3-Heights™ PDF Producer API

Version 4.5

User Manual

Contact: pdfsupport@pdf-tools.com

Owner: **PDF Tools AG**

Kasernenstrasse 1
8184 Bachenbülach
Switzerland

http://www.pdf-tools.com

Table of Contents

1 Introduction

This document describes:

- How to install the 3-Heights™ PDF and TIFF Producer with the provided command line tool
- How to install the 3-Heights™ PDF and TIFF Producer with the provided installation library
- How to install and use the Office Converter API

1.1 Packages

Software Kit

The 3-Heights™ PDF and TIFF Producer in a software kit with the code name CRED. The MSI-Installer offers the option to install the Producer SDK. This option is required to use the functions described in this manual.

Documentation

There are two manuals: one for the PDF Desktop Producer and one for the PDF Producer API (this manual).

The PDF Desktop Producer Manual is mainly addressed to the user of the products and contains the following topics:

- Installation uninstallation of the product on a desktop computer
- Printer settings
- Configuration of the port monitor
- Usage scenarios and printing application examples

The PDF Producer API is mainly addressed to developers and integrators. It contains the following topics:

- Installation Command Line Tool and deployment scenarios
- Installation API and integration into own installation programs
- The PDF Producer API
- The Office Converter API

2 Installation Command Line Tool

For customers who want to deploy the PDF and TIFF Producer to various target systems in a batch process can use the Installation command line tool *installpdfproducer.exe*.

The command line tool covers various usage scenarios such as

- Installation and uninstallation of the PDF and TIFF Producer
- Installation of PDF and TIFF Producer Drivers on a server only
- Cross platform installation X86 and X64 on the same server
- Add and remove of virtual printers with different setting sets and ports
- Add and remove of ports to a virtual printer
- Create a port pool for a virtual printer on a server
- Configuring port parameters

A typical call would be:

```
C> installpdfproducer.exe -p “Prompt=True” -cp 4 install
```

This call installs the PDF Producer, creates a pool of 4 ports and checks the “Prompt for file name” checkbox.

The executable comes as a 32 and a 64 bit application and uses the installation API *PdfPrnInstAPI.dll* which is described in the next chapter. The SDK contains the sub-directories “X86” and “X64” which contain the programs and DLLs for the corresponding platform. Each executable can only be run on the platform for which it is compiled. Running the 32 bit executable on an X64 platform will cause an error.

The executable requires that the driver DLLs of the own platform are contained in the same directory otherwise the installation will fail. If the cross platform installation requested (through the corresponding option) the drivers in the corresponding sibling directory “..\X86” or “..\X64” must be present as well.

2.1 Command line parameters and error codes

The command line tool supports various parameters to instruct the tool what the users intends to do. There are two types of parameters: options and commands. An option is preceded by a dash whereas the dash is missing for a command. Only one command can be present on a command line.

The following commands are supported

`Install`	`Perform a full installation`
`Uninstall`	`Perform a full uninstallation`
`AddDriver`	`Install the drivers only without printers and ports`
`DeleteDriver`	`Remove the drivers`
`AddPrinter`	`Add a printer with the drivers already installed`
`DeletePrinter`	`Delete a printer object and leave the drivers`

```
AddPort           Add a port to an existing printer
DeletePort        Remove a port from an existing printer
SetPortConfig     Set the configuration information of a port
List              List all PDF and TIFF Producer printers
```

The following options are supported:

```
  -cp portcount   Create a port pool with the specfied number of ports
  -d              Set as default printer
  -nd descr       Set the printer description
  -npr name       Printer name (e.g. 3-Heights(TM) PDF Producer)
  -npt name       Port name (e.g. DocPort0:)
  -p param=value  Port configuration parameters (see below)
  -t              Use TIFF Producer (Default: PDF Producer)
  -x              Install cross platform drivers (x86, x64)
  -v              Verbose mode
```

The port configuration parameters, a string of comma separated key / value pairs:

```
OutputFolder=<path>  The path of the document output folder
Command=<command>    The command to execute after the document creation
Unique=true / false  Make the file names unique
Prompt=true / false  Display the prompt dialog
RemovePrefix=true / false Remove MS Office prefixes in file names
AddTime=true / false Add the current time to the file name
AddUser=true / false Add the current user to the file name
```

The tool returns the following codes:

```
  0: Success
  3: Invalid parameter or switch
  4: Command failed (a detailed error message is printed to stderr)
  5: Some applications have locked at least one of the driver modules
```

3 Installation API

For customers who want to write their own installation program, e.g. in an OEM scenario, can use the installation API. The API is simple to use and support various installation and configuration scenarios.

