Teknologi Web Service

SOA dengan XML

Toni Wijanarko Adi Putra, M.Kom

ISBN 978-623-6141-27-4

Teknologi Web Service

SOA

dengan

XML

Oleh : Toni Wijanarko Adi Putra., M.Kom.

ISBN 978-623-6141-27-4

Teknologi Web Service SOA dengan XML

Penulis:
Toni Wijanarko Adi Putra., M.Kom.

ISBN : 978-623-6141-27-4

Editor:
Indra Ava Dianta, S.Kom., M.T

Penyunting :
Zaenal Mustofa, S.Kom.,M. Kom

Desain Sampul dan Tata Letak :
Hendri Rasminto, S.Kom.,M.Si

Penerbit :
Yayasan Prima Agus Teknik
Redaksi: Jln Majapahit No 605 Semarang
Tlpn. (024) 6723456
Fax . 024-6710144
Email: penerbit_ypat@stekom.ac.id

Distributor Tunggal:
UNIVERSITAS STEKOM
Jln Majapahit No 605 Semarang
Tlpn. (024) 6723456
Fax . 024-6710144
Email: info@stekom.ac.id

KATA PENGANTAR

Terima kasih kepada Tuhan Yang Maha Esa atas kasih dan karunia-Nya, karena hanya bimibingan dan rahmat-Nya buku ini bisa selesai dengan lancar tanpa terhambat oleh sesuatu apapun.

Terimakasih kembali saya ucapkan kepada Mahasiswa yang telah membantu dalam hal pengumpulan gambar, teman teman dosen yang telah memberikan masukan berupa kritik dan saran serta semua pihak yang tidak bisa saya sebutkan satu persatu yang telah membantu sehingga tersusunnya buku ini.

Web service sendiri merupakan sebuah sistem perangkat lunak yang dirancang untuk mendukung interoperasi dalam interaksi mesin ke mesin melalui sebuah jaringan. Interaksi dilakukan melalui sebuah mekanisme atau protokol tertentu. Web service juga memiliki sebuah antarmuka yang mendeskripsikan seluruh layanan yang tersedia dalam format yang dapat diproses oleh mesin, yaitu Web serviceDescription Language (WSDL). Sistem-sistem lain yang nantinya akan berinteraksi dengan web servicedengan mengacu pada antarmuka tersebut akan melalui suatu protokol, yaitu Simple Object Access Protocol (SOAP).

Semoga dengan kehadiran buku ini dapat memberikan manfaat yang sebesar besarnya khususnya bagi Mahasiswa yang mengambil matakuliah Teknologi Web Service SOA dengan XML dan para pembaca pada umumnya.

Penulis

Toni Wijanarko Adi Putra, S.Kom., M.Kom

DAFTAR ISI

BAB 1
PENGENALAN WEB SERVICE

A. PENDAHULUAN

1. Deskripsi Singkat *:*

Pada mata kuliah ini khususnya pada BAB 1 akan di bahas Mengapa Perlu Menggunakan Layanan Web, yang mencakup diantaranya: Penggunaan Layanan Web, Blok Bangun Layanan Web, Keputusan Desain Layanan Web, Apa yang Tidak ada dalam Layanan Web.

2. Kemampuan Akhir yang diharapkan :

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan Penggunaan Layanan Web, Blok Bangun Layanan Web, Keputusan Desain Layanan Web, Apa yang Tidak ada dalam Layanan Web.

B. PENYAJIAN

Mengapa Perlu Menggunakan Layanan Web?

Sekarang, pemrograman berbasis komponen menjadi populer. Saat ini, hampir semua aplikasi dibangun dengan menggunakan komponen pengaturan dalam bentuk tertentu. Semakin berkembangnya, semakin bertambah pula kebutuhan untuk mengatur komponen yang dapat terdistribusi lewat mesin-mesin jarak jauh.

Contoh aplikasi berbasis komponen adalah solusi *e-commerce* ujung keujung. Sebuah aplikasi e-*commerce* pada Web farm harus mengirimkan perintah ke aplikasi *back-end* ERP (*Enterprise Resource*

Planning). Dalam kebanyakan kasus, aplikasi ERP berada dalam perangkat keras yang terpisah dan mungkin memiliki sistem operasi yang berbeda.

DCOM (*Distributed Component Object Model*) Microsoft, merupakan infrastruktur objek terdistribusi yang memungkinkan aplikasi mengambil komponen COM (*Component Object Model*) yang diinstal di *server* lain. Infrastruktur ini telah disertakan pada banyak platform non-Windows, tapi DCOM tidak pernah diterima secara luas pada platform-platform tersebut, sehingga jarang digunakan untuk menyediakan fasilitas komunikasi antara komputer Windows dan non-Windows. Perusahaan yang menjual perangkat lunak ERP sering membuat komponen untuk platform Windows yang berkomunikasi dengan sistem *back-end* melalui protokol yang sesuai.

Beberapa layanan yang diatur lewat aplikasi e-*commerce* mungkin sama sekali tidak ada di pusat data. Misalnya, jika aplikasi e-*commerce* menerima pembayaran kartu kredit untuk barang yang dibeli pelanggan, maka aplikasi itu harus membuka layanan bank penanggung untuk memproses informasi kartu kredit pelanggan. Namun supaya praktis, DCOM serta teknologi-teknologi lainnya seperti COREA dan Java RMI dibatasi untuk aplikasi dan komponen yang diinstal di pusat data perusahaan. Dua alasan utamanya adalah bahwa secara default, teknologi-teknologi ini mengatur ketepatan protokol dan protokol-protokol ini memang ditujukan untuk koneksi.

Client yang melakukan komunikasi dengan *server* lewat internet menghadapi banyak hambatan potensial dalam melakukan komunikasi

dengan *server*. Semua administrator di seluruh dunia yang peduli pada masalah keamanan telah menerapkan *router* dan *firewall* perusahaan untuk menyaring setiap komunikasi lewat Internet. Seringkali, dapat meminta administrator jaringan membuka port melebihi batas minimum adalah suatu keajaiban.

Jika beruntung, Kita dapat meminta administrator jaringan membuka port yang sesuai untuk mendukung layanan Kita. Tantangannya adalah *client* Kita tidak akan sama-sama beruntung. Akibatnya, protokol-protokol seperti yang digunakan oleh DCOM, COREA, dan Java RMI kurang praktis untuk aplikasi internet.

Masalah lainnya; dengan teknologi-teknologi tersebut, orientasi koneksi internalnya tidak dapat menangani interupsi jaringan dengan baik. Karena internet berada langsung di bawah kontrol Kita, maka Kita tidak dapat membuat asumsi kualitas ataupun kehkitalan koneksinya. Jika terjadi interupsi koneksi, panggilan berikutnya yang dilakukan *client* ke *server* tidak akan berhasil.

Ciri khas orientasi koneksi teknologi-teknologi ini juga menjadi tantangan untuk infrastruktur keseimbangan beban yang dibutuhkan supaya dapat tercapai kemampuan mengubah skala aplikasi yang tinggi. Setelah koneksi antara *client* dan *server* terancam, Kita biasanya tidak dapat dengan mudah meneruskan permintaan selanjutnya ke *server* lain.

Para pengembang telah berusaha mengatasi batasan ini dengan menggunakan satu model yang disebut pemrograman tanpa status, tapi mereka kurang berhasil karena teknologi untuk membangun koneksi

dengan objek jarak jauh memang sulit dan mahal.

Karena proses kartu kredit nasabah dilakukan oleh server jarak jauh lewat Internet, DCOM tidak cocok sebagai fasilitas komunikasi antara *client e-commerce* dan *server* pemrosesan kartu kredit. Seperti pada solusi ERP, komponen pihak ketiga sering diinstal ke dalam pusat data *client*. Dalam hal ini adalah provider solusi proses kartu kredit. Komponen ini berfungsi lebih banyak dibandingkan proxy yang mempermudah komunikasi antara perangkat lunak *e-commerce* dan bank penanggung lewat protokol yang sesuai.

Akibat keterbatasan teknologi dalam menyediakan fasilitas komunikasi antara sistem komputer, pembuat perangkat lunak seringkali memutuskan untuk membangun infrastruktur sendiri. Artinya, sumber daya yang seharusnya dapat digunakan untuk meningkatkan fungsionalitas sistem ERP atau sistem pemrosesan kartu kredit malah terpakai untuk menulis protokol jaringan yang sesuai.

Dalam usaha memperbaiki dukungan skenario semacam internet, Microsoft mula-mula menggunakan strategi, yaitu dengan memperbaiki teknologi yang sudah ada, termasuk CIS (COM *Internet Services*), yang memungkinkan Kita membangun koneksi DCOM antara client dan komponen jarak jauh lewat port 80. Karena berbagai alasan, akhirnya CIS ditolak.

Jelaslah sudah bahwa dibutuhkan satu pendekatan baru. Oleh karena itu, Microsoft memutuskan untuk memperbaiki masalah ini dari awal sampai akhir. Mari kita lihat beberapa persyaratan yang harus dilakukan agar solusi ini dapat berhasil:

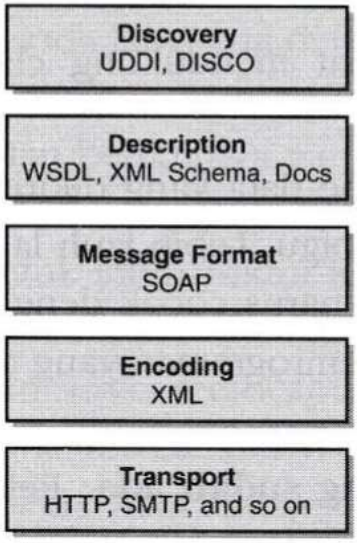
1. **Kerja sama yang baik.** Layanan jarak jauh harus dapat digunakan oleh *client* dengan *platform* berbeda.
2. **Internet yang bersahabat.** Solusi harus dapat mendukung *client* yang mengakses layanan jarak jauh lewat internet.
3. **Model hubungan antar muka yang kuat.** Tipe data yang dikirim ke dan diterima dari layanan jarak jauh tidak boleh ambigu. Lebih jauh lagi, model data yang didefinisikan oleh layanan jarak jauh harus cocok dengan model data yang didefinisikan oleh mayoritas bahasa pemrograman yang telah ditetapkan.
4. **Kemampuan memakai stkitar internet yang sudah ada.** Pelaksanaan layanan jarak jauh sedapat mungkin memakai stkitar internet yang ada dan jangan mengubah solusi untuk masalah yang sudah mempunyai solusinya. Solusi yang sudah tersedia di dalam stkitar internet yang banyak dipakai dapat memakai toolset yang ada dan dapat menghasilkan produk dari teknologi tersebut.
5. **Mendukung banyak bahasa.** Solusi jangan terikat oleh bahasa pemrograman tertentu. Misalnya, Java RMI yang sangat terikat dengan bahasa Java.Visual Basic atau Perl akan mengalami kesulitan untuk memanggil objek yang difungsikan dengan Java dari jarak jauh. Client harus dapat menerapkan layanan Web baru atau menggunakan layanan Web yang ada, dengan setiap bahasa pemrograman yang dipakai oleh *client*.
6. **Mendukung semua infrastruktur komponen terdistribusi.** Solusi jangan terikat dengan infrastruktur komponen tertentu. Sebenarnya, Kita tidak perlu membeli, menginstal, atau pun

mengelola infrastruktur objek terdistribusi hanya untuk membuat atau menggunakan layanan jarak jauh Protokol yang mendasari harus menyediakan komunikasi tingkat dasar di antara infrastruktur objek terdistribusi yang sudah ada, seperti DCOM dan CORBA.

Sebagaimana yang disebut dalam judul bab ini, tentu tidak mengherankan jika solusi yang dibuat dikenal dengan sebutan layanan Web. Layanan Web mempertemukan muka antar muka untuk memanggil aktivitas tertentu berdasarkan *client. Client* dapat mengakses layanan Web melalui penggunaan internet stkitar.

Blok Bangunan Layanan Web

Gambar 1.1 berikut ini menunjukkan inti blok bangunan untuk menyediakan fasilitas komunikasi jarak jauh antara dua aplikasi.



Gambar 1.1. Blok bangunan untuk menyediakan fasilitas komunikasi

jarak jauh antara dua aplikasi

Mari kita bahas maksud dari tiap-tiap blok bangunan ini.

1. **Penemuan**. Aplikasi *client* dengan kebutuhan akses fungsionalitas yang dimunculkan oleh layanan Web membutuhkan cara untuk menemukan lokasi layanan jarak jauh. Hal ini dilakukan lewat proses yang umumnya dikenal sebagai penemuan. Penemuan dapat menyediakan fasilitas lewat direktori terpusat maupun oleh metode tertentu. Di dalam DCOM, SCM (*Service Control Manager*) menyediakan layanan penemuan.
2. **Penjelasan**. Setelah lokasi layanan Web ditemukan, client membutuhkan informasi yang memadai untuk melakukan interaksi dengan layanan tersebut. Penjelasan tentang layanan Web mencakup metadata terstruktur antar muka yang akan digunakan oleh aplikasi client serta dokumen layanan Web yang tertulis, termasuk contoh penggunaannya. Komponen DCOM mengolah metadata terstruktur antar muka melalui typelib. Metadata di dalam sebuah typelib komponen disimpan dalam format biner yang sesuai, kemudian diakses lewat API (*Application Programming Interface*) yang sesuai.
3. **Format pesan.** Untuk pertukaran data, *client* dan *server* harus sama-sama menggunakan kode dan format pesan yang sama. Cara stkitar pengodean akan memastikan bahwa data yang oleh *client* dibuat kodenya akan diterjemahkan dengan benar oleh *server*. Di dalam DCOM, pesan yang dikirim antara *client* dan *server* diformat

sebagaimana didefinisikan oleh protokol ORPC (*Object* RPC) pada DCOM.

Tanpa pesan stkitar, membuat toolset pengembangan yang membebaskan pengembang protokol yang mendasarinya akan menjadi mustahil. Membuat lapisan pemisah antara pengembang dan protokol yang mendasari memungkinkan mereka terfokus pada masalah bisnis yang ada dan mengurangi pekerjaan dalam infrastruktur yang dibutuhkan untuk menerapkan solusinya.

4. **Pengkodean.** Data yang dialirkan antara client dan server harus dikodekan di dalam tubuh pesan. DCOM memakai skema kode biner untuk melakukan serialisasi data di dalamnya lewat pertukaran parameter antara *client* dan *server*.
5. **Transpor.** Setelah pesan diformat dan datanya diserialisasi dalam tubuh pesan, pesan itu harus ditransfer antara client dan server melalui protokol transpor tertentu. DCOM mendukung sejumlah protokol yang terikat ke protokol jaringan seperti, TCP, SPX, NetBEUI, dan NetBIOS melalui IPX.

Keputusan Desain Layanan Web

Mari kita membahas tentang keputusan desain di balik blok bangunan layanan Web.

1. Memilih Protokol Transpor

Langkah pertama adalah menentukan bagaimana *client* dan *server* akan saling berkomunikasi. *Client* dan *server* dapat berada pada satu LAN, tapi sangat mungkin client berkomunikasi dengan *server* lewat internet.

Oleh karena itu, protokol transpor harus cocok dengan lingkungan LAN dan internet.

Seperti yang telah diutarakan, teknologi seperti DCOM, CORBA, dan Java RMI kurang cocok untuk mendukung komunikasi antara client dan server lewat internet. Protokol seperti HTTP (*Hypertext Transfer Protocol*) dan SMTP (*Simple Mail Transfer Protocol*) sudah terbukti sebagai protokol internet. HTTP mendefinisikan pola pesan permintaan /jawaban untuk mengirimkan permintaan sekaligus mendapatkan tanggapan. SMTP mendefinisikan protokol pesan yang dapat disalurkan untuk komunikasi tidak sejajar. Mari kita lihat mengapa HTTP dan SMTP cocok untuk internet.

HTTP yang berbasis aplikasi Web secara inheren tidak memiliki status. HTTP tidak bergantung pada koneksi yang terus menerus antara *client* dan *server*. Hal ini membuat HTTP menjadi protokol yang ideal untuk tersedianya banyak konfigurasi seperti *firewall*. Jika server yang menangani permintaan asli dari client tidak ada, permintaan selanjutnya akan disalurkan ke *server* lain tanpa sepengetahuan *client*. Hampir semua perusahaan sudah memiliki infrastruktur yang mendukung SMTP karena sangat cocok untuk komunikasi tak sejajar. Jika layanan mengalami gangguan, infrastruktur *e-mail* secara otomatis akan berusaha mengulanginya.

Tidak seperti HTTP, pesan SMTP dapat diteruskan ke *server e-mail* lokal yang akan berusaha mengirimkan pesan Kita. Keuntungan lain dari HTTP dan SMTP adalah penggunaannya yang luas. Para karyawan sudah bergantung pada *e-mail* dan browser Web, dan administrator

jaringan sudah terbiasa menggunakan kedua layanan ini. Teknologi semacam *server proxy* dan NAT (*Network Address Translation* / penerjemah alamat jaringan) menyediakan satu cara untuk mengakses Internet melalui HTTP dari LAN perusahaan lainnya yang terpisah. Administrator akan sering memunculkan *server* SMTP yang ada di dalam *firewall*. Pesan yang masuk ke *server* ini kemudian akan disalurkan ke tujuannya melalui internet.

Perangkat lunak yang memproses kartu kredit, membutuhkan tanggapan yang cepat dari bank penanggung untuk menentukan tujuannya, apakah pesanan itu harus dikirim ke sistem- ERP atau tidak. HTTP bersama pola pesan permintaan/jawaban, sangat cocok untuk tugas ini.

Kebanyakan paket perangkat lunak ERP tidak mampu menangani perintah dalam jumlah besar yang dapat dikendalikan dari aplikasi *e-commerce*. Selain itu, permintaan tidak harus dikirimkan ke sistem ERP secara bersamaan. Dengan demikian, SMTP dapat menggunakan antrian permintaan, sehingga pemrosesan dapat dilakukan secara bertahap oleh sistem ERP.

Jika sistem ERP mendukung transaksi terdistribusi, pilihan lainnya adalah menggunakan MSMQ (*Microsoft Message Queue Server*). Selama aplikasi *e-commerce* dan sistem ERP ada di dalam satu LAN, koneksi lewat protokol non-internet tidak terlalu bermasalah. Keuntungan MSMQ dibanding SMTP adalah pesan dapat disimpan dan dihapus dari antrian dalam ruang lingkup transaksi. Jika usaha untuk memproses pesan yang diambil dari antrian mengalami kegagalan,

pesan tersebut secara otomatis akan dimasukkan kembali ke dalam antrian.

2. Memilih Skema Pembuatan Kode

HTTP dan SMTP menyediakan cara penggunaan data antara *client* dan *server*. Namun, keduanya tidak menentukan bagaimana data di dalam tubuh pesan harus dikodekan. Microsoft membutuhkan satu stkitar, yaitu cara yang terlepas dari platform untuk mengkodekan data yang dipertukarkan antara *client* dan *server*.

Karena tujuannya adalah memakai protokol berbasis internet, XML (*Extensible Markup Language*) adalah pilihan yang biasa diambil. XML menawarkan banyak keuntungan, termasuk dukungan lintas-platform, sistem tipe yang umum, serta dukungan untuk setting karakter stkitar industri.

Skema pengodean biner seperti yang digunakan oleh DCOM, COREA, dan Java RMI harus memiliki kinerja yang sama antar platform hardware yang berlainan. Misalnya, platform hardware yang berbeda memiliki perbedaan representasi biner internal pada angka multibyte. Platform Intel mengurutkan byte-byte pada angka multibyte dengan menggunakan konvensi kecil endian. Banyak prosesor RISC mengurutkan byte-byte pada angka multibyte dengan menggunakan konvensi besar endian.

XML menghindari masalah pengodean biner karena menggunakan skema pengodean berbasis teks yang memanfaatkan rangkaian karakter stkitar. Selain itu, beberapa protokol transpor, seperti SMTP, dapat berisi pesan berbasis teks saja. Metode pembuatan kode biner, seperti

yang dipakai oleh DCOM dan COREA, tidak luwes dan membutuhkan infrastruktur yang mendukung untuk membantu pengembang dalam hal-hal yang mendetail. XML jauh lebih ringan dan lebih mudah digunakan karena dapat dibuat dan digunakan memakai teknik pemilahan teks stkitar.

