APL64 Windows Runtime Executable

Contents

Overview

The APL64 runtime license is non-exclusive, royalty-free and perpetual. For the Windows operating system environment, the APL64 runtime component encapsulates the APL64 components and APL64 programmer-provided workspace(s) and file(s) into a single Window runtime executable (WRE) with the '.exe' file name extension. Each APL64 programmer-created application and every version of an APL64 programmer-created application has a separate (WRE) file.

An APL64 Windows runtime executable (WRE) is a single 'exe' file containing all the components necessary to deploy an APL64-based application system in the Microsoft Windows operating system environment. The WRE includes the APL64 interpreter, the application start-up workspace and any other application-specific workspaces and files required for the deployment and use of the application in a production environment.

The application-specific files may include:

- APL64 xml-format configuration file
- APLNow32 ini-format configuration file
- APLNow32 adf-format configuration file
- APLNow32 manifest file
- Other application-specific files such as:
    - APL64 component file
    - Native files

Minimum Requirements

The WRE file must contain a runtime-ready APL64 start-up workspace. When the WRE is run, this start-up workspace is loaded into an APL64 instance. The □LX value in this workspace must contain the name of an APL64 programmer-defined function in the workspace that starts the APL programmer-developed application system. As a runtime application, the APL64 programmer is responsible for providing an appropriate graphical user interface (GUI) which contains a 'Wait' for end user input. Without such a 'Wait', the WRE will end execution immediately. Such a GUI can be created using the APL64 □WI tools.

To create a WRE, separate source and target folders must exist on the APL64 programmer's workstation. The runtime-ready APL64 start-up workspace must exist in the target folder. Any additional application-

required files can exist anywhere they are accessible to the workstation creating the WRE. It is suggested that these additional application-required files be placed in the target folder to assure the integrity of any versions of the WRE. The runtime-ready APL64 start-up workspace path is entered in the WRE creation dialog. The target folder path is also entered in the WRE creation dialog.

Each WRE should have separate source and target folders. Using the same source or target folders when creating different WREs can result in a WRE with components which are unnecessary or incorrect. The target folder and its sub-folders should be empty prior to running the WRE creation utility, except for prior versions of the same WRE executable.

Runtime-ready APL64 Start-up Workspace

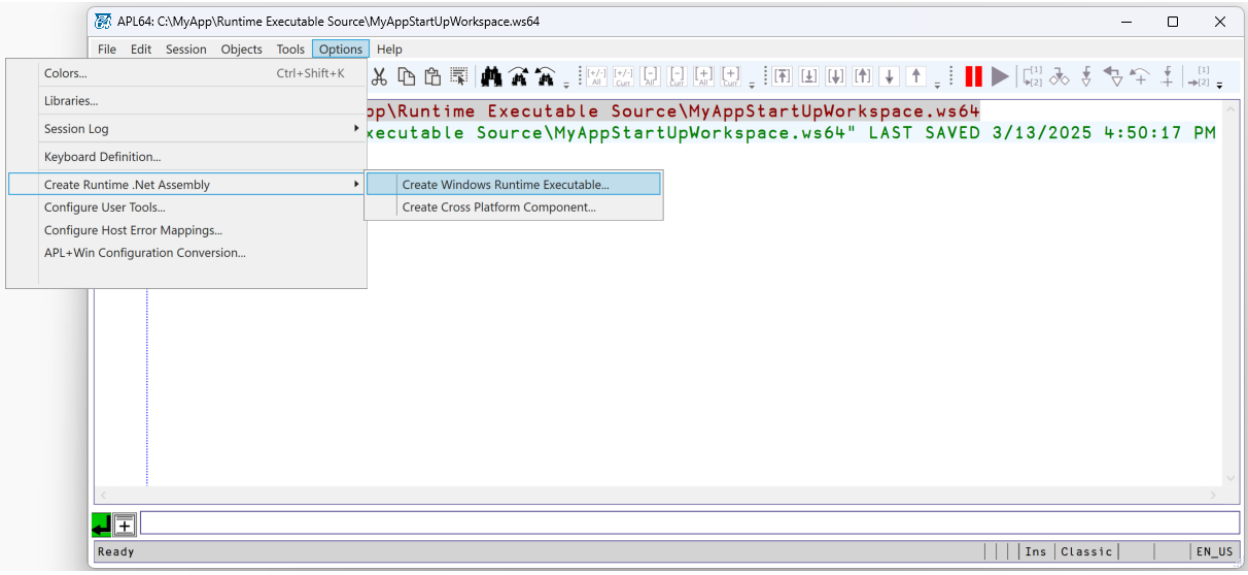
The runtime-ready APL64 start-up workspace is embedded into the WRE when the WRE is created by the APL64 programmer. This workspace does not materialize when the application is run on the application system end user's workstation. The workspace is loaded into the runtime APL64 instance as a memory stream, assuring the integrity and confidentiality of this workspace.

APL64 Capabilities available for a WRE

All features of APL64 which are available in the Windows operating system environment may be used by the APL64 programmer in the development of an APL64 runtime application.

Sample Application #1: WRE Example

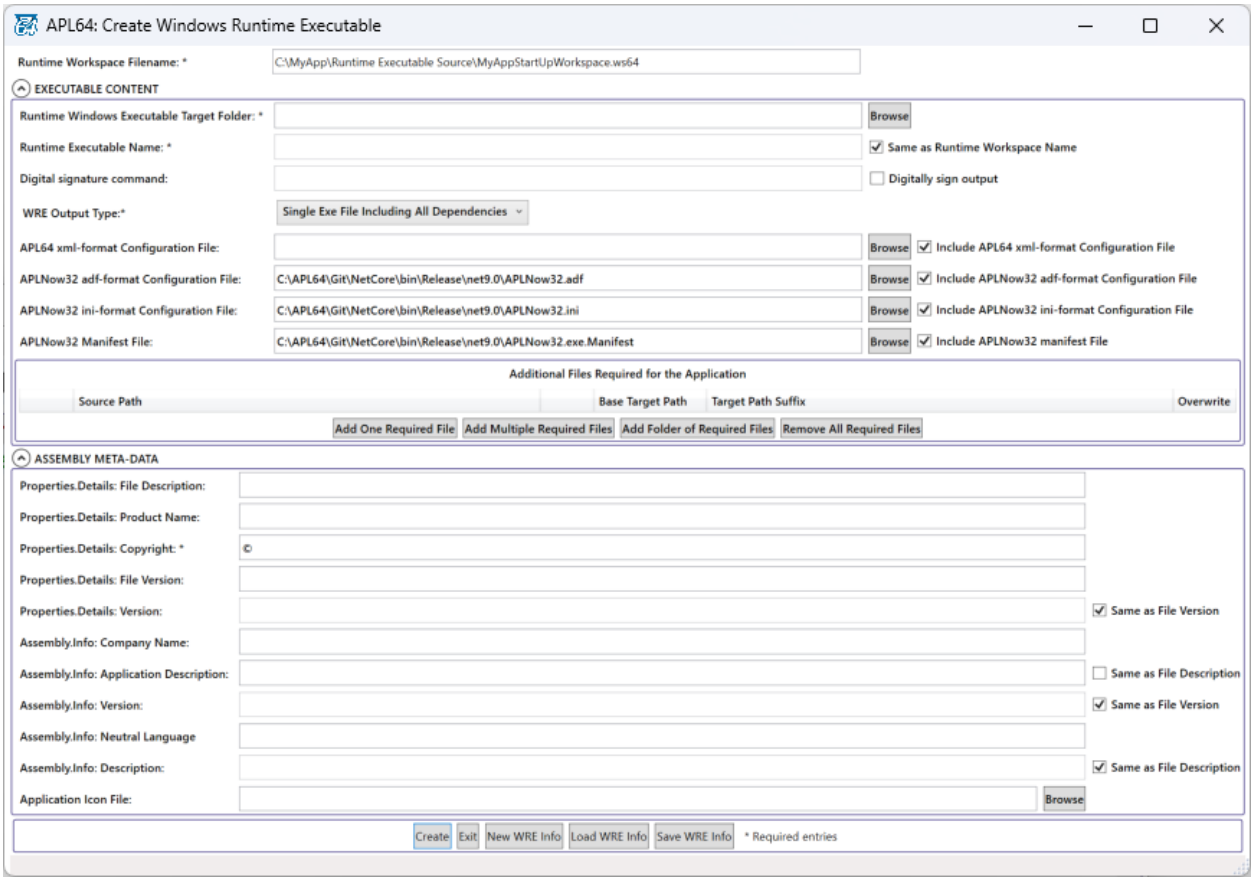
To create a WRE file, xload the application start-up workspace and then use the **Options | Create Runtime .Net Assembly | Create Windows Runtime Executable…** menu item to display the WRE utility dialogue in the APL64 Developer version GUI.



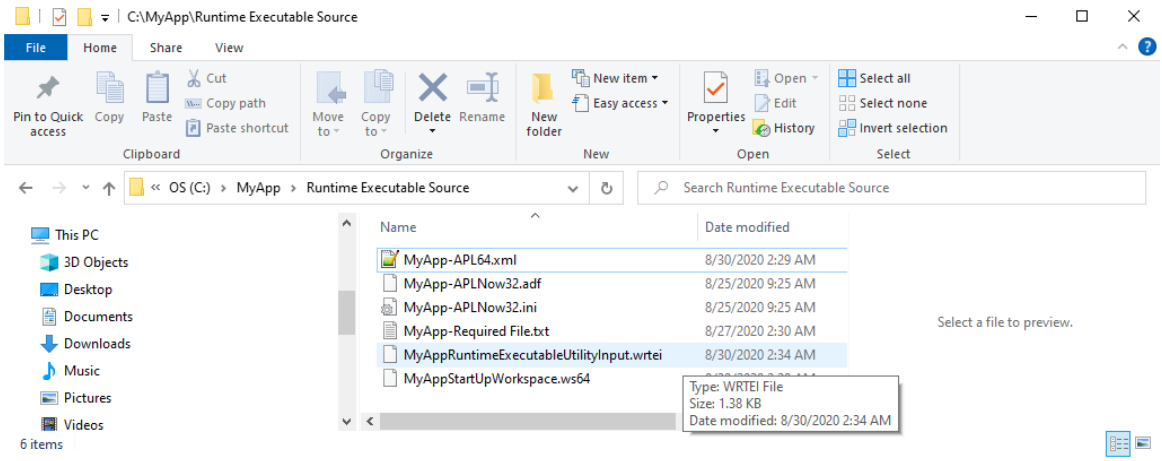
The workflow for creating an APL64 Windows runtime executable is illustrated using an example. The example is available for download at http://apl2000.com/APL64/UserDocumentation/APL64-WRE-Example.zip.

