**WinPrint 2.0 Design Specification**

Kindel Systems

Copyright © 1994 Charles E. Kindel, Jr.

# Product Description

WinPrint is a Windows application that facilitates the easy printing of standard text files[[1]](#footnote-1). WinPrint 2.0 is a completely rewritten version of the original WinPrint 1.x. Unless otherwise noted, *WinPrint* refers to WinPrint 2.0 throughout this documentation.

WinPrint 1.x was very sucessful because it clearly addressed a problem many people had: “How do I print this text file quickly and easily?” Windows itself provides basically two ways users can do this.

The first is to load the file into Notepad and print it from there. This has several limitations:

* Minimal formatting abilities. Headers and footers are available, but are very limited. In particular there is no choice of fonts and no way to set margins.
* Limited file length support. Under Win16 Notepad could only print files less than about 30K in size. Win32 fixes this.
* No advanced user interface such as drag-n-drop, multiple file selection, etc...

The other method is to use Windows Write. Using Write gets around the file length and formatting problems, but it simply takes too much effort to use.

Most users have an advanced word processor such as Word for Windows or WordPerfect. Trying to use a word processor to print standard ASCII text files is really no different than trying to use Windows Write.

The WinPrint design provides a mechanism whereby other types of files besides text can be supported. The WinPrint Printing Engine supports the notion of File Type Drivers. These DLLs know how to process an input stream and output that stream to a output medium that Printing Engine provides. The initial release of WinPrint will support several standard text file type drivers (DOS Text, ANSI Text, Source Code). File type drivers for bitmaps, metafiles, and other file types will be made available as resources permit.

WinPrint will be sold and distributed as shareware. The shareware license will be per-user. Because WinPrint can be used as part of other applications as well as standalone, this must be taken into account. The basic premise is that every user of WinPrint must have a valid license. A valid license allows that user to use the product standalone or as part of some custom solution. If someone wants to distribute a solution that uses WinPrint as a component, the creator of the solution must purchase a license for each user.[[2]](#footnote-2)

## The Customer

WinPrint is designed to do primarily one thing, and to do it well: Print text files from Windows. First and foremost, WinPrint is a text file printing engine that can be utilized by other applications. WinPrint includes a highly intuitive user interface that allows it be a solid stand-alone application as well. For these reasons, WinPrint is appealing to a wide variety of customers.

### End Users

WinPrint makes printing in Windows easier. This alone is a reason why end users will find WinPrint appealing.

Shareware documentation and README files.

Output from legacy programs (e.g. DOS applications)

### Software Developers

Shareware documentation and README files.

Output from legacy programs (e.g. DOS applications)

Source code

### ISVs, Solutions Providers, and Corporate Developers

WinPrint is programmable (via OLE Automation) which allows it to be utilized as part of larger solutions and products.

Output from legacy programs (e.g. DOS applications, mainframe data, etc...)

[[ does it makes sense to spend the cycles developing a C/C++ development kit? ]]

[[ is the programmability available at additional cost?? ]]

## Platforms Supported

### Windows 3.1

It is expected that WinPrint 2.0 will be finished long before Chicago ships, thus it is essential that it run natively on Windows 3.1. It is possible that the binary that runs on Windows 3.1 is Win32s.

### Windows NT

There will be both an ANSI and Unicode version of WinPrint for Windows NT. The ANSI version will be developed first

### Chicago

The ANSI Windows NT version of WinPrint will run on Chicago. A future version of WinPrint will be written specifically for Chicago.

# Features

## Engine Features

### Headers/Footers

### Multiple Pages Up

Concept of logical pages. Any number of rows and columns.

### Duplex Printing

Print even # pages first, then prompt to change paper, then print odd # pages.

### Font Selection

User can pick fonts for each header/footer component as well as for the main output[[3]](#footnote-3)

### Margin Settings

### OLE Automation

### OLE Control

Much like a timer control the WinPrint engine will not be visible at run-time on a form. Each instance of the engine as a control will save with it one ‘configuration’.

### Multiple File Printing

The ability to print multiple files in one print job. With or without a logical page break between files.

### Print Preview

The engine will provide a mechanism whereby print jobs can be displayed in a print preview window.

## File Type Drivers

WinPrint removes the code that actually parses and creates output for a file from the code that provides physical page formatting. The code that understands a given file type is contained within a File Type Driver.