Selain itu, ketersediaan berbagai pemilah XML lebih memudahkan pembuatan dan pemakaian dokumen XML di hampir setiap platform modern. XML tidak berat dan memiliki dukungan tool yang bagus. Oleh karena itu, pembuatan kode XML memungkinkan jangkauan yang sangat luas karena hampir semua client pada platform apa pun dapat berkomunikasi memakai layanan Web.

3. Memilih Konvensi Format

Seringkali diperlukan tindakan untuk memasukkan metadata tambahan dengan tubuh pesan. Misalnya, Kita mungkin ingin memasukkan informasi mengenai tipe layanan yang dibutuhkan layanan Web untuk memenuhi permintaan Kita, seperti memasukkan ke dalam daftar transaksi atau informasi routing. XML tidak menyediakan mekanisme untuk membedakan tubuh pesan dengan data yang terkait oleh pesan tersebut.

Protokol transpor seperti HTTP menyediakan mekanisme yang dapat diperluas untuk data header, tapi beberapa data yang terkait dengan pesan mungkin tidak hanya untuk protokol transpor saja. Misalnya, client mungkin mengirimkan pesan yang harus disalurkan ke berbagai tujuan dan kemungkinan besar melalui protokol transpor yang berbeda. Jika informasi *routing* diletakkan di dalam *header* HTTP, maka harus

diterjemahkan dahulu sebelum dikirim ke perantara berikutnya pada protokol transpor yang lain, seperti SMTP. Karena informasi routing spesifik untuk pesan itu dan tidak untuk protokol transpor, maka harus menjadi bagian dari pesan.

SOAP (*Simple Object Access Protocol*) menyediakan cara nyata protokol untuk menghubungkan informasi *header* dengari tubuh pesan. Setiap pesan SOAP harus mendefinisikan sebuah "amplop". Amplop ini memiliki badan yang mengandung isi dari pesan serta *header* yang dapat berisi metadata yang terkait dengan pesan.

SOAP tidak membatasi cara isi pesan tersebut diformat. Hal ini dapat menjadi masalah karena tanpa cara yang konsisten dalam pengodean data, toolset yang membatasi Kita dari protokol yang mendasari akan sulit dibuat. Mungkin, waktu Kita akan dihabiskan hanya untuk membuat antar muka layanan Web daripada memperbaiki masalah bisnis yang ada.

Cara stkitar dalam memformat pesan RPC (*Remote Procedure Call* / panggilan prosedur jarak jauh) dan pengkodean daftar parameternya lebih dibutuhkan. Cara inilah yang akan disediakan dalam spesifikasi SOAP. Bagian itu akan menjelaskan mengenai konvensi penamaan stkitar serta corak pembuatan kode untuk pesan berorientasi prosedur.

Karena SOAP menyediakan format stkitar untuk serialisasi data ke dalam pesan XML, platform semacam ASP.NET dan Remoting dapat menangani hal-hal yang kecil untuk Kita.

4. Memilih Mekanisme Penjelasan

SOAP menyediakan cara stkitar memformat pesan yang dipertukarkan

di antara layanan Web dan *client*. Namun, *client* membutuhkan informasi tambahan agar dapat dengan tepat membuat serialisasi permintaan dan menerjemahkan tanggapannya. Skema XML menyediakan cara pembuatan skema yang dapat dipakai untuk menjelaskan isi pesan.

Skema XML menyediakan serangkaian data type inti yang dapat digunakan untuk menjelaskan isi pesan. Kita juga dapat membuat data type sendiri. Misalnya, bank penanggung dapat membuat suatu data type yang kompleks untuk menjelaskan isi dan struktur tubuh sebuah pesan supaya mengirimkan permintaan pembayaran kartu kredit.

Sebuah skema berisi serangkaian data type dan pendefinisian elemen. Sebuah layanan Web menggunakan skema tidak saja untuk komunikasi tipe data yang ingin dimasukkan dalam sebuah pesan tapi juga untuk membuat validasi pesan masuk dan keluar.

Namun, skema itu sendiri tidak menyediakan informasi yang memadai untuk menjelaskan layanan Web secara efektif. Skema tidak menjelaskan pola pesan antara *client* dan *server*. Misalnya, *client* harus tahu apakah perlu meminta tanggapan ketika perintah dikirimkan ke sistem ERP. *Client* juga perlu tahu protokol transpor apa yang diminta sekaligus diperlukan oleh layanan Web untuk menerima permintaan. Akhirnya, *client* harus tahu alamat layanan Web tersebut bisa dicapai.

Informasi ini disediakan oleh dokumen WSDL (*Web Services Description Language*) WSDL adalah dokumen XML yang secara lengkap menerangkan layanan Web tertentu. Tool-tool seperti WSDL.exe pada ASP.NET dan SOAP-SUDS.exe pada Remoting dapat

menggunakan WSDL dan secara otomatis membangun sejumlah *proxy* untuk para pengembang.

Seperti pada semua komponen yang digunakan untuk membangun perangkat lunak, layanan Web juga harus disertai catatan dokumen untuk pengembang yang membuatnya. Dokumentasi harus menjelaskan apa yang dilakukan layanan Web, antar muka yang ditampilkan, serta beberapa contoh cara menggunakannya. Dokumentasi yang baik sangatlah penting jika layanan Web itu dibuka untuk client lewat internet.

5. Memilih Mekanisme Penemuan

Setelah Kita membuat layanan Web dan dokumentasinya, bagaimana client menemukannya? Jika layanan Web dirancang untuk digunakan oleh seorang anggota tim pengembangan, caranya mungkin dilakukan secara informal, seperti membagi URL pada dokumen WSDL dengan rekan-rekan dalam tim. Namun, jika *client* mengaksesnya lewat internet, mengiklankan layanan Web secara efektif merupakan cerita lain.

Apa yang dibutuhkan adalah kesamaan dalam cara mengiklankan layanan Web. UDDI (*Universal Description, Discovery, and Integration*) menyediakan mekanisme seperti ini. UDDI adalah layanan direktori terpusat dengan stkitar industri yang dapat digunakan untuk mengiklankan dan menemukan layanan Web. Dengan UDDI, pengguna dapat menemukan layanan Web menggunakan sejumlah kriteria pencarian, seperti nama perusahaan, kategori, serta tipe layanan Web.

Layanan Web juga dapat diiklankan melalui DISCO, format dokumen XML yang ditentukan oleh Microsoft agar layanan Web dapat mengiklankan layanan yang ditawarkan. DISCO mendefinisikan protokol sederhana untuk mempermudah *style hyperlink* dalam menemukan sumber. Microsoft Visual Studio.NET adalah pemakai DISCO yang paling sering. Seorang pengembang dapat menggunakan *server* Web tertentu dan menelusuri berbagai layanan Web yang dibuka oleh server itu.

Apa yang Tidak Ada dalam Layanan Web?

Kita mungkin sudah tahu bahwa beberapa item yang ada di dalam distribusi infrastruktur komponen tidak didefinisikan oleh layanan Web. Dua lagi yang dihilangkan adalah sebuah API yang terdefinisi jelas untuk membuat dan menggunakan layanan Web, kemudian mengatur layanan Web komponen, seperti dukungan untuk transaksi terdistribusi. Marilah kita membahas setiap bagian yang hilang ini.

1. **API-spesifik untuk layanan Web**. Kebanyakan infrastruktur komponen terdistribusi mendefinisikan API untuk melakukan tugas-tugas tertentu seperti memulai runtime, membuat contoh komponen, dan menampilkan metadata yang digunakan untuk menjelaskan komponen tersebut. Karena bahasa pemrograman tingkat tinggi menyediakan beberapa kerja sama dengan C#, API biasanya diekspos sebagai rangkaian daftar tkita tangan metode C. RMI terlalu kuat menyatukan API-nya dengan Java sebagai satu bahasa tingkat tinggi.

   Untuk memastikan bahwa layanan Web merupakan bahasa

pemrograman skeptis, Microsoft telah membiarkan penjual *software* mengikatkan dukungan layanan Web bagi platform tertentu.

2. **Layanan komponen.** Platform layanan Web tidak menyediakan layanan yang biasa digunakan dalam infrastruktur komponen terdistribusi, seperti pengelolaan masa hidup objek jarak jauh, pooling objek, dan dukungan untuk transaksi terdistribusi. Pelaksanaan layanan-layanan ini diserahkan kepada infrastruktur komponen terdistribusi.

   Beberapa layanan, seperti dukungan untuk transaksi terdistribusi, nanti bisa diperkenalkan setelah teknologi ini matang. Layanan lainnya, seperti pooling objek dan pengelolaan masa hidup objek dapat dianggap sebagai pelaksanaan detail platform. Misalnya, Remoting mendefinisikan ekstensi penyediaan dukungan pengelolaan masa hidup objek dan menyediakan dukungan Microsoft Componen Services untuk pooling objek.

C. PENUTUP

Ringkasan

Pemrograman berbasis komponen terbukti telah menjadi anugerah bagi produktivitas pengembang. Namun, sebagian layanan tidak dapat dibungkus oleh komponen yang ada di dalam pusat data *client*. Teknologi *legacy* seperti DCOM, CORBA, dan Java RMI tidak cocok digunakan bagi *client* yang ingin mengakses layanan lewat internet, maka Microsoft merasa perlu memulai dari dasar dan membangun

sesuai stkitar industri dalam mengakses layanan jarak jauh.

Layanan Web merupakan istilah serupa payung yang menerangkan sekumpulan layanan dan protokol stkitar industri yang digunakan untuk mempermudah kerja sama tingkat dasar antar aplikasi. Dukungan industri yang diterima oleh layanan Web tiada bandingnya. Sebelumnya tidak ada perusahaan berteknologi maju memutuskan untuk mendukung stkitar yang mempermudah kerja sama antar aplikasi dengan platform yang berbeda.

Salah satu faktor penyebab suksesnya layanan Web adalah karena dibangun di atas stkitar Internet yang sudah ada seperti XML dan HTTP. Oleh karena itu, setiap sistem yang mampu memilah teks dan berkomunikasi lewat protokol transpor internet stkitar dapat berkomunikasi dengan layanan Web. Perusahaan juga dapat mengalihkan investasi mereka ke dalam teknologi ini.

Pertenyaan

1. Mengapa Perlu Menggunakan Layanan Web?
2. Apa itu Layanan Web Services?
3. Persyaratan apa saja yang harus dilakukan agar solusi perbaikan dapat berhasil?
4. Apa yang dimaksud dengan ERP?
5. Apa yang dimaksud dengan API?

E. DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,

2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.

14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 2

SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 2 akan di bahas SOAP (*Simple Object Access Protocol*) diantaranya: Inti dari layanan SOAP, Spesifikasi SOAP, Anatomi Pesan SOAP, SOAP Actors, Elemen Geader, Atribut mustUnderstand, Atribut Actor, Elemen Body, Elemen Fault, SOAP Encoding, Tipe tipe Sederhana, Tipe Campuran, Struktur, Arry, Optimisasi, Actor Root, Pengikatan Protokol.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan Inti dari layanan SOAP, Spesifikasi SOAP, Anatomi Pesan SOAP, SOAP Actors, Elemen Geader, Atribut mustUnderstand, Atribut Actor, Elemen Body, Elemen Fault, SOAP Encoding, Tipe tipe Sederhana, Tipe Campuran, Struktur, Arry, Optimisasi, Actor Root, Pengikatan Protokol.

B. PENYAJIAN

SOAP

Inti dari layanan Web adalah *Simple Object Access Protocol* (SOAP) yang menyediakan sebuah cara stkitar untuk memaketkan pesan. SOAP menarik banyak perhatian karena memfasilitasi komunikasi

bergaya RPC antara client dan server remote. Namun, sudah banyak protokol yang dibuat untuk memfasilitasi komunikasi antara dua aplikasi, termasuk Sun's RPC, Microsoft's DCE, JAVA's RMI, dan CORBA's ORPC. Jadi, mengapa SOAP menarik banyak perhatian?

Salah satu alasan utama adalah karena SOAP mempunyai dukungan industri yang hebat. SOAP adalah protokol pertama dari jenisnya yang akan diterima secara praktis oleh setiap perusahaan perangkat lunak besar dunia. Perusahaan-perusahaan yang jarang bekerja sama satu sama lain akan turut mendukung protokol ini. Microsoft, IBM, Sun Microsystems, SAP, dan Ariba termasuk dari sebagian perusahaan besar yang mendukung SOAP.

1. **SOAP berpasangan erat dengan satu bahasa.** Para pengembang yang terlibat dengan proyek baru dapat memilih untuk mengembangkan bahasa pemrograman terbaru dan terhebat. Namun, para pengembang yang bertanggung jawab memelihara aplikasi-aplikasi lama mungkin tidak punya pilihan terhadap bahasa pemrograman yang mereka gunakan. SOAP tidak menyebutkan sebuah API, jadi implementasi API dipasrahkan pada bahasa pemrogramannya (seperti misalnya Java) dan platformnya (seperti misalnya Microsoft. NET).
2. **SOAP tidak berpasangan erat dengan protokol transpor tertentu.** Spesifikasi SOAP menerangkan bagaimana pesan-pesan SOAP seharusnya diikat ke HTTP. Namun, sebuah pesan SOAP tidak lebih daripada sebuah dokumen XML. Jadi, ia dapat diangkut melalui protokol apa pun yang mampu mengirimkan teks.

3. **SOAP tidak terpaku pada satu infrastruktur objek terdistribusi manapun.** Sebagian besar sistem objek terdistribusi dapat diperluas (sebagian dari mereka) untuk mendukung SOAP. Perlu disadari bahwa pada SOAP sekalipun, *middleware* semacam COM+ tetap memainkan peranan penting dalam perusahaan besar. *Middleware* komponen tetap bertanggung jawab untuk sebagian fitur manajemen objek yang lebih kompleks, seperti misalnya usia objek dan penampungan sumberdaya. SOAP memungkinkan suatu derajad kerja sama di antara sistem-sistem berbeda yang menjalankan *middleware* komponen dari para vendor yang berkompetisi.
4. **SOAP mempengaruhi berbagai stkitar industri yang ada.** Para kontributor utama untuk spesifikasi SOAP pada mulanya menghindari penemuan ulang apa pun. Mereka lebih memilih untuk memperluas stkitar yang telah ada bagi pemenuhan kebutuhan mereka. Sebagai contoh, SOAP mempengaruhi XML untuk pengodean pesan. Daripada menggunakan sistem tipenya sendiri, SOAP mengatur definisi-definisi tipenya yang sudah didefinisikan ke dalam spesifikasi XML Schema. Seperti yang telah disebutkan, SOAP tidak mendefinisikan sarana untuk mengangkut pesan tersebut. Pesan-pesan SOAP dapat diikat ke protokol transpor yang ada seperti misalnya, HTTP dan SMTP.
5. **SOAP memungkinkan kerja sama ke beberapa lingkungan sekaligus.** SOAP dibangun di atas stkitar industri yang ada, sehingga aplikasi-aplikasi yang berjalan pada platform yang

mendukung stkitar-stkitar ini dapat secara efektif berkomunikasi dengan aplikasi-aplikasi yang berjalan pada platform lain lewat pesan SOAP. Sebagai contoh, aplikasi yang berjalan pada sebuah PC dapat secara efektif berkomunikasi dengan sebuah aplikasi backend yang berjalan pada sebuah mainframe yang sanggup mengirim dan menerima XML lewat HTTP.

Bab ini mencakup aspek-aspek kunci berikut pada spesifikasi SOAP:

1. **SOAP Envelope.** Aspek ini digunakan untuk mengodekan informasi header mengenai pesan dan tubuh pesannya sendiri.
2. **SOAP Encoding.** Aspek ini adalah cara stkitar untuk menyerialkan data ke dalam tubuh pesan SOAP.
3. **Pesan-pesan bergaya-RPC.** Disini membahas protokol yang dapat Kita gunakan untuk memfasilitasi komunikasi berorientasi prosedur lewat pola pesan permintaan / jawaban.
4. **Pengikatan protokol HTTP POST.** Aspek ini adalah metode stkitar pengikatan pesan-pesan SOAP ke HTTP.

Sebelum beranjak lebih jauh, mari kita bahas status SOAP. Bab ini ditulis untuk versi 1.1 dari spesifikasi SOAP (http://www.w3.org/TRISOAP) . *World Wide Web Consortium* (W3C) melanjutkan pengembangan SOAP dan pada tanggal 9 Juli 2001, naskah kerja SOAP 1.2 telah diterbitkan (http://www.wJ.org/TR/ 2001/ WD-soap12-20010709) oleh XML *Protocol Working Group*.

Sebagai sebuah pernyataan bagi dukungan industri yang fenomenal, XML *Protocol Working Group* bertekad untuk menjaga path migrasi yang mulus dari SOAP 1.1 ke SOAP 1.2. Banyak modifikasi yang

diajukan bersifat *fit and finish* dan tidak secara radikal mengubah pemakaian SOAP. Banyak dari yang telah Kita pelajari pada SOAP 1.1 akan langsung diterjemahkan ke SOAP 1.2.

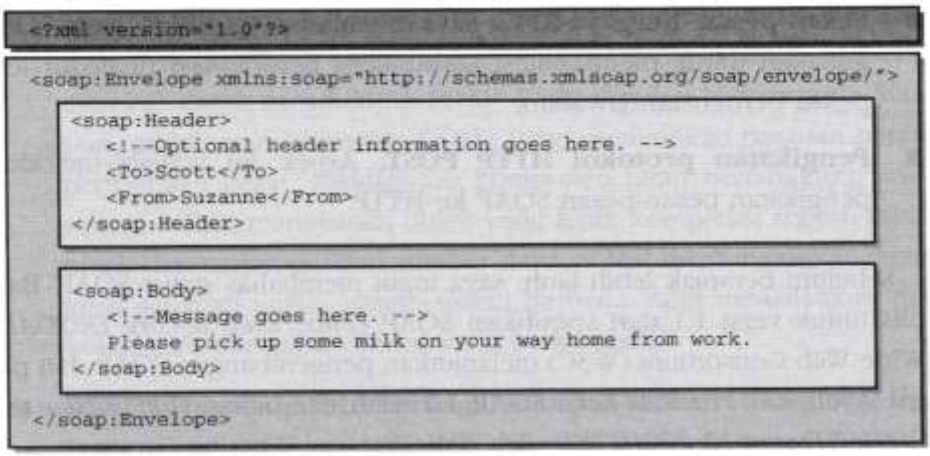
Di samping itu, mayoritas produk Microsoft yang menyertai SOAP sepertinya tidak akan mengadopsi SOAP 1.2 sampai SOAP 1.2 resmi direkomendasikan oleh W3C. Oleh karena itu, Penulis menyarankan supaya Kita memfokuskan diri mempelajari *protokol* SOAP 1.1 dengan tetap mengamati perkembangan versi 1.2.

Anatomi Pesan SOAP

SOAP menyediakan cara stkitar pemaketan sebuah pesan. Pesan SOAP tersusun dari sebuah amplop yang berisi tubuh pesan dan informasi *header* yang dipakai untuk menerapkan pesan tersebut. Berikut ini contohnya:

Elemen root dari dokumen tersebut adalah *elemen Envelope*. Contoh tersebut berisi dua subelemen, yaitu *elemen Body* dan *elemen Header*. Dalam amplop tersebut, pesan SOAP yang sah dapat juga berisi elemen-elemen anak lainnya. Kita akan melihat contoh-contohnya saat membahas penyerialan referensi dengan menggunakan SOAP *Encoding*.

Amplop dapat berisi sebuah *elemen Header* opsional yang mengandung informasi pesan. Dalam contoh terdahulu, *header* tersebut berisi dua elemen yang menjelaskan individu pembuat pesan dan penerima pesan yang dituju seperti ditunjukkan pada Gambar 2.1.



Gambar 2.1 Elemen Header dan Elemen Body

Amplop harus berisi satu *elemen Body*. Tubuh tersebut berisi muatan pesan. Dalam contoh tersebut, tubuh tersebut berisi sebuah string karakter sederhana.