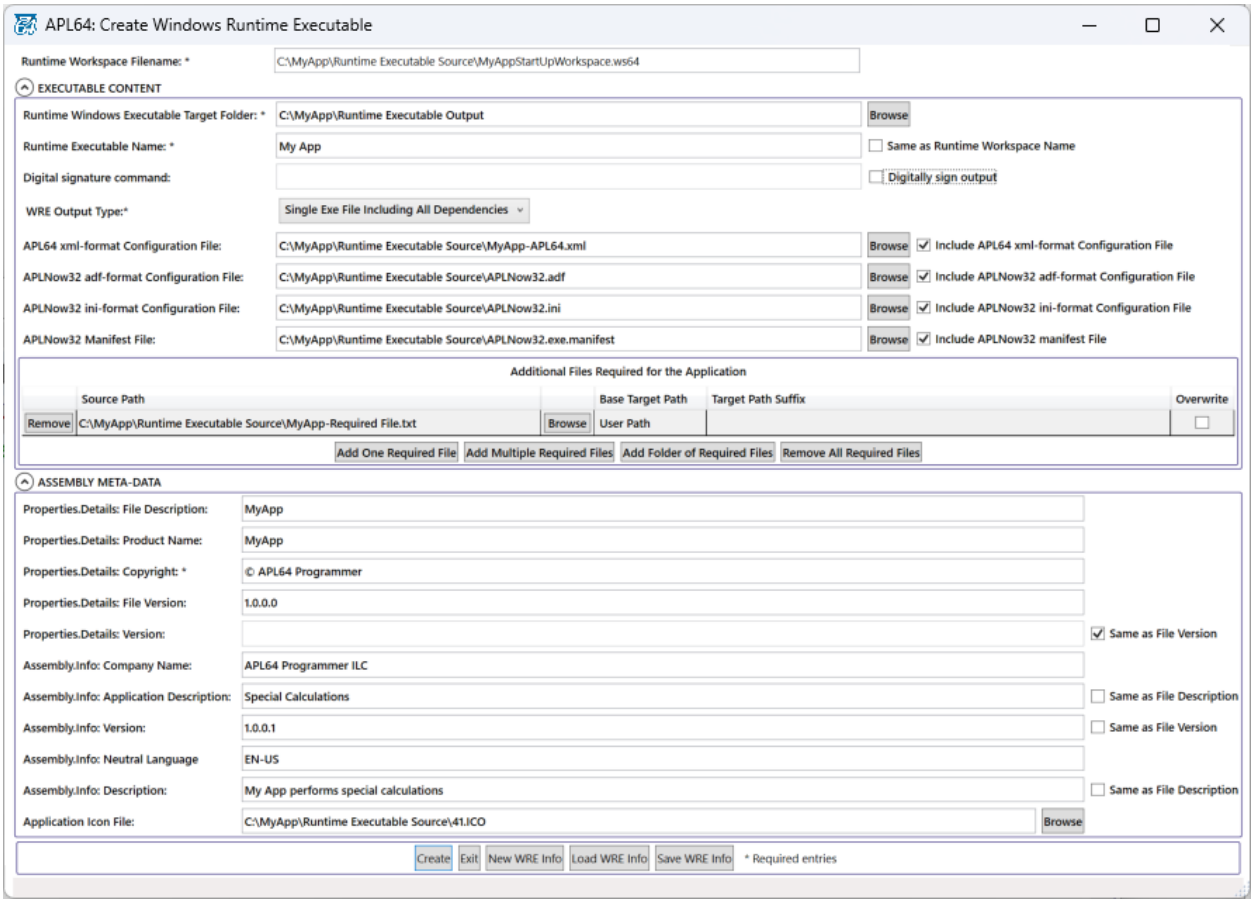
Expand the APL64-WRE-Example.zip file to the c:\ drive of the target workstation to create the c:\MyApp\Runtime Executable Source\ and c:\MyApp\Runtime Executable Output\ folders

- Use the APL64 **Options | Create Runtime .Net Assembly | Create Windows Runtime Executable…** menu item to display the WRE creation dialogue



- Click the ‘Load WRE Info’ button, browse to the C:\MyApp\Runtime Executable Source\ folder and select the example information file, MyAppRuntimeExecutableUtilityInput.wrtei. This action will fill the input fields of the WRE utility dialogue with the entries for the provided example.



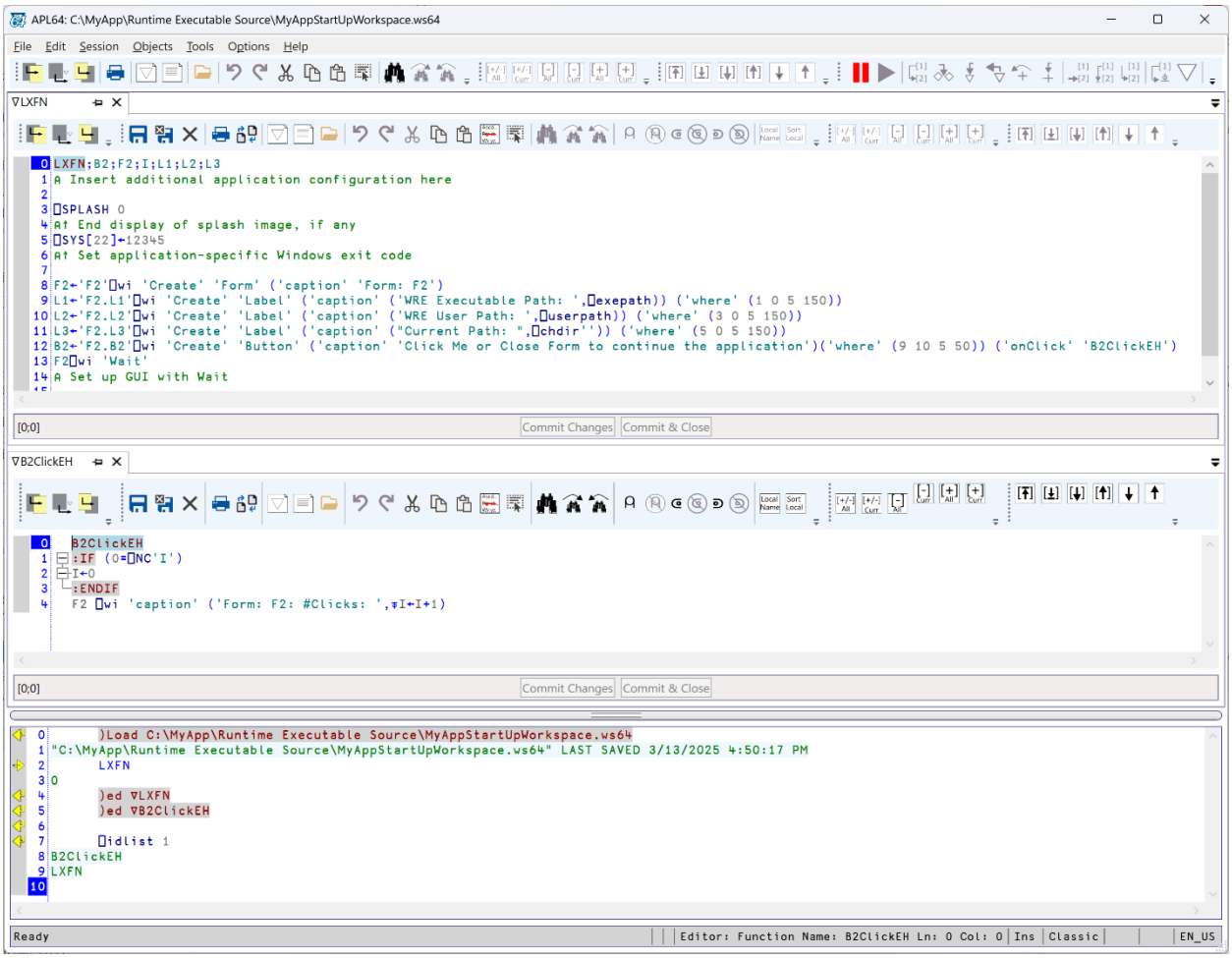


- Saving a Windows Runtime Executable information file for each APL programmer-developed application will make using the WRE utility simple.

Runtime-ready Start-Up APL64 Workspace

In the example provided the runtime-ready workspace contains the APL64 user-defined function LXFN

- Its name is the value of the APL64 ⎕LX system function, so when the runtime-ready start-up workspace is loaded, the LXFN starts which can be set by the APL64 programmer via: ⎕LX←'LXFN'
- The LX function creates the application system GUI with a Wait
- The GUI in the example displays information about the ‘user path’ and the ‘exe path’ when the application runs.



The example also provides a simple application, which is run whenever the APL64 user-defined 'B2ClickEH' event handler function is run.

Input the APL64 Developer Version Utility Information

The APL64 developer version includes a utility to create a WRE. Access this utility via the **Options | Create Runtime .Net Assembly | Create Windows Runtime Executable…** menu:

When using this utility, the tooltips on the input fields provide important information about the required entries. Especially note the tooltips on the *headings* in the 'Additional Files Required for the Application' list.

Which entries are required to create a WRE?

The required APL64 programmer entries for the WRE creation utility window include:

- Runtime Workspace Filename must be provided with the name of an existing 'runtime-ready', APL64 workspace.
- Runtime Windows Executable Target Folder, which must be different than the path of the Runtime Workspace Filename, must be provided with the path of an existing path accessible to the APL64 programmer's workstation. This path is where the WRE utility will create the WRE file and is not necessarily the path from which the resulting WRE will be run.

- Runtime Executable Name, if different than the Runtime Workspace Filename, must be provided. The resulting WRE will have this filename and will be created in the Runtime Windows Executable Target Folder. The .Net SDK, which is used by the WRE creation utility to create the WRE executable, restricts the Runtime Executable Name. For example, a hyphen in the 'Runtime Executable Name' field. An underscore may be used instead of the hyphen.
- A non-empty Copyright text must be provided. This text will be included in the Windows Runtime Executable suffixed by: - Portions © APLNow, LLC & APLNext, LLC. A '/' character is not permitted in the copyright text value.

All other entries on the WRE utility window are optional and will depend on the APL64 programmer's design of the application.

If necessary, for the proper operation of the application, custom configuration files for APL64 and APLNow32 may need to be created by the APL64 programmer and included in the WRE.

