Biblioteki łączone dynamicznie *(DLL)*

- Łączenie statyczne - *linker*
  dynamiczne - *load-time, run-time*
- Typy dynamicznego łączenia
  - *load-time* - wywołania funkcji z DLL'a jakby były lokalne *(Import Library)*
  - *run-time* - pobranie adresu funkcji z DLL'a w trakcie wykonania programu
- Zalety
  - jedna kopia dla wielu procesów - oszczędność pamięci
  - modyfikacja funkcji w DLL'u nie powoduje potrzeby przekompilowania całej aplikacji
  - możliwość zmiany DLL'a na nowszy
  - niezależność od języka programowania

Tworzenie DLL

- Zawartość plików źródłowych:
  - eksportowane funkcje i dane
  - wewnętrzne funkcje i dane
  - opcjonalna funkcja wejścia *(entry-point function)*
- Eksportowanie funkcji
  - VC++: `__declspec(dllexport)` albo plik .def

```
LIBRARY     "SSCD"
EXPORTS
            Run @1
            ApplyDefaultSettings @2
```

- *Import Library*
  - zawiera informacje potrzebne linkerowi do zrealizowania odwołań do eksportowanych z DLL'a funkcji
  - system może wyszukiwać DLL'a i funkcje w czasie wykonania

Funkcja wejścia *(entry-point function)*

- Opcjonalna funkcja wywoływana automatycznie
- Definicja

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL,
                    DWORD fdwReason,
                    LPVOID lpReserved)
```

  - fdwReason:
    DLL_PROCESS_ATTACH, DLL_THREAD_ATTACH,
    DLL_THREAD_DETACH, DLL_PROCESS_DETACH
- Zwracana wartość
  - FALSE dla DLL_PROCESS_ATTACH powoduje, że **`LoadLibrary()`** zwraca NULL
- Momenty wywołania:
  - załadowanie DLL'a - **`LoadLibrary()`**, **`LoadLibraryEx()`**
  - zwolnienie DLL'a - **`FreeLibrary()`**

Łączenie DLL

- *Load-Time*
  - w momencie uruchomienia programu system szuka DLL'a, z którym program jest związany
    - nie znajduje – przerwanie procesu, wyświetlenie komunikatu o błędzie
    - znajduje – wołana jest funkcja wejścia
- *Run-Time*
  - wczytanie DLL'a: **`LoadLibrary()`**, **`LoadLibraryEx()`**
    - jeśli DLL nie może być zlokalizowany, zwraca NULL
    - jeśli DLL został zlokalizowany, wołana jest funkcja wejścia
  - pobranie adresu funkcji: **`GetProcAddress()`**
  - informacje: **`GetModuleHandle()`**,**`GetModuleFileName()`**
  - zwolnienie: **`FreeLibrary()`**, **`FreeLibraryAndExitThread()`**

Dane DLL

- Globalne zmienne w źródłach DLL'a są globalne dla każdego procesu
  - każdy proces ma własne instancje globalnych i statycznych zmiennych DLL'a
- Dynamiczna alokacja pamięci
  - **`GlobalAlloc(), LocalAlloc(), HeapAlloc(), VirtualAlloc()`**
  - pamięć jest alokowana w wirtualnej przestrzeni adresowej wołającego procesu - jest dostępna tylko dla wątków z tego procesu
  - *file mapping* pozwala na dzielenie zaalokowanej pamięci
- *Thread Local Storage* (TLS)
  - różne wartości dla różnych wątków jednego procesu

Biblioteki łaczone dynamicznie w .NET

- Koniec z *DLL Hell*
- *Assembly* – wielokrotnego użytku, z numeracją wersji, z opisem własnej zawartości
  - .exe, .dll, wieloplikowe
- Łączenie z *assembly*
  - dodać referencję
  - użyć klasy **`Assembly`** podczas wykonania
- *Strong name* – unikalny identyfikator *assembly*
- Numeracja wersji
  - dwa lub więcej *assembly* o tej samej nazwie lecz innych numerach wersji mogą współistnieć w systemie
- *Global Assembly Cache*

Wykonanie kodu niezarządzanego

- Czynności wykonywane podczas próby wywołania funkcji z biblioteki DLL (*Platform Invoke*):
  1. znaleźć bibliotekę DLL zawierającą funkcję
  2. załadować DLL do pamięci
  3. znaleźć adres funkcji w pamięci, położyć wartości parametrów na stosie
  4. przekazać sterowanie do funkcji
- Użycie funkcji z biblioteki DLL:
  1. znaleźć nazwy bibliotek DLL zawierających funkcje
  2. utworzyć klasę, która będzie zawierała prototypy funkcji
  3. utworzyć prototypy funkcji
  4. wywoływać funkcje jako statyczne metody

Identyfikacja funkcji w bibliotece DLL

- Identyfikacja funkcji:
  - nazwa biblioteki DLL
  - nazwa lub numer funkcji
- Biblioteki Win32 API:
  - GDI32.dll – *Graphics Device Interface* (GDI)
  - Kernel32.dll – zarządzanie pamięcią i zasobami
  - User32.dll – zarządzanie oknami (komunikaty, menu etc.)
- Wersje funkcji dla kodowania ANSI i Unicode, np.:
  - MessageBoxA - ANSI
  - MessageBoxW - Unicode

Przykład użycia DLLImportAttribute

podczas wywołania sposób kodowania i długość tekstu zostaną automatycznie rozpoznane