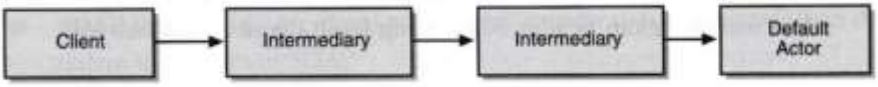
Perhatikan bahwa setiap elemen SOAP yang spesifik mempunyai awalan namespace soap. Awalan ini didefinisikan dalam elemen Envelope dan menunjuk ke skema SOAP yang menerangkan struktur sebuah pesan SOAP. Awalan tersebut ditambahkan ke elemen yang didefinisikan *Form_Load* namespace SOAP. Elemen-elemen ini sepenuhnya teruji. Awalan soap menunjukkan bahwa elemen Envelope adalah sebuah persyaratan dari tipe *Envelope* SOAP.

SOAP Actors

Sebelum Penulis menerangkan bagian-bagian individual dari pesan SOAP, Penulis ingin mendefinisikan sepasang istilah yang akan Penulis gunakan. SOAP *actor* adalah semua yang beraksi atas isi pesan

SOAP. Ada dua tipe SOAP *actor*, default *actors* (*aktor default*) dan *intermediaries* (perantara).

Aktor default adalah penerima akhir yang dituju oleh sebuah pesan SOAP. Perantara menerima sebuah pesan SOAP dan dapat melakukan sesuatu atas pesan tersebut (termasuk memodifikasinya dengan cara tertentu) sebelum meneruskannya bersama path pesan yang dituju, seperti diperlihatkan dalam diagram berikut. Sekalipun perantara dapat memodifikasi data yang ditransfer dari client ke aktor default, pesannya masih dianggap sama.



Gambar 2.2 Alur Client ke Aktor

Elemen Header

Elemen Header opsional digunakan untuk menyalurkan data yang mungkin tidak sesuai untuk dikodekan ke dalam tubuhnya. Sebagai contoh, jika aktor default menerima sebuah pesan yang tubuhnya telah dikompresi, nantinya aktor default perlu mengetahui tipe algoritma kompresi yang dipakai untuk membuka pesan itu. Tidak mungkin menanam informasi mengenai algoritma ke dalam tubuh pesan karena tubuhnya sendiri akan dikompresi. Jauh lebih cocok menempatkan tipe informasi ini di dalam header pesan.

Penggunaan header untuk yang lain, termasuk yang berikut ini:

1. **Otentikasi.** Penerima dapat meminta pengirim untuk mengotentikkan dirinya sendiri sebelum pesan diproses.
2. **Informasi pernyataan keamanan.** Jika penerima memerlukan

jaminan bahwa isi pesan belum berubah, pengirim dapat menkitatangani tubuh pesan secara digital dan menempatkan pemberitahuan hasilnya ke dalam header.

3. **Informasi routing.** Jika pesan tersebut perlu dirunut ke banyak tujuan, tujuan dan urutannya dapat dimasukkan ke dalam header.
4. **Transaksi.** Penerima mungkin menjalankan sejumlah aksi dalam lingkup transaksi pengirim.
5. **Informasi pembayaran.** Jika penerima pesan menyediakan layanan ke client berdasarkan biaya per pemakaian, informasi yang diperlukan untuk menarik pembayaran dapat ditanamkan dalam headernya.

Elemen Header dapat ditambahkan sebagai sebuah elemen anak dalam *Envelope* SOAP. Entri-entri header muncul sebagai simpul-simpul anak dalam elemen Header SOAP. Berikut ini adalah sebuah contohnya:

```
<?xml version="1.0" encoding=="utf-8"?>
<soap:Envelope xmlns: soap="http://schemas.xml soap.org/soap/envelope/">
<soap:Header>
        <Digest>B839D234A3F87</Digest>
</soap:Header>
<soap:Body>
        <StockReport>
                <Symbol>MSFT</Symbol>
                <Price>74.56</Price>
        </StockReport>
</soap:Body>
</soap:Envelope>
```

Dalam headernya, pesan SOAP berisi sebuah *elemen Digest* yang dapat digunakan aplikasi *remote* untuk memastikan bahwa pesan belum berubah. Jika client melakukan sebuah pemeriksaan rutin untuk melihat bagaimana sahamnya ditutup, ia mungkin tidak peduli dengan pengesahan pesan tersebut. Namun, jika harga saham memicu sebuah kejadi an dalam paket perangkat lunak keuangan, ia mungkin akan lebih tertarik dalam pengesahan pesan. Sebagai contoh, tentunya akan disesalkan sekali jika paket perangkat lunak keuangan secara otomatis melikuidasi portofolionya akibat menerima sebuah pesan salah yang dikirimkan seorang bocah 14 tahun.

Atribut mustUnderstand

Karena *header* sifatnya opsional, penerima pesan dapat memilih untuk mengabaikannya. Bagaimanapun juga, sejumlah informasi yang dapat ditanamkan ke dalam *header* tidak boleh diabaikan oleh penerima yang dituju. Jika *header* tidak dipahami atau ditangani dengan benar, aplikasi tersebut mungkin tidak dapat berfungsi benar. Oleh karena itu, Kita perlu cara untuk membed akan antara informasi header yang sifatnya informatif dan yang sifatnya kritikal.

Kita dapat menyebutkan perlunya penerima pesan memahami sebuah elemen dalam *header* dengan menyebutkan atribut *mustUnderstand* bersama sebuah nilai 1 dalam root elemen *header*. Sebagai contoh, pesan SOAP mungkin meminta sebuah aplikasi remote menjalankan sebuah aksi atas nama *client*. Contoh berikut memperbarui informasi rekening pengguna dalam lingkup transaksi:

```
<?xml version="l.O" encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
        <Transactionld soap:mustUnderstand="l">l23</Transactionld>
</soap:Header>
<soap:Body>
        <UpdateAccountlnfo>
                <email>sshort@microsoft.com</email>
                <firstName>Scott</firstName>
                <lastName>Short</lastName>
        </UpdateAccountlnfo>
</soap:Body>
</soap:Envelope>
```

Penerima pesan harus memperbarui informasi rekening pengguna dalam lingkup transaksi *client*. Jika transaksi dibatalkan, aplikasi remote harus membalik perubahan yang diminta ke informasi rekening pengguna. Oleh karena itu, ID transaksi dikodekan ke dalam *header* dan mengatur *atribut must Understand* ke 1. Aplikasi remote harus menghargai transaksi tersebut atau tidak memproses pesan itu sama sekali.

Atribut Actor

Pesan SOAP dapat saja telah dirutekan melalui banyak perantara sebelum mencapai tujuan akhirnya. Sebagai contoh, dokumen terdahulu mungkin telah dirutekan melewati sebuah perantara yang bertanggung jawab membuat sebuah konteks transaksi. Dalam hal ini, Kita mungkin perlu menyebut dengan jelas apakah header

TransactionId dikehendaki untuk diproses oleh perantara transaksi daripada oleh aktor default.

Spesifikasi SOAP menyediakan atribut actor bagi anotasi header-header SOAP yang ditujukan untuk perantara tertentu. Nilai atribut ini adalah *Uniform Resource Identifier* (URI) dari perantara ke tujuan porsi pesan tersebut. Jika sebuah *header* dikehendaki untuk diproses oleh perantara berikutnya agar menerima pesan SOAP, actor dapat diatur ke http://schemas.xmlsoap.org/soap/actor/next. Jika tidak, atribut actor dapat diatur ke sebuah URI yang mengenali sebuah perantara spesifik. Berikut ini contohnya:

```
<?xml version="1.0"encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/">
<soap:Header>
        <Transactionld soap:mustUnderstand="1"
        Actor="urn:TransactionCoordinator>123</Transactionid>
</soap:Header>
<soap:Body>
        <TransferFunds>
                <Source>804039836</Source>
                <Destination>804039836</Destination>
                <Amount>151.43</Amount>
        </TransferFunds>
</soap:Body>
</soap:Envelope>
```

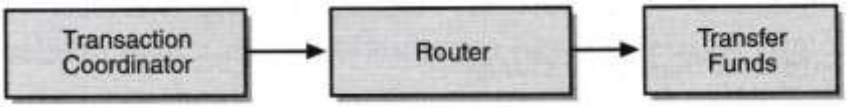
Karena elemen header *TransactionId* ditujukan untuk perantara koordinasi transaksi, atribut actor-nya diatur ke URL perantara. *Atribut mustUnderstand* juga telah diatur demikian karena akan terjadi kesalahan sekitainya perantara koordinator transaksi tidak memahami

elemen header *TransactionId*.

Jika pesan disalurkan ke penerima lain, segala elemen header yang dirancang untuk perantara tersebut harus dihapus sebelum pesan diteruskan. Bagaimanapun juga, perantara dapat memberikan elemen header tambahan sebelum meneruskan pesan ke penerima berikutnya. Dalam contoh ini, perantara koordinator transaksi harus membuang elemen router sebelum meneruskannya ke aplikasi penagihan.

Satu hal penting yang perlu diperhatikan adalah bahwa routing pesan secara langsung ke aktor default tidak dianggap sebuah kesalahan. Mengatur atribut mustUnderstand ke 1 dikombinasikan dengan mengatur atribut actor ke urn: *TransactionCoordinator* tidak memastikan bahwa pesan tersebut akan dirutekan melalui perantara. Ini berarti, jika pesan tidak mencapai perantara koordinator transaksi, maka ia harus mengerti entri header *TransactionId* atau melemparkan sebuah kesalahan.

Dalam contoh awal, perantar perlu menjalankan tugas penting sebelum pesan dirutekan ke aktor default. Ingatlah kembali, jika pesan mencapai perantara koordinator transaksi, perantara harus menghapus header Transactionld sebelum meneruskan pesannya. Oleh karena itu, aktor default dapat memeriksa adanya header *TransactionId*, yang akan menunjukkan apakah pesan belum melewati perantara yang sesuai. Bagaimanapun juga, tidak selalu ideal menentukan apakah semua header telah diproses setelah pesan mencapai aktor default. Bagaimana jika permintaan SOAP perlu dirutekan melalui perantara yang diperlihatkan di sini?



Gambar 2.3 Rute SOAP

Permintaan untuk mentransfer dana harus melalui perantara router sebelum dana ditransfer. Anggaplah router mengenakan biaya kepada pelanggan untuk memproses lanjutan permintaan ke layanan Web bank yang bersangkutan. Bagaimanapun juga, sebelum dana dikurangi, pesan tersebut harus dirutekan melalui koordinator transaksi untuk memulai sebuah transaksi sebelum data dimodifikasi. Oleh karena itu, perantara router dan aktor default harus menjalankan semua pekerjaan dalam lingkup transaksi. Karena layanan Web bank adalah aktor default, ia dapat memeriksa header tersebut untuk melihat apakah pesan telah dirutekan melalui perantara yang diperlukan.

Namun, bagaimana jika layanan Web bank menemukan bahwa pesan tidak pernah dirutekan melewati perantara manajer transaksi? Jika terjadi sebuah kesalahan selama transfer dana, Kita mungkin tidak dapat membatalkan pekerjaan yang telah dilaksanakan oleh perantara router. Lebih buruk lagi, pesan SOAP mungkin telah dirutekan melalui perantara router sebelum dirutekan melalui koordinator transaksi. Jika demikian halnya, mungkin tidak ada cara untuk memberitahukan bahwa aplikasi pengadaan telah menjalankan tugasnya di luar lingkup transaksi. Penulisngnya, SOAP tidak menyediakan suatu mekanisme untuk memastikan bahwa pesan melewati semua perantara yang

dimaksudkan dalam urutan yang benar.

Elemen Body

Pesan SOAP yang sah harus mempunyai satu elemen Body. Tubuh tersebut berisi muatan pesan. Tidak ada batasan tentang bagaimana tubuh tersebut dapat dikodekan. Pesan tersebut dapat berupa sebuah string sederhana dari karakter-karakter, array byte yang telah dikodekan, atau XML. Satu-satunya persyaratan adalah bahwa isinya tidak boleh mempunyai karakter yang akan membatalkan dokumen XML yang dihasilkan.

Spesifikasi SOAP menjelaskan sebuah metode tentang pengodean yang dapat digunakan untuk menyerialkan data ke dalam tubuh pesan. Sebaiknya Kita mematuhi skema pengodean yang telah mapan seperti misalnya yang ini karena memungkinkan pengirim untuk lebih mudah bekerja sama dengan penerima menggunakan seperangkat aturan serialisasi yang telah dikenal dengan baik.

Pesan-pesan SOAP dapat secara umum dimasukkan ke dalam dua kategori; pesan-pesan berorientasi prosedur dan pesan-pesan berorientasi dokumen. Pesan-pesan berorientasi prosedur menyediakan komunikasi dua arah dan biasanya disebut sebagai pesan-pesan RPC (*Remote Procedure Call*). Tubuh sebuah pesan RPC berisi informasi mengenai aksi yang diminta dari server dan segala parameter input dan ouput. Pesan-pesan berorientasi dokumen biasanya menyediakan komunikasi searah. Dokumen-dokumen bisnis seperti perintah pembelian merupakan contoh dari pesan-pesan berorientasi dokumen.

Mari kita amati lebih dekat masing-masing tipe dokumen ini.

Kedua pesan SOAP dipasangkan bersama untuk memfasilitasi sebuah panggilan metode RPC dengan SOAP; pesan permintaan dan pesan jawabannya. Informasi mengenai metode target bersama dengan segala parameter input disalurkan ke server lewat sebuah pesan permintaan. Server tersebut kemudian mengaktifkan sejumlah perilaku atas nama client dan mengembalikan hasil serta segala parameter kembalian. Sebagian besar contoh dalam bab ini berkaitan dengan pemanggilan metode RPC dan semuanya mengikuti panduan spesifikasi SOAP untuk mengodekan pesan-pesan RPC.

Sebuah dokumen bisnis seperti misalnya order pembelian atau sebuah faktur dapat dikodekan dalam tubuh pesan SOAP dan dirutekan ke penerima yang dituju. Penerima dokumen mungkin dapat atau tidak dapat mengirim kembali sebuah pesan pemberitahuan ke pengirimnya (Bagian "SOAP *Encoding*" dalam bab ini akan menerangkan penggunaan aturan serialisasi untuk mengodekan data yang dimuat dalam dokumen-dokumen bisnis ini). Karena dokumen-dokumen bisnis seringkali mencakup banyak perusahaan sekaligus, organisasi seperti BizTalk.org dan RosettaNet bertindak sebagai fasilitator dan penyimpanan bagi skema-skema yang mendefinisikan pertukaran dokumen umum.

Di dalam buku ini, juga akan diberikan penjelasan tentang bagaimana meminta platform.Net untuk membuat dan memakai pesan-pesan berorientasi dokumen dan RPC.

Elemen Fault

Rencana tidak selalu berjalan sesuai dengan harapan. Kadang-kadang, server menemui kesalahan saat sedang memproses pesan client. SOAP menyediakan sebuah cara stkitar untuk menyampaikan kembali pesan kesalahan kepada client.

Tanpa memedulikan gaya pengodean yang digunakan untuk membuat pesan, spesifikasi SOAP mengijinkan format untuk melaporkan kesalahan. Tubuh pesan harus berisi sebuah elemen Fault dengan struktur berikut:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
        <soap:Fault>
                <soap:faultcode>Client.Security</soap:faultcode>
                <soap:faultstring>Accessdenied</soap:faultstring>
                <soap:faultactor>http://abc.com</soap:faultactor>
                <soap:detail>
                        <MyError>
                        <Originator>File System</Originator>
                        <Resource>MySecureFile.txt</Resource>
                        </MyError>
                </soap:detail>
        </soap:Fault>
</soap:Body>
</soap:Envelope>
```

Kode kesalahan berisi sebuah nilai yang secara programatis dipakai untuk menentukan sifat kesalahannya. Spesifikasi SOAP mendefinisikan sekumpulan kode kesalahan yang dapat Kita gunakan

untuk menggambarkan kesalahan- kesalahan SOAP dasar. Kode-kode kesalahan dicantumkan dalam Tabel 2-1.

Tabel 2-1 Kode-Kode Kesalahan Dasar SOAP

Kode Kesalahan	**Keterangan**
VersionMismatch	Sebuah *namespace invalid* untuk elemen amplop SOAp yang telah disebutkan.
MustUnderstand	Sebuah elemen anak, langsung dalam header SOAP yang berisi sebuah atribut mustUnderstand yang diatur ke 1 dan tidak dimengerti atau diabaikan oleh server.
Client	Isi pesan ditemukan sebagai akar penyebab dari kesalahan. Tampaknya, root yang menyebabkan kesalahan yang dihasilkan dalam kode kesalahan Client berisi sebuah pesan *malformed* atau informasi tidak lengkap dalam pesan.
Server	Akar penyebab kesalahan tidak secara langsung bersifat mudah diatributkan pada isi pesan. Contoh-contoh kesalahan yang dihasilkan dalam kode kesa- lahan Server, termasuk server yang tidak dapat mengambil sumber daya yang sesuai (seperti misalnya koneksi database) untuk memproses pesan atau sebuah kesalahan logika saat memproses pesan.

Kita dapat menambahkan kode kesalahan yang lebih spesifik ke kode kesalahan SOAP yang tercantumkan dalam tabel tersebut dengan

menggunakan notasi "dot" dan menyusun kode kesalahan individual dari yang paling tidak spesifik sampai yang paling spesifik. Sebagai contoh, jika server tidak dapat membuka sebuah koneksi database yang diperlukan untuk memproses pesan client, kode kesalahan berikut akan dihasilkan:

```
<faultcode>Server.Database.Connection</faultcode>
```

Karena kesalahan tersebut bukan hasil langsung dari pesan client, kode kesalahan basis akan ditambahkan ke akhir kode kesalahan basis. Dalam contoh, Penulis mendefinisikan sebuah kategori kode bagi database dan sebuah kode kesalahan yang spesifik untuk kesalahan - kesalahan yang berhubungan dengan koneksi.

Elemen *faultstring* akan memuat sebuah string yang dapat kita mengerti dan menjelaskan kesalahan yang ditemui. Berikut ini nilai *faultstring* untuk kesalahan koneksi ke database:

```
<faultstring>Unable to open connection to the database</faultstring>
```

Kita dapat menggunakan elemen *faultactor* opsional untuk menkitai sumber pasti kesalahan tersebut. Satu-satunya pengecualian adalah jika sebuah perantara menghasilkan kesalahan. Jika kesalahan tersebut dihasilkan pada suatu saat selain penerima akhir pesan SOAP, elemen faultactor harus berisi sebuah URL yang menunjukkan sumber kesalahan. Jika tidak, URL dapat absen.

Menggunakan Pesan-Pesan RPC SOAP

Salah satu sasaran desain asli dari SOAP adalah menyediakan sebuah cara stkitar dan terbuka untuk memfasilitasi RPC menggunakan teknologi Internet seperti misalnya XML dan HTTP. Dalam bagian ini,

Penulis menerangkan metode pengodean pesan-pesan bergaya RPC yang dijelaskan dalam spesifikasi SOAP versi 1.1.

Seperti yang telah Penulis nyatakan sebelumnya, spesifikasi SOAP tidak mendiktekan cara mengodekan pesan-pesan, dan pengodean pesan-pesan bergaya RPC bukanlah pengecualian. Bagian ketujuh dari spesifikasi SOAP 1.1 menjelaskan cara yang dianjurkan untuk mengodekan pesan-pesan permintaan dan jawaban. Pengembang bebas membuat metode sendiri untuk mengodekan komunikasi RPC. Dalam bagian ini, Penulis akan membatasi diskusi pada metode stkitar mengodekan pesan-pesan SOAP bergaya-RPC.

Untuk memfasilitasi perilaku permintaan/jawaban yang dibutuhkan oleh RPC, Kita membutuhkan dua pesan SOAP; satu untuk permintaan dan satu untuk jawaban. Berikut ini cara mengodekan pesan permintaan bagi sebuah fungsi C# sederhana yang menambahkan dua angka:

```
public int Add(int x. int y)
{
        return x + y;
}
```

Metode Add menerima dua integer sebagai parameter input dan menyalurkan kembali hasilnya ke client sebagai sebuah parameter kembalian. Parameter-parameter input harus dipaketkan dalam tubuh pesan permintaan supaya dapat dikirimkan ke aplikasi target. Ini dilakukan dengan memaketkan parameter-parameter dalam format mirip struktur. Berikut ini adalah pesan permintaan yang di- hasilkan untuk Add(l,2):

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/">
<soap:Body>
        <Add>
                <x>1</x>
                <y>2</y>
        </Add>
</soap:Body>
</soap:Envelope>
```

Elemen Body berisi sebuah elemen Add. Masing-masing parameter input dinyatakan sebagai sebuah subelemen dalam elemen Add. Umtan elemen x dan y hams sama dengan umtan penyebutan parameter dalam tkita tangan metode. Dengan kata lain, menempatkan y sebelum x akan menjadi invalid. Lebih jauh, nama-nama dan tipe-tipe elemen Add, x, dan y harus sama dengan metode target dan parameternya. Penulis akan menerangkan pembuatan tipe data dalam bab berikutnya. Untuk sekarang, cukuplah Penulis katakan bahwa tubuh pesan permintaan harus dalam format yang diinginkan oleh aplikasi remote.