The File Type Driver obtains either a filename and prints that file, or it calls into the Engine to read from a stream. This allows File Type Drivers to potentially get input from some specialized input medium (e.g. an external device) as well as from files.

### Multiple File Type Drivers Per DLL

File Type Drivers are contained within DLLs[[4]](#footnote-4). These File Type Driver DLLs may support more than one File Type Driver.

## User Interface Features

### OLE 2.0 Drag-n-Drop

Multiple drop targets (by configuration/file type)

### Printing of OLE 2.0 objects

Via drag-n-drop & clipboard

### Print Preview

When files are dropped onto either the main frame or a drop-target that is not iconized the print job is put into the queue as ‘held’. The user can ‘start’, ‘preview’, or ‘kill’ a held job.

### Configuration Management

### Queue Managment

Non redundant with the Windows Print Manager. Shows files queued up by WinPrint internally.

# Design

This section details the architecture of WinPrint 2.0.

WinPrint can use GUIDs in the range 0002A5xx-0000-0000-C000-000000000046.

## Printing Engine

The WinPrint Printing Engine is a DLL. It exposes it’s functionality both through exported “C” functions and OLE Automation.

Object design...

It’s clear that FTDs are COM objects; they implement IDispatch, IPersistFile, and IWinPrintFileType. The IDispatch is used to set/query properties, not to actually do anything.

The engine itself is a COM object. It supports IDispatch. The Prog. Interface lets clients enumerate configurations and set any global ‘WinPrint’ state (which is?).

Question?: If client uses Engine to get a configuration, what does it get? An IDispatch of an object that implements the persistent state of a configuration, or something that can be acted upon to start a job? What is a Job?

A Job is a combination of a *Configuration and a Data Source.* It’s lifetime covers the actual print operation. Jobs can be started, stopped, deleted, and queued. The configuration that a Job uses must be a snapshot of an existing configuration or changes to a configuration chould alter a job mid stream.

Who are clients of File Type Drivers? Jobs and configurations. A configuration may create an instance of a FTD in order to display it’s setup dialog and save/restore the FTDs state. A Job will create an instance of a FTD when it wants to print...that is when the Job needs to start sending meaningful data to the printer it will instantiate the appropriate FTD and ask it to paint.

Should FTDs be Controls? No. But for simplicity they can be implemented using the CDK classes.

A Configuration of a *File Type Driver*

### Development Plan

Currently the Engine is written in C and it needs to be converted to C++ (as appropriate). The current implementation will be left alone until there is a suitable Primary User Interface component (prototype). Once the entire product is prototyped using the current Win16 engine the engine will be restructured into an OLE Control.

The OLE Control Development Kit will be used to generate a framework control. The control will be an invisible control (at design time does it need anything visible? Or can this be accomplished through property sheets).

The transition from C code to C++/MFC code will cause some internal changes to the Engine. However, this should be minimized as much as possible. This may mean simply wrapping current implementation even if the current implemention is not optimal.

#### Schedule

Five days

### OLE Control

What is saved as part of the control’s state? A single configuration? What does the programmability interface look like given this (when used standalone as an automation server the engine exports functions that provide configuration management, right?)

### Printing Core

### File Type Drivers

File Type Drivers are COM objects. They must support a set of interfaces including a newly defined interface IWinPrintFileType.

A configuration is a “container” for a file type driver object. Thus just like embedded objects, the configuration provides a place for the file type driver to place it’s persistent information (a stream). Since it is useful for a configuration to have multiple file types associated with it, each with it’s own saved state there is the need for configurations to name the stream.

Should it be a storage instead? Are the circumstances where a file type driver would require sub-storages or sub-streams?

Thus a file type driver is responsible for maintaining both persistent state (configuration) information and run-time information (created fonts, pointers back to the engine etc...).

How does a FTD get it’s data? It QI’s on IWinPrintConfig for IStream and reads from the stream. Thus the engine must always somehow convert the source into an IStream. For files this is accomplished by calling

StgCreateDocFile(szFile, STGM\_READWRITE | STGM\_TRANSACTED | STGM\_CONVERT, 0, &pStg) ;

Should the file be opened STGM\_SHARE\_EXCLUSIVE?

The “CONTENTS” stream of this storage is then used.

