

Table of Contents

Introduction

Which Product to Choose

System Requirements

Basics

Virtual Environment

Virtual Environment

Attached Processes

How Attachment Works

Virtual Process

Shared Memory

Virtual File System

Virtual Registry

BoxedApp SDK

Introduction

Virtual Files

Creating Virtual Files

API

Functions

Options

Use Cases

Using COM / ActiveX Object without Registering It in the Registry

Loading DLL from Memory

Starting Application Directly from Memory

Intercepting Functions

License

BoxedApp Packer

Introduction

Plugins

Virtual Files

Virtual Registry

Command Line Overriding

License

BoxedApp Packer API

Introduction

API

Functions

Interfaces

License

BoxedApp

BoxedApp is a family of products for meeting the challenges in the area of application virtualization for Windows ® operating systems.

Application virtualization is primarily a simulation of a filesystem ("[virtual filesystem](#)"), registry ("[virtual registry](#)"), COM/ActiveX subsystem, and the process start system ("[virtual processes](#)").

BoxedApp is based on the interception and emulation of system calls, which an application makes to call the operating system with its requests to create files, list, remove, read, write them, etc., requests to create COM objects, run registry-related processes.

BoxedApp family of products:

- [BoxedApp SDK](#)
- [BoxedApp Packer](#)
- [BoxedApp Packer API](#)

BoxedApp SDK

BoxedApp SDK is a developer library for emulating the file system, registry and other Windows subsystems.

BoxedApp Packer

BoxedApp Packer is an end-user solution for packing third-party applications with all their dependencies into a single executable file (the so-called portable applications); it requires no programming whatsoever.

BoxedApp Packer API

BoxedApp Packer API is a developer library to create packed executables. BoxedApp Packer uses this API.

Which Product to Choose

[BoxedApp SDK](#) is a library for developers who want to emulate the file system and registry, run processes by loading them directly from the memory, use ActiveX without actually registering in the registry, and so on. Thus, BoxedApp SDK is a great choice for application developers (See also: [Typical use scenarios for BoxedApp SDK](#)).

[BoxedApp Packer](#) is an executable file packer solution. Use it the files to be packaged are known. You can use BoxedApp Packer for turning your or a third-party application into a portable application. BoxedApp Packer does not require the source code of the application; it deals with the binary executable files. The result of BoxedApp Packer's performance is an executable binary file that contains all the source files in the packed form. When started, the compressed files are extracted and placed in a virtual environment, so the application runs exactly as the original application would run if the files actually existed on the disk.

If your goal is to create a packer, use [BoxedApp Packer API](#) (BoxedApp Packer uses this API).

All the products in the BoxedApp line of products use the same technology, which supports virtual environment.

System Requirements

All the products of the BoxedApp family support both 32-bit and 64-bit environments. Supports all current versions of Windows, beginning with Windows 2000.

[BoxedApp Packer](#) packs both 32-bit and 64-bit executable files. It can even pack 64-bit files on a 32-bit Windows system (and vice versa).

BoxedApp does not have any special requirements for RAM or disk space.

Basics

- [Virtual Environment](#)
 - [Virtual Environment](#)
 - [Attached Processes](#)
 - [How Attachment Works](#)
 - [Virtual Process](#)
 - [Shared Memory](#)
- [Virtual File System](#)
- [Virtual Registry](#)

Virtual Environment

Virtual environment is virtual file system and virtual registry that are shared by multiple processes.

Attached Processes

Suppose, some process has initialized BoxedApp (see the function `BoxedAppSDK_Init`). When initializing the shared memory (uses MMF - memory mapped files), BoxedApp creates a virtual registry and a virtual file system. Now, other process can also use the virtual file system and the virtual registry. We say that these two processes *share a virtual environment*. Each of these processes is attached to the virtual environment. The initialization of BoxedApp, when an "attachment" to a virtual environment takes place, is called attaching to virtual environment.

The primary way of attaching a process to a virtual environment is running it in the context of an already attached process.

Such inheritance is regulated by the BoxedApp SDK options:

[`DEF_BOXEDAPPSDK_OPTION__EMBED_BOXEDAPP_IN_CHILD_PROCESSES`](#) and
[`DEF_BOXEDAPPSDK_OPTION__EMULATE_OUT_OF_PROC_COM_SERVERS`](#).

If the option [`DEF_BOXEDAPPSDK_OPTION__EMBED_BOXEDAPP_IN_CHILD_PROCESSES`](#) is set, all the child processes inherit the link to the virtual environment; i.e. use the same virtual file system and registry as the parent process.

The option [`DEF_BOXEDAPPSDK_OPTION__EMULATE_OUT_OF_PROC_COM_SERVERS`](#) matters when dealing with non-hosted COM servers. If the COM object is implemented as an EXE file that is virtual, BoxedApp will run the virtual EXE file to create the COM object.

If the COM object is implemented as an EXE file that does not exist in the virtual file system, by default the process based on the EXE file will be created by the system. And even if the option [`DEF_BOXEDAPPSDK_OPTION__EMBED`](#) is set, the new process will not be attached to the virtual environment, and hence will not have access, for example, to the virtual files. To attach such processes, set the option [`DEF_BOXEDAPPSDK_OPTION__EMULATE_OUT_OF_PROC_COM_SERVERS`](#).

To explicitly attach a process to a virtual environment, use the function [`BoxedAppSDK_AttachToProcess`](#). To detach a process, use the function [`BoxedAppSDK_AttachToProcess`](#).

How Attachment Works

Technically, a process can be attached by running a thread (uses the winapi function `CreateRemoteThread`) in the context of the process to be attached. The thread takes care of configuring the system function interceptors.

When attaching to a child process that is to be created, once initialized, the thread passes the control to the main thread of the child process.

Virtual Process

If the executable file is virtual, creating a process based on that file (such process is called a virtual process) requires a different approach. To create a virtual process, BoxedApp SDK selects a suitable exe file, which it can use for creating a new process. That exe file is called stub exe. Stub exe is run by CreateProcess with the `CREATE_SUSPENDED` flag. The actual stub exe file will not be run. BoxedApp expands the code of the virtual exe file in the context of the created process and passes the control to it. A virtual process is always attached to a virtual environment, since it's created from a virtual file. For the stub executable, BoxedApp uses the most suitable exe file from the Windows system folder.

Shared Memory

All attached processes have access to the shared memory, implemented on the basis of memory mapped files. It's the type of memory that stores data of the virtual file system and virtual registry. Developer can allocate and release memory blocks in the shared memory. These actions are facilitated by the [BoxedAppSDK_SharedMem_Alloc](#) and [BoxedAppSDK_SharedMem_Free](#) functions. [BoxedAppSDK_SharedMem_Alloc](#) returns a handler pointing at the memory block. To get the direct access to the allocated memory block, use the function [BoxedAppSDK_SharedMem_Lock](#), which returns the pointer. Once the data from the memory block is read, or the block is modified, BoxedApp calls the function [BoxedAppSDK_SharedMem_Unlock](#). This API is very similar to the winapi functions LocalAlloc / LocalLock / LocalUnlock / LocalFree.

Virtual File System

The heart of BoxedApp is the mechanism of intercepting calls to the operating system in the so-called user-mode). Unlike the systems that run in the kernel mode, BoxedApp does not require installing drivers and, hence, having the administrator privileges. In particular, BoxedApp intercepts all the calls that an application makes for using the file system.

BoxedApp introduces the concept of virtual file - a "file" that does not physically exist on the disk, but the application runs as if the "file" actually existed there. For example, an application attempts to open a virtual file C:\1.dll, by issuing the respective call. BoxedApp gets the control and checks the path to the file to be opened.

If the path points to a virtual file, instead of calling the original file open function, BoxedApp returns a virtual handle that points to the virtual file.

A set of virtual files forms a virtual file system.

Virtual Registry

Similar to the [virtual file system](#), virtual registry is an internal structure of BoxedApp SDK, located in the memory.

Virtual registry allows emulating registry keys and key values. And since the registry is especially critical for using the COM system, the virtual registry allows setting up the virtual registration of COM objects in the virtual registry.

To create a virtual registry key, use the function [BoxedAppSDK_CreateVirtualRegKey](#).

Virtual keys and values can also be created implicitly, if the option "[all changes are virtual](#)" is set. The virtual registry is shared among all the processes that share a [virtual environment](#).

BoxedApp SDK

BoxedApp SDK is a developer library for emulating the file system, registry and other Windows subsystems.

BoxedApp SDK comes as:

- bxsdk32.dll (for 32 bit applications) and bxsdk64.dll (for 64 bit applications), which can be used from any development environment, such as Visual C++, VB6, C#, VB.Net, Delphi, Builder C++.
- bxsdk32.lib (for 32 bit applications) and bxsdk64.lib (for 64 bit applications), a static library for Visual C++ (for all versions).
- BoxedAppSDK_Static.pas, a delphi file for static linking with applications written in Delphi or Builder C++ (for all versions).

Supports both x86 and x64 platforms.

Virtual Files

- [Creating Virtual Files](#)

Creating Virtual Files

[Virtual files](#) can be created explicitly and implicitly. Explicitly, a virtual file can be created with the following functions:

- [BoxedAppSDK_CreateVirtualFile](#) creates a virtual file with the contents located in the [shared memory](#).
- [BoxedAppSDK_CreateVirtualFileBasedOnStream](#) - creates a virtual file with the behavior determined by the implementation of the standard interface IStream.
- [BoxedAppSDK_CreateVirtualFileBasedOnBuffer](#) - creates a virtual file with the contents located in the memory buffer, which is used when reading and for writing.
- [BoxedAppSDK_CreateVirtualDirectory](#) - creates a virtual directory with the contents located in the [shared memory](#).

