Автоматическая печать документов

П. А. Куратов

Институт системного анализа Российской академии наук,
Россия, 117312 Москва, пр. 60-летия Октября, 9

В работе рассмотрены возможности печати документов из службы Windows с использованием компонент Microsoft Office, а также системных и свободно распространяемых компонент.

Введение

Что может быть проще, чем напечатать документ под Windows? Щелкаем по нему в Проводнике правой кнопкой «мыши» и в контекстном меню нажимаем «Печать». Однако все становится гораздо сложней, если требуется проводить печать в автоматическом режиме, да еще из службы Windows.

Возьмем документы TIFF, HTML и DOC и попробуем напечатать их при помощи функции API ShellExecute. Если служба запускается без взаимодействия с рабочим столом, то при стандартных настройках Windows ни один из файлов не будет напечатан. Если взаимодействие с рабочим столом разрешить, то DOC будет сразу отправлен на печать, а в двух других случаях понадобится инициировать печать вручную. Таким образом, универсальный способ печати через ShellExecute не подходит для автоматической печати.

Рассмотрим далее возможности печати распространенных типов документов: текстовые файлы TXT, изображения, документы Microsoft Office, HTML-документы, PDF-документы и электронные письма EML. Предполагаем, что на компьютере установлен пакет Microsoft Office. Остальные использованные компоненты входят в состав Windows или являются свободно распространяемыми.

Примеры кода написаны на C#, причем для печати изображений требуется NetFramework версии 3.0 и выше, а остальные будут работать с любой версией NetFramework.

1. Файлы изображений

Обычно используемые для работы с изображениями классы GDI+ из пространства System.Drawing не рекомендуются Microsoft для использования в службах Windows. Поэтому для печати изображений из службы Windows мы воспользуемся пространством System.Printing, имеющимся в NetFramework версии 3.0 и выше.

Классы этого пространства поддерживают работу с документами в формате XPS. Поэтому сначала создадим XPS-документ:

```
FixedDocument MakeDoc(string fileToPrint)
{
   FixedDocument fd = new FixedDocument();
   fd.DocumentPaginator.PageSize =
   new System.Windows.Size(PrintableWidth,
PrintableHeight);
   PageContent
pageContent = CreatePageContent(fileToPrint);
   fd.Pages.Add(pageContent);
   return fd;
}

private PageContent CreatePageContent(string
fileToPrint)
{
   BitmapImage bitmapImage = new BitmapImage(
   new Uri(fileToPrint, UriKind.RelativeOrAbsolute));
   Image image = new Image();
   image.Source = bitmapImage;
   Canvas.SetTop(image, 0);
   Canvas.SetLeft(image, 0);

   FixedPage fixedPage = new FixedPage();
   fixedPage.Background = Brushes.BlanchedAlmond;
   fixedPage.Children.Add(image);

   // Масштабируем и добавляем поля

   const double inch = 96;
   double xMargin = 0.25 * inch;
   double yMargin = 0.25 * inch;
   Double xScale =
      (PrintableWidth – xMargin * 2) / PrintableWidth;
   Double yScale =
      (PrintableHeight – yMargin * 2) /
      PrintableHeight;
   fixedPage.RenderTransform = new
```

```
        MatrixTransform(xScale, 0, 0, yScale,
        xMargin, yMargin);

      PageContent pageContent = new PageContent();
        ((IAddChild)pageContent).AddChild(fixedPage);
        return pageContent;
}
```

В приведенном фрагменте PrintableWidth и PrintableHeight — размеры области печати принтера. Их можно получить следующим образом:

```
PrintServer ps = (ServerName = = "")?
   new LocalPrintServer(): new PrintServer(ServerName);
PrintQueue Spooler = new PrintQueue(ps, PrinterName);
PrintTicket pt = Spooler.UserPrintTicket;
PrintableWidth = pt.PageMediaSize.Width.Value;
PrintableHeight = pt.PageMediaSize.Height.Value;
```

Здесь PrinterName и ServerName — имена принтера и компьютера, где он установлен. Теперь мы можем напечатать документ:

```
FixedDocument fd = MakeDoc(fileToPrint);
XpsDocumentWriter xdwPrint =
   PrintQueue.CreateXpsDocumentWriter(Spooler);
xdwPrint.Write(fd);
```

Метод Write класса XpsDocumentWriter обеспечивает синхронную печать. После возвращения управления из него файл fileToPrint может быть удален.

