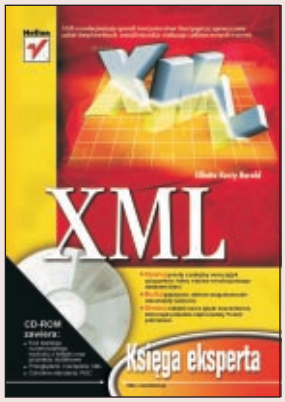
XML. Księga eksperta



Autor: Rusty Harold
Tłumaczenie: Tomasz Żmijewski
ISBN: 83-7197-275-X
Tytuł oryginału: XML. Bible
Format: B5, stron: 944
oprawa twarda
Zawiera CD-ROM

Opanuj prosty a potężny nowy język znaczników, który właśnie rewolucjonizuje działanie Sieci! Naucz się, jak tworzyć poprawne, dobrze zorganizowane dokumenty sieciowe! Poznaj nowe języki znacznikowe, które będą idealnie odpowiadały Twoim potrzebom!
XML rewolucjonizuje sposób tworzenia stron WWW poprzez uproszczenie zadań dotąd trudnych, umożliwia także realizację całkiem nowych wyzwań. Książka jest oparta na najnowszych standardach W3C; jest to pełny przewodnik wraz z kompletną dokumentacją. „Księga Eksperta" przeprowadzi Cię przez najtrudniejsze zadania:

- dowiesz się, jak działa XML;
- poznasz podstawy tworzenia w tym języku dokumentów;
- zastosowanie prostych rozwiązań opartych na XML.

Dzięki licznym przykładom, specyfikacjom i odnośnikom do zasobów internetowych żadne pytanie nie pozostanie bez odpowiedzi.

- Jeśli będziesz przestrzegał prostych zasad, stworzysz poprawne dokumenty XML.
- Definiuj znaczniki, które będą coś mówiły o zawartości Twojego dokumentu.
- Formatuj dokumenty za pomocą arkuszy stylów.
- Twórz własne języki znaczników.
- Poznaj RDF, VML, CDF i inne aplikacje XML.

Na załączonej płycie znajdziesz najważniejsze narzędzia języka XML i przykłady kodu, w tym:

- kod wszystkich numerowanych wydruków z tej książki;
- przeglądarki i narzędzia XML;
- standardy World Wide Web Consortium związane z XML;

Jeśli można to zrobić w XML, i Ty możesz to zrobić...

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 6.

Dokumenty XML poprawnie sformułowane

HTML 4.0 ma około setki różnych znaczników. Większość z nich może mieć pół tuzina atrybutów, co daje kilka tysięcy możliwych kombinacji. Jako że XML jest narzędziem silniejszym niż HTML, mógłbyś się spodziewać jeszcze większej ilości znaczników, ale tak nie jest. XML jest tak skuteczny dzięki swej prostocie i możliwości rozszerzania, a nie przez mnogość znaczników.

Tak naprawdę XML niemalże w ogóle nie ma znaczników. Zamiast tego możesz sam sobie znaczniki definiować według potrzeb. Jednak znaczniki te i budowane z nich dokumenty nie są całkowicie dowolne, gdyż muszą być zgodne z pewnymi ogólnymi zasadami, które omówimy dokładnie w tym rozdziale. Dokument zgodny z tymi zasadami nazywamy dokumentem *poprawnie sformułowanym*. Poprawne sformułowanie jest minimalnym poziomem, przy którym prawidłowo mogą pracować procesory XML i przeglądarki. W tym rozdziale poznasz wymogi stawiane prawidłowo sformułowanemu kodowi XML i HTML. Szczególny nacisk położymy na różnice między XML a HTML.

Z czego składają się dokumenty XML

Dokument XML składa się ze znaczników i danych tekstowych. Jest to zbiór bajtów ustalonej długości zgodny z pewnymi wytycznymi. Może to być plik lub nie. Dokumentem XML może na przykład być:

- zbiór przechowywany w bazie danych;
- zbiór tworzony w biegu, w pamięci, przez program CGI;
- zestaw kilku odrębnych plików, wzajemnie w sobie zanurzonych;
- zbiór, który nigdy nie był osobnym plikiem.

Nic istotnego jednak nie stracisz, jeśli będziesz o dokumencie XML myślał jako o pliku, o ile tylko pamiętasz, że jednak nie musi to być plik na dysku twardym.

Dokumenty XML składają się z jednostek alokacji zwanych *encjami*. Każda encja zawiera tekst lub dane binarne, nigdy oba te rodzaje danych jednocześnie. Dane tekstowe składają się ze znaków, dane binarne to na przykład obrazki, aplety i im podobne. Konkretnym przykładem może być plik HTML zawierający tylko znaczniki `<IMG>` – jest to encja, ale nie dokument. Dokumentem staje się dopiero ten plik wraz z obrazkami opisanymi znacznikami `<IMG>`.

W tym rozdziale i kilku następnych będziemy zajmować się tylko prostymi dokumentami XML, które składają się z pojedynczej encji, samego dokumentu. Co więcej, dokumenty te będą zawierać tylko dane tekstowe, bez żadnych danych binarnych, jak obrazki czy aplety. Takie dokumenty można w pełni same zinterpretować, bez konieczności czytania jakichkolwiek innych plików. Tego typu dokumenty zwykle mają w deklaracji XML wartość parametru `standalone` ustawioną na `yes`, na przykład:

```
<?xml version="1.0" standalone="yes"?>
```

Encje zewnętrzne i odwołania do encji mogą zostać użyte w celu połączenia szeregu plików i innych źródeł danych w jeden dokument XML. Takie dokumenty nie mogą być w pełni zanalizowane bez odwołania się do innych plików. Zwykle mają one w deklaracji parametr `standalone` ustawiony na `no`:

```
<?xml version="1.0" standalone="no"?>
```

Encje zewnętrzne i odwołania do encji omówimy w rozdziale 9.

Znaczniki i dane tekstowe

Dokumenty XML to tekst. Tekst składa się ze znaków. Znak to litera, cyfra, znak przestankowy, spacja, tabulator lub coś tego typu. XML używa zestawu znaków Unicode, który zawiera nie tylko zwykłe litery i symbole z alfabetu łacińskiego i innych alfabetów środkowo– i zachodnioeuropejskich, zawiera też cyrylicę, pismo greckie, hebrajskie, arabskie, dewanagari (sanskryt). Co więcej, zestaw ten zawiera najczęściej używane ideogramy Han do pisma chińskiego i japońskiego oraz sylaby Hangul do koreańskiego. W tym rozdziale jeszcze ograniczymy się do alfabetu łacińskiego.

Międzynarodowe zestawy znaków omówimy w rozdziale 7.

Tekst dokumentu XML tworzy dwie struktury: dane znakowe i znaczniki. Dane znakowe to zasadnicza treść dokumentu, znaczniki natomiast opisują przede wszystkim strukturę logiczną dokumentu. Przypomnij sobie na przykład wydruk 3.2 z rozdziału 3., *pozdrowienie.xml*, podany niżej:

```
<?xml version="1.0" standalone="yes"?>
<POZDROWIENIE>
Hello XML!
</POZDROWIENIE>
```

W tym wypadku `<?xml version="1.0" standalone="yes"?>`, `<POZDROWIENIE>` i `</POZDROWIENIE>` to znaczniki, natomiast `Hello XML!` to dane znakowe. Przewagą XML nad innymi formatami danych jest to, że faktyczne dane są odróżnione od znaczników.

Ściślej rzecz biorąc oznakowanie dokumentu obejmuje komentarze, odwołania do znaków, odwołania do encji, ograniczniki sekcji CDATA, instrukcje przetwarzania i definicje DTD. Wszystko inne to dane znakowe. Jednak taka definicja może być nieco myląca, gdyż podczas przetwarzania dokumentu część oznakowania może zmienić się w dane znakowe. Na przykład znacznik `&gt;` zmienia się na znak większości (>). Dane znakowe, które zostają po przetworzeniu dokumentu oraz znaczniki zamienione na dane przez nie reprezentowane nazywamy zanalizowanymi danymi tekstowymi.