Implicitly, a virtual file can appear when the ["all changes are virtual"](#) mode is on. In this mode, a virtual file is created when you create a new file. If the parent directory of the file was created with [BoxedAppSDK_CreateCustomVirtualDirectory](#), the same method will be called for creating a file within it. And if the parent directory is neither a virtual file nor a file created with [BoxedAppSDK_CreateVirtualDirectory](#), then, depending on whether it's a directory or a regular file that is to be created, BoxedApp will use the [BoxedAppSDK_CreateVirtualFile](#) or [BoxedAppSDK_CreateVirtualDirectory](#) method.

API

- [Functions](#)
 - [BoxedAppSDK__Init](#)
 - [BoxedAppSDK__EnableDebugLog](#)
 - [BoxedAppSDK__SetLogFile](#)
 - [BoxedAppSDK__WriteLog](#)
 - [BoxedAppSDK__EnableOption](#)
 - [BoxedAppSDK__IsOptionEnabled](#)
 - [BoxedAppSDK__RemoteProcess__EnableOption](#)
 - [BoxedAppSDK__RemoteProcess__IsOptionEnabled](#)
 - [BoxedAppSDK__CreateVirtualFile](#)
 - [BoxedAppSDK__CreateVirtualFileBasedOnIStream](#)
 - [BoxedAppSDK__CreateVirtualFileBasedOnBuffer](#)
 - [BoxedAppSDK__CreateVirtualDirectory](#)
 - [BoxedAppSDK__DeleteFileFromVirtualFileSystem](#)
 - [BoxedAppSDK__CreateVirtualRegKey](#)
 - [BoxedAppSDK__EnumVirtualRegKeys](#)
 - [BoxedAppSDK__RegisterCOMLibraryInVirtualRegistry](#)
 - [BoxedAppSDK__RegisterCOMServerInVirtualRegistry](#)
 - [BoxedAppSDK__AttachToProcess](#)
 - [BoxedAppSDK__DetachFromProcess](#)
 - [BoxedAppSDK__HookFunction](#)
 - [BoxedAppSDK__GetOriginalFunction](#)
 - [BoxedAppSDK__EnableHook](#)
 - [BoxedAppSDK__UnhookFunction](#)
 - [BoxedAppSDK__RemoteProcess__LoadLibrary](#)
 - [BoxedAppSDK__SharedMem__Alloc](#)
 - [BoxedAppSDK__SharedMem__Free](#)
 - [BoxedAppSDK__SharedMem__Lock](#)
 - [BoxedAppSDK__SharedMem__Unlock](#)
 - [BoxedAppSDK__SharedMem__CreateStreamOnSharedMem](#)
- [Options](#)

Functions

- [BoxedAppSDK__Init](#)
- [BoxedAppSDK__EnableDebugLog](#)
- [BoxedAppSDK__SetLogFile](#)
- [BoxedAppSDK__WriteLog](#)
- [BoxedAppSDK__EnableOption](#)
- [BoxedAppSDK__IsOptionEnabled](#)
- [BoxedAppSDK__RemoteProcess__EnableOption](#)
- [BoxedAppSDK__RemoteProcess__IsOptionEnabled](#)
- [BoxedAppSDK__CreateVirtualFile](#)
- [BoxedAppSDK__CreateVirtualFileBasedOnStream](#)
- [BoxedAppSDK__CreateVirtualFileBasedOnBuffer](#)
- [BoxedAppSDK__CreateVirtualDirectory](#)
- [BoxedAppSDK__DeleteFileFromVirtualFileSystem](#)
- [BoxedAppSDK__CreateVirtualRegKey](#)
- [BoxedAppSDK__EnumVirtualRegKeys](#)
- [BoxedAppSDK__RegisterCOMLibraryInVirtualRegistry](#)
- [BoxedAppSDK__RegisterCOMServerInVirtualRegistry](#)
- [BoxedAppSDK__AttachToProcess](#)
- [BoxedAppSDK__DetachFromProcess](#)
- [BoxedAppSDK__HookFunction](#)
- [BoxedAppSDK__GetOriginalFunction](#)
- [BoxedAppSDK__EnableHook](#)
- [BoxedAppSDK__UnhookFunction](#)
- [BoxedAppSDK__RemoteProcess__LoadLibrary](#)
- [BoxedAppSDK__SharedMem__Alloc](#)
- [BoxedAppSDK__SharedMem__Free](#)
- [BoxedAppSDK__SharedMem__Lock](#)
- [BoxedAppSDK__SharedMem__Unlock](#)
- [BoxedAppSDK__SharedMem__CreateStreamOnSharedMem](#)

BoxedAppSDK_Init

Description

Initializes BoxedApp SDK, installs all the hooks, necessary for SDK operation, creates a virtual file system and a virtual registry in the memory.

Syntax

C++

```
BOOL __stdcall BoxedAppSDK_Init();
```

Delphi

```
function BoxedAppSDK_Init: BOOL; stdcall;
```

BoxedAppSDK_EnableDebugLog

Description

Enables or disables outputting debug data. During its operation, the SDK can record debug data that contains details on the performance of the SDK. By default, outputting debug data is disabled, but it can be enabled with `BoxedAppSDK_EnableDebugLog`. The SDK will output the data to the debugger window, but you can also have it duplicate all the output to a file using [BoxedAppSDK_SetLogFile](#).

Syntax

C++

```
void __stdcall BoxedAppSDK_EnableDebugLog (BOOL bEnable);
```

Delphi

```
procedure BoxedAppSDK_EnableDebugLog (bEnable: BOOL); stdcall;
```

Parameters

bEnable

Set it to TRUE to enable logging, or to FALSE to disable.

See Also

- [BoxedAppSDK_SetLogFile](#)
- [BoxedAppSDK_WriteLog](#)

BoxedAppSDK_SetLogFile

Description

During its operation, the SDK can record debug data that contains details on the performance of the SDK. By default, outputting debug data is disabled, but it can be turned on with [BoxedAppSDK_EnableDebugLog](#). The SDK will output the data to the debugger window, but you can also have it duplicate all the output to a file with `BoxedAppSDK_SetLogFile`.

Syntax

C++

```
void __stdcall BoxedAppSDK_SetLogFile(LPCTSTR szLogFilePath);
```

Delphi

```
procedure BoxedAppSDK_SetLogFile(szLogFilePath: PAnsiChar); stdcall;
```

Parameters

szLogFilePath

Path to the log file.

See Also

- [BoxedAppSDK_EnableDebugLog](#)
- [BoxedAppSDK_WriteLog](#)

BoxedAppSDK_WriteLog

Description

During its operation, the SDK can record debug data that contains details on the performance of the SDK. By default, outputting debug data is disabled, but it can be enabled with [BoxedAppSDK_EnableDebugLog](#). Plus, you can add some custom data to the log; that is what this function is meant for.

Syntax

C++

```
void __stdcall BoxedAppSDK_WriteLog(LPCTSTR szMessage);
```

Delphi

```
procedure BoxedAppSDK_WriteLog(szLogFilePath: PAnsiChar); stdcall;
```

Parameters

szMessage

A text to write to the log.

See Also

- [BoxedAppSDK_EnableDebugLog](#)
- [BoxedAppSDK_SetLogFile](#)

BoxedAppSDK_EnableOption

Description

You can control the behavior of the SDK by setting the required options. The function [BoxedAppSDK_EnableOption](#) sets option values: each option can be either enabled or disabled. Also, each option has a default value. The details on the available options are accumulated here: [BoxedApp SDK Options](#).

Syntax

C++

```
void __stdcall BoxedAppSDK_EnableOption(DWORD dwOptionIndex, BOOL bEnable);
```

Delphi

```
procedure BoxedAppSDK_EnableOption(dwOptionIndex: DWORD; bEnable: BOOL); stdcall;
```

Parameters

dwOptionIndex

Index of the option: [BoxedApp SDK Options](#).

bEnable

It's TRUE to enable the option. Otherwise, pass FALSE to disable the option.

See Also

- [BoxedAppSDK_IsOptionEnabled](#)
- [BoxedApp SDK Options](#)

BoxedAppSDK_IsOptionEnabled

Description

You can control the behavior of the SDK by setting the required options. With the function [BoxedAppSDK_IsOptionEnabled](#), you can find out the current value of the option; each option can be either enabled or disabled. Each option has a default value. The details on the available options are accumulated here: [BoxedApp SDK Options](#).

Syntax

C++

```
BOOL __stdcall BoxedAppSDK_IsOptionEnabled(DWORD dwOptionIndex);
```

Delphi

```
function BoxedAppSDK_IsOptionEnabled(dwOptionIndex: DWORD): BOOL; stdcall;
```

Parameters

dwOptionIndex

Index of the option: [BoxedApp SDK Options](#).

See Also

- [BoxedAppSDK_EnableOption](#)
- [BoxedApp SDK Options](#)

BoxedAppSDK_RemoteProcess_EnableOption

Description

You can control the behavior of the SDK by setting the required options. If for current process that's the function [BoxedAppSDK_EnableOption](#), for other, [attached process](#), that's the function `BoxedAppSDK_RemoteProcess_EnableOption`. Here the list of available options: [BoxedApp SDK Options](#).

Syntax

C++

```
void __stdcall BoxedAppSDK_RemoteProcess_EnableOption(  
    DWORD dwProcessId,  
    DWORD dwOptionIndex,  
    BOOL bEnable);
```

Delphi

```
procedure BoxedAppSDK_RemoteProcess_EnableOption(  
    dwProcessId: DWORD;  
    dwOptionIndex: DWORD;  
    bEnable: BOOL); stdcall;
```

Parameters

dwProcessId

The identifier of the process to be opened.

dwOptionIndex

Index of the option: [BoxedApp SDK Options](#).

bEnable

It's TRUE to enable the option. Otherwise, pass FALSE to disable the option.