APL64 xml-Format Configuration File

There is no restriction on the file name of the APL64 xml-format configuration file. There are three ways to provide a pre-prepared APL64 xml-format configuration file for the WRE:

1. Enter the filename of, or browse to, an APL64 xml-format configuration file on the APL64 programmer's workstation in the 'APL64 xml-format Configuration File' field in the WRE creation utility. Using this method:
    - The file will be embedded in the WRE
    - The file will not be materialized at runtime on the end user's workstation, it cannot be examined by the user of the WRE.
    - Each time the application is run the file will be read as a memory stream into the APL64 instance

    When the WRE utility creates the WRE, the APL64 xml-format configuration file, identified by the programmer-entered file name in the in the 'APL64 xml-format Configuration File' field, will be embedded in the WRE exe-format file. When the WRE exe file is run, this APL64 xml-format file is read from the WRE exe file as a 'memory stream' and used to configure the APL64 instance used by the WRE.
2. Add an APL64 xml-format configuration on the APL64 programmer's workstation in the 'Additional Files Required for the Application' section of the WRE dialogue. If this option is used:
    - The file will be embedded in the WRE
    - When the application is run, the file will be materialized on the end user's workstation with the file name which is the concatenation of the APL64 programmer-specified 'Base Target Path' and 'Target Path Suffix' fields of the WRE creation dialogue.
    - If the 'Overwrite' option is not checked, the file will be materialized on the end user's workstation only if the file does not already exist.
    - The APL64 programmer must provide an 'A64CP' command-line argument to the WRE when it is run on the end user's workstation with value equal to the target location provided by the APL64 programmer for this 'Additional File' in the WRE dialogue.
3. Embed no APL64 xml-format configuration file in the WRE.

- The APL64 programmer must assure, via other means than the WRE executable, e.g. an APL64 programmer-developed installer program, that the appropriate xml-format file, if any, at runtime exists on the end user's workstation
- The APL64 programmer must provide an 'A64CP' command-line argument to the WRE when it is run on the end user's workstation with value equal to the path of this file

In this sample application, there is a custom APL64 xml-format configuration file provided.

APLNow32 Files for Win32-dependent APL64 features

The Win32-dependent features of APL64 depend on the APLNow32.exe executable and, if necessary, adf and ini format configuration files and the manifest file provided by the APL64 programmer. **Note**: In runtime, the APLNow32.exe.manifest file is required when displaying GUI controls like CommandBars and CommandButtons.

When the WRE is created by the APL64 programmer, the APLNow32.exe executable is always embedded in the WRE. When the WRE exe file is run, the APLNow32.exe file is materialized on the target workstation running the application in the ⎕ACBD folder. The same applies to the APLNow32 adf and ini configuration files and manifest file when they're included.

To specify an APLNow32 adf or ini configuration file or manifest file for the WRE, check the 'Include APLNow32 Configuration Components' in the WRE creation dialogue.

The APL64 programmer-provided adf, ini and manifest files must exist on the APL64 programmer's workstation when the WRE created. These files can have any file name, but they are renamed to APLNow32.adf, APLNow32.ini, and APLNow32.exe.manifest when they are embedded in the WRE exe file created by the utility.

If the APL64 programmer specifies an APLNow32.adf file in the 'APLNow32 adf-format Configuration File' field of the WRE creation utility:

- The programmer-provided adf-format file will be embedded in the WRE exe file created by the utility.
- When the WRE exe file is run, the adf-format file is materialized on the target workstation running the application in the ⎕ACBD folder.

There are three ways to provide a pre-prepared APLNow32 ini-format configuration file for the WRE:

1. Enter the filename of or browse to an APLNow32 ini-format configuration file on the APL64 programmer's workstation in the 'APLNow32 ini-format Configuration File field in the WRE creation utility. Using this method:
   - The file will be embedded in the WRE
   - When the WRE exe file is run, the ini-format file is materialized on the target workstation running the application in the ⎕ACBD folder.
2. Add an APLNow32 ini-format configuration on the APL64 programmer's workstation in the 'Additional Files Required for the Application' section of the WRE dialogue. If this option is used:
   - The file will be embedded in the WRE
   - When the application is run, the file will be materialized on the end user's workstation with the file name which is the concatenation of the APL64 programmer-specified 'Base Target Path' and 'Target Path Suffix' fields of the WRE creation dialogue.

- If the ‘Overwrite’ option is not checked, the file will be materialized on the end user’s workstation only if the file does not already exist.
- Depending on the file name, the APL64 programmer must provide a ‘A32CP‘ command line argument to the WRE when it is run on the end user’s workstation with value equal to the target location provided by the APL64 programmer for this ‘Additional File’ in the WRE dialogue.

3. Embed no APLNow32 ini-format configuration file in the WRE.
   - The APL64 programmer must assure, via other means than the WRE executable, e.g. an APL64 programmer-developed installer program, that the appropriate ini-format file, if any, at runtime exists on the end user’s workstation
   - Depending on the file name, the APL64 programmer must provide a ‘A32CP‘ command line argument to the WRE when it is run on the end user’s workstation with value equal to the path of this

In this sample application APLNow32.adf, APLNow32.ini, and APLNow32.exe.manifest files are included.

Additional Files Required for the Application

If the application requires more files in addition to the start-up workspace, configuration files and manifest file, the additional files are added to this list.

- APL64 component files required by the application system
- Additional APL64 workspace from which objects will be copied into the start-up workspace
- Windows native files required by the application system
- Application-specific documentation files
- Splash image file
- Window shortcut for the application, including an appropriate icon.
- Digital signature for the application file
- Activation and licensing wrapper for the application
- User command files (UCMDS.sf, UCMDS2.sf, UCMDS3.sf and UCMDSW.sf) when a user command is invoked in the application system (**Note**: Base Target Path field set to User Path)

If application-specific files, e.g. APL64 component files or native files, are required for the WRE application, the APL64 programmer may:

- Embed these files in the WRE exe file created by the utility using the ‘Additional Files Required for the Application’ section of the utility.
- Use a programmer-provided installer program to separately deploy these files to the target workstation which will run the application.

To add required files to the WRE utility list:

- Click the ‘Add One Required File’ button to add one required file in the list. Browse to the source file and select it. The source file will be listed in the ‘Source Path’ field. Select the ‘Base Target Path’ from the list. Modify the ‘Target Path Suffix’ as desired. Check/UnCheck the ‘Overwrite’ option as desired.
- Clicking the ‘Add Multiple Required Files’ button will present the file selection dialog. Browse to the required files and select them. Modify the ‘Target Path Suffix’ as desired. Check/UnCheck the ‘Overwrite’ option as desired. Each of the selected files will be listed separately.

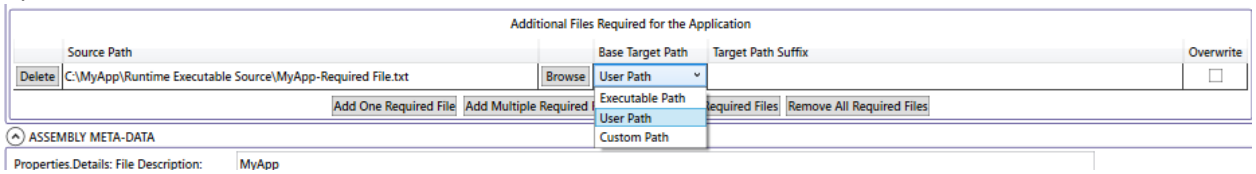
- Clicking the ‘Add Folder of Required Files’ button will present the folder selection dialog. Browse to the required folder and select it. Modify the ‘Target Path Suffix’ as desired. Check/UnCheck the ‘Overwrite’ option as desired. The files will not be listed separately. The ‘Source Path’ field will contain the source folder path.
- To remove files or folders from the list:
  - Click ‘Delete’ button to remove the entry in the selected row of the list. The physical file, if any, associated with the removed list element is not deleted.
  - Click the ‘Remove All Required’ files to clear the list of all files in the list. The physical files, if any, associated with the removed list elements are not deleted.
  - Use Ctrl+Click to select one or more cells in the list and then click the ‘Delete’ keyboard key to delete the selected entries in the list. The physical files, if any, associated with the removed list elements are not deleted.

Each row of the ‘Additional Files Required for the Application’ list specifies:

- An APL64 programmer-provided file or folder to be embedded in the WRE exe file
- How the embedded file will be materialized on the application end-user’s workstation when the WRE exe file is run. The file materialization path on the end-user’s workstation is the combination of the ‘Base Target Path’ and the ‘Target Path Suffix’.

The APL64 programmer must appropriately select the ‘Source Path’, ‘Base Target Path’, ‘Target Path Suffix’ entries and ‘Overwrite’ option for each row of the list:

- The ‘Source Path’ field should contain the full path of the existing file or folder on the APL64 programmer’s workstation to be included in the WRE. The ‘Browse’ button facilitates completing the ‘Source Path’ field. The ‘Source Path’ does not have to match the desired file name of the end user’s workstation.
- The ‘Base Target Path’ determines the initial portion of the path where the files in the WRE exe file are materialized on the end-user’s workstation. The ‘Base Target Path’ is selected from a list of options:



  - The ‘User Path’ option indicates a target workstation path that provides guaranteed read/write access the end-user of the target workstation running the WRE application. When the WRE application is running, ☐USERPATH provides this path. The end user’s workstation credential and the ‘Assembly.Info: Company Name’ and ‘Assembly.Info: Version’ fields of the WRE creation dialogue determine this path. This ‘User Path’ can be selected for application-required files which are end-user editable.
  - The ‘Executable Path’ option indicates the path of the WRE exe-format file installed on the end-user’s workstation. When the WRE application is running, ☐EXEPATH provides this path. The ‘Executable Path’ can be selected for application-required files which are not end-user editable.

    - The ‘Custom Path’ option indicates a fixed APL64 programmer-specified target workstation path.
- The ‘Target Path Suffix’ is the second part of the file materialization path on the end-user’s workstation:
    - If the ‘Base Target Path’ was selected as ‘User Path’, the ‘Target Path Suffix’ is optional.
        - The file embedded in the WRE exe will be materialized at runtime to the end-user’s workstation at the path which is the combination of the □USERPATH and the ‘Target Path Suffix’.
        - If no ‘Target Path Suffix’ is provided, the embedded file will be materialized with the same file name as the ‘Source Path’ file name.
        - If the name of the embedded file should be materialized to a different name, provide the desired materialized file name in the ‘Target Path Suffix’.
        - The ‘Target Path Suffix’ can be a path including folder(s) which will be materialized as sub-folders of the □USERPATH folder.
    - If the ‘Base Target Path’ was selected as ‘User Path’, the ‘Target Path Suffix’ is optional.
        - The file embedded in the WRE exe will be materialized at runtime to the end-user’s workstation at the path which is the combination of the □EXEPATH and the ‘Target Path Suffix’.
        - If no ‘Target Path Suffix’ is provided, the embedded file will be materialized with the same file name as the ‘Source Path’ file name.
        - If the name of the embedded file should be materialized to a different name, provide the desired materialized file name in the ‘Target Path Suffix’.
        - The ‘Target Path Suffix’ can be a path including folder(s) which will be materialized as sub-folders of the □EXEPATH folder.
    - If the ‘Base Target Path’ was selected as ‘Custom Path’, the ‘Target Path Suffix’ must be provided by the APL64 programmer as the full file name to be materialized on the end-user’s workstation, e.g. ‘d:\dir1\...\filename.ext’.
- When the ‘Overwrite’ option is checked, at WRE application runtime, the file embedded in the WRE will be materialized on the target workstation and overwrite a previously-existing version of that file. If the ‘Overwrite’ option is not checked, the file embedded in the WRE is materialized to the target workstation only if there is no prior version of the file on the target workstation.
- When a folder is select as required for the application, the folder content is captured in a zip-format file in the application exe-format file. When the application exe-format file is run by the end user, the folder content is unzipped to the target path specified by the APL64 programmer when the WRE was created.

