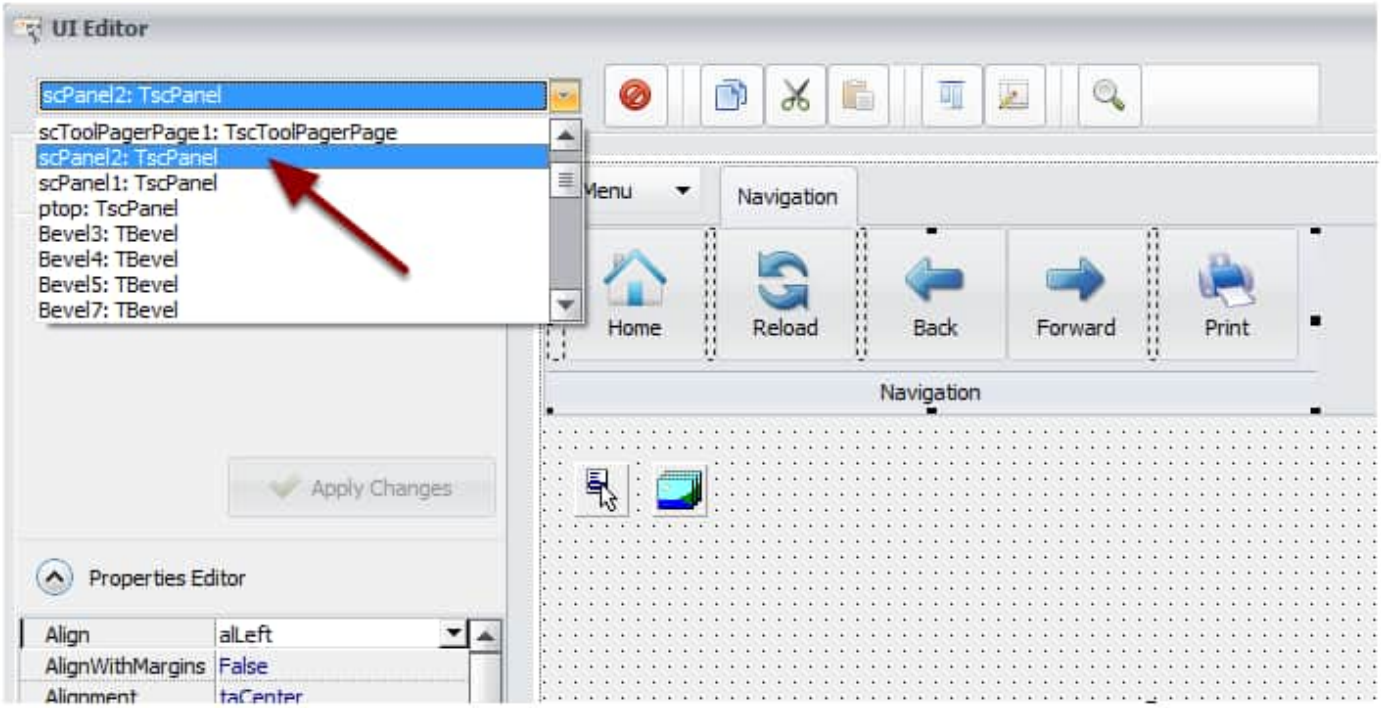
To **create a ribbon**, add a Ribbon to the "Components Used" list and then click **Edit Ribbon**.

The UI editor will appear, displaying a ready-to-use ribbon that you can modify.

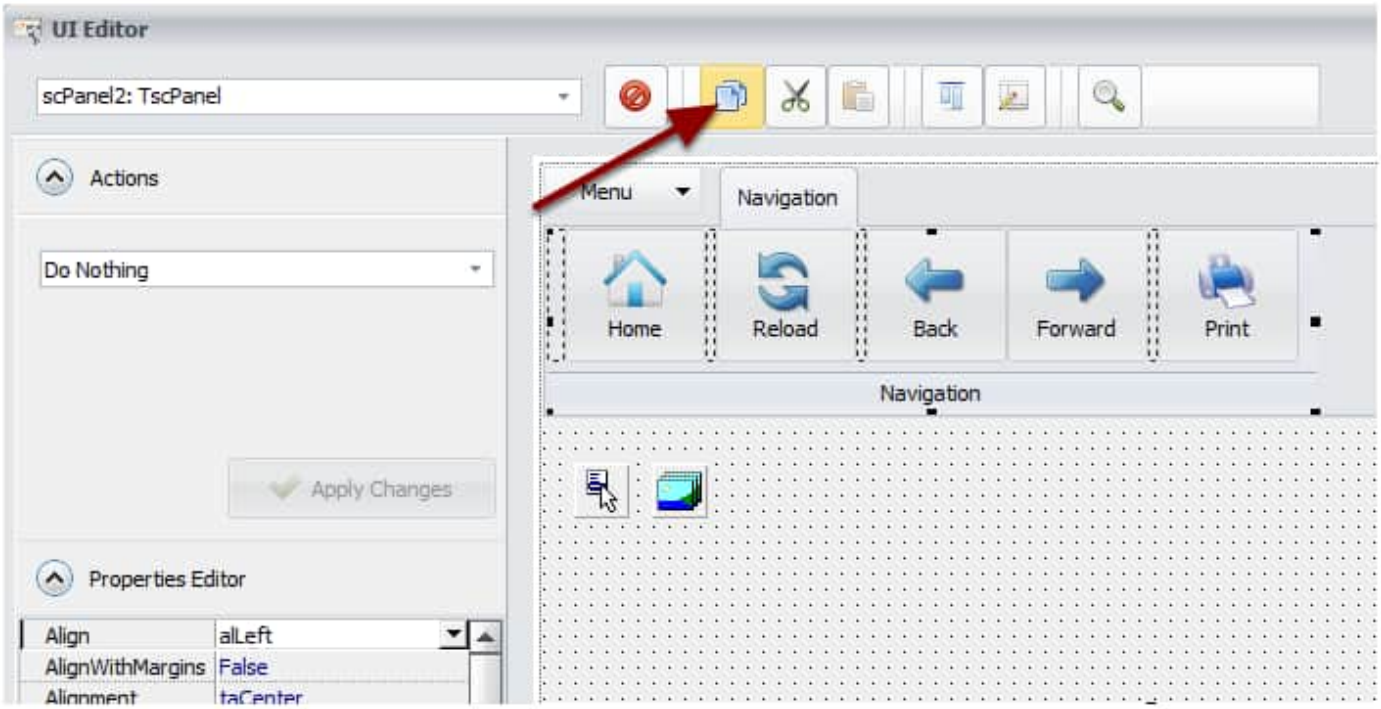
10.9.1 Adding a New Toolbar to the Ribbon

You can create a new toolbar by cloning an existing one.

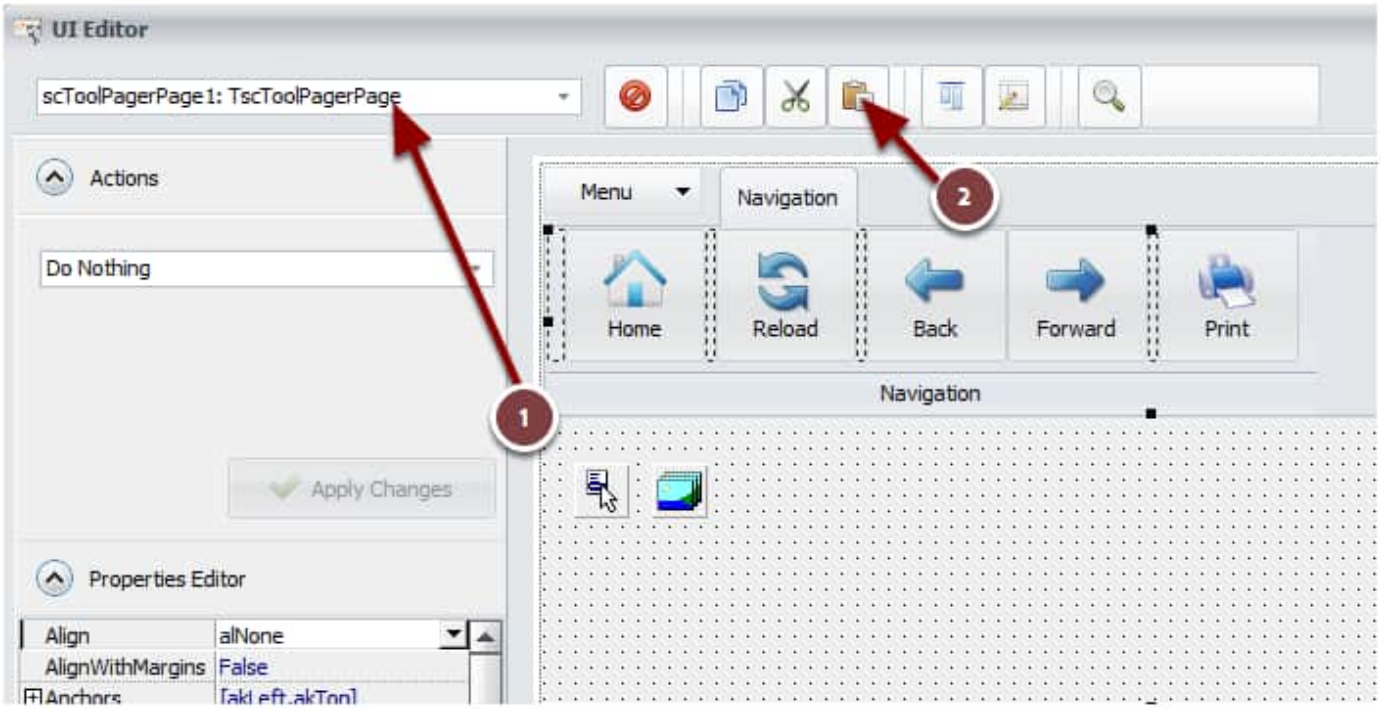
1. Select the original toolbar (named `scPanel2`):



2. Click **Copy**.

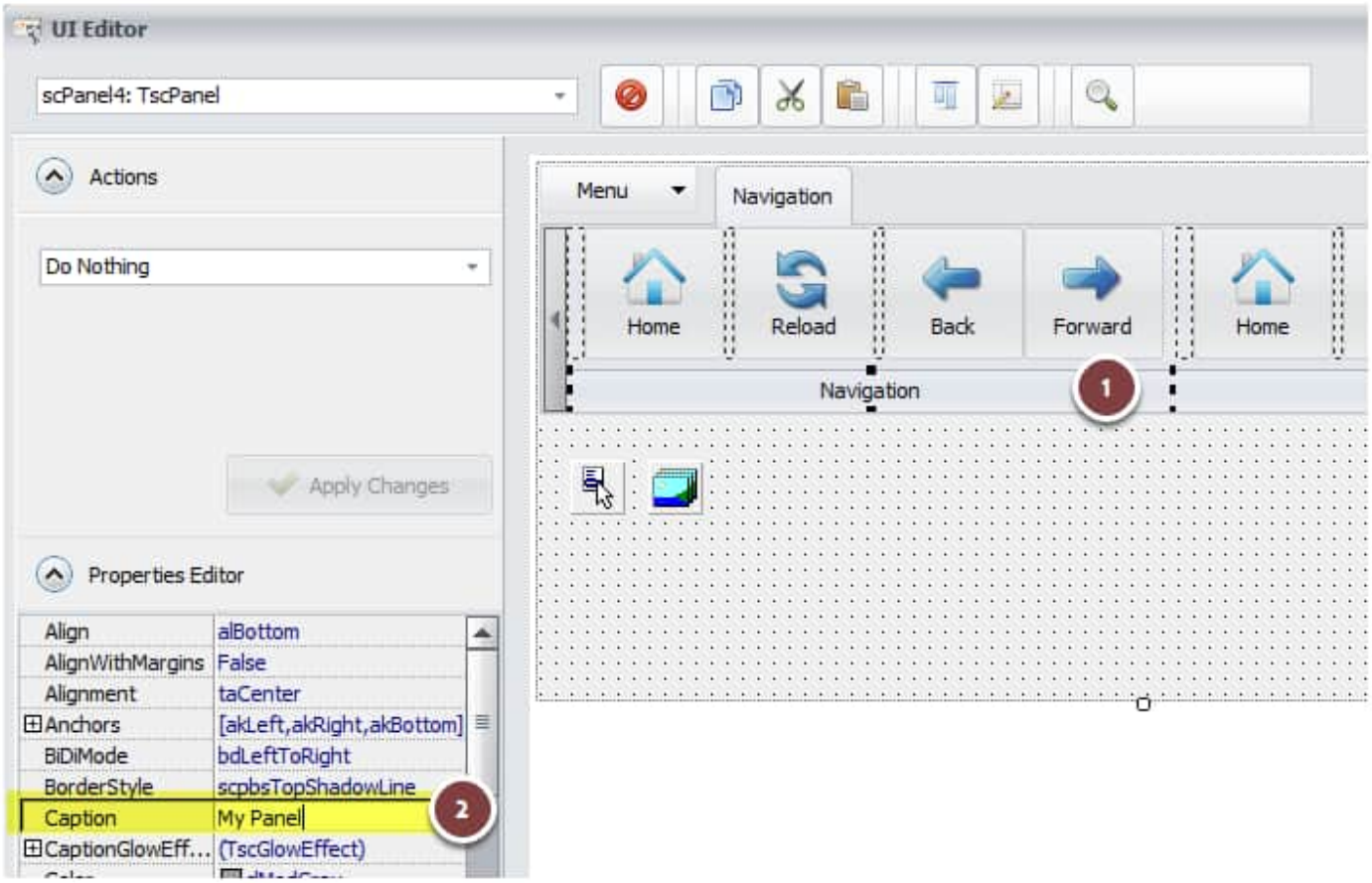


3. Select the entire page (named `scToolPagerPage1` here) and then click **Paste** to create the new toolbar.



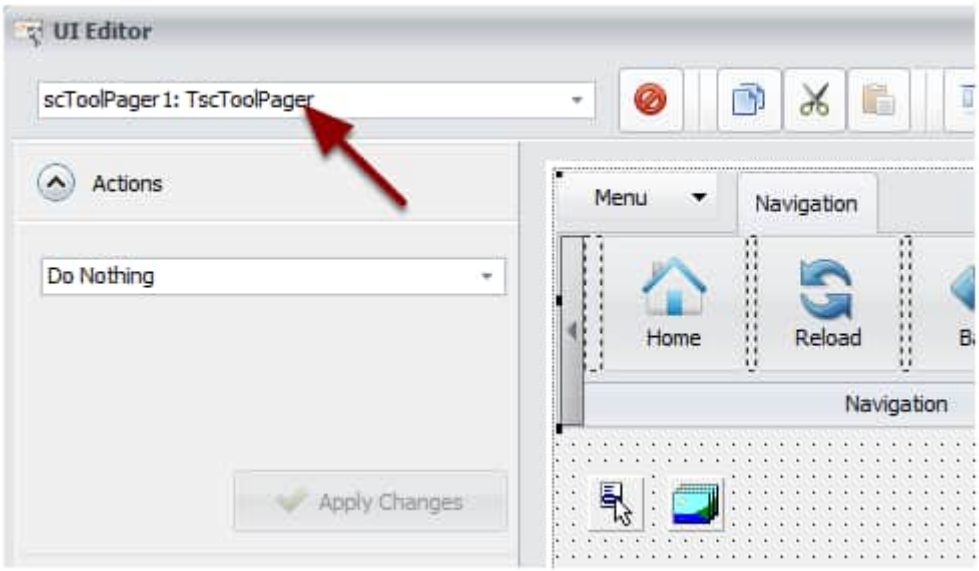
The new toolbar will be created. You can then remove any unwanted controls, change buttons, etc.

4. You can change the toolbar's description by selecting its description panel and modifying the `Caption` property.

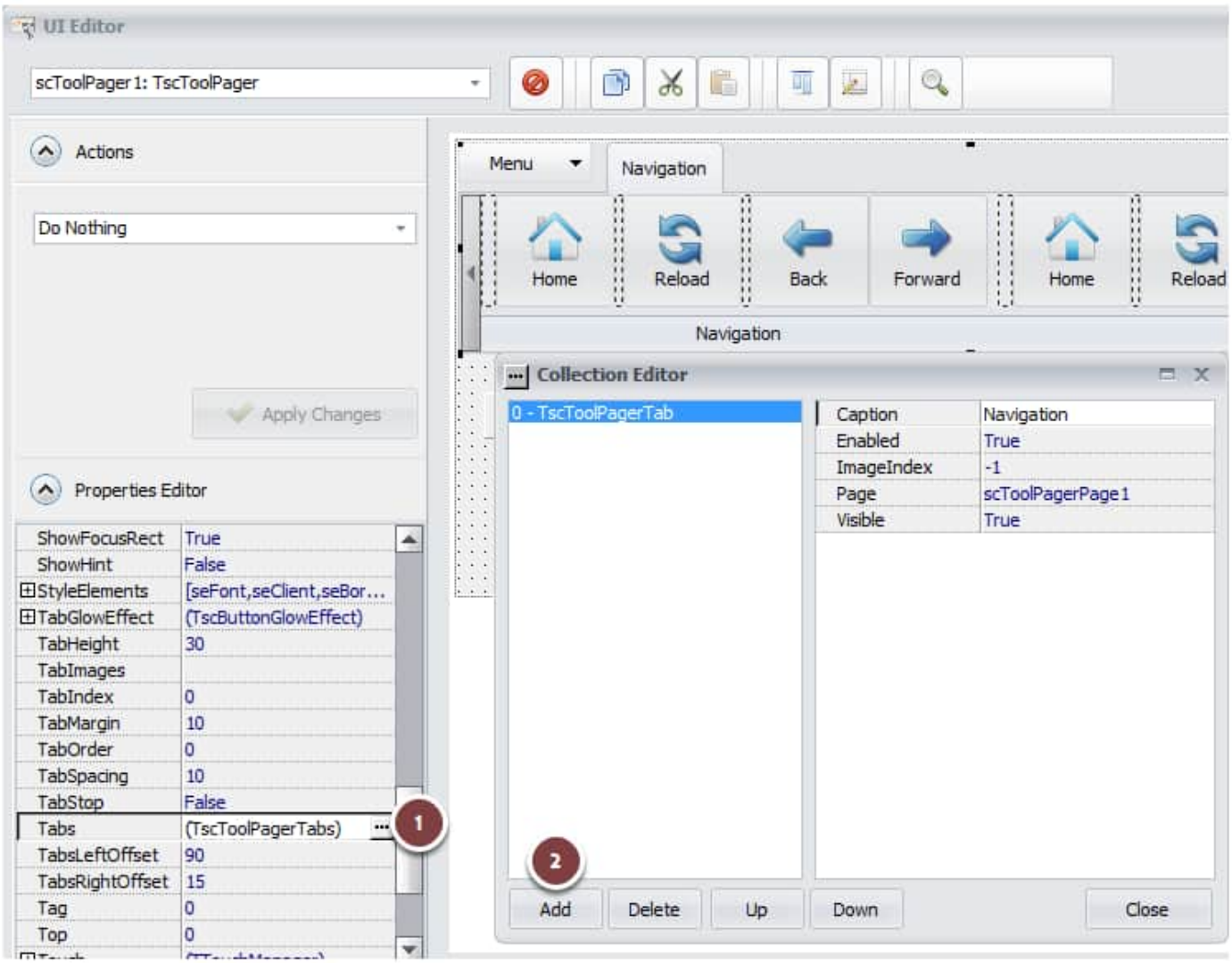


10.9.2 Adding a New Page to the Ribbon

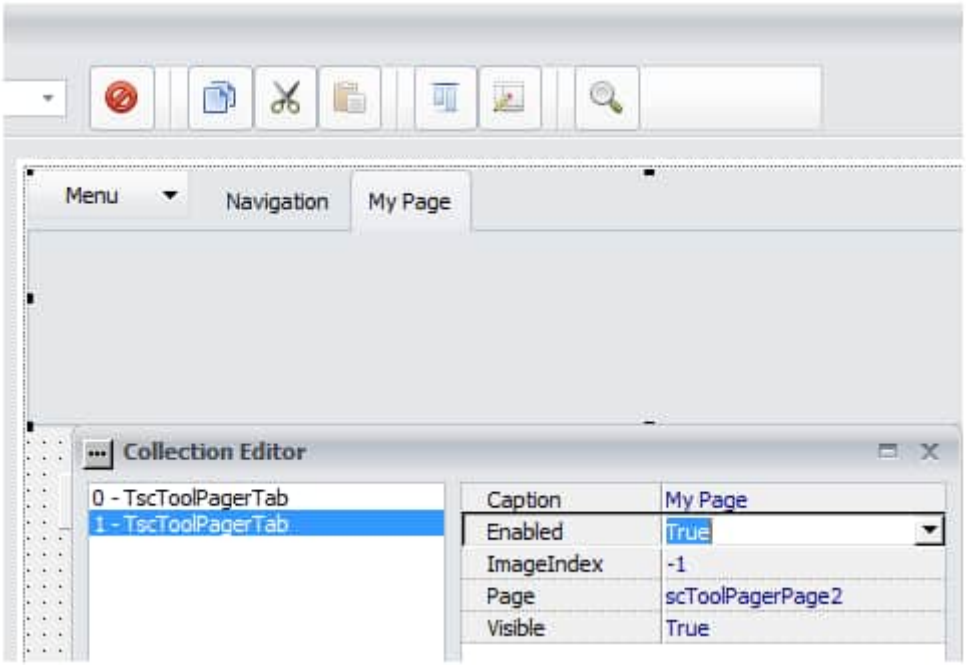
1. Select the entire ribbon (named `scToolPager1` here) in the editor.



2. Scroll down and double-click the `Tabs` property to open the Collection Editor. Click **Add**, and a new page will be added.



3. Modify its `Caption` and close the window.



4. Select the new page, then add a toolbar by copying and pasting an existing one as explained above.

👉 For the `Caption` property, you can use Resource Strings for easier localization. To insert a resource string, use `%RESID%` and replace `RESID` with the name of the resource string.