See Also

- [BoxedAppSDK_RemoteProcess_IsOptionEnabled](#)
- [BoxedApp SDK Options](#)

BoxedAppSDK_RemoteProcess_IsOptionEnabled

Description

You can control the behavior of the SDK by setting the required options. If for current process that's the function [BoxedAppSDK_IsOptionEnabled](#), for other, [attached process](#), that's the function `BoxedAppSDK_RemoteProcess_IsOptionEnabled`. Here the list of available options: [BoxedApp SDK Options](#).

Syntax

C++

```
BOOL __stdcall BoxedAppSDK_RemoteProcess_IsOptionEnabled(  
    DWORD dwProcessId,  
    DWORD dwOptionIndex);
```

Delphi

```
function BoxedAppSDK_RemoteProcess_IsOptionEnabled(  
    dwProcessId: DWORD;  
    dwOptionIndex: DWORD): BOOL; stdcall;
```

Parameters

dwProcessId

The identifier of the process to be opened.

dwOptionIndex

Index of the option: [BoxedApp SDK Options](#).

bEnable

It's TRUE to enable the option. Otherwise, pass FALSE to disable the option.

See Also

- [BoxedAppSDK_RemoteProcess_EnableOption](#)
- [BoxedApp SDK Options](#)

BoxedAppSDK_CreateVirtualFile

Description

The function creates a virtual file with the contents located in the memory, shared among all the attached processes. The arguments of BoxedAppSDK_CreateVirtualFile are similar to the arguments of the winapi function CreateFile.

Syntax

C++

```
HANDLE __stdcall BoxedAppSDK_CreateVirtualFile(
    LPCTSTR szPath,
    DWORD dwDesiredAccess,
    DWORD dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD dwCreationDisposition,
    DWORD dwFlagsAndAttributes,
    HANDLE hTemplateFile);
```

Delphi

```
function BoxedAppSDK_CreateVirtualFile(
    lpFileName: PAnsiChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle): THandle; stdcall;

function BoxedAppSDK_CreateVirtualFileA(
    lpFileName: PAnsiChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle): THandle; stdcall;

function BoxedAppSDK_CreateVirtualFileW(
    lpFileName: PWideChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle): THandle; stdcall;
```

Parameters

lpFileName

The name, relative or full path of the virtual file to be created.

dwDesiredAccess

The requested access.

dwShareMode

The sharing mode.

lpSecurityAttributes

A pointer to a SECURITY_ATTRIBUTES structure that optionally specifies security descriptor, and whether the file handle can be inherited by child processes. **It's ignored by BoxedApp currently.**

dwCreationDisposition

An action to take on a virtual file that exists or doesn't exist.

dwFlagsAndAttributes

Combination of FILE_FLAG_* and FILE_ATTRIBUTE_*. You can pass 0.

See Also

- [Virtual File System](#)

BoxedAppSDK_CreateVirtualFileBasedOnIStream

Description

The function creates a virtual file, the behavior of which is determined by the implementation of the standard interface *IStream*. The arguments of `BoxedAppSDK_CreateVirtualFileBasedOnIStream` are similar to the arguments of the winapi function `CreateFile`.

[BoxedAppSDK_CreateVirtualFile](#) creates a file with the contents located in the shared memory. This is the easiest way to create a virtual file, which is sufficient for the majority of cases. However, sometimes you may need to have a virtual file with its data stored in other places: database, encrypted file or the Internet. For that purpose, BoxedApp SDK allows creating a virtual file, based on the implementation of the standard interface *IStream*. `BoxedAppSDK_CreateVirtualFileBasedOnIStream`.

Reading from such file takes the method *IStream::Read()*; writing to it calls *IStream::Write()*.

Creating a new handle that points to such file requires *IStream::Clone()*, which returns *IStream* with its own current pointer.

To set and get the current position in that file, BoxedApp uses *IStream::Seek()*.

To resize the file, it uses *IStream::SetSize()*. To get the size of the file, it can call the method *IStream::Stat()*.

Syntax

C++

```
HANDLE __stdcall BoxedAppSDK_CreateVirtualFileBasedOnIStream(
    LPCTSTR szPath,
    DWORD dwDesiredAccess,
    DWORD dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD dwCreationDisposition,
    DWORD dwFlagsAndAttributes,
    HANDLE hTemplateFile,
    LPSTREAM pStream);
```

Delphi

```
function BoxedAppSDK_CreateVirtualFileBasedOnIStream(
    lpFileName: PAnsiChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle;
    pStream: IStream): THandle; stdcall;

function BoxedAppSDK_CreateVirtualFileBasedOnIStreamA(
    lpFileName: PAnsiChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle;
    pStream: IStream): THandle; stdcall;

function BoxedAppSDK_CreateVirtualFileBasedOnIStreamW(
    lpFileName: PWideChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle;
    pStream: IStream): THandle; stdcall;
```

Parameters

lpFileName

The name, relative or full path of the virtual file to be created.

dwDesiredAccess

The requested access.

dwShareMode

The sharing mode.

lpSecurityAttributes

A pointer to a SECURITY_ATTRIBUTES structure that optionally specifies security descriptor, and whether the file handle can be inherited by child processes. **It's ignored by BoxedApp currently.**

dwCreationDisposition

An action to take on a virtual file that exists or doesn't exist.

dwFlagsAndAttributes

Combination of FILE_FLAG_* and FILE_ATTRIBUTE_*. You can pass 0.

pStream

The virtual file will read and write data from this stream.

See Also

- [Virtual File System](#)

BoxedAppSDK_CreateVirtualFileBasedOnBuffer

Description

The function creates a virtual file with the contents located in the specified buffer. Thus, the contents is both read from the buffer and written to the specified buffer. That file is useful when you need to create a virtual file of an invariable size (in particular, when the file is read-only). The arguments of BoxedAppSDK_CreateVirtualFileBasedOnBuffer are similar to the arguments of the winapi function CreateFile.

Syntax

C++

```
HANDLE __stdcall BoxedAppSDK_CreateVirtualFileBasedOnBuffer(
    LPCTSTR szPath,
    DWORD dwDesiredAccess,
    DWORD dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD dwCreationDisposition,
    DWORD dwFlagsAndAttributes,
    HANDLE hTemplateFile,
    PVOID pData,
    DWORD dwSize);
```

Delphi

```
function BoxedAppSDK_CreateVirtualFileBasedOnBuffer(
    lpFileName: PAnsiChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle;
    pData: Pointer;
    dwSize: DWORD): THandle; stdcall;

function BoxedAppSDK_CreateVirtualFileBasedOnBufferA(
    lpFileName: PAnsiChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle;
    pData: Pointer;
    dwSize: DWORD): THandle; stdcall;

function BoxedAppSDK_CreateVirtualFileBasedOnBufferW(
    lpFileName: PWideChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle;
    pData: Pointer;
    dwSize: DWORD): THandle; stdcall;
```

Parameters

lpFileName

The name, relative or full path of the virtual file to be created.

dwDesiredAccess

The requested access.

dwShareMode

The sharing mode.

lpSecurityAttributes

A pointer to a SECURITY_ATTRIBUTES structure that optionally specifies security descriptor, and whether the file handle can be inherited by

child processes. **It's ignored by BoxedApp currently.**

dwCreationDisposition

An action to take on a virtual file that exists or doesn't exist.

dwFlagsAndAttributes

Combination of FILE_FLAG_* and FILE_ATTRIBUTE_*. You can pass 0.

pData

The virtual file will read and write data from this buffer.

dwSize

The buffer size.

See Also

- [Virtual File System](#)

BoxedAppSDK_CreateVirtualDirectory

Description

The function creates a [virtual directory](#) with the contents located in the shared memory. The arguments of BoxedAppSDK_CreateVirtualDirectory are similar to the arguments of the winapi function CreateDirectory.

Syntax

C++

```
BOOL __stdcall BoxedAppSDK_CreateVirtualDirectory(  
    LPCTSTR lpPathName,  
    LPSECURITY_ATTRIBUTES lpSecurityAttributes);
```

Delphi

```
function BoxedAppSDK_CreateVirtualDirectory(  
    lpPathName: PAnsiChar;  
    lpSecurityAttributes: PSecurityAttributes): BOOL; stdcall;  
  
function BoxedAppSDK_CreateVirtualDirectoryA(  
    lpPathName: PAnsiChar;  
    lpSecurityAttributes: PSecurityAttributes): BOOL; stdcall;  
  
function BoxedAppSDK_CreateVirtualDirectoryW(  
    lpPathName: PWideChar;  
    lpSecurityAttributes: PSecurityAttributes): BOOL; stdcall;
```

Parameters

lpPathName

The name, relative or full path of the virtual directory to be created.

lpSecurityAttributes

A pointer to a SECURITY_ATTRIBUTES structure that optionally specifies security descriptor, and whether the file handle can be inherited by child processes. **It's ignored by BoxedApp currently.**

See Also

- [Virtual File System](#)

BoxedAppSDK_DeleteFileFromVirtualFileSystem

Description

The function deletes the file from the virtual file system. If the file is real, it is marked as deleted. If the file is virtual, the function completely removes it from the virtual environment. It is important to note the difference between this case and the situation when the virtual file is deleted by the regular winapi function `DeleteFile`. When deleting the file this way, the virtual file is only marked as deleted but is not actually removed. It is done this way, so that when the virtual file is created all over, its behavior would be determined by the function that the file was originally created with. Suppose, the virtual file was created based on `IStream`. It is removed with the `DeleteFile`, which is followed by calling the `CreateFile` with the name of that virtual file. In this case, the behavior of the newly created file will also be determined by the implementation of `IStream`. On the other hand, if the virtual file is deleted using `BoxedAppSDK_DeleteFileFromVirtualFileSystem`, the new file can be created as a real one. Thus, `BoxedAppSDK_DeleteFileFromVirtualFileSystem`, deletes information on the behavior of the virtual file.