Komentarze

Komentarze XML wyglądają niemalże tak samo, jak komentarze HTML: zaczynają się od `<!--` i kończą `-->`. Wszystkie znaki między tymi dwoma napisami są ignorowane przez procesor XML, jakby ich tam nie było. Komentarzy używa się do robienia notatek lub do chwilowego wyłączenia tych części dokumentu, które jeszcze nie są gotowe. Na przykład:

```
<?xml version="1.0" standalone="yes"?>
<!--Wydruk 3.2 z Biblii XML-->
<POZDROWIENIE>
Hello XML!
<!--Bye, XML-->
</POZDROWIENIE>
```

Używając komentarzy należy postępować zgodnie z podanymi niżej zasadami:

1. Komentarze nie mogą znaleźć się przed deklaracją XML, która bezwzględnie musi być pierwszym elementem dokumentu. Poniższy kod jest nie do przyjęcia:

   ```
   <!--Wydruk 3.2 z Biblii XML-->
   <?xml version="1.0" standalone="yes"?>
   <POZDROWIENIE>
   Hello XML!
   <!--Bye, XML-->
   </POZDROWIENIE>
   ```

2. Komentarzy nie można umieszczać wewnątrz znaczników, zatem poniższy kod jest błędny:

   ```
   <?xml version="1.0" standalone="yes"?>
   <POZDROWIENIE>
   Hello XML!
   </POZDROWIENIE <!--Bye, XML--> >
   ```

3. Komentarzy można użyć do otaczania i ukrywania znaczników. W poniższym przykładzie znacznik `<ANTYPOZDROWIENIE>` wraz z elementami potomnymi znajduje się w komentarzu i podczas analizy dokumentu nie będzie w ogóle uwzględniony, jakby nie istniał:

```
<?xml version="1.0" standalone="yes"?>
<DOKUMENT>
<POZDROWIENIE>
Hello XML!
<!--Bye, XML-->
</POZDROWIENIE>
<!--
<ANTYPOZDROWIENIE>
Bye, XML!
</ANTYPOZDROWIENIE>
-->
</DOKUMENT>
```

Jako że komentarze faktycznie wycinają fragment tekstu, należy starannie je wstawiać, aby nie umieścić w nim znacznika początkowego lub końcowego. Na przykład poniższy fragment jest błędny:

```
<?xml version="1.0" standalone="yes"?>
<POZDROWIENIE>
Hello XML!
<!--
</POZDROWIENIE>
-->
```

Kiedy bowiem usuniemy tekst komentarza, otrzymamy:

```
<?xml version="1.0" standalone="yes"?>
<POZDROWIENIE>
Hello XML!
```

Jako że znacznik `<POZDROWIENIE>` nie ma już znacznika końcowego, dokument XML przestaje być poprawnie sformułowany.

4. Wewnątrz komentarza nie może pojawić się ciąg składający się z dwóch minusów (`--`), poza znacznikiem początku i końca komentarza. Na przykład poniższy komentarz jest niepoprawny:

```
<?xml version="1.0" standalone="yes"?>
<DOKUMENT>
  <POZDROWIENIE>
    Hello XML!
  </POZDROWIENIE>
<!--
  <ANTYPOZDROWIENIE>
    <!--Bye, XML!-->
  </ANTYPOZDROWIENIE>
-->
</DOKUMENT>
```

Wynika z tego, że jeśli chcesz skomentować dużo kodu C, Java lub JavaScript, to możesz mieć kłopoty, gdyż w kodzie takim roi się wprost od wyrażeń typu `i--` czy `--numberLeft`. W zasadzie jednak, kiedy rzecz już rozpoznasz, nietrudno ci będzie sobie z tym problemem poradzić.

Odwołania do encji

Odwołania do encji to znaczniki, które podczas analizy dokumentu są zamieniane na dane znakowe. XML ma pięć encji predefiniowanych, zestawiono je w tabeli 6.1. Odwołań do encji używa się w dokumentach XML zamiast wstawiania pewnych znaków, które mogłyby być zinterpretowane jako część oznakowania. Na przykład odwołanie do encji `<` zastępuje znak mniejszości (<), który normalnie byłby zinterpretowany jako początek znacznika.

Tabela 6.1. *Predefiniowane encje XML*

Encja	Odpowiadający jej znak
&amp;	&
&lt;	<
&gt;	>
&quot;	"
&apos;	'

W XML, w przeciwieństwie do HTML, odwołania do encji muszą kończyć się średnikiem, wobec czego o ile > jest poprawnym odwołaniem, to &gt już nie.

W normalnym tekście XML znaki mniejszości i znaki `&` zawsze interpretowane są odpowiednio jako początek znacznika lub odwołania do encji (nienormalny tekst sekcji CDATA omówiony zostanie za chwilę). Wobec tego < i `&` należy zawsze zamieniać na `<` i `&`. Na przykład tekst „Meyer & Meyer" należy zapisać jako `Meyer & Meyer`.

Znaki większości oraz cudzysłów pojedynczy i podwójny muszą być zastąpione encjami wtedy, gdy użyte bezpośrednio mogłyby zostać zinterpretowane jako część oznakowania. Zwykle jednak łatwiej jest po prostu nabrać nawyku używania encji, niż zastanawiać się, czy tutaj dany znak może być zinterpretowany jako część oznakowania, czy też nie.

Odwołań do encji można też użyć podając wartości atrybutów, na przykład:

```
<PARAM NAME="zdanie" VALUE="Kiedy programujesz, nie myl &quote;
 z &apos;">
</PARAM>
```

CDATA

Zwykle cokolwiek znajdzie się wewnątrz pary trójkątnych nawiasów (<>), należy do oznakowania, reszta to dane znakowe. Jest jednak pewien wyjątek. W sekcjach CDATA cały tekst to same dane znakowe. Wszystko, co wygląda jak znaczniki lub odwołania do encji, tak naprawdę jest tylko tekstem znacznika lub tekstem odwołania. Procesor takich znaczników czy encji w ogóle nie analizuje.

Sekcji CDATA używa się, kiedy chcesz, aby cały fragment tekstu był zinterpretowany jako same dane znakowe, a nie jako znaczniki. Szczególnie jest to przydatne, gdy masz do czynienia z większą porcją tekstu pełnego znaków <, >, & i ", nie zawierającego znaczników. Dobrym przykładem może być kod źródłowy C czy Java.

Sekcje CDATA są także szczególnie użyteczne, jeśli chcesz pisać o XML stosując XML. Na przykład ta książka zawiera mnóstwo niewielkich fragmentów kodu XML. Procesor tekstu, którego używam, specjalnie się tym nie przejmuje. Jeśli jednak przyjdzie mi tę książkę konwertować na XML, zamiana wszystkich mniejszości na `<` i znaków & na `&` byłaby trudnym zadaniem:

```
&lt;?xml version="1.0" standalone="yes"?&gt;
&lt;POZDROWIENIE&gt;
    Hello XML!
&lt;/POZDROWIENIE&gt;
```

Aby nie być zmuszonym do robienia tego, można użyć sekcji CDATA, która nakaże prezentację bloku tekstu bez jego interpretacji. Sekcja CDATA zaczyna się od `<![CDATA[` i kończy na `]]>`, na przykład:

```
<![CDATA[
<?xml version="1.0" standalone="yes"?>
<POZDROWIENIE>
  Hello XML!
</POZDROWIENIE>
]]>
```

Jedyny napis, którego w sekcji CDATA nie można użyć, to ciąg tę sekcję zamykający, czyli `]]>`. W sekcji CDATA mogą wystąpić komentarze, ale nie są one tam traktowane jako komentarze, czyli przekazana będzie tak treść komentarza, jak i jego znaczniki otwierający i zamykający.

> Jako że w sekcji `CDATA` nie może wystąpić ciąg `]]>`, sekcje te nie mogą być w sobie zagnieżdżane. Wobec tego trudno jest w XML pisać o sekcjach `CDATA`. Możliwe rozwiązanie to użycie odwołań do encji znakowych `<` i `&`.

Znaczniki

Tym, co odróżnia pliki XML od zwykłych plików, to oznakowanie. Oznakowanie to przede wszystkim znaczniki. Jako że w poprzednim rozdziale widziałeś sposób użycia znaczników, w tej sekcji zdefiniujemy znaczniki i dokładniej omówimy sposób ich użycia.

Krótko mówiąc, znacznik w XML to cokolwiek zaczynającego się znakiem <, kończącego znakiem > i nie znajdującego się w sekcji CDATA ani w komentarzu. Wobec tego znacznik XML ma taką samą postać, jak znacznik HTML: najpierw mamy znak <, potem nazwę znacznika, zamykający znak >. Znacznik zamykający elementu zaczyna się od </, później znów nazwa i >.

Nazwy znaczników

Każdy znacznik ma nazwę. Nazwy znaczników muszą zaczynać się literą lub podkreśleniem (_). Dalej mogą występować litery, cyfry, podkreślenia, znaki minus i kropki. Nie mogą pojawić się natomiast żadne białe znaki (często jako odpowiedników spacji używa się podkreśleń). Oto poprawne znaczniki XML:

```
<HELP>
<Ksiazka>
<Książka>
<volume>
<naglowek1>
<sekcja.akapit>
<Ewa_Adamska>
<_8ball>
```

Formalnie rzecz biorąc, dwukropek jest znakiem dopuszczalnym w nazwach znaczników, jednak zarezerwowano go dla przestrzeni nazw. Przestrzenie nazw umożliwiają mieszanie i łączenie zbiorów znaczników, w których te nazwy się powtarzają. Przestrzenie nazw omówione zostaną w rozdziale 18.