10.9.3 Modifying a Button or Ribbon at Runtime

It is possible to change any property of a button, or even the entire ribbon, at runtime.

👉 Refer to the dedicated topic "How To Modify Controls At Runtime".

10.9.4 Changing a Ribbon Page's Name at Runtime

Use the following HEScript code:

```
SetUIProp("ribbon1scToolPager1", "Tabs.items[0].Caption", "My new caption");
```

- `[0]` refers to the first page.
- `[1]` refers to the second page, and so on.

10.10 Toolbars in Your PHP Application

The **Toolbar** component displays an **enhanced bar with large buttons** that allows users to quickly access the most-used commands. By default, applications are configured with several predefined buttons: *Home, Reload, Back, Next, and Print*.

The roles of these buttons are as follows:

1. **Home**: Returns to the homepage.
2. **Reload**: Refreshes the current page.
3. **Back**: Returns to the previously loaded page.
4. **Forward**: Goes to the next page (if the user has already pressed Back).
5. **Print**: Displays the print dialog for the current page.

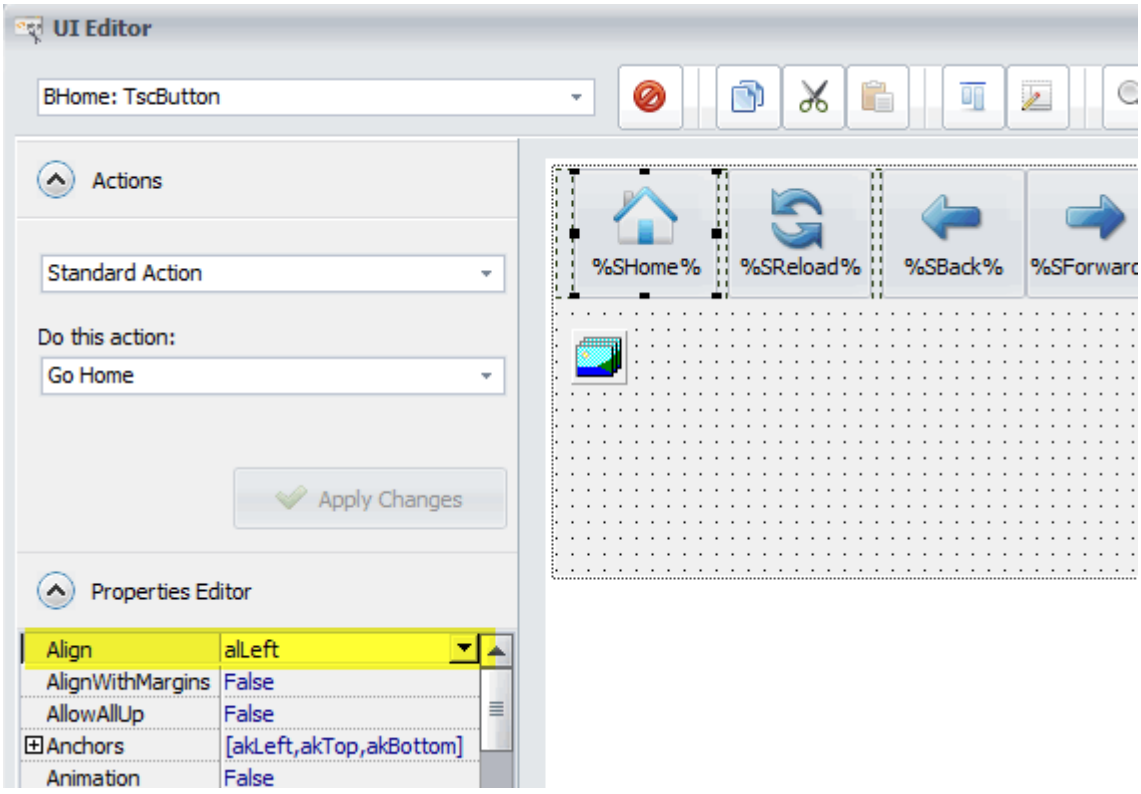
To **create a toolbar** in your PHP application, add a "Standard Toolbar" to the "Components Used" list and then click **Edit Toolbar**.

The UI editor will appear, displaying a ready-to-use toolbar that you can modify.

In the UI editor, you can **add or remove buttons**, **associate commands with them**, and **configure their properties.** You can also reorder the buttons.

10.10.1 Toolbar Control Description

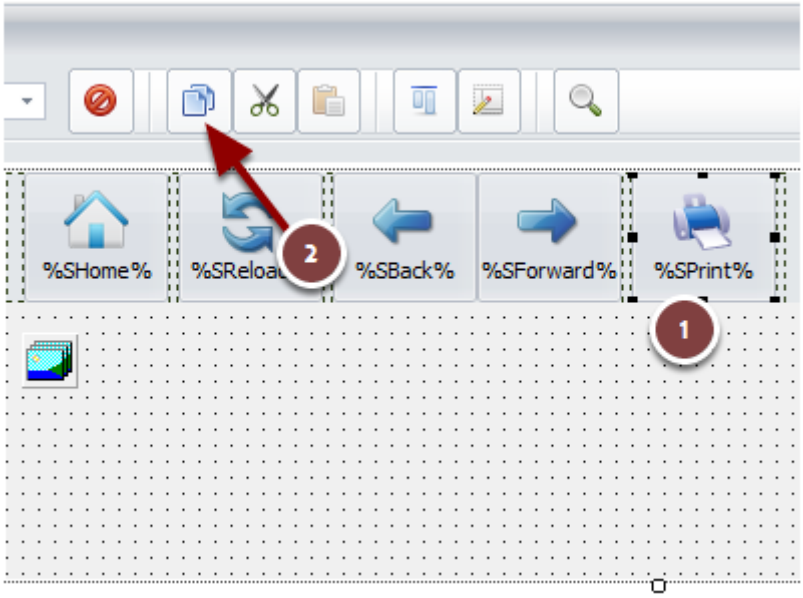
A panel container (`TscPanel`) is used to hold all of the buttons. To align all buttons to the left, each button's `Align` property must be set to `alLeft` .



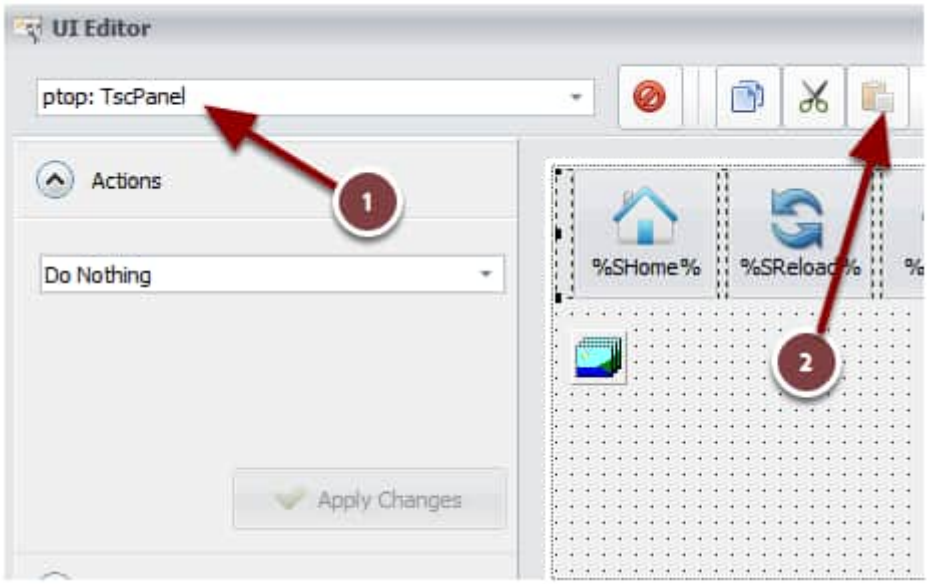
10.10.2 Adding a New Button and Optional Separator

To add a new button, you can clone an existing one.

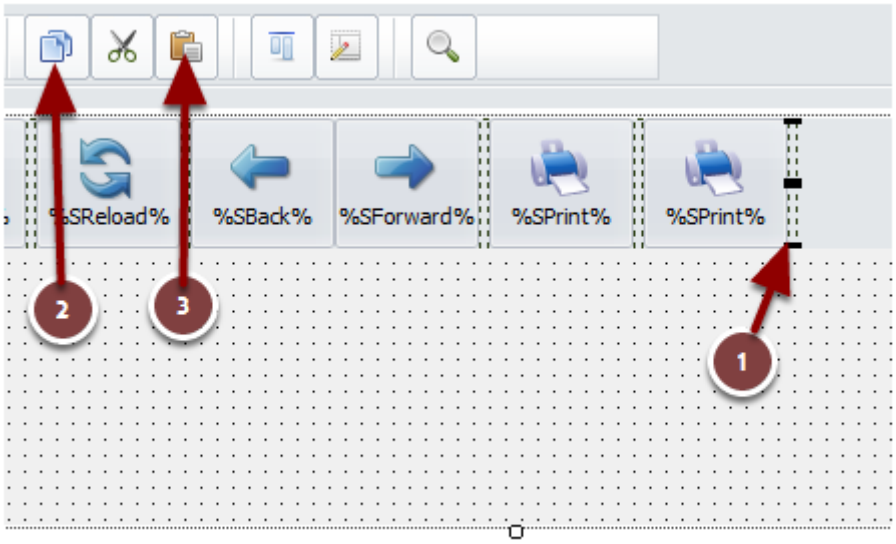
1. Select the button to clone and click **Copy**.



2. Select the entire toolbar panel (named `ptop` here) and click **Paste**.



3. Repeat the same steps with a `Bevel` control to create an invisible separator between buttons.



10.10.3 Configuring Button Properties

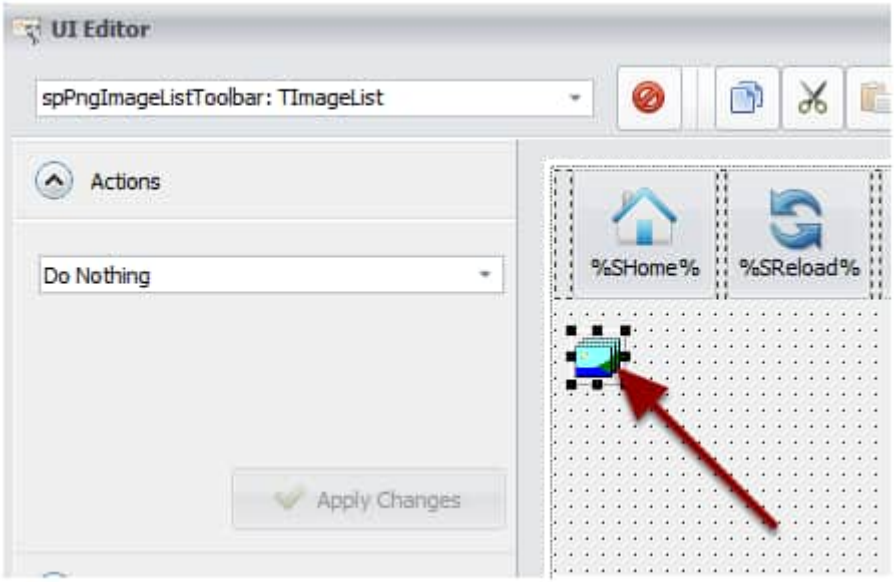
Useful properties for buttons include `Caption`, `ImageIndex`, `Margins`, `ShowCaption`, etc.

👉 For the `Caption` property, you can use Resource Strings for easier localization. To insert a resource string, use `%RESID%` and replace `RESID` with the name of the resource string.

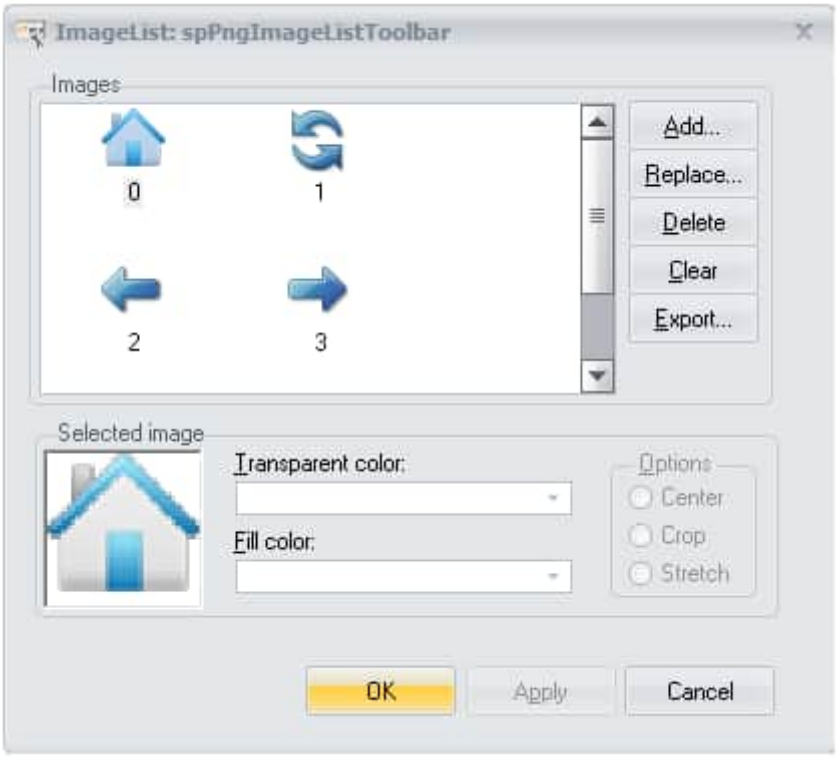
10.10.4 Changing a Button's Image

> **Tip**
>
> We recommend using 32-bit PNG image files with an alpha channel for transparency.

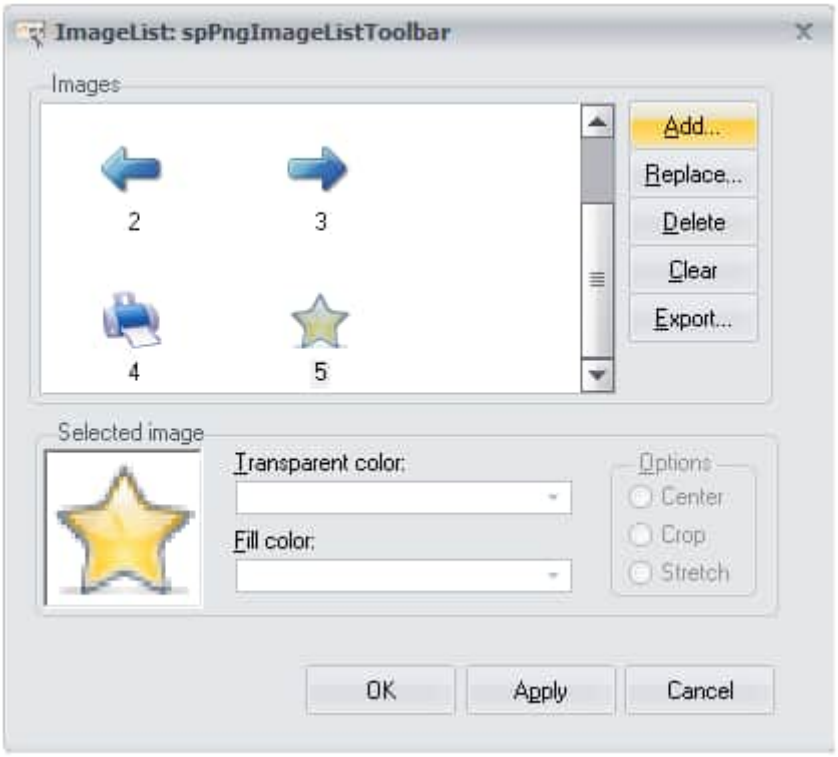
1. Double-click the default image list (named `spPngImageListToolbar` here).



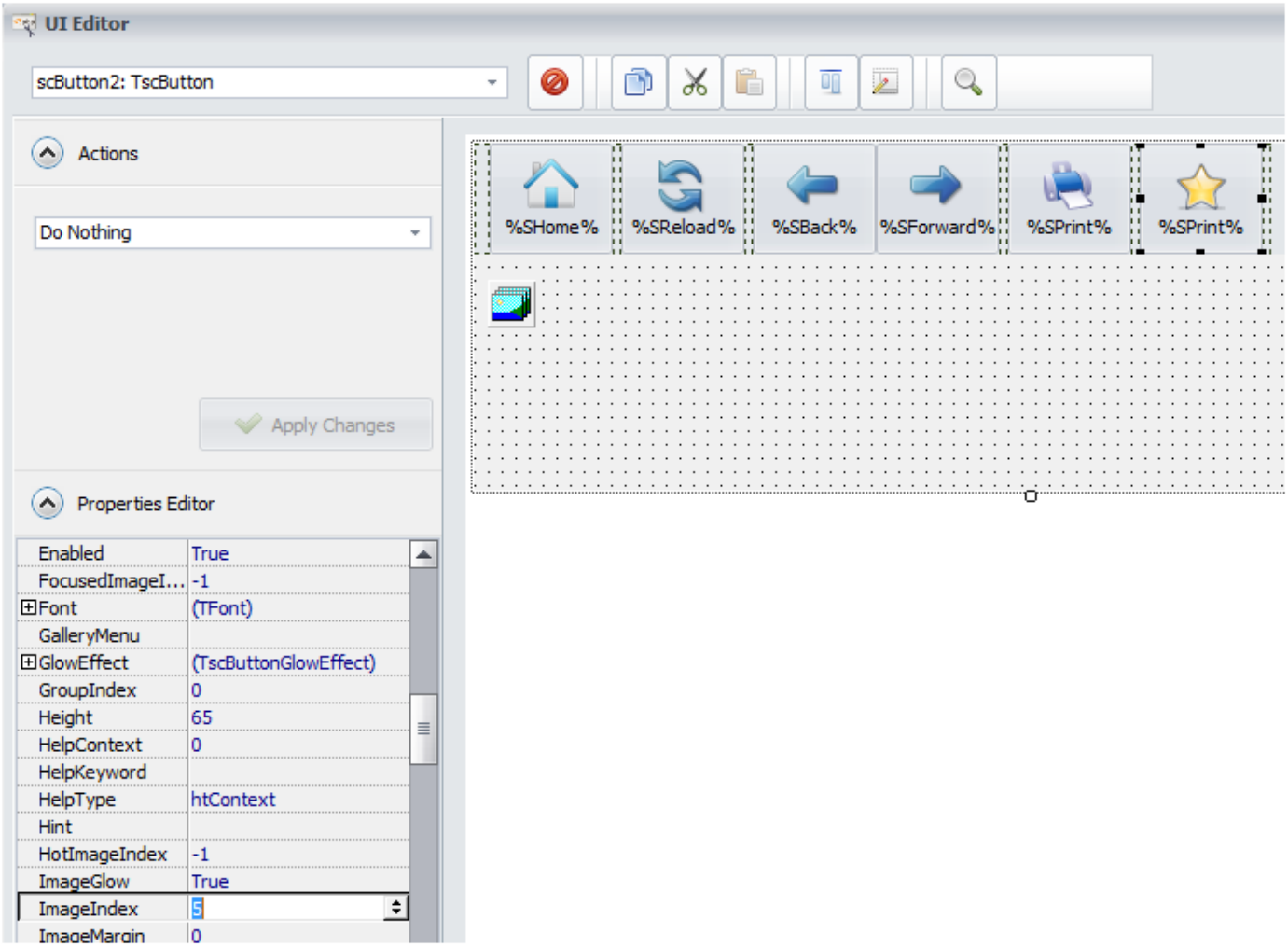
2. When the Image Editor appears, click **Add** to import a new PNG file.



3. Take note of the number associated with the imported image and click **OK**.



4. Select the desired button, scroll down the Properties Editor, and set the `ImageIndex` to the number you noted (e.g., 5). The button's image will update instantly.



10.10.5 Modifying a Button at Runtime

It is possible to change any property of any button at runtime.

👉 Refer to the dedicated topic "How To Modify Controls At Runtime".

10.11 Menu Bar in your PHP application

The **Menu Bar** component displays standard GUI menus at the top of the main window. By default, an application contains four predefined menu items: *File, Edit, Navigate, and Help*. These items provide various commands that let users navigate your web pages, select and copy text, print pages, and more.

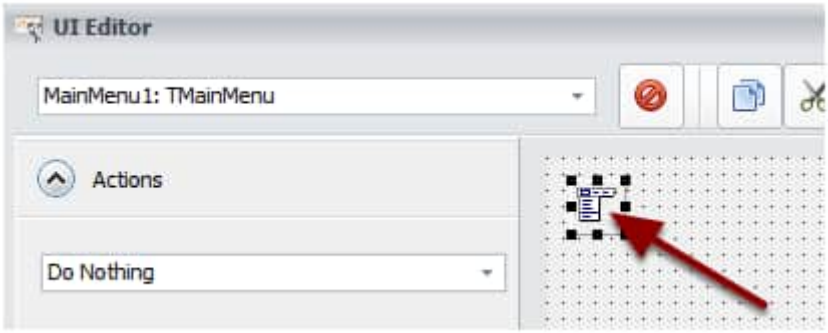
To **create a menu bar** in your PHP application, add a Menu Bar to the "Components Used" list and then click **Edit Menu**.

The UI editor will appear, which includes a ready-to-use menu that you can modify.