Syntax

C++

```
BOOL __stdcall BoxedAppSDK_DeleteFileFromVirtualFileSystem(LPCTSTR szPath);
```

Delphi

```
function BoxedAppSDK_DeleteFileFromVirtualFileSystem(szPath: PAnsiChar): Boolean; stdcall;  
function BoxedAppSDK_DeleteFileFromVirtualFileSystemA(szPath: PAnsiChar): Boolean; stdcall;  
function BoxedAppSDK_DeleteFileFromVirtualFileSystemW(szPath: PWideChar): Boolean; stdcall;
```

Parameters

szPath

The name, relative or full path of the file to be deleted from virtual file system.

See Also

- [Virtual File System](#)

BoxedAppSDK_CreateVirtualRegKey

Description

The function creates a [virtual registry](#) key. Its arguments are similar to the arguments of the winapi function RegCreateKeyEx.

Syntax

C++

```
LONG __stdcall BoxedAppSDK_CreateVirtualRegKey(  
    HKEY hKey,  
    LPCTSTR lpSubKey,  
    DWORD Reserved,  
    LPCTSTR lpClass,  
    DWORD dwOptions,  
    REGSAM samDesired,  
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
    PHKEY phkResult,  
    LPDWORD lpdwDisposition);
```

Delphi

```
function BoxedAppSDK_CreateVirtualRegKey(  
    hKey: HKEY;  
    lpSubKey: PAnsiChar;  
    Reserved: DWORD;  
    lpClass: PAnsiChar;  
    dwOptions: DWORD;  
    samDesired: REGSAM;  
    lpSecurityAttributes: PSecurityAttributes;  
    var phkResult: HKEY;  
    lpdwDisposition: PDWORD): Longint; stdcall;
```

```
function BoxedAppSDK_CreateVirtualRegKeyA(  
    hKey: HKEY;  
    lpSubKey: PAnsiChar;  
    Reserved: DWORD;  
    lpClass: PAnsiChar;  
    dwOptions: DWORD;  
    samDesired: REGSAM;  
    lpSecurityAttributes: PSecurityAttributes;  
    var phkResult: HKEY;  
    lpdwDisposition: PDWORD): Longint; stdcall;
```

```
function BoxedAppSDK_CreateVirtualRegKeyW(  
    hKey: HKEY;  
    lpSubKey: PWideChar;  
    Reserved: DWORD;  
    lpClass: PWideChar;  
    dwOptions: DWORD;  
    samDesired: REGSAM;  
    lpSecurityAttributes: PSecurityAttributes;  
    var phkResult: HKEY;  
    lpdwDisposition: PDWORD): Longint; stdcall;
```

See Also

- [Virtual Registry](#)

BoxedAppSDK_EnumVirtualRegKeys

Description

The function lists all virtual registry keys. The argument of the function is a pointer to a callback function that is called for each virtual key. The callback function returns TRUE to continue or FALSE to halt listing the keys.

Syntax

C++

```
typedef BOOL (WINAPI *P_BoxedAppSDK_EnumVirtualRegKeysCallback) (
    HKEY hRootKey,
    LPCTSTR szSubKey,
    LPARAM lParam);

BOOL __stdcall BoxedAppSDK_EnumVirtualRegKeys(
    P_BoxedAppSDK_EnumVirtualRegKeysCallback pEnumFunc,
    LPARAM lParam);
```

Delphi

```
TBoxedAppSDK_EnumVirtualRegKeysCallbackA = function(hRootKey: HKEY; szSubKey: PAnsiChar; lParam: Cardinal): Boolean; stdcall;
TBoxedAppSDK_EnumVirtualRegKeysCallbackW = function(hRootKey: HKEY; szSubKey: PWideChar; lParam: Cardinal): Boolean; stdcall;
TBoxedAppSDK_EnumVirtualRegKeysCallback = TBoxedAppSDK_EnumVirtualRegKeysCallbackA;

function BoxedAppSDK_EnumVirtualRegKeys(
    pEnumFunc: TBoxedAppSDK_EnumVirtualRegKeysCallback;
    lParam: Cardinal): Boolean; stdcall;

function BoxedAppSDK_EnumVirtualRegKeysA(
    pEnumFunc: TBoxedAppSDK_EnumVirtualRegKeysCallbackA;
    lParam: Cardinal): Boolean; stdcall;

function BoxedAppSDK_EnumVirtualRegKeysW(
    pEnumFunc: TBoxedAppSDK_EnumVirtualRegKeysCallbackW;
    lParam: Cardinal): Boolean; stdcall;
```

See Also

- [Virtual Registry](#)

BoxedAppSDK_RegisterCOMLibraryInVirtualRegistry

Description

The function registers a COM/ActiveX library in the virtual registry, thus making the COM objects that are implemented in the library available to all the [attached processes](#). This allows [using COM objects without registering in the system registry](#).

Syntax

C++

```
DWORD __stdcall BoxedAppSDK_RegisterCOMLibraryInVirtualRegistry(LPCTSTR szPath);
```

Delphi

```
function BoxedAppSDK_RegisterCOMLibraryInVirtualRegistry(szVirtualFilePath: PAnsiChar): Longint; stdcall;  
function BoxedAppSDK_RegisterCOMLibraryInVirtualRegistryA(szVirtualFilePath: PAnsiChar): Longint; stdcall;  
function BoxedAppSDK_RegisterCOMLibraryInVirtualRegistryW(szVirtualFilePath: PWideChar): Longint; stdcall;
```

See Also

- [Virtual Registry](#)

BoxedAppSDK_RegisterCOMServerInVirtualRegistry

Description

The function registers a COM server, implemented as an executable file (exe), in the virtual registry, thus making the COM objects that are implemented in the server available to all the [attached processes](#). This allows [using COM objects without registering in the system registry](#).

Syntax

C++

```
DWORD __stdcall BoxedAppSDK_RegisterCOMServerInVirtualRegistry(LPCTSTR szCommandLine);
```

Delphi

```
function BoxedAppSDK_RegisterCOMServerInVirtualRegistry(szCommandLine: PAnsiChar): Longint; stdcall;  
function BoxedAppSDK_RegisterCOMServerInVirtualRegistryA(szCommandLine: PAnsiChar): Longint; stdcall;  
function BoxedAppSDK_RegisterCOMServerInVirtualRegistryW(szCommandLine: PWideChar): Longint; stdcall;
```

Parameters

szCommandLine

A command-line to register the exe server. Usually it is a path of the exe server with the command-line argument */RegServer* or *-RegServer*.

See Also

- [Virtual Registry](#)

BoxedAppSDK_AttachToProcess

Description

The function attaches the process to the [shared virtual environment](#). Please note that the argument of the function is **the handle of the process, and not its id**.

Syntax

C++

```
BOOL __stdcall BoxedAppSDK_AttachToProcess(HANDLE hProcess);
```

Delphi

```
function BoxedAppSDK_AttachToProcess(hProcess: THandle): BOOL; stdcall;
```

Parameters

hProcess

The handle of the process to attach.

See Also

- [Virtual Registry](#)

BoxedAppSDK_DetachFromProcess

Description

The function detaches the process from the [shared virtual environment](#). Please note that the argument of the function is **the handle of the process, and not its id**.

Syntax

C++

```
BOOL __stdcall BoxedAppSDK_DetachFromProcess (HANDLE hProcess);
```

Delphi

```
function BoxedAppSDK_DetachFromProcess (hProcess: THandle): BOOL; stdcall;
```

Parameters

hProcess

The handle of the process to detach.

BoxedAppSDK_HookFunction

Description

Creates a hook for the specified function.

For more information on using hooks, please see ["Intercepting Functions"](#).

Syntax

C++

```
HANDLE __stdcall BoxedAppSDK_HookFunction(  
    PVOID pFunction,  
    PVOID pHook,  
    BOOL bEnable);
```

Delphi

```
function BoxedAppSDK_HookFunction(  
    pFunction: Pointer;  
    pHook: Pointer;  
    bEnable: BOOL): THandle; stdcall;
```

Parameters

pFunction

The address of the function to hook.

pHook

The address of the hook function.

bEnable

If TRUE, the hook is enabled immediately. If FALSE, it is not enabled; you can enable it later using [BoxedAppSDK_EnableHook](#).

See Also

- [Intercepting Functions](#)

BoxedAppSDK_GetOriginalFunction

Description

Returns an address, which you can use to call the original function.

For more information on using hooks, please see ["Intercepting Functions"](#).

Syntax

C++

```
PVOID __stdcall BoxedAppSDK_GetOriginalFunction(HANDLE hHook);
```

Delphi

```
function BoxedAppSDK_GetOriginalFunction(hHook: THandle): Pointer; stdcall;
```

Parameters

hHook

The handle of the hook returned by [BoxedAppSDK_HookFunction](#).

See Also

- [Intercepting Functions](#)

BoxedAppSDK_EnableHook

Description

Activates or deactivates the hook. If the hook was created with `bEnable = FALSE` (see the function [BoxedAppSDK_HookFunction](#)), the hook is created but not activated. With the function `BoxedAppSDK_EnableHook`, you can do both activate a hook and deactivate it.