Oto znaczniki XML niepoprawne składniowo:

```
<Książka%7>
<volume control>
<1naglowek>
<Ewa Adamska>
<.pracownik.pobory>
```

Reguły dotyczące nazw znaczników odnoszą się również do określania nazw atrybutów, wartości atrybutów typu ID, nazw encji i wielu innych konstrukcji, które poznasz w dalszych rozdziałach

Znaczniki zamykające mają taką samą nazwę jak otwierające, ale uzupełniane są znakiem / po otwierającym nawiasie kątowym. Jeśli na przykład znacznikiem początkowym jest <COKOLWIEK>, to znacznikiem końcowym jest </COKOLWIEK>. Oto znaczniki końcowe podanych wcześniej znaczników początkowych:

```
</HELP>
</Ksiazka>
</Książka>
</volume>
</naglowek1>
</sekcja.akapit>
</Ewa_Adamska>
</_8ball>
```

Nazwy w XML zależne są od wielkości liter. Odwrotnie niż w HTML, gdzie <P> i to ten sam znacznik, więc element może otworzyć znacznik <P>, a zamknąć go może . Znaczniki w poniższym zestawie nie są znacznikami końcowymi znaczników początkowych podanych wcześniej w tej sekcji.

```
</help>
</ksiazka>
</książka>
</VOLUME>
</Naglowek1>
</SEKCJA.akapit>
</EWA_ADAMSKA>
</_8Ball>
```

Choć w znacznikach można używać tak wielkich, jak i małych liter, w tej książce zwykle stosowane będą wielkie litery, głównie dlatego, że lepiej wyglądają. Kiedy jednak będziemy używać czyjegoś zestawu znaczników, to konieczne będzie dostosowanie się do jego zwyczajów.

Znaczniki puste

Wiele elementów HTML nie zawierających danych nie ma znaczników końcowych, przykładami mogą być `</LI>`, `</IMG>`, `</HR>` czy `</BR>`. Niektórzy autorzy stron umieszczają znacznik końcowy `</LI>` po pozycji listy, podobnie robią niektóre narzędzia. Jednak HTML 4.0 jasno mówi, że nie jest to obowiązkowe. Jak to się dzieje ze wszystkimi nierozpoznanymi znacznikami HTML, `</LI>` jest pomijane.

Całkiem inaczej rzecz się ma w XML. Cała idea XML to umożliwienie używania coraz to nowych elementów, kiedy tylko stają się potrzebne. Wobec tego takie nierozpoznane znaczniki nie mogą być po prostu ignorowane. Co więcej, procesor XML musi określić w biegu, czy nowo pojawiający się znacznik ma, czy nie ma znacznika końcowego.

XML rozróżnia znaczniki mające zamknięcie i go nie mające – te ostatnie nazywamy *elementami pustymi*. Znaczniki elementów pustych zakończone są ukośnikiem przed znakiem większości (czyli `/>`). Przykładowo mogą to być znaczniki `<BR/>` lub `<HR/>`.

Obecnie używane przeglądarki sieciowe różnie traktują tego typu znaczniki. Jeśli jednak chcesz zachować wsteczną kompatybilność, możesz po prostu użyć zwykłego znacznika kończącego, nie umieszczając przed nim żadnego tekstu, na przykład:

```
<BR></BR>
<HR></HR>
<IMG></IMG>
```

Kiedy w następnych rozdziałach nauczysz się o DTD i arkuszach stylów, poznasz kilka innych metod zapewnienia zgodności dokumentów HTML, które muszą być obsługiwane przez stare przeglądarki.

Atrybuty

Jak wyjaśniono wcześniej, znaczniki początkowe i znaczniki elementów pustych mogą zawierać *atrybuty*. Atrybuty to pary nazwa-wartość rozdzielane znakiem równości (=), na przykład

```
<POZDROWIENIE LANGUAGE="English">
  Hello XML!
  <FILM SRC="Machanie.mov"/>
</POZDROWIENIE>
```

W tym wypadku znacznik `<POZDROWIENIE>` ma atrybuty `LANGUAGE`, którego wartością jest *`English`*. Znacznik `<FILM>` ma atrybut `SRC`, którego wartością jest *`Machanie.mov`*.

Nazwy atrybutów

Nazwy atrybutów to napisy zgodne z tymi samymi regułami, które odnoszą się do nazw znaczników. Oznacza to, że nazwy atrybutów muszą zaczynać się od litery lub podkreślenia, dalej występują litery, cyfry, podkreślenia, znaki minus i kropki. Nie mogą pojawiać się białe znaki (spacje często zastępowane są przez podkreślenia).

Jeden znacznik nie może mieć dwóch atrybutów z takimi samymi nazwami. Na przykład poniższy kod jest nieprawidłowy:

```
<PROSTOKĄT BOK="8cm" BOK="10cm"/>
```

Nazwy atrybutów zależne są od wielkości liter. Atrybut `BOK` to inny atrybut niż `bok` czy `Bok`. Wobec tego można zastosować poniższe:

```
<KOSTKA BOK="8cm" bok="10cm" Bok="31cm"/>
```

Jednak takie postępowanie może prowadzić do nieporozumień, więc zdecydowanie odradzam kodowanie w ten sposób.

Wartości atrybutów

Wartości atrybutów to także napisy. Nawet jeśli napis zawiera liczbę, jak poniższy atrybut `LENGTH`, to atrybut ma wartość dwóch cyfr: 7 i 2, nie jest to liczba 72.

```
<RULE LENGTH="72"/>
```

Jeśli piszesz program przetwarzający XML, przed robieniem jakichkolwiek obliczeń będziesz musiał przekształcić napis na liczbę.

W przeciwieństwie do nazw atrybutów, w tym wypadku istnieje bardzo niewiele ograniczeń na wartości atrybutów. Wartości mogą zawierać białe znaki, zaczynać się od liczby lub zawierać dowolne znaki przestankowe (no, czasami poza cudzysłowem bądź apostrofem).

Wartości atrybutów XML ograniczone są cudzysłowem (pojedynczym lub podwójnym), ale w przeciwieństwie do HTML te cudzysłowy są obowiązkowe. Zwykle używane są cudzysłowy podwójne, ale czasami, jeśli wartość atrybutu taki cudzysłów ma zawierać, można do jego ograniczenia z kolei użyć cudzysłowu pojedynczego, na przykład:

```
<RURA SREDNICA='1"' DLUGOSC='10"'/>
```

Jeśli wartość atrybutu zawiera zarówno pojedynczy, jak i podwójny cudzysłów, żaden z nich nie mógłby być użyty w charakterze ogranicznika. W takiej sytuacji można użyć odwołań do encji – jednej, wybranej lub obu. Oto przykład:

```
<RAMKA DLUGOSC='8&apos;7&quot;' SZEROKOSC="10&apos;6&quot;"/>
```

Poprawnie sformułowane, samodzielne dokumenty XML

Choć możesz używać tylu znaczników, ilu dusza zapragnie, twoje dokumenty muszą być zgodne z pewnymi regułami, które określa się mianem *poprawnego sformułowania*. Jeśli dokument tych reguł nie spełnia, podczas próby jego czytania czy analizy zwykle pojawią się błędy.

Tak naprawdę sama specyfikacja XML zdecydowanie zabrania parserom XML próbować poprawiać i „rozumieć" dokumenty niepoprawnie sformułowane. Jedyne, co w takiej sytuacji może zrobić parser zgodny ze standardem, to zgłoszenie błędu, nie wolno mu natomiast tego błędu poprawić. Niedopuszczalne jest także pominięcie takiego błędnego oznakowania, po prostu należy błąd zgłosić i zakończyć swoje działanie.

> Celem tak restrykcyjnej polityki jest uniknięcie mnóstwa błędów, z którymi przeglądarki muszą sobie radzić, co wstrzymało rozwój HTML i uczyniły pisanie parserów tego języka zadaniem tak trudnym. Jako że przeglądarki sieciowe pozwalają używać nawet źle sformułowanego kodu HTML, projektanci stron nie wysilają się specjalnie, aby swoje HTML dostosować do wymagań standardu. Tak naprawdę nieraz zdarza się korzystanie z pewnych niedoróbek poszczególnych przeglądarek w celu osiągnięcia zamierzonego efektu. Aby prawidłowo obsłużyć ogromną ilość istniejących stron HTML, powstające przeglądarki muszą obsłużyć każdy szczegół realizowany we wszystkich przeglądarkach wcześniej istniejących. Użytkownicy po prostu zignorowaliby przeglądarki ściśle zgodne ze standardem HTML. Aby uniknąć tej przykrej sytuacji, od parserów XML wymaga się przetwarzania tylko i wyłącznie poprawnego kodu.

