

Generator albumu HTML ze zdjęciami jako przykład tworzenia dokumentów XML, schematów i wykorzystania XSLT oraz używania niektórych dobrodziejstw klasy Image

czyli szybkie wykorzystanie XML, Xslt, XSD, System.Drawing i System.IO

Streszczenie

Artykuł opisuje tworzenie prostego (aczkolwiek dość praktycznego) programu generującego kod XML strony-albumu ze zdjęciami. Możliwe jest przekształcenie tego kodu do dowolnego formatu – program domyślnie wykorzystuje dostarczony plik XSLT do stworzenia kodu HTML. Aplikacja może również przeprowadzić walidację i przekształcić stworzony przez nas samych odpowiedni plik XML (musi być zgodny z dostarczonym schematem XSD) do gotowej strony HTML.

Artykuł ma na celu ukazanie jak łatwe jest tworzenie, parsowanie, walidowanie i przekształcanie plików XML oraz podstawowe operacje na plikach graficznych na platformie .NET.

Artykuł

Po wykonaniu odpowiednich czynności związanych z projektowaniem GUI i potrzebnych nam opcji, powinniśmy zająć się metodą pozwalającą na wybranie zdjęć, które docelowo mają znaleźć się na stronie.

Program umożliwia dwa sposoby generowania albumu:

1. Przez wybranie zdjęć z dysku twardego (wygeneruje wtedy kod XML i przekształci go jeśli będziemy chcieli za pomocą XSLT)
2. Przez wskazanie odpowiedniego pliku XML, w którym jest już opis albumu. Dokument ten będzie najpierw walidowany w celu sprawdzenia zgodności z założonym schematem. Następnie jeśli walidacja się powiedzie, użytkownik będzie mógł wskazać własny plik XSLT w celu przekształcenia pliku XML w dowolny inny format.

Najpierw zajmiemy się „oprogramowaniem” punktu 1.

Początkowe ustawienie opcji

Ponieważ program oferuje kilka opcji do wyboru (cztery z nich implementowane jako odpowiednie checkboxy), należy ustawić początkowe wartości tych opcji. W konstruktorze głównej klasy piszemy w tym celu:

```
// zapisujemy do zmiennej katalog z jakiego uruchamiany jest program
currentDir = Directory.GetCurrentDirectory();
// zapamiętujemy odpowiednie wartości początkowe opcji
createMins = chkCreateMins.Checked;
createMorePages = chkMorePages.Checked;
```

```

showNames = chkShowNames.Checked;
addSubtitles = chkAddSubtitler.Checked;
// tworzymy listy do przechowywania ścieżek i nazw plików ze zdjęciami
pathsToPhotos = new ArrayList();
photoNames = new ArrayList();

```

Metoda służąca do wczytywania zdjęć

Zacniemy od dialogu oferującego dostęp do plików.

```

OpenFileDialog open = new OpenFileDialog();
    open.InitialDirectory = "c:\\\" ;
    open.Multiselect = true;
    open.Filter = "pliki JPG | *.jpg" ;

```

Ustawiamy jego katalog startowy, pozwalamy na multiselekcję (możliwość wyboru więcej niż jednego pliku, bo o to nam chodzi), a później ustawiamy odpowiedni filtr. Filtr służy do wymuszenia pokazywania plików tylko jednego typu. W naszym przypadku chcemy wczytywać zdjęcia, wybieramy więc pliki z rozszerzeniem *.jpg. Następnie sprawdzamy czy nie wystąpiły żadne nieoczekiwane błędy i dodajemy po kolei zdjęcia do wcześniej utworzonej listy „pathsToPhotos”.

```

if( open.ShowDialog() == DialogResult.OK) {
    foreach(String s in open.FileNames) {
        pathsToPhotos.Add(s);
    }
}

```

Tworzenie miniaturki (thumbnaili)