FTDs support the following interfaces:

#### IPersistStream

Allows file client of type driver to cause the FTD to load and save it’s configuration from a stream. For example the engine may implement code like this:

pFTD->QueryInterface( IID\_IPersistStream, &pPS ) ;

pPS->Load( pStm ) ;

or

pS->Save( pStm ) ;

#### IWinPrintFileType

IWinPrintFileType

{

QueryInterface();

AddRef();

Release();

GetDialogTemplate( LPDLGBOXHEADER FAR \*ppDlgTemplate ) ;

HandleDialogMessage( HWND hwnd, UINT uiMsg, WPARAM wParam, LPARAM lParam, LRESULT FAR \*lResult ) ;

StartDoc( IWinPrintJob \*pJob ) ;

Verify( IStream\* pStream ) ;

OutputPage() ;

EndDoc() ;

}

##### GetDialogTemplate

Returns a pointer to a dialog box template that is used for the file type driver’s configuration.

##### HandleDialogMessage

Called by the setup dialog code to give the file type driver an opportunity to respond to dialog messages.

##### StartDoc

Initializes the file type driver. The IWinPrintJob interface (documented in the Engine section) provides the file type driver with a mechanism for accessing job settings as well as the data itself. The filetype driver should QueryInterface on this pointer for IStream and use the returned pointer to access the data during the OutputPage() calls.

The implementation should save a pointer to the IWinPrintJob (calling AddRef of course). It will be useful to call back on this interface during the OutputPage method.

This is a good place to create fonts and other graphics objects.

##### Verify

This function is called whenever a client wants to determine whether the file type driver can print the data in the passed stream. The File Type Driver should do exaustive tests on the stream to determine if it contains suitable data. It should return S\_FALSE if the data is foriegn, S\_OK if everything is cool.

##### OutputPage

Output page is called by the engine for each logical page in a print job. The device context that the file type driver paints on is retreived through the IWinPrintJob::GetDeviceContext member. The device context will have the viewport origin set so that the FTD can start drawing at 0, 0 and the right thing will happen. The size of the logical page and other information can be obtained by the FTD by calling members on the IWinPrintJob interface.

##### EndDoc

If StartDoc did an AddRef on IWinPrintJob, a Release should be performed here. Any other clean up required may also be performed.

#### IDispatch

Filetype Drivers must implement IDispatch so that their persistent state information can be set via OLE Automation (accessed through the WinPrintEngine.Configuration.FileTypeDriver method)

### Configuration Mangement

### Setup/Config User Interface

### Shareware Licensing

WinPrint 2.0 will support shareware registration licensing. The underlying mechanism will be significantly different than that found in WinPrint 1.x. Part of the reasoning for this comes from the fact that WinPrint 2.0 is implemented in a DLL as a COM object.

The infrastructure for the licensing support will come from the OLE Controls licensing model. There are several ways WinPrint can be started, and in all cases licensing needs to be enforced.

#### Standalone Product

When run standalone WinPrint will determine at startup whether a valid license is available. It will do so by looking in the configuration (Registry or INI file) for a RegistrationNumber key.

#### OLE Control at Design Time

#### OLE Control at Run Time

This model is setup such that clients that want to use a control use IClassFactory2 rather than IClassFactory. IClassFactory2 is defined as follows:

DECLARE\_INTERFACE\_(IClassFactory2, IClassFactory)

{

// IUnknown methods

STDMETHOD(QueryInterface)(THIS\_ REFIID riid, LPVOID FAR\* ppvObj) PURE;

STDMETHOD\_(ULONG,AddRef)(THIS) PURE;

STDMETHOD\_(ULONG,Release)(THIS) PURE;

// IClassFactory methods

STDMETHOD(CreateInstance)(THIS\_ LPUNKNOWN pUnkOuter, REFIID riid,

LPVOID FAR\* ppvObject) PURE;

STDMETHOD(LockServer)(THIS\_ BOOL fLock) PURE;

// IClassFactory2 methods

STDMETHOD(GetLicInfo)(THIS\_ LPLICINFO pLicInfo) PURE;

STDMETHOD(RequestLicKey)(THIS\_ DWORD dwResrved, BSTR FAR\* pbstrKey) PURE;

STDMETHOD(CreateInstanceLic)(THIS\_ LPUNKNOWN pUnkOuter,

LPUNKNOWN pUnkReserved, REFIID riid, BSTR bstrKey,

LPVOID FAR\* ppvObject) PURE;

};

IClassFactory2::RequestLicKey is called by a client of a control to get a license key it can later pass into ::CreateInstanceLic. ::GetLicInfo

### External Programmability

#### C/C++

All C/C++ programmability will be through COM interfaces.