2. Документы Microsoft Office

Чтобы не разбираться с каждым типом файлов по отдельности, воспользуемся свободно распространяемым компонентом dsoFramer [1]. Объект dsoFramer представляет собой надстройку над объектами Microsoft Office. В зависимости от типа файла он обеспечивает загрузку соответствующего объекта, показ и печать документа. Использование объекта предполагает наличие на локальном компьютере установленного пакета Microsoft Office. Нас будут интересовать возможности печати в автоматическом режиме, предоставляемые объектом dsoFramer.

Объект dsoFramer имеет два метода, обеспечивающие его печать:

```
_PrintOutOld(VARIANT PromptToSelectPrinter);
```

и

```
PrintOut(VARIANT PromptUser, VARIANT PrinterName,
   VARIANT Copies, VARIANT FromPage, VARIANT ToPage,
   System.Reflection.Missing.Value);
```

Первый из них использует метод DoOleCommand соответствующего объекта Microsoft Office и выполняет печать на принтер по умолчанию. К сожалению, реализация метода _PrintOutOld выполнена не лучшим образом. При значении параметра false было бы логично ожидать выполнения печати в автоматическом режиме. В действительности в метод DoOleCommand передается значение «по умолчанию» и результат зависит от загруженного объекта. Например, Word печатает автоматически, а Excel выводит диалог печати. К счастью, исходные коды dsoFramer доступны [2] и можно изменить поведение метода нужным образом. Для этого надо заменить первую команду в методе _PrintOutOld, реализация которого находится в файле dsoprint.cpp, на следующую:

```
DWORD
dwOption = BOOL_FROM_VARIANT(PromptToSelectPrinter,
FALSE)
   ? OLECMDEXECOPT_PROMPTUSER:
OLECMDEXECOPT_DONTPROMPTUSER;
```

Теперь мы можем выполнить печать документа Microsoft Office следующим образом:

```
AxDSOFramer.AxFramerControl.axFramerControl =
   new AxDSOFramer.AxFramerControl();
((System.ComponentModel.ISupportInitialize)(axFramerCon
trol)).
   BeginInit();
axFramerControl.Open(fileToPrint);
axFramerControl._PrintOutOld(false);
```

Печать будет производиться в автоматическом режиме без вывода диалога на принтер, определенный в системе как принтер «по умолчанию».

Если требуется производить печать на заданный принтер, можно воспользоваться методом PrintOut. Этот метод использует интерфейс IPrint соответствующего объекта. К сожалению, реализация метода также не обошлась без ошибок. В функции FGetPrinterSettings из файла utilities.cpp в строках

```
*ppwszDevice = DsoCopyString(pinfo->pDriverName);
*ppwszDevice = DsoConvertToLPWSTR(pinfo->pDriverName);
```

следует заменить pDriverName на pPrinterName. Теперь печать можно выполнить следующим образом:

```
axFramerControl.PrintOut(
   false, PrinterName, copies, fromPage, toPage,
   System.Reflection.Missing.Value);
```

3. Текстовые документы в формате TXT

Поскольку документы в формате TXT поддерживаются Microsoft Word, для их печати можно воспользоваться объектом dsoFramer, однако при открытии файла надо явно задать тип объекта Microsoft Word:

```
axFramerControl.Open(m_fileToPrint, true,
"Word.Document",
   System.Reflection.Missing.Value,
   System.Reflection.Missing.Value);
axFramerControl.PrintOut(
   false, PrinterName, copies, fromPage, toPage,
   System.Reflection.Missing.Value);
```

4. HTML-документы

В NetFramefork имеется элемент управления WebBrowser, предназначенный для работы с HTML документами. WebBrowser должен иметь родительское окно, например, он может быть расположен на форме.

Печать из WebBrowser производится следующим образом. Сначала открываем файл:

```
webBrowser.Url = new Uri(printFile);
```

Затем производим печать в обработчике события DocumentCompleted:

```
private void PrintWebDocument(object sender,
   WebBrowserDocumentCompletedEventArgs e)
{
   WebBrowser web = (WebBrowser)sender;
      if
(web.ReadyState = = WebBrowserReadyState.Complete)
      {
         web.Print();
         return;
   }
}
```

Описанным способом можно напечатать только один файл на каждое внешнее событие (например, на нажатие кнопки пользовательского интерфейса). Если организовать цикл по набору файлов, то событие DocumentCompleted возникает только на последнем из них. Для обработки нескольких документов можно запускать таймер по событию DocumentCompleted, а в обработчике события по таймеру загружать следующий Url.