Nikt nie chce oglądać stron, które długo się ładują. Dlatego my, jako twórcy generatora stron HTML powinniśmy odpowiednio przygotować pliki, jakie będą później używane wraz ze stroną. Program jako jedną z opcji ma „Generowanie miniaturki”, co oznacza dokładnie tyle co zmniejszenie obrazków przeznaczonych do wyświetlania na stronie jako odnośniki do ich większych wersji. Z pomocą przychodzi nam klasa Image, a dokładnie jedna z jej metod GetThumbnailImage(). Dodatkowo aby uniknąć śmiesznego rozciągania zdjęć, jeżeli wysokość jest większa niż szerokość zdjęcia (zdjęcie pionowe) zmieniamy kolejność parametrów dla metody GetThumbnailImage(). W przypadku gdy zdjęcie jest kwadratowe, rozmiar jego miniaturki będzie wynosił 120x120. Celowo rozmiar miniaturki nie jest zależny od rozmiaru zdjęcia, bo nawet „10% miniaturka” zdjęcia o rozdzielczości 2304x1728 byłaby zbyt duża na stronę. Kod odpowiedzialny za tworzenie miniaturki wygląda następująco:

```

private void CreateMins()
{
    if(!Directory.Exists("thumbs")) Directory.CreateDirectory("thumbs");
    foreach(String s in pathsToPhotos) {
        Image.GetThumbnailImageAbort myCallBack =
            new Image.GetThumbnailImageAbort(ThumbnailCallback);
        Bitmap photo = new Bitmap(s);

        //generating thumbnails:
        if(photo.Height > photo.Width)
        {
            resized = photo.GetThumbnailImage(

```

```

        120, 160, myCallBack, IntPtr.Zero);
    }
    else if(photo.Height < photo.Width)
    {
        resized = photo.GetThumbnailImage(
            160, 120, myCallBack, IntPtr.Zero);
    }
    else
    {
        resized = photo.GetThumbnailImage(
            120, 120, myCallBack, IntPtr.Zero);
    }
    string[] tmp = s.Split('\\');

    //saving thumbnails:
    resized.Save("thumbs/"+tmp[tmp.Length-1], ImageFormat.Jpeg);
}

minsCreated = true;
}

```

Następnie miniatury zdjęć zapisujemy w katalogu thumbs/ stworzonym przez program. Warto zauważyć „dziwną” składnię metody GetThumbnailImage(), której jako jeden z parametrów przekazujemy delegata Image.GetThumbnailImageAbort. Sami musimy stworzyć funkcję zwracającą false lub true i przekazać ją utworzonemu delegatowi. Funkcja powinna zwracać true tylko i wyłącznie wtedy, gdy chcemy przerwać całkowicie działanie metody GetThumbnailImage(), w pozostałych przypadkach (i w naszym) powinna zwracać false.

```

public bool ThumbnailCallback()
{
    return false;
}

// gdzies dalej w metodzie
// tworzymy delegata do naszej funkcji
Image.GetThumbnailImageAbort myCallBack = new
Image.GetThumbnailImageAbort(ThumbnailCallback);

```

Na tym zakończyliśmy pisanie kodu do tworzenia miniaturki zdjęć. Dzięki odpowiednim klasom i metodom, cały kod zajął nam kilkanaście linijek.

Struktura dokumentu XML z albumem

Przed przystąpieniem do dalszego pisania, powinniśmy zastanowić się nad sposobem przechowywania danych o zdjęciach, ich ścieżkach, ścieżkach do ich miniaturki. Ja przyjąłem następującą budowę pliku XML:

```

<album>
  <title>tytuł albumu</title>
  <table>
    <row>
      <image>
        <name>nazwa</name>
        <path>ściezka</path>

```

```

        <thumb>ściezka do miniaturki</thumb>
        <description>opis obrazka</description>
    </image>
</row>
</table>
</album>

```

Takie założenie pozwoli na dokładny i jednocześnie bardzo przejrzysty opis każdego zdjęcia. Dzięki temu, bardzo łatwo będzie później napisać odpowiedni plik XSL mający przekształcić nasz album.

Tworzenie pliku XML z albumem

Ponieważ daliśmy użytkownikowi możliwość ustalenia największej ilości zdjęć, jakie mogą być wyświetlane na jednej stronie i w jednym wierszu, może zdarzyć się sytuacja, że będziemy potrzebować więcej niż jednego pliku opisującego album. Jednocześnie użytkownik może nie chcieć tworzyć większej ilości stron i chce wszystkie zdjęcia umieścić na jednej. Dlatego decydujemy najpierw ile plików program będzie tworzył i liczymy ile wierszy potrzebnych jest do „upakowania” wszystkich zdjęć. Jest to bardzo ważna decyzja, ponieważ (jak to już ustaliliśmy) każdy plik ma mieć odpowiednią strukturę, co wymusza na nas odpowiednie rozplanowanie ilości wierszy i zdjęć. Nie może się zdarzyć sytuacja, że program nie zamknie jakiegoś taga, ponieważ w takim przypadku wynikowy plik XML nie byłby poprawnie wyświetlany. W takim wypadku użytkownik miałby jednak możliwość (przy transformacji za pomocą XSL) przeszukać jeszcze strukturę dokumentu za pomocą XPath i upewnić się czy wszystkie węzły są kompletne. Ponieważ jednak zakładamy, iż użytkownicy nie będą mieli ochoty robić nic więcej poza dostarczeniem reguł jak ma wyglądać wyjściowy dokument i jakiego ma być formatu, to my zajmiemy się zapewnieniem, że każdy otwierający znacznik ma swój zamykający odpowiednik ;).

```

int nOfFiles = 1;
int maxOnPage = Int32.Parse(labHowManyPhotos.Text);
if(createMorePages == true && (Int32.Parse(labHowManyPhotos.Text) !=
Int32.Parse(txtMax.Text)))
{
if((Int32.Parse(labHowManyPhotos.Text) % Int32.Parse(txtMax.Text)) == 0)
    nOfFiles = (Int32.Parse(labHowManyPhotos.Text) /
Int32.Parse(txtMax.Text));
else
    nOfFiles = (Int32.Parse(labHowManyPhotos.Text) /
Int32.Parse(txtMax.Text)) + 1;

maxOnPage = Int32.Parse(txtMax.Text);
}