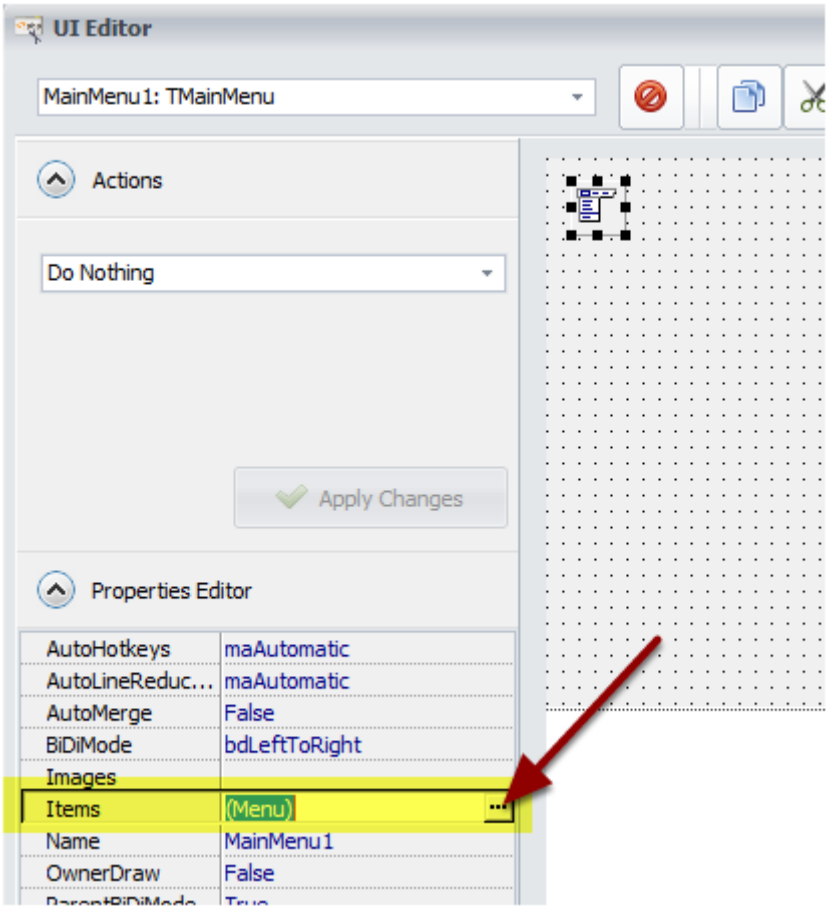
In the UI editor, you can **manage menu items**, **associate commands with them**, and **configure their properties.** You can also reorder these items.

10.11.1 Manage Menu Items

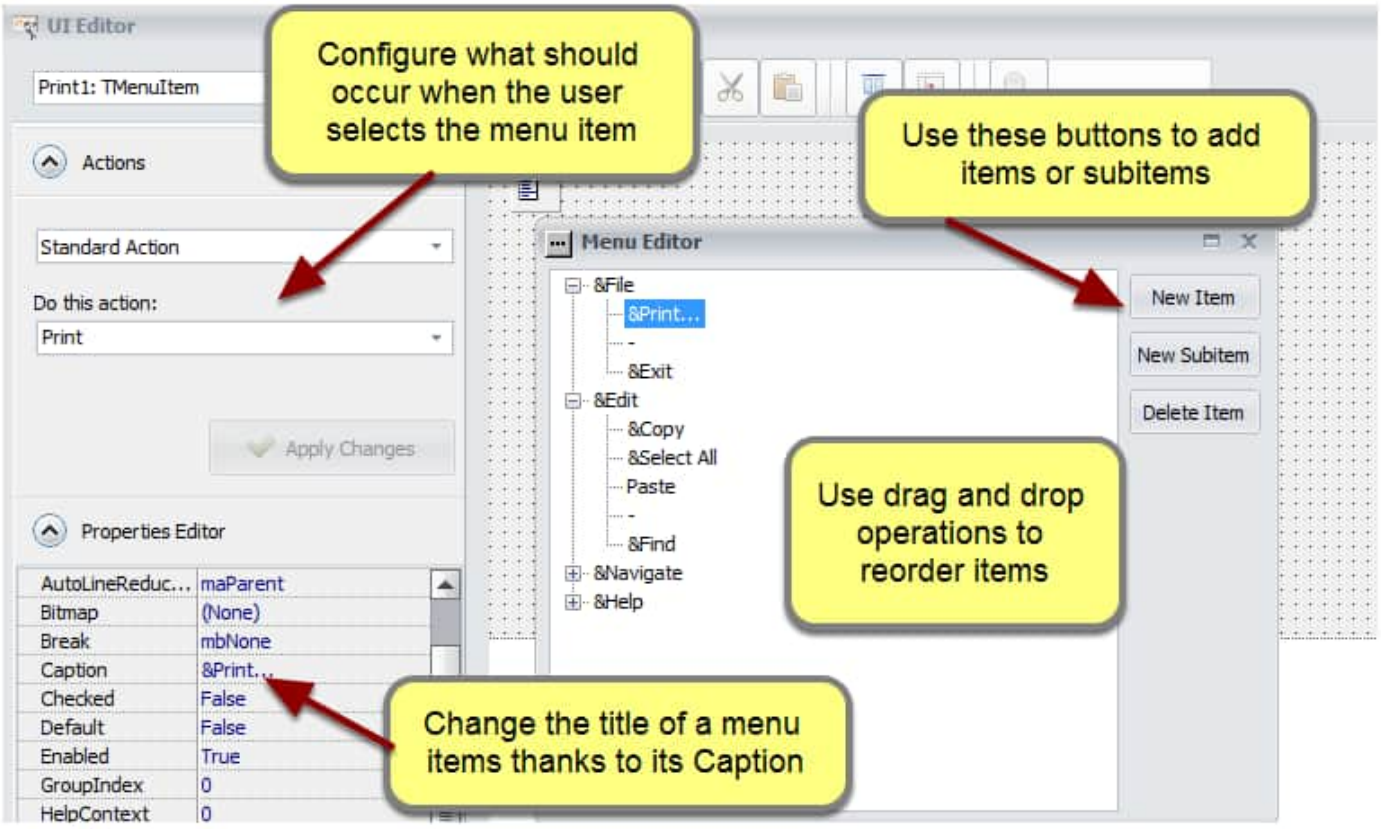
1. Select the `MainMenu1` control (TMainMenu type).



2. In the Properties Editor, double-click `(Menu)` corresponding to the `Items` property.



3. Use the Menu Editor tree to manage menu items:



👉 For the `Caption` property, you can use Resource Strings for easier localization. To insert a resource string, use `%RESID%` and replace `RESID` with the name of the resource string.

10.11.2 Menu Separator

To create a separator, set a menu item's `Caption` property to a single **hyphen** (-).

The action for a separator should be set to "Do nothing".

10.11.3 Menu Captions

If you associate a standard action with a menu item, its `Caption` property will be ignored. Instead, the compiled application will display the name of the action. You can modify these action names in the Localization editor.

10.11.4 Modify a Menu Item at Runtime

It is possible to change any property of any menu item at runtime.

👉 Refer to the dedicated topic "How To Modify Controls At Runtime".

10.12 Adding an Image or Logo to the UI

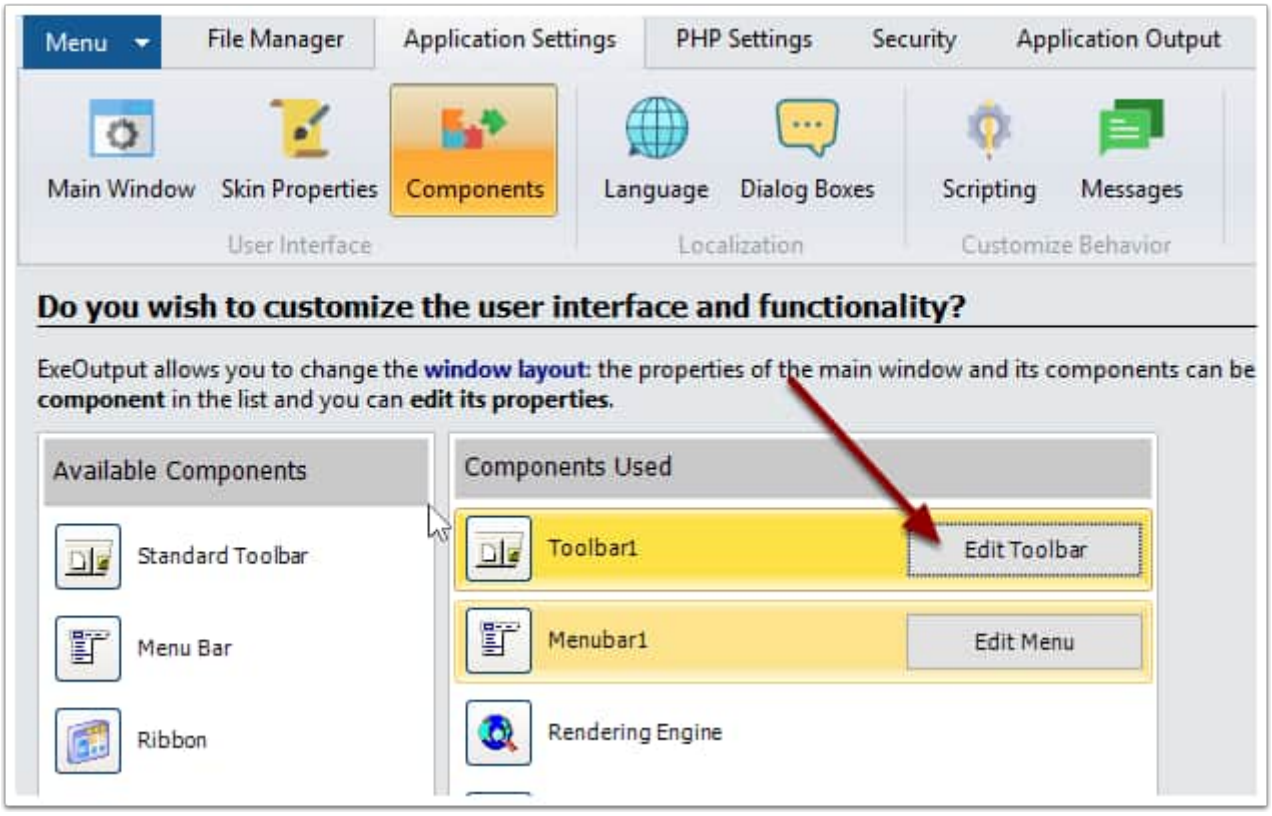
The **Image** component displays images in various formats. Its purpose is to let you insert custom logos on the toolbar or ribbon to display your brand in your applications.

> **Tip**
>
> Images are stored directly in the application, so there is no need to distribute the source image file with your EXE.

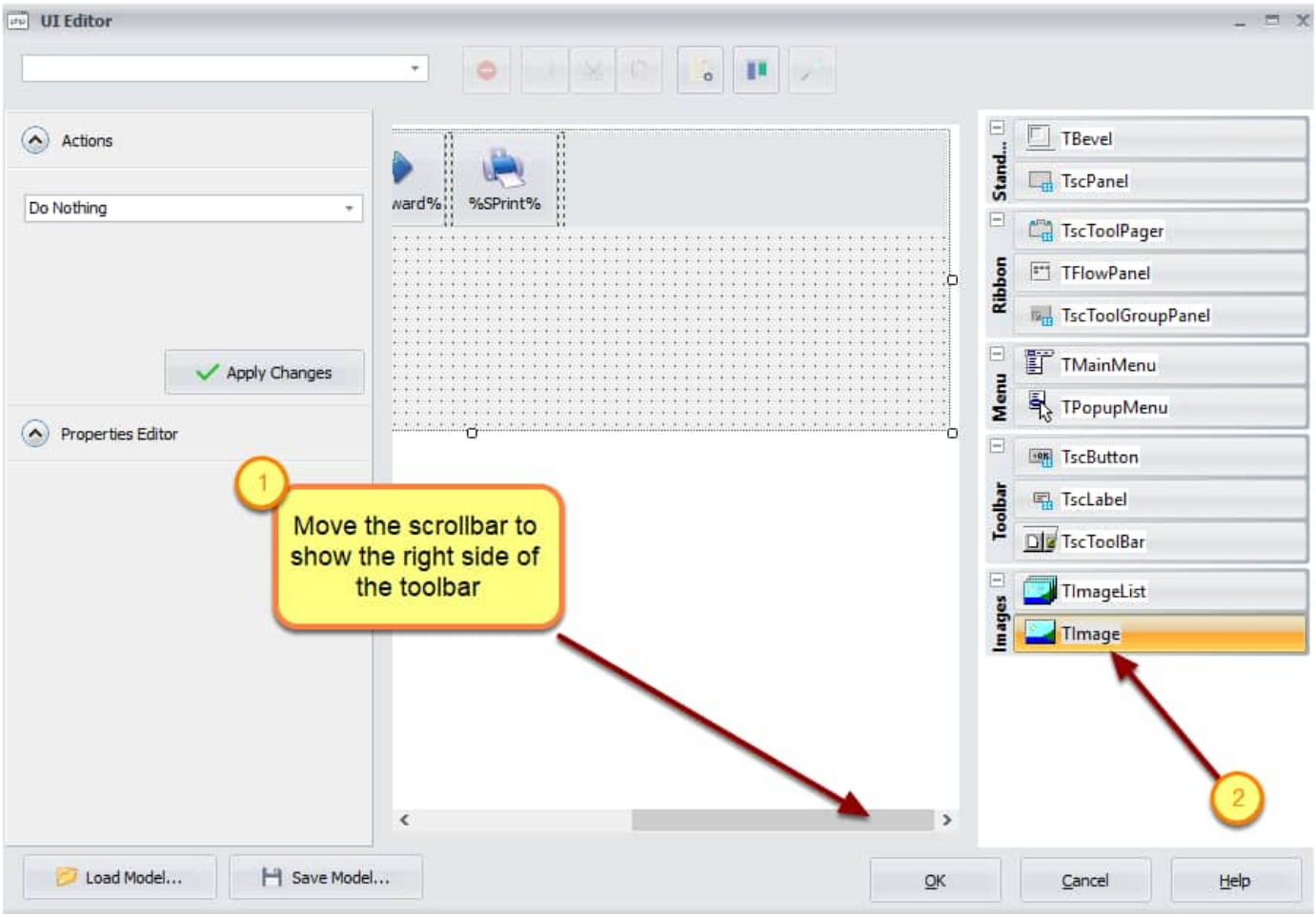
Use the UI editor to add new image components.

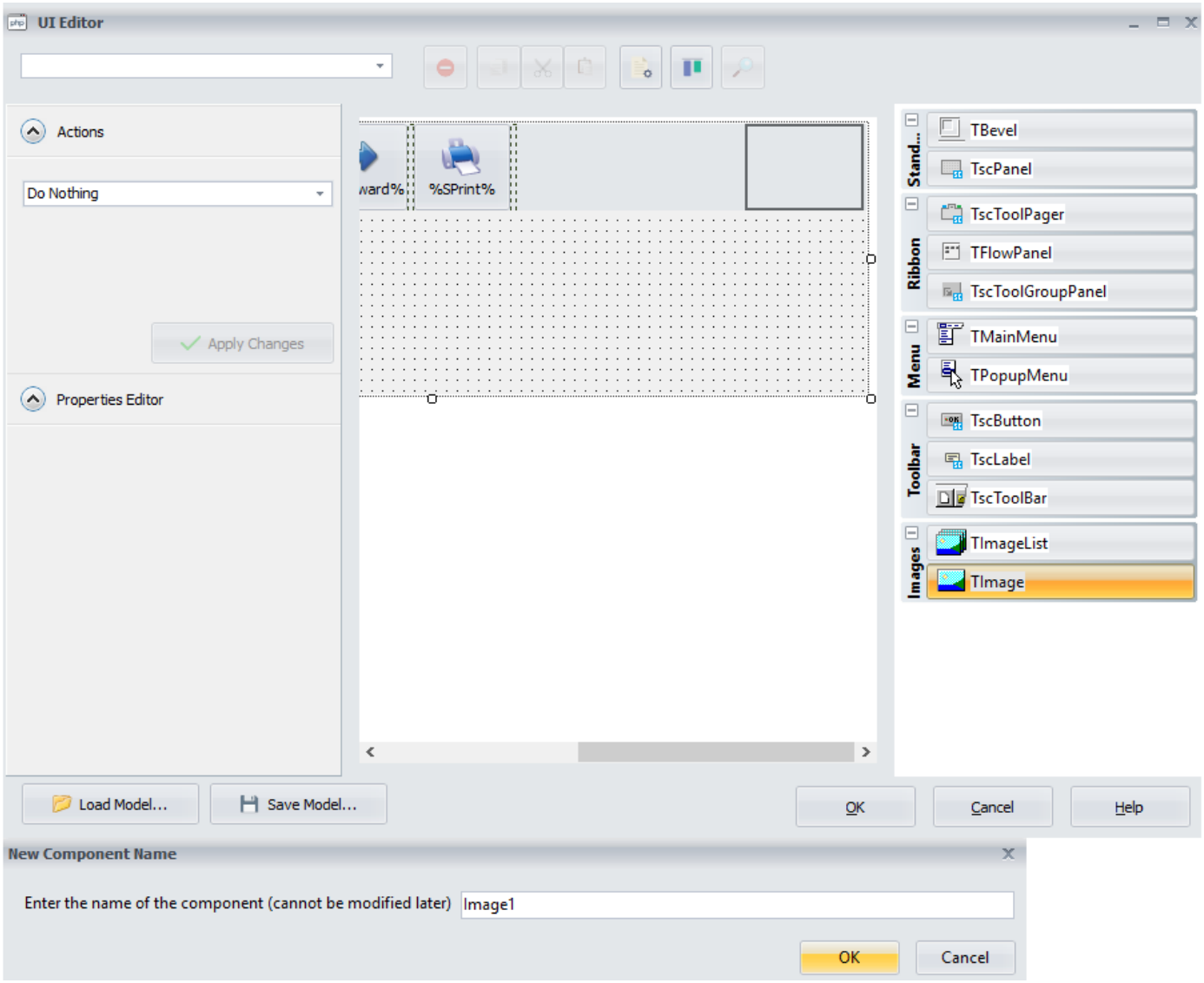
10.12.1 How to Add a Logo Image

1. In **Components**, choose the toolbar or ribbon where you want your logo to appear, and click **Edit Toolbar**.

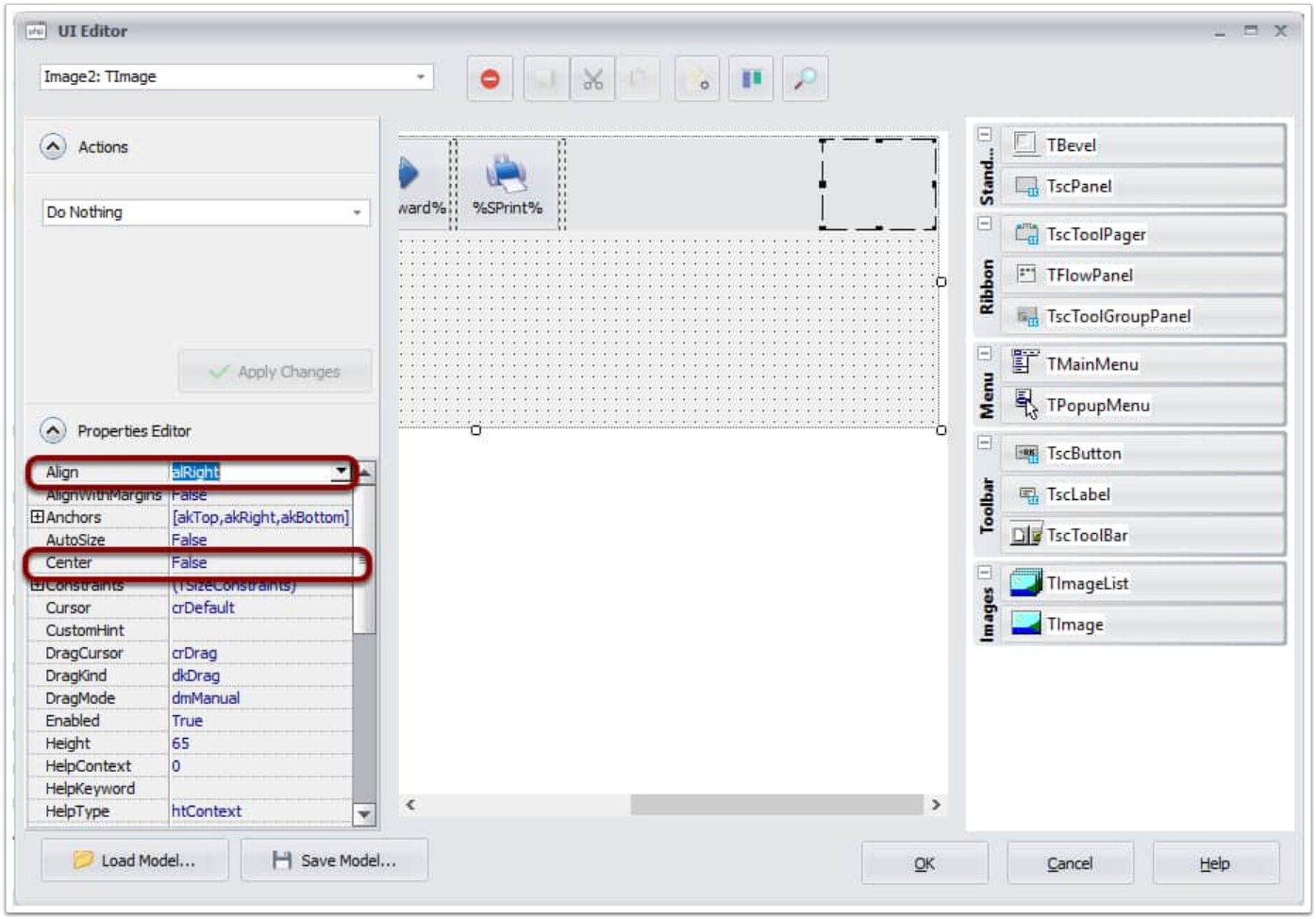


2. In the UI editor, select the `TImage` component and draw it on the toolbar. You can leave the default component name.

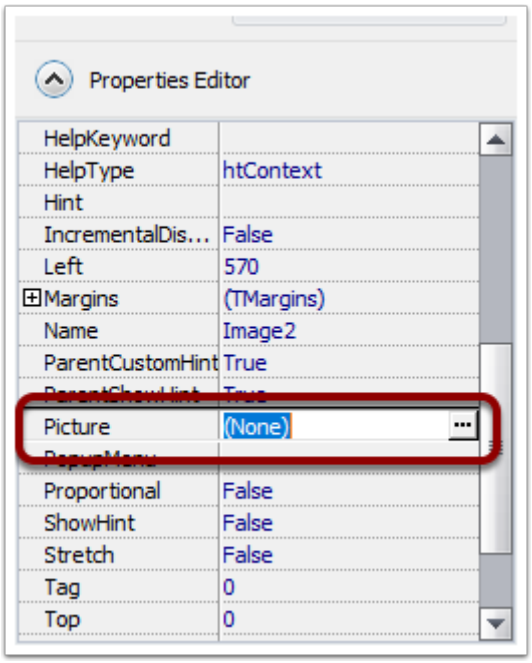




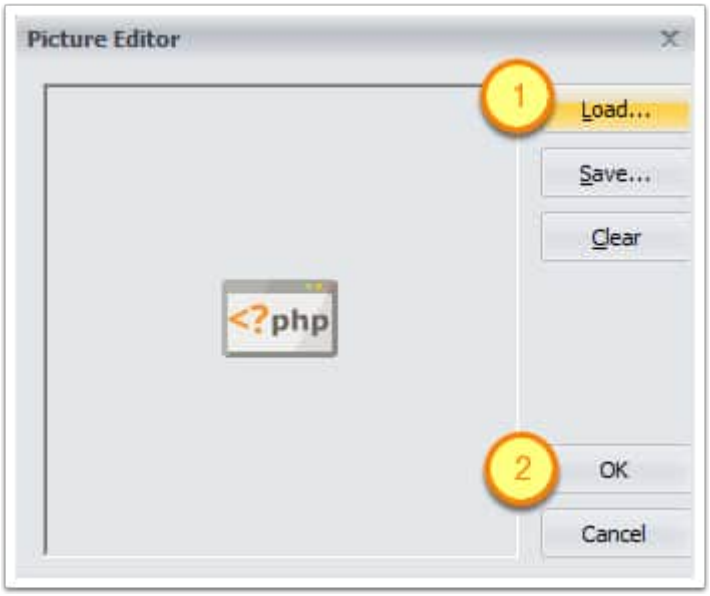
3. Modify the properties of the new image component. For example, set `Align` to `alRight` and `Center` to `True` (or `Stretch` to `True`).