#### OLE Automation (Programmability Interface)

##### WinPrint Custom Control

dispinterface IWinPrintHFSectionDisp

{

properties:

BSTR Text;

IFontDisp\* Font ;

};

dispinterface IWinPrintHeaderDisp

{

properties:

long Flags ; //review

long Height ; //Is this minimum height? review

IWinPrintHFSectionDisp\* Left ;

IWinPrintHFSectionDisp\* Center ;

IWinPrintHFSectionDisp\* Right ;

long BorderFlags ; // review

long ShadeFlags ; // review

};

dispinterface IWinPrintControlDisp

{

properties:

[id(n)]IWinPrintConfigDisp\* Configuration ; // Read only

BSTR ConfigName ;

BSTR FileTypeDriverName ;

IDispatch\* FileTypeDriver ;

BSTR Printer ; // Device (HP LaserJet III) (Win16 only)

BSTR PrinterDevice ; // Driver (HPLJIIID.DRV)

BSTR PrinterOutput ; // Output (LPT1) (Win16 only)

short Orientation ;

short PaperSize ;

short PaperLength ; // review

short PaperWidth ; // review

short Scale ; // review

short Copies ;

short DefaultSource ; // review

short PrintQuality ; // review

short Color ; // review

short Duplex ; // review

long TopMargin ;

long BottomMargin ;

long LeftMargin ;

long RightMargin ;

long BorderFlags ; // review how to expose this

long ShadeFlags ; // review how to expose this

IWinPrintHeaderDisp\* Header ;

IWinPrintHeaderDisp\* Footer ;

int Rows ;

int Columns ;

int HorzSeparation ;

int VertSeparation ;

methods:

[id(n)] bool LoadConfiguration( BSTR ConfigName );

[id(n)] bool SaveConfigurationAs( BSTR ConfigName ) ;

[id(n), propput] void Source( BSTR SourceFileName );

[id(n), propput] void Source( IDataObject\* pSource );

[id(n)] bool Print( BSTR SourceFileName );

[id(n)] bool Print( IDataObject\* pSource );

};

##### WinPrintEngine Object

CLSID = {} = “WinPrint 2.0”

ProgID = “WinPrint20.Engine.1”

###### WinPrintEngine.Configurations

Collection of Configuration objects

###### WinPrintEngine.[Constants]

##### Configuration Object

The Configuration Object abstracts the configuration data. It knows how to save/restore the configuration data to/from persistant store (via IPersistStream) and exposes the configuration data via Automation properties and methods. The Configuration Object has *no* verbs associated with it.

CLSID = {} = “WinPrint 2.0 Configuration”

ProgID = “WinPrint20.Configuration.1”

###### Configuration.Name

###### Configuration.FileTypeDriverName

###### Configuration.FileTypeDriver

Returns an IDispatch of the file type driver.

###### Configuration.PrinterName

On Win16 this is the “Driver”, on Win32 this is the “rinter Name”.

###### Configuration.Orientation

###### Configuration.PaperSize

###### Configuration.PaperLength (no UI)

###### Configuration.PaperWidth (no UI)

###### Configuration.Scale (no UI)

###### Configuration.Copies

###### Configuration.DefaultSource (no UI)

###### Configuration.PrintQuality (no UI)

###### Configuration.Color (no UI)

###### Configuration.Duplex (no UI)

###### Configuration.Collate (no UI/Win32)

###### Configuration.Margins

Defines the printable area (nothing will ever be printed outside of this area). See the “Margins” object below

## Configuration.BorderFlags

[[Think about this further]]

###### Configuration.ShadeFlags

[[Think about this further]]

###### Configuration.Header

See “HeaderFooter” object below

###### Configuration.Footer

See “HeaderFooter” object below

##### Font Object

See OLE Control spec and mimic.

###### HeaderFooter Object

###### HeaderFooter.Flags

[[ Needs thought ]]

## HeaderFooter.BorderFlags

[[Think about this further]]

###### HeaderFooter.ShadeFlags

[[Think about this further]]

###### HeaderFooter.Height

###### HeaderFooter.LeftSection

See “HFSection” object.

###### HeaderFooter.CenterSection

See “HFSection” object.

###### HeaderFooter.RightSection

See “HFSection” object.

##### HFSection Object

HFSection.Text

HFSection.Font

##### Margins Object

Defines the printable area (nothing will ever be printed outside of this area).