```
using System.Runtime.InteropServices;

public class Win32 {
  [DllImport("user32.dll", CharSet=CharSet.Auto)]
  public static extern int MessageBox(int hWnd,
        String text, String caption, uint type);
}

public class HelloWorld {
  public static void Main() {
    Win32.MessageBox(0, "Hello World",
                     "Platform Invoke Sample", 0);
  }
}
```

Schowek

- Zbiór funkcji i komunikatów umożliwiających wymianę danych pomiędzy aplikacjami
  - wszystkie aplikacje mogą korzystać ze schowka
- Formaty danych w schowku
  - obiekt pamięci w schowku może być dowolnego typu formatu
  - każdy format jest identyfikowany stała liczbą całkowitą dodatnią
- Właściciel schowka – okno powiązane z danymi trzymanymi w schowku (do czasu zamknięcia schowka lub wyczyszczenia go przez inne okno)

Kopiowanie do schowka

1. Otworzyć schowek - `OpenClipboard()`
2. Wyczyścić zawartość schowka - `EmptyClipboard()`
3. Ustawić zawartości - `SetClipboardData()`
   - zawartość – `HANDLE`
   - przydzielać pamięć przy pomocy `GlobalAlloc()` z `GMEM_MOVEABLE`
   - nigdy nie zwalniać pamięci wstawionej do schowka
4. Zamknąć schowek - `CloseClipboard()`

Wklejanie ze schowka

1. Otworzyć schowek - `OpenClipboard()`
2. Sprawdzić dostępne formaty - `IsClipboardFormatAvailable()`, `EnumClipboardFormat()`, `GetPriorityClipboardFormat()`, `CountClipboardFormats()`
3. Pobrać zawartość - `GetClipboardData()`
4. Zamknąć schowek - `CloseClipboard()`

Formaty danych schowka

- Wybrane standardowe formaty danych dla schowka:
  - CF_BITMAP, CF_DIB, CF_TIFF - obrazy
  - CF_ENHMETAFILE, CF_METAFILEPICT - metapliki
  - CF_WAVE, CF_RIFF - dźwięki
  - CF_TEXT, CF_UNICODETEXT - teksty
- Możliwość rejestrowania formatów przez aplikacje:
  - `RegisterClipboardFormat()`
  - `GetClipboardFormatName()`
- Prywatne formaty danych używane przez jedną aplikację
  - zakres wartości CF_PRIVATEFIRST - CF_PRIVATELAST

Użycie schowka w Windows Forms

- Klasa `Clipboard` ze statycznymi metodami:
  - `ContainsData(), GetData(), SetData()`
  - `ContainsAudio(), GetAudioStream(), SetAudio()`
  - `ContainsFileDropList(), GetFileDropList(), SetFileDropList()`
  - `ContainsImage(), GetImage(), SetImage()`
  - `ContainsText(), GetText(), SetText()`

Użycie schowka w WPF

- Podczas kopiowania lub wycinania WPF umieszcza w schowku dane w postaci tekstowej i w formacie XAML
- Podczas wklejania WPF użyje formatu XAML gdy dane pochodzą ze źródła o co najmniej takim samym poziomie zaufania
- Klasa **`System.Windows.Clipboard`** zawiera statyczne składowe umożliwiające ustawienie, pobranie i sprawdzenie zawartości schowka
- Dla wymuszenia wykorzystania określonego formatu tekstu można wykorzystać **`System.Windows.DataFormats`**:
  - **`Text, UnicodeText, Rtf, Html, CommaSeparatedValue, Xaml`**

Rejestr

- Struktura
  - drzewo kluczy i wartości
- Predefiniowane klucze
  - HKEY_CLASSES_ROOT - klasy COM i *shell'a*
  - HKEY_CURRENT_CONFIG - profil sprzętowy
  - HKEY_CURRENT_USER - ustawienia dla zalogowanego użytkownika
  - HKEY_LOCAL_MACHINE - informacje sprzętowe i systemowe dla komputera