Setelah Penulis membuat sebuah pesan permintaan yang telah diformat dengan benar, mari perhatikan respon yang dihasilkan oleh aplikasi remote:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
        <Add Result>
        <result>1</result>
        </Add Result>
</soap:Body>
```

```
</soap: Envelope>
```

Pesan respon yang dikembalikan oleh aplikasi remote berisi hasil dari metode Add. Parameter hasil tersebut dikodekan sekali lagi dalam format mirip stmktur di tubuh pesan SOAP. Aturan penama an subelemen dalam tubuh adalah nama metode dengan tambahan Result. Bagaimanapun juga, aturan penamaan ini tidak didiktekan oleh spesifikasi. Parameter pertama (dalam hal ini, adalah satu-satunya) berisi parameter panggilan metode. Seperti pada elemen *AddResult*, nama elemen yang memuat parameter kembalian tidak didiktekan oleh spesifikasi.

Bagaimana jika lebih dari satu parameter dikembalikan ke client? Mari kita amati sebuah variasi dari metode *Add*. *Add2* mengembalikan jumlah dari dua angka lewat sebuah parameter output.

```
public int Add2(int x,int y,out int sum)
{
Sum=x +y;
return sum;
}
```

Memanggil Add2(1, 2) menghasilkan pesan SOAP berikut ini:

```
<?xml version="l.O" encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
        <Add2>
                <x>l</x>
                <y>2</y>
        </Add2>
</soap:Body>
</soap:Envelope>
```

Perhatikan bahwa parameter ketiga, sum tidak dikodekan. Karena sum

dinyatakan sebagai sebuah parameter output, maka tidak ada alasan untuk mengirim nilai inisialnya ke aplikasi remote. Berikut ini adalah respon tersebut:

```
<?xml version="1.0" encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
        <Add2Response>
                <Add2Result>3</Add2Result>
                <sum>3</sum>
        </Add2Response>
</soap:Body>
</soap:Envelope>
```

Pesan jawaban berisi nilai kedua parameter. Sebagaimana yang telah dimaksudkan sebelumnya, parameter hasil harus selalu dicantumkan terlebih dulu. Penulis memanggil elemen berisi parameter hasil *Add2Result* untuk memperagakan bahwa namanya tidak *relevan*. Nilai parameter *sum* dicantumkan berikutnya.

SOAP Encoding

SOAP Encoding mendefinisikan cara menyerialkan data dalam sebuah pesan SOAP. SOAP Encoding dibangun atas tipe-tipe yang didefinisikan dalam spesifikasi XML yang mendefinisikan sebuah cara stkitar mengodekan data dalam dokumen XML. SOAP Encoding mengklarifikasi bagaimana data seharusnya dikodekan dan meliputi item-item yang tidak secara eksplisit tercakup dalam spesifikasi XML, seperti misalnya array dan bagaimana mengodekan referensi dengan tepat.

Tipe-Tipe Sederhana

Tipe-tipe sederhana termasuk berbagai string, integer, date/time, Boolean, dan banyak lagi. Spesifikasi SOAP menunda seksi "*Built in datatypes*" pada spesifikasi "XML Schema Datatypes ".

Sebuah persyaratan datatype dapat dikodekan sebagai sebuah elemen XML. Sebagai contoh, *integer* bernama *Age* akan dikodekan sebagai berikut:

```
<Age>31</Age>
```

Perhatikan bahwa untuk pesan-pesan RPC, nama elemen harus berkorelasi dengan nama parameternya.

Tipe Campuran

Seringkali, tidak cukup menyalurkan tipe-tipe sederhana, seperti misalnya integer dan string sebagai parameter. Kita perlu juga menyalurkan tipe-tipe campuran seperti misalnya struktur atau array. Dalam bagian ini, membahas bagaimana SOAP *Encoding* menangani tipe-tipe campuran.

Struktur

Struktur adalah sebuah koleksi tipe yang bertindak sebagai template bagi catatan pengelompokkan data secara logis. Sebagai contoh, anggaplah Kita perlu membuat sebuah fungsi yang menghitung volume sebuah balok padat. Daripada menyalurkan panjang, lebar, dan tinggi balok pada sebuah parameter terpisah, Kita dapat secara logika mengelompokkan data dimensi ke dalam sebuah struktur *RectSolid*. Kemudian, metode yang menghitung. volume isi dapat menerima

sebuah syarat struktur *RectSolid*. Berikut ini adalah sebuah contohnya:

```
Public struct RectSolid
{
public int length;
public int width;
public int height;
}
public int CalcVolume(RectSolid r)
[
return(r.length *r.width *r.height);
```

Pertama, Penulis definisikan sebuah struktur yang memuat dimensi-dimensi isi. Kemudian, Penulis mendefinisikan areanya. Permintaan untuk menghitung volume balok dengan panjang 2, lebar 3, dan tinggi 1 dapat dikodekan sebagai berikut:

```
<?xml version="l. O" encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
        <CalcVolume>
        <r>
                <length>2</length>
                <width>3</width>
                <height>l</height>
        </r>
        </CalcVolume>
</soap:Body>
</soap:Envelope>
```

Seperti yang dapat Kita lihat, struktur-struktur dipetakan dengan manis ke XML. Setiap *variabel* yang dimuat dalam syarat struktur *RectSolid* diserialkan sebagai sebuah elemen anak dari r.

Array

Data campuran lainnya yang umum adalah array. Pada saat penulisan ini, spesifikasi XML tidak menyebutkan cara sebuah array dikodekan. Spesifikasi SOAP 1.1 mengisi kekosongan ini. Contohnya adalah sebagai berikut:

```
Public int Add Array (int [] numbers))
{
int total=0;
foreach(int number in numbers)
{
total +=n umber;
        }
return total;
```

Metode AddArray menerima sebuah array integer dan mengembalikan total.

Berikut ini adalah caranya supaya client dapat memanggil fungsi AddArray:

```
int [] a == {1,2,3);
int total;
total =AddArray (a);
```

Panggilan ke AddArray menghasilkan pesan permintaan berikut:

```
<?xml version="1.0" encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/"
xmlns:soap-enc=http://schemas.xml soap.org/soap/encoding/
xmlns::xsi="http://www.w3.org/2001/XMLSchema-instance">
<soap:Body>
        <AddArray>
        <a soap-enc:arrayType="xsi:int[3]">
        <int>l</int>
```

```
        <int>2</int>
        <int>3</int>
        </a>
</AddArray>
</soap:Body>
</soap:Envelope>
```

Array dinyatakan dengan sebuah elemen tunggal dalam tag tubuh. Elemen tersebut harus memuat atribut soap-enc: *arrayType*. Nilai atribut menjelaskan isi array dan dimensinya. Dalam contoh awal, xsi:int [3] menyebutkan bahwa array berisi tiga integer.

Setiap nilai dalam array dicantumkan sebagai subelemen. Nama-nama subelemen tidak relevan, namun seringkali nama-nama elemen dalam array akan berkorelasi dengan tipe data yang mereka muat.

Array-array SOAP *encoded* dapat memuat elemen berbeda pada tipe-tipe berbeda. Kode berikut menghasilkan sebuah array berisi sebuah *integer*, *float*, dan *string*:

```
object [] stuff ==new object [3];
stuff [0] == (int)100;
stuff [1] == (float)2.456;
stuff [2] == (string)"Kitchen Sink";
CollectThings(stuff);
public void CollectThings(object [] things))
{
        //...
}
```

Array objek yang disebut stuff dibuat dan kemudian nilai ketiga tipe diberikan ke masing-masing elemennya. Pesan respon SOAP yang dihasilkan akan dikodekan sebagai berikut:

```
<?xml version="1.O" encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xml soap.org/soap/encoding/"xmlns::xsi="http://
www.w3.org/2001/XMLSchema-instance">
<soap:Body>
<CollectThings>
        <things soap-enc:arrayType="xsi:ur-type[3 ]">
                <object>lOO</object>
                <object>2.456</object>
                <object>Kitchen Sink</object>
        </things>
</CollectThings>
</soap:Body>
</soap:Envelope>
```

Array things didefinisikan sebagai tipe xsi: ur-type, yang berarti bahwa elemen- elemen tersebut dapat berisi data sembarang tipe.

Dua tipe array terakhir yang akan Penulis bahas adalah array multidimensi dan array bersegi banyak (bergerigi). Array multidimensi sifatnya persegi. Kita dapat menganggap sebuah array bergerigi sebagai sebuah array di dalam array. SOAP mendefinisikan sebuah metode untuk mengodekan kedua tipe array tersebut. Contoh ini membuat sebuah array multidimensi:

```
//Create a block of seats 3 rows deep and 4 seats wide.
string [.] seats ==new string [3,4 ];
for(int i =O;i <2;i ++)
(
for(int j =O;i <2;j++)
(
seats [i .j ] ==string.Format("row (O).seat {1} ");
                }
```

```
}
PrintSeatLabels(seats);
public void PrintSeatLabels(string [.] labels))
(
// . . .
        }
```

Array multidimensi label dibuat dan kemudian array disalurkan ke PrintSeat Labels. Pesan yang dihasilkan dikodekan sebagai berikut:

```
<?xml version="l.O" encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xml soap.org/soap/encoding/"xmlns::xsi="http://
www.w3.org/2001/XMLSchema-instance">
<soap:Body>
<PrintSeatlabels soap-enc:arrayType="xsi:string[3,4 ]">
<seats>
<string>row l,seat l</string>
<string>row l,seat 2</string>
<string>row l,seat 3</string>
<string>row l,seat 4</string>
<string>row 2,seat l</string>
<string>row 2,seat 2</string>
<string>row 2,seat 3</string>
<string>row 2,seat 4</string>
<string>row 3,seat l</string>
<string>row 3,seat 2</string>
<string>row 3,seat 3</string>
<string>row 3,seat 4</string>
</seats>
</PrintSeatlabels>
</soap:Body>
</soap:Envelope>
```

Seperti yang dapat Kita lihat, nilai-nilai elemen sisi kanan berubah lebih cepat daripada yang sebelah kiri. Karena seat adalah elemen paling kanan, semua seat untuk baris tertentu dikodekan sebelum loop berlanjut ke baris berikutnya.

Dalam sebuah array bergerigi yang dapat dianggap sebagai sebuah array dalam array, setiap elemen dapat memuat sebuah array dengan panjang berbeda. Berikut ini sebuah contohnya:

```
string [ ][ ] teams ==new string [3 ][ ];
teams [0 ] ==new string [3 ];
teams [0 ][0 ] =="Bob";
teams [0 ][1 ] =="Sue";
teams [0 ][2 ]  =="Mike";
teams [1 ] ==new string [2 ];
teams [1 ][0 ] =="Jane";
teams [1 ][1 ] =="Mark";
teams [2 ] ==new string [4 ];
teams [2 ][0 ] =="Mary";
teams [2 ][1 ] =="Jill";
teams [2 ][2 ] =="Jim";
teams [2 ][3 ] =="Tom";
RegisterTeams(teams);
Public voidRegisterTeams(string [][] teams))
{
        // . . .
}
```

Fungsi *RegisterTeams* menerima sebuah daftar tim. Karena para pemain tim dapat berbeda, sebuah array string bergerigi *multidimensi* disalurkan ke fungsi terebut. Setiap elemen array menyatakan sebuah tim dan berisi sebuah array mapemain di tim itu. Berikut ini adalah bagaimana array bergerigi tersebut dikodekan:

```
<?xml version="1.0 " encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xml soap.org/soap/encoding/"xmlns::xsi="http://
www.w3.org/2001/XMLSchema-instance">
<soap:Body>
<RegisterTeams>
<teams soap-enc:arrayType="xsi:string [3 ]">
<team soap-enc:arrayType="xsi:string [3 ]">
<player>Bob</player>
<player>Sue</player>
<player>Mike</player>
</team>
<team soap-enc:arrayType="xsi:string [2 ]">
<player>Jane</player>
<player>Mark</player>
<!team>
<team soap-enc:arrayType="xsi:string [4 ]">
<player>Mary</player>
<player>Jill</player>
<player>Jim</player>
<player>Tom</player>
</team>
</teams>
</RegisterTeams>
</soap:Body>
</soap:Envelope>
```

Konsisten dengan aturan pengodean array yang telah Penulis bahas sebelumnya, nama elemen ini tidaklah penting. Supaya lebih jela s, Penulis menamai setiap elemen dalam array teams dengan team dan menamai setiap elemen dalam array team dengan player. Dalam array bergerigi, tidak hanya elemen teams yang berisi atribut soap-enc:

arrayTypeAttribute, namun juga setiap elemen dalam array tim. Yang berisi atribut soap-enc: *arrayType*.

Optimisasi

Dalam banyak kasus, Kita mungkin tidak ingin mengodekan seluruh array dalam tubuh pesan. Spesifikasi SOAP 1.1 menjelaskan dua cara untuk mengodekan sebuah array; array parsial dan array ringan. Array parsial mengodekan kisaran elemen tertentu ke dalam array. Array ringan mengodekan elemen-elemen pilihan yang tersebar ke seluruh array.

Misalnya saja Kita ingin membuat array yang dapat menampung nama sampai 1.000 pendaftar untuk sebuah acara yang akan berlangsung. Secara berkala, daftar peserta perlu dikirimkan ke berbagai pihak yang berkepentingan. Segera setelah acara diumumkan, boleh jadi hanya ada 5 orang yang terdaftar. Jika Kita mengirimkan pendaftar, tidaklah efisien mengodekan 1.000 elemen tersebut karena baru ada 5 saja yang akan berisi nilai:

```
//Create an array of attendees.record the first five.
//and then pass the array to RegisteredAttendees.
string [] a ttendees [1000 ];
attendees [0 ] =="Bi11 Clinton";
attendees [1 ] =="Jimmy Carter";
attendees [2 ] =="Ronald Reagan";
attendees [3 ] =="George Bush";
attendees [4 ] =="AlGore";
RegisteredAttendees(attendees);
Public void RegisteredAttendees(string [] attendees))
(
```

```
// . . .
        }
<?xml version="1.0 " encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xml soap.org/soap/encoding/" xmlns::xsi="http://
www.w3.org/2001/XMLSchema-instance">
<soap:Body>
<Registered Attendees>
<attendees soap-enc:arrayType="xsi:string [1000 ]">
<string>Bill Clinton</string>
<string>Jimmy Carter</string>
<string>Ronald Reagan</string>
<string>George Bush</string>
<string>Al Gore</string>
</attendees>
</Registered Attendees>
</soap:Body>
</soap:Envelope>
```

Seperti yang Kita lihat dalam pesan yang dihasilkan, atribut soap-enc: array Type menunjukkan bahwa array tersebut berisi 1.000 elemen walaupun baru 5 yang dikodekan. Jika Kita ingin mengodekan sebagian dari array yang tidak dimulai dengan elemen pertama, Kita dapat menyebutkan elemen awalnya dengan menggunakan atribut soap-enc: offset. Sebagai contoh, jika Kita ingin mengodekan lima hadirin pertama yang terdaftar untuk acara tersebut, pesan yang dihasilkan akan menjadi seperti ini:

```
<?xml version="1.0 " encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xml soap.org/soap/encoding/" xmlns::xsi
="http://
```

```
www.w3.org/2001/XMLSchema-instance">
<soap:Body>
<Registered Attend ees >
<attendees soap-enc:arrayType="xsi:string [1000 ]"
soap-enc:offset="[5 ]">
<string>Gerald Ford</string>
<string>George W.Bush</string >
<string>Dick Cheney</string>
<string>Walter Mondale</string >
<string>Dan Quayle</string>
</attendees>
</RegisteredAttendees>
</soap:Body>
</soap:Envelope>
```

Seperti yang dapat Kita lihat, elemen soap-enc: offset menyebutkan bahwa array telah diimbangi sebanyak lima. Oleh karena itu, isi array berisi enam sampai sepuluh elemen.

Bagaimana jika elemen-elemen yang akan dikodekan dalam array tersebut tidak kontinyu satu sama lain? Cara pengodean parsial lainnya adalah menggunakan sintaks array ringan. Sebagai contoh, katakanlah Penulis ingin membuat sebuah pesan berisi nama semua hadirin yang telah terdaftar untuk menghadiri acara. Ordina l setiap hadirin mempunyai signifikansi. Jadi, Kita buat sekali lagi 1.000 elemen dalam sebuah array dan mengumpulkan subset elemennya saja bersama data. Saat ini, data tersebut tidak akan ditempatkan dalam urutan set elemen. Melainkan, data itu akan dimuat dalam elemen-elemen ke seluruh array. Kita dapat mengatasi masalah ini dengan mengodekan sebuah array untuk yang tidak hadir dengan memakai sintaks array ringan.

Berikut ini pesan hasilnya:

```
<?xml version="1.0 " encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xml soap.org/soap/encoding/" xmlns::xsi
="http://
www.w3.org/2001/XMLSchema-instance">
<soap:Body>
<NoShows>
<registrants soap-enc:arrayType="xsi:string [1000 ]">
<string soap-enc:position="[10 ]">Dan Quayle</string>
<string soap-enc:position="[231 ]">Newt Gingrich</string>
<string soap-enc:position="[357 ]">Trent Lott</string>
<string soap-enc:position="[842 ]">Hillary Rodham Clinton
</string>
</registrants>
</NoShows>
</soap:Body>
</soap:Envelope>
```

Sekali lagi, atribut soap-enc: arrayType digunakan untuk menyebutkan bahwa array berisi total 1.000 elemen. Bagaimanapun juga, elemen-elemen yang berisi data adalah elemen-elemen yang dikodekan saja dalam pesan SOAP. Karena posisi elemen dalam array relevan, atribut soap-enc. position digunakan untuk menkitai ke mana elemen ditempatkan dalam array tersebut.

Menyalurkan Parameter dengan Referensi

Sampai saat ini, Penulis telah menerangkan bagaimana mengodekan parameter-parameter yang disalurkan berdasar nilai ke sebuah layanan Web. Namun, biasanya perlu juga menyalurkan parameter berdasar

referensi. Sebagai contoh, sebuah client mungkin perlu menyalurkan informasi mengenai seorang pelanggan ke server agar server dapat mengupdate informasi atas nama client tersebut. Jika struktur client disalurkan berdasar nilai, perubahan yang dibuat ke informasi client tidak akan terlihat pada client.

Mari kita perhatikan bagaimana sebuah parameter yang disalurkan berdasar referensi dikodekan dalam sebuah pesan SOAP. Dalam contoh pertama, Penulis buat serangkaian angka *Fibonacci*. Angka dalam deret *Fibonacci* ditentukan dengan menambahkan dua angka langsung yang mendahuluinya. Sebagai contoh, nl = 1 dan n2 = 1, maka n3 = 1 + 1 = 2 dan n4 = 1 + 2 = 3. inilah metode yang Penulis gunakan untuk menghasilkan deret *Fibonacci*:

```
public void FibonaccilNcrement(ref int nl,ref int n2)
{
int temp =n2;
//Set nl and n2 to the next two Fibonacci numbers.
nl +=n2;
n2 =temp +nl;
}
//The following code prints the following output:
//1,1,2,3,5,8,13,21,34,55,
int x =l;
int y =l;
for(int i =1,i <11.i +=2)
{
Console.Write("{O},{l}",x ,y);
FibonaccilNcrement(x,y);
}
```

FibonaccilIncrement menerima dua angka terakhir dan kemudian menghasilkan dua angka berikutnya dalam deret tersebut. Inilah pesan-pesan permintaan dan jawaban untuk panggilan pertama ke *FibonaccilIncrement*:

```
<!--Request Message-->
<?xml version="l.0 " encoding=="utf-8"?>
<soap:Envelope xmlns:soap= "http://schemas.xml soap.org/soap/envelope/">
<soap:Body>
<FibonacciIncrement>
<nl>l</nl>
<n2>1</n2>
</FibonacciIncrement>
</soap:Body>
</soap:Envelope>
<!--Request Message-->
<?xml version="l.0 " encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/">
<soap:Body>
<FibonacciincrementResponse>
<n1>2</nl>
<n2>3</n2>
</FibonacciincrementResponse>
</soap:Body>
</soap:Envelope>
```

Tidak ada yang mengejutkan dengan pesan pertama. Kedua parameter dikodekan seperti biasa. Apa yang membedakan sebuah parameter yang dibawa oleh referensi dari parameter yang dibawa oleh nilai adalah karena client perlu diberitahu setiap perubahan pada nilai tersebut. Oleh karena itu, nilai-nilai baru untuk nl dan n2 dikodekan dalam pesan jawaban. Perhatikan bahwa Penulis juga mengikuti aturan

untuk penambahan Response ke elemen metode dalam tubuhnya.

Alasan lain untuk menyalurkan parameter dengan referensi adalah untuk menjaga identitas variabel yang disalurkan. Perhatikan contoh berikut:

```
//Server Code:
public struct Person
{
public double Height;
public int Weight;
public int Age;
public string Hobby;
}
public string Introduce(ref Person pl.ref Person p2)
(
string result ="";

//Do pl and p2 reference the same variable?
if(pl.ReferenceEquals(p2))
{
throw new Exception("Can "t introduce to self.");

//Are pl and p2 equal in value?
if(pl.Equals(p2))
{
result ="We have a lot. in common!";
}
else
{
result ="Nice to meet you .";
}
return result;
}
```

```
//Client Code:
Person p =new Person();
p.Height =5.7;
p .Weight =150;
p .Age =31 ;
p .Hobby ="Skiing";
//Attempt to introduce a person to himself.
Introduce(ref p,ref p );
```

Metode Introduce menerima dua referensi untuk variabel-variabel bertipe Person. Ini serupa dengan penyaluran dua integer dengan referensi ke Fibonacci Increment. Bagaimanapun juga, tidak seperti Fibonacciincrement, metode Introduce berperilaku berbeda bergantung pada apakah kedua parameter sama atau indentik (menunjuk ke syarat Person yang sama).

Cara mengodekan parameter yang disalurkan dengan referensi dalam contoh *Fibonacci* tidaklah cukup untuk metode Introduce karena metode ini cidak menjaga identitas parameter. SOAP menyediakan polaid/href untuk menjaga identitas parameter. Beginilah caranya mengodekan panggilan ke Introduce:

```
<?xml version="l.O " encoding=="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xml soap.org/soap/encoding/">
<soap:Body>
<Introduce>
<pl soap-enc:href="#refl"/>
<p2 soap-enc:href="#refl"/>
</Introduce>
<Person soap-enc:id="refl">
<Height>5.7</Height>
```

```
<Weight>150</Weight>
<Age>31</Age>
<Hobby>Skiing</Hobby>
</Person>
</soap:Body>
</soap:Envelope>
```

Parameter yang dikodekan tidak berisi data apapun. Sebagai gantinya, karena kedua parameter merujuk syarat tipe Person yang sama, maka data hanya dikodekan sekali dalam tubuh pesan. Elemen root data parameter diberi sebuah ID unik lewat atribut id.

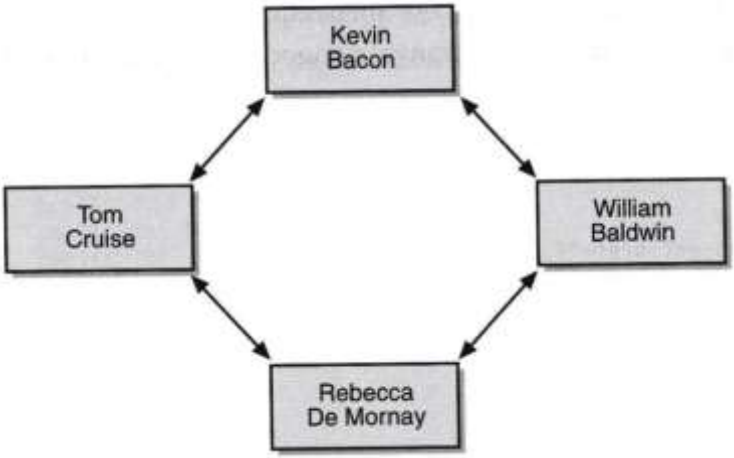
Daripada memuat datanya sendiri, setiap parameter merujuk elemen berisi sesungguhnya. Ini dilakukan dengan mengatur atribut href elemen parameter dengan ID elemen berisi data tersebut.

Seperti yang akan Kita lihat dalam bab-bab selanjutnya, teknologi-teknologi-NET yang berbeda mempunyai beragam tingkat dukungan dalam cara mereka mengodekan parameter-parameter referensi. Bagian ini menunjukkan kepada Kita perlunya memahami tingkatan tersebut, yang mana dengan referensi pengodean didukung oleh teknologi yang mendasari aplikasi Kita. Pemahaman ini dapat membantu menghindari perilaku yang tidak diharapkan dalam aplikasi Kita.

Atribut Root

Kadang kala, root dari bagan objek yang telah diserialkan belum siap muncul dalam pesan SOAP yang dihasilkan. Anggap saja, Kita ingin menyerialkan bagan objek berikut yang memperlihatkan hubungan

antara Kevin Bacon dan aktor lainnya:



Gambar 2.4 Hubungan antar actor

Bagan tersebut menya jikan dua path dari Kevin Bacon ke Rebecca De Mornay. Rebecca De Mornay bermain di Risky Business (1983) bersama Tom Cruise, dan Tom Cruise bermain di A Few Good Men 0992) bersama Kevin Bacon. Rebecca De Mornay juga bermain dalam Backdraft (1991) bersama William Baldwin, dan William Baldwin bermain dalam Flatliners (1990) bersama Kevin Bacon. Langka h selanjutnya adalah menyerialkan data ini ke dalam sebuah pesan SOAP.

Setelah objek bagan ini diserialkan, Kita tidak lagi dapat membedakan elemen mana yang merupakan elemen root. Untuk hal ini, SOAP Encoding mendefinisikan atribut root. Kita dapat menggunakan atribut ini untuk membedakan root serialisasi dengan elemen lain yang ada dalam sebuah serialisasi, tap·bukan root bagan nilai yang diserialkan.

Bagan objek terdahulu akan diserialkan seperti yang terlihat di sini:

```
<?xml version="l.O " encoding== "utf-8 "?>
<soap:Envelope xmlns:soap="http://schemas.xml soap.org/soap/envelope/"
xmlns:soap-enc="http://schemas.xml soap.org/soap/encoding/"
xmlns:hw="urn:hollywood>
<soap:Body>
<RelationsToKevinBaconResult>
<objectGraph soap-enc:href="#actorl"/>
</RelationsToKevinBaconResult>
<Actor soap-enc:id="actorl " soap--enc:root="l">
<Name>Kevin Bacon</Name>
<Relationships soap-enc:arrayType="hw:Actor [2 ]">
<Actor soap-enc:href="actor2"/>
<Actor soap-enc:href="actor3"/>
</Relationships>
</Actor>
<Actor soap-enc:id="actor2">
<Name>Tom Cruise</Name>
<Relationships soap-enc:array Type="hw:Actor [2 ]">
<Actor soap-enc:href="actorl"/>
<Actor soap-enc:href="actor4 "/>
</Relationships>
</Actor>
<Actor soap-enc:id="actor3">
<Name> William Baldwin</Name>
<Relationships soap-enc:array Type="hw :Actor [2 ]">
<Actor soap-enc:href="actorl"/>
<Actor soap-enc:href="actor4"/>
</Relationships>
</Actor>
<Actor soap-enc:id="actor4">
<Name>Rebecca De Mornay</Name>
<Relationships soap-enc:arrayType="hw:Act or [2 ]">
```

```
<Actor soap-enc:href="actor2"/>
<Actor soap-enc:href="actor3"/>
</Relationships>
</Actor>
</soap:Body>
</soap:Envelope>
```

Atribut root mengenali Kevin Bacon sebagai root dalam bagan objek aktor. Penulis juga dapat menghias objek-objek non root dengan atribut root dan mengaturnya ke 0.

Pengikatan Protokol

Kita telah belajar tentang cara mengodekan sebuah pesan SOAP dengan benar. Namun, Kita masih membutuhkan cara untuk mengirimkan pesan ke aplikasi remote. Satu kelebihan dari SOAP adalah karena tidak terikat pada sebuah protokol transpor tertentu. Pesan-pesan SOAP dapat dikirim lewat sembarang protokol transpor yang mampu mengangkut XML.

Banyak orang mengatakan bahwa protokol transpor paling populer yang digu- nakan untuk mengirim pesan-pesan SOAP adalah HTTP. Bagaimanapun juga, pesan-pesan SOAP juga dapat dikirim lewat SMTP atau fax ke sebuah kapal lewat radio gelombang pendek atau apa pun lainnya yang dapat dibayangkan.

Bagaimana sebuah pesan SOAP dibawa oleh protokol transpor tertentu dikenal sebagai pengikatan protokol (protocol-binding). Pengikatan protokol dapat didefinisikan untuk mengeksploitasi suatu karakteristik unik pada protokol transpor. Seperti yang sebentar lagi Kita pelajari,

pengikatan HTTP POST akan memperluas protokol sehingga firewall HTTP-aware dapat menyaring pesan-pesan SOAP.

Spesifikasi SOAP menerangkan hanya satu pengikatan protokol: mengirim pesan-pesan SOAP lewat HTTP POST. Oleh karena itu, satu-satunya pengikatan protokol yang akan Penulis bahas adalah HTTP POST.

Sebagian besar pelaksanaan SOAP, termasuk .NET, mendukung protokol HTTP. Karena kebanyakan sistem mendukung HTTP, wajar saja protokol tersebut dijadikan pilihan untuk memastikan bahwa layanan Web mempunyai tingkat kerja sama yang tinggi di antara bermacam-macam platform. Kelebihan dari protokol HTTP di antaranya:

1. **Firewall-friendly,** protokol-protokol lama seperti misalnya DCOM (*Distributed Component Object Model*) tidak termasuk. Sebagian besar firewall mempunyai port 80, setidaknya yang terbuka untuk lalu lintas HTTP.
2. **Mempunyai infrastruktur pendukung yang mantap.** Sudah diperkenalkan banyak teknologi untuk meningkatkan kemampuan penyesuaian dan penemuan aplikasi-aplikasi berbasis-HTTP.
3. **Bersifat stateless secara inheren.** Sifat stateless dari HTTP membantu memastikan bahwa komunikasi antara client dan server terjaga, khususnya lintas internet. Koneksi-koneksi yang tiba-tiba drop mengakibatkan permasalan bagi protokol-protokol seperti DCOM dan CORBA

Sederhana. Protokol HTTP terdiri dari sebuah bagian header dan

sebuah bagian tubuh.

1. **Dipetakan dengan manis pada pertukaran pesan bergaya RPC**. HTTP adalah protokol alami bagi komunikasi bergaya RPC karena permintaan selalu dibarengi dengan sebuah jawaban.
2. **Bersifat terbuka.** Pada prakteknya setiap sistem jaringan mendukung HTTP.

Sebuah permintaan HTTP terdiri dari dua bagian, header dan tubuh. Header berisi informasi mengenai permintaan dan client yang mengirimkan permintaan cersebut. Tubuh mengikuti header dan dibatasi oleh dua tkita paragraf baru (*linefeed*). Tubuh berisi muatan yang dalam hal ini dapat berupa pesan SOAP. Berikut ini sebuah contoh permintaan HTTP berisi pesan SOAP:

```
POST /SomeWebService HTTP/1.1
Content-Type:text/xml
SOAPAction:"http://somedomain.com/SomeWebService.wsdl"
Content-Length: 243
Host : sshort3
<?xml version="1.0 " encoding=="utf- 8 "?>
<soap:Envelope xmlns:soap=http://schemas.xml soap.org/soap/envelope/
xmlns:soap-enc="http://schemas.xml soap.org/soap/encoding/">
<soap:Body>
<Add>
<x>2</x>
<y>2</y>
</Add>
</soap:Body>
</soap:Envelope>
```

Header HTTP untuk sebuah pesan SOAP serupa dengan permintaan

HTTP dengan sepasang perbedaan; entri header Content-Type selalu diatur ke text/xml, dan tubuh pesan berisi pesan SOAP. Perbedaan lain adalah setiap permintaan SOAP HTTP POST harus berisi entri header SOAP Action.

Entri header SOAP Action dipakai untuk menyampaikan maksud pesan SOAP. URI dapat dinyatakan dalam sembarang format dan tidak perlu dipecahkan. Dalam contoh Penulis, URL dicocokkan pada dokumen WSDL (*Web Services Description Language*) untuk layanan Web.

Nilai header SOAP Action dapat dikosongkan jika maksud pesan SOAP dimuat dalam entri header permintaan HTTP. Permintaan HTTP adalah entri pertama dalam header dan berisi aksi (dalam hal ini selalu POST) dan URL targetnya. Jika dalam entri header permintaan HTTP menyampaikan dengan memadai maksud dari pesan SOAP, entri-entri berikut akan menjadi valid:

```
SOAPActi on : ‟”
SOAPActi on :
```

Respon HTTP digunakan untuk menyampaikan hasil dari permintaan SOAP.

```
HTTP/1.1 200 OK
Server:Microsoft-IIS/5.0
Date:Sun, 25 Mar 2001 19:44 :55 GMT
Content-Type:text/xm l Content-Length:243
<?xml version= "l.O " encoding=="utf-8 "?>
<soap:Envelope xmlns:soap="http ://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<Add Response>
```

```
<result>4</result>
</Add Response>
</soap:Body>
</soap:Envelope>
```

Sekali lagi, tipe MIME diatur ke text/xml. Untuk pesan-pesan bergaya-RPC, tubuh HTTP berisi pesan jawaban SOAP. Jika tidak, tubuh HTTP akan dikosongkan. Karena contoh pesan permintaan menghasilkan sebuah pesan jawaban yang akan dibuat dari server, tubuh HTTP berisi hasilnya.

Status yang dilaporkan dalam baris pertama pada header HTTP harus berisi nilai antara 200 dan 299. Jika terjadi kesalahan saat pemrosesan pesan SOAP, status HTIP harus menjadi 500, sehingga menunjukkan terjadinya kesalahan server internal.

C. PENUTUP

Ringkasan

Dalam bab ini, Kita telah mempelajari protokol messaging yang mendasari layanan Web, yaitu SOAP. Lebih khusus lagi, tentang yang berikut ini:

1. **Amplop SOAP**. Bagaimana penggunaannya untuk mengodekan informasi header tentang pesan dan tubuh pesan itu sendiri.
2. **SOAP Encoding**. Bagaimana data dapat diserialkan ke dalam sebuah pesan SOAP.
3. **Pesan-pesan RPC**. Bagaimana pesan-pesan RPC memfasilitasi komunikasi berorientasi prosedur lewat pola-pola permintaan /jawaban Pengikatan protokol HITP POST. Bagaimana pesan-

pesan SOAP dapat dibawa lewat HTTP.

Kita telah mempela jari; setidaknya sebuah pesan SOAP harus dimuat dalam sebuah amplop SOAP well-formed. Sebuah amplop dapat terdiri dari elemen Envelope tunggal. Amplop tersebut dapat memuat elemen Header dan harus berisi sebuah elemen Body. Jika ada, header harus berupa elemen anak langsung dalam amplop, dengan tubuh mengikuti headernya. Tubuh berisi muatan pesan, dan header berisi data tambahan yang tidak perlu ada dalam tubuh pesan.

Di samping mendefinisikan sebuah amplop SOAP, spesifikasi SOAP juga mendefinisikan sebuah cara untuk mengodekan data yang dimuat dalam bagian 1 dari spesifikasi XML Schema. Ini mencakup berbagai array dan referensi untuk syarat tipe-tipe data.

Spesifikasi SOAP juga menyediakan sebuah pola pesan stkitar untuk memfasilitasi perilaku bergaya-RPC. Dua pesan SOAP dipadukan bersama untuk memfasilitasi sebuah pesan permintaan dan jawabannya.

Panggilan metode dan parameternya diserialkan dalam tubuh pesan permintaan dalam bentuk struktur. Elemen root membawa nama yang sama seperti metode target, dengan masing-masing parameter tak terikat dikodekan dalam sebuah subelemen.

Pesan jawaban akan memuat hasil panggilan metode atau sebuah struktur kesalahan yang terdefinisi dengan baik. Hasil panggilan metode diserialkan dalam tubuh permintaan sebagai sebuah struktur. Menurut peraturan, elemen root membawa nama yang sama dengan panggilan metode aslinya ditambah Result. Parameter kembalian akan

diserialkan sebagai elemen anak, dengan parameter kembalian muncul pertama-tama. Jika ditemui sebuah kesalahan, tubuh pesan jawaban akan memuat sebuah struktur kesalahan yang terdefinisi dengan baik.

Pertanyaan

1. Jelaskan inti dari layanan web Simple Object Access Protocol (SOAP)?
2. Apa Fungsi penggunaan elemen header?
3. Sebutkan dan jelaskan Rute SOAP?
4. Sebutkan dan jelaskan kode kode kesalahan dasar SOAP?
5. Sebutkan dan jelaskan kelebihan dari protokol HTTP?

E. DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.

9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 3
MEMBUAT LAYANAN WEB DASAR

A. PENDAHULUAN

1. Deskripsi Singkat :

Pada mata kuliah ini khususnya pada BAB 3 akan di bahas Membuat dan Menggunakan Layanan Web diantaranya: Memahami Layanan Web Berbasis XML, SOAP dan Layanan Web, Kebutuhan Tambahan Layanan Web, Membuat sebuah Layanan Web, Mendeklarasikan Layanan Web, Membuat Kelas Layanan Web, Mewarisi dari Web Services, Menggunakan Atribut untuk menyebutkan Namespace XML, Atribut Metadata, Menggunakan Atribut untuk Mengekspos Metode, Properti Atribut, Membuat Layanan Web Hello World, Menggunakan DataType Primitif, Mengakses Data dalam Layanan Web, Mengiklankan Layanan Web, Mengamankan Layanan Web.

2. Kemampuan Akhir yang diharapkan :

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan dan Memahami Layanan Web Berbasis XML, SOAP dan Layanan Web, Kebutuhan Tambahan Layanan Web, Membuat sebuah Layanan Web, Mendeklarasikan Layanan Web, Membuat Kelas Layanan Web, Mewarisi dari Web Services, Menggunakan Atribut untuk menyebutkan Namespace XML, Atribut Metadata, Menggunakan Atribut untuk Mengekspos Metode, Properti Atribut, Membuat Layanan Web Hello World, Menggunakan DataType Primitif,

Mengakses Data dalam Layanan Web, Mengiklankan Layanan Web, Mengamankan Layanan Web..

B. PENYAJIAN

Membuat Layanan Web Dasar

Dalam bab ini, akan ditunjukkan cara membuat beberapa layanan Web. Dalam bab-bab selanjutnya yang membahas protokol-protokol pokok, Kita akan menjadi lebih tahu berbagai kerumitan yang dapat diatasi oleh platform.NET secara otomatis. Dalam bab ini, banyak membuat referensi ke teknologi dasar supaya Kita memiliki pemahaman dasar tentang kapan harus menggunakan teknologi teknologi tersebut dalam ukuran yang lebih besar.

Dalam contoh pertama, Kita akan membuat aplikasi komersial dalam Microsoft ASP.NET, yaitu Web Form untuk mengumpulkan informasi pembayaran. Kemudian Kita akan membuat layanan Web yang melaksanakan logika bisnis pengesahan kartu kredit dengan mewakili aplikasi. Contoh ini menunjukkan kepada Kita, begitu mudahnya meningkatkan aplikasi.NET yang ada untuk memindahkan logika bisnis ke dalam layanan Web, sehingga dapat digunakan oleh aplikasi lain.

Dalam contoh kedua, Kita akan membuat layanan Web untuk mengirim dan menerima file biner. Karena XML adalah bahasa markup berbasis teks, maka kita dapat memasukkan kode ke dalam pesan yang dapat menyenangkan orang, yaitu isi dari file biner. File biner terdapat dalam tipe yang kompleks. Tipe yang kompleks juga

berisi informasi tentang file tersebut. Contoh ini menunjukkan dasar .NET Framework demikian besar.

Kedua sekenario ini dapat menjelaskan kepada Kita sejumlah tawaran yang diberikan oleh platform .NET. Dalam bab ini, akan ditunjukkan juga kemampuan pengembangan aplikasi yang cepat dalam Microsoft Visual Studio .NET yang dapat mempermudah pengembangan aplikasi dengan layanan Web. Karena cara belajar terbaik adalah dengan praktek, Penulis menganjurkan kepada Kita untuk langsung menggunakan Visual Studio.NET dan mengikuti tahap-tahap yang Penulis jelaskan untuk membuat contoh aplikasi sendiri.

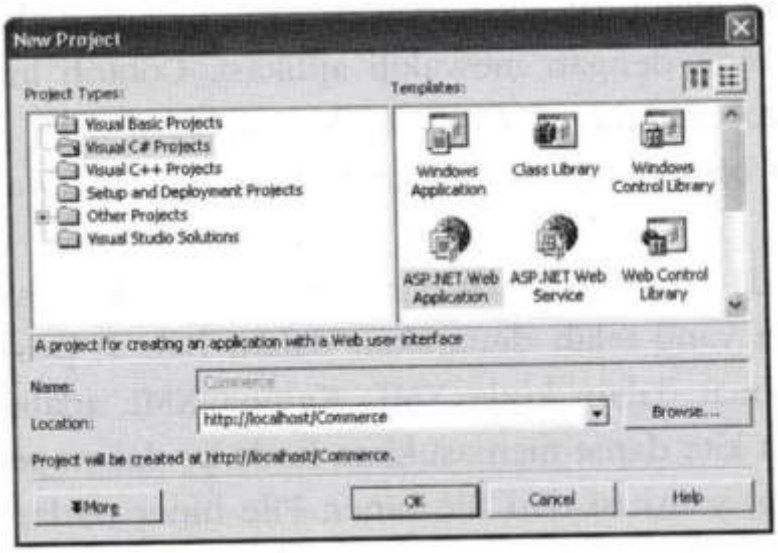
Aplikasi Komersial Sederhana

Pertama-tama, Kita akan membuat Web Form ASP.NET yang mengumpulkan dan melakukan pengesahan informasi kartu kredit. Dengan Web Form, pengguna dapat memasukkan informasi kartu kredit. Web Form akan memberitahukan kepada pengguna, apakah informasi yang mereka masukkan itu sah. Setelah itu, Kita akan mengalihkan logika pengesahan kartu kredit itu ke dalam layanan Web yang akan memodifikasi Web Formnya. Akhirnya, layanan Web akan terpanggil untuk melakukan pengesahan informasi kartu kredit tersebut.

Membuat Web Form

Kita dapat membuat Web Form dengan membuka Visual Studio.NET. Selanjutnya, Kita dapat membuat proyek Web Application baru. Ganti

tipe proyek Kita dengan pilihan bahasa Kita lalu pilih template Web *Application*. Namai proyek Kita dengan *Commerce* dan ganti lokasi URL yang menuju ke *server* Web target, seperti yang ditunjukkan pada Gambar 3.1 berikut ini:

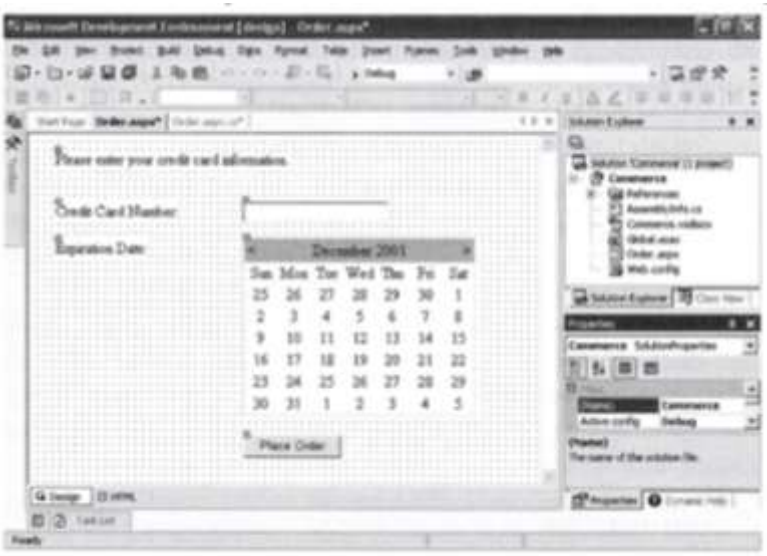


Gambar 3.1. ASP.NET Web Application

Visual Studio.NET membuat proyek pada *server* Web yang ditentukan dan menaruh template file aplikasi Web ke dalam direktori itu. Di dalam file ini terdapat satu halaman Web Form bernama **WebForml.aspx**. Ganti namanya dengan **Order.aspx** lalu ganti nama kelas yang didefinisikan dalam **Order.aspx** dari **WebForml** menjadi **Order**. Ikuti langkah berikut ini:

- Buka file *code-behind* Order.aspx dengan mengklik kanan nama file dalam Solution Explorer kemudian pilih *View Code*.

- Ganti nama kelas *WebForml* beserta konstruktornya dengan nama *Order*.
- Di dalam panel Properties, ganti nama labelnya dengan Status, teks untuk label tersebut; *Please enter your credit card information*, nama tombolnya dengan *PlaceOrder*, dan teks pada tombol tersebut dengan *Place Order*. Setelah Kita mengatur layout dan memasukkan teks tambahan, Web Form harus terlihat seperti ditunjukkan pada Gambar 3.2 ini:



Gambar 3.2 Web Form

Kemudian, siapkan pelaksanaan untuk tombol *Place Order*. Untuk menambahkan *eventhandler* tombol pemesanan itu, klik dua kali tombol itu dan masukkan kode berikut ini, supaya saat pengguna mengklik tombol itu, panel *Validate* akan dipanggil untuk menentukan

apakah informasi kartu kredit yang dimasukkannya itu sah. Kode ini juga akan mengatur teks label untuk menampilkan informasi berupa teks kepada pengguna mengenai keabsahan kartu kreditnya.