Assembly Meta-data

The ‘Properties.Details’ entries affect the information exposed by Microsoft Windows when the properties of the WRE executable are accessed using Windows Explorer. The ‘Assembly.Info’ entries affect the information exposed when the WRE executable is examined by .Net methods.

The ‘Application Icon Path’, if provided by the APL64 programmer, will apply to the APL64 runtime executable created by this utility. It will not apply to the application-specific GUI created by the APL64 programmer which will be presented when the APL64 runtime executable runs. For example, if the

application-specific GUI is created using the APLNow32 ⎕WI system function, the APL64 programmer would need to add the applicable application icon to the forms of this GUI.

Digitally sign the executable output

It has become quite common to demand that executable applications have a digital signature.

This optional section allows users to opt-in to have the executable output digitally signed.

To opt-in, check the box labelled “Digitally sign output”. If you do, the following field will become enabled:

Digital signature command: The process to digitally sign the executable file requires a third-party signing tool with a command line interface that signs without the need for human intervention. Enter the complete signing command including the flag or parameter for specifying the file to be signed, but not the file itself. The file that will be signed is already specified in the Runtime Executable filed in the WRE dialog.

The following is the example when code signing with signtool.exe:

> Syntax: signtool.exe sign /f <certificate> /p <certificate password> /fd <file digest algorithm> /t <timestamper URL>
>
> Digital signature command: *signtool.exe sign /f certificate.pfx /p mypassword /fd sha256 /t http://timestamp.digicert.com*

The following is the example when code signing with smctl.exe (Signing Manager Controller):

- With key-pair alias

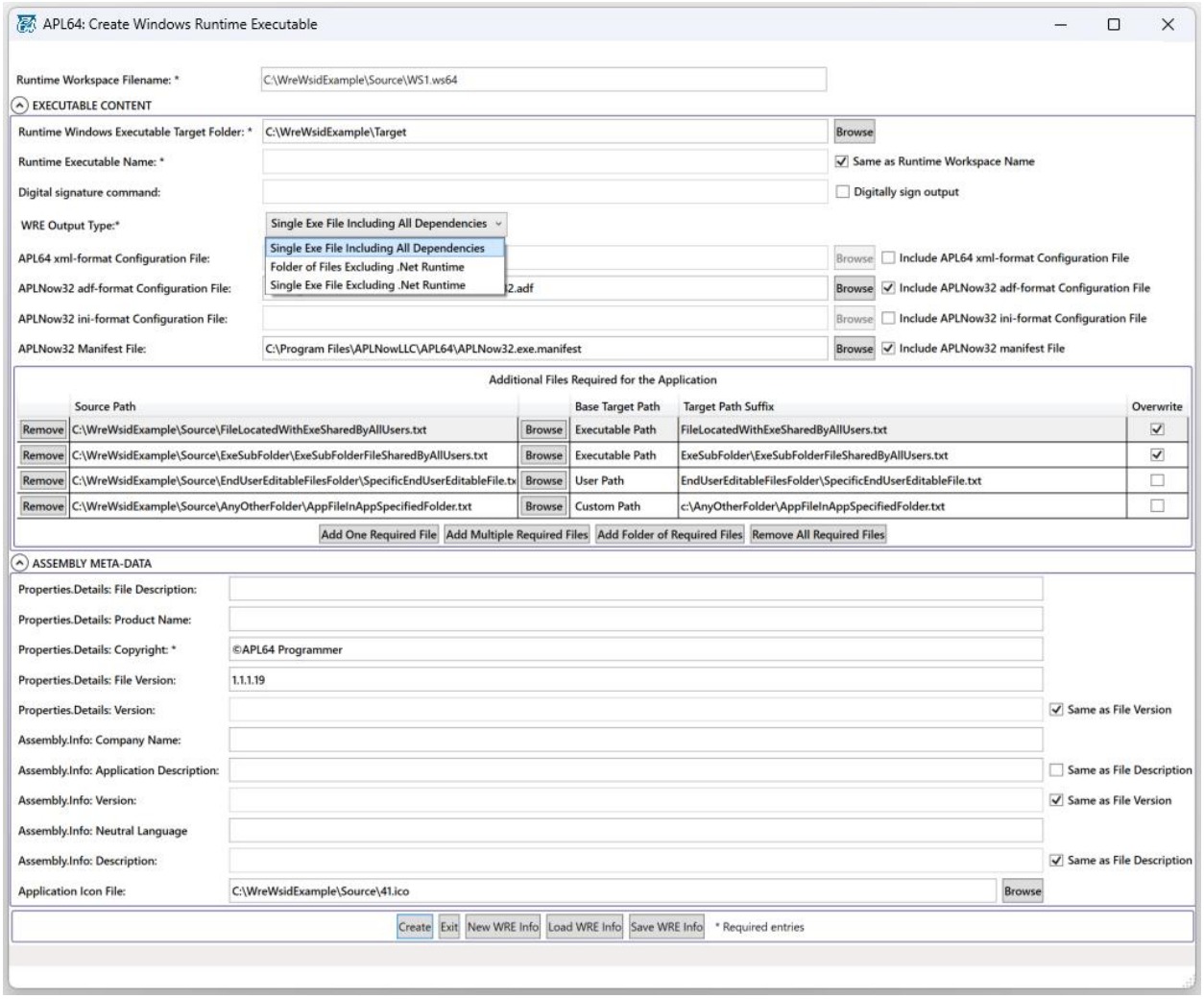
  > Syntax: smctl sign --keypair-alias <keypair alias> -input
  >
  > Digital signature command: *smctl sign --keypair-alias key_123456789 --input*

- With certificate fingerprint

  > Syntax: smctl sign --fingerprint <certificate fingerprint> --input
  >
  > Digital signature command: *smctl sign --fingerprint 01234abcde --input*

WRE Output Type

The WRE Output Type determines the format and content of the result of the WRE creation process. The result of the WRE creation process is placed in the WRE ‘target’ folder which is specified in the WRE creation utility.



Single Exe File Excluding All Dependencies

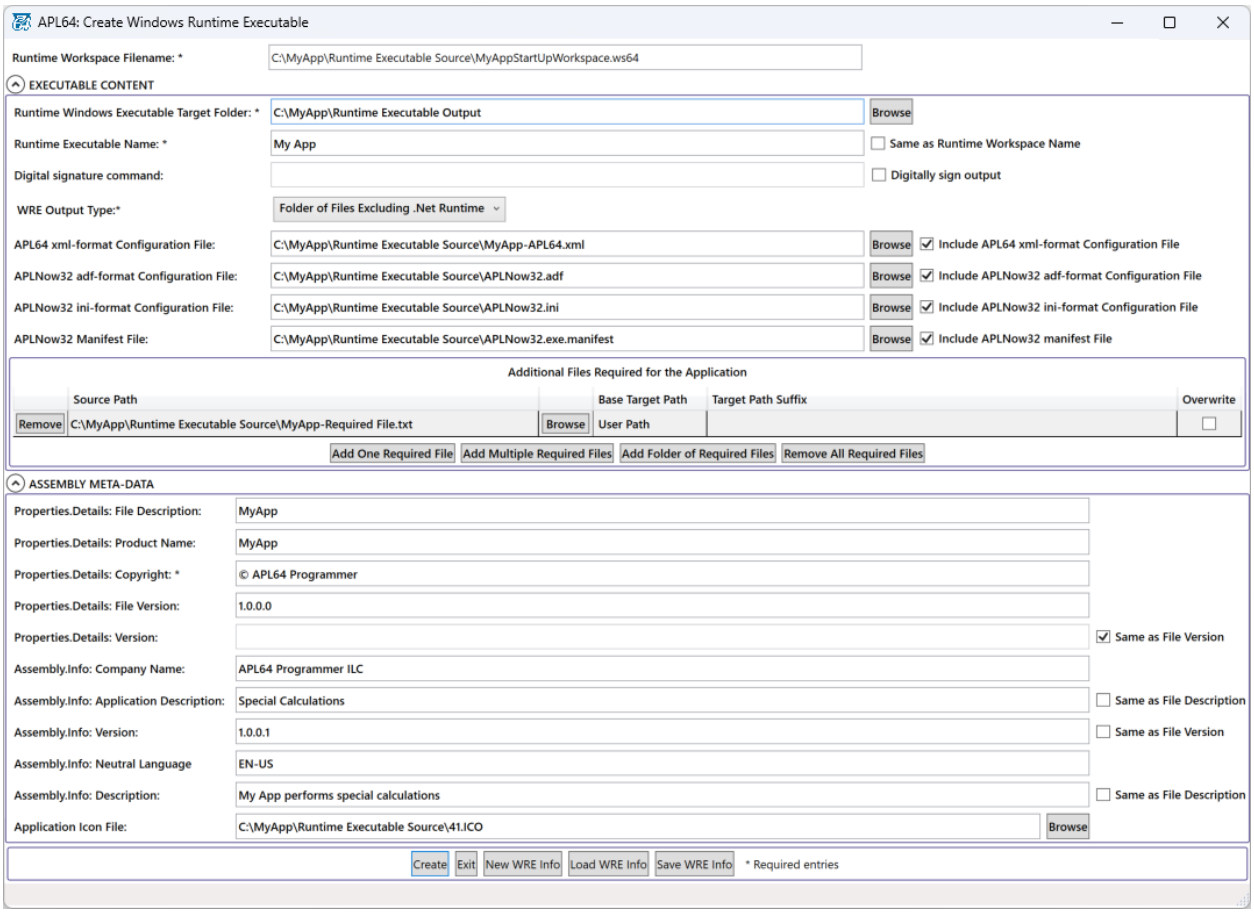
The output is a single, ready-to-run exe-format file containing the APL64 programmer-selected application specific files, the APL64 runtime files, and the .Net Runtime files. The output file can be copied to the target Windows workstation, and double left mouse clicked to run the application system. No additional end user software installation is required.

Folder of Files Excluding .Net Runtime

The output is a folder named Publish containing the exe-format WRE and the APL64 runtime files. The files in this folder must be deployed together to the target Windows workstation. This WRE output format requires that the appropriate .Net runtime is installed on the target Windows workstation. Installing that .Net runtime is the responsibility of the end user of the application system. When the WRE application is run, missing .Net Runtime libraries will be detected, and a prompt to download and install them will be presented.