PdfPrnInstAPI.dll provides a C interface to install and uninstall the 3-Heights™ PDF Producer and TIFF Producer. The declarations of the interface are contained in the file *pdfprninstapi_c.h*. The file *PdfPrnInstAPI.lib* contains the linker stub library.

The interface uses the C standard calling convention und supports the UNCODE character set. The MCBS character set is not supported by this module.

A typical call sequence is to install the PDF Producer, set the port configuration to display the prompt dialog, and set the printer as the default printer:

```
PDFPRN_INSTALL_INFO info;

PdfPrnInstSetDefaultInfo(&info, eTechnologyPDF);

BOOL bOk = PdfPrnInstExecuteCommand(&info, "Prompt=True", 1,

                                    eFlagSetDefaultPrinter);

if (!bOk)

{

    fwprintf(stderr, info.szErrorMessage);

}
```

3.1 PdfPrnInstSetDefaultInfo

```
BOOL PdfPrnInstSetDefaultInfo(struct PDFPRN_INSTALL_INFO* pInfo,
                              TPdfPrnTechnology iTechnology);
```

The function initializes the installation information structure for being used by the PdfPrnInstExecute() function depending on the technology parameter.

The installation information structure contains the name of the printer object etc. The names can be overwritten by the application as required.

Parameters:

- pInfo: The installation information structure
- iTechnology: The driver technology, one of the following constants:
  `eTechnologyPDF`
  `eTechnologyTIFF`

Return value:

- If a parameter is not valid the function returns FALSE. The last error code is set to ERROR_INVALID_PARAMETER in this case.
- Otherwise the function returns TRUE.

3.2 PdfPrnInstExecuteCommand

```
BOOL PdfPrnInstExecuteCommand(struct PDFPRN_INSTALL_INFO* pInfo,
                              TPdfPrnInstCommand iCommand,
                              const WCHAR* szParams,
                              int nPorts,
                              int iFlags);
```

The function performs the installation or uninstallation according to the installation command and based on the information contained in the installation information structure. The installation information defines the technology, the platform, the names of the objects and the paths to the required DLLs. If the information is not correct the function fails.

Parameters:

- pInfo: The installation information structure
- iCommand: the installation command, one of the following constants:

eCommandInstall	Perform a full installation
eCommandUninstall	Perform a full uninstallation
eCommandAddDriver	Install the drivers only without printers and ports
eCommandDeleteDriver	Remove the drivers
eCommandAddPrinter	Add a printer with the drivers already installed
eCommandDeletePrinter	Delete a printer object and leave the drivers
eCommandAddPort	Add a port to an existing printer
eCommandDeletePort	Remove a port from an existing printer
eCommandSetPortConfig	Set the configuration information of a port
eCommandEnumPrinters	Enumerate all Producer printers

- szParams: The port configuration parameters, a string of comma separated key / value pairs:

OutputFolder=<path>	The path of the document output folder
Command=<command>	The command to execute after the document creation
Unique=true / false	Make the file names unique
Prompt=true / false	Display the prompt dialog
RemovePrefix=true / false	Remove MS Office prefixes in file names
AddTime=true / false	Add the current time to the file name
AddUser=true / false	Add the current user to the file name

- nPorts: the number of ports to install
- iFlags: the installation function instruction flags, a combination of the contants:

eFlagCrossPlatform	Install cross platform, e.g. x64 on x86
eFlagSetDefaultPrinter	Set the installed printer to the default printer

Return value:

- If a parameter is not valid the function returns FALSE. The last error code is set to ERROR_INVALID_PARAMETER in this case.
- Otherwise, if the function fails it returns FALSE and sets the last error code. A descriptive error message is contained in the error message field of the installation information structure.
- Otherwise, the function succeeds it returns TRUE

3.3 PdfPrnInstEnumPrinters

```
int PdfPrnInstEnumPrinters(struct PDFPRN_INSTALL_INFO* pInfo,
                           struct PDFPRN_ENUM_RESULT* pResult,
                           size_t nSize,
                           size_t* nNeededSize);
```

The function lists the printer objects with its associated driver, port and print processor names. In order to determine the size of the result variable the function must first be called with a size value of zero. The needed size is then used to allocate the result variable which can be used as a parameter to the second call.