For more information on using hooks, please see ["Intercepting Functions"](#).

Syntax

C++

```
BOOL __stdcall BoxedAppSDK_EnableHook(HANDLE hHook, BOOL bEnable);
```

Delphi

```
function BoxedAppSDK_EnableHook(hHook: THandle; bEnable: BOOL): BOOL; stdcall;
```

Parameters

hHook

The handle of the hook returned by [BoxedAppSDK_HookFunction](#).

bEnable

If `TRUE`, the hook is activated. Otherwise, the hook is deactivated.

See Also

- [Intercepting Functions](#)

BoxedAppSDK_UnhookFunction

Description

Clears and removes the hook.

For more information on using hooks, please see ["Intercepting Functions"](#).

Syntax

C++

```
BOOL __stdcall BoxedAppSDK_UnhookFunction(HANDLE hHook);
```

Delphi

```
function BoxedAppSDK_UnhookFunction(hHook: THandle): BOOL; stdcall;
```

Parameters

hHook

The handle of the hook to clear. The handle is returned by [BoxedAppSDK_HookFunction](#).

See Also

- [Intercepting Functions](#)

BoxedAppSDK_RemoteProcess_LoadLibrary

Description

Loads the specified DLL to the specified process.

Syntax

C++

```
HMODULE __stdcall BoxedAppSDK_RemoteProcess_LoadLibrary(DWORD dwProcessId, LPCTSTR szPath);
```

Delphi

```
function BoxedAppSDK_RemoteProcess_LoadLibrary(dwProcessId: DWORD; szPath: PAnsiChar): HMODULE; stdcall;  
function BoxedAppSDK_RemoteProcess_LoadLibraryA(dwProcessId: DWORD; szPath: PAnsiChar): HMODULE; stdcall;  
function BoxedAppSDK_RemoteProcess_LoadLibraryW(dwProcessId: DWORD; szPath: PWideChar): HMODULE; stdcall;
```

Parameters

dwProcessId

The process id of the target process.

szPath

The path of the DLL to load.

BoxedAppSDK_SharedMem_Alloc

Description

The function allocates a memory block in [the memory shared among all attached processes](#).

Syntax

C++

```
typedef LONGLONG BOXEDAPP_SHARED_PTR;  
  
BOXEDAPP_SHARED_PTR __stdcall BoxedAppSDK_SharedMem_Alloc(int nSize);
```

Delphi

```
type  
    BOXEDAPP_SHARED_PTR = LargeInt;  
  
function BoxedAppSDK_SharedMem_Alloc(dwSize: DWORD): BOXEDAPP_SHARED_PTR; stdcall;
```

Parameters

dwSize

The size of the memory block to allocate.

BoxedAppSDK_SharedMem_Free

Description

The function releases the memory block in [the memory shared among all attached processes](#).

To allocate a memory block, use the function [BoxedAppSDK_SharedMem_Alloc](#).

Syntax

C++

```
typedef LONGLONG BOXEDAPP_SHARED_PTR;  
  
void __stdcall BoxedAppSDK_SharedMem_Free (BOXEDAPP_SHARED_PTR shared_ptr);
```

Delphi

```
type  
    BOXEDAPP_SHARED_PTR = LargeInt;  
  
procedure BoxedAppSDK_SharedMem_Free (shared_ptr: BOXEDAPP_SHARED_PTR); stdcall;
```

Parameters

shared_ptr

The shared pointer of the memory block to release.

BoxedAppSDK_SharedMem_Lock

Description

The function returns a pointer that can be used for accessing the memory block, allocated in [the memory shared among all attached processes](#).

To allocate a memory block, use the function [BoxedAppSDK_SharedMem_Alloc](#).

Syntax

C++

```
typedef LONGLONG BOXEDAPP_SHARED_PTR;  
  
PVOID __stdcall BoxedAppSDK_SharedMem_Lock(BOXEDAPP_SHARED_PTR shared_ptr);
```

Delphi

```
type  
    BOXEDAPP_SHARED_PTR = LargeInt;  
  
function BoxedAppSDK_SharedMem_Lock(shared_ptr: BOXEDAPP_SHARED_PTR): Pointer; stdcall;
```

Parameters

shared_ptr

The shared pointer of the memory block.

BoxedAppSDK_SharedMem_Unlock

Description

The function releases the pointer (**not the memory!**), which was used for accessing the memory block (see the [BoxedAppSDK_SharedMem_Lock](#)), allocated in [the memory shared among all attached processes](#).

To allocate a memory block, use the function [BoxedAppSDK_SharedMem_Alloc](#).

Syntax

C++

```
typedef LONGLONG BOXEDAPP_SHARED_PTR;  
  
BOOL __stdcall BoxedAppSDK_SharedMem_Unlock(BOXEDAPP_SHARED_PTR shared_ptr);
```

Delphi

```
type  
    BOXEDAPP_SHARED_PTR = LargeInt;  
  
function BoxedAppSDK_SharedMem_Unlock(shared_ptr: DWORD): BOOL; stdcall;
```

Parameters

shared_ptr

The shared pointer of the memory block.

BoxedAppSDK_SharedMem_CreateStreamOnSharedMem

Description

Returns IStream with its data located in [the memory shared among all attached processes](#).

Syntax

C++

```
HRESULT __stdcall BoxedAppSDK_SharedMem_CreateStreamOnSharedMem(LPSTREAM* ppStream);
```

Delphi

```
function BoxedAppSDK_SharedMem_CreateStreamOnSharedMem(var stm: IStream): HRESULT; stdcall;
```

Parameters

shared_ptr

The shared pointer of the memory block.

BoxedApp SDK Options

DEF_BOXEDAPPSDK_OPTION__ALL_CHANGES_ARE_VIRTUAL

By default, this option is not set. If this option is set, all the changes, made in the file system or the registry, are stored in the [virtual environment](#). For example:

C++

```
void main()
{
    BoxedAppSDK_Init();

    BoxedAppSDK_EnableOption(DEF_BOXEDAPPSDK_OPTION__ALL_CHANGES_ARE_VIRTUAL, TRUE);
    {
        // File "virtual.txt" will be created as virtual
        HANDLE hFile = CreateFile(
            _T("virtual.txt"),
            GENERIC_WRITE,
            FILE_SHARE_READ,
            NULL,
            CREATE_NEW,
            0,
            NULL);
    }
    BoxedAppSDK_EnableOption(DEF_BOXEDAPPSDK_OPTION__ALL_CHANGES_ARE_VIRTUAL, FALSE);
}
```

Delphi

```
var
    fs: TFileStream;
begin
    BoxedAppSDK_Init();
    BoxedAppSDK_EnableOption(DEF_BOXEDAPPSDK_OPTION__ALL_CHANGES_ARE_VIRTUAL, true);
    // File 'virtual.txt' will be created as virtual
    fs := TFileStream.Create('virtual.txt', fmCreate or fmOpenWrite);
    fs.Free();
    BoxedAppSDK_EnableOption(DEF_BOXEDAPPSDK_OPTION__ALL_CHANGES_ARE_VIRTUAL, false);
end;
```

DEF_BOXEDAPPSDK_OPTION__EMBED_BOXEDAPP_IN_CHILD_PROCESSES

By default, this option is not set. If this option is set, all the child processes will inherit [virtual environment](#).

DEF_BOXEDAPPSDK_OPTION__ENABLE_VIRTUAL_FILE_SYSTEM

By default, this option is set. If this option is set, all the calls to the file system are intercepted by BoxedApp. Disabling this option turns off the [virtual file system](#).

DEF_BOXEDAPPSDK_OPTION__ENABLE_VIRTUAL_REGISTRY

By default, this option is set. If this option is set, all the calls to the registry are intercepted by BoxedApp. Disabling this option turns off the [virtual registry](#).

DEF_BOXEDAPPSDK_OPTION__EMULATE_OUT_OF_PROC_COM_SERVERS

By default, this option is not set. If this option is set, BoxedApp assumes the creation of COM objects that are implemented as executable files. This allows attaching a COM server process to a shared [virtual environment](#).

DEF_BOXEDAPPSDK_OPTION__INHERIT_OPTIONS

By default, this option is not set. If this option is set, the newly created child process will get the same set of option values as the process that is creating the child process.

Use Cases

- [Using COM / ActiveX Object without Registering It in the Registry](#)
- [Loading DLL from Memory](#)
- [Starting Application Directly from Memory](#)
- [Intercepting Functions](#)

Using COM / ActiveX Object without Registering It in the Registry

COM/ActiveX components are traditionally registered in the system registry to allow the corresponding Windows API functions find the modules (DLL or EXE) that contain the code of the components.

A major problem that occurs with the registration is the lack of user privileges, necessary for writing to the registry.

Also, when creating portable applications, it is important to be able to run applications without prior installation.

BoxedApp SDK allows registering COM / ActiveX components in the virtual registry.

The two functions that facilitate this are:

- [BoxedAppSDK_RegisterCOMLibraryInVirtualRegistry](#), which registers the library of COM components made out as a DLL in the virtual registry.
- [BoxedAppSDK_RegisterCOMServerInVirtualRegistry](#), which registers the library of COM components made out as an EXE in the virtual registry.

After calling the required function, the application can create COM / ActiveX objects just as if they were actually registered in the system registry.

Loading DLL from Memory

Suppose that a third-party component is available exclusively as a DLL. Nevertheless, we need to get a single exe file at the output. There are several reasons for that:

- We may need to conceal the fact of using the DLL.
- Or we may need to protect the DLL from tampering by hackers.