```
Public void PlaceOr der_Click(object sender.Syste.EventArgs e)
{if (Validate (TextBoxl.Text,Calendarl.SelectedDate))
{Status.Text="Thank you for your order.";}
else
{Status. Text ="Cred it card invalid.";}}
```

Kemudian, buat metode *Validate* itu persis di bawah metode *PlaceOrder_Click*.

```
Public bool Validate (string cardNumber.DateTime expDate)
(if(expDate >=DateTime.Today)
(int total =0;
 int temp =0;
 char [] ccDigits ==cardNumber.ToCharArray();
 for(int i=0;i <cardNumber.Length;i++)
(if(((i+l)%2)==0)
(total +=int.Parse (ccDigits [i].ToString());}
else
{temp =int.Parse (ccDigits [i}.ToString())*2;
if(temp >9)
{temp =(int)temp -9;}
total +=temp;}}
if((total%10)==0)
{return true;}
else
(return false;}}
else
```

```
{return false;}}
```

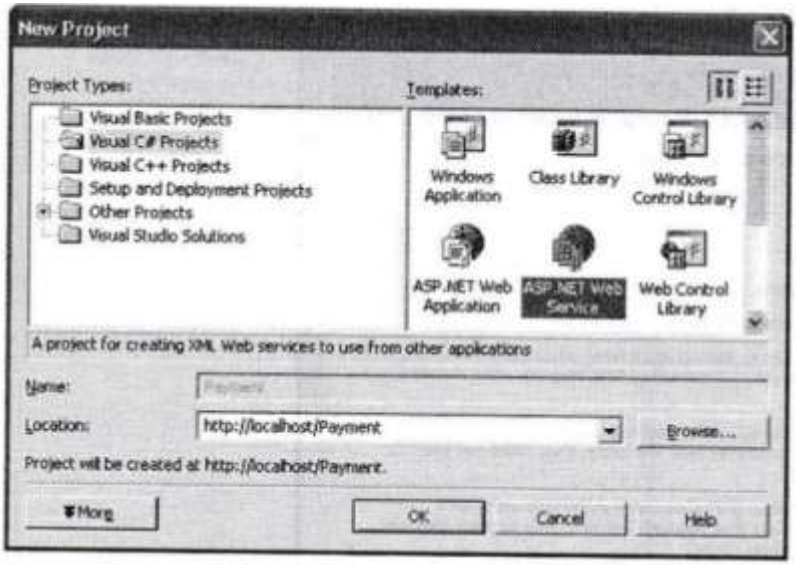
Metode *Validate* menentukan apakah tanggal kadaluwarsanya sudah terlewati. Jika belum, metode *Validate* akan menggunakan *algoritma mod10* stkitar untuk melakukan pengesahan nomor kartu kredit.
Sekarang kompilasikan, kemudian cobalah aplikasi Web tersebut untuk memastikan apakah dapat berjalan seperti yang diharapkan.

Membuat Layanan Web Pembayaran

Sampai sekarang, fungsionalitas pengesahan ditempelkan di dalam *Web Form*. Penulis akan menunjukkan cara menggunakan fungsionalitas pengesahan di dalam aplikasi lain. Misalnya, Kita mungkin ingin membuat aplikasi *WinForm* untuk menjalankan penarikan tunai. Aplikasi ini mungkin harus memanggil logika yang sama dengan pengesahan kartu kredit. Jika Kita memunculkan logika pengesahan sebagai layanan Web, setiap aplikasi yang mengirimkan dan menerima XML lewat HTTP harus dapat memanggil layanan itu untuk melakukan pengesahan informasi kartu kredit.
Untuk membuat layanan Web baru, Kita mempunyai beberapa pilihan. Kita tinggal menambahkan layanan Web ini ke aplikasi yang sudah ada, seperti *Add We Service* dari menu Project atau Kita dapat membuat proyek baru untuk menampung layanan Web tersebut.
Ingat, maksud Kita memakai logika pengesahan kartu kredit adalah sebagai sumber yang dapat dipakai bersama di antara berbagai aplikasi. Logika ini perlu memiliki infrastruktur dan tingkat persyaratan yang berbeda-beda di antara aplikasi komersial. Jadi, dalam hal ini Kita akan

membuat satu aplikasi terpisah, yaitu sebuah aplikasi layanan Web Visual Studio.NET.

Pertama, buka Visual Studio.NET lalu pilih *Create New Project*. Ganti tipe proyeknya dengan pilihan bahasa Kita, kemudian pilih *template Web Service*. Namai proyek ini dengan nama *Payment*, kemudian ganti lokasi URL agar menuju ke *server Web target*, seperti yang ditunjukkan berikut ini:



Gambar 3.3 ASP.NET Web Service

Ganti nama file *Servicel.asmx* menjadi *CreditCard.asmx*. Ada baiknya nama kelas disesuaikan dengan nama *file.asmx*, ganti nama kelas *Servicel* menjadi *CreditCard*. Visual Studio.NET akan membuat file *code-behind* untuk *file.asmx* sama dengan yang dibuatnya untuk

file.aspx. Kita harus membuka *CreditCard*. *asmx.cs* untuk memodifikasi pernyataan kelas. Untuk menampilkan file *code-behind*, klik kanan *CreditCard.asmx* lalu pilih *View Code*. Ganti nama kelas *Seroicel* beserta konstruktornya dengan nama *CreditCard*.

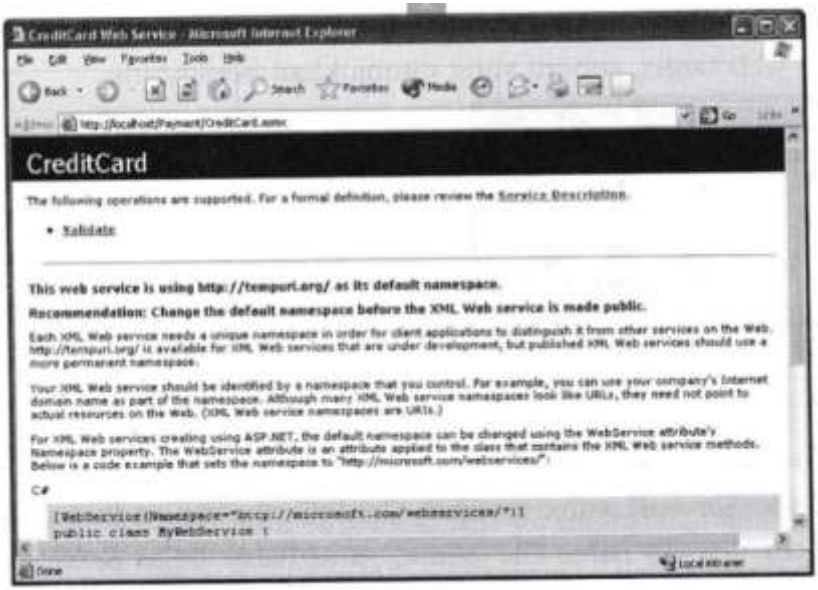
Kemudian, potong metode Validate dari *Web Form Order* dan tempelkan ke dalam kelas *CreditCard*. Kita dapat perindah kelas. tersebut dengan atribut *WebMethod* lalu kompilasikan aplikasi itu. Atribut *WebMethod* adalah satu-satunya kode yang dibutuhkan untuk memunculkan metode *Validate* pada Web!

```
[WebMethod]
public bool Validate (string cardNumber. DateTime expDate)
{
 :
}
```

Atribut *WebMethod* memerintahkan runtime ASP.NET untuk menyediakan semua persyaratan yang dibutuhkan untuk memunculkan metode kelas dalam Web, termasuk kemampuan fungsionalitas untuk mengiklankan fungsionalitas layanan Web di dalam form WSDL (*Web Services Description Language*). Visual Studio.NET secara otomatis akan menggunakan WSDL yang dihasilkan untuk membuat referensi kelayanan Web dalam aplikasi komersial Kita.

Layanan Web lain yang disediakan ASP.NET pada layanan Web payment adalah pembuatan dokumen secara otomatis untuk layanan Web dan sabuk tes. Untuk memanggil fungsionalitas ini, gunakan browser Web untuk membuka file *CreditCard.asmx* seperti Gambar

3.4 ini



Gambar 3.4 Tampilan file *CreditCard.asmx*

Kemudian, cobalah metode *Validate* dengan menggunakan sabuk tes yang disediakan beserta dokumentasinya. Dengan memasukkan nomor kartu kredit yang sah serta tanggal 1/ 1/2052 akan memberikan hasil berikut ini (sekitainya Kita tidak membaca data berusia 50 tahun):

```
<?xml version="1.0" encoding="utf -8" ?>
<boolean xmlns="http://tempuri.org/">true</boolean>
```

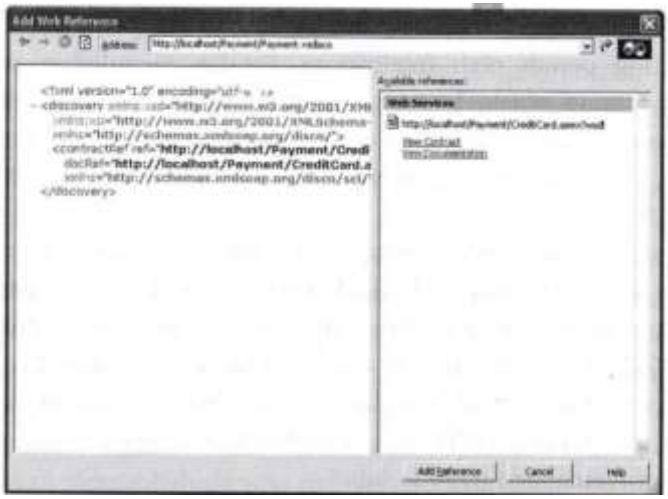
Seperti yang diperkirakan, hasilnya benar. Namun, pesan respon XML bukanlah pesan SOAP yang sah. Terdapat beberapa cara di mana layanan Web berhosting-ASP.NET dapat dipanggil. Layanan Web

akan menghasilkan pesan SOAP dengan format yang bagus jika permintaannya berupa pesan SOAP.

Mengupdate Order Web Form

Tugas selanjutnya adalah kembali ke aplikasi *Commerce* dan mengupgrade *Order Web Form*, sehingga dapat memanggil layanan Web *CreditCard* untuk melakukan pengesahan kartu kredit pengguna. Mula-mula, Kita menambahkan referensi Web untuk layanan Web *CreditCard* ke aplikasi komersial dengan menggunakan Web *Reference Wizard*.

Kita menjalankan *wizard* dengan memilih *Add Web Reference* dari menu Project. Dalam kotak teks Address, masukkan URL untuk *server* yang menyimpan layanan Web target. Klik link untuk layanan *Web Payment* yang ada dalam daftar di dalam panel kanan, seperti Gambar 3.5 ini:



Gambar 3.5 Tampilan Web Payment

Klik *Add Reference*. Akan ditampilkan di sana daftar referensi Web di dalam *Solution Explorer*. Visual Studio.NET akan memproses metadata WSDL yang berkaitan dengan layanan Web Payment dan secara otomatis membuat sebuah *proxy*. Kita dapat menganggap WSDL sebagai persamaan layanan Web bertipe TypeLib. *Proxy* yang dibuat akan memungkinkan layanan Web diperlakukan seperti kelas-kelas .NET lainnya.

Kemudian, gantilah pelaksanaan metode *PlaceOrder_Click* untuk memanggil layanan Web daripada fungsi Validate lokal. Mula-mula, buatlah sebuah fungsi pada kelas *CreditCard* dengan menggunakan kata kunci new sebagaimana yang Kita lakukan pada tipe .NET lainnya. Perhatikan bahwa namespace default untuk layanan Web adalah nama server tempat beradanya layanan tersebut.

```
public void PlaceOrder_Click(object sender.System.EventArgs e)
{
localhost.CreditCard cc = new localhost.CreditCard();
```

Terakhir, panggillah metode *Validate* pada objek cc agar terjadi pertukaran komunikasi antara client (*Order Web Form*) dengan layanan *Web CreditCard*.