Parameters:

- pInfo: The installation information structure
- pResult: The array of result structures
- nSize: The size of the result variable in bytes
- nNeedeSize: The needed size of the result variable in bytes

Return value:

- If the function fails it returns -1 (minus one)
- If the function succeeds it returns the number of entries in the result array.

3.4 PdfPrnInstEnumProcesses

```
int PdfPrnInstEnumProcesses(struct PDFPRN_INSTALL_INFO* pInfo,
                            WCHAR** szModuleNames,
                            size_t nSize,
                            size_t* nNeededSize);
```

The function lists the module names which lock any of the driver DLLs and must be closed to install or uninstall the requested drivers. In order to determine the size of the result variable the function must first be called with a size value of zero. The needed size is then used to allocate the result variable which can be used as a parameter to the second call.

Parameters:

- pInfo: The installation information structure
- szModuleNames: The array of module name strings
- nSize: The size of the result variable in bytes
- nNeedeSize: The needed size of the result variable in bytes

Return value:

- If the function fails it returns -1 (minus one)
- If the function succeeds it returns the number of entries in the result array.

4 Licensing Interface

The licensing service provider interface is called by the 3-Heights™ PDF and TIFF Producers to check for the validity of a user license. In this context, a user license is a license that is granted to an end user by the OEM.

The checking process also involves an OEM license key. This OEM license key is provided to OEM customers by PDF Tools AG. This OEM key does not change for the individual user licenses, that is, it remains constant. This OEM license key differs from the license key necessary for the use of the Installation Interface.

If not otherwise agreed between the OEM customer and PDF Tools AG, the name of the licensing DLL must be *pdfprnl.dll*. It must reside in the driver directory, where the driver DLLs of the 3-Heights™ PDF and TIFF Producers are installed. This can either be achieved by directly copying the license check DLL to the driver directory or by using the installation API with the name of the licensing DLL as an additional file. Installing the license check DLL this way, makes it become part of the printer driver. It will be downloaded to clients, whenever they install a 3-Heights™ PDF or TIFF Producer as a network printer.

For license types and the procedure to provide the information at runtime, please consult the description of the method *Check()*. For invalid, expired and evaluation licenses, the outcome of the check will be written to the created PDF or TIFF file in form of a license type specific watermark. For invalid and expired licenses, only the watermark and no user data will be written to the file.

The license is checked at least when an application submits its first print job to a PDF or TIFF Producer. After that, the application may cache the settings until it shuts down.

If the user license changes while the application is running, it may be necessary to restart the application for the new license to take effect.

It may also be necessary to adjust the global and per user printer settings after a license change, since e.g. the "Print Test Page" command of the printer property page does not acquire the printer settings through the printer driver.

The Licensing Interface uses the C standard calling convention.

The declaration of the interface is contained in *licensecheck_c.h*.

4.1 Check

`int Check(HLICCHECK hLicCheck)`

This method is called by the 3-Heights(TM) library to get the OEM information for a user license.

The caller sets the field nSeed to a random value before the call. It also copies the context information into the field pointed to by hLicCheck->licContext.pContext, if applicable.

The callee computes and sets a check digest in the field hLicCheck->licHash using the SHA-1 hash function.

This is done in the following way:

- Start a new SHA-1 hash
- Add the field hLicCheck->nSeed to the hash
- Add the predefined OEM license key to the hash
- Add the filled hLicCheck->licInfo field to the hash
- Set the field hLicCheck->licHash to the hashvalue from the hash

Parameters:

- hLicCheck: the license check handle previously allocated by Create().

Return Value:

- A value != 0, if the check could be performed, 0 otherwise.

4.2 Create

`HLICCHECK Create()`

This method must allocate and initialize a new TLicCheck structure. If necessary, this also includes allocating the space for the context information and setting it's size accordingly.

Upon return of the TLicCheck structure, the field nSize must contain a valid size.

Return Value:

- A handle (pointer) to a new and initialized TLicCheck structure.

4.3 Destroy

`void Destroy(HLICCHECK hLicCheck)`

This method destroys a check handle previously allocated by Create(). It also frees any context information, if applicable.

Parameters:

- hLicCheck: The license check handle previously allocated by Create().

4.4 Initialize

`void Initialize()`

The Initialize() method is the first method called, before any other call to the library will be made. It can be used to perform one-time initialization tasks.

4.5 Uninitialize

`void Uninitialize()`

This method un-initializes the license check DLL. This method is the last method called before the library will be unloaded.

It should perform any necessary clean up tasks and free any resources acquired during the lifetime of the library.