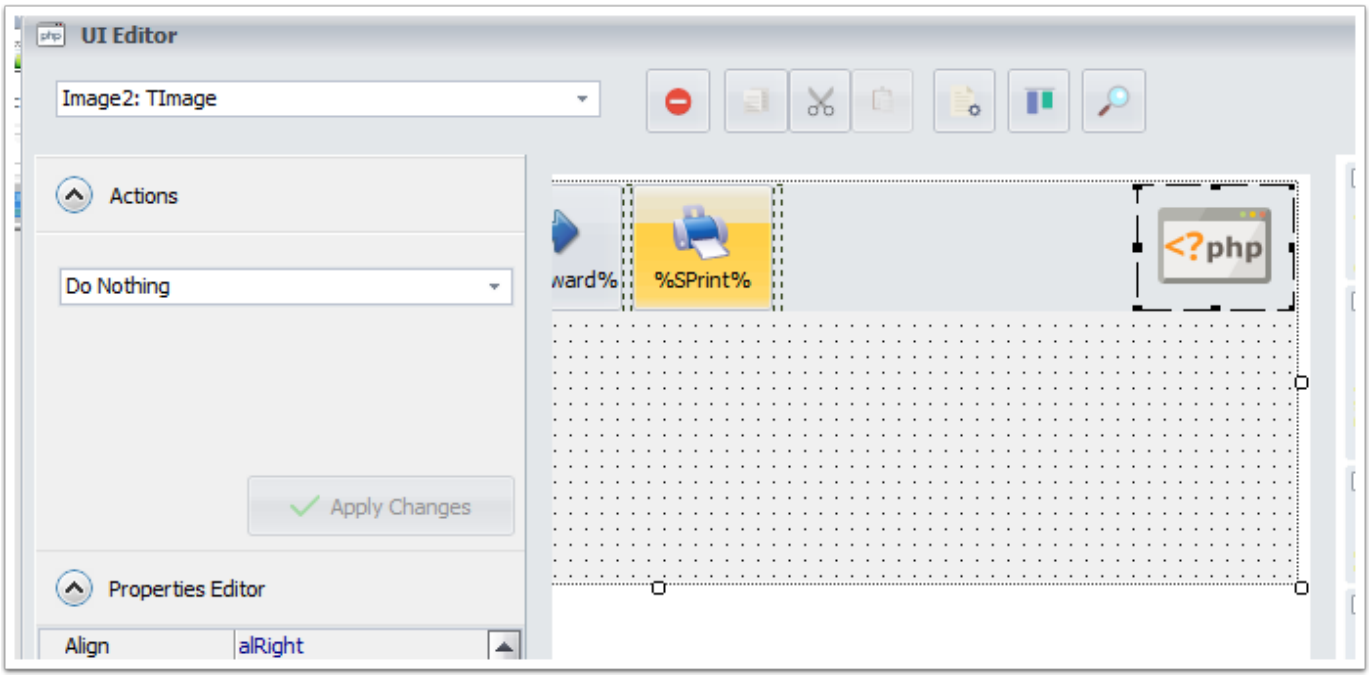
4. Click the `...` button next to the `Picture` property to open the image editor.



5. Click **Load** to select your image file, then click **OK**.



Your image is now loaded into the `TImage` component.



10.13 Using Timers and Cron Jobs in Your Application

ExeOutput for PHP allows you to create **cron jobs** using the TTimer component. This advanced feature allows you to **run HEScript functions or PHP code in the background** at regular intervals.

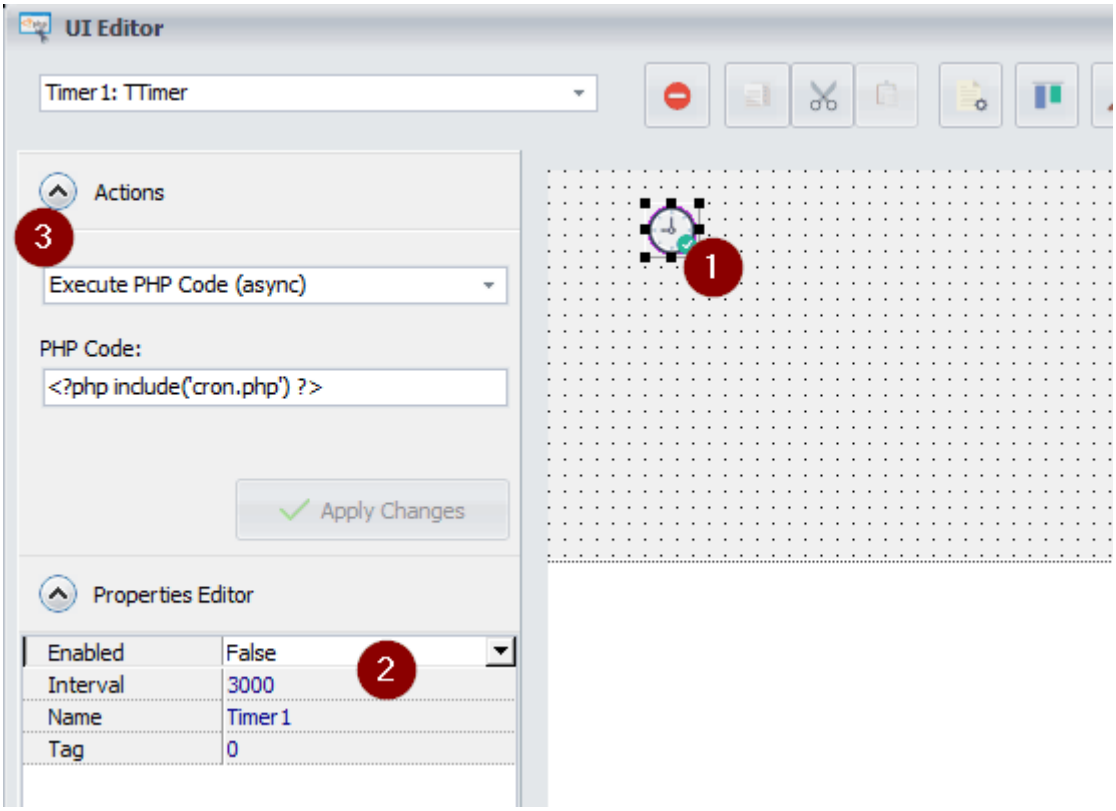
10.13.1 Creating a Timer / Cron Job

To **create a timer**, add a "Timer (cron)" to the "Components Used" list and then click **Edit Timer**.

The UI editor will appear, displaying a ready-to-use timer that you can configure.

10.13.2 Configuring a Timer

1. In the UI editor, select the Timer component (**Step 1**):



2. Configure the properties of the Timer component (**Step 2**):

- **Enabled**: When `False`, the timer is not running. It is recommended to only enable timers programmatically (see below).
- **Interval**: Specify the interval in **milliseconds** (for example, 1000 equals one second). This determines how frequently the defined action occurs. Each time the specified interval passes, the action is triggered.

3. Define the action to be executed by the Timer component (**Step 3**). The only actions useful for timers are **Execute PHP code** and **Execute an HEScript function**. Learn more about control actions.

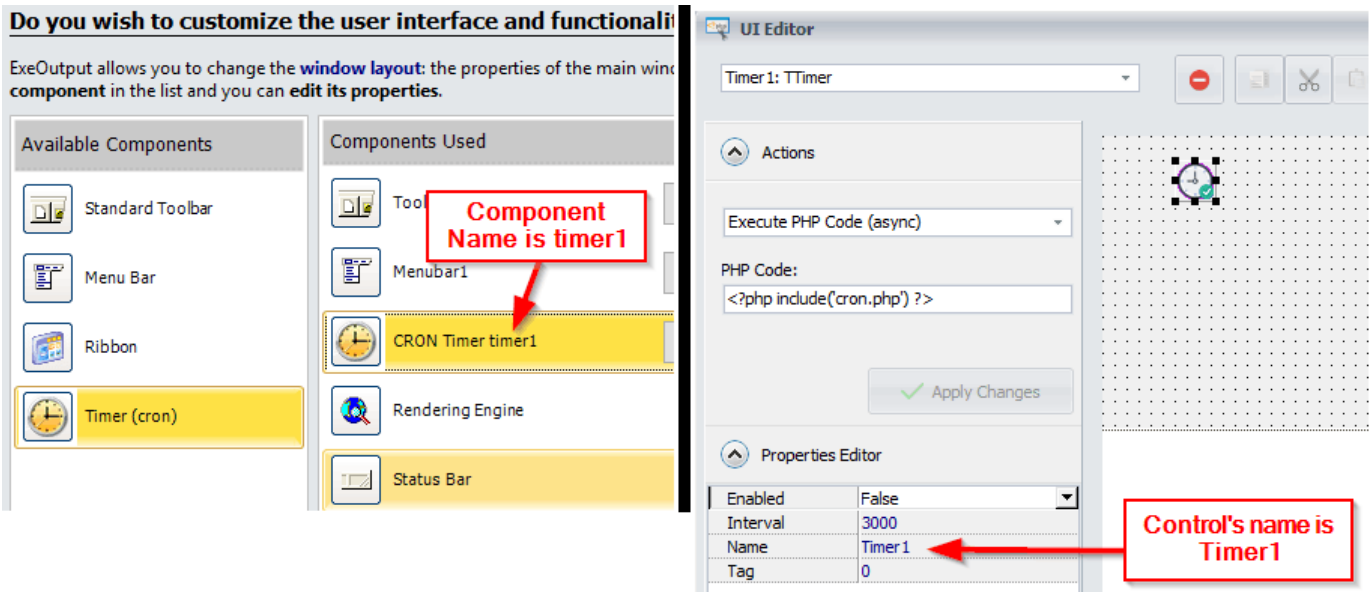
   For example, to configure the timer to execute a PHP script, ensure the script is available in the application's root folder. Then, use the PHP `include` command as shown in the screenshot above (Step 3):

```
<?php include('cron.php'); ?>
```

10.13.3 Starting and Stopping Timers Programmatically

👉 To **start a timer component**, use the StartTimer function defined in HEScript.

For instance, to start the timer named `Timer1`, pass its reference, which is `Component Name` + `Control Name`. In our case, it is `timer1Timer1`.



In this example, we want to start the timer when the application starts. In the UserMain script, insert the following code:

```
procedure OnStartMainWindow;
begin
  // When the main window is going to be displayed (just before the homepage is shown).
  StartTimer("timer1Timer1", 15000); // Starts the timer named Timer1 in the timer1 component and sets its interval to 15000ms.
end;
```

> **Warning**
>
> Once a timer is started, it will not stop until the application is closed or you explicitly stop it with the `StopTimer` function.

👉 To **stop a timer component**, use the StopTimer function defined in HEScript.

```
procedure StopIt;
begin
  StopTimer("timer1Timer1");
end;
```

11. Security

11.1 Security - Global Protection

To view your compiled PHP website, users must launch the .exe file. It is not possible to unpack a compiled application with a file archiver (like WinZip or 7-Zip). Ensure you back up your source files, because they cannot be extracted from a compiled application.

Applications built with ExeOutput for PHP feature several **security options** that you can configure on this page.

👉 See also additional security options and recommendations for PHP files.

11.1.1 Global Password

To restrict access to your application, you can **password-protect** it. Users will be prompted for the password before the application starts. If an incorrect password is entered, an error message is displayed (defined by the `SInvalidPassword` resource string), and the application closes immediately.

You can also customize the application's behavior for incorrect password entries using the `UserMain.OnInvalidPasswordAtStartup` HEScript boolean event. The application invokes this event if the user provides an incorrect password. If you set the event's result to `True`, the application will not exit.

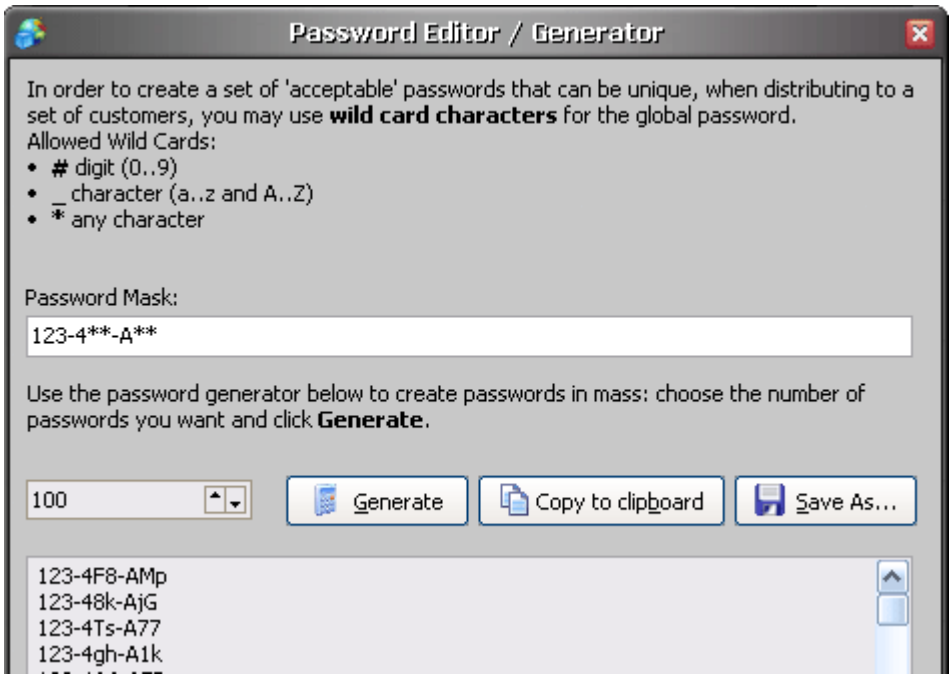
```
function OnInvalidPasswordAtStartup: Boolean;
begin
  Result := True;
end;
```

To create a set of unique passwords for distribution to different customers, you can use **wildcard characters**. For example, you can set a global password like `123-4**-A**`, where any character can be substituted for the wildcard placeholders.

Allowed Wildcard Characters

- `#`: Any digit (0-9).
- `_`: Any single letter (a-z, A-Z).
- `*`: Any character.

ExeOutput for PHP includes a **password generator** that creates lists of random passwords based on a provided mask.



Choose the number of passwords to create and click **Generate**.

11.1.2 Set a Global Expiration Date

If you want your application to expire after a specific date, select the desired expiration date. After that date, the application will display an error message (defined by the `SPublicationExpired` resource string) and close immediately. Changing the system clock will not affect the expiration state.

When testing on your own computer, you can remove the expiration state by clicking **Clear expiration info**. This function only works if the application has already expired.

Finally, you can customize the application's behavior when the expiration date is reached using the `UserMain.OnExpiredPublication` HEScript boolean event. The application invokes this event, and if you set its result to `True`, the application will not exit.

```
function OnExpiredPublication: Boolean;
begin
  Result := True;
end;
```

11.1.3 Check Application Size at Startup

If an application is not downloaded successfully from the internet, it may result in a truncated file. Running an incomplete application may be unsafe. To prevent issues from truncated downloads or modified file sizes, you can enable this option. When enabled, it forces the application to verify its size at startup. If the size does not match the size at build time, an error message is displayed.

Note that this option is superseded by digital signatures. If you have a code signing certificate, it is better to sign your application digitally.

11.1.4 Disable Print Screen

The Print Screen key allows Windows users to capture the entire screen to a bitmap (a screenshot). This screenshot is saved to the clipboard, and users can then paste it into any image editor or word processor. To disable this functionality, turn on the **Disable Print Screen** option. Pressing the key will no longer take screenshots while the application is running.

Notes:

- However, this function does **not** stop dedicated screen capture tools.
- Some programs (like screen capture tools) may also try to override the Print Screen hotkey, which could cause conflicts.

11.1.5 Disable Printing Feature in Entire Application

This option disables printing functionality across the entire application. When this option is enabled, users will be unable to print any content, including HTML pages and PDF documents. This ensures that no part of the application's content can be printed.

> **Note**
>
> This option does not disable the Print button on any toolbars or ribbons. You must remove any Print buttons manually; otherwise, clicking them will do nothing.

11.1.6 User Action Restrictions

To enhance the security of your application, ExeOutput for PHP disables certain user actions by default:

- **View Source (Ctrl+U)**: The keyboard shortcut `Ctrl+U`, which normally allows users to view the HTML source code of a page, is disabled to protect your code.
- **Drag and Drop to Popups**: Dragging external files and dropping them into pop-up windows is disabled.

11.1.7 Allow Only One Instance of the Application

Enable this option to ensure that only **one** instance of your application can run at a time. If a user tries to run a second instance, it will immediately exit, and the original instance will be brought into focus.

Moreover, command-line arguments are passed directly to the running instance. This feature is useful for help files, as it lets you change the current topic, for instance, without having to close and restart the application.

11.1.8 Do Not Hide Virtual Files in Dialog Boxes

By default, ExeOutput for PHP sets the "hidden" attribute for all virtual files in the `Data` subfolder (see Accessing Source Files from PHP). Thus, these files will not appear in the "Open" or "Save As" dialog boxes invoked when, for instance, a user chooses a file to upload.

If you want these files to appear for any reason, enable the **Do not hide virtual files in open/save dialog boxes** option. However, please note that virtual files can be copied to physical folders from these dialog boxes (though you can still encrypt your PHP source files).

Note: You can also try the option to Use an absolute path for the virtual "Data" subfolder.

11.1.9 HTTP Authentication

For applications that require user registration and login, ExeOutput for PHP provides support for HTTP Basic Authentication. This allows you to protect specific pages or your entire application with a username and password.

11.2 Security - PHP Protection

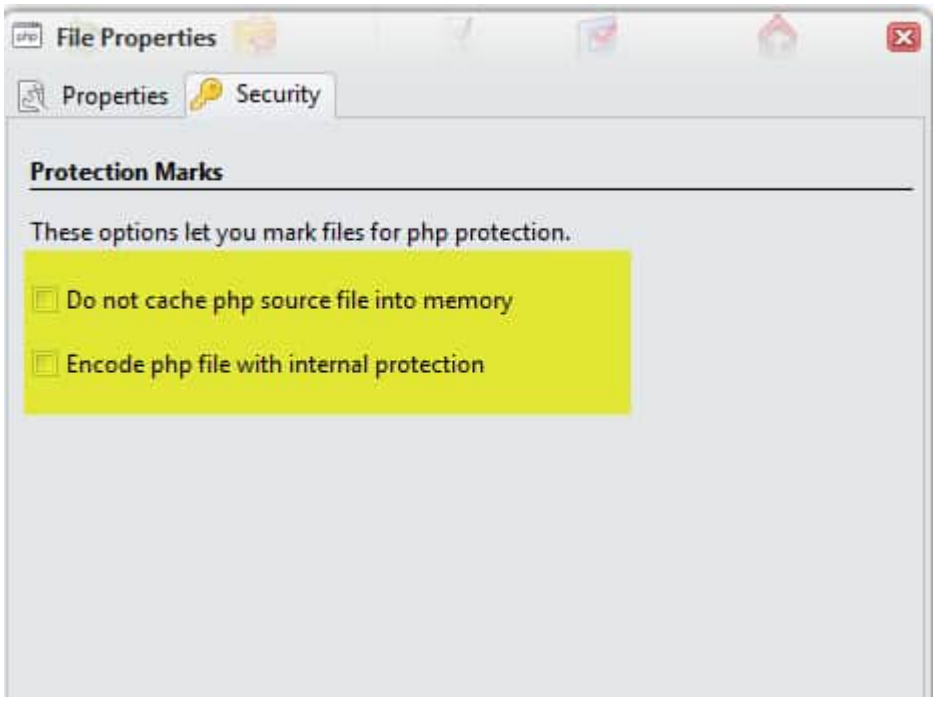
During compilation, ExeOutput for PHP **compresses and encodes source files** into the final executable file. The resulting EXE file cannot be unpacked with an archiver like WinRAR or 7-Zip. When you run the compiled application, files such as PHP pages, HTML, images, and JavaScript are never unpacked to the hard disk. Therefore, it is not possible for an average user to copy them.