With BoxedApp SDK, developer creates a virtual file (for example, using the function [BoxedAppSDK_CreateVirtualFile](#)) and writes the contents of the DLL file to that file. It can obtain data from any source: Internet, LAN, database or application resources. Finally, the data can be generated on the fly.

Now you can download the library using the LoadLibrary function, and the application will run just as if the DLL file actually existed on the disk.

Starting Application Directly from Memory

With a virtual file at hand, you can create a process based on the file by calling any function that creates a process: `CreateProcess`, `WinExec` or `ShellExecute`.

BoxedApp SDK intercepts the process creation request. If it detects an attempt to create a process based on the virtual file, it loads the so-called "stub" executable, creates the process based on it, writes the contents of the virtual file to it and then passes the control to it. Such [processes are called virtual](#).

Intercepting Functions

As you already know, BoxedApp is based on intercepting system functions. The interception mechanism used by BoxedApp is also available to developers. It includes the following functions:

- [BoxedAppSDK_HookFunction](#), which creates a hook and, optionally, activates it.
- [BoxedAppSDK_EnableHook](#), which activates the hook.
- [BoxedAppSDK_GetOriginalFunction](#), which returns the pointer, which can be used for calling the original function.
- [BoxedAppSDK_UnhookFunction](#), which clears the hook.

Here is how hooks work. The address of the function, calls from which are to be intercepted, is passed to [BoxedAppSDK_HookFunction](#). For example, for the function kernel32.dll!CreateFileW:

C++

```
PVOID pCreateFileW = (PVOID)GetProcAddress(GetModuleHandleW(L"kernel32.dll"), "CreateFileW");

HANDLE g_hCreateFileWHook = BoxedAppSDK_HookFunction(
    pCreateFileW,
    &My_CreateFileW,
    TRUE);
```

The interceptor function gets the control when someone calls the function. In this example, that's the function kernel32.dll!CreateFileW, which is called when creating or opening files. You can always call the original function, the address of which you can get using [BoxedAppSDK_GetOriginalFunction](#):

C++

```
HANDLE WINAPI My_CreateFileW(
    LPCWSTR lpFileName,
    DWORD dwDesiredAccess,
    DWORD dwShareMode,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    DWORD dwCreationDisposition,
    DWORD dwFlagsAndAttributes,
    HANDLE hTemplateFile)
{
    // ...
    // You can call original function if you need
    typedef HANDLE (WINAPI *P_CreateFileW)(
        LPCWSTR lpFileName,
        DWORD dwDesiredAccess,
        DWORD dwShareMode,
        LPSECURITY_ATTRIBUTES lpSecurityAttributes,
        DWORD dwCreationDisposition,
        DWORD dwFlagsAndAttributes,
        HANDLE hTemplateFile);

    P_CreateFileW pCreateFileW = (P_CreateFileW)BoxedAppSDK_GetOriginalFunction(g_hCreateFileWHook);

    return pCreateFileW(
        lpFileName,
        dwDesiredAccess,
        dwShareMode,
        lpSecurityAttributes,
        dwCreationDisposition,
        dwFlagsAndAttributes,
        hTemplateFile);
}
```

Here is a similar example in Delphi:

Delphi

```
type TCreateFileW = function(
    lpFileName: PWideChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle): THandle; stdcall;

var
    OriginalCreateFileW: TCreateFileW;

function My_CreateFileW(
```



```

    lpFileName: PWideChar;
    dwDesiredAccess, dwShareMode: Integer;
    lpSecurityAttributes: PSecurityAttributes;
    dwCreationDisposition, dwFlagsAndAttributes: DWORD;
    hTemplateFile: THandle): THandle; stdcall;
begin
    ...
    Result := OriginalCreateFileW(
        lpFileName,
        dwDesiredAccess,
        dwShareMode,
        lpSecurityAttributes,
        dwCreationDisposition,
        dwFlagsAndAttributes,
        hTemplateFile);
end;

var
    pCreateFileW: Pointer;
    hHook__CreateFileW: THandle;
begin
    BoxedAppSDK_Init;
    pCreateFileW := GetProcAddress(GetModuleHandle(kernel32.dll), CreateFileW);
    hHook__CreateFileW := BoxedAppSDK_HookFunction(pCreateFileW, @My_CreateFileW, FALSE);
    OriginalCreateFileW := BoxedAppSDK_GetOriginalFunction(hHook__CreateFileW);
    BoxedAppSDK_EnableHook(hHook__CreateFileW, TRUE);
end.

```

THIS IS A CONTRACT. YOU SHOULD CAREFULLY READ ALL THE TERMS AND CONDITIONS BEFORE INSTALLING THIS SOFTWARE. BY INSTALLING THE SOFTWARE, YOU ACCEPT ALL THE TERMS AND CONDITIONS OF THIS LICENSE.

This License Agreement is a legal Agreement between you, an individual or organization, and Softanics.

Subject

The subject of the present Agreement is "BoxedApp SDK" hereinafter mentioned as Software including the whole delivery package, namely the software itself and the covering documentation.

Accepting or rejecting the Agreement

Any usage of this product including installation of the Software or any of its parts on a computer, loading the product into operating memory (RAM) or permanent storage on a computer's disk or other storage medium, as well as any other type of usage means that you accept all the terms and conditions of the present Agreement. If you do not agree with any statement of this License, you should promptly terminate usage of the product and delete all the files referred to it, as its components, so as the results of its work, from your computer. You should also return all the existing mediums containing the Software to the place where you obtained it.

Copyright

This Software is owned by Softanics and is protected by international copyright treaties. Any changes to the Software and its components, any additions to it, including the case of running and executing on your computer any software not built-in into your operating system and affecting the Software's operation or changing its results, as well as storage and distribution of such a changed or augmented Software, are strictly forbidden.

Usage

The Software is licensed by the present Agreement to be used for any commercial and other proper purposes. Applications built using our components are royalties free, but if you need to create a development tool (as DCU, DLL, OCX etc..) integrating this functionality you need to obtain a special license. You acknowledge that the Software in source code form remains a confidential trade secret of Softanics or its suppliers and therefore you agree not to modify the Software or attempt to decipher, decompile, disassemble or reverse engineer the Software. In order to obtain the source codes you should turn to Softanics so as to purchase and register the Software's version including source codes.

Applications that either (a) produce file / files that use BoxedApp SDK, or (b) produce file / files that should be passed to a separate application that uses BoxedApp SDK, are called "packers". Vendors of "packers" need to obtain a special license.

Time restriction

This product has no limits in terms of its usage.

Licensed copies number restriction

Single developer license is assigned for installation and execution on one computer assembled as a solitary system units and by one user only. If you have an upgrade to this Software, it constitutes a single product together with the product that you upgraded, and may not be used to increase the total number of licensed installations of the Software. Group programming projects making use of this software must purchase a copy of the software for each member of the group or purchase a Site Developer License. Site Developer License is assigned for installation and execution on unlimited numbers of computers by any user inside one building of organization located in one city. Any possible delivery of the product or its parts as well as registration codes or access codes you received with purchasing and/or registering this Software to any individual or organization is strictly prohibited.

Distribution restrictions

Except as provided in this License Agreement, you may not transfer, sell, rent, lease, lend, copy, modify, emulate, clone, decompile, translate, sublicense, time-share or electronically transmit or receive the Software or any of its parts or to distribute it in any other way.

Registration codes and access codes

Registration codes (e.g., serial number) or access codes (e.g. archive file password) you received with purchasing and/or registering this Software are not the subject to the present Agreement. You may store any number of copies of such codes in any form if only you are a registered user and if you obtained them in the registration process or purchasing the Software from Softanics or authorized persons, individuals or organizations. Any distribution or delivery of the registration and access codes to any third party as well as any form of publication is prohibited. The copyright of all registration and access codes remains the property of Softanics which reserves the right to withhold or withdraw authorization of use of all registration codes issued to any user if there is reasonable evidence to indicate that the user is involved in a breach of the terms of this License Agreement.

Modifications to the Software

Softanics reserves the right to change the executable code and to remove, add or change the Software's functions.

Upgrade policy

A registered user can receive software updates free of charge within 1 year period that begins on the purchasing date. A registered user can buy the right to get updates for the next 1 year period for a discounted price (75% of the regular price) during grace period that starts on the purchasing date and ends on the purchasing date + 2 months.

Warranty and guarantee

This documentation and the source library are provided "as is" without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and/or suitability for a particular purpose. The user assumes the entire risk of any damage caused by this software.

Changing terms and conditions of this Agreement

Any variation to the terms of this Agreement shall only be valid if made and delivered by Softanics. If you do not agree to any clause of this Agreement you may ask for explanations from Softanics. However, this does not authorize you to consider this Agreement nullified.

Introduction

BoxedApp Packer is a packer application for packing exe files, DLLs, ActiveXs and other files into a single executable file. You can place all libraries that the original exe file depends on, ActiveX controls, and just all kinds of files in it.

The primary problem that BoxedApp Packer resolves is creating applications that don't require the installation. But, at the same time, you would be free to choose components for creating the application - you can use any third-party DLL, any ActiveX. Simply "tell" BoxedApp Packer which files the application depends on, and it will generate a self-sustaining exe file.

An executable file made this way doesn't create temporary files on the disk, it doesn't modify the registry to ensure that all the embedded files run as if they were real. The embedded files are extracted directly to the memory, while ActiveX / COM libraries are registered in the virtual registry.