```

Teraz możemy przejść do pisania kodu odpowiedzialnego za zapisywanie danych do pliku XML. Cały ten proces „wrzucamy” do pętli:

```

for(int i=1; i <= nOfFiles; i++)
{
    if(nOfFiles == 1)

```

```

    {
        xmlWriter = new XmlTextWriter(fileName+".xml", null);
    }
    else
    {
        xmlWriter = new XmlTextWriter(fileName+" "+i+".xml", null);
    }
    //
    // tutaj kod do operacji na plikach
    //
}

```

Następnie wstawiamy odpowiednie tagi XML zgodnie z ustanowioną wcześniej strukturą. Zaczynamy od metody WriteStartDocument(), później używamy metod WriteStartElement() w celu otwarcia odpowiedniego znacznika, WriteString() w celu wpisania jego wartości i pamiętamy o zamknięciu znacznika metodą WriteEndElement().

```

xmlWriter.WriteStartDocument();

//tworzymy "korzen", na razie go nie zamykamy!
xmlWriter.WriteStartElement("album");

//tworzymy poszczególne znaczniki
xmlWriter.WriteStartElement("title");
xmlWriter.WriteString(txtTitle.Text);
xmlWriter.WriteEndElement();

//otwieramy znacznik <table>
xmlWriter.WriteStartElement("table");

```

Wstawianie zdjęć nastręczy nam jednak kilka problemów, mianowicie:

- chcemy aby w wierszu była wybrana przez użytkownika ilość zdjęć
- w pewnym momencie może się okazać, że na stronie jest już ilość zdjęć, którą użytkownik uznał za maksymalną i **musimy otworzyć nowy plik**
- jeśli nastąpi punkt powyższy, **musimy** zamknąć wszystkie otwarte znaczniki
- wstawiamy zdjęcia dopóki nie okaże się, że wstawiliśmy już wszystkie

Sam kod odpowiedzialny za odpowiednie rozmieszczenie zdjęć i za spełnienie powyższych punktów zajmuje ok. 40 linijek, dlatego zapraszam do wglądu w dostarczony kod programu.

Natomiast kod potrzebny do wygenerowania znacznika <image> wraz z jego elementami sprowadza się znowu do wykorzystania metod WriteStartElement(), WriteString() i WriteEndDocument(). Dane o każdym zdjęciu (a więc jego nazwa i ścieżka) zawarte są w liście photoNames. Mamy więc (poglądowo):

```

xmlWriter.WriteStartElement("image");
//jeżeli użytkownik wybrał opcje pokazywania nazw
if(showNames == true)
{
    xmlWriter.WriteStartElement("name");
    xmlWriter.WriteString(photoNames[index].ToString());
    xmlWriter.WriteEndElement();
}

```

```

xmlWriter.WriteStartElement("path");
xmlWriter.WriteString(photoNames[index].ToString());
xmlWriter.WriteEndElement();

xmlWriter.WriteStartElement("thumb");
if(minsCreated == true)
    xmlWriter.WriteString("thumbs/"+photoNames[index].ToString());
else
    xmlWriter.WriteString(photoNames[index].ToString());
xmlWriter.WriteEndElement();

//jeżeli użytkownik wybrał opcje pokazywania podpisów
if(addSubtitles == true)
{
    xmlWriter.WriteStartElement("description");
    xmlWriter.WriteString("_podpis_zdjecia_");
    xmlWriter.WriteEndElement();
}

```

Jak widać jako podpis jest do każdego obrazka tworzony napis „_podpis zdjęcia_”. Użytkownik musi więc według własnego uznania podpisać zdjęcia w wygenerowanym pliku XML. Mogliśmy zdecydować, że podpisy będą czytane z zewnętrznego pliku, ale to wymagałoby niemięjszej ilości pracy od użytkownika (też przecież musiałby podpisać każde zdjęcie) a dodatkowo wprowadziłoby zamęt w przypadku dodania innego zdjęcia (co np. zrobić jeśli mamy podpisane tylko parę zdjęć ze wszystkich wybranych?).