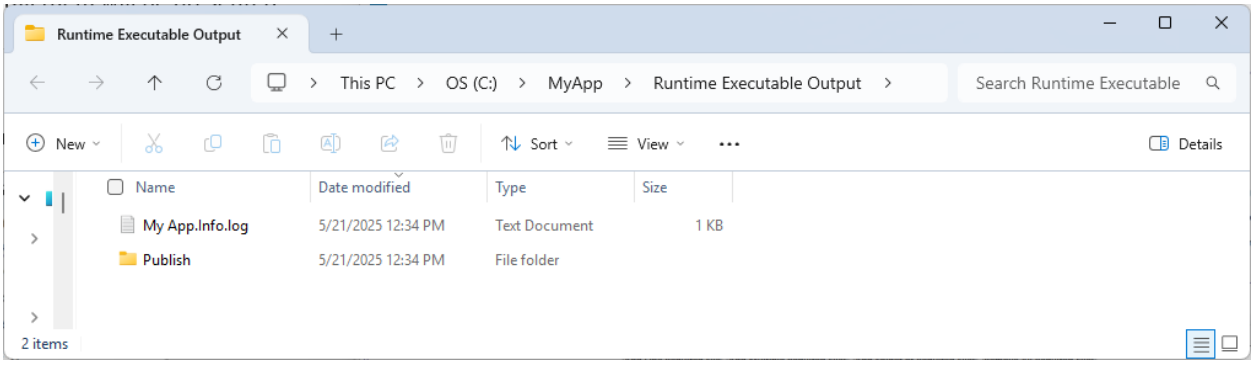
This option will reduce the resulting WRE application by approximately 150 MB.

Example: Using the earlier WRE example #1 with the WRE Output Type option set to Folder of Files Without .Net Runtime:

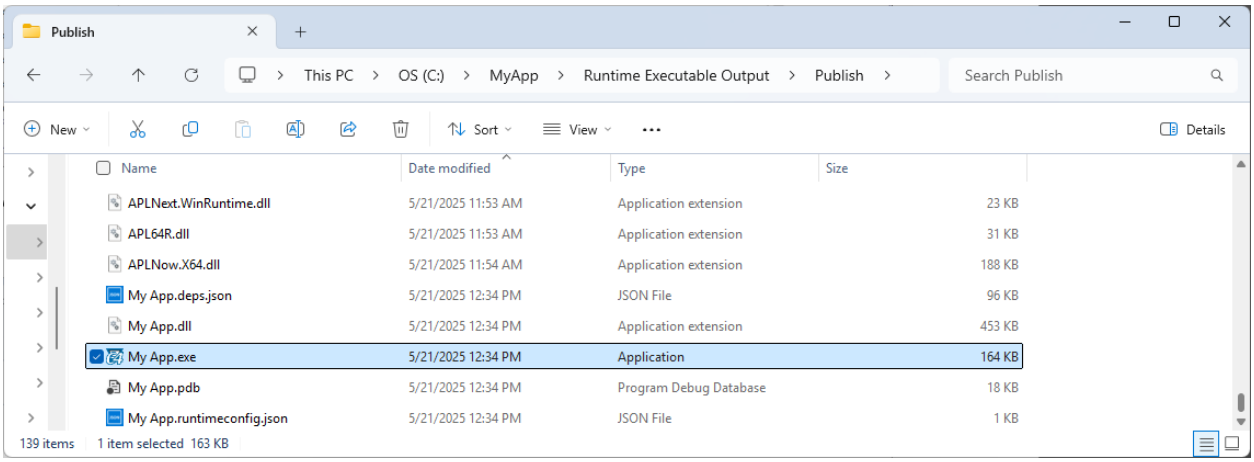


Result when the WRE example was created successfully:

Folder: C:\MyApp\Runtime Executable Output



Folder: C:\MyApp\Runtime Executable Output\Publish



Single Exe File Excluding .Net Runtime

The output is a single, exe-format file containing the APL64 programmer-selected application specific files and the APL64 runtime files, without the .Net Runtime files. This WRE output format requires that the appropriate .Net runtime is installed on the target Windows workstation. Installing that .Net runtime is the responsibility of the end user of the application system. When the WRE application is run, missing .Net Runtime libraries will be detected, and a prompt to download and install them will be presented.

This option will reduce the resulting WRE application by approximately 150 MB.

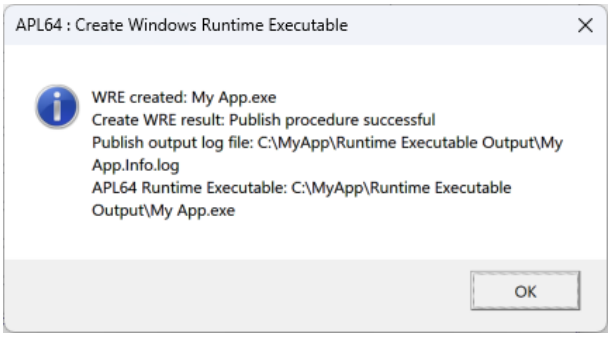
Create the WRE using the WRE Utility

When the applicable entries to this dialogue have been made, use the 'Save Current Windows Executable Information' to save these entries for future use at the location specified by the APL64 programmer.

To create the WRE, click the 'OK' button on the WRE creation dialogue. The WRE creation method uses the .Net Core 'Publish' method to prepare the WRE executable in the specified folder on the APL64 programmer's workstation.

If the WRE is successfully created, a message box will indicate:

- Success of the process
- Publish log file name in the WRE output folder
- WRE executable name in the WRE output folder

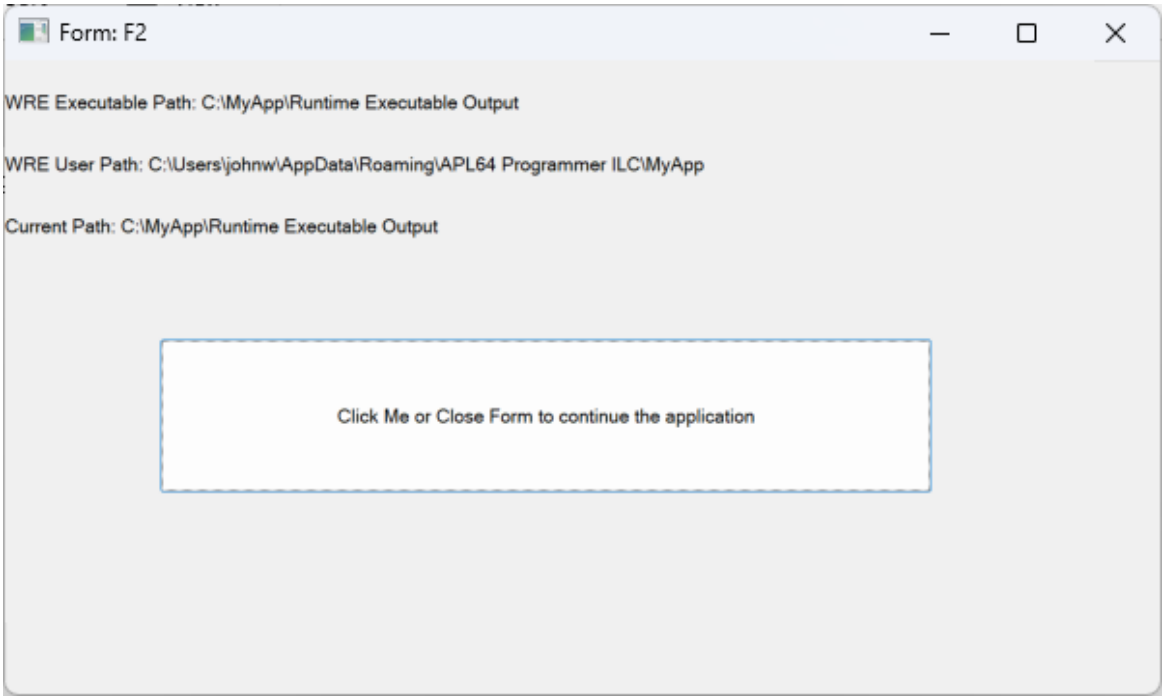


The WRE output folder will contain the '…Info.log' file. This file may contain warnings even if the WRE is successfully created. If the WRE is not successfully created, the message box will indicate this outcome and the '…Errors.log' file will be in the WRE output folder.

Run the Resulting WRE

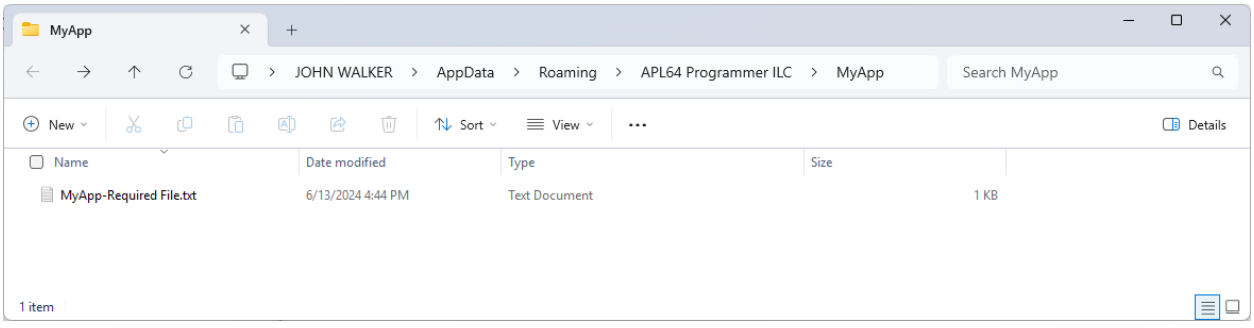
To run the WRE executable on the end user's workstation a Windows short-cut may be create by the APL64 programmer. Alternatively, use Windows Explorer to navigate to the location of the WRE executable on the end user's workstation and double [left] click the WRE exe-format file. The APL64 programmer may prepare a Windows installer program containing the WRE which may aid the end user deployment of the application.

When the WRE is run using the example information, the APL64 programmer-developed GUI will be presented to the end user.