Since PHP scripts must be unpacked into memory to be interpreted by the PHP runtime, it may be possible for a skilled hacker to extract portions of the compiled PHP files. To make this task more complicated and time-consuming, ExeOutput for PHP includes several security measures.

> **Warning**
>
> We strongly recommend that you do not include private passwords, database login info, or other sensitive information in applications released to the public. Instead, use encryption, server authentication, HEScript calls, or at a minimum, the string protection feature.

Moreover, ExeOutput for PHP provides **additional security options for sensitive PHP scripts**.

These **options are global**. Since they should not be applied to all PHP scripts, **you must mark the specific scripts** that require protection. This is done using "Protection Marks," available in the **Security** tab of the File Properties window in the File Manager:



11.2.1 Encode Marked PHP Files with the Internal Protection System

ExeOutput for PHP encrypts the PHP source file so that it does not appear in clear text in memory, although the script remains functional.

Encoding is performed while ExeOutput for PHP compresses the files. The original files are not replaced; instead, they are encoded into memory and then compressed into the final EXE.

> **Note**
>
> The internal protection system may not be compatible with all PHP files. In such cases, ExeOutput for PHP may fail to perform the conversion correctly. If an error occurs, it is logged in the compilation log, and ExeOutput for PHP will compile the original, unencoded PHP source file instead.

11.2.2 Do Not Cache Marked PHP Files in Memory

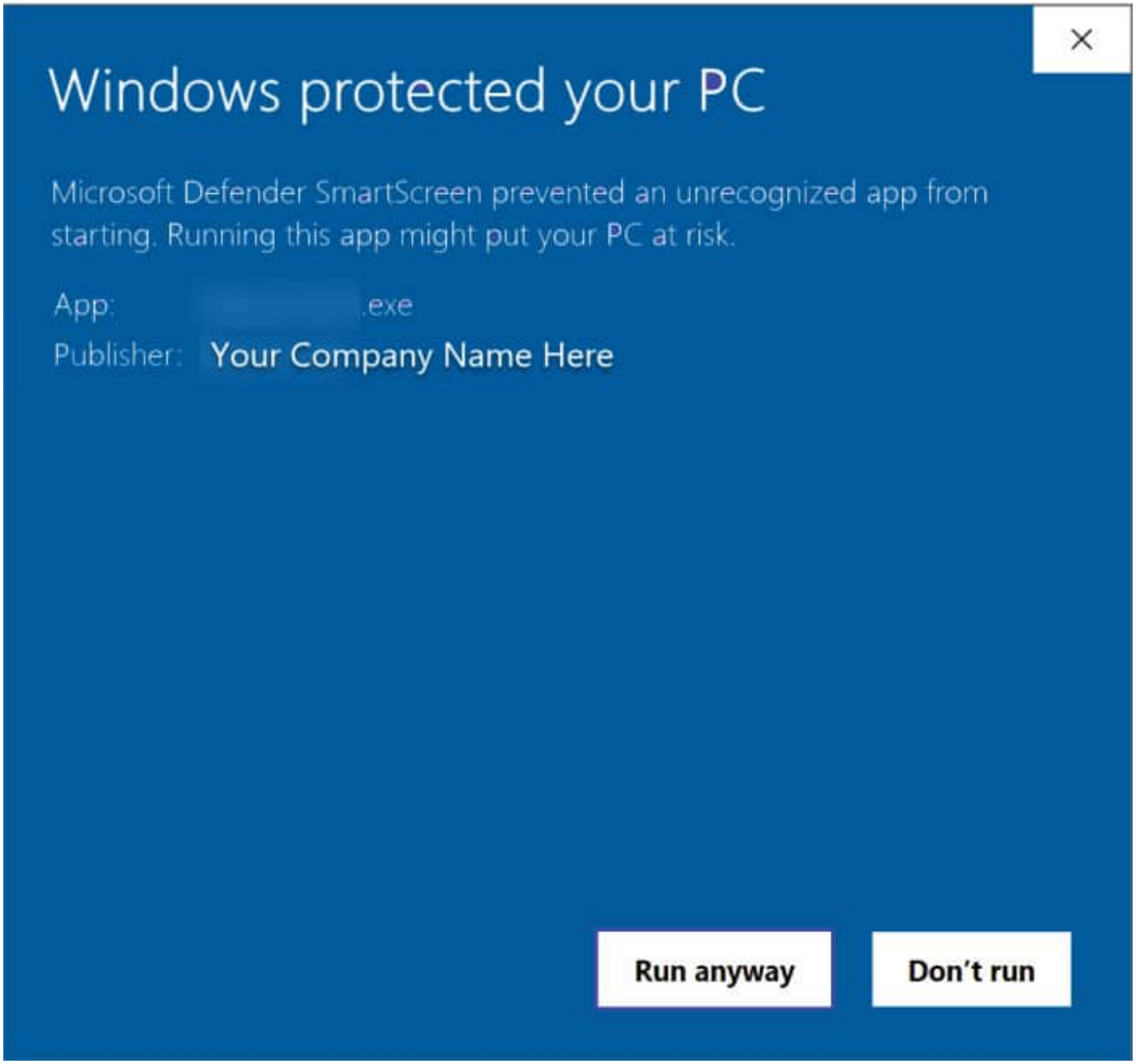
PHP scripts are unpacked into memory to be interpreted by the PHP runtime. Since some PHP scripts (such as includes) may be required multiple times, the runtime module caches them in memory for better performance. This means they remain in memory until the cache is full or the application is closed. This makes the application more responsive, as the decompression step is skipped on subsequent requests.

This option lets you **specify which PHP scripts should not be kept in memory after execution**. Note that in this case, the application will have to decompress these non-cached scripts every time they are requested by the PHP runtime.

11.3 Security - Code Signing (Digital Signatures)

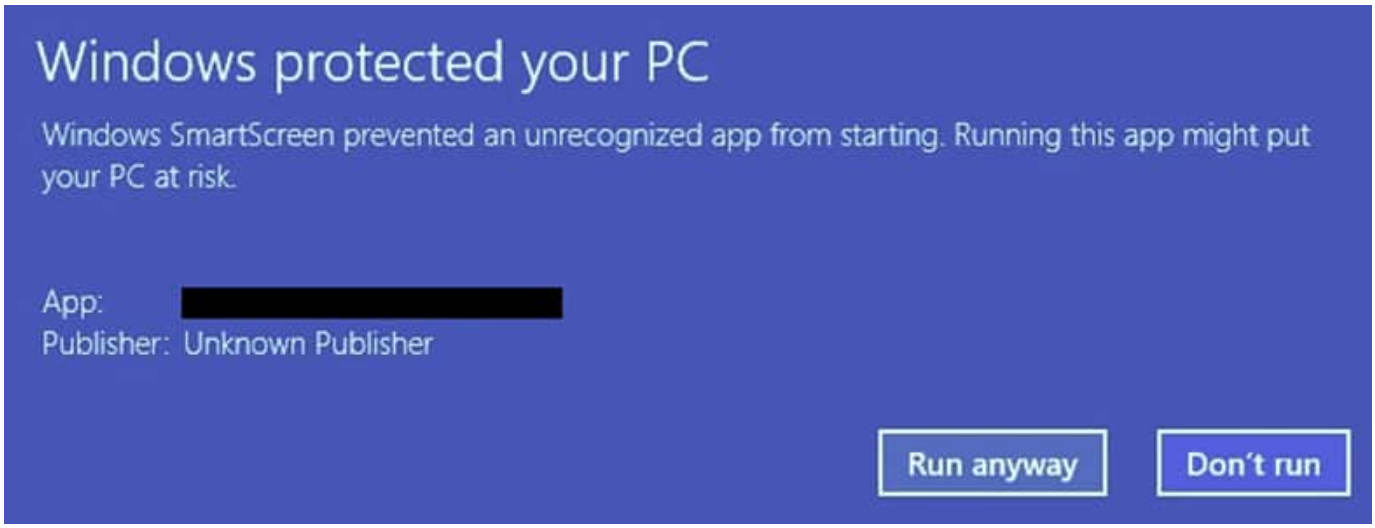
When you digitally sign an application (a process known as code signing), you assure users that the code has not been tampered with or altered. Digital signing is based on *Microsoft Authenticode® technology*. This enables both users and the operating system to verify that the program code comes from the legitimate publisher. ExeOutput for PHP makes it easy to **sign your compiled application .exe files** by calling the necessary programs automatically.

If you digitally sign your software, users are generally presented with a **digital certificate** when they download your application:



For **signed applications**, the publisher's name is displayed. Your users will know that the .exe file is authentic and has not been tampered with.

For **unsigned applications**, Windows shows the following warning message:



To **digitally sign your application**, enable the "**Digitally sign my application**" option in **Security -> Code Signing** and follow the steps below.

> **Info**
>
> This Microsoft article explains most of what you need to know about code signing with Authenticode: Introduction to Code Signing

> **Warning**
>
> Current Windows limitations prevent the signing of EXE files larger than 2 GB. If code signing is a requirement and your EXE file exceeds 2 GB, consider keeping some files external.

11.3.1 How to Obtain a Code Signing Certificate

To sign your application, you need a valid code signing certificate from a trusted Certificate Authority (CA), such as Sectigo or DigiCert. CAs offer different types of certificates, but only code signing certificates are compatible with Authenticode.

You can only digitally sign your .EXE after you have received your certificate and token from a CA.

11.3.2 Steps for Code Signing

ExeOutput for PHP simplifies the code signing process with its integrated `GSignCode.exe` utility. No third-party software installation is necessary. Follow these steps to sign your application:

1. **Specify the location of your code signing certificate**, either by providing the path to the Personal Information Exchange (PFX) file or by selecting the certificate from the Windows Certificate Store (Local Computer, Personal section). You must provide either the path to the PFX file, the certificate's subject name, or the certificate's thumbprint.
2. **If using a PFX file, enter the associated password** for added security.
3. **Alternatively, specify the certificate's subject name or thumbprint** for direct access from the Windows Certificate Store.

Application Information URL

This URL is included in your digital certificate to direct users to a webpage where they can learn more about your product or company. If you do not specify a URL, ExeOutput for PHP will use the default from the **Icon / Version** page.

11.3.3 Code Signing with a Token

Following changes implemented by the Certificate Authority/Browser (CA/B) Forum, effective June 1, 2023, the code signing process has shifted significantly. The forum now mandates that code signing certificate keys be stored on a hardware security module (HSM) or a token that meets or exceeds FIPS 140-2 Level 2 or Common Criteria EAL 4+ standards. This change is primarily aimed at combating the malicious use of stolen code signing keys to sign and distribute malware.

With this new requirement, the traditional PFX (Personal Information Exchange) format, which can be stored and accessed digitally, is becoming obsolete. Instead, it is recommended to use the subject name or thumbprint of the certificate after installing it (as a .CER file) in the personal Windows certificate store.

ExeOutput for PHP handles code signing with a required token without issue. Just ensure the token containing the private key is physically inserted into the computer.

> **Tip**
>
> If your CA uses the SafeNet client, you will be prompted for your password with each code signing instance. To streamline this process, you can activate the "Enable single logon" option. This setting requires the password to be entered only once per session, rather than for each signature, thereby reducing redundancy.

11.3.4 Digest Algorithms

While SHA-1 is being deprecated due to security vulnerabilities, newer algorithms like SHA-256, SHA-384, and SHA-512 are recommended for stronger security. They are supported across all modern Windows systems. It is important to note that SHA-1 is also being deprecated and should no longer be used for code signing. Please choose the algorithm that meets your CA's specifications.

Dual Code Signing (SHA-256 and SHA-1)

It is now mandatory to use signatures with an SHA-256 message digest instead of SHA-1. However, older Windows versions like Vista or XP do not recognize SHA-256 signatures. In this situation, it is possible to add two signatures to the .EXE file in a process called “dual code signing”.

> **Warning**
>
> By default, ExeOutput for PHP will use dual code signing if run on Windows 8 or later. On Windows 7, an SHA-256 signature is used by default, and on previous Windows versions, an SHA-1 signature is used. Therefore, we recommend using ExeOutput for PHP on Windows 8 or higher to benefit from all code signing features.

Elliptic Curve Cryptography (ECC) Support

In addition to RSA, ExeOutput for PHP now supports certificates that use Elliptic Curve Cryptography (ECC). ECC certificates offer stronger security with shorter key lengths, making them more efficient. For instance, a 256-bit ECC key provides security comparable to a 3072-bit RSA key, enhancing both performance and security.

11.3.5 Digital Signature Timestamp

A timestamp is added to your application's digital signature to ensure it never expires, even after the signing certificate has expired. Ensure your system has an internet connection during the signing process for this time-stamping to work.

Two timestamp servers are used: an Authenticode-compatible server and an RFC-3161-compatible server. You can configure their URLs in the Environment Options.

11.4 Security - Licensing

ExeOutput for PHP secures your PHP applications by compiling them into single executable files.

These executables can also be protected with Obsidium. Obsidium offers powerful licensing features, allowing you to sell licenses for your applications.

Hardware-locked license keys are also supported, preventing customers from sharing their licenses. This helps to reduce piracy of your PHP apps.

> **Info**
>
> Please see our complete video tutorial on protecting and licensing your PHP apps with ExeOutput and Obsidium.

To **enable support for the Obsidium API in your PHP application**, you must check the `Enable third-party Obsidium software protection API` option on the Output Filename page. When enabled, ExeOutput for PHP will use a special version of the EXE stub.

> **Warning**
>
> You must first install the Obsidium API package using the Web Update utility.

If you protect your application with Obsidium, you can invoke the Obsidium APIs directly from PHP or HEScript. The Obsidium API package includes a script file named `obsidiumAPI.xml` that can be imported on the Scripts page. This script file is installed in the `CEFRuntime` subfolder of the ExeOutput for PHP installation directory.

> **Info**
>
> Feel free to watch the video tutorial on importing and invoking Obsidium APIs in your ExeOutput for PHP app.

12. Application Output

12.1 Application Output Settings

12.1.1 Output Path

The **Output Path** specifies the location where ExeOutput for PHP will create the executable file for distribution.

In the **Output Path** field, provide the full path for the final application file, including the directory and filename. This file must have a `.exe` extension.

Notes:

1. It is recommended that you save your project before compiling.
2. If the application file already exists, it will be overwritten without warning.
3. If the output folder does not exist, it will be created automatically.
4. Additional subfolders, such as `Data`, may be created to store external files.
5. Ensure the output folder is not read-only, especially on Windows versions with User Account Control (UAC) enabled.
6. All resources (e.g., splash screens) and source files must be present during compilation.
7. Ensure you have sufficient free disk space, as ExeOutput for PHP does not check for available storage before compiling.

How to Compile Your Project

12.1.2 Application Title

ExeOutput for PHP requires a title for display in message boxes and window title bars. The application title should be a short, descriptive phrase, such as "My Wonderful Site" or "Demonstration for E=mc²". This title will appear in the Windows taskbar and Task Manager.

12.1.3 Application GUID

Similar to how an ISBN identifies a book, the **Application GUID** uniquely identifies your application, allowing it to store and manage its settings on the user's computer. To generate a new GUID, click the button to the right of the field.

Note: Do not change the GUID after starting a new project. If you want to share settings between different applications, assign them the same GUID.

Request Elevated Rights (User Account Control)

Configure how elevated rights are requested on Windows versions with User Account Control (UAC). If your application needs to perform tasks that require administrative privileges, you can set the appropriate execution level.

You have three options for setting the requested execution level:

- **As Invoker** (Default): Turn off the **Request Elevated Rights** option. The application runs with the same access token as the parent process.
- Turn on the option and choose between two levels:
- **Require Administrator**: The application will only run for administrators and must be launched with a full administrator access token.
- **Highest Available**: The application runs with the highest privileges the current user can obtain.

12.1.4 Enable Third-Party Obsidium Software Protection API

If you protect your application with Obsidium software protection, you can invoke its API directly from PHP or HEScript. Refer to our complete video tutorial on adding licensing features to your PHP apps with ExeOutput and Obsidium.

To enable support for the Obsidium API in your PHP application:

1. Check the option **Enable Third-Party Obsidium Software Protection API**. This allows ExeOutput for PHP to use a special version of the EXE stub.
2. Ensure this special version has been downloaded and installed with the Web Update utility.

Please also read the licensing topic dedicated to Obsidium.

> **Warning**
>
> **This option is not compatible with UPX compression.**

12.1.5 Use the Same EXE Filename for PHP Child Processes

By default, PHP child processes created by ExeOutput for PHP have a standard name. This can sometimes cause issues with Windows security solutions. In such cases, you can enable the **Use the Same EXE Filename for PHP Child Processes** option.

Important: Enabling this option may lead to occasional, random crashes of PHP processes. If you experience such crashes, do not enable this option. It is disabled by default.

12.1.6 Disable Multithreading for Troubleshooting Purposes

By default, ExeOutput for PHP enables multithreading support. Each PHP script called by the internal browser is executed in a separate thread or process, independent of the main UI thread. This ensures that the UI remains responsive.

If you encounter issues, you can disable multithreading to help diagnose the problem:

- **Disable Multithreading**: This executes all PHP scripts on the main thread, which can help identify if multithreading is the source of conflicts or other issues.

12.2 Output - Deployment Options

ExeOutput for PHP is designed to generate the smallest possible application files, primarily through file compression.

12.2.1 Compressing Application EXE Files

If you plan to distribute your application online, we recommend compressing it to minimize its size and download time. Several methods are available:

- **Compress the final `.EXE` file with UPX**: UPX is a free executable packer available at upx.sourceforge.net. ExeOutput for PHP can automatically call UPX if you enable this option.

  > **Note**
  >
  > For legal reasons, UPX is not included with ExeOutput for PHP. You must download the program from upx.sourceforge.net and extract the archive into the `UPX` subfolder (e.g., `C:\Program Files\ExeOutput for PHP\UPX`). Alternatively, use the **Web Update utility** ("Check for updates" option) to automatically install this add-on.

  Finally, if you enable Obsidium API support, UPX compression will be ignored.

- **Distribute the final `.EXE` file using an installer**: You can use Paquet Builder to create an installer for your `.EXE` file, optimized for online distribution.

  See Distributing Applications.

12.2.2 Do Not Cache CEF Files Locally

Chromium Embedded Framework (CEF) runtime files are large (approximately 50 MB), which typically results in several seconds of decompression during application startup. To eliminate this decompression step and speed up loading, the application will decompress and **store CEF runtime files in a shared cache folder the first time it runs**. If the runtime files already exist, they are used directly without re-decompression.

Learn more about the CEF engine.

If you prefer not to store CEF runtime files (e.g., to save disk space) on the user's computer, you can enable the **Do not cache CEF files locally** option. However, this will cause the application to take longer to load on subsequent runs.

The cache folder is shared across all applications built with ExeOutput for PHP. By default, the cache folder path is:

```
C:\ProgramData\GDG Software\ExeOutput for PHP App Cache\CEFXXXX
```

where `XXXX` represents the version number of the Chromium Embedded Framework used in ExeOutput for PHP.

A different name will be used if you have configured a custom name for the storage folder (see below).

12.2.3 Leaving CEF Runtime Files Outside the EXE

ExeOutput for PHP does not compile the essential Chromium Embedded Framework runtime files into your application's `.EXE` file, saving approximately 40 MB in size. However, the application will not function if these files are missing, and you will encounter the error message: `missing CEF3 files!`

You can use this option only if you are certain to install these files yourself. For example, these files can be installed automatically using the `exocefruntimeXXXX.exe` installer available in the `Redist` subfolder of your ExeOutput for PHP installation. Alternatively, you can find them in the `CEFRuntime` subfolder of your ExeOutput for PHP installation. The files must be placed in the **shared cache folder** (see the section above).

12.2.4 Portable Applications

ExeOutput for PHP can create **portable versions of your applications**.

- No installation is required; your application is stored on a removable device, such as a USB flash drive, allowing it to be used on multiple computers.
- User preferences and application settings are stored with the software (i.e., they are written to the USB drive). Thus, users retain their preferences even when running the application on different PCs.
- The Windows Registry is not used.
- No permanent modifications are made to any PC after the application is used.

Portable applications create one **data file** in addition to their `.EXE` file:

- The state file (user preferences, global variables, etc.) is named: `[name of the EXE].userpref`

The state file is saved in the same folder as the `.EXE` file. Consequently, the storage device **must not be read-only**. If the application is unable to write the file in the same folder as its `.EXE` file, it will use the default location on the hard disk. The default location is a subfolder in the User Data directory, which you can customize.

If you do not create a portable version, you can specify a custom name for the **storage folder where the application will store its settings**. By default, it will be a subfolder in the User Data directory. You can obtain the full path at runtime from the global variable `HEPubStorageLocation`.

Explore Storage Folder

To easily access this folder for testing and debugging, you can click the **Explore Storage Folder** button on the Deployment page. This will open the storage folder for the last compiled application in Windows Explorer.

Warning

Avoid using expiration features for portable applications. Portable applications store their settings on the USB drive. Consequently, trial settings are also saved on the drive, allowing users to easily reset their trial period by removing the settings files.

12.3 Application Loading Screens

During initialization, applications can show a **splash screen** (an image displayed briefly at the start) and/or a **"Please Wait..." dialog box**. A splash screen is useful for displaying your company's logo or branding, while a dialog box informs users that the application is starting. Because the initialization phase can take several seconds due to the size of the Chromium engine, this dialog box lets users know the application is loading and prevents them from trying to launch it again immediately.

12.3.1 Splash Screen

To display a splash screen, go to **Application Settings -> Loading Screen** and turn on "**Display a splash screen at startup**".