5 PDF Producer API

The PDF Producer API is a component which provides programmatic access to various features of the PDF and TIFF producer. The main functions are:

- Create and remove ports
- Get and set the port configuration
- Get and set the printer settings
- Add XMP metadata to a PDF document

There exists a C interface and a COM interface. The C interface provides access to the PDF and the TIFF Producer whereas the COM interface is restricted to the PDF Producer.

The interface is documented in the C header file PdfProducerAPI_c.h. The COM interface methods and properties are contained in the PdfProducerAPI.idl file. Both files are contained in the software distribution kit.

5.1 C Interface

```
PDFPRODUCERAPI int PDFPRODUCERCALL PdfProducerInitialize(const char* szLicenseKey);

/**
 * @brief This function gets the configuration data of a specific port.
 *
 * @param szName The port's name.
 * @param pConfig The configuration structure.
 * @return 0 if succeeded, last error if failed.
 */
PDFPRODUCERAPI int PDFPRODUCERCALL PdfProducerGetPortConfigurationW(const WCHAR*
szName, struct CONFIGURE_PORT_INFOW* pConfig);

/**
 * @brief This function sets the configuration data of a specific port.
 *
 * @param szName The port's name.
 * @param pConfig The configuration structure. If NULL then the configuration is
cleared.
 * @return 0 if succeeded, last error if failed.
 */
PDFPRODUCERAPI int PDFPRODUCERCALL PdfProducerSetPortConfigurationW(const WCHAR*
szName, const struct CONFIGURE_PORT_INFOW* pConfig);

/**
 * @brief This function gets the user settings of a specific printer.
 *
 * @param szPrinter The printer's name.
 * @param pDevMode The device mode.
 * @param nSize The size of the device mode structure.
 * @return 0 if succeeded, last error if failed.
 */
```

```
PDFPRODUCERAPI int PDFPRODUCERCALL PdfProducerGetPrinterSettingsW(const WCHAR*
szPrinter, struct DEVMODEW* pDevMode, size_t nSize);

/**
 * @brief This function sets the user settings of a specific printer.
 *
 * @param szPrinter The printer's name.
 * @param pDevMode The device mode.
 * @param nSize The size of the device mode structure.
 * @return 0 if succeeded, last error if failed.
 */
PDFPRODUCERAPI int PDFPRODUCERCALL PdfProducerSetPrinterSettingsW(const WCHAR*
szPrinter, const struct DEVMODEW* pDevMode, size_t nSize);

///////////////////////////////////////////////////////////////////////////////

typedef struct _TPdfPrnDataHandle* TPdfPrnDataHandle;

/*
 * @brief Create a data object (using data name "XMP").
 *        The data object must be created before a print job is started.
 * @param szUserName The name of the printing user
 * @param szDocumentName The name of the printed document
 * @return The handle to the data object
 */
PDFPRODUCERAPI TPdfPrnDataHandle PDFPRODUCERCALL PdfPrnCreateData(const WCHAR*
szUserName, const WCHAR* szDocumentName);

/*
 * @brief Create a data object given a data name (e.g. "XMP", "Links").
 *        The data object must be created before a print job is started.
 * @param szUserName The name of the printing user
 * @param szDocumentName The name of the printed document
 * @param szDataName The type of the data object (currently only "XMP" is supported)
 * @return The handle to the data object
 */
PDFPRODUCERAPI TPdfPrnDataHandle PDFPRODUCERCALL PdfPrnCreateData2(const WCHAR*
szUserName, const WCHAR* szDocumentName, const WCHAR* szDataName);

/*
 * @brief Destroy a data object.
 *        The data object must be closed after the corresponding print job has
started.
 * @param handle The handle to the data object
 */
PDFPRODUCERAPI void PDFPRODUCERCALL PdfPrnCloseData(TPdfPrnDataHandle handle);

/*
 * @brief Writes data to the object.
 *        All data must be written before the corresponding print job is started.
 * @param handle The handle to the data object
 * @param pData A pointer to the data buffer
 * @param nSize The size of the data buffer
 * @return The number of bytes written; 0 if the function fails
 */
PDFPRODUCERAPI size_t PDFPRODUCERCALL PdfPrnWriteData(TPdfPrnDataHandle handle,
const void* pData, size_t nSize);
```