Aby dokument XML był poprawnie sformułowany, całe jego oznakowanie i wszystkie jego dane znakowe muszą być zgodne z opisanymi wcześniej w tym rozdziale wymaganiami. Co więcej, istnieje kilka dodatkowych reguł dotyczących zależności między znacznikami i danymi:

1. Każdy dokument zaczynaj deklaracją XML.
2. W przypadku niepustych elementów używaj znacznika początkowego i końcowego.

3. Kończ puste elementy stosując „/>”.
4. Jeden element musi całkowicie zawierać wszystkie inne.
5. Elementy nie mogą się przeplatać.
6. Umieszczaj wartości atrybutów w cudzysłowie.
7. Znaków < i & używaj tylko do otwierania znaczników i encji.
8. Używaj tylko pięciu encji predefiniowanych: `&`, `<`, `>`, `&apos;` i `&quot;`.

Tych osiem reguł musi być nieznacznie zmodyfikowanych w przypadku dokumentów z DTD; w tym wypadku są dodatkowe wymagania, zajmiemy się nimi w następnych rozdziałach. Obecnie przyjrzyjmy się tym podstawowym regułom dotyczącym dokumentów bez DTD.

DTD opisano w części drugiej.

Zasada 1.: Dokument musi zaczynać się deklaracją XML

W XML 1.0 deklaracja samodzielnego dokumentu wygląda następująco:

```
<?xml version="1.0" standalone="yes"?>
```

Jeśli deklaracja się pojawia w dokumencie, bezwzględnie musi być pierwszą rzeczą w pliku, gdyż procesor XML odczytuje pierwszych kilka bajtów pliku i porównuje je z różnymi postaciami napisu `<?xml` w celu określenia, jakiego zestawu znaków użyto (UTF-8, Unicode). Przed tym ciągiem nie powinno się absolutnie nic pojawić (no, może poza niewidocznym znacznikiem porządku bajtów), nawet spacje. Na przykład poniższy wiersz jest niedopuszczalny jako początek pliku XML, ma dodatkowe spacje (oznaczone tutaj jako `<SP>`):

```
<SP><SP><?xml version="1.0" standalone="yes"?>
```

UTF-8 i różne odmiany Unicode omówiono w rozdziale 7.

XML pozwala deklarację XML pominąć, ale zasadniczo praktyka ta nie jest zalecana. Czasami jednak jest to przydatne, na przykład pomijając tę deklarację możesz jeden poprawny dokument XML utworzyć z szeregu innych poprawnych dokumentów, którą to technikę omówimy dokładnie w rozdziale 9. Co więcej, pozwala to napisać poprawnie sformułowane dokumenty HTML, co omówimy dalej w tym rozdziale.

Zasada 2.: W przypadku niepustych elementów używaj znacznika początkowego i końcowego

Przeglądarki sieciowe są względnie wyrozumiałe dla twórców zapominających o końcowych znacznikach elementów HTML. Jeśli na przykład w dokumencie znajdzie się znacznik `<B>`, ale zabraknie odpowiadającego mu `</B>`, cały tekst dokumentu za tym znacznikiem zostanie pogrubiony. Co ważne, cały dokument i tak zostanie wyświetlony.

XML nie jest tak wyrozumiały. Każdy znacznik początkowy musi mieć odpowiadający mu znacznik końcowy. Jeśli w dokumencie jakiś znacznik pozostanie niedomknięty, przeglądarka czy parser po prostu zgłosi błąd i samego dokumentu nie pokaże w żaden sposób.

Zasada 3.: Kończ puste elementy „/>”

W HTML elementy nie zawierające danych, takie jak `<BR>`, `<HR>` i `<IMG>`, nie wymagają zamknięcia. Jednak w XML puste elementy muszą mieć znacznik zamknięty `/>`, a nie `>`. Na przykład odpowiednikami wymienionych znaczników w XML będą `<BR/>`, `<HR/>` i `<IMG/>`.

Istniejące obecnie przeglądarki sieciowe z takimi znacznikami postępują różnie. Jeśli chcesz zachować zgodność z poprzednimi wersjami, możesz użyć zwykłych znaczników zamykających i między znacznikiem otwierającym a zamykającym nie umieszczać żadnej treści, na przykład:

```
<BR></BR>
<HR></HR>
<IMG></IMG>
```

Nawet w tym przypadku jednak przeglądarka Netscape będzie miała kłopoty z `<BR></BR>` (oba znaczniki zostaną zinterpretowane jako nakaz skończenia wiersza, a nie tylko pierwszy z nich), więc nie zawsze praktyczne jest umieszczanie poprawnie sformułowanych znaczników w HTML.

Zasada 4.: Jeden element musi całkowicie zawierać wszystkie inne

Dokument XML ma element główny, który całkowicie zawiera wszystkie inne elementy dokumentu. Czasami mówimy w tym przypadku o elemencie dokumentu. Jeśli element główny nie jest pusty (co jest prawie zawsze prawdą), musi być ograniczony znacznikami początkowym i końcowym. Na przykład w poniższym przykładzie elementem głównym jest element `POZDROWIENIE`:

```
<?xml version="1.0" standalone="yes"?>
<POZDROWIENIE>
  Hello XML!
</POZDROWIENIE>
```

Deklaracja XML nie jest elementem, jest to instrukcja przetwarzania, więc nie musi być umieszczona w elemencie głównym. Podobnie inne dane nie będące elementami, DTD i komentarze, nie muszą być zawarte w elemencie głównym.

Zasada 5.: Elementy nie mogą się przeplatać

Elementy mogą zawierać (i bardzo często zawierają) inne elementy, jednak nie mogą na siebie zachodzić. Jeśli dany element zawiera znacznik początkowy innego elementu, musi także zawierać odpowiadający mu znacznik końcowy. Podobnie, element nie może zawierać elementu końcowego, jeśli nie zawiera odpowiadającego mu znacznika początkowego. Poniższy kod jest w XML dopuszczalny:

```
<PRE><KOD>n = n + 1;</KOD></PRE>
```

Z kolei poniższy fragment jest niedopuszczalny, gdyż znacznik kończący `PRE` znajduje się przed końcowym znacznikiem `KOD`:

```
<PRE><KOD>n = n + 1;</PRE></KOD>
```

Większość przeglądarek HTML z taką konstrukcją poradzi sobie bez problemu, natomiast przeglądarki XML zgłoszą w takim wypadku błąd.

Oczywiście elementy puste mogą pojawiać się gdziekolwiek, na przykład:

```
<DRAMATURDZY>Oscar Wilde<HR/>Joe Orton</DRAMATURDZY>
```

Omawiana zasada w połączeniu z zasadą 4 wymaga, aby wszystkie elementy nie będące elementem głównym zawarte były w jednym takim elemencie, jest „najmniejszym" elementem je zawierającym – ten element bezpośrednio nadrzędny nazywamy *rodzicem* omawianego elementu nie głównego. Z kolei element zawarty to *dziecko* owego rodzica. Zatem każdy element, oprócz głównego, ma jednego rodzica, może mieć natomiast dowolnie wiele elementów dzieci, może też nie mieć ich wcale.

Rozważ poniższy przykład z wydruku 6.1. Element główny to `DOKUMENT`. Zawiera dwa elementy dzieci `STAN`. Pierwszy element `STAN` zawiera czworo dzieci: `NAZWA`, `DRZEWO`, `KWIAT` i `STOLICA`. Drugi element `STAN` zawiera tylko troje dzieci: `NAZWA`, `DRZEWO` i `STOLICA`. Każdy z elementów dzieci zawiera tylko dane znakowe, nie ma już dalszych dzieci.

Wydruk 6.1. *Rodzice i dzieci*

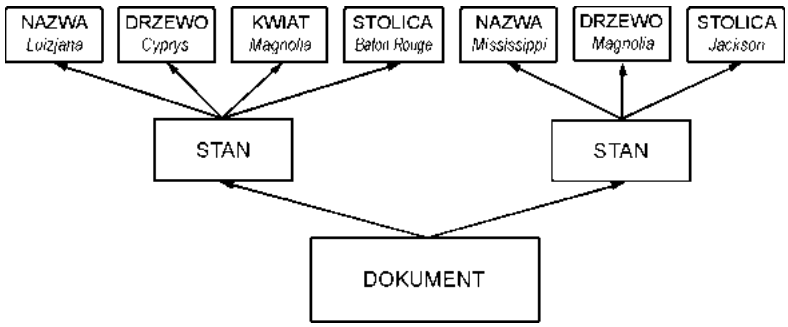
```
<?xml version="1.0" standalone="yes"?>
<DOKUMENT>
  <STAN>
    <NAZWA>Luizjana</NAZWA>
    <DRZEWO>Cyprys</DRZEWO>
    <KWIAT>Magnolia</KWIAT>
    <STOLICA>Baton Rouge</STOLICA>
  </STAN>
```

```
  <STAN>
    <NAZWA>Mississippi</NAZWA>
    <DRZEWO>Magnolia</DRZEWO>
    <STOLICA>Jackson</STOLICA>
  </STAN>
</DOKUMENT>
```

Z punktu widzenia programisty oznacza to, że dokument XML ma postać drzewa. Na rysunku 6.1 pokazano drzewiastą strukturę dokumentu z wydruku 6.1, wyraźnie też widać, skąd nazwa „drzewo". Zaczynamy od elementu głównego na dole i stopniowo rozgałęziamy, aż po liście, które już nie mają elementów potomnych (dzieci).