Moreover, you can further expand the functionality of the obtained application by creating a plugin - a special DLL - for it. Plugins are called when the application starts or terminates. Plugins provide a special API - [BoxedApp SDK](#), which allows creating virtual files "on the fly", working with the virtual registry, and a lot more. Thus, you can create truly flexible applications. For instance, your application, when it starts, could load necessary DLL over network or through the Internet and then use them as if they were really present on the disk.

Plugins

An individual category of files is plugins. They are DLLs that expand the functionality of the application being created.

Plugins can utilize the full power of [BoxedApp SDK](#) and namely the capability of creating any virtual files, virtual registry operations, etc.

Virtual Files

Virtual file is a file that exists for packed executable and for processes in the [virtual environment](#) only.

Virtual file are useful when:

- you want to hide the use of third-party components
- you want to protect individual files (animation, video, images)
- you want to protect your intellectual property, etc.
- you don't want to pollute your disk and so on.

Also you can create a virtual file using powerfull [BoxedApp SDK](#) from a [plugin](#).

Virtual Registry in BoxedApp Packer

BoxedApp Packer provides so-called "virtual registry", is a set of registry keys and values that are visible only for a packed application.

Virtual registry is useful when:

- you want to provide some registry settings to an application
- and so on.

See Also

- [Virtual Registry](#)

Command Line Overriding

If you specify embedded arguments, a final EXE will run as if the arguments are really passed to the program.

You can specify variables; for example, you pack notepad.exe, add a virtual file "1.txt" and specify "<BoxedAppVar:ExeFullPath> 1.txt" as an overridden command line. A packed notepad.exe will show 1.txt loaded from a virtual file.

THIS IS A CONTRACT. YOU SHOULD CAREFULLY READ ALL THE TERMS AND CONDITIONS BEFORE INSTALLING THIS SOFTWARE. BY INSTALLING THE SOFTWARE, YOU ACCEPT ALL THE TERMS AND CONDITIONS OF THIS LICENSE.

This License Agreement is a legal Agreement between you, an individual or organization, and Softanics.

Subject

The subject of the present Agreement is "BoxedApp Packer" hereinafter mentioned as Software including the whole delivery package, namely the software itself and the covering documentation.

Accepting or rejecting the Agreement

Any usage of this product including installation of the Software or any of its parts on a computer, loading the product into operating memory (RAM) or permanent storage on a computer's disk or other storage medium, as well as any other type of usage means that you accept all the terms and conditions of the present Agreement. If you do not agree with any statement of this License, you should promptly terminate usage of the product and delete all the files referred to it, as its components, so as the results of its work, from your computer. You should also return all the existing mediums containing the Software to the place where you obtained it.

Copyright

This Software is owned by Softanics and is protected by international copyright treaties. Any changes to the Software and its components, any additions to it, including the case of running and executing on your computer any software not built-in into your operating system and affecting the Software's operation or changing its results, as well as storage and distribution of such a changed or augmented Software, are strictly forbidden.

Usage

The Software is licensed by the present Agreement to be used for any commercial and other proper purposes. Applications built using our components are royalties free, but if you need to create a development tool (as DCU, DLL, OCX etc..) integrating this functionality you need to obtain a special license. You acknowledge that the Software in source code form remains a confidential trade secret of Softanics or its suppliers and therefore you agree not to modify the Software or attempt to decipher, decompile, disassemble or reverse engineer the Software. In order to obtain the source codes you should turn to Softanics so as to purchase and register the Software's version including source codes.

If application built by BoxedApp Packer is not used as is, but instead is used to prepare some sort of "packages" (e.g. packed application acts as a "launcher", and it is designed to play some kind of videos / documents that are packed externally by some tool), such applications are called "packers". Vendors of "packers" need to obtain a special license.

Time restriction

This product has no limits in terms of its usage.

Licensed copies number restriction

Single License is assigned for installation and execution on one computer assembled as a solitary system units and by one user only. If you have an upgrade to this Software, it constitutes a single product together with the product that you upgraded, and may not be used to increase the total number of licensed installations of the Software. Group programming projects making use of this software must purchase a copy of the software for each member of the group or purchase a Site License. Site License is assigned for installation and execution on unlimited numbers of computers by any user inside one building of organization located in one city. Any possible delivery of the product or its parts as well as registration codes or access codes you received with purchasing and/or registering this Software to any individual or organization is strictly prohibited.

Distribution restrictions

Except as provided in this License Agreement, you may not transfer, sell, rent, lease, lend, copy, modify, emulate, clone, decompile, translate, sublicense, time-share or electronically transmit or receive the Software or any of its parts or to distribute it in any other way.

Registration codes and access codes

Registration codes (e.g., serial number) or access codes (e.g. archive file password) you received with purchasing and/or registering this Software are not the subject to the present Agreement. You may store any number of copies of such codes in any form if only you are a registered user and if you obtained them in the registration process or purchasing the Software from Softanics or authorized persons, individuals or organizations. Any distribution or delivery of the registration and access codes to any third party as well as any form of publication is prohibited. The copyright of all registration and access codes remains the property of Softanics which reserves the right to withhold or withdraw authorization of use of all registration codes issued to any user if there is reasonable evidence to indicate that the user is involved in a breach of the terms of this License Agreement.

Modifications to the Software

Softanics reserves the right to change the executable code and to remove, add or change the Software's functions.

Upgrade policy

A registered user can receive software updates free of charge within 1 year period that begins on the purchasing date. A registered user can buy the right to get updates for the next 1 year period for a discounted price (75% of the regular price) during grace period that starts on the purchasing date and ends on the purchasing date + 2 months. 2 Warranty and guarantee

This documentation and the source library are provided "as is" without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and/or suitability for a particular purpose. The user assumes the entire risk of any damage caused by this software.

Changing terms and conditions of this Agreement

Any variation to the terms of this Agreement shall only be valid if made and delivered by Softanics. If you do not agree to any clause of this Agreement you may ask for explanations from Softanics However, this does not authorize you to consider this Agreement nullified.

Add your introductions here!

API

- [Functions](#)
 - [BxPackerApi_CreateProject](#)
- [Interfaces](#)
 - [IBxProject](#)
 - [Methods](#)
 - [put_InputPath](#)
 - [get_InputPath](#)
 - [put_OutputPath](#)
 - [get_OutputPath](#)
 - [put_ShareVirtualEnvironmentWithChildProcesses](#)
 - [get_ShareVirtualEnvironmentWithChildProcesses](#)
 - [put_EnableDebugLog](#)
 - [get_EnableDebugLog](#)
 - [put_EnableVirtualRegistry](#)
 - [get_EnableVirtualRegistry](#)
 - [put_HideVirtualFileFromFileDialog](#)
 - [get_HideVirtualFileFromFileDialog](#)
 - [put_AllChangesAreVirtual](#)
 - [get_AllChangesAreVirtual](#)
 - [put_SetIcon](#)
 - [get_SetIcon](#)
 - [put_IconPath](#)
 - [get_IconPath](#)
 - [put_EnableSplashScreen](#)
 - [get_EnableSplashScreen](#)
 - [put_SplashScreenPath](#)
 - [get_SplashScreenPath](#)

Functions

- [BxPackerApi_CreateProject](#)

BxPackerApi_CreateProject

Description

Creates [project file](#).

Syntax

C++

```
HRESULT BXPACKERAPI BxPackerApi_CreateProject (IBxProject** ppProject);
```

Interfaces

- [IBxProject](#)
 - [Methods](#)
 - [put_InputPath](#)
 - [get_InputPath](#)
 - [put_OutputPath](#)
 - [get_OutputPath](#)
 - [put_ShareVirtualEnvironmentWithChildProcesses](#)
 - [get_ShareVirtualEnvironmentWithChildProcesses](#)
 - [put_EnableDebugLog](#)
 - [get_EnableDebugLog](#)
 - [put_EnableVirtualRegistry](#)
 - [get_EnableVirtualRegistry](#)
 - [put_HideVirtualFileFromFileDialog](#)
 - [get_HideVirtualFileFromFileDialog](#)
 - [put_AllChangesAreVirtual](#)
 - [get_AllChangesAreVirtual](#)
 - [put_SetIcon](#)
 - [get_SetIcon](#)
 - [put_IconPath](#)
 - [get_IconPath](#)
 - [put_EnableSplashScreen](#)
 - [get_EnableSplashScreen](#)
 - [put_SplashScreenPath](#)
 - [get_SplashScreenPath](#)

IBxProject

Description

This is a BoxedApp Packer project interface.

Methods

- [put_InputPath](#)
- [get_InputPath](#)
- [put_OutputPath](#)
- [get_OutputPath](#)
- [put_ShareVirtualEnvironmentWithChildProcesses](#)
- [get_ShareVirtualEnvironmentWithChildProcesses](#)
- [put_EnableDebugLog](#)
- [get_EnableDebugLog](#)
- [put_EnableVirtualRegistry](#)
- [get_EnableVirtualRegistry](#)
- [put_HideVirtualFileFromFileDialog](#)
- [get_HideVirtualFileFromFileDialog](#)
- [put_AllChangesAreVirtual](#)
- [get_AllChangesAreVirtual](#)
- [put_SetIcon](#)
- [get_SetIcon](#)
- [put_IconPath](#)
- [get_IconPath](#)
- [put_EnableSplashScreen](#)
- [get_EnableSplashScreen](#)
- [put_SplashScreenPath](#)
- [get_SplashScreenPath](#)

Methods

- [put_InputPath](#)
- [get_InputPath](#)
- [put_OutputPath](#)
- [get_OutputPath](#)
- [put_ShareVirtualEnvironmentWithChildProcesses](#)
- [get_ShareVirtualEnvironmentWithChildProcesses](#)
- [put_EnableDebugLog](#)
- [get_EnableDebugLog](#)
- [put_EnableVirtualRegistry](#)
- [get_EnableVirtualRegistry](#)
- [put_HideVirtualFileFromFileDialog](#)
- [get_HideVirtualFileFromFileDialog](#)
- [put_AllChangesAreVirtual](#)
- [get_AllChangesAreVirtual](#)
- [put_SetIcon](#)
- [get_SetIcon](#)
- [put_IconPath](#)
- [get_IconPath](#)
- [put_EnableSplashScreen](#)
- [get_EnableSplashScreen](#)
- [put_SplashScreenPath](#)
- [get_SplashScreenPath](#)

put_InputPath

Description

Sets the path of the input file to pack.