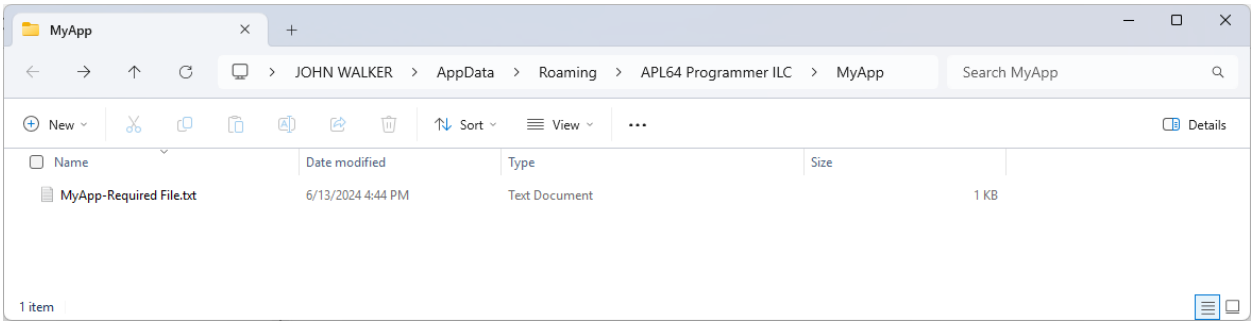


In this example:

- The Current Path is the location of the WRE executable, 'My App.exe'
- The WRE Executable Path is where the WRE executable, 'My App.exe', was installed on the end-user's workstation.



- The WRE User Path is the APL64 programmer-specified target path for the ‘MyApp-Required File.txt’ ‘Addition File’ included in the WRE had materialized.



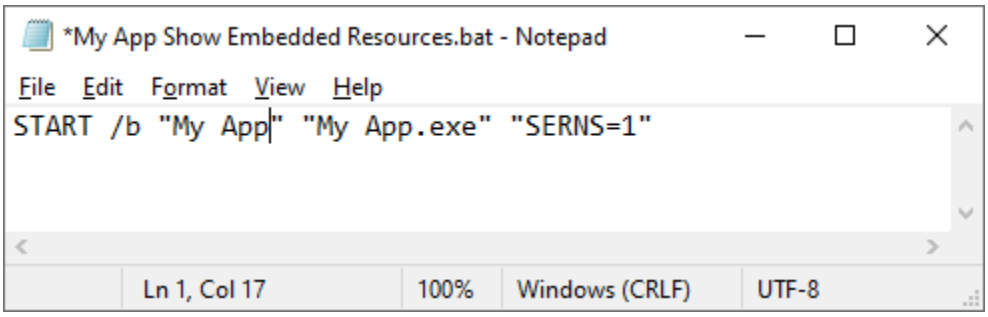
To deploy the WRE to a different workstation, copy it to that target workstation. The WRE may be run from a Windows batch file or a Windows short-cut.

APL64 command line arguments are applicable to the WRE executable.

Checking Embedded Resources

To check the inclusion of the embedded resources in the APL64 runtime executable, create a Windows batch file which starts the runtime executable with the “SERNS=1” command line argument and run the batch file.

```
Start /b "My App" "My App.exe"  "SERNS=1"
```



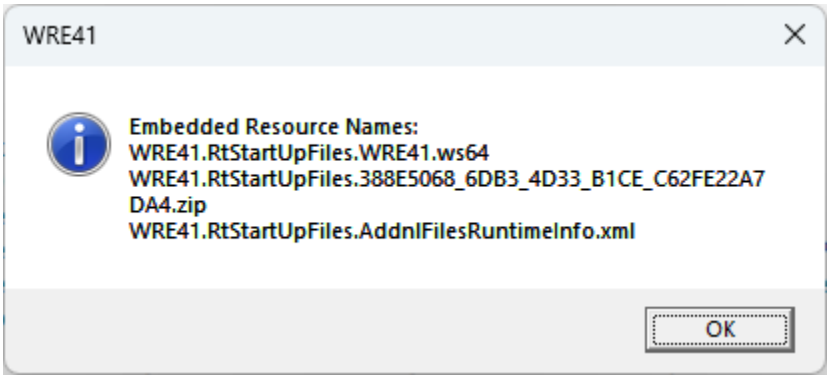
When the batch file is run, a message box will be presented listing the embedded resource name prior to any application-specific GUI. The WRE creation utility may modify the names of some of the embedded resources to accommodate the requirements of the .Net Core Publish process.

The APL64 ws64-format workspace and the APL64 xml-format configuration are embedded in the WRE and are not materialized to the end user's workstation.

The 'My App.RtStartUpFiles.MayApp_APL64.xml' resource name is associated with an embedded xml-format file providing materialization information for the 'Additional Files Required for the Application' which were embedded in the WRE exe file when it was created by the APL64 programmer.

The APLNow32.adf, APLNow32.ini and MyApp-Required File.txt files were embedded in the WRE and are materialized to the end user's workstation.

When a required folder has been included in a WRE, the folder content prior to materialization is contained in the WRE exe-format file as a zip-format file. Each required folder included in the WRE has a unique name. The unique name is recorded in the 'AdditionalFilesRuntimeInfo.xml' file also included in the WRE. This xml-format file contains an entry associating the unique name of the required folder with the APL64 programmer-specified target path for that folder when it is materialized on the target workstation.



Sample Application #2: Additional Files Required for the Application

The 'WreWsidExample' sample WRE application illustrates more detailed usage of the 'Additional Files Required for the Application' section of the WRE creation utility. Follow the workflow to create and run the sample APL64 WRE 'WS1' sample application:

Download the 'WreWsidExample' zip-format file

http://apl2000.com/APL64/UserDocumentation/APL64-WreWsidExample.zip

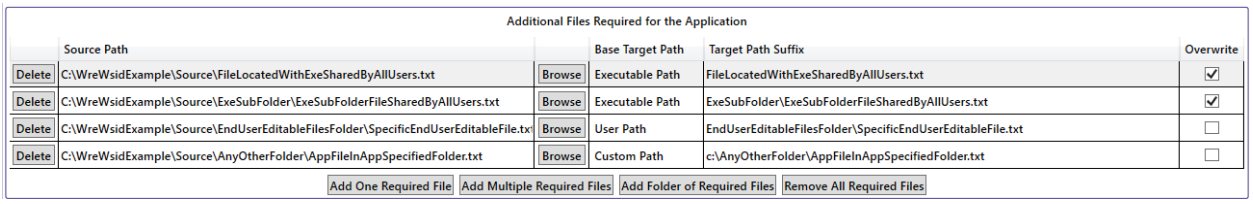
Unzip the downloaded file to the 'c:\' drive

- The C:\WreWsidExample\Source folder contains:
  - The WS1.ws64 workspace containing the application □LX function (LXFN)
  - The WRE utility information file (WS1.wrtei)
  - The 'AnyOtherFolder' containing the 'AppFileInAppSpecifiedFolder.txt' file which will be materialized to the end-user's workstation using the 'Custom Path' option of the WRE creation utility
  - The 'EndUserEditableFilesFolder' containing the 'SpecificEndUserEditableFile.txt' file which will be materialized to the end-user's workstation using the 'User Path' option of the WRE creation utility
  - The 'ExeSubFolder' folder containing the 'ExeSubFolderFileSharedByAllUsers.txt' file which will be materialized to the end-user's workstation using the 'Exe Path' option of the WRE creation utility.
  - The 'FileLocatedWithExeSharedByAllUsers.txt' which will be materialized to the end-user's workstation using the 'Exe Path' option of the WRE creation utility.
- The C:\WreWsidExample\Target folder will contain the WRE exe-format file created by the WRE utility and the creation log file.

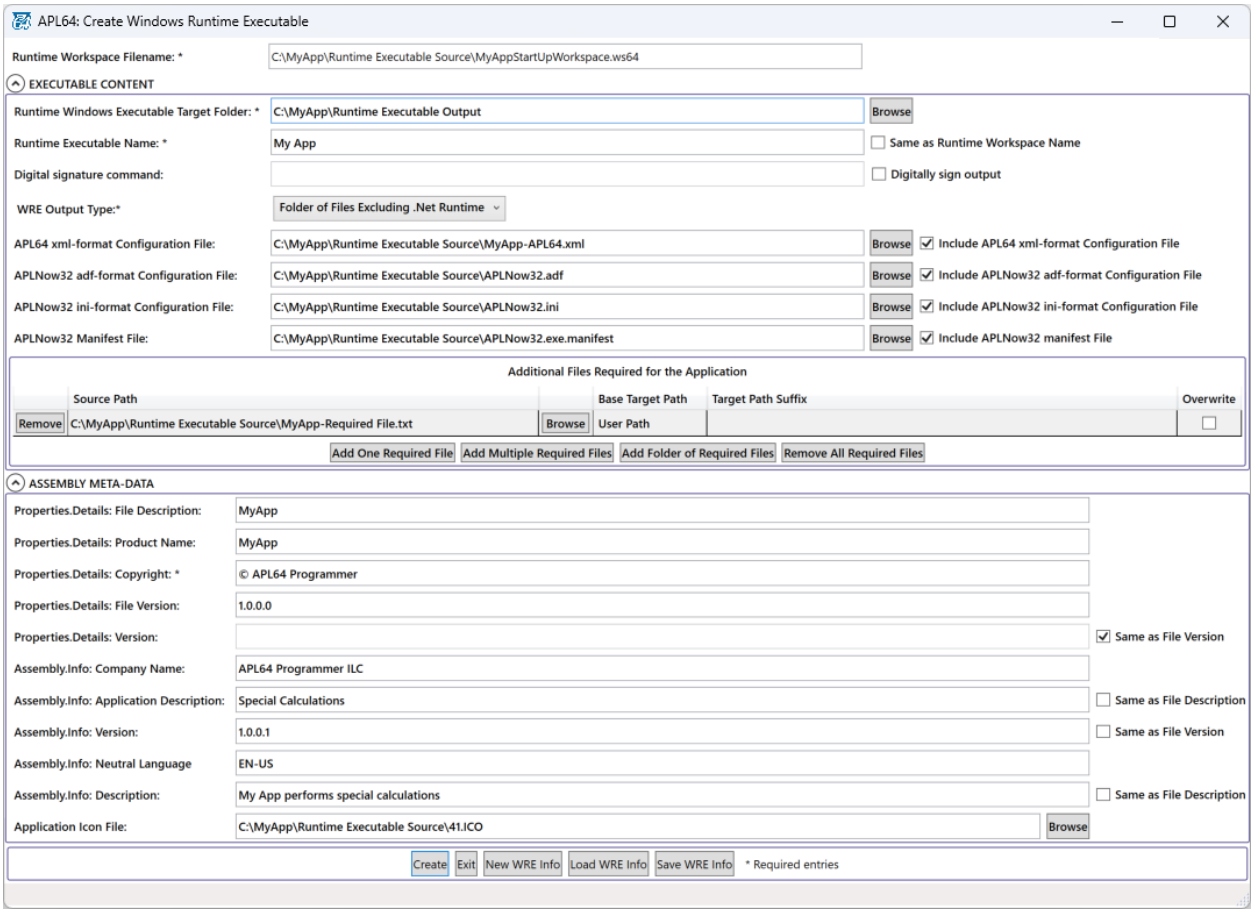
- C:\WreWsidExample\AppInstallationFolder folder is the analogue of the installation folder on the end-user's workstation.