Struktura drzewa jest łatwa do analizy programowej, choć dla ciebie, jako autora dokumentów, ma to akurat niewielkie znaczenie.

Rysunek 6.1. *Struktura drzewiasta z wydruku 6.1*



Drzewa zwykle rysuje się od góry do dołu, tutaj zrobiliśmy odwrotnie. Dzięki temu nasz obrazek bardziej przypomina prawdziwe drzewo, nie ma to jednak żadnego wpływu na samą strukturę.

Zasada 6.: Umieszczaj wartości atrybutów w cudzysłowie

XML wymaga, aby wszystkie wartości atrybutów były ujęte w cudzysłowy, niezależnie od tego, czy zawierają białe znaki, czy też nie, na przykład:

```
<A HREF="http://metalab.unc.edu/xml/">
```

W HTML nie jest to prawdą; na przykład HTML pozwala używać atrybutów bez ujmowania ich w cudzysłów. Wobec tego poniższy przykład opisuje prawidłowy w HTML znacznik <A>:

```
<A HREF=http://metalab.unc.edu/xml/>
```

Jedynym ograniczeniem jest to, że wartość atrybutu nie może zawierać żadnych spacji.

Jeśli wartość atrybutu sama zawiera podwójny cudzysłów, możesz tę wartość ująć w cudzysłów pojedynczy, na przykład:

```
<IMG SRC="sistinechapel.jpg"
     alt='I rzekł Bóg: "Niech się stanie światłość",
          i stała się światłość'/>
```

Jeśli wartość atrybutu zawiera cudzysłowy pojedyncze i podwójne, możesz użyć odwołań do encji `&apos;` dla cudzysłowu pojedynczego (apostrof) i `&quot;` dla cudzysłowu podwójnego, na przykład:

```
<PARAM nazwa="zdanie" wartosc="cudzysłów pojedynczy to &apos;,
       podwójny to &quot;">
```

Zasada 7.: Znaków < i & używaj tylko do otwierania znaczników i encji

XML zakłada, że każdy otwierający nawias kątowy (czyli znak mniejszości) zaczyna znacznik, a każdy znak & zaczyna odwołanie się do encji (często jest to prawda także w przypadku HTML, ale jeśli średnik pominiesz w HTML, to przeglądarka sama się go domyśli). Rozważmy na przykład poniższy kod:

```
<H1>Firma Meyer & Meyer</H1>
```

Przeglądarka sieciowa rzecz prawdopodobnie poprawnie wyświetli kod, ale w celu zapewnienia maksymalnego bezpieczeństwa zamiast znaku & powinieneś użyć odwołania do odpowiedniej encji:

```
<H1>Firma Meyer &amp; Meyer</H1>
```

Podobnie ma się z otwierającym nawiasem kątowym. Rozważ poniższy typowy fragment kodu języka Java:

```
<KOD> for (int i = 0; i <= args.length; i++ ) { </KOD>
```

Zarówno w XML, jak i w HTML, znak < w <= uznany zostanie za początek znacznika. Znacznik ciągnie się aż do najbliższego znaku >, zatem powyższy wiersz zostanie zinterpretowany jako

```
for (int i = 0; i
```

zamiast jako

```
for (int i = 0; i <= args.length; i++ ) {
```

Jednak `= args.length; i++ ) {` zinterpretowane zostanie jako część nierozpoznanego znacznika.

Zarówno w HTML, jak i w XML, znak mniejszości zapisać można jako `<`, na przykład:

```
<KOD> for (int i = 0; i &lt;= args.length; i++ ) { </KOD>
```

Poprawnie sformułowany XML wymaga, aby `&` zapisywać jako `&` i < jako `<`, aby uniknąć interpretowania ich jako znaczników lub odwołań do encji.

Zasada 8.: Używaj tylko pięciu encji predefiniowanych

Zapewne szereg odwołań do encji znasz z HTML, na przykład `&copy;` wstawia znak ochrony praw autorskich ©, ® wstawiam symbol zastrzeżenia znaku ®. W przeciwieństwie do tego, w XML można użyć tylko pięciu encji wyliczonych wcześniej, wszystkie inne trzeba najpierw zadeklarować w DTD.

O DTD jeszcze nic nie wiesz. Jeśli gdziekolwiek w twoim dokumencie pojawi się znak &, zaraz po nim powinien następować jeden z napisów: `amp;`, `lt;`, `gt;`, `apos;` lub `quot;`. Wszystkie inne sposoby użycia są pogwałceniem poprawności sformułowania.

W rozdziale 9. nauczysz się, jak DTD umożliwia definiowanie nowych encji w celu odwoływania się do poszczególnych symboli lub fragmentów tekstu traktowanych jako szablony.

Poprawnie sformułowane HTML

XML możesz ćwiczyć, jeszcze zanim przeglądarki zaczną go bezpośrednio obsługiwać – wystarczy, że będziesz pisał poprawnie sformułowany kod HTML. Oznacza to, że HTML jest zgodny z wymaganiami XML, tyle że używa swoich specyficznych znaczników. Poprawnie sformułowany HTML jest znacznie czytelniejszy od niechlujnego HTML, tworzonego przez większość ludzi za pomocą narzędzi typu FrontPage. Łatwiej też takie dokumenty automatycznie przetwarzać, a wyszukiwarki lepiej je rozumieją. Dokumenty są po prostu porządniejsze, zmniejsza się też prawdopodobieństwo, że przy jakiejś zmianie wszystko się „posypie". Zmniejsza się też ryzyko, że wszystko będzie się zmieniać przy zmianie przeglądarki lub platformy. Co więcej, prawidłowo sformułowane dokumenty HTML mogą być przetwarzane narzędziami XML, a będą one i tak czytelne dla starszych przeglądarek, nie obsługujących XML.

Problemy z istniejącymi stronami sieciowymi

Na prawdziwych stronach sieciowych panuje wyjątkowy bałagan. Znaczniki nie są zamykane, elementy się przeplatają, używane są bezpośrednio znaki mniejszości, na końcu encji pomijane są średniki. Tego typu strony formalnie są niepoprawne, ale większość przeglądarek i tak je obsługuje. Jednak jeśli twoje strony będą bardziej poprawne, to będą się szybciej wyświetlać i łatwiej będzie je utrzymać oraz rozwiązywać pojawiające się problemy.

Oto zestaw typowych problemów dotyczących stron Sieci:

1. Brak znaczników końcowych (nie domknięte elementy).
2. Znaczniki końcowe bez znaczników początkowych.
3. Elementy na siebie zachodzą.

4. Wartości atrybutów nie ujęte w cudzysłowy.
5. Bezpośrednio wpisywane znaki <, >, & i ".
6. Brak elementu głównego.
7. Różna wielkość liter w znaczniku początkowym i końcowym.

Kolejność powyższych punktów z grubsza wyznacza, na ile poszczególne zagadnienia są istotne. Szczegóły zmieniają się w zależności od tego, o jaki znacznik chodzi. Na przykład brak znacznika końcowego <STRONG> spowoduje wyróżnienie (pogrubienie) wszystkich dalszych elementów, natomiast brak znacznika końcowego <LI> czy <P> w ogóle nie stanowi problemu.
Niektóre reguły z kolei dotyczą jedynie dokumentów XML, w przypadku zastosowania ich do stron HTML mogą wystąpić problemy. Są to:

1. Każdy dokument zaczynaj deklaracją XML.
2. Kończ puste elementy stosując „/>".
3. Używaj tylko pięciu encji predefiniowanych: &, <, >, ' i ".

Rozwiązanie wszystkich tych problemów nie jest skomplikowane, jest jednak kilka pułapek czyhających na nieostrożnych. Przyjrzyjmy się temu dokładniej.

Zamykaj wszystkie znaczniki początkowe

Każdy element z jakąkolwiek zawartością, czy to tekstem, czy elementami potomnymi, powinien mieć znacznik początkowy i znacznik końcowy. HTML nie przestrzega tego tak bezwzględnie. Na przykład znaczniki <P>, <DT>, <DD> i <LI> często używane są pojedynczo. Jednak robiąc tak polegamy na tym, że przeglądarka prawidłowo odgadnie położenie końca takiego elementu, natomiast przeglądarki nie zawsze są w stanie dobrze zinterpretować zamiary bądź oczekiwania autora strony. Wobec tego najlepiej zamykać elementy.