###### Margins.Top

###### Margins.Bottom

###### Margins.Left

###### Margins.Right

## Primary User Interface

The primary user interface will be an MFC application. The executable will be “WINPRT2.EXE” and the icon label will read “WinPrint 2.0”. It will be initially generated by ClassWizard 1.5 and thus will be an MFC 2.5 applicaiton. OLE Automation will be enabled, but no objects will be implemented in the first version.

The Primary User Interface will be referred to has “PUI” below.

The purpose of the PUI is to provide an easy to use interface where users can manage configuration, drop files and objects, and view/modify the queue. The PUI will provide only limited facilities for actually selecting files for printing (a FileOpen dialog box that supports multiple selection).

The PUI will support a full comand line syntax very similar to that supported in WinPrint 1.x.

How can full command line support work if WINPRT2.EXE is single instance? One idea is to actually provide command line support through a ‘mini’ exe. Or, just come up with an IPC mechanism. This means WINPRT2.EXE must not use MFC250.DLL on Win16 and must not be large model. On Win32 it can use MFC300.DLL.

The PUI will be a single instance application.

During Phase 2 the PUI will be ported to Win32 using Visual C++ 2.0 and MFC 3.0. No Chicago features will be implemented in the first release, although it is important that Chicago design issues be takien into account during development.

Print Preview should be implemented (via MFC).

### Main Frame

SDI with several panes, a menu, a status bar, and a toolbar. The tool bar and status bar can be turned on or off via user interface (View menu). SDI because that fits into the Chicago model better.

The main frame will be a WinFile and OLE 2.0 drag-drop target (superceded by child windows). WinPrint will attempt to decern the configuration to use as follows

If it’s a file drop

Look at extension and if there is an association check the print key

If the print key exists check if it’s WInPrint

If it’s WinPrint then

grab the configuration name and use it.

Else

Use ShellExecute() to print (unless “alwasy prompt” is checked)

Else

Display a list of configurations.

Else

#### Layout

The two panes will be separated by a splitter. The splitter position will be saved and restored.

##### Configuration pane

Owner draw listbox (CColumnListBox) with columns for configuration name, printer, orientation, # columns, # rows, drop target, file type.

Given a configuration is selected, ‘modify’ and ‘delete’ toolbar buttons are available. In addiion an ‘add’ command is available.

The list box contains a check box for drop target support or not. If checked a drop target icon will appear for this config. If not checked no icon will be available. Dragging a configuration from the list onto the desktop will cause a droptarget icon to be placed at the drop point. Upon startup drop target icon locations will be restored.

The config list should be single select..

Each item in the list in the configuration pane will be an OLE 2.0 and WinFile drop target. Thus the user can drop files directly on an item in the list and that configuration will be used.

##### Queue pane

Owner draw listbox (CColumnListBox) with columns for filename, path, configuration, time/date?, pages. When files are dropped onto either the main frame or a drop-target that is not iconized the print job is put into the queue as ‘held’. The user can ‘start’, ‘preview’, or ‘kill’ a held job.

Thus this pane must have buttons for ‘hold’, ‘start’, ‘kill’, and ‘preview’. These buttons will be on the toolbar and will only be enabled if an item in the queue list is selected.

The queue list should be single select.

Reording of jobs can be accomplished via drag&drop (low priority feature).

#### Menu

#### Toolbar

### Drop Targets

There are 2 kinds of Drop Targets.

#### Printer Drop Target

Dropping on this type of icon always results in printing to the represented printer, regardless of the configuration that may be choosen (automatically or via prompting).

Double clicking on a printer drop target pops up the configuration for that printer.

#### Configuration Drop Target

Dropping on this type forces a particular configuration to be used (and thus a specific printer...unless the configuration uses the ‘default printer’). This is useful for being able to drop files with different extensions and always have them print using a given config.

The file type driver is responsible for determining whether it can actually print the file (via the IWinPrintFileType::VerifyInput function).

Double clicking on a configuration drop target pops up the setup dialog on that configuration.

1. A standard text file is a file that does not contain any formatting beyond line layout. For example a word processor file (\*.DOC) is not considered a standard text file. [↑](#footnote-ref-1)
2. The license policy must have a provision for this so that people who want to do this can get a discount. [↑](#footnote-ref-2)
3. This is actually a file type driver feature. Each filetype driver can choose how text is printed. [↑](#footnote-ref-3)
4. The standard file type drivers that ship with WinPrint are actually contained within the Engine DLL. [↑](#footnote-ref-4)