  - HKEY_USERS - konfiguracja dla nowych użytkowników
- Ustawienia aplikacji
  - HKEY_CURRENT_USER\Software\<firma>\<app>\<ver>
  - HKEY_LOCAL_MACHINE\Software\<firma>\<app>\<ver>

Użycie rejestru

- Klucze
  - utworzenie z otwarciem - **`RegCreateKeyEx()`**
  - otwarcie - **`RegOpenKeyEx()`**
  - zamknięcie - **`RegCloseKey()`**
  - usunięcie - **`RegDeleteKey()`**
  - wyliczenie podkluczy - **`RegEnumKeyEx()`**
  - pobranie informacji - **`RegQueryInfoKey()`**
- Wartości
  - ustawienie - **`RegSetValueEx()`**
  - pobranie - **`RegQueryValueEx()`**, **`RegQueryMultipleValues()`**
  - usunięcie - **`RegDeleteValue()`**
  - wyliczenie wartości z klucza - **`RegEnumValue()`**

Typy danych w rejestrze

- Binarne
  - REG_BINARY
- Liczby
  - REG_DWORD - 32 bity
  - REG_QWORD - 64 bity
- Ciągi znaków
  - REG_SZ - z '\0' na końcu
  - REG_MULTI_SZ - z dwoma '\0' na końcu
  - REG_EXPAND - ze zmiennymi środowiskowymi (np. %PATH%) **`ExpandEnvironmentStrings()`**

Klasa `Registry` z .NET

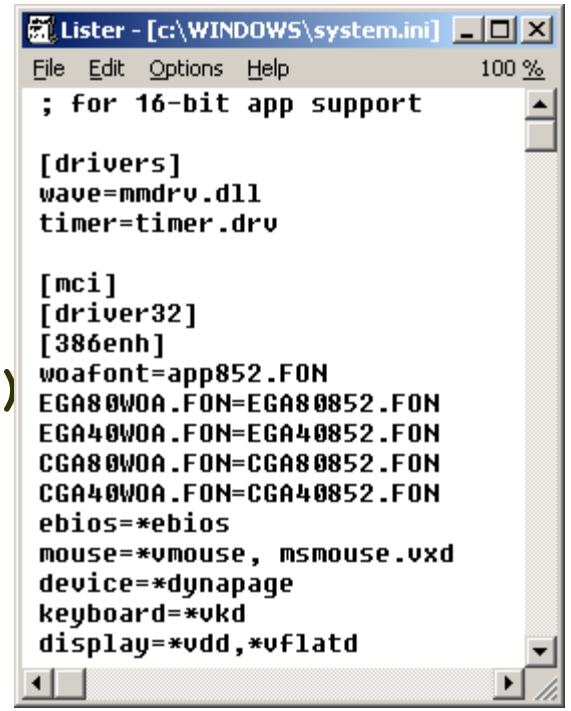
- Klasa **`Microsoft.Win32.Registry`**
  - statyczne pola definiujące klucze główne (zwracają obiekty klasy **`RegistryKey`**):
    - **`ClassesRoot`** - HKEY_CLASSES_ROOT
    - **`CurrentConfig`** - HKEY_CURRENT_CONFIG
    - **`CurrentUser`** - HKEY_CURRENT_USER
    - **`DynData`** - HKEY_DYN_DATA
    - **`LocalMachine`** - HKEY_LOCAL_MACHINE
    - **`PerformanceData`** - HKEY_PERFORMANCE_DATA
    - **`Users`** - HKEY_USERS

Klasa `RegistryKey` w .NET

- Metody i właściwości klasy `RegistryKey` :
  - `OpenSubKey()`
  - `CreateSubKey()`
  - `SubKeyCount, GetSubKeyNames()`
  - `ValueCount, GetValueNames()`
  - `GetValue(), SetValue()`
  - `DeleteSubKey(), DeleteSubKeyTree(), DeleteValue()`

Pliki .ini

- Wygodny format pliku tekstowego
- Win.ini
  - `GetProfileSection()`,
  - `GetProfileInt()`, `GetProfileString()`
  - `WriteProfileSection()`
  - `WriteProfileString()`
- <app>.ini
  - `GetPrivateProfileSection()`
  - `GetPrivateProfileInt()`, `GetPrivateProfileString()`, `GetPrivateProfileStruct()`
  - `WritePrivateProfileSection()`
  - `WritePrivateProfileString()`, `WritePrivateProfileStruct()`



Drukowanie

- Drukowanie w Windows jest niezależne od urządzenia
- Schemat drukowania:
  - pobrać kontekst urządzenia drukarki - **`PrintDlgEx()`**, **`PrintDlg()`**
  - sprawdzić możliwości drukarki - **`GetDeviceCaps()`**
  - rozpocząć drukowanie - **`StartDoc()`**
  - rozpocząć drukowanie strony - **`StartPage()`**
  - drukować stronę - funkcje GDI
  - zakończyć drukowanie strony - **`EndPage()`**
  - zakończyć drukowanie - **`EndDoc()`**

Interfejs drukowania

- GDI
- Sterownik drukarki
- *Print spooler*
  - plik wykonywalny zarządzający procesami drukowania
  - tworzy zadania drukowania w postaci dokumentów zawierających metapliki
- *Print processor*
  - DLL konwertuje metapliki do wywołań funkcji sterownika
  - wykorzystuje *graphics engine*
- *Port monitor*
  - DLL przesyłający dane do drukarki

Drukowanie w Windows Forms

- Schemat drukowania:
  - utworzyć `System.Drawing.Printing.PrintDocument` i określić drukarkę