Na koniec zamykamy plik XML, który właśnie edytowaliśmy:

```

xmlWriter.WriteEndElement();
xmlWriter.WriteEndDocument();
xmlWriter.Flush();
xmlWriter.Close();

```

Metoda służąca do wczytania i walidacji wcześniej wygenerowanego pliku XML

Według założeń, chcemy dać użytkownikom możliwość przekształcania stworzonych przez nich plików XML opisujących album do pliku HTML (domyślnie to robi program) lub do dowolnego innego formatu (do którego da się przekształcić plik XML za pomocą XSLT). Razem z programem dostarczony jest plik transformator.xsl, który generuje kod HTML. Użytkownik może jednak sam stworzyć odpowiedni plik XSL lub XSLT i przetworzyć za pomocą programu dowolny plik XML.

Na początku musimy jednak zdefiniować odpowiedni schemat po to, by wczytywane pliki na pewno opisywały album ;). Tworzenie plików XSD to niełatwe zagadnienie i trzeba naprawdę dużo przeczytać, żeby stworzyć „dobry” schemat. Ponieważ jednak żyjemy w czasach „zbliżających się dead-line’ów” ;) i powinniśmy skupić się raczej na tworzonej aplikacji, Microsoft ułatwia nam zadanie utworzenia naprawdę dobrego schematu do kilku kliknięć myszką...

Tworzenie schematu z pliku XSD za pomocą VS .NET

Znamy już strukturę naszego albumu (patrz wyżej). Tworzymy więc odpowiedni plik XML i otwieramy go w Visual Studio. Następnie z menu wybieramy „XML / Create Schema”. Studio stworzy odpowiedni plik .xsd i wyświetli jego zawartość.

Czytanie i walidacja dostarczonych plików

Na wstępie musimy wczytać plik XML. Użyjemy w tym celu obiektu klasy XmlValidatingReader, ponieważ musimy sprawdzić czy wczytywany plik jest zgodny ze schematem.

```
// tworzymy potrzebne referencje
XmlValidatingReader xmlValid;
XmlTextReader reader;

// tworzymy obiekt klasy XmlSchemaCollection
// do którego dodamy utworzony wcześniej przez VS plik ze schematem
XmlSchemaCollection xsc = new XmlSchemaCollection();
xsc.Add(null, pathToSchema);
// ***
// tutaj znajduje sie tworzenie dialogu do wyboru plikow
// to zagadnienie było poruszone już wyzej
// ***

reader = new XmlTextReader(file);
xmlValid = new XmlValidatingReader(reader);

// dodajemy nasza kolekcje schematow do kolekcji Schemas obiektu xmlValid
xmlValid.Schemas.Add(xsc);
// ustawiamy typ walidacji na "Schema"
xmlValid.ValidationType = ValidationType.Schema;
// dodajemy callback podobnie jak w metodzie GetThumbnailImage()
xmlValid.ValidationEventHandler += new ValidationEventHandler
(ValidationCallback);

//czytamy (jednocześnie walidujac) plik
while (xmlValid.Read()){
xmlValid.Close();
```

Znowu pamiętamy o dodatkowej metodzie:

```
private void ValidationCallback (object sender, ValidationEventArgs args)
{
    validationSuccessfull = false;
}
```

Transformacje XSL na platformie .NET

Jeżeli walidacja się powiodła i rzeczywiście mamy do czynienia z plikiem reprezentującym album, przekształcamy go wykorzystując w tym celu plik XSL wskazany wcześniej przez użytkownika. Wymuszamy wcześniej na użytkowniku wskazanie takiego pliku, ponieważ jest on konieczny do transformacji. Jeden taki przykładowy plik jest dołączony do programu.