5.2 COM Interface

```
typedef enum TPDFConformance
{
    ePdfConformancePDF14  = 1,
    ePdfConformancePDF15,
    ePdfConformancePDF16,
    ePdfConformancePDF17,
    ePdfConformancePDFA1B,
    ePdfConformancePDFA2U
} TPDFConformance;

import "oaidl.idl";
import "ocidl.idl";
[
 uuid(1ED16CB4-3849-4c49-9D2E-C6A2ECE8E2B0),
 version(1.0),
 helpstring("3-Heights(TM) Pdf Producer API 2.0")
]
library PdfProducerAPILib
{
 importlib("stdole32.tlb");
 importlib("stdole2.tlb");

    [
     object,
     uuid(A8E6BE77-8FB0-4E67-8118-E43AA8DB3726),
     dual,
     nonextensible,
     helpstring("IPdfPrinter Interface"),
     pointer_default(unique)
    ]
    interface IPdfPrinter : IDispatch
    {
        [propget, id(1), helpstring("The name of a stamp control file (XML).")]
HRESULT bstrStampFileName([out, retval] BSTR* pVal);
        [propput, id(1), helpstring("The name of a stamp control file (XML).")]
HRESULT bstrStampFileName([in] BSTR newVal);
        [propget, id(2), helpstring("Linearize the output file")] HRESULT
Linearize([out, retval] VARIANT_BOOL* pVal);
        [propput, id(2), helpstring("Linearize the output file")] HRESULT
Linearize([in] VARIANT_BOOL newVal);
        [propget, id(3), helpstring("The name of a background stamp control file
(XML).")] HRESULT BGStampFileName([out, retval] BSTR* pVal);
        [propput, id(3), helpstring("The name of a background stamp control file
(XML).")] HRESULT BGStampFileName([in] BSTR newVal);
        [propget, id(4), helpstring("Conformance level of the PDF output file.")]
HRESULT Conformance([out, retval] TPDFConformance* pVal);
        [propput, id(4), helpstring("Conformance level of the PDF output file.")]
HRESULT Conformance([in] TPDFConformance newVal);
    };
    [
     object,
     uuid(52022ABC-4662-4F96-B8D4-AF8D2068D623),
     dual,
     nonextensible,
     helpstring("IPdfPort Interface"),
     pointer_default(unique)
    ]
    interface IPdfPort : IDispatch
    {
        [propget, id(1), helpstring("The output folder")] HRESULT Directory([out,
retval] BSTR* pVal);
        [propput, id(1), helpstring("The output folder")] HRESULT Directory([in]
BSTR newVal);
        [propget, id(2), helpstring("Add a postfix to make file names unique")]
HRESULT MakeFileNamesUnique([out, retval] VARIANT_BOOL* pVal);
        [propput, id(2), helpstring("Add a postfix to make file names unique")]
HRESULT MakeFileNamesUnique([in] VARIANT_BOOL newVal);
        [propget, id(3), helpstring("Remove Microsoft Prefix from file name")]
HRESULT RemovePrefixes([out, retval] VARIANT_BOOL* pVal);
        [propput, id(3), helpstring("Remove Microsoft Prefix from file name")]
HRESULT RemovePrefixes([in] VARIANT_BOOL newVal);
        [propget, id(4), helpstring("Prompt for the output file name")] HRESULT
PromptFileName([out, retval] VARIANT_BOOL* pVal);
        [propput, id(4), helpstring("Prompt for the output file name")] HRESULT
PromptFileName([in] VARIANT_BOOL newVal);
        [propget, id(5), helpstring("Execute the command line")] HRESULT
ExecuteCommand([out, retval] VARIANT_BOOL* pVal);
        [propput, id(5), helpstring("Execute the command line")] HRESULT
ExecuteCommand([in] VARIANT_BOOL newVal);
        [propget, id(6), helpstring("Add the date and time to the output file
name")] HRESULT AddTimeStamp([out, retval] VARIANT_BOOL* pVal);
        [propput, id(6), helpstring("Add the date and time to the output file
name")] HRESULT AddTimeStamp([in] VARIANT_BOOL newVal);
        [propget, id(7), helpstring("Add the user name to the output file name")]
HRESULT AddUser([out, retval] VARIANT_BOOL* pVal);
        [propput, id(7), helpstring("Add the user name to the output file name")]
HRESULT AddUser([in] VARIANT_BOOL newVal);
        [propget, id(8), helpstring("The command line to be executed")] HRESULT
Command([out, retval] BSTR* pVal);
        [propput, id(8), helpstring("The command line to be executed")] HRESULT
Command([in] BSTR newVal);
        [propget, id(9), helpstring("The output file name")] HRESULT FileName([out,
retval] BSTR* pVal);
        [propput, id(9), helpstring("The output file name")] HRESULT FileName([in]
BSTR newVal);
        [id(10), helpstring("Clears the user configuration")] HRESULT
Clear([out,retval] VARIANT_BOOL* pDone);
    };

  [
   object,
   uuid(DE8861CC-0392-49ef-91D8-770841D23839),
   dual,
   helpstring("IPdfProducer Interface"),
   pointer_default(unique)
  ]
```

```
 interface IPdfProducer : IDispatch
 {
        [propget, id(1), helpstring("The port")] HRESULT Port([in] BSTR
bstrPortName, [out, retval] IPdfPort** pVal);
        [propget, id(2), helpstring("Get the printer object")] HRESULT Printer([in]
BSTR bstrPrinter, [out, retval] IPdfPrinter** pVal);
    };

 [
  uuid(C125C4B2-2023-40BA-8253-55AEE71CA964),
  helpstring("PdfPrinter Class")
 ]
 coclass PdfPrinter
 {
  [default] interface IPdfPrinter;
 };
 [
  uuid(577BDE2C-C551-431E-AD2D-5DDA9A4290A5),
  helpstring("PdfPort Class")
 ]
 coclass PdfPort
 {
  [default] interface IPdfPort;
 };
 [
  uuid(3B0318E9-0FBB-43fc-9595-8B8DDA9D5727),
  helpstring("PdfProducer Class")
 ]
 coclass PdfProducer
 {
  [default] interface IPdfProducer;
 };
};
```