```
PrintDocument pd = new PrintDocument();
pd.PrinterSettings.PrinterName = printerName;
```

  - ustawić właściwości drukarki (`PrinterSettings`) i strony (`PageSettings`)
  - ustawić metodę obsługi zdarzenia wydruku strony

```
pd.PrintPage += new PrintPageEventHandler(
                                    myPrintPage);
```

  - wydrukować dokument

```
pd.Print();
```

Zdarzenia drukowania (WinForms)

- **`BeginPrint`**
  - po wywołaniu metody **`Print()`**, przed wydrukiem pierwszej strony
  - najlepsze miejsce do inicjalizacji
  - **`PrintEventArgs: Cancel`**
- **`PrintPage`**
  - przed wydrukiem strony
  - **`PrintPageEventArgs: Cancel, Graphics, HasMorePages, MarginBounds, PageBounds, PageSettings`**
- **`EndPrint`**
  - po wydruku ostatniej strony
  - najlepsze miejsce dla zwalniania zasobów

Using PrintPage Event (WinForms)

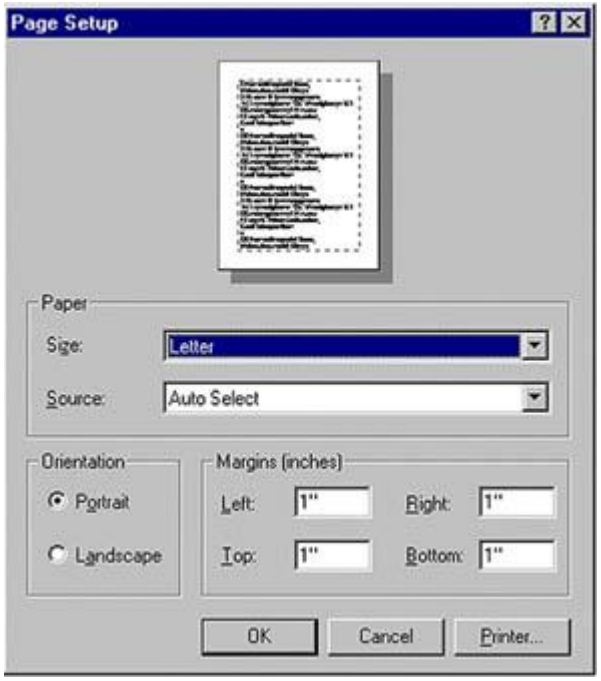
```
private void myPrintPage(object sender,
    PrintPageEventArgs ev)
{
    float linesPerPage = 0;
    float yPos = 0;
    int count = 0;
    float leftMargin = ev.MarginBounds.Left;
    float topMargin = ev.MarginBounds.Top;
    string line = null;
    linesPerPage = ev.MarginBounds.Height /
        printFont.GetHeight(ev.Graphics);