Napisanie kodu potrzebnego do przekształcenia plików XML jest na platformie .NET naprawdę proste dzięki klasie XslTransform (z przestrzeni nazw System.Xml.Xsl) i jej

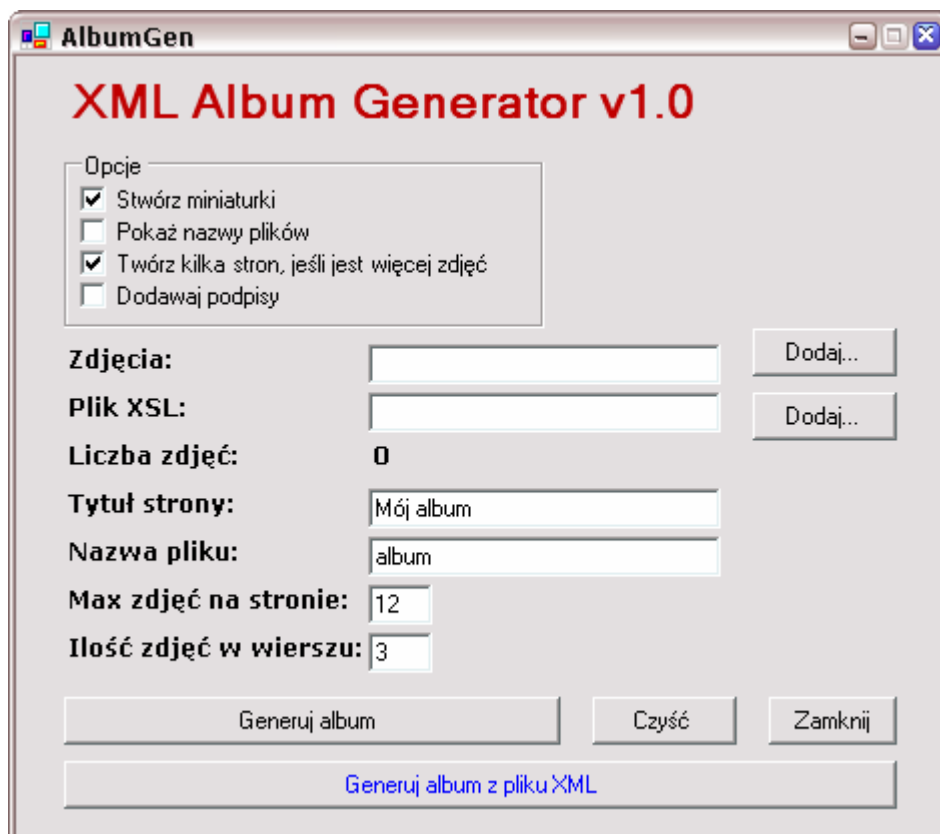
metodom. Tworzymy obiekt tej klasy, następnie za pomocą metody Load() ładujemy plik XSL i zdajemy się na metodę Transform(), która wykona całą „robotę”. Metoda ta przyjmuje jako argumenty plik źródłowy i nazwę pliku wyjściowego, który chcemy utworzyć.

```
if (validationSuccessfull == true)
{
    XslTransform transform = new XslTransform();
    transform.Load(pathToXslFile);
    transform.Transform(pathToXmlFile, "album.html");
}
```

To już koniec...

Ponieważ nikt nie lubi czytać zbyt długich artykułów, starałem się umieścić tylko najważniejsze rzeczy. Informacje tu zawarte mogą tylko pomóc w tworzeniu aplikacji. Pomiąłem w artykule kilka oczywistych spraw (jak np. sprawdzanie czy użytkownik dodał jakiegokolwiek zdjęcia przy próbie tworzenia albumu). Pełny kod wraz ze wspomnianymi plikami dostępny jest w załączeniu, zapraszam do jego przeglądania.

Przed kopiowaniem kodu, warto samemu popробować ;). Gotowa aplikacja wygląda tak:



W przypadku korzystania programu warto zebrać wszystkie zdjęcia w jednym katalogu i z niego uruchomić program. Dzięki temu utworzone strony będą zawierać linki względne i w razie przeniesienia na serwer nie będą „szukały” zdjęć na „C:\Moje Obrazy\Prywatne\Zdjecia\Zwakacji\nieposortowane\” :-D.

Zakończenie:

W artykule próbowałem przedstawić najważniejsze zagadnienie potrzebne przy tworzeniu programu, który w pewien sposób może nam ułatwić życie. Nawet jeśli nie przyda się generator albumu, warto wiedzieć jak można tworzyć miniatury obrazów, tworzyć, czytać i walidować pliki XML, tworzyć schematy i stosować transformacje XSLT.

Dodatkowo warto wspomnieć, że dzięki C# i Visual Studio .NET to wszystko jest naprawdę łatwe...