Największa zmiana, jaką będziesz musiał zrobić, to myślenie o elemencie <P> jako o swojego rodzaju pojemniku, a nie jako o zwykłym znaku rozpoczęcia nowego akapitu. Na przykład dawniej tworząc stronę o historii najnowszej, przemówienie Władysława Gomułki na wiecu ludności Warszawy z października 1956 roku oznakowałbyś mniej więcej tak:

```
Towarzysze! Obywatele! Ludu Pracujący Stolicy!
<P>
Witam Was w imieniu Komitetu Centralnego Polskiej Zjednoczonej
Partii Robotniczej, który na swoim ostatnim plenarnym posiedzeniu
oddał ster partii w ręce nowego kierownictwa.
<P>
W ciągu ubiegłych lat nagromadziło się w życiu Polski wiele zła,
nieprawości i bolesnych rozczarowań. Idee socjalizmu, przeniknięte
duchem wolności człowieka i poszanowania praw obywatela, w praktyce
uległy głębokim wypaczeniom. Słowa nie znajdowały pokrycia w
rzeczywistości. Ciężki trud klasy robotniczej i całego narodu nie
dawał oczekiwanych owoców.
```

```
<P>
VIII Plenum KC naszej partii dokonało historycznego zwrotu.
Stworzyło ono nowy okres naszej pracy, nowy okres w dziejach
budownictwa socjalistycznego w Polsce, w dziejach narodu.
<P>
```

Poprawne sformułowanie z kolei wymaga użycia następującego oznakowania:

```
<P>
Towarzysze! Obywatele! Ludu Pracujący Stolicy!
</P>
<P>
Witam Was w imieniu Komitetu Centralnego Polskiej Zjednoczonej
Partii Robotniczej, który na swoim ostatnim plenarnym posiedzeniu
oddał ster partii w ręce nowego kierownictwa.
</P>
<P>
W ciągu ubiegłych lat nagromadziło się w życiu Polski wiele zła,
nieprawości i bolesnych rozczarowań. Idee socjalizmu, przeniknięte
duchem wolności człowieka i poszanowania praw obywatela, w praktyce
uległy głębokim wypaczeniom. Słowa nie znajdowały pokrycia w
rzeczywistości. Ciężki trud klasy robotniczej i całego narodu nie
dawał oczekiwanych owoców.
</P>
<P>
VIII Plenum KC naszej partii dokonało historycznego zwrotu.
Stworzyło ono nowy okres naszej pracy, nowy okres w dziejach
budownictwa socjalistycznego w Polsce, w dziejach narodu.
</P>
```

Prawdopodobnie nauczono cię myśleć o `<P>` jako o końcu akapitu. Teraz musisz myśleć o nim jako o początku akapitu. Daje ci to jednak pewne dodatkowe korzyści, na przykład łatwo możesz akapitowi przypisać szereg dodatkowych atrybutów formatowania; oto początek powyższego tekstu już nieco sformatowany:

```
<center>
<p><h2>dokumenty po VIII Plenum KC PZPR (19-21 X 1956r.)</h2>

<p><h1>Przemówienie Władysława Gomułki na wiecu ludności
Warszawy</h1>

<p>[Październik 1956 roku]
<p><b>Towarzysze! Obywatele! Ludu Pracujący Stolicy!</b>
<p>Witam Was w imieniu Komitetu Centralnego Polskiej Zjednoczonej
Partii Robotniczej, który na swoim ostatnim plenarnym posiedzeniu
oddał ster partii w ręce nowego kierownictwa.
</center>
```

Oto ten sam tekst, ale tym razem już poprawnie sformułowany. Atrybut `align` zastępuje teraz wyklinany znacznik `center`, natomiast atrybut stylu CSS zastępuje znacznik `<b>`.

```
<h2 align="center">dokumenty po VIII Plenum KC PZPR
(19-21 X 1956r.)</h2>

<h1 align="center">Przemówienie Władysława Gomułki na wiecu ludności
Warszawy</h1>
```

```
<p align="center">[Październik 1956 roku]</p>
<p align="center" style="font-weight:bold">Towarzysze! Obywatele!
Ludu Pracujący Stolicy!</p>
<p align="center">Witam Was w imieniu Komitetu Centralnego Polskiej
Zjednoczonej Partii Robotniczej, który na swoim ostatnim plenarnym
posiedzeniu oddał ster partii w ręce nowego kierownictwa.</p>
```

Usuwaj osamotnione znaczniki końcowe, nie pozwól elementom na siebie zachodzić

Podczas edytowania nieraz zdarza się, że usuwasz znacznik początkowy i zapominasz usunąć odpowiadający mu znacznik końcowy. W HTML takie osierocone znaczniki, jak `</STRONG>` czy `</TD>` raczej nie powodują żadnych problemów. Jednak przez nie plik jest dłuższy niż być musi, wolniej się ładuje i może sprawić kłopoty ludziom i narzędziom analizującym go i edytującym. Wobec tego zawsze powinieneś się upewnić, że znaczniki końcowe odpowiadają znacznikom początkowym.

Częściej jednak zdarza się, że z powodu dodatkowego znacznika końcowego inne znaczniki zaczynają się przeplatać. W większości wypadków tego typu błędy łatwo jest naprawić, rozważmy następujący, typowy błąd:

```
<B><I>Ten tekst jest pisany pogrubioną kursywą.</B></I>
```

Jako że element `I` zaczyna się wewnątrz elementu `B`, powinien się też wewnątrz niego skończyć. W tym wypadku wystarczy, że zmienisz kolejność tych elementów:

```
<I><B>Ten tekst jest pisany pogrubioną kursywą.</B></I>
```

Czasami możesz mieć poważniejszy problem, spójrz na fragment strony Białego Domu (*http://www.whitehouse.gov/*) z 4 października 1998 roku. Niepoprawne znaczniki wyróżniłem pogrubiłem.

```
<TD valign=TOP width=85>
<FONT size=+1>
<A HREF="/WH/New"><img border=0 src="/WH/images/pin_calendar.gif"
align=LEFT height=50 width=75 hspace=5 vspace=5</A><br> </TD>
<TD valign=TOP width=225>
<A HREF="/WH/New"><B>What's New:</B></A><br>
</FONT>
What's happening at the White <nobr>House - </nobr><br>
 <font size=2><b>
<!-- New Begin -->
<a href="/WH/New/html/19981104-12244.html">Remarks Of The President
Regarding Social Security</a>
<BR>
<!-- New End -->
 </font>
</b>
</TD>
```

W tym przykładzie element `<FONT size=+1>` zaczyna się w pierwszym elemencie `<TD valign=TOP width=85>`, ale wychodzi poza ten element, aby się skończyć w `<TD valign=TOP width=225>`. Prawidłowe rozwiązanie polega na zamknięciu

elementu FONT przed pierwszym znacznikiem zamykającym </TD>, a później dodanie kolejnego <FONT size=+1> zaraz po otwarciu kolejnego elementu TD, jak to pokazano poniżej:

```
<TD valign=TOP width=85>
<FONT size=+1>
<A HREF="/WH/New"><img border=0 src="/WH/images/pin_calendar.gif"
align=LEFT height=50 width=75 hspace=5 vspace=5</A><br>
</FONT></TD>
<TD valign=TOP width=225>
<FONT size=+1>
<A HREF="/WH/New"><B>What's New:</B></A><br>
</FONT>
What's happening at the White <nobr>House - </nobr><br>
 <font size=2><b>
<!-- New Begin -->
<a href="/WH/New/html/19981104-12244.html">Remarks Of The President
Regarding Social Security</a>
<BR>
<!-- New End -->
 </font>
</b>
</TD>
```

Wszystkie atrybuty ujmuj w cudzysłów

Atrybuty HTML muszą być ujęte w cudzysłów tylko wtedy, gdy zawierają białe znaki. Nic jednak nie zaszkodzi, jeśli będziesz stosował je zawsze. Co więcej, przyda się to w przyszłości, jeśli zdecydujesz się zmieniać wartość atrybutu na coś, co będzie zawierało na przykład spacje. Później łatwo jest zapomnieć o cudzysłowie, szczególnie jeśli jest to na przykład atrybut ALT znacznika <IMG>, gdyż błędne sformułowanie tego znacznika nie będzie w przeglądarce zbytnio widoczne.