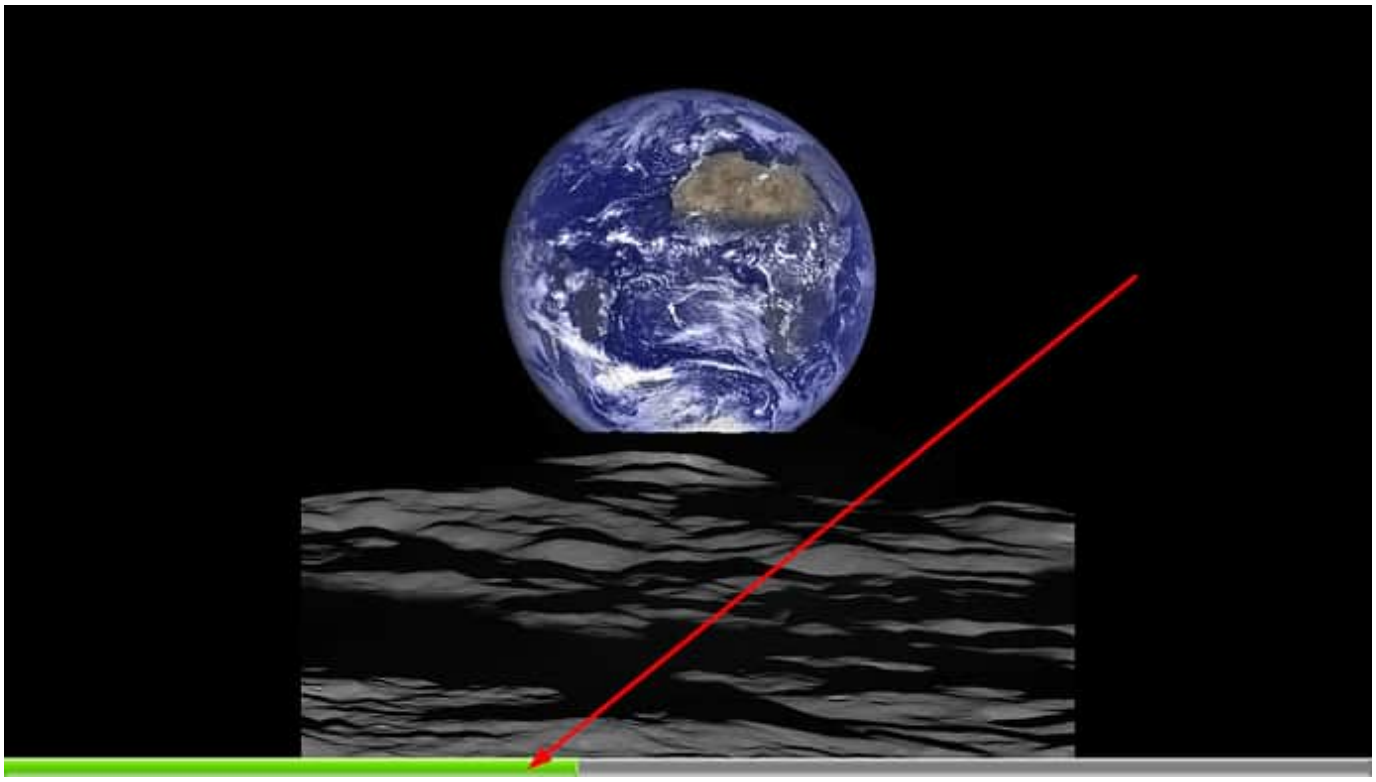
Then, you must select the image file for your splash screen. It can be a BMP, JPG, PNG, or GIF (with animation support) file. We recommend using 32-bit PNG files for transparency effects.

- **Image file**: The full path to the image file.
- **Duration**: How many seconds the splash screen should be displayed.
- **Text color**: The color of the optional text that can be displayed at the bottom of the splash screen.
- **Display text at the bottom of the splash screen**: You can enter custom text here (accepts constants such as %APPVERSION%).

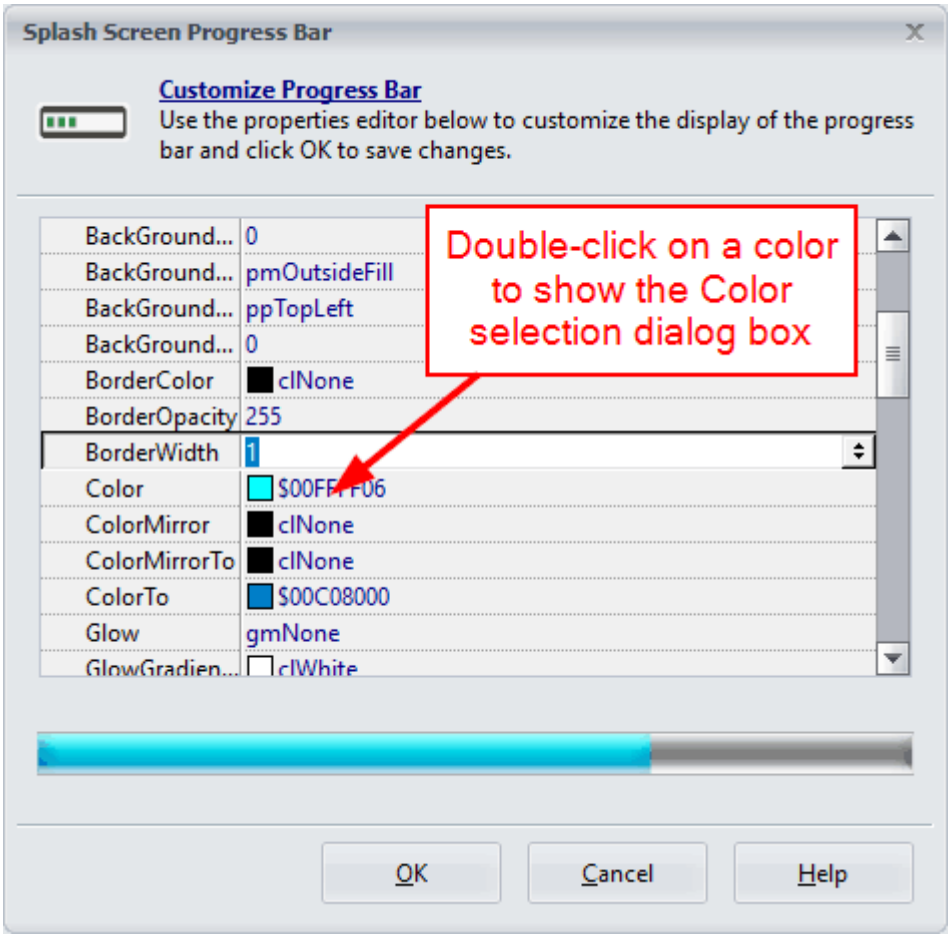
> **Info**
>
> ExeOutput for PHP supports non-rectangular, alpha-blended (semi-transparent) splash screens if you use 32-bit PNG files. This can give your application a unique look.

Splash Screen with Progress Bar

The splash screen can also serve as a progress indicator. With this option, a **progress bar is displayed at the bottom** to show that the application is busy.



Click **Customize Progress Bar** to edit the progress bar's color properties.



> **Info**
>
> This option is not compatible with the "Duration" setting. Moreover, the splash screen cannot be closed by users when in progress mode.

External Splash Screen File

By default, ExeOutput for PHP compiles the source splash screen file into the EXE. If you prefer a customizable splash screen, you can place your own image file in the `Data` subfolder and name it `splash.img`. The application will load this custom splash screen file instead of the default one.

12.3.2 "Please Wait" Dialog Box

To display a "Please Wait..." dialog box, go to **Application Settings -> Loading Screen** and turn on "**Display a 'Please wait' dialog box**".

- **Title**: The title of the dialog box.
- **Text**: The text of the dialog box. You can use constants such as %APPVERSION%.
- **Show progress bar**: If you want a progress bar to be displayed.

> **Note**
>
> The "Please Wait" dialog box closes automatically when the application is ready to display the main window and the homepage.

12.3.3 Notes

1. Try to use **small** splash screens. Large splash screens may take longer to draw because the data may need to be decompressed first (like for PNG or JPEG files).
2. The splash screen file is not stored in the project file; it must be available as an external file during compilation. Path variables like **[PROJECTPATH]** are supported.

12.4 Output - EXE Icon and Version Information

When compiling your application, ExeOutput for PHP creates a single executable file ready for distribution. Most Windows executable files (PE format) contain a **resource section** that stores version information, icons, cursors, string tables, and more. This page allows you to customize some of these resources to insert your own logo and copyright information.

12.4.1 Changing the EXE File Icon

ExeOutput for PHP allows you to **change the default icon of the executable file**. You can replace the default icon by specifying the **path to another icon file** (it must be a valid `.ico` file). ExeOutput for PHP supports various icon sizes and color depths (e.g., 32x32 with 16 colors, or 48x48 with 256 colors). If you need to create or extract icons easily, try GConvert, our companion tool that lets you extract icons or convert images into icons.

Tip

Vista icons are also accepted.

Note

The icon file must be available as an external file during compilation. Path variables like **[PROJECTPATH]** are supported. Some usable icon files are available in the **Resources** subfolder of your ExeOutput for PHP installation.

12.4.2 Version Information

The version information of an executable program is a special resource section that contains details such as its version number, intended operating system, original filename, and copyright information. This information is included in the compiled code. When version information is included, users can right-click the program icon and select **Properties** to display it (or press **Alt+Enter** in Explorer).

ExeOutput for PHP allows you to insert your own version information into the EXE file.

Some company information items are stored within the application data. They are displayed in the About box, when errors occur, and when setting version information parameters for the .exe output file. This helps users contact you for further information.

You must, at a minimum, specify the company's name. However, it is recommended that you provide users with as much information as possible about your company.

- **Company Name**: The name of your company.
- **Legal Copyright**: This will appear under the "Legal Copyright" entry. Enter your own copyright, such as "Copyright 2012 by Yourself. All rights reserved."
- **File Description**: A description of your application's contents.
- **Web Homepage**: The URL of your website.
- **Version Number**: The current release number of your .exe file. The format must be **X.X.X.X**, where X is an integer (e.g., 1.20.34.45).
- **Product Number**: The current release number of your application, using the same format as above.
- **Legal Trademarks**: The text that will appear in the Legal Trademarks field. This can only be changed if you purchased the "No Branding" option.

Tip

These information items and the default logo file can be set in the Environment Options. This ensures that the default settings are used each time you create a new application.

Auto-Increment Version Number During Build Process

If enabled, the last component of the **Version Number** will automatically increment each time you build your project. For instance, `1.0.0.0` will become `1.0.0.1`, then `1.0.0.2`, and so on. If the maximum value of 65535 is reached, the counter resets, and a warning is displayed in the compilation log. You must then manually update the version number. For instance, you might change it to `1.0.1.0`.

12.5 Output - Creating Installers or Zip Archives

ExeOutput for PHP generates single executable files that are ready for distribution. You only need to distribute the application's `.exe` file (and, in some cases, additional external files located in the `Data` and/or `MySQL` subfolders) to your users or customers.

After building your application, you may want to distribute it over the internet. You can choose to package your compiled application into a **Zip archive** or an **installer/setup package**. Installers are advantageous because they can install multiple applications, install redistributables, include additional files (like README documents), create shortcuts in the Windows Start menu, and offer an uninstaller to allow users to remove all traces of your application from their computer.

👉 Once an application is created, you can use the **Export to Zip** or **Create Installer** commands in **Application Output**.

12.5.1 Exporting to Zip

ExeOutput for PHP will prompt you to specify the location for the Zip archive. It will then scan for all files used by your application (including the `.exe` file, `Data` subfolder, and the MySQL subfolder) and compress them into the Zip archive.

> **Note**
>
> ExeOutput for PHP stores the correct path information within the Zip archive. You only need to deploy the Zip archive; users can extract it on their computer and run your application immediately.

12.5.2 Generating an Installer or Setup Package

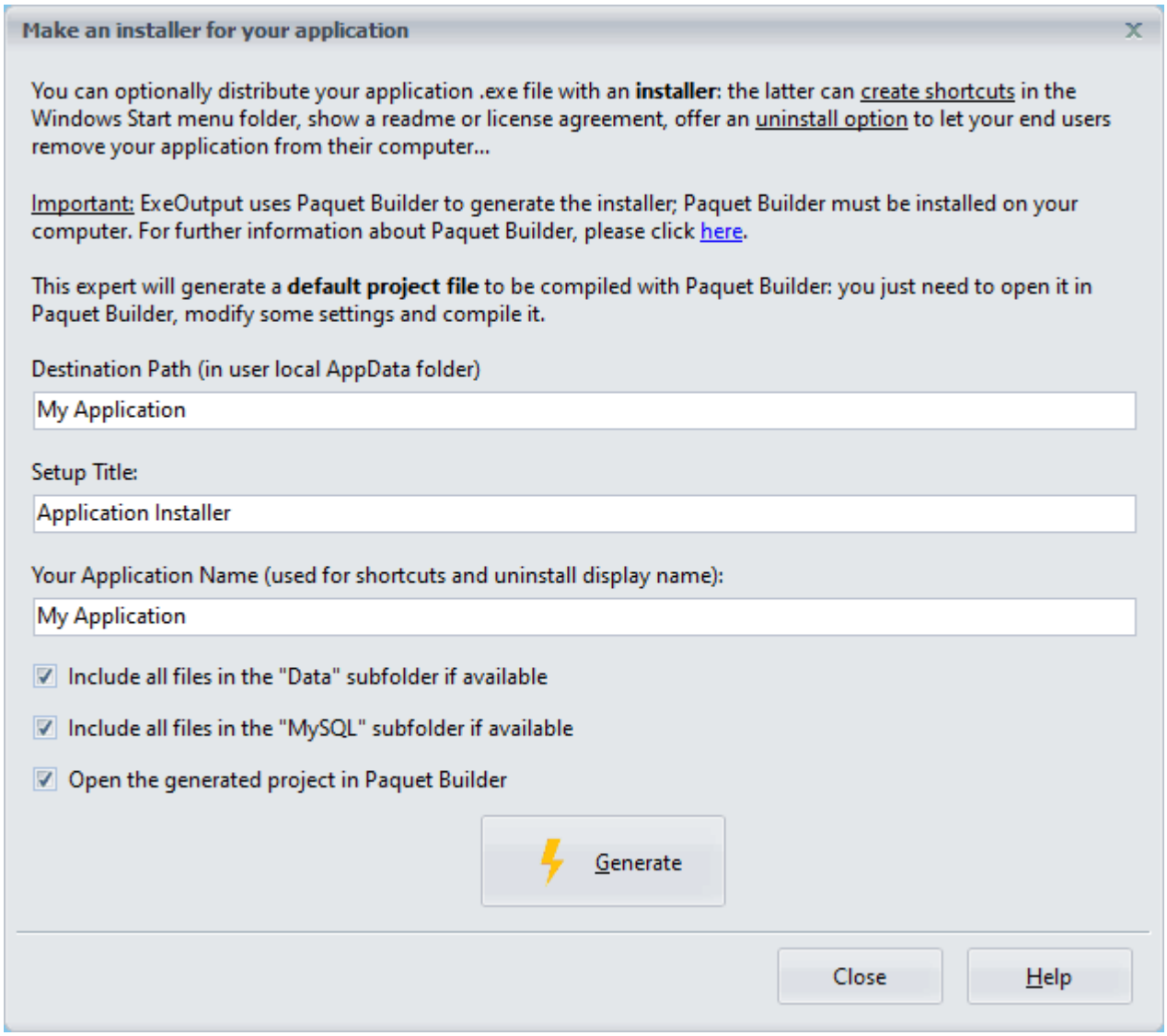
ExeOutput for PHP utilizes **Paquet Builder** to generate custom, compact setup programs for distribution. You must install Paquet Builder before using this feature.

Paquet Builder combines the functionalities of a 7z self-extracting archive creator and a setup routine generator. With its comprehensive feature set, you can create flexible and compact self-extractors for professional file and software distribution. Package any document or program files, visually construct simple or sophisticated multilingual distribution and installation packages, generate updates and patches, and wrap multimedia presentations or multiple Windows Installer MSI setups into single `.exe` files for internet distribution.

☑ More information about Paquet Builder can be found at https://www.installpackbuilder.com.

How It Works

You will encounter a window with the following fields:



ExeOutput for PHP will create a default project for Paquet Builder. You can then edit this project within Paquet Builder and compile it to create the setup package.

☑ First, fill in the three fields: **Destination Path**, **Setup Title**, and **Your Application Name**.

- **Destination Path**: The default installation folder for your application.
- **Setup Title**: The title that will appear on all setup windows.
- **Your Application Name**: The name of your application.

☑ Next, click **Generate** to create the project. ExeOutput for PHP will then launch Paquet Builder, allowing you to modify the project as needed.

> **Tip**
>
> ExeOutput for PHP will automatically include external files from your application's `Data` and `MySQL` subfolders, unless you disable this option.

12.5.3 Exploring the Storage Folder

ExeOutput for PHP applications store data such as settings, cache, and user preferences in a dedicated **storage folder** on the user's computer.

To easily access this folder for testing and debugging purposes, you can click the **Explore Storage Folder** button on the Deployment page in ExeOutput for PHP. This will open the storage folder for the last compiled application in Windows Explorer.

The location of the storage folder can be customized. See the deployment options for more details.

13. Scripting with HEScript

13.1 Introduction to Scripting with HEScript

ExeOutput for PHP applications are script-driven and feature a **built-in script engine**. An application is managed by a collection of scripts that are generated and compiled into p-code by ExeOutput for PHP. When the application runs, its runtime module executes these scripts to simulate a web browser and respond to user actions.

You can, therefore, extend the functionality of your applications by **writing and calling your own HEScript functions**. This feature offers a high degree of flexibility.

> **Info**
>
> Note that you are not required to work with scripting to use ExeOutput for PHP and compile applications. Scripting is an advanced feature for users who want full control over their applications.

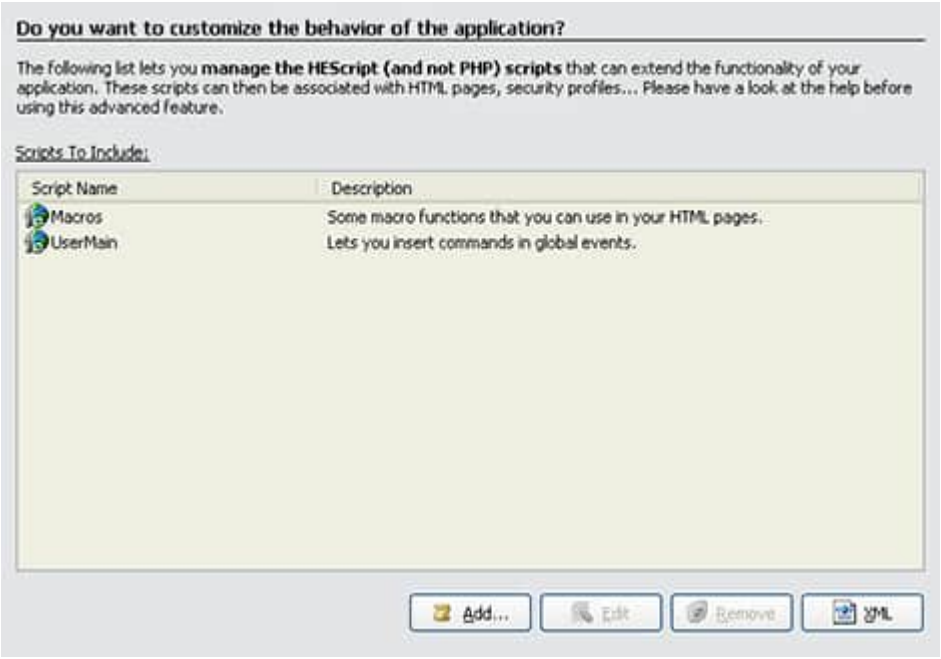
13.1.1 About the HEScript Language

The script language used by ExeOutput for PHP is called **HEScript**. It is based on the Object Pascal language syntax (similar to Embarcadero® Delphi and FreePascal) with some minor changes.

Unlike JavaScript and PHP, you cannot write HEScript functions directly into HTML or PHP pages. HEScript code must first be compiled into p-code. For this reason, you must use the **User Script Manager** to write and manage your scripts.

You can use HEScript in addition to PHP and JavaScript. In fact, all three can be easily combined.

13.1.2 The User Script Manager

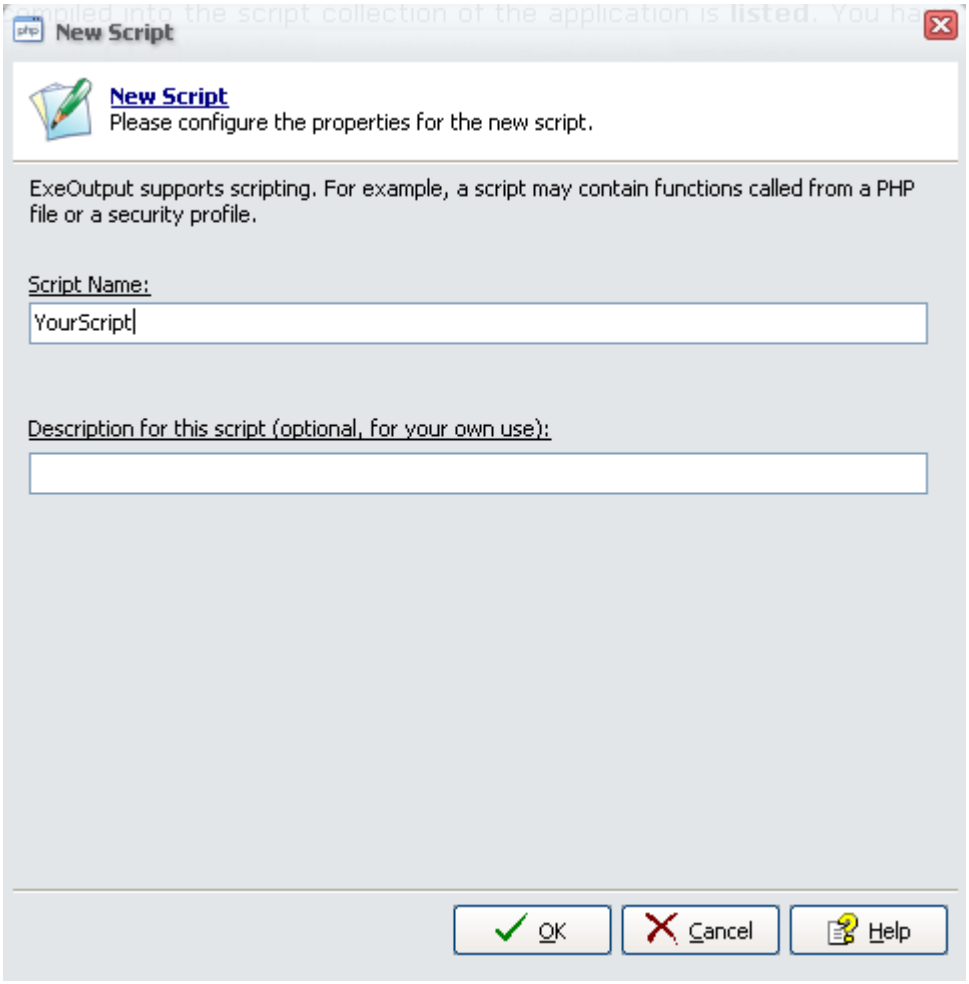


Each script compiled into the application is **listed** with its **name** and an optional **description**. Each script must have a **unique name**, which acts as a namespace.