Use the WRE utility to create the WS1.exe in the target folder

Load the WRE utility information file: C:\WreWsidExample\Source\WS1.wrtei and carefully observe the 'Additional Files Required for the Application' list items which illustrate the available WRE file materialization options:



Create the C:\WreWsidExample\Target\WS1.exe

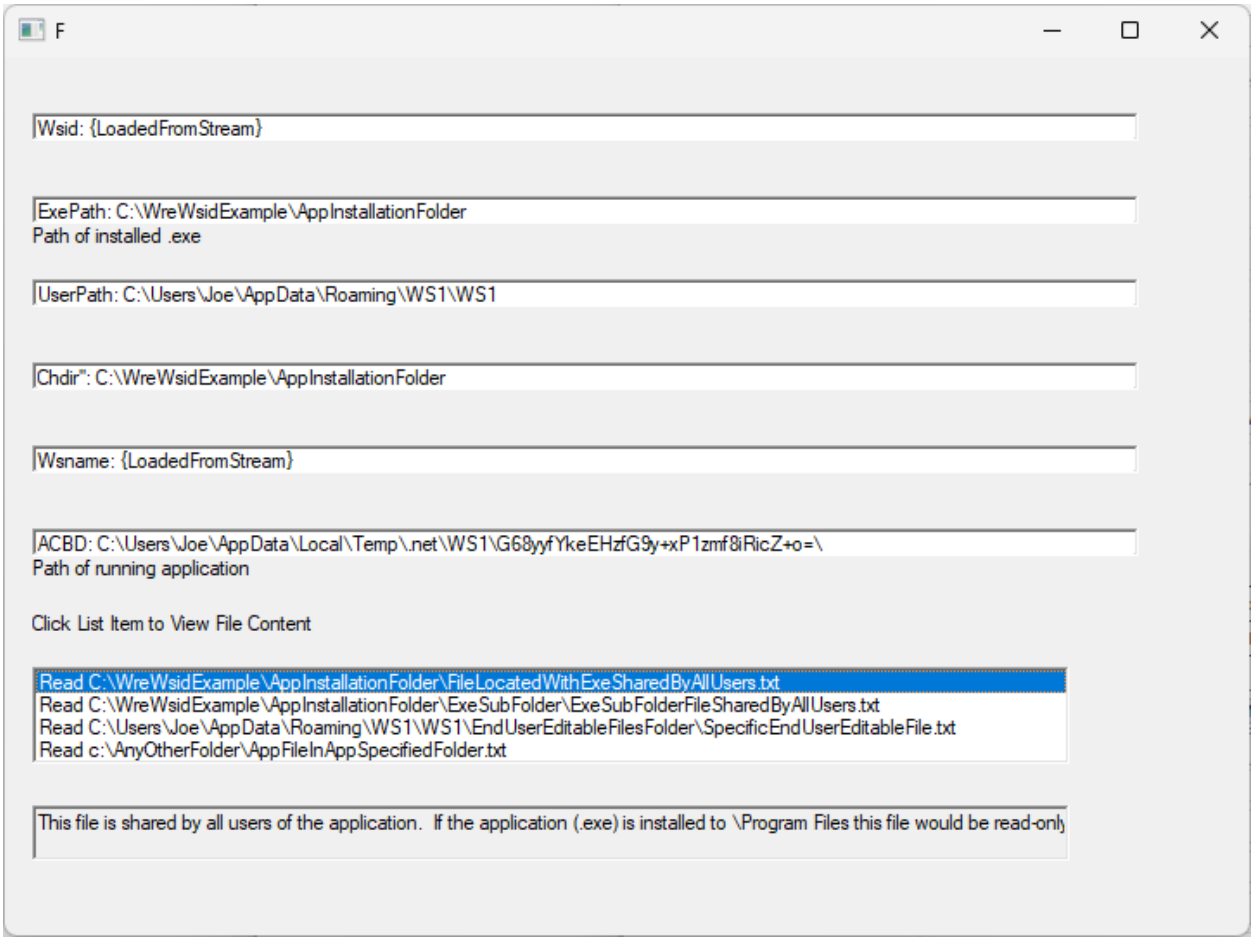


Install the WRE to the End-User's Workstation

Copy C:\WreWsidExample\Target\WS1.exe to the C:\WreWsidExample\AppInstallationFolder folder to simulate the installation of the WRE exe file to the end-user's workstation.

Run the WRE

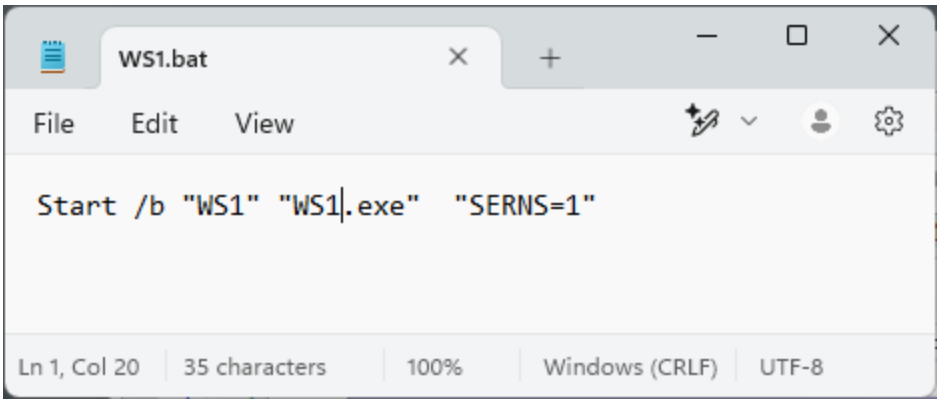
Run the C:\WreWsidExample\AppInstallationFolder\WS1.exe file to present the application GUI. Click on the file list items in the application GUI to view the content of the files in the WRE which were materialized on the target workstation, e.g.



View the Additional Files Embedded in the WRE

The files embedded in the WRE may be viewed by creating a Windows batch file in the same folder as the WS1.exe WRE file:

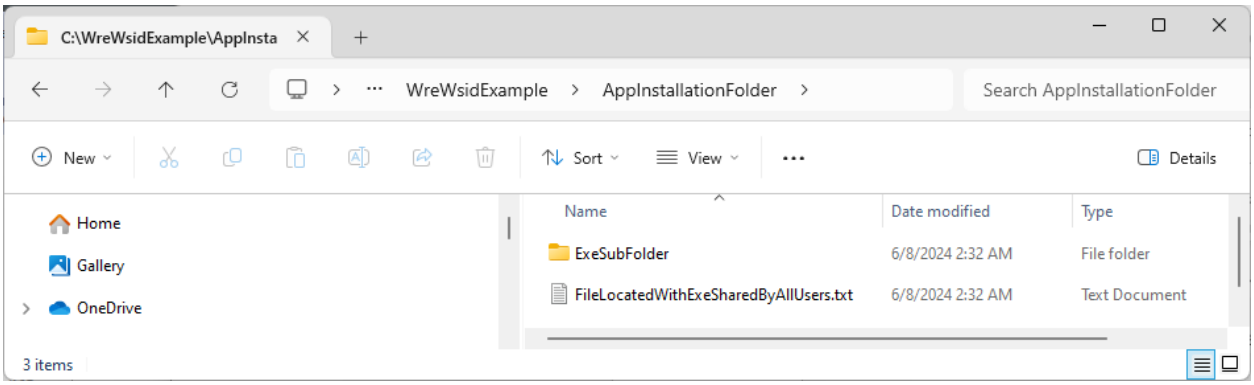
```
Start /b "WS1" "WS1.exe"  "SERNS=1"
```



The additional files and folders embedded in the WRE have GUID-based filenames to avoid name conflicts. When these files are materialized the APL64 programmer-selected 'Target Path Suffix' will be used as the physical file names.

Check the Materialized Files

The materialized files may be checked using Windows Explorer to browse to the materialization folders: □USERPATH, □EXEPATH and 'C:\AnyOtherFolder' and □ACBD folders to observe the files which were materialized when the WRE exe file was run, for example:



□WRE

The New, Load, Create, Run and Save actions of the WRE dialogue may also be performed under program control using the □WRE system function which is available in the APL64 Developer version.

The WRE dialogue and the □WRE system function share the same current WRE data set. The result of loading a WRE data set using the □WRE system function may be observed by viewing that WRE data set using the WRE dialogue.

The current WRE data set remains available during the current APL64 developer version instance but is not automatically saved when the current APL64 developer version instance ends. If appropriate, use the □WRE system function 'Save' action, or the WRE dialogue to save the current WRE data set for future APL64 developer instances with a programmer-selected file name.

If the WRE dialogue is used to create or modify the current WRE data set, the □WRE system function may be used to create or run a WRE based on that data set.

The □WRE system function is available only in the APL64 Developer version.

Syntax: result ← action □wre argument

Arguments:
action is the action to be performed. Action argument text is case-insensitive.

argument is the required argument to the action.

Result:
result is a two-element variable where the first element is an integer: 0 = success, ¯1 = not successful, and the second element is a character vector describing the action result.

□WRE New Action

Syntax: 'New' □WRE ''

The New action will initialize the WRE configuration data set in the current APL64 developer version instance.

⎕WRE Load Action

Syntax: 'Load' ⎕WRE pathToWreConfigurationFile
The Load action will load a pre-prepared WRE xml-format configuration file into the WRE configuration data set in the current APL64 developer version instance.

⎕WRE Create Action

Syntax: 'Create' ⎕WRE ''
The Create action will create a WRE executable using the WRE configuration data set in the current APL64 developer version instance.

⎕WRE Run Action

Syntax: 'Run' ⎕WRE pathToWreExecutable waitForExit commandLineArguments

The Run action will start an instance of the WRE application specified by the three elements of the right argument:

Element #	Data type	Right Argument Element Value
1	String or Character vector	Full path to WRE executable
2	Integer	Wait For Exit (milliseconds) or 0/Infinite wait
3	String or Character vector	WRE executable command line arguments

The result of the ⎕WRE run action is available only after the earlier of the expiration of the 'Wait For Exit' value or the closing of the ⎕WRE application instance.

⎕WRE PropValue Action

Syntax: priorPropValue ← 'PropValue' ⎕WRE propertyName newPropertyValue

The PropValue action will get, and optionally set, the properties in the WRE xml-format configuration file in the current APL64 instance. These properties correspond to the **Options | Create Runtime .Net Assembly | Create Windows Runtime Executable** dialog. The property names are not case sensitive. The PropValue action may be used to modify the WRE xml-format configuration file in the current APL64 instance.