К сожалению, описанный метод печати HTML-документов не работает в службе Windows: в этом случае событие DocumentCompleted не возникает. Поэтому для печати HTML-документов воспользуемся Microsoft Word.

Попытка использовать dsoFramer так же, как для документов Microsoft Office, оканчивается неудачей: даже при явном указании в методе Open объекта Word.Document файл открывается с использованием объекта Web-Browser. Чтобы использовать Word, загрузим его непосредственно:

```
Type objClassType = Type.GetTypeFromProgID
   ("Word.Application");
object wordApp_Late = Activator.CreateInstance
   (objClassType);
```

Задаем принтер:

```
wordApp_Late.GetType().InvokeMember("ActivePrinter",
   System.Reflection.BindingFlags.SetProperty,
   null, wordApp_Late, new object[] {PrinterName});
```

Загружаем файл:

```
object nullObj = System.Reflection.Missing.Value;
object wordDocuments =
   wordApp_Late.GetType().InvokeMember("Documents",
   System.Reflection.BindingFlags.GetProperty, null,
   wordApp_Late, null);
wordDocuments.GetType().InvokeMember("Open",
   System.Reflection.BindingFlags.InvokeMethod,
   null, wordDocuments, new object[]
      {fileToPrint, false, true, nullObj,
      nullObj, nullObj, nullObj, nullObj,
      nullObj, nullObj, nullObj, nullObj,
      nullObj, nullObj, nullObj, nullObj });
```

Печатаем:

```
wordApp_Late.GetType().InvokeMember("PrintOut",
   System.Reflection.BindingFlags.InvokeMethod,
   null, wordApp_Late, new object[]
   {false, nullObj, nullObj,
   nullObj, nullObj, nullObj, nullObj,
   nullObj, nullObj, nullObj, nullObj,
   nullObj, nullObj, nullObj, nullObj,
   nullObj, nullObj, nullObj, nullObj });
```

Закрываем Word:

```
wordApp_Late.GetType().InvokeMember("Quit",
   System.Reflection.BindingFlags.InvokeMethod,
   null, wordApp_Late, new object[] {false, nullObj,
      nullObj});
```

5. PDF-документы

Вместе с Acrobat Reader поставляется ActiveX — управляющий элемент [3]. Однако он не работает в службе Windows, поэтому запустим приложение Acrobat Reader. Это можно сделать двумя способами. Первый более простой — использовать свойство UseShellExecute объекта Process:

```
ProcessStartInfo si = new ProcessStartInfo();
si.CreateNoWindow = true;
si.UseShellExecute = true;
si.WindowStyle = ProcessWindowStyle.Hidden;
si.ErrorDialog = true;
si.FileName = fileToPrint;
si.Verb = "print";
Process printProc = Process.Start(si);
```

Однако таким образом печать всегда производится на принтер «по умолчанию»: команду «printto», позволяющую печатать на заданный принтер, Acrobat Reader не воспринимает.

Второй способ позволяет производить печать на заданный принтер. Для этого нужно вызвать Acrobat Reader с помощью командной строки:

```
AcroRd32.exe /t "C:\test.pdf" "\\servername\printername"
```

Поскольку расположение файла AcroRd32.exe в системе не фиксировано, сначала мы найдем его, а затем запустим с требуемыми параметрами. Соответствующий код на C# будет выглядеть следующим образом:

```
[DllImport("Shell32.dll",
EntryPoint = "FindExecutableA")]
internal static extern UInt32 FindExecutable(
   string fname, string dir, IntPtr exeName);

IntPtr pExeFile = Marshal.AllocCoTaskMem(256);
UInt32 fr = FindExecutable(
   fileToPrint, Environment.CurrentDirectory, pExeFile);
if (fr > 32)
{
   ProcessStartInfo si = new ProcessStartInfo();
   si.CreateNoWindow = true;
   si.UseShellExecute = false;
   si.WindowStyle = ProcessWindowStyle.Hidden;
   si.ErrorDialog = true;
   si.RedirectStandardError = true;
   si.Arguments =
      String.Format("/t \"{0}\" \"{1}\"",
```

```
            fileToPrint, printer);
        si.FileName = Marshal.PtrToStringAnsi(pExeFile);
        Process printProc = Process.Start(si);
    }
    Marshal.FreeCoTaskMem(pExeFile);
```