👉 Quickly Add HEScript Code

To add a new script:

1. Press **Add**.
2. Provide a **name** (alphanumeric only, no spaces or special characters, 255-character maximum).
3. Optionally, enter a description.



Press **OK** to create your script. The script editor will open, allowing you to start coding. Alternatively, you can press **Save** and return to it later.

👉 More Information About the Script Editor 👉 More Information About the Predefined `UserMain` Script

Managing Scripts

- **Edit**: Select a script and press **Edit** (or double-click it).
- **Remove**: Select a script and press **Remove**. (The `UserMain` script cannot be removed).
- **Import/Export**: Select a script and press **XML → Import/Export**. Scripts are stored in XML format and can be edited with an external editor. Imported scripts are automatically checked for syntax errors.

13.1.3 Example: The `UserMain` Script

By default, ExeOutput for PHP automatically adds a **UserMain** script when you create a project. This script contains predefined global events related to the application.

```
// UserMain
// This script contains special functions related to some of the events triggered by the application.
// You can then optionally add new commands.

function OnBeforeNavigate(NewURL, TargetFrame: String): Boolean;
begin
  // Before the application displays a page.
  // Set Result to True to stop the operation.
  Result := False;
end;

procedure OnNavigateComplete;
begin
  // When a page has been displayed.
end;
```

A script file is, therefore, a **group of procedures and functions**. Each procedure or function name must follow these rules:

- Use only alphanumeric characters (no spaces).
- Each name must be unique within its script file.
- However, different script files can contain procedures or functions with the same name.

13.1.4 Important Notes and Best Practices

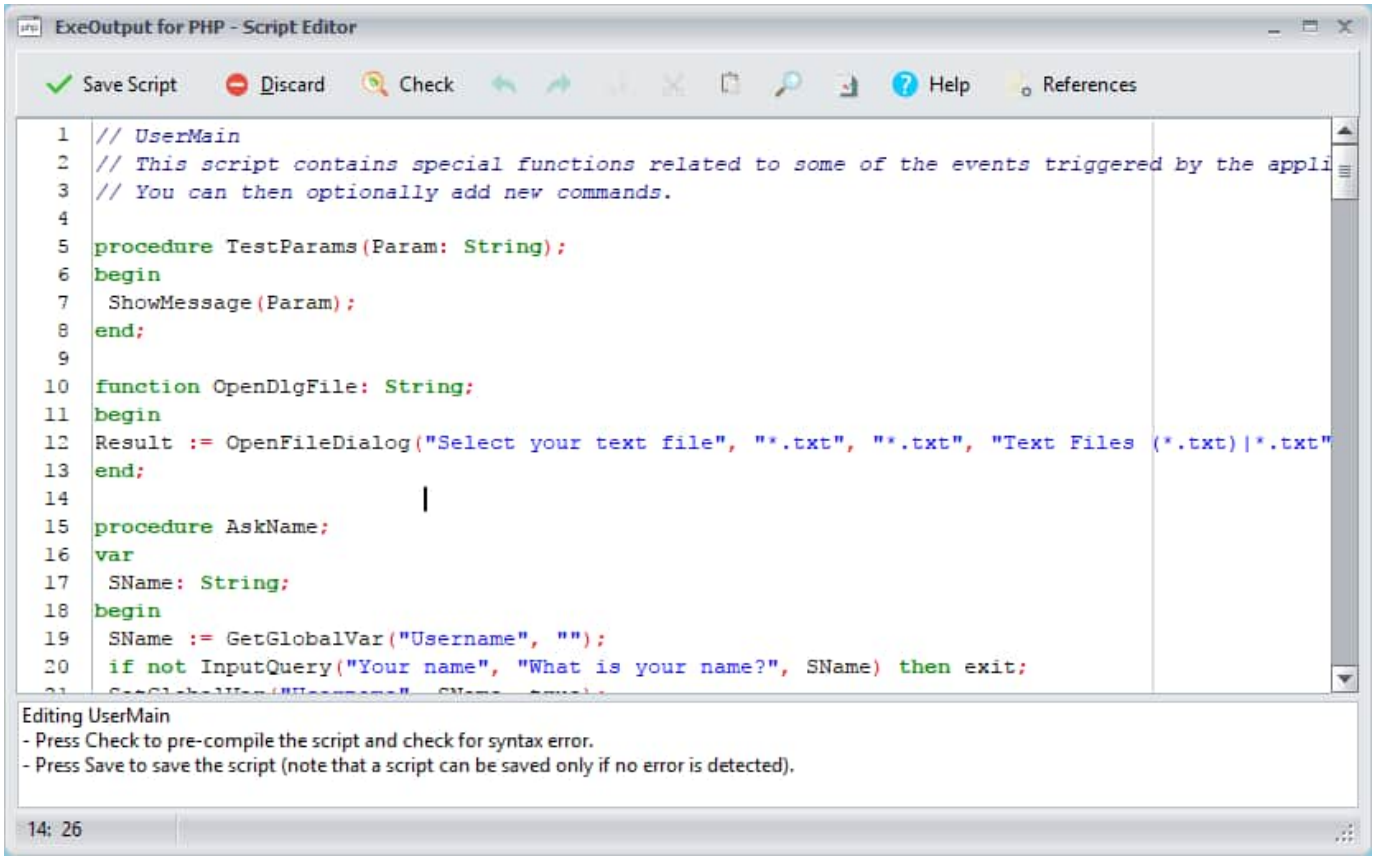
1. Script files act as **namespaces**. When calling or assigning a procedure/function to an event, use the following syntax: `ScriptName.FunctionName` (Example: `UserMain.OnNavigateComplete`).
2. Each script is managed by an **independent script engine**. Use global variables to exchange data between scripts.
3. Using local variables is possible but **not recommended**.
4. Scripts are stored inside the project file; no external files are required.
5. You can call HEScript functions from your PHP code, HTML links, and even from JavaScript.

13.2 The HEScript Editor

The **Script Editor** is the primary interface for editing HEScript code.

- You can create, remove, import, and export scripts with the Script Manager.
- To edit a script, select it in the Script Manager and click **Edit**, or simply double-click the script.
- The script editor is also displayed automatically when you import a script.

The script editor interface:

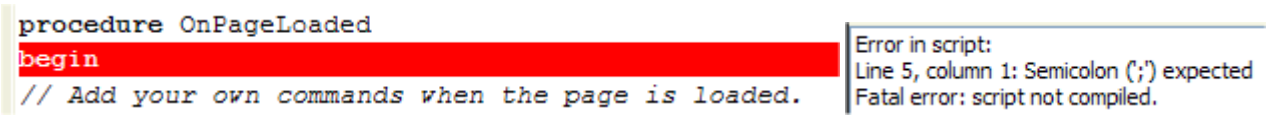


The script's code appears in the main edit box.

13.2.1 Editor Features

- **Syntax Highlighting**: The editor provides automatic syntax highlighting. Pascal keywords are bolded, and comments are italicized.
- **Code Auto-Completion**: If you do not remember the name of a built-in function, press **Ctrl+Space** to display a list of suggestions.
- **Parameter Hints**: When you type the name of a built-in function followed by an opening parenthesis `(`, the editor displays the function's parameters. If the hint does not appear, you can press **Shift+Click** on the keyword.

- **Syntax Checking**: To check your script for errors, click **Check** in the toolbar. The **Check** command pre-compiles the script to verify its syntax, allowing you to easily find errors.
- If the script is error-free, you will get the message: "**Script successfully compiled**".
- Otherwise, you will receive an error message that specifies the reason and location of the error. The line containing the error is highlighted in red:



  Fix the problem (in this example, by adding a semicolon `;` after `OnPageLoaded`), then press **Check** again. The red highlight will disappear.

- **Editing Commands**: You can access various editing commands, such as **Undo**, **Redo**, **Cut**, **Copy**, **Paste**, **Find**, and **Replace**, from the toolbar or the right-click context menu.

13.2.2 Saving and Discarding Changes

When you are finished with your modifications, save the script by clicking **Save**. Before saving, the script is pre-compiled (as if you clicked **Check**). **Only** if no syntax errors are found will the editor close and the script be saved.

If an error is found, the editor will not close. To close the editor without saving your changes, click **Discard**.

Only **error-free scripts** are accepted. For this reason, scripts are always pre-compiled when you click **Save** or when they are imported. This is a security measure; since scripts are compiled and linked during the final application build, it is best that they are already free of syntax errors.

13.2.3 Help and References

The **Help** button provides access to these help topics, and the **References** button opens the script function catalog.

> **Warning**
>
> A script that is free of syntax errors is not guaranteed to be free of logical errors. You must always test your applications to ensure your scripts function correctly.

Introduction to Scripting

Using the Script Manager

Script Function Reference

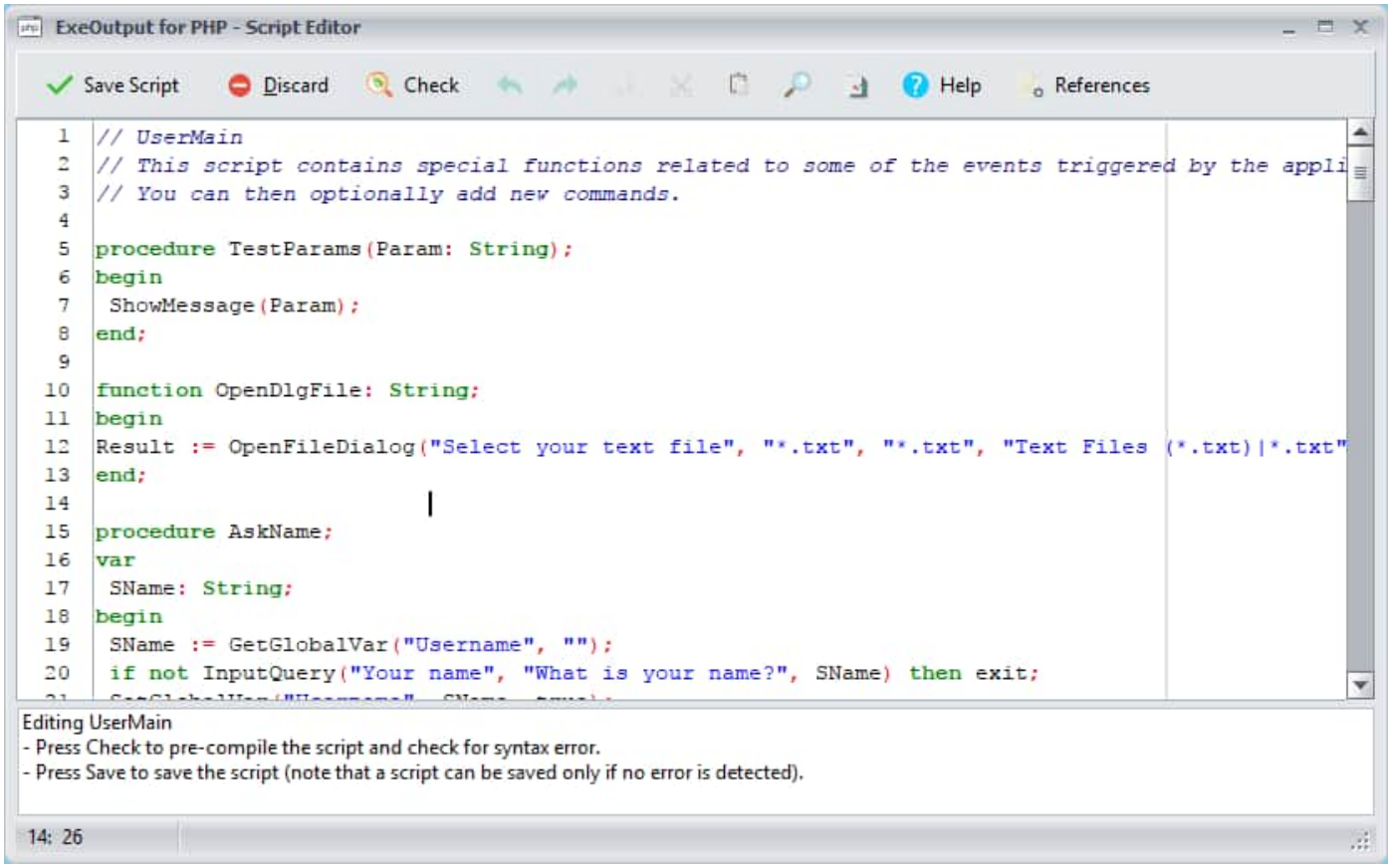
13.3 Adding HEScript Code to Your Application

HEScript allows you to extend the functionality of your PHP application, complementing PHP and JavaScript.

This topic shows you how to **quickly add an HEScript function or procedure to your application**. For instance, you may have found some code on our forum that you want to use.

13.3.1 Steps to Add Code

1. In ExeOutput for PHP, go to **Application Settings** => **Scripting**.
2. Double-click `UserMain` in the list. The script editor will open:



See the full description of the script editor and its features

3. Paste the entire function you want to use into the editor. For instance, the following HEScript code returns the path to the user's "My Documents" directory:

```
function GetDocPath: String;
begin
  Result := GetSpecialFolderPath(16);
end;
```

4. Click **Save Script**. Your script is ready!

13.3.2 Executing the Script

JavaScript and PHP can be used to invoke HEScript.

To execute the code shown previously, you would use the following in PHP:

```
<?php
$path = exo_return_hescriptcom('UserMain.GetDocPath', '');
?>
```

Click the link above for more examples and syntax.

👉 Introduction to Scripting

👉 Using the Script Manager

👉 Script Function Reference

13.4 Running and Calling HEScript Procedures/Functions

HEScript can be used in multiple ways:

- PHP and HTML pages
- HTML links
- JavaScript functions
- Global application events

This topic explains how to call HEScript functions and procedures.

13.4.1 Using HTML Links

You can call HEScript procedures and functions from HTML **links**. Use the `hescript://` protocol prefix (instead of `http://`) to tell the application to execute an HEScript procedure or function.

The syntax is `hescript:[ScriptName].[ProcedureOrFunctionName]`.

Examples: `hescript://MyScript.Procedure1` or `hescript://UserMain.OnNavigateComplete`

In HTML code:

```
<a href="hescript:MyScript.Procedure1">Click here to execute my first function</a>
```

For instance, suppose we wrote a small script named **demo1** in the Script Manager that contains the following procedure:

```
procedure MyFirstDemo;
begin
  MessageBox("This is a simple message.", "First Demo", MB_OK + MB_ICONINFORMATION);
end;
```

The following HTML code would then call it:

```
<a href="hescript:demo1.MyFirstDemo">Click here to execute this function</a>
```

If your procedure or function uses **string parameters**, you can pass them using a `|` (`%7C`) separator. Note: only string parameters are supported with this method.

> **Warning**
>
> Recent versions of Chromium require the `|` character to be encoded in HTML links: %7C

Syntax: `hescript://[scriptname].[functionprocedurename]%7Cparam1%7Cparam2%7C...%7CparamN`

Example: `hescript://MyScript.ProcedureA%7Ctest`

> **Warning**
>
> Important: Please properly URL-encode any non-ASCII characters in parameters. ExeOutput for PHP will decode them automatically.
>
> URL encoding replaces unsafe ASCII characters with a `%` followed by two hexadecimal digits. For example, spaces in filenames should be replaced by `%20`. **To convert characters online**, you can use a tool like the one at w3schools.com.

Demonstration with Parameters

Here is a second example. The script procedure in "demo1" is:

```
procedure MySecondDemo(cond: String);
var
  S: String;
begin
  S := "first";
  if cond = "2" then S := "second";
  MessageBox("The " + S + " link was clicked", "Second Demo", MB_OK + MB_ICONINFORMATION);
end;
```

It accepts a single string parameter. We can then use two different links, changing only the parameter from "1" to "2":

First link:

```
<a href="hescript:demo1.MySecondDemo%7C1">This is the 1st link</a>
```

Second link:

```
<a href="hescript://demo1.MySecondDemo|2">This is the 2nd link</a>
```

> **Info**
>
> String parameters should only be used in links and JavaScript calls.

13.4.2 Using JavaScript

You can use HEScript in conjunction with **JavaScript**.

👉 Please refer to the exeoutput object API for calling HEScript from JavaScript.

13.4.3 Using PHP

ExeOutput for PHP provides two built-in PHP functions for calling HEScript functions:

`exo_runhescriptcom(string $script)`

- `$script` is the reference to the script's name and the function to call.

`string exo_return_hescriptcom(string $script, string $defaultvalue)`

👉 Please refer to the ExeOutput for PHP Internal Functions topic for samples.

13.5 HEScript Function Reference

HEScript is specially designed to control application behavior and communicate with the Windows environment.

There are many **internal functions** that can be called from your HEScript procedures and functions.

Notes:

- Many internal functions are not listed here, as they may be used, for instance, in system HTML pages. Please ignore them or contact us for more information.
- You can find additional Object Pascal references at delphibasics.co.uk.

13.5.1 Interface Functions

Function Name	Description & Prototype	Parameters, Example, Remarks
MessageBox	Displays a message box (similar to the Windows API `MessageBox`). **function** MessageBox(**const** Text, Title: **String**; **const** Flags: Integer): Integer;	• **Text**: The contents of the message box. • **Title**: The title of the message box. • **Flags**: A set of bit flags that determines the contents and behavior of the dialog box. More information here. Can be `MB_OK`, `MB_OKCANCEL`, `MB_YESNO`, etc., combined with `MB_ICONSTOP`, `MB_ICONINFORMATION`, etc. • **Result (Integer)**: Can be `IDOK`, `IDCANCEL`, `IDYES`, `IDNO`, etc., depending on the button the user selects.
MessageDlg	Displays a themeable message box. **function** MessageDlg(**const** Message, Title: **String**; DialogType: TMsgDlgType; Buttons: TMsgDlgButtons): Integer;	• **DialogType**: Can be `mtWarning`, `mtError`, `mtInformation`, `mtConfirmation`, or `mtCustom`. • **Buttons**: A set of flags defining the buttons, e.g., `[mbYes, mbNo]` or `[mbOK]`. **Note:** This function should not be called during the initialization event or when the application exits.
SetUIProp	Sets the value of a specified property for a control. **procedure** SetUIProp(**const** id, propname, propval: **String**);	• **id**: Name of the control. • **propname**: Name of the property. • **propval**: The new string value to assign.
GetUIProp	Gets the value of a specified property for a control. **function** GetUIProp(**const** id, propname: **String**): **String**;	• **result**: The string value of the property. An error may occur if the control is not found.
ChangeStatusBar	Sets the text of the status bar. **procedure** ChangeStatusBar(**const** Text: **String**; Reset: Boolean);	• **Reset**: If `True`, this text will be set as the default.
ShowAboutBox	Shows the About box. **procedure** ShowAboutBox;	
ManageWaitForm	Shows or hides a “Please wait…” dialog box. **procedure** ManageWaitForm(Show: Boolean; Text: **String**);	• **Show**: `True` to show the dialog box; `False` to hide it. After calling `ManageWaitForm(True, "...")`, you must call `ManageWaitForm(False, "...")` to hide it.

13.5.2 Navigation Functions

GoToPage	Displays or executes the specified URL. **procedure** GoToPage(**const** Name, Window: **String**);	• **Name**: The partial or full URL to display. • **Window**: For internal use only. Leave empty (`""`).
ExitPublication	Terminates the application. **procedure** ExitPublication;	
NavigateCommand	Executes a standard navigation command. **procedure** NavigateCommand(**const** Command: Integer);	**Command** can be: 0 (Back), 1 (Forward), 3 (Copy), 4 (Select All), 5 (Paste), 6 (Zoom In), 7 (Zoom Out), 9 (Cut), 10 (View Source), 11 (Delete), 12 (Dev Tools), 13 (Update Preferences).
ExecuteLiveHEScript	Compiles and executes HEScript code at runtime. **function** ExecuteLiveHEScript(const Code: String): String;	The **Code** must be a complete block, including `begin` and `end;` .
ExecutePHPScript	Executes PHP code and optionally returns the output. **function** ExecutePHPScript(const PHPCode: String): String;	The **PHPCode** must include the `<?php` and `?>` tags.
ExecuteHTMLScript	Executes a JavaScript function in the current HTML page. **procedure** ExecuteHTMLScript(const CommandLine, scriptURL: String);	**CommandLine** is the script function to call, including arguments.

13.5.3 Management Functions

SetGlobalVar	Sets the value of a global variable. **procedure** SetGlobalVar(const Name, Value: **String**; const IsStored: Boolean);	**IsStored**: If `True` , the variable is persistent (saved and restored on next run).
GetGlobalVar	Gets the value of a global variable. **function** GetGlobalVar(const Name, DefaultIfNotFound: **String**): **String**;	Returns **DefaultIfNotFound** if the variable doesn't exist.
GetString	Returns the value of a resource string. **function** GetString(const ID: **String**): **String**;	Returns the value of the resource string, or empty if not found.
GetManualHardwareID	Returns a unique system ID based on hardware specs. **function** GetManualHardwareID(method: integer): **String**;	**method**: 0 (HDD Serial), 1 (Media ID), 2 (CPU ID), 3 (HDD Manufacturer ID).