Property Names		
Description	**Property Name**	**Type**
Runtime Windows Executable Target Folder	TgtFolder	Text
Runtime Executable Name	ExeName	Text
Runtime Executable Name: Same as Runtime Workspace Name	ExeNameSameAsWsName	Bool

Digital signature command	DigiSignCmd	Text
Digitally sign output	SignWre	Bool
WRE Output Type	WreOutputType	'SelfContained' 'FolderNoNet' 'SingleFileNoNet'
APL64 xml-format Configuration File	APL64ConfigFile	Text
Include APL64 xml-format Configuration File	InclAPL64ConfigFile	Bool
APLNow32 adf-format Configuration File	APLNow32AdfConfigFile	Text
Include APLNow32 adf-format Configuration File	Incl APLNow32AdfConfigFile	bool
APLNow32 ini-format Configuration File	APLNow32IniConfigFile	Text
Include APLNow32 ini-format Configuration File	InclAPLNow32IniConfigFile	bool
APLNow32 Manifest File	APLNow32ManifestFile	Text
Include APLNow32 Manifest File	InclAPLNow32ManifestFile	bool
Addnlappfiles	AddnlAppFiles	AAF-Matrix
Properties.Details: File Description	FileDesc	Text
Properties.Details: Product Name	ProdName	Text
Properties.Details: Copyright	Copyright	Text
Properties.Details: File Version	FileVersion	Text(#.#.#.#)
Properties.Details: Version: Same As File Version	VersionSameAsFileVersion	Bool
Properties.Details: Version	Version	Text(#.#.#.#)
Assembly.Info: Company Name	CoName	Text
Assembly.Info: Application Description	AppDesc	Text
Assembly.Info: Application Description: Same As File Description	AppDescSameAsFileDesc	bool
Assembly.Info: Assembly Version	AsmVer	Text(#.#.#.#)
Assembly.Info: Assembly Version Same As File Version	AsmVerSameAsFileVer	bool

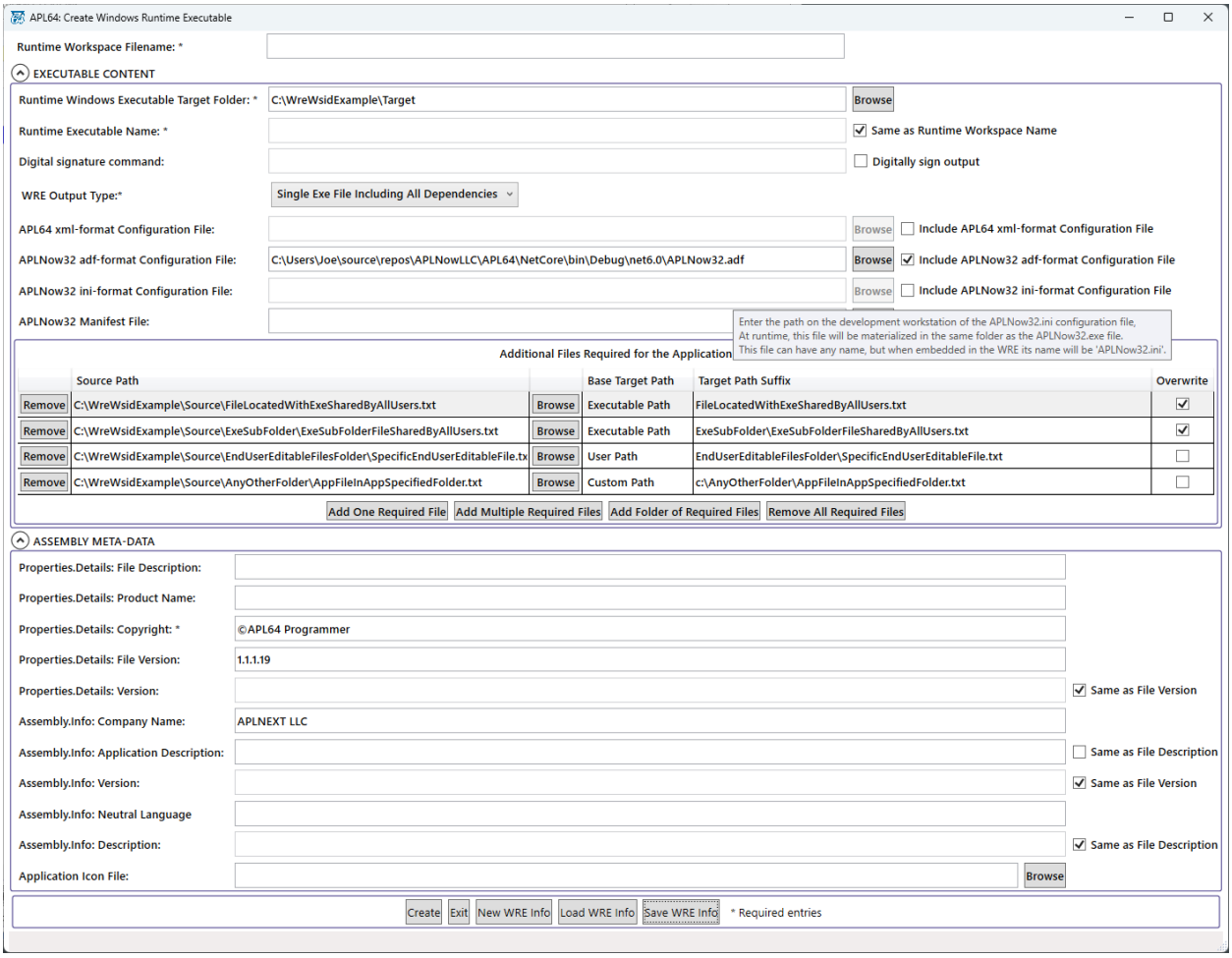
Assembly.Info: Neutral Language"	NeutLang	text
Assembly.Info: Description	Desc	text
Assembly.Info: Description Same As File Description	DescSameAsFileDesc	bool
Application Icon File	AppIconFile	text

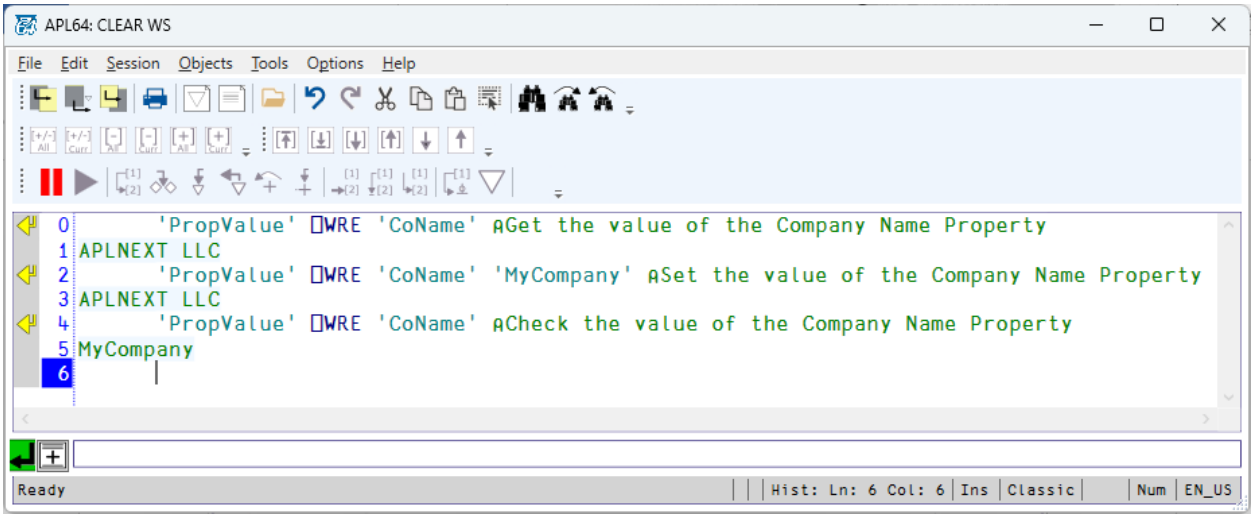
Text: Character vector or string scalar

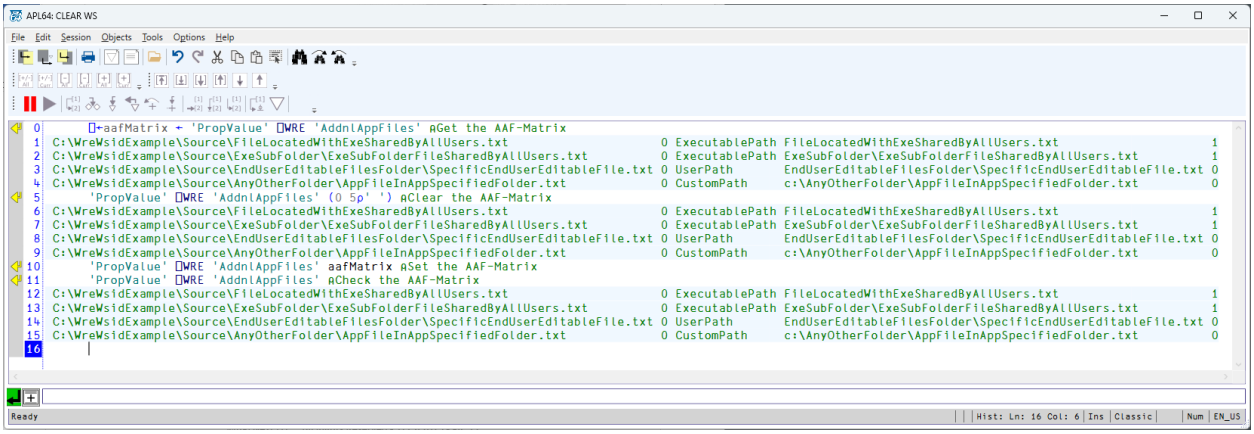
AAF-Matrix: One row for each additional application file	
Column Description	**Data Type**
Source Path	Text
Source Path is Folder	bool
Base Target Path	'ExecutablePath' or 'CustomPath'
Target Path	Text
Overwrite	bool

Examples:

WRE xml-format configuration file in the current APL64 instance:







⎕WRE Save Action

Syntax: 'Save' ⎕WRE targetFileName

The Save action will save the WRE configuration data set in the current APL64 developer version instance to the file specified in the right argument.

How the WRE Creation Utility Works

The APL64 programmer input to the WRE creation utility is used by the utility to define a .Net, C# language project in the target folder. This project contains:

- A console program which is the basis of the Windows desktop executable
- The APL64 programmer-specified APL64 start-up workspace
- The APL64 programmer-specified files required for the application system
- References to the .Net Nuget packages and assemblies required to run the APL64 interpreter

The WRE creation utility uses the .Net SDK installed on the APL64 programmer's workstation to publish the project into a Windows desktop executable (.exe) in the WRE target folder.

The .Net SDK publication parameters are set to create a 'ready to run' Windows desktop executable. A 'ready to run' executable contains the appropriate .Net runtime components, so that it is not necessary for the end user to download them from a Microsoft web site or install them on the end user's target workstation.