По окончании печати процесс нужно закрыть, а файл при необходимости — удалить. Чтобы определить момент окончания печати, проведем мониторинг очереди печати. Для этого в настройках принтера должно быть указано, что используется очередь печати (это настройка по умолчанию). Строго говоря, в процессе печати задание проходит несколько состояний: «постановка в очередь», «печать», «удаление». Однако состояния могут меняться очень быстро, так что даже подписка на уведомления об изменении состояния очереди не гарантирует возможности отследить все этапы прохождения задания. Поэтому мы отследим только сам факт появления и исчезновения задания из очереди печати.

```
    public bool CheckPrinted(string fileToPrint)
    {
        // Будем ожидать появление задания в очереди
        // не более 10 сек

        int jobID = 0;
        DateTime endTime = DateTime.Now.Add
            (new TimeSpan(0, 0, 10));
        do
        {
            pq.Refresh();
            PrintJobInfoCollection jobs =
            pq.GetPrintJobInfoCollection();
            foreach (PrintSystemJobInfo item in jobs)
{
            // Для Arcobat Reader имя задания совпадает
            // с именем файла, который печатается

            if (item.Name == fileToPrint)
            {
                jobID = item.JobIdentifier;
                break;
            }
        }
}
while (jobID == 0 &&
            DateTime.Compare(DateTime.Now, endTime) < = 0);

if (jobID == 0)
```

```
         return false; // задание в очереди не появилось
// Ожидаем, пока задание исчезнет из очереди
// или примет статус окончания, успешного или нет

while(true)
{
       Thread.Sleep(100);
       pq.Refresh();
       try
       {
          PrintSystemJobInfo job = pq.GetJob(jobID);
          if (job.JobStatus = = PrintJobStatus.Printed ||
             job.JobStatus = = PrintJobStatus.Completed)
          {
             return true;
          }
          else
          if (job.JobStatus = = PrintJobStatus.Error)
          {
             return false;
          }
       }
       catch (Exception)
       {
          return true; // Задания с номером jobID
             // в очереди нет
       }
       }
}
```

6. Электронные письма (eml-файлы)

Файлы eml открываются и печатаются программой Outlook Express. Однако она не поддерживает автоматизацию, поэтому для автоматической печати eml-файлов воспользуемся программой Microsoft Word. Использование ее аналогично тому, как было описано выше для HTML-файлов. Печать производится асинхронно, поэтому перед вызовом метода Quit нужно убедиться, что печать закончена. Это можно сделать так, как описано в предыдущем разделе.

Microsoft Word печатает содержимое письма, причем вместе с картинками, если использован HTML-формат. Однако тему письма, отправителя и дату получения Word не печатает. Чтобы извлечь их из файла, воспользуемся библиотекой CDO (Collaboration Data Objects) [4], входящей в состав Windows.

Сначала получаем список eml-файлов в папке:

```
CDO.DropDirectoryClass dir = new
CDO.DropDirectoryClass();
CDO.IMessages imsgs = dir.GetMessages(DirectoryName);
```

Теперь для каждого сообщения мы можем получить необходимые атрибуты:

```
foreach (CDO.Message msg in imsgs)
{
   string sender = msg.Sender;
   string subject = msg.Subject;
   DateTime dt = msg.ReceivedTime;
}
```

Заключение

Не существует универсального способа печати документов из службы Windows. Однако печать наиболее распространенных типов документов можно выполнить, используя компоненты Microsoft Office, компоненты, входящие в состав Windows, а также свободно распространяемые компоненты.

Литература

1. Visual C++ ActiveX Control for hosting Office documents in Visual Basic or HTML. http://support.microsoft.com/kb/311765.
2. Microsoft Developer Support Office Framer Control 1.3 Sample (KB 311765). http://www.microsoft.com/downloads/details.aspx?FamilyId = CE2CA4FD-2169–4FAC-82AF-770AA9B60D77&displaylang = en
3. Acrobat Developer Center. http://www.adobe.com/devnet/acrobat
4. Collaboration Data Objects Roadmap. http://msdn.microsoft.com/en-us/library/ms978698.aspx