    while (count < linesPerPage &&
        ((line = streamToPrint.ReadLine()) != null))
    {
        yPos = topMargin + (count *
            printFont.GetHeight(ev.Graphics));
        ev.Graphics.DrawString(line, printFont,
            Brushes.Black, leftMargin, yPos,
            new StringFormat());
        count++;
    }
    ev.HasMorePages = (line != null);
}
```

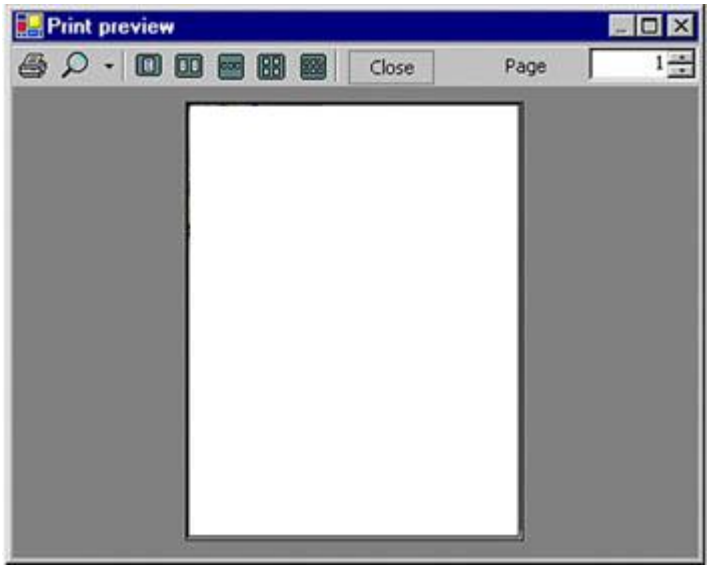
Klasa `PrinterSettings` (WinForms)

- Dostęp:
  - `PrintDocument.PrinterSettings`,
  - `PageSettings.PrinterSettings`
- Właściwości:
  - `InstalledPrinters` (wszystkie drukarki w systemie)
  - `PrinterName, IsValid`
  - `PrinterResolutions, PaperSources, PaperSizes`
  - `CanDuplex, Duplex, SupportsColor,`
  - `MaximumCopies`
  - `PrintToFile, PrintFileName`
  - `Copies, FromPage, ToPage, PrintRange`
  - `DefaultPageSettings`

Okna dialogowe drukowania (WinForms)

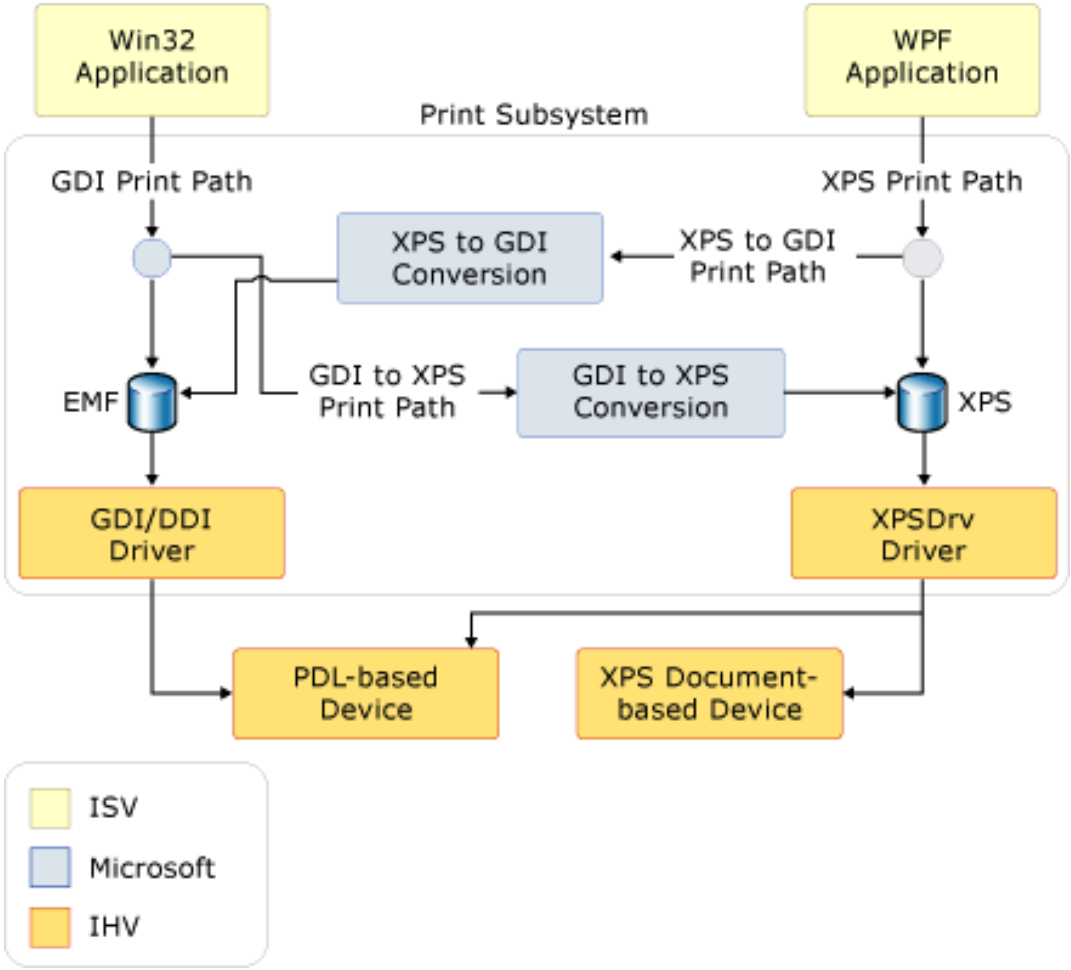
- PrintDialog
- PageSetupDialog
- PrintPreviewDialog





Zarządzanie wydrukiem w Windows Vista

- Windows Vista i .NET Framework wprowadziły nową ścieżkę wydruku (alternatywną wobec klasycznego wydruku z wykorzystaniem GDI)



Ścieżka wydruku XPS

- Ścieżka wydruku XPS wykorzystuje format XPS podczas całego procesu wydruku
  - zastępuje wszystkie standardowe elementy wydruku: język prezentacji dokumentu (np. RTF), format *print spooler-a* (np. WMF) i język opisu strony (np. Postscript lub PCL)
- Ścieżka wydruku XPS pozwala na:
  - wykorzystanie zaawansowanych profili kolorów
  - ulepszenie wydajności wydruku zarówno dla aplikacji .NET Framework jak i natywnych aplikacji Win32
  - wykorzystanie formatu XPS
- Przestrzenie nazw **`System.Windows.Xps`** i **`System.Printing`** zawierają wiele klas pozwalających na programistyczną obsługę funkcjonalności wydruku

System.Windows.Controls.PrintDialog