Rozważmy na przykład następujący znacznik <IMG>:

```
<IMG SRC=cup.gif WIDTH=89 HEIGHT=67 ALT=Cup>
```

Należy go zmienić następująco:

```
<IMG SRC="cup.gif" WIDTH="89" HEIGHT="67" ALT="Cup">
```

Cytuj znaki <, > i &

HTML jest bardziej wyrozumiały niż XML, jeśli chodzi o używanie znaków mniejszości i &. Jednak nawet w czystym HTML znaki te mogą sprawić kłopoty, szczególnie jeśli zaraz za nimi występuje jakiś inny znak. Na przykład rozważ adres poczty elektronicznej w takiej postaci, w jakiej kopiowany jest z sekcji From: programu Eudora:

```
Elliotte Rusty Harold <elharo@metalab.unc.edu>
```

Po przeanalizowaniu tego przez parser XML otrzymamy:

```
Elliotte Rusty Harold
```

Tak więc sam adres `<elharo@metalab.unc.edu>` zostanie przypadkowo ukryty przez nawiasy kątowe. Za każdym razem, kiedy chcesz umieścić bezpośrednio znak mniejszości lub & w HTML, powinieneś używać encji `&lt;` i `&amp;`. Poprawnie zapisany powyższy kod powinien więc wyglądać następująco:

```
Elliotte Rusty Harold &lt;elharo@metalab.unc.edu&gt;
```

Mniejsze jest prawdopodobieństwo, że napotkasz problemy w przypadku znaku większości. Problemy mogą pojawić się tylko wtedy, gdy znak ten pojawi się w nie domkniętym znaczniku. Jednak w dokumencie takie znaczniki mogą się pojawiać, wtedy znajdujący się w pobliżu znak większości może je niepoprawnie zakończyć. Na przykład rozważmy następujący fragment kodu Java:

```
for (int i=0;i<10;i++) {
  for (int j=20;j>10;j-) {
```

Kod ten prawdopodobnie zostanie zinterpretowany następująco:

```
for (int i=0;i10;j-) {
```

Jeśli są to tylko dwa takie wiersze w stuliniowym programie, jest duże prawdopodobieństwo, że ten błąd przegapisz. Z drugiej strony, jeśli znak większości zostanie zacytowany, znak mniejszości zakryje całe zakończenie programu i błąd będzie łatwiej wykryć.

Używaj elementu głównego

Elementem głównym w dokumencie HTML powinno być `html`. Większość przeglądarek jednak brak tego elementu wybaczy. Tym niemniej jednak zdecydowanie lepiej użyć jako pierwszego znacznika `<html>` i jako ostatniego `</html>`. Jeśli przed `<html>` lub za `</html>` pojawi się jakiś tekst lub dodatkowe znaczniki, przenieś je do wewnątrz elementu `html`.

Powszechną praktyką jest na przykład pomijanie kończącego `</html>` za dokumentem. Ja zawsze zaczynam pisanie dokumentów HTML od wstawienia obu opisywanych znaczników, nie odkładam tego na koniec, bo mogę zapomnieć.

Do wpisywania znaczników używaj stałej wielkości liter

W HTML wielkość liter nie ma znaczenia, w przeciwieństwie do XML. Zalecam wybrać jedną wielkość liter i już dalej w całym dokumencie się jej trzymać. Zwykle wybieram małe litery, gdyż są one łatwiejsze do wpisywania. Co więcej, W3C podczas ponownego tworzenia specyfikacji HTML jako aplikacji XML użyło takiej właśnie konwencji.

W rozdziale 20. zajmiemy się szczegółowo specyfikacją HTML jako aplikacji XML. Na razie jednak będziemy musieli zaczekać z dalszym badaniem, gdyż wymaga to użycia kilku technik XML, które poznasz dopiero później.

Zamykaj puste znaczniki stosując />

Puste znaczniki to przekleństwo dla tych, którzy chcą konwertować HTML na poprawnie sformułowane dokumenty XML. HTML formalnie nie dopuszcza używania składni `<element/>` do opisywania elementów pustych. Możesz bez problemu zmienić `<br>` na `<br/>`, `<hr>` na `<hr/>`, `<img>` na `<img/>` i tak dalej. Jednak zagadką pozostanie, czy używana akurat przeglądarka zechce taki znacznik poprawnie zinterpretować, czy nie.

> Nie myl elementów naprawdę pustych, takich jak <br>, <hr> czy <img> z elementami, które mają zawartość, ale często pojawia się jedynie ich znacznik początkowy, jak <p>, <li>, <dt> i <dd>.

Rozwiązaniem najprostszym, a przy tym jedynym dopuszczonym przez specyfikację XML, jest zamiana pustych znaczników na pary znacznika otwierający/zamykający bez zawartości. Przeglądarka powinna znacznik końcowy, jako nieznany, po prostu zignorować. Spójrz na następujący przykład:

```
<br></br>
<hr></hr>
<IMG SRC="cup.gif" WIDTH="89" HEIGHT="67" ALT="Cup"></IMG>
```

Wydaje się, że rzecz działa poprawnie, poza jednym, acz znaczącym wyjątkiem: w Netscape 4.5 i poprzednich `</br>` traktowane jest tak samo, jak `<br>`, zatem pojawiają się dwa znaczniki nakazujące zakończenie wiersza, co wizualnie daje wynik podobny do nowego akapitu. Co więcej, Netscape całkowicie ignoruje `<br/>`. Strony wyświetlane w starych przeglądarkach nie mogą zatem używać ani konstrukcji `<br></br>`, ani `<br/>`. W praktyce, w przeglądarkach i w XML działa jedynie

```
<br />
```

Zwróć uwagę na spację między `<br` a `/>`. Tak naprawdę nie wiem, czemu to działa, skoro nie działają wersje bardziej naturalne. Jest to jedyne, co mogę Ci zaproponować, jeśli naprawdę zależy ci na poprawnie sformułowanym HTML.

Używaj tylko encji &, <, >, ' oraz "

Na wielu stronach użycie jakichkolwiek encji poza wymienionymi w nagłówku jest zbędne. Jednak w HTML 4.0 zdefiniowano więcej encji, w tym:

- `&trade;` – znak handlowy (™)
- `&copy;` – symbol ochrony praw autorskich (©)
- `&infin;` – symbol nieskończoności (∞)
- `&pi;` – grecka mała litera pi (π)

Istnieją jeszcze setki innych encji, jednak używanie zbyt wielu z nich uczyni twój dokument źle sformułowanym. Prawdziwym rozwiązaniem tego problemu jest użycie DTD. Użycie DTD do encji omówione zostanie w rozdziale 9n, a na razie podamy kilka rozwiązań tymczasowych.

Najprostsza metoda to pisanie dokumentu za pomocą takiego zestawu znaków, który zawiera wszystkie potrzebne symbole, a następnie podanie tego zestawu w dyrektywie <META>. Aby na przykład nakazać użycie do wyświetlania dokumentu zestawu UTF-8 (zestaw ten omówimy w rozdziale 7., zawiera on wszystkie znaki, jakich możesz potrzebować), należy użyć w nagłówku dokumentu (HEAD) następującej postaci dyrektywy META:

```
<META http-equiv="Content-Type"
      content="text/html; charset=UTF-8">
```

Z drugiej strony, możesz po prostu nakazać swojemu serwerowi sieciowemu generować odpowiedni nagłówek typu, choć zwykle łatwiej jest użyć znacznika <META>.

```
Content-Type: text/html; charset=UTF-8
```

Problem w tym wypadku polega na tym, że wiele przeglądarek może nie być w stanie prawidłowo wyświetlać zestawu znaków UTF-8. To samo dotyczy także większości zestawów znaków, które zawierają różne znaki specjalne.

HTML 4.0 obsługuje encje znakowe dokładnie tak samo, jak XML, czyli każdy znak możesz zastąpić ciągiem &# i dalej dziesiętnym lub szesnastkowym kodem tego znaku w Unicode, na przykład:

- ™ – znak handlowy (™)
- © – symbol ochrony praw autorskich (©)
- ∞ – symbol nieskończoności (∞)
- π – grecka mała litera pi (π)

HTML 3.2 oficjalnie obsługuje jedynie odwołania do znaków o kodach od 0 do 255 (ISO Latin-1), ale wersja 4.0 i nowsze wersje Netscape Navigatora i Internet Explorera rozpoznają znacznie szerszy zestaw znaków Unicode.

Jeśli naprawdę bardzo zależy ci na poprawnie sformułowanym kodzie XML zgodnym wstecz z HTML, możesz odpowiednie znaki wstawić jako elementy graficzne typu inline (włączane w wiersz), na przykład:

- <img src="tm.gif" width="12" height="12" alt="TM"></img> – znak handlowy (™)
- <img src="copyright.gif" width="12" height="12" alt="(C)"></img> – symbol ochrony praw autorskich (©)
- <img src="infinity.gif" width="12" height="12" alt="(nieskończoność)"></img> – symbol nieskończoności (∞)
- <img src=pi.gif" width="12" height="12" alt="liczba pi)"></img> – grecka mała litera pi (π)

Praktycznie takiej metody jednak nie polecam. Poprawność sformułowania nie jest w HTML tak ważna, aby usprawiedliwiać dodatkowe ładowanie i wyświetlanie tego typu obrazków.