```
If(cc.Validate(TextBoxl.Text.Calendarl.SelectedDate) )
(
Status.Text ="Thank you for your order.";
else
```

```
(
Status.Text ="Credit card invalid.";}}
```

Jika Kita menulis kode seperti yang terlihat di bab ini, perlu diingat bahwa objek cc didukung penuh oleh *IntelliSense*. Setelah menuliskan cc, daftar metode yang didukung objek, termasuk metode *Validate*, akan ditampilkan. Setelah Kita menulis kurung pembuka untuk metode tersebut, parameter untuk metode tersebut akan ditampilkan, termasuk tipe parameter layanan Web yang setara di dalam .NET.

Seperti yang sudah Kita ketahui, mengubah aplikasi .NET agar memunculkan logika bisnisnya lewat layanan Web tidak sulit. Kita tinggal menambahkan metode lokal lalu meletakkannya ke dalam file *code-behind .asmx* untuk layanan Web tersebut. Kode yang dibutuhkan untuk memunculkan metode itu lewat HTTP adalah atribut WebMethod. Setelah kode itu dikompilasi dan digunakan, tiap client yang mendukung HTTP dan memiliki kemampuan untuk memilah-milah string dapat memanggil metode ini.

Menggunakan File Web Bersama-sama

Contoh berikut ini akan membuat layanan Web yang memungkinkan file biner ditransfer lewat HTTP dan SOAP. Karena SOAP memakai metode pengodean berbasis-XML, Kita harus mewakili parameter metode dan menghasilkan nilai dengan cara yang kondusif bagi XML. Dalam contoh sebelumnya, digunakan cara yang lebih cepat untuk melakukan pengodean nomor kartu kredit dan tanggal kadaluwarsanya. Dalam contoh ini, layanan Web akan membuat kode bertipe kompleks

yang antara lain berisi data biner. Seperti yang akan Kita lihat, platform .NET akan menangani pengodean dan pendecodean secara otomatis. Tindakan ini akan memastikan bahwa array byte dibuat kodenya supaya array tersebut tidak memasukkan karakter khusus ke dalam pesan SOAP yang dapat menggagalkan pengesahan XML.

Sebelum masuk lebih jauh, Penulis ingin memberitahukan sesuatu: Ada beberapa cara yang lebih efisien untuk melakukan pengiriman isi file lewat Web daripada membuat kodenya ke dalam pesan SOAP. Contoh ini tidak ditujukan untuk menggantikan FTP, tapi hanya menunjukkan kemampuan dan keluwesan layanan Web serta kehkitalan platform .NET.

Membuat Layanan Web-Web File Share

Layanan Web *WebFileShare* akan memunculkan dua metode, *ListFiles* dan *Get File*. *ListFiles* digunakan untuk memperoleh daftar file yang ada dari layanan Web dan *GetFile* dipakai agar *client* dapat mengambil file tertentu.

Pertama-tama, Kita membuat proyek layanan Web Visual Studio.NET dengan memakai langkah-langkah yang sama dengan contoh sebelumnya. Buka Visual Studio.NET, kemudian buat buatlah sebuah proyek baru. Ganti tipe proyeknya dengan pilihan bahasa Kita, kemudian pilih template Web *Service*. Namai proyek Kita dengan *WebFileShare*, kemudian ganti lokasi URL yang menuju ke server Web target.

Ganti nama file *Servicel.asmx* menjadi *WebDirectory.asmx*. Klik

kanan Web *Directory.asmx*, kemudian pilihan *View Code*. Ganti nama kelas Seroicel beserta konstruktornya dengan nama *WebDirectory*. Kemudian, lakukan modifikasi pada properti Inherits dalam header ASMX untuk menunjuk ke nama kelas yang baru tersebut.

Sekarang, Kita sudah membuat proyek Visual Studio.NET. Selanjutnya, Kita akan menambah dengan persyaratan agar client dapat meminta dan mengambil file. Pertama, Kita hams membuat sebuah alias untuk *namespace System*. IO dengan kata kunci using. Di bawah pernyataan using yang membuat alias untuk *namespace System*. *Web*.*Services*, ketiklah pernyataan berikut ini:

```
using System.IO;
```

Kemudian, definisikan struktur *WebFile* yang membungkus isi file bersama metadatanya. Struktur file berisi nama file, isi file, dan atribut sistem file yang berkaitan dengan file tersebut.

Definisikan struktur di bawah ini persis setelah *namespace WebFileShare* didefinisikan:

```
public struct WebFile
{
public string Name;
public byte [] Contents;
public DateTime CreationTime;
public DateTime LastAccessTime;
public DateTime LastWriteTime;
}
```

Kemudian, Kita harus membuat sejumlah metode di dalam kelas

WebFileShare. Metode-metode ini akan didefinisikan persis sebelum metode contoh *Hello World*.

Metode pertama yang hams didefinisikan akan menghasilkan daftar file yang bisa didownload dalam direktori c:\Public. Daftar file ini dihasilkan sebagai array string. Secara otomatis, NET Framework akan melakukan serialisasi (mengubah transmisi paralel ke transmisi serial) array yang sesuai.

```
[WebMethod]
public string [] ListFiles (()
{
return Directory.GetFiles("c:\\Public ");
```

Kemudian, buatlah sebuah metode *GetFile* agar client dapat meminta file. Jika file tersebut ada, sebuah persyaratan dari stmktur *WebFile* akan dikirim balik ke client. Dalam metode ini, Kita menggunakan objek *System*.IO. File untuk mendapatkan informasi yang dibutuhkan mengenai file tersebut. Kita mendapatkan objek Stream dari objek File lalu menuliskan isi dari aliran itu ke dalam array byte di dalam stmktur *WebFile*. Kita juga menata field-field *CreationTime*, *LastAccessTime*, dan *LastWriteTime* dengan nilai-nilai yang didapatkan dari metode statis terkait yang dimunculkan oleh kelas File.

```
[WebMethod]
public WebFile GetFile(string fileName)
{
WebFile webFile =new WebFile();
String filePath ="c:\\Public \\" ++fileName;
//Set the name of the file.
```

```
webFile.Name =fileName;
//Obtain the contents of the requested file.
Streams =File.Open(filePath.FileMode.Open) ;
webFile.Contents =new byte [s.Length];
s.Read(webFile.Contents,0,(int)s.Length);
s.Close
//Retrieve the date/time stamps for the file.
webFile.CreationTime =File.GetCreationTime(filePath);
webFile.LastAccess Time =File.GetlastAccess Time(filePath);
webFile.LastWriteTime =File.GetlastWriteTime(filePath);
return webFile;
}
```

Setelah diinisialisasi, struktur *WebFile* akan dikirim balik ke client. Sekali lagi, .NET Framework melakukan serialisasi yang sesuai. Dalam hal ini, dilakukan serialisasi persyaratan pada struktur, termasuk datanya. *Array byte Contents* dikodek an ke dalam representasi Base-64. Kode Base-64 memetakan tiap byte pada *array* ke dalam karakter alfanumerik yang dapat diletakkan di dalam dokumen XML dengan disertai karakter yang sah. Selain itu, ke dalam pesan SOAP dilakukan serialisasi nilai pada field-field Name, *CreationTime*, *LastAccessTime*, dan *LastWriteTime*.

Nanti di dalam bab ini, akan membahas dengan rinci bagaimana.NET Framework melakukan pengodean tipe data seperti struktur, *array byte*, dan enumerasi. Kita perlu mengetahui bahwa pesan yang dihasilkan seratus persen sesuai dengan spesifikasi SOAP stkitar industri. Oleh karena itu, client yang memakai platform, sistem operasi, atau hardware yang berbeda dapat berinteraksi dengan layanan

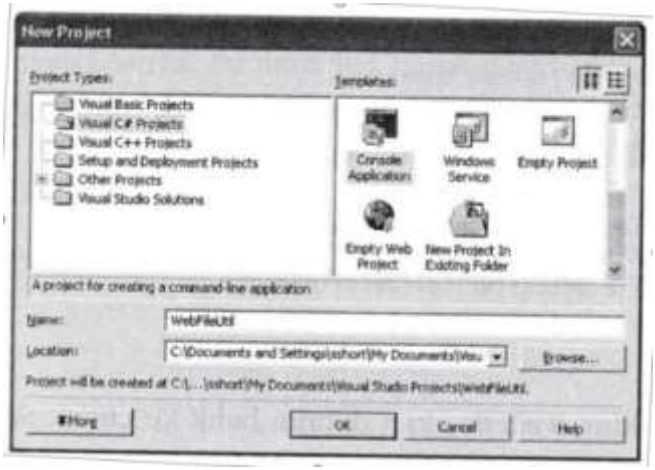
Web *WebDirectory*.

Sekarang, Kita sudah membangun layanan Web. Selanjutnya, kita akan membuat satu client untuk mengambil file lewat layanan Web *WebFileSystem*.

Membuat Program WebFileUtil

Selanjutnya, Kita akan membuat sebuah aplikasi konsol untuk mengambil file dari layanan Web *WebFileShare*. Aplikasi konsol ini akan menerima paling sedikit dua argumen baris perintah, yaitu satu perintah yang menyebutkan tindakan yang harus dilakukan dan nama file target. Jika berminat, Kita juga dapat menyebutkan argumen ketiga untuk menyebutkan nama file yang dituju. Jika tidak, nilai defaultnya adalah nama asli file tersebut.

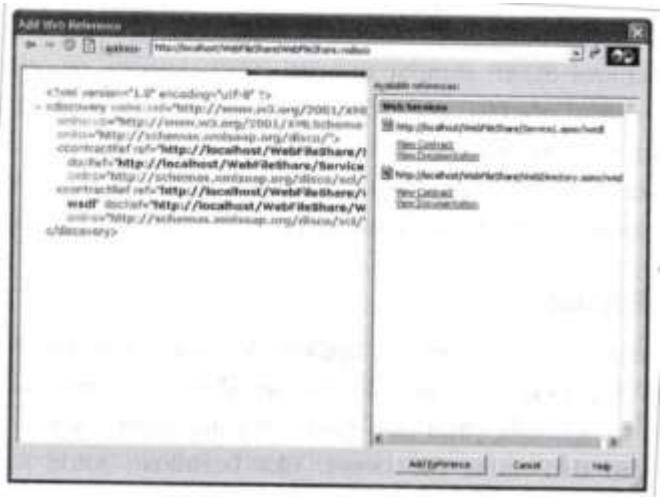
Mulailah dengan membuka Visual Studio.NET dan pilih *Create New Project*. Ganti tipe proyek Kita dengan pilihan bahasa Kita, kemudian pilih template *Console* Application, dan namai proyek itu *WebFileUtil*, seperti Gambar 3.6 berikut ini:



Gambar 3.6 Template Console Application

Kita dapat mengganti nama file Service! (dengan ekstensi yang bergantung pada pilihan bahasa Kita) dengan *WebFileUtil* lalu hapus pelaksanaan default dalam *WebFileUtil* yang disediakan oleh Visual Studio.NET.

Tambahkan referensi Web kelayanan *Web WebDirectory*. Jalankan *Web Reference Wizard* dengan memilih *Add Web Reference* dari menu Project. Dalam kotak teks Address, masukkan URL untuk server yang menyimpan layanan *Web* target. Klik link ke direktori *WebFileShare*. Layanan Web *WebFileShare* secara otomatis akan dipilih, seperti Gambar 3.7 berikut ini:



Gambar 3.7 Layanan Web *WebFileShare*

Klik tombol *Add Reference*. Ganti nama referensi baru *Web* dalam daftar Solution Explorer ke *WebFileShare*.

Sekarang, Kita sudah membuat dan mengkonfigurasi referensi Web. Mari kita lanjutkan ke pelaksanaan *WebFileUtil* pada C#.

Pertama-tama, Kita perlu mengimpor namespace System.IO ke dalam proyek. Ini akan mempermudah kode saat Kita menggunakan kelas File di dalam kode di bawah. Kita jug a harus mengimpor namespace *WebFileShare* pada referensi Web Kita.

```
using System;
using System.IO;
using WebFileUtil.WebFileShare;
namespace WebFileUtil
(
```

Kemudian, ganti nama kelas dari *Classl* ke nama yang lebih jelas, yaitu

Web FileUtil.

```
Public class WebFileUtil
```

Pastikan bahwa program itu meneruskan jumlah argumen baris perintah yang benar. Selanjutnya, lakukan inisialisasi pada variabel sumber dan variabel tujuan.

```
//Validate number of command-line arguments.
if(args.Length <1 ||args.Length >3)
(
DisplayUsage();
return 1;
)
//Initialize variables.
string source =args [1];
string destination =source;
if(args.Length ==3)
destination =args [2];
```

Lalu, proseslah perintah yang diteruskan tersebut. Fungsi pembantu yang sesuai akan dipanggil untuk perintah DIR atau GET

```
//Process command.
switch(args [0].ToUpper())
(
case "DIR":
ListFiles();
break;
case "GET":
GetFile(source,destinationl;
break;
default:
```

```
DisplayUsage ();
break;
)
return 0;
}
```

Kemudian, buatlah metode *ListFiles* untuk menampilkan pada konsol daftar file yang ada di layanan Web. Caranya, buatlah satu objek *WebDirectory* dan satu referensi ke array string. Lalu, atur referensi ke array string tersebut agar sama dengan nilai yang dihasilkan dari metode *WebDirectory.ListFiles*. Akhirnya, ulangi kembali untuk semua array dan tuliskan nama setiap file yang dihasilkan ke dalam konsol.

```
private static void ListFiles ()
{
WebDirectory webDir =new WebDirectory();
string [] Files;;
files =webDir.ListFiles ();
foreach(string file in files)
{
Console.Writeline(file);
Console.Writeline("\n{0}file(s)in directory.",files.Length);
```

Untuk membuat metode GetFile yang akan mengambil file yang diminta dan menyimpannya ke sistem file. Kita perlu terlebih dulu membuat persyaratan baru pada kelas *WebDirectory* dan referensi ke *WebFile*. Kemudian, panggil metode *GetFile* pada objek *WebDirectory* untuk mengambil objek *WebFile*. Buka file yang dituju

dan gunakan kelas stream untuk menuliskan *array byte* ke dalam file. Akhirnya, atur tkita waktu untuk file tersebut dengan nilai yang ada di dalam field-field *CreationTime*, *LastAcc essTime*, dan *LastWriteTime*.

```
private static void GetFile(string source.string destination)
{
WebDirectory webDir =new WebDirectory();
WebFile webFile =new WebFile();

//Retrieve the requested file and then save it to disk.
webFile =webDir.GetFile(source);

//Save the retrieved Webfile to the file system.
FileStream fs =File.Open Write(destination);
fs.Write(webFile.Contents.0.webFile.Contents.Lengthl;
fs.Close ():
//Set the date/time stamps for the file.
File.SetCreationTime(destination,webFile.CreationTime);
File.SetlastAccessTime(destination,webFile.LastAccessTime);
File.SetlastWriteTime(destination,webFile.LastWriteTime);
}
```

Akhirnya, buatlah sebuah metode *DisplayUsage* untuk menuliskan sintaks yang sesuai untuk *WebFileUtil* ke dalam konsol.

```
private static void DisplayUsage()
(
Console.Writeline("WebFile is used to retrieve files from the
WebDirectory Webservice.");
Console.WriteLine("\nUsage:WebFile command source
[destination]");
Console.Writeline("\tcommand   - Either DIR or GET.");
```

```
Console.Writeline ("\tsource   - The name of the file to
retrieve.");
Console.Writeline("\tdestination      - Optionally the name of
the
destination file.");
Console.Writeline("\nExamples:");
Console.Writeline("\tWebFile GET somefile.exe");
Console.Writeline("\tWebFile GET somefile.exe c:\temp");
Console.Writeline("\tWebFile GET somefile.exe c:\temp
\myfile.exe");
}
}
```

Perhatikan bahwa keseluruhan contoh tersebut menggunakan tipe .NET. Layanan Web menyatakan kelas-kelas yang memunculkan metode dengan parameter bertipe kuat dan menghasilkan nilai. Misalnya, layanan Web *WebFileShare* menyatakan sebuah struktur *WebFile*. Kode client menggunakan struktur *WebFile* seolah tipe asli .NET, meskipun pesan yang diteruskan antara client *WebFileUtil* dan layanan Web *WebDirectory* ada di dalam XML.

Hasilnya, banyak fitur Visual Studio.NET tersedia bagi pengembang, seperti *IntelliSense* dan kemampuan untuk menangkap kesalahan penyesuaian tipe sewaktu kompilasi. Misalnya, struktur *WebFile* tersedia bagi *client* tanpa kehilangan ketepatannya. Jika *client* berusaha mengubah field *LastAccessTime* menjadi string, sebuah kesalahan akan dihasilkan pada waktu kompilasi. Hal ini dicapai karena struktur *WebFile* dimunculkan oleh file WSDL yang secara otomatis dihasilkan oleh runtime ASP.NET. Visual Studio.NET menggunakan informasi

di dalam file WSDL untuk membuat *proxy* bertipe kuat untuk layanan Web.

Seperti yang telah diduga, array byte harus dikodekan dalam cara tertentu sebelum dapat disisipkan ke dokumen XML. Namun, infrastruktur .NET telah menangani semua pengodean itu untuk Kita. *Array byte* diubah ke dalam string berkode-Base-6 4, diletakkan ke dalam pesan SOAP, dikirim lewat kabel, kemudian dikodekan kembali ke dalam array byte.

C. PENUTUP

Ringkasan

Bab ini memperlihatkan kemudahan membuat dan memakai layanan Web dengan Visual Studio.NET. Kita membuat dua layanan Web, satu untuk melakukan validasi kartu kredit dan satu lagi untuk mengirim dan menerima file.

Contoh pertama menunjukkan kemudahan memasukkan layanan Web ke dalam aplikasi .NET yang ada. Kita juga dapat dengan mudah memindahkan logika bisnis untuk aplikasi Web dan meletakkan ke dalam layanan Webnya sendiri. Contoh kedua di mana Kita membuat layanan Web untuk menerima file menunjukkan keluwesan layanan Web dan kehkitalan .NET Framework.

Hampir semua kode yang Kita tulis dalam bab ini berkaitan dengan logika bisnis daripada pembuatan infrastruktur. Satu-satunya kode yang Kita tulis untuk memunculkan metode seperti layanan Web adalah untuk menghias metode dengan atribut *WebMethod*.

Framework ASP.NET menangani pengodean pesan permintaan dari client dan membuat kode pesan respons.

Microsoft.NET juga menyediakan banyak layanan untuk client. Visual Studio.NET secara otomatis menghasilkan proxy dari dokumen WSDL layanan Web. Kita dapat menganggap dokumen WSDL sebagai persamaan layanan Web bertipe TypeLib pada COM. WSDL juga berisi informasi tentang properti, metode, dan enumerasi yang diekspos oleh layanan Web.

Proxy yang dihasilkan Visual Studio.NET memungkinkan Kita membuat kode pada layanan Web seolah-olah layanan Web itu adalah objek .NET bertipe kuat lainnya. Salah satu keuntungan utama tipe kuat adalah dapat menangkap kesalahan pasangan tipe saat kompilasi daripada saat runtime. Keuntungan lainnya adalah lntelliSense. Saat membuat kode untuk objek *proxy* yang dihasilkan Visual Studio.NET, IntelliSense menampilkan daftar metode yang dimunculkan oleh layanan Web. Setelah metode dipilih, IntelliSense menampilkan nama dan tipe setiap parameter di dalam metode itu.

Dalam bab-bab selanjutnya, Kita akan belajar mengenai protokol yang terdapat di dalam layanan Web, seperti WSDL dan SOAP, dan tentang teknologi Web seperti ASP.NET yang dapat Kita gunakan untuk membangun dan menggunakan layanan Web.

Pertanyaan

1. Apa itu Web Form?
2. Buat contoh aplikasi layanan web pembayaran?

3. Sebutkan Langkah Order Web Form?
4. Bagaimana cara Menggunakan File Web Bersama?
5. Bagaimana cara Membuat Layanan Web File Share?

E. DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.

11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 4

Membuat dan Menggunakan layanan Web

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 4 akan di bahas Membuat Layanan Web Dasar diantaranya: Aplikasi Komersial Sederhana, membuat Web Form, Membuat Layanan Web Pembayaran, Menupdate Order Web Form, Menggunakan File Web Bersama, Membuat Layanan Web File Share, Membuat Program WebFileUtil.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan dan memnerapkan Aplikasi Komersial Sederhana, membuat Web Form, Membuat Layanan Web Pembayaran, Menupdate Order Web Form, Menggunakan File Web Bersama, Membuat Layanan Web File Share, Membuat Program WebFileUtil.

B. PENYAJIAN

Membuat dan Menggunakan layanan Web

Bab-bab terdahulu telah memfoku kan pada pembuatan Windows ASP.NET, juga teknologi-teknologi yang memungkinkan pengembangan ASP.NET dan yang menjadikannya sangat sederhana dan cepat untuk pembuatan aplikasi- aplikasi Web yang kaya fitur dan hkital. Bagian penting dan sangat berguna lainnya dari ASP.NET yang

belum dibahas adalah layanan Web berbasis-XML.

Sekarang ini, layanan Web belumlah sebuah teknologi yang dimengerti dengan baik. Bagian ini adalah disebabkan oleh ketidak-akuratan laporan dari media, yang menjelaskan layanan web sebagai segala yang berasal dari perangkat-lunak penyumbang yang dimiliki Microsof t untuk mengambil alih Internet. Cerita-cerita ini mungkin terdengar hebat di atas kertas, namun mereka berdasarkan pada kesalahapahaman dan ketidak-pedulian terhadap teknologi yang revolusioner ini.

Memahami Layanan Web Berbasis-XML

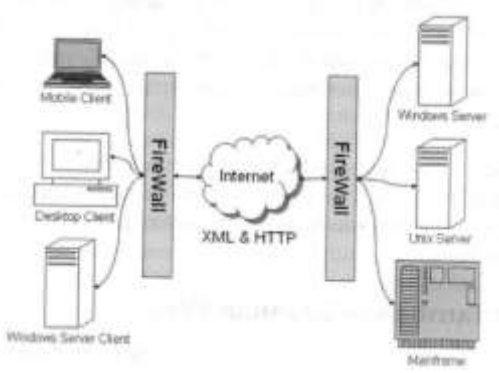
Selama beberapa tahun, para pengembang telah mencari cara untuk memanfaatkan ulang produk-produk kerja mereka, atau menghadirkan bersama sistem-sistem terpisah yang telah dimiliki oleh organisasi mereka ke dalam bentuk yang kohesif. Pencarian ini telah menghadapi berbagai kesulitan sejak dari platform yang berbeda hingga sistem yang tidak memiliki stkitar untuk komunikasi, sampai ukuran keamanan yang menghalangi client potensial dari luar jaringan perusahaan untuk bisa menggunakan fungsionalitas yang ada.

Sebagian dari solusi. atas kesulitan ini telah memasukkan pembayaran front end Web custom untuk fungsionalitas yang ada, atau menggunakan komunikasi superior dan/atau teknologi pemaketan untuk menjembatani kesenjangan antar sistem. Solusi terdahulu bekerja dengan baik dalam sejumlah situasi, namun organisasi yang

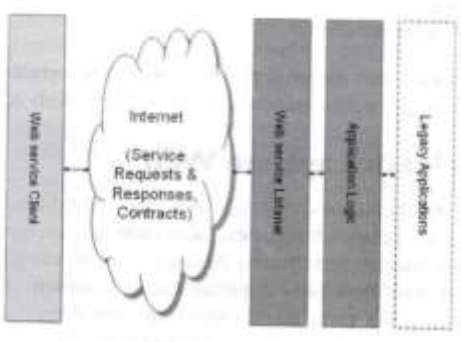
menampung aplikasi harus membuat dan menjaga sebuah UI untuk fungsionalitas yang telah mereka sediakan, termasuk membuat sebuah front-end untuk semua fungsionalitas tambahan berikutnya. Lebih jauh, UI ini mungkin atau mungkin tidak benar-benar memenuhi kebutuhan client mereka yang ada atau yang akan datang. Penggunaan komunikasi superior atau perangkat-lunak perantara berarti biasanya pengeluaran yang lebih besar, ketergantungan pada vendor, dan tergantung pada perawatan dan upgrade vendor. Layanan Web berbasis-XML secara langsung mengatasi masalah-masalah ini, yang memungkinkan fungsionalitas programatis diekspos tanpa membutuh kan UI berbasis-Web, dan tidak memerlukan perangkat-lunak perantara.

Layanan Web merupakan satuan diskrit dari fungsionalitas programatis yang diekspos kepada client via protokol komunikasi, dan format data stkitar bernama HTTP dan XML. Protokol-protokol ini mengatasi masalah komunikasi lintas Internet dan lintas firewall perusahaan tanpa beralih: ke solusi superior yang memerlukan port-port komunikasi tambahan yang harus dibuka untuk akses eksternal. Oleh karena layanan Web menggunakan XML memformat permintaan dan jawaban, ia bisa diekspos dari dan digunakan oleh segala platform yang bisa memformat dan menguraikan sebuah pesan XML. Ini memungkinkan layanan Web berbasis-XML menghadirkan bersama potongan-potongan fungsionalitas berbeda, baik yang ada maupun yang baru. internal maupun eksternal pada sebuah organisasi, dalam satu

kekompakan yang koheren. Gambar-gambar berikut memperlihatkan bagaimana layanan Web berbasis-XML bisa menghadirkan bersama beragam aplikasi dan platform. Perhatikan bahwa karena layanan Web XML bisa berkomunikasi lewat protokol tkitar seperti misalnya HTTP, mereka bisa bekerja sama lewat firewall. Selain interaksi yang diperlihatkan, panggilan layanan Web server-to- server juga dimungkinkan.



Layanan Web diakses dengan memanggil ke pendengar yang bisa memberikan sebuah kontrak untuk menjelaskan layanan Web tersebut, dan yang juga memproses permintaan yang masuk atau meneruskan mereka ke logik aaplikasi lain dan menyalurkan kembali segala respons ke client. Gambar berikut memperlihatkan bagaimana sebuah aplikasi layanan Web bisa berlapis. Ia juga bisa menggun akan layanan Web untuk mengakses logika warisan serta logika aplikasi baru, atau untuk membungkus logika warisan dengan sebuah layanan Web .



SOAP dan Layanan Web

Protokol SOAP adalah sebuah kunci pengaktif layanan Web berbasis-XML sebagai sebuah strategi untuk integrasi aplikasi dan layanan-layanan perangkat lunak berbasis-Internet. SOAP, yang merupakan singkatan dari *Simple Object Access Protocol*, adalah sebuah protokol pesan berbasis-XML yang distkitarisasi oleh *World Wide Web* (W3C), dan didukung luas dari berbagai vendor. Spesifikasi SOAP menyediakan sebuah format wajib untuk pesan-pesan SOAP (disebut sebagai amplop SOAP), serta stkitar opsional untuk pengkodean data, memproses permintaan/tanggapan, dan pengikatan pesan- pesan SOAP ke HTIP.

SOAP dan layanan Web berbasis-XML terlalu rumit untuk dibahas dalam bab tunggal ini. Untunglah, salah satu manfaat dari modul pemrograman ASP. NET adalah bahwa ia mengabstraksikan kompleksitas pesan SOAP dari pengembangan layanan Web .

Kebutuhan Tambahan Layanan Web

Seperti disebutkan sebelumnya dalam bab ini, layanan Web tidak lebih dari sebuah cara untuk menyediakan unit-unit diskrit dari fungsionalitas aplikasi lewat protokol Web stkitar, menggunakan format pesan yang telah distkitarisasi. Agar betul-b etul berguna, bagaiman apun juga, layanan Web juga perlu menyediakan yang berikut ini:

Sebuah kontrak yang menyebutkan parameter dan tipe-data yang diharapkannya, serta tipe pengembalian (jika ada) yang dikirimkannya ke pemanggil, dan Sarana untuk menemukan layanan Web, atau cara menemukan layanan Web yang ditawarkan oleh server atau aplikasi pemberi, dan keterangan dari layanan-layanan tersebut.

Beberapa seksi berikut akan menguji pembuatan, iklan, pencarian, dan penggunaan layanan Web dari sudut-pkitang pengembang Web ASP.NET.

Membuat sebuah Layanan Web

Dalam ASP.NET, kompleksitas pesan SOAP dan penghantaran HTIP diabstraksikan dari para pengembang, sehingga pembuatan layanan Web berbasis-XML sungguh sangat sederhana. Abstraksi ini dimungkinkan melalui atribut-atribut yang memberi. tahu runtime bahasa umum (runtime) untuk menganggap sebuah kelas yang disebutkan dan metodenya sebagai layanan Web. Berdasarkan pada atribut-atribut ini, runtime tersebut menyedi akan semua keperluan

untuk mengekspos kelas dan metode tersebut sebagai layanan Web, serta menyediakan sebuah kontrak yang bisa digunakan client untuk menentukan bagaimana memanggil layanan Web dan tipe-tipe kembalian yang diharapkan darinya.

Mendeklarasikan Layanan Web

Layanan Web dalam ASP. NET didefinisikan dalam file-file berekstensi .asmx. Kode sebenarnya untuk layanan Web ASP.NET bisa ditempatkan dalam file .asmx atau dalam sebuah kelas prakompilasi. Bisa juga, layanan Web dideklarasikan dalam file .asmx dengan menambahkan direktif @ WebService ke file tersebut. Sintaks untuk mendeklarasikan sebuah layanan Web ditempat kelas tersebut akan didefinisikan dalam file-file .asmx adalah;