```
public void InvokePrint(string xpsFilePath)
{
    // Create the print dialog object and set options
    PrintDialog pDialog = new PrintDialog();
    pDialog.PageRangeSelection =
        PageRangeSelection.AllPages;
    pDialog.UserPageRangeEnabled = true;

    // Display the dialog
    if (pDialog.ShowDialog() == true) {
        XpsDocument xpsDocument = new XpsDocument(
            xpsFilePath, FileAccess.ReadWrite);
        FixedDocumentSequence fds =
            xpsDocument.GetFixedDocumentSequence();
        pDialog.PrintDocument(fds.DocumentPaginator,
            "Test print job");
    }
}
```

Wydruk z kodu C# (System.Printing)

```
private void PrintAllXpsFiles(string path)
{
  // Create print server and print queue.
  LocalPrintServer localPrintServer = new LocalPrintServer();
  PrintQueue defaultPrintQueue =
      LocalPrintServer.GetDefaultPrintQueue();

  DirectoryInfo dir = new DirectoryInfo(path);
  foreach (FileInfo f in dir.GetFiles("*.xps")) {
    string filePath = path + "\\" + f.Name;
    try {
      // Print the Xps file while providing
      // XPS validation and progress notifications.
      PrintSystemJobInfo xpsPrintJob =
        defaultPrintQueue.AddJob(f.Name, filePath, false);
    }
    catch (PrintJobException e) {
    }
  }
}
```

Aplikacja MDI

- 3 rodzaje okien:
  - *frame* - główne okno aplikacji
  - *MDI client* - okno klasy "`MDICLIENT`" - tło dla okien *child*
  - *child* - okno robocze
- Cechy
  - okna robocze wewnątrz jednego głównego okna aplikacji
  - menu
    - standardowe pozycje: Tile, Cascade, Arrange Icons, Close All, lista okien roboczych
    - `WM_MDISETMENU`
  - akceleratory
    - `TranslateMDISysAccel()`
    - ALT+-, CTRL+F4, CTRL+F6

Okna robocze aplikacji MDI

- 3 sposoby tworzenia okna roboczego aplikacji MDI:
  - **`CreateMDIWindow()`**
  - wysłanie **`WM_MDICREATE`** do okna *MDI client*
  - **`CreateWindowEx()`** z **`WS_EX_MDICHILD`**
- Aktywacja
  - **`WM_MDIACTIVATE`**
  - **`WM_MDIGETACTIVE`**
- Rozmiar
  - **`WM_MDIMAXIMIZE`**, **`WM_MDIRESTORE`**
- Ułożenie
  - **`WM_MDICASCADE`**, **`WM_MDITILE`**, **`WM_MDIICONARRANGE`**
- Niszczenie
  - **`WM_MDIDESTROY`**

Schemat tworzenia aplikacji MDI

1. Rejestracja klasy okna głównego
2. Rejestracja klasy okien roboczych
3. Utworzenie okna głównego
4. Utworzenie okna *MDI client*
   - `CreateWindow()` z `"MDICLIENT"`
5. TranslateMDISysAccel() w pętli komunikatów
6. Procedura głównego okna:
   - używać `DefFrameProc()` zamiast `DefWindowProc()`
7. Procedura okien roboczych:
   - używać `DefMDIChildProc()` zamiast `DefWindowProc()`
8. Tworzenie okien roboczych

Aplikacja MDI w Windows Forms

- Główne okno aplikacji - *MDI parent form*
  - zwykły formularz z wartością **`true`** dla właściwości **`IsMDIContainer`**
- Okna robocze - *MDI child forms*
  - zwykły formularz z referencją do głównego okna ustawioną jako wartość właściwości **`MdiParent`**

```
protected void MDIChildNew_Click(
    object sender, System.EventArgs e)
{
    Form2 newMDIChild = new Form2();
    newMDIChild.MdiParent = this;
    newMDIChild.Show();
}
```

Aplikacja MDI w Windows Forms c.d.

- Operacje na głównym oknie aplikacji:
  - aktywne okno robocze: **`ActiveMdiChild`**
  - standardowe ułożenie okien roboczych: **`LayoutMdi()`**
- Ograniczenia okien roboczych:
  - **`Opacity`** nie działa
  - **`CenterToParent()`** nie działa
- Standardowe menu okien aplikacji MDI:
  - używając kontrolki **`MenuStrip`** [2.0]: **`MenuStrip.MdiWindowListItem`**
  - używając kontrolki **`MainMenu`** [1.x]: **`MenuItem.MdiList`**

Aplikacja MDI w WPF

- WPF nie ma wbudowanych żadnych mechanizmów obsługi MDI
- Użycie MDI nie jest zalecane dla nowych aplikacji
  - zaleca się wykorzystanie kontrolki **`TabControl`**
- Istnieją zewnętrzne biblioteki obsługujące MDI w WPF (wraz z zaawansowanymi możliwościami "dokowania" okienek)
  - komercyjne, np. Actipro Docking & MDI lub SandDock
  - *open source*, np. AvalonDock