13.5.4 Program, File, and Folder Functions

OpenFile	Opens an external document or program file. **function** OpenFile(const Filename, Parameters: String; State: Integer): Integer;	**Result** is successful if greater than 31.
UnpackTemporaryResource	Extracts a compiled file to a temporary file and returns the path. The file is removed when the application closes. **function** UnpackTemporaryResource(const SourceName: **String**): **String**;	Useful with `OpenFile` for files that can't be handled internally (videos, etc.).
RunAProgram	Executes the specified program file. **function** RunAProgram(const Filename, Params, WorkingDir: String; Wait: Boolean; DispWindow: Integer): Boolean;	**Wait**: If `True`, the application waits for the program to finish.
OpenFileDialog	Displays the standard "File Open" dialog box. **function** OpenFileDialog(**const** aTitle, aFilename, aDefaultExt, aFilter, aInitialDir: **String**): **String**;	Returns the full path to the selected file, or an empty string if canceled. **Warning:** The main window must be available.
SaveFileDialog	Displays the standard "File Save As" dialog box. **function** SaveFileDialog(**const** aTitle, aFilename, aDefaultExt, aFilter, aInitialDir: **String**): **String**;	Returns the full path to the selected file, or an empty string if canceled. **Warning:** The main window must be available.
SelectDirectory	Displays a "Browse For Folder" dialog box. **function** SelectDirectory(**const** Caption: **String**; **const** Root: **String**): **String**;	Returns the full path to the selected folder, or an empty string if canceled.

13.5.5 Miscellaneous Functions

MD5OfAString	Computes the MD5 hash of a string (converts to UTF-8 first). **function** MD5OfAString(**const** Str: String): **String**;	
InputBox	Prompts the user for input with a dialog box. **function** InputBox(**const** Query, Title, Default: **String**): **String**;	Returns the user's input, or an empty string if they chose “Cancel”.
StartTimer	Starts a timer to trigger an event after a specified interval. **procedure** StartTimer(**const** TimerName: **String**; **const** Interval: Cardinal);	**Interval** is in milliseconds (e.g., 1000 for one second).
StopTimer	Stops a timer started by `StartTimer`. **procedure** StopTimer(**const** TimerName: **String**);	

13.5.6 Delphi Class Support

Some Delphi classes, such as **TStringList** and **TMemoryStream**, are also supported.

To use these classes, you must add the `Classes` unit to the `uses` clause at the top of your script:

```
uses Classes;
```

To use the `TRegistry` object, add the `Registry` unit:

```
uses Registry;

procedure Test1;
var
  reg: TRegistry;
begin
  reg := TRegistry.Create;
  reg.RootKey := HKEY_LOCAL_MACHINE;
  reg.DeleteKey("Software\HEViewer\Coucou");
  reg.CloseKey();
  reg.Free;
end;
```

Other Windows dialog boxes, such as the Color Picker dialog, are also available through the script engine if you add the `Dialogs` unit:

```
uses Dialogs;

procedure StartIt;
var
  CD: TColorDialog;
  S: String;
begin
  CD := TColorDialog.Create(nil);
  if CD.Execute(0) then
  begin
    S := inttostr(CD.Color);
    ShowMessage(S);
  end;
  CD.Free;
end;
```

👉 Introduction to Scripting

👉 Using the Script Manager

13.6 Script Templates

Some HEScript scripts such as UserMain contain **pre-defined events**, i.e. functions which are called by the application at appropriate times.

This topic describes these events.

13.6.1 About the UserMain script

The UserMain script is automatically created when a new project is started. It contains numerous global events that allow you to insert special commands.

Event name	Description
function OnBeforeNavigate (NewURL, TargetFrame: String): Boolean;	Triggered just before the application displays an HTML page. Set `Result` to `True` to halt the operation. `NewURL` is the full URL to be displayed, and `TargetFrame` is the frame where the page is displayed (if applicable).
function OnRouterURL(RequestedURL, RequestedFilename, PathInfo, QueryString: String): String;	Triggered by the router engine, allowing you to reroute any URL. Further description and samples.
procedure OnPHPErrorMessage(ErrorMsg: String);	Triggered when the PHP runtime returns an error. `ErrorMsg` contains the error message.
procedure OnNavigateComplete (URL: String);	Triggered when a page has finished loading. `URL` is the full URL of the loaded page.
procedure OnDownloadComplete(FullPath: String; OriginalURL: String; MimeType: String; TotalSize: Integer);	Triggered when a file download has completed. `FullPath` is the complete path to the downloaded file, `OriginalURL` is the original URL of the download, `MimeType` is the reported MIME type of the file, and `TotalSize` is the total size of the downloaded file in bytes.
function OnPubLoaded: Boolean;	Triggered when the application starts, before the homepage is displayed. Set `Result` to `True` to exit immediately without warning.
procedure OnPubBeingClosed;	Triggered when the application is about to terminate.
procedure OnDisplayWindow (WindowName: String);	Triggered when a window (whose name is specified by `WindowName`) is displayed. This applies to both main and secondary windows, such as pop-ups.
procedure OnStartMainWindow;	Triggered just before the main window is displayed (i.e., before the homepage is shown).
procedure OnCloseWindow (WindowName: String);	Triggered when a window is closed by the user.
function OnWindowCloseQuery (WindowName: String): Boolean;	Triggered when a user attempts to close a window. Set `Result` to `False` to prevent the window from closing, or `True` to allow it. This event is not created by default; you must manually add it to your project's `UserMain` script to utilize it.
function OnTimer (TimerName: String): Boolean;	This event is triggered by a timer created using the `StartTimer` HEScript function. It occurs when the specified duration, set by `StartTimer`, has elapsed. Set the function's `Result` to `True` to disable the timer, or use the `StopTimer` function.
procedure OnPrintPage;	Triggered when the user prints the current page.
function OnInvalidPasswordAtStartup: Boolean;	Triggered when an invalid global password is provided at startup. Note: Set `Result` to `True` to prevent the application from displaying an error message and exiting.

Event name	Description
function OnExpiredPublication: Boolean;	Triggered when the global expiration date is reached. Note: Set `Result` to `True` to prevent the application from displaying an error message and exiting.
procedure OnTrayIconClick;	Triggered when the user clicks the tray icon.
procedure OnTrayIconDblClick;	Triggered when the user double-clicks the tray icon.

Introduction to Scripting

Using the Script Manager

Script Function Reference

13.7 How to Prompt a User for Their Name Once and Store It

The goal of this script is to:

- Prompt a user to enter their name the first time they run the application.
- Store the name in a persistent global variable.
- Display the name on an HTML page.

13.7.1 Steps

We will name our global variable `TheUserName`.

Step 1: Write the Script

Go to the Script Manager, double-click on `UserMain`, and paste this script into the `OnPubLoaded` function event:

```
function OnPubLoaded: Boolean;
var
  S: String;
begin
  // Check if the user was already prompted.
  // If the `TheUserName` global variable already has a value, do not prompt the user again.
  if GetGlobalVar("TheUserName", "") <> "" then exit;

  // Prompt the user.
  S := InputBox("Welcome!"#13#10"Please enter your name:", "What is your name?", "");

  // If the user does not provide a name, set it to "Mysterious User".
  if S = "" then
    S := "Mysterious User"
  else
  begin
    // Store the result only if the user has provided a name.
    // This way, they will be prompted again next time if they leave it blank.
    SetGlobalVar("TheUserName", S, True);
    // `True` means the global variable is persistent.
  end;

  // This event occurs when the application is starting, before the homepage is displayed.
  // Set `Result` to `True` to exit immediately without any warning.
  Result := False;
end;
```

Step 2: Display the Name in an HTML Page

Use this JavaScript code:

```
document.write(window.external.GetGlobalVariable('TheUserName', ''));
```

13.7.2 Notes

You can, of course, customize this example. Instead of using a dialog box, you could display an HTML page at startup.

👉 Introduction to Scripting

👉 Using the Script Manager

👉 Script Function Reference

13.8 How to Run an Executable Program

This guide illustrates how to run an external program or a program compiled into your application. For instance, you may want to launch a program that you ship with your application.

13.8.1 Running an External Program File

Use the internal `RunAProgram` HEScript function to execute a program. You only need the path to the program file you want to launch. Fortunately, ExeOutput for PHP provides a `HEPublicationPath` global variable that points to the folder containing the application's EXE file.

Script Example:

```
procedure RunTutor;
var
  EbookPath, MyProgram: String;
begin
  EbookPath := GetGlobalVar("HEPublicationPath", "");
  MyProgram := EbookPath + "hehvdemo.exe";
  RunAProgram(MyProgram, "", EbookPath, false, SW_SHOWNORMAL);
end;
```

First, we get the path to our application and store it in the `EbookPath` variable. Then, we append the filename of our executable and pass the resulting path to the `RunAProgram` function.

13.8.2 Running a Program Compiled into the Application

You can include program files directly into your application's EXE. In that case, you need to extract the program file before running it. You can use the `heopenit` special protocol or the following code:

```
procedure RunACompiledProgram;
var
  MyProgram: String;
begin
  MyProgram := UnpackTemporaryResource("programfilename.exe");
  RunAProgram(MyProgram, "", ExtractFilePath(MyProgram), false, SW_SHOWNORMAL);
end;
```

This code uses the `UnpackTemporaryResource` internal function to extract a compiled file to a temporary location. Then, you can execute the file.

Introduction to Scripting

Using the Script Manager

Script Function Reference

13.9 How to Call DLL Functions

The script engine can **import procedures and functions from external DLL files**, allowing you to extend your application by calling your own DLL functions.

13.9.1 How to Import a DLL Function in HEScript

This is done with the **external** keyword.

Syntax:

```
function functionName(arguments): resultType;
[callingConvention]; external "libName.dll" [name 'ExternalFunctionName'];
```

For example, the following declaration:

```
function MyFunction(arg: integer): integer; external 'CustomLib.dll';
```

This imports a function called `MyFunction` from `CustomLib.dll`. The default calling convention is `register` if not otherwise specified.

HEScript allows you to declare a different calling convention (`stdcall`, `register`, `pascal`, `cdecl`, or `safecall`) and to use a different name for the DLL function, as in the following declaration:

```
function MsgBox(hWnd: Pointer; lpText, lpCaption: String;
uType: Cardinal): Integer; stdcall; external "user32.dll" name 'MessageBoxW';
```

This imports the `MessageBoxW` function from `user32.dll` (a Windows API library) and renames it to `MsgBox` for use in the script:

```
procedure TestDLL;
var
  S: String;
begin
  S := InputBox("What do you want to show?", "Query", "");
  MsgBox(0, S, "You entered:", MB_OK + MB_ICONINFORMATION);
end;
```

> **Warning**
>
> Pointer and `out` parameters are not handled by the script engine.

13.9.2 Other Examples

The following Windows APIs can be imported into the HEScript engine with these declarations:

```
function GetDiskFreeSpace(root: AnsiString; var secPerCluster, bytesPerCluster, freeClusters, totalClusters: integer): boolean; stdcall; external
"Kernel32.dll" name 'GetDiskFreeSpaceA';

function FindWindow(className, windowName: AnsiString): integer; stdcall; external "User32.dll" name 'FindWindowA';

function GetKeyState(virtKey: integer): smallint; stdcall; external "User32.dll";
```

Introduction to Scripting

Using the Script Manager

Script Function Reference

14. PHP Samples

14.1 PHP Samples

Several PHP samples with source code are available to demonstrate additional features of ExeOutput for PHP.

👉 Please visit the PHP App Samples page on our website to download PHP samples with their full source code, which you can compile yourself with ExeOutput for PHP.

> **Tip**
>
> 👉 See also our General Demonstration for more **demonstrations and PHP samples**.

15. Other Topics

15.1 Environment Options

This dialog box allows you to configure **global options** for customizing ExeOutput for PHP, **default settings** for new projects, and various other properties. You can access the **Environment Options** dialog box by selecting the "Environment Options" menu command from the Application menu: 

15.1.1 General Options

- **Create a file backup when an HTML page is modified**: If you edit a PHP or HTML file using the internal HTML editor, ExeOutput for PHP can back up the original file before saving changes. The backup filename will be the same, but with a `~` character in the extension (e.g., `.htm` becomes `.~htm`). This is recommended.
- **Directly add files to application using the relative path**: When new files are added (manually or using the Source File List Update operation), ExeOutput for PHP can prompt you to specify the application virtual path for these files, or automatically find and use the best virtual path. This option disables the prompt and lets ExeOutput for PHP automatically determine the virtual paths. However, this is **not recommended**, as it is better to monitor how files are accessed in the application at runtime.
- **Archive cache**: This option enhances the archive caching feature by comparing file date-times during compilation. ExeOutput for PHP only recompresses files if they have been modified or if the source file lists have been updated. If this option is enabled, it stores the file date-times during the initial compression. For subsequent builds, it then compares the current file date-times with the stored ones. If any pair differs, all source files will be recompressed. This option is highly recommended.
- **Do not show Windows notifications after build**: ExeOutput for PHP displays a system notification when a build finishes. Enable this option to suppress that behavior.
- **Clear CEF and PHP runtimes cache**: To speed up compilation, Chromium Embedded Framework (CEF) and PHP Runtime files are compressed once and stored in ExeOutput for PHP's cache for reuse. If source files are modified, you must clear the cache.
- **Custom Temporary Folder**: Allows you to specify a custom location for temporary files created during compilation (for compression and caching). Using a fast drive, such as a RAM disk, for this folder can improve performance.
- **Manage Update Checks**: Allows you to start the Web Update utility or configure automatic update checks.

15.1.2 Appearance

The **Appearance** page allows you to select the desired theme for the ExeOutput for PHP interface, ensuring the application's look and feel align with your preferences.

Available Themes

- **Light Silver (Default)**: By default, the **Light Silver** theme is selected, offering a clean and modern appearance.
- **Windows Default**: The **Windows Default** theme ensures that all controls adopt the native styles of the operating system without additional customization. This theme is particularly beneficial if you intend to leverage Windows' accessibility features, such as high-contrast themes. When a user activates a high-contrast theme in their operating system, ExeOutput will automatically switch to the **Windows Default** theme to maintain compatibility and ensure optimal accessibility.

> **Note**
>
> ExeOutput for PHP intelligently detects if a user has enabled a high-contrast theme within their Windows settings. In such cases, it automatically selects the **Windows Default** theme to enhance accessibility and provide a consistent user experience. This automatic adjustment ensures that your application remains accessible to all users, including those who rely on assistive technologies.

15.1.3 Default Settings

These settings are used each time you start a new project.

- **Author Name**: Your full name.
- **Company Name**: Enter your company name.
- **Web Homepage**: Enter the URL of your website.
- **Default Copyright**: Enter the default copyright information for your packages.

15.1.4 Default Files

These files are used each time you start a new project.

Note: The default language file must exist. If it is not found, ExeOutput for PHP will attempt to use the one in its program folder. If that also does not exist, an error will occur.

15.1.5 Source File List Update

Additional options for the Source File List Update operation.

- **Automatically remove non-existing files**: When enabled, ExeOutput for PHP automatically removes missing files from source file lists (a file is considered missing if it is not found on the hard disk at its expected location). Otherwise, you will be prompted. Note that this will also remove the properties associated with the file, but not any potential references.
- **Monitor the source folder for changes**: If enabled, ExeOutput for PHP will use Windows Shell extensions to monitor for changes to files in the source folder and its subdirectories. If changes are detected, the entire file lists will be rebuilt (and the File Manager refreshed). If disabled, file lists are updated only when a Source File List Update operation occurs.
- **Perform a file list update when a project is loaded**: The source folder will be scanned for changes, and file lists will be refreshed as soon as the project is loaded.
- **Perform a file list update when a project is compiled**: The source folder will be scanned for changes, and file lists will be refreshed just before the application compilation begins.

15.1.6 Exclude Files

For the Source File List Update operation.

You can exclude files based on their file extension, such as executables, projects, backups, and other file types. Just add the desired extension(s) to the list. Wildcards are supported.

Notes:

1. You should exclude all files associated with ExeOutput for PHP: projects (`.exop`), languages (`.exol`), backups (`.bak` and `.~*`), skins (`.skn`), etc.
2. To **exclude a single file** from compilation, you can set this in the File Properties dialog box.

15.1.7 Code Signing

This page contains options related to code signing; please refer to the dedicated code signing topic.

The code signing utility used by ExeOutput for PHP requires an internet connection to timestamp the application's signature. It will use the two defined URLs to contact the timestamp servers.

Two timestamp servers are used: an Authenticode-compatible server and an RFC-3161-compatible server. You can configure their URLs or use the default ones provided by ExeOutput for PHP by clicking the buttons next to the fields.

15.2 Technical Notes Regarding Applications

15.2.1 System Requirements and Limitations

☑ **Minimum Configuration:** Note that this list may change in the future.

- Microsoft Windows® 11, 10, 8, or 7 (32-bit or 64-bit Intel; no WinRT or ARM editions).
- 256 MB RAM or greater (recommended: 512 MB to 2 GB).
- Visual C++ Redistributable for Visual Studio 2015-2019

> **Info**
>
> If you want your app to run on Windows XP or Vista, download and install ExeOutput for PHP 2 to create Windows XP and Vista compatible PHP apps.

☑ **Size Limitations**

ExeOutput for PHP supports creating application .exe files **up to 4 GB**. EXE files larger than 4 GB are not recognized by Windows.

It is always better to split very large applications into smaller ones. If your application contains hundreds of files, it will take longer to load at startup. Please consider the Keep Files External feature.

15.2.2 Internal Compression Engine

ExeOutput for PHP 2025 features a new internal compression engine using **Zstandard (zstd)**, replacing the previous Brotli library. This change brings several benefits:

- **Faster Compilation**: zstd offers significantly faster, multi-threaded file compression, resulting in quicker project build times.
- **Improved Application Performance**: The redesigned compression system leads to more responsive applications.

15.2.3 Notes on the Internal Browser, PHP Version, and Runtime Module

- ExeOutput for PHP is a true **Unicode-enabled application**. Compiled applications also work with Unicode. However, the PHP runtime does not natively handle Unicode.
- ASP, PHP, JSP, and ASPX pages designed to run on a server may not work properly in applications. They will be treated as normal webpages.
- ExeOutput for PHP builds 32-bit applications. However, they should work fine on 64-bit operating systems.
- If the application is not portable, it stores its settings on the user's computer in a unique application folder. On Windows Vista/7&8/8.1/10/11, this is typically `C:\Users\[User name]\AppData\Roaming\ExeOutput for PHP\Userapplication\[application GUID]`. On Windows XP/NT, it is `C:\Documents and Settings\[User name]\Application Data\ExeOutput for PHP\Userapplication\[application GUID]`.

  This folder can be changed as explained here. If you do not want your application to leave files behind, you must add a reference to this folder for the uninstaller, so it can remove the contents.

15.3 Cloning a Project

If you want to compile a website using the settings of an existing project, you can use ExeOutput for PHP's **Clone Project** feature.

Normally, once a project is started, you cannot change its source folder. The **Clone Project** feature allows you to define a **new source folder** by selecting a new index page.

☑ You can access this feature by selecting the menu command **Project** | **Clone Project** from the Application menu: 

15.3.1 Steps to Clone a Project

1. Open the project file you would like to clone.
2. Use the **File** | **Clone Project** menu command. A window with instructions will appear; click **Continue**.
3. You will be prompted to select the **new index page**. This works exactly as if you were starting a new project: select the HTML file that will be used as the index page of your new cloned application. The folder containing this page will also become the **new source folder**.
4. The next dialog will ask you to enter the filename for the **new project**.
5. ExeOutput for PHP then starts a new project, uses the settings of the previously loaded project, and scans the new source folder. Once complete, the project file is saved and then reloaded to clear memory.

15.4 Command Line Options

This page is designed for advanced users and describes how to use ExeOutput for PHP to create and compile applications (even silently) with **command line switches** (useful for daily and automated builds, for instance).

15.4.1 What is a Switch?

All switches are specified with a forward slash (`/`) and may be followed by a value. Example: `/C`

If you are specifying files or folders with spaces, you must enclose them in quotation marks. Example: `/U "My value" /C`

15.4.2 Command Line Options to Manage Project Files

You can specify project files for ExeOutput for PHP via the command line. When you launch ExeOutput for PHP manually (e.g., using the "Run" command from the Windows Start menu), you can pass parameters to the program.

The following command line opens a project file: `EXO4PHP.EXE "C:\mywork\myproject\myproject.exop"`.

ExeOutput for PHP will open and read the project settings. You then only need to press the **Compile** button to create the application.

15.4.3 Available Switches

- `/c` : Forces ExeOutput for PHP to compile the EXE whose project file was specified.
- `/q` : Forces ExeOutput for PHP to exit after a successful compilation.
- `/s` : Forces ExeOutput for PHP to run silently (no progress bar).

You can combine these switches. For example, `EXO4PHP.EXE "C:\mywork\myproject\myproject.exop" /c/q/s` will force ExeOutput for PHP to compile the EXE silently and then exit.

15.5 Contact Information

☑ For the latest version of ExeOutput for PHP and add-ons, visit our **website**:

https://www.exeoutput.com

☑ For questions, ideas, suggestions, or help, visit our **forum**:

https://www.gdgsoft.info

☑ Follow us on X:

https://x.com/gdgsoft

☑ If you have questions, bug reports, or feedback regarding ExeOutput for PHP, you can reach us through our ticket support system or by email:

https://www.exeoutput.com/contact

Please include your full name, a valid email address, and the version of ExeOutput for PHP (and your computer configuration if reporting a bug). Registered users should also provide their **user ID to receive priority technical support**.

👉 About this documentation

15.6 About this documentation

☑ Stay informed about ExeOutput for PHP by visiting our websites:

https://www.exeoutput.com

https://www.gdgsoft.com