Deklaracja XML

Dokumenty HTML nie potrzebują deklaracji XML, mogą jednak ją mieć. Przeglądarki po prostu zignorują nieznane sobie znaczniki. Z ich punktu widzenia poniższy wiersz to po prostu kolejny, nieznany znacznik:

```
<?xml version="1.0" standalone="yes"?>
```

Jako że przeglądarki nie rozumiejące XML nie rozumieją także znacznika `<?xml?>`, bez dyskusji zignorują go. Przeglądarki z kolei rozumiejące XML taki dokument zinterpretują jako poprawnie sformułowany dokument XML i jako taki będą go traktować.

Niestety, przeglądarki nie w pełni rozumiejące XML mają z taką składnią problemy. W szczególności Internet Explorer 4.0 w wersji dla komputera Mac potraktuje to jako żądanie załadowania pliku, a nie wyświetlenia go (nie dotyczy to Netscape Navigatora ani innych wersji IE). Wobec tego osobiście usunąłem deklarację XML ze swoich stron.

Postępuj zgodnie z zasadami

Nie jest specjalnie trudno pisać poprawnie sformułowane dokumenty XML, zgodne z opisanymi w tym rozdziale zasadami. Jednak przeglądarki XML są znacznie mniej wyrozumiałe niż przeglądarki HTML, więc musisz być uważny.

Jeśli naruszysz zasady poprawnego sformułowania, parsery i przeglądarki XML zgłoszą błąd składniowy. Wobec tego proces pisania kodu XML może być trochę podobny do pisania w jakimś języku programowania: najpierw piszesz, później kompilujesz i jeśli kompilacja się nie powiedzie, sprawdzasz znalezione błędy i poprawiasz je.

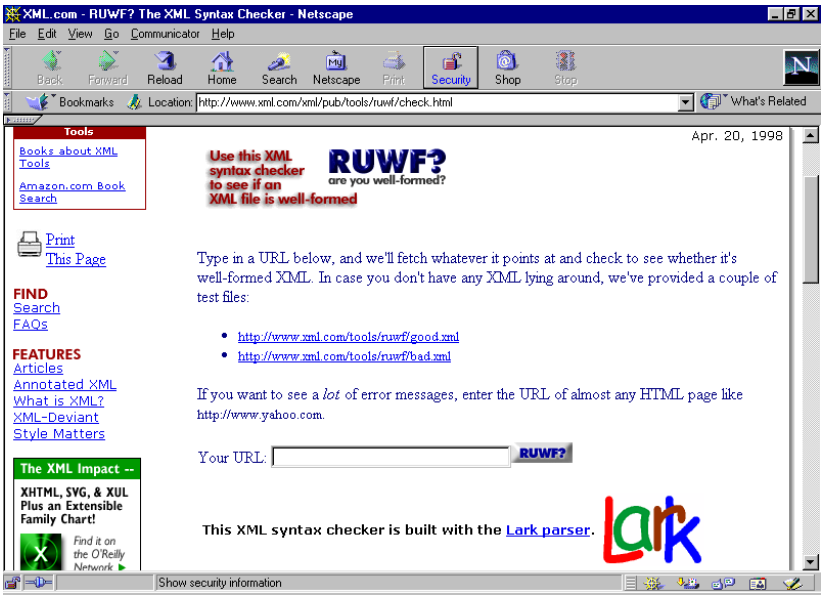
W zasadzie jest to proces iteracyjny, w którym przechodzisz przez kolejne cykle edycji i kompilacji, zanim uzyskasz gotowy dokument. Mimo to ręczne pisanie XML jest znacznie prostsze od pisania kodu języka C czy Java, a kiedy nabędziesz nieco praktyki, będziesz otrzymywał naprawdę niewiele błędów i kod XML będziesz w stanie pisać niemalże tak szybko, jak szybko potrafisz pisać na klawiaturze.

Narzędzia do czyszczenia kodu HTML

Istnieje kilka narzędzi, które pomogą ci oczyścić swoje strony, a najbardziej godnymi uwagi są RUWF z XML.COM i HTML Tidy od Dave'a Raggetta z W3C.

RUWF

Każde narzędzie sprawdzające poprawność sformułowania dokumentów XML może też sprawdzić pod tym samym względem dokumenty HTML. Jednym z najłatwiejszych w użyciu narzędzi jest RUWF z XML.COM. Na rysunku 6.2 pokazano ekran główny tego testera. Po prostu podajesz adres URL strony, którą chcesz sprawdzać, a RUWF zwróci ci stronę z tuzinami komunikatów o błędach.



Rysunek 6.2. *Kontroler poprawności sformułowania RUWF*

Oto pierwszy zestaw błędów, jakie RUWF znalazł na stronie domowej Białego Domu. Większość z nich dotyczy błędnie sformułowanego XML, które jednak w HTML jest dopuszczalne (choć nie należy do dobrego stylu programowania). Jednak co najmniej jeden z błędów („Line 55, column 30: Encountered </FONT> with not start-tag.") stanowi problem tak dla XML, jak i HTLM.

```
Line 28, column 7: Encountered </HEAD> expected </META>
...assumed </META> ...assumed </META> ...assumed </META>
...assumed </META>
Line 36, column 12, character '0': after AttrName= in start-tag
Line 37, column 12, character '0': after AttrName= in start-tag
Line 38, column 12, character '0': after AttrName= in start-tag
Line 40, column 12, character '0': after AttrName= in start-tag
Line 41, column 10, character 'A': after AttrName= in start-tag
Line 42, column 12, character '0': after AttrName= in start-tag
Line 43, column 14: Encountered </CENTER> expected </br>
...assumed </br> ...assumed </br>
Line 51, column 11, character '+': after AttrName= in start-tag
Line 52, column 51, character '0': after AttrName= in start-tag
Line 54, column 57: after &
Line 55, column 30: Encoutnered </FONT> with not start-tag.
Line 57, column 10, character 'A': after AttrName= in start-tag
Line 59, column 15, character '+': after AttrName= in start-tag
```

HTML Tidy

Kiedy już problemy zidentyfikowałeś, czas je naprawić. Wiele typowych problemów, na przykład wstawianie wartości atrybutów w cudzysłów, można rozwiązać automatycznie. Najwygodniejszym narzędziem do robienia tego jest program HTML Tidy Dave'a Raggetta działający w trybie wiersza poleceń. Tidy to program działający w trybie tekstowym napisany w ANSI C, dzięki czemu można go kompilować i uruchamiać na większości platform, w tym Windows, Unix, BeOS i Mac.

Tidy znajduje się na załączonej płycie CD w katalogu *utilities/tidy*. Załączono binaria dla Windows NT i BeOS, umieszczono także przenośny kod źródłowy na wszystkie platformy. Najnowszą wersję tego programu znajdziesz pod adresem *http://www.w3.org/People/Raggett/tidy/*.

Tidy czyści pliki HTML na kilka sposobów, z których nie wszystkie zgodne są z zasadami poprawności XML. Taka naprawdę domyślnie Tidy usuwa wszystkie zbędne dla HTML (ale nie dla XML) znaczniki końcowe, jak `</LI>`, robi też szereg innych rzeczy, które psują poprawność sformułowania. Możesz jednak użyć przełącznika `-asxml` nakazującego dawać wyniki w postaci poprawnie sformułowanego XML. Aby na przykład przerobić plik *index.html* na poprawnie sformułowany XML, w oknie poleceń DOS wpisz:

```
C:\> tidy -m -asxml index.html
```

Flaga `-m` nakazuje robić konwersję przetwarzanego pliku. Flaga `-asxml` nakazuje dać wyniki zgodne ze standardem języka XML.

Podsumowanie

W tym rozdziale dowiedziałeś się, jak tworzyć poprawnie sformułowane XML. W szczególności dowiedziałeś się, że:

- Dokumenty XML to ciągi znaków spełniające pewne kryteria poprawności sformułowania.
- Tekst dokumentów XML dzieli się na dane znakowe i znaczniki.
- Komentarze pozwalają dokumentować kod. Możesz też użyć komentarzy do czasowego wyłączenia pewnych, jeszcze nie gotowych sekcji dokumentu.
- Użycie encji pozwala ci wstawić do dokumentu takie znaki, jak <, >, &, " i '.
- Sekcje CDATA są użyteczne do wstawiania tekstu zawierającego mnóstwo znaków <, > i &.
- Znacznik to wszystko, co zaczyna się od < i kończy >, a nie znajduje się w sekcji CDATA.
- Znaczniki początkowe i znaczniki elementów pustych mogą zawierać atrybuty, które opisują elementy.
- Dokumenty HTML mogą także być poprawnie sformułowane, wymaga to jednak dodatkowej pracy.

W następnym rozdziale zajmiemy się pisaniem XML w językach innych niż angielski, takich jak na przykład arabski, chiński i grecki.