```
<%@WebService
Language="VB"Class="ClassName"%>
```

ClassName adalah nama kelas yang berisi metode-metode layanan Web. Sintaks untuk mendeklarasikan sebuah layanan Web di tempat kode yang mendefinisikan kelas tersebut berada dalam kelas yang telah dikompilasi adalah;

```
<%@WebService
Class="NamespaceNameClassName,AssemblyNa
me"%>
```

NamespaceName adalah nama namespace tempat kelas untuk layanan Web berada, ClassName adalah nama kelas yang mengandung metode-

metode layanan Web, dan AssemblyName adalah nama assembly yang mengandung kode terkompilasi untuk layanan Web. Assembly ini harus berada dalam direktori bin pada aplikasi Web di tempat file .asmx berada. Bagian Namespace Name dari atribut Class bersifat opsional jika Kita belum menyebutkan sebuah namespace bagi kelas Kita, namun ada baiknya selalu menyebutkan sebuah namespace bagi kelas-kelas Kita. Bagian AssemblyName pada atribut Class juga opsional, karena ASP.NET akan mencari direktori bin untuk assembly tersebut jika tidak ada nama assembly yang diberikan. Bagaimanapun juga, Kita harus selalu menyediakan sebuah nama assembly untuk menghindari dampak kinerja ketika melakukan pencarian saat pertama layanan Web diakses.

Catatan

Tidak seperti pengguna code-behind pada Web Form ASP.NET, Kita tidak bisa menggunakan atribut src pada sebtian lanan Web. ini berarti Kita harus melakukan prakompilasi kelas code-behind yang akan dipakai sebagai basis untuk layanan Web, atau bisa juga secara manual menggunakan compiler baris perintah atau dengan membangun sebuah aplikasi Visual Studio.NetWeb Service.

Membuat Kelas Layanan Web

Setiap file .asmx diasosiasikan dengan sebuah kelas tunggal, baik

sebaris dengan file .asmx atau dalam kelas code-behind. Bisa juga, sintaks sebuah kelas layanan Web seperti yang berikut ini:

```
Imports System.Web.Services Public Class
ClassName
Inherits WebService
'Methodsexposed by the Webservice End
Class
```

ClassName adalah nama kelas layanan Web.

Mewarisi dari WebService

Dalam contoh sintaks ini, ada sebuah pernyataan Imports yang mengimpor namespace System.Web. Services. Namespace ini berisi kelas-kelas dan atribut- atribut yang menyedi akan dukungan ASP.NET built-in untuk layanan Web berbasis-XML. Dalam hal ini, kelas yang kami maksud adalah WebService, yang mewarisi dalam contoh ini menggu nakan kata-kunci Inherits.

Pewarisan dari kelas WebServices tidak dibutuhk an agar sebuah kelas bisa berfungsi sebagai sebuah layanan Web, melainkan kelas WebServices menyediakan plumbing berguna untuk layanan Web. Ini mencakup akses ke Application, Session, dan obyek-obyek ASP.NET lain, termasuk obyek Context yang memberikan akses ke informasi mengenai permintaan HTTP yang ditimbulkan oleh layanan Web.

Menggunakan Atribut <WebService ()> untuk Menyebutkan Namespace XML

Layanan Web berbasis-XML menggunakan namespace XML untuk menkitai secara unik endpoint atau pendengar ke mana sebuah client mengirimkan permintaan. Secara default, ASP.NET menyeting namespace XML untuk sebuah layanan Web ke <http://tempuri.org/>. Apabila Kita berniat membuat layanan Web sendiri yang tersedia secara umum lewat Internet, gantilah namespace default untuk menghindari konflik potensial dengan layanan-layanan Web berbasis-XML lainnya yang menggunakan nama sama dengan layanan Web Kita, dan yang mungkin menggunakan namespace default tersebut.

Kita bisa memodifikasi namespace default untuk sebuah kelas layanan Web dengan menambahkan atribut metadata WebService ke definisi kelas, dan menyeting properti Namespacenya ke nilai yang diinginkan bagi namespace baru tersebut. Nilai ini bisa berupa URI unik yang Kita kontrol, seperti misalnya nama domain organisasi Kita.

Dalam Visual Basic.NET, atribut-atribut metadata menggunakan sintaks <AttributeName (PropenyName: =value)> dan ditempatkan pada baris yang sama dengan entitas tersebut (kelas, metode, dan lain-lain) yang mereka modifikasi. Demi keterbacaan, Kita bisa menggunakan karakter penerusbaris Visual basic untuk melipat definisi kelas ke baris berikut :

```
<WebServiceCNamespace:="http://www.aspne
tsbs.com/webservices/'')>_PublicClassCla
ssName
```

Dalam C#, atribut metadata menggunakan sintak? [AttributeName (Property-Name=value)} dan ditempatkan pada baris sebelum entitas yang mereka modifik asi.

```
[WebService(Namespace="http://www.aspnet
sbs.com/webservices/'')J public class
ClassName:WebService{
```

Dalam Visual Basic .NET dan C#, multi-properti bisa dibuat dalam sebuah deklarasi atribut dan dipisahkan oleh koma.

Atribut-Atribut Metadata

Sebelum bab ini, istilah atribut-atribut mengacu pada pasangan kunci/ nilai yang ditambahkan ke tag-tag kontrol server ASP.NET atau HTML untuk mendefinisikan properti-properti spesifik dari entitas atau kontrol tersebut. Bab ini memperkenalkan konsep atribut-atribut Metadata.

ASP.NET dan .NET Framework memperluas penggunaan atribut untuk memberikan plumbing programatis bagi kelas-kelas, juga untuk menambahkan informasi deskriptif ke assembly yang telah tersusun. Kita bisa juga menggunakan atribut-atribut Kita sendiri (dengan membuat kelas-kelas yang berasal dari System.Attribute) untuk menambahkan informasi deskriptif sendiri ke kelas-kelas Kita.

Atribut-atribut dalam layanan Web ASP.NET, selain untuk menyediakan akses ke namespace XML yang dipakai untuk layanan

Web, juga dipakai untuk memberitahu ASP. NET bahwa sebuah metode yang diberikan akan diekspos sebagai sebuah WebMethod agar client layanan Web bisa mengakses WebMethod tanpa mengharuskan programmer menuliskan kode tambahan.

Menggunakan Atribut <WebMethod()> untuk Mengekspos Metode-Metode

Setelah Kita mendeklar asikan kelas yang menerapkan layanan Web, Kita perlu menambahkan satu atau beberapa metode yang menyediakan fungsionalitas bagi layanan Web. Setiap metode akan menggunakan atribut Web Method Metadata untuk mendeklarasikan metode-metode sebagai metode ke mana ASP.NET harus memberikan plumbing yang diperlukan untuk mengeksposnya ke client-client .

Untuk menambahkan atribut WebMethod ke sebuah metode Visual Basic.NET, gunakan sintaks berikut:

```
<WebMethod()>
PublicFunctionMethodName()As ReturnType
methodcode End Function
```

MethodName adalah nama metode yang akan diekspos oleh layanan Web, dan ReturnType adalah tipe nilai (jika ada) yang akan dikembalikan oleh metode tersebut. Walaupun metode Kita tidak mengembalikan sebuah nilai yang merupakan hasil perhitungan atau aksi lainnya, biasanya ada baiknya paling tidak mengembalikan sebuah nilai yang menkitai berhasil tidaknya metode tersebut.

Properti-Properti Atribut <WebMethod ()>

Seperti atribut WebService, atribut WebMethod memiliki properti-properti yang bisa Kita gunakan untuk mengubah perilaku metode Web. Properti- properti ini dijelaskan dalam tabel berikut.

Properti properti WebMethod

Properti	Keterangan
BufferResponse	Menentukan apakah output dari metode akan ditahan dalam memori sebelum dikirimkan ke client. Defaultnya adalah True.
CacheDuration	Memungkinkan output metode layanan Web disimpan sementara pada server untuk selang tertentu, dalam detik. Defaultnya adalah 0, yang berarti mematikan caching untuk metode tersebut.
Description	Menyeting keterangan teks dari metode layanan Web. Keterangan ini diperlihatkan dalam halaman-halaman doku mentasi yang dibuat otomatis oleh ASP.NET, dan ini dimasukkan dalam kontrak *Web Service Description Languange* (WSDL) bagi layanan Web.
EnableSession	Menentukan apakah dukungan sesi diaktifkan bagi metode layanan Web. Defaultnya adalah False. Mengaktifkan status sesi mengharuskan kelas layanan Web untuk mewarisi dari

	WebService. Perhatikan mengaktifkan status sesi untuk sebuah layanan Web bisa berdampak negatif terhadap kinerja.
MessageName	Memungkinkan metode-metode diberikan nama aliasnya. Ini bisa berguna saat Kita memiliki metode yang overload, di mana nama metode yang sama memiliki banyak implementasi yang menerima parameter-parameter input berbeda. Dalam hal ini, Kita bisa memberikan sebuah Messag eName berbeda untuk masing-masing versi metode yang overload, sehingga Kita bisa mengenalinya masing-masing. Defaultnya adalah nama metode.
Transaction Option	Memungkinkan penambahan dukungan bagi transaksi COM+ dalam metode layanan Web. Nilai-nilai yang diperkenankan adalah Disabled, NotSupported, Supported, Required, dan RequiresNew. Defaultnya adalah Disabled. Perhatikan bahwa properti ini mengharuskan penggunaan sebuah direktif @Assembly untuk memuat assembly System .Enterprise Services ke dalam aplikasi layanan Web.

Membuat sebuah Layanan Web HelloWorld

Sekarang mari kita gunakan informasi yang telah Kita pelajari sejauh ini untuk menggunakan sebuah layanan Web. Layanan Web ini akan memiliki sebuah metode, yang berguna untuk mengembalikan sebuah nilai string Hello, World!

1 Dalam sebuah folder yang didefinisikan sebagai sebuah aplikasi dalam IIS, buatlah sebuah file baru bernama HelloSer vice.asmx.

2 Tambahkan direktif @ WebSer vice ke file, yang menyebutkan Visual Basic.NET sebagai bahasanya dan “Hello” sebagai nama yang diterapkan layanan Web.

```
<%WebService Language="Visual
Basic.NET"Class="Hello"%>
```

3 Tambahkan sebuah pernyataan Import untuk mengimpor System. Web. Services namespace, berhubung Kita akan mewarisi dari kelas Web Service yang dimuat dalam namespace ini nanti dalam proses ini.

```
Imports System.Web.Services
```

4 Tambahkan definisi kelas, termasuk atribut WebService (agar Kita bisa memodifikasi namespace default), dan pernyataan Inherits untuk mewarisi dari kelas WebService.

```
<WebService(Namespace:="http://www.aspne
tsbs.com/webservices/")>_Public Class
Hello
Inherits WebService
methoddeclaration(s)End Class
```

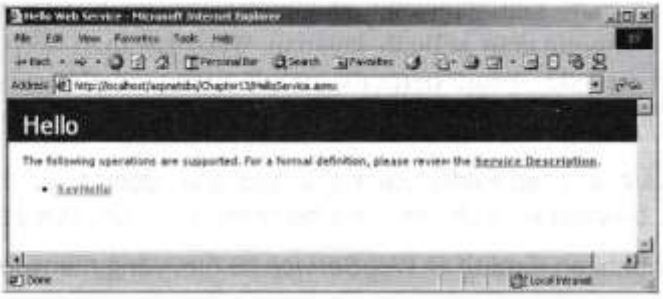
5 Tambahkan sebuah definisi metode untuk sebuah metode yang disebut SayHello, dengan sebuah tipe kembalian berupa String, yang

mengembalikan string Hello, World! Gunakan atribut WebMethod untuk menkitakan ASP.NET bahwa metode ini adalah sebuah metode layanan Web.

```
<WebMethodCl>
Public Function Say Hello ClAsString
Return"Hello,World!"
EndFunction
```

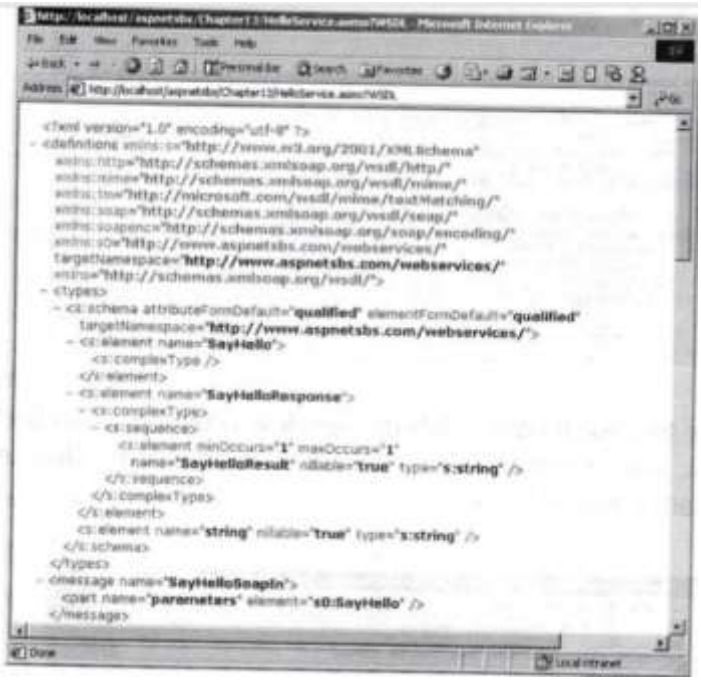
6 Simpan