Syntax

C++

```
HRESULT put_InputPath([in] BSTR strPath);
```

get_InputPath

Description

Returns the path of the input file to pack.

Syntax

C++

```
HRESULT get_InputPath([out, retval] BSTR* pstrPath);
```

put_OutputPath

Description

Sets the path of the output (i.e. packed) file.

Syntax

C++

```
HRESULT put_OutputPath([in] BSTR strPath);
```

get_OutputPath

Description

Returns the path of the output (i.e. packed) file.

Syntax

C++

```
HRESULT get_OutputPath([out, retval] BSTR* pstrPath);
```

put_ShareVirtualEnvironmentWithChildProcesses

Description

If *bValue* is VARIANT_TRUE, child processes of the packed exe share [virtual environment](#).

Syntax

C++

```
HRESULT put_ShareVirtualEnvironmentWithChildProcesses([in] VARIANT_BOOL bValue);
```

get_ShareVirtualEnvironmentWithChildProcesses

Description

If returns VARIANT_TRUE, child processes of the packed exe share [virtual environment](#).

Syntax

C++

```
HRESULT get_ShareVirtualEnvironmentWithChildProcesses([out, retval] VARIANT_BOOL* pbValue);
```

put_EnableDebugLog

Description

If *bValue* is VARIANT_TRUE, the packed exe produces debug log.

Syntax

C++

```
HRESULT put_EnableDebugLog([in] VARIANT_BOOL bValue);
```


get_EnableDebugLog

Description

If returns VARIANT_TRUE, the packed exe produces debug log.

Syntax

C++

```
HRESULT get_EnableDebugLog([out, retval] VARIANT_BOOL* pbValue);
```

put_EnableVirtualRegistry

Description

If *bValue* is VARIANT_TRUE, the packed exe uses virtual registry.

Syntax

C++

```
HRESULT put_EnableVirtualRegistry([in] VARIANT_BOOL bValue);
```

get_EnableVirtualRegistry

Description

If returns VARIANT_TRUE, the packed exe uses virtual registry.

Syntax

C++

```
HRESULT get_EnableVirtualRegistry([out, retval] VARIANT_BOOL* pbValue);
```

put_HideVirtualFileFromFileDialog

Description

If *bValue* is VARIANT_TRUE, the packed exe doesn't show virtual file in the system file dialogs.

Syntax

C++

```
HRESULT put_HideVirtualFileFromFileDialog([in] VARIANT_BOOL bValue);
```

get_HideVirtualFileFromFileDialog

Description

If returns VARIANT_TRUE, the packed exe doesn't show virtual file in the system file dialogs.

Syntax

C++

```
HRESULT get_HideVirtualFileFromFileDialog([out, retval] VARIANT_BOOL* pbValue);
```

put_AllChangesAreVirtual

Description

If *bValue* is VARIANT_TRUE, the packed exe stores all changes of the file system and the registry in virtual environment.

Syntax

C++

```
HRESULT put_AllChangesAreVirtual([in] VARIANT_BOOL bValue);
```

get_AllChangesAreVirtual

Description

If returns VARIANT_TRUE, the packed exe stores all changes of the file system and the registry in virtual environment.

Syntax

C++

```
HRESULT get_AllChangesAreVirtual([out, retval] VARIANT_BOOL* pbValue);
```

put_SetIcon

Description

If *bValue* is VARIANT_TRUE, the packed exe gets the icon specified by [put_IconPath\(\)](#).

Syntax

C++

```
HRESULT put_SetIcon([in] VARIANT_BOOL bValue);
```


get_SetIcon

Description

If returns VARIANT_TRUE, the packed exe gets the icon specified by [put_IconPath\(\)](#).

Syntax

C++

```
HRESULT get_SetIcon([out, retval] VARIANT_BOOL* pbValue);
```

put_IconPath

Description

Specifies the icon of the packed exe.

Syntax

C++

```
HRESULT put_IconPath([in] BSTR Value);
```

get_IconPath

Description

Returns the icon of the packed exe.

Syntax

C++

```
HRESULT get_IconPath([out, retval] BSTR* pValue);
```

put_EnableSplashScreen

Description

If *bEnable* is VARIANT_TRUE, the packed exe shows a splash screen at startup.

Syntax

C++

```
HRESULT put_EnableSplashScreen([in] VARIANT_BOOL bEnable);
```

get_EnableSplashScreen

Description

If returns VARIANT_TRUE, the packed exe shows a splash screen at startup.

Syntax

C++

```
HRESULT get_EnableSplashScreen([out, retval] VARIANT_BOOL* pbEnabled);
```

put_SplashScreenPath

Description

Specifies the path of the image for splash screen.

Syntax

C++

```
HRESULT put_SplashScreenPath([in] BSTR bstrValue);
```

get_SplashScreenPath

Description

Returns the path of the image for splash screen.

Syntax

C++

```
HRESULT get_SplashScreenPath([out, retval] BSTR* pbstrValue);
```

THIS IS A CONTRACT. YOU SHOULD CAREFULLY READ ALL THE TERMS AND CONDITIONS BEFORE INSTALLING THIS SOFTWARE. BY INSTALLING THE SOFTWARE, YOU ACCEPT ALL THE TERMS AND CONDITIONS OF THIS LICENSE.

This License Agreement is a legal Agreement between you, an individual or organization, and Softanics.

Subject

The subject of the present Agreement is "BoxedApp Packer API" hereinafter mentioned as Software including the whole delivery package, namely the software itself and the covering documentation.

Accepting or rejecting the Agreement

Any usage of this product including installation of the Software or any of its parts on a computer, loading the product into operating memory (RAM) or permanent storage on a computer's disk or other storage medium, as well as any other type of usage means that you accept all the terms and conditions of the present Agreement. If you do not agree with any statement of this License, you should promptly terminate usage of the product and delete all the files referred to it, as its components, so as the results of its work, from your computer. You should also return all the existing mediums containing the Software to the place where you obtained it.

Copyright

This Software is owned by Softanics and is protected by international copyright treaties. Any changes to the Software and its components, any additions to it, including the case of running and executing on your computer any software not built-in into your operating system and affecting the Software's operation or changing its results, as well as storage and distribution of such a changed or augmented Software, are strictly forbidden.

Usage

The Software is licensed by the present Agreement to be used for any commercial and other proper purposes. Applications built using our components are royalties free, but if you need to create a development tool (as DCU, DLL, OCX etc..) integrating this functionality you need to obtain a special license. You acknowledge that the Software in source code form remains a confidential trade secret of Softanics or its suppliers and therefore you agree not to modify the Software or attempt to decipher, decompile, disassemble or reverse engineer the Software. In order to obtain the source codes you should turn to Softanics so as to purchase and register the Software's version including source codes.

Time restriction

This product has no limits in terms of its usage.

Licensed copies number restriction

Single developer license is assigned for installation and execution on one computer assembled as a solitary system units and by one user only. If you have an upgrade to this Software, it constitutes a single product together with the product that you upgraded, and may not be used to increase the total number of licensed installations of the Software. Group programming projects making use of this software must purchase a copy of the software for each member of the group or purchase a Site Developer License. Site Developer License is assigned for installation and execution on unlimited numbers of computers by any user inside one building of organization located in one city. Any possible delivery of the product or its parts as well as registration codes or access codes you received with purchasing and/or registering this Software to any individual or organization is strictly prohibited.

Distribution restrictions

Except as provided in this License Agreement, you may not transfer, sell, rent, lease, lend, copy, modify, emulate, clone, decompile, translate, sublicense, time-share or electronically transmit or receive the Software or any of its parts or to distribute it in any other way.

Registration codes and access codes

Registration codes (e.g., serial number) or access codes (e.g. archive file password) you received with purchasing and/or registering this Software are not the subject to the present Agreement. You may store any number of copies of such codes in any form if only you are a registered user and if you obtained them in the registration process or purchasing the Software from Softanics or authorized persons, individuals or organizations. Any distribution or delivery of the registration and access codes to any third party as well as any form of publication is prohibited. The copyright of all registration and access codes remains the property of Softanics which reserves the right to withhold or withdraw authorization of use of all registration codes issued to any user if there is reasonable evidence to indicate that the user is involved in a breach of the terms of this License Agreement.

Modifications to the Software

Softanics reserves the right to change the executable code and to remove, add or change the Software's functions.

Upgrade policy

A registered user can receive software updates free of charge within 1 year period that begins on the purchasing date. A registered user can buy the right to get updates for the next 1 year period for a discounted price (75% of the regular price) during grace period that starts on the

purchasing date and ends on the purchasing date + 2 months.

Warranty and guarantee

This documentation and the source library are provided "as is" without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and/or suitability for a particular purpose. The user assumes the entire risk of any damage caused by this software.

Changing terms and conditions of this Agreement

Any variation to the terms of this Agreement shall only be valid if made and delivered by Sofianics. If you do not agree to any clause of this Agreement you may ask for explanations from Sofianics. However, this does not authorize you to consider this Agreement nullified.