5.3 Sample programs

Here's a list of sample programs that show various capabilities of the PDF Producer API.

C\AddXMPMetadata	Add a XMP Metadata packet to the PDF file
C\GDISample	Create a PDF file using Windows GDI calls
C\PipeSample	Create a PDF file in memory
C\PlayEMFSample	Convert an EMF file to a PDF file
VB\PrinterSample	Use the VB intrinsic printing to create a PDF file
VB.NET\PrintDocument	Use the VB.NET language to create a PDF file
VBA*.xls	Use a VBA script in an Excel to create a PDF file
VBS*.vbs	Use the VB Script language to perform various tasks

The samples can be found in the software distribution kit.

6 Office Converter API

The Officer Converter API is used to programmatically convert office documents to PDF documents. It can be accessed via the following interfaces: COM, C, .NET. This means it can be used from Visual Basic, Visual Basic Script, C#, C, Delphi, etc. It requires the native applications (e.g. MS Office 2010) to be installed locally.

6.1 Interfaces

The native library of the Office Converter API is "OfficeConvertAPI.dll", it is used by all interfaces.

COM Interface

Before you can use the Office Converter API component in your COM application program you have to register the component using the regsvr32.exe program that is provided with the Windows operating system.

The registration command requires administrator privileges and looks like this:

```
C:\> regsvr32 OfficeConvertAPI.dll
```

Upon registering a dialog box is displayed that informs whether it was successful or not. In order to register silently, e.g. for deployment, you can use the switch /s.

C Interface

- The header file *officeconverterapi_c.h* needs to be included in the C/C++ program.
- The Object File Library lib\OfficeConverterAPI.lib must to be linked to the project.
- OfficeConverterAPI.dll must be included in the environment variable PATH or, if using MS Visual Studio, in the directory for executable files.

.NET Interface

The Office Converter API does not provide a pure .NET solution. Instead, it consists of a .NET assembly, which is added to the project and a native DLL which is called by the .NET assembly. This has to be accounted for when installing and deploying the tool.

The .NET assembly (OfficeConvertNET.dll) is to be added as references to the project. It is required at compilation time.

OfficeConverterAPI.dll is not a .NET assembly, but a native DLL. It is not to be added as a reference in the project.

The native DLL OfficeConverterAPI.dll is called by the .NET assembly OfficeConvertNET.dll.

OfficeConverterAPI.dll must be found at execution time by the Windows operating system. There are various approaches to ensure this. Below two are listed:

- OfficeConverterAPI.dll is copied to a directory that is on the environment variable "PATH". e.g. it is either copied to
  - a new dedicated directory, which is then added to the "PATH", or
  - an existing directory which already is on the "PATH", such as %winroot%\System32\ (e.g. C:\Windows\System32\).

  This approach is usually used for development.
- OfficeConverterAPI.dll is copied to the directory where the compiled executable resides.

  This approach is usually used for deployment.

6.2 Sample programs

There is sample code available in different programming languages. Please refer to: convert.cs and convert.vbs which are contained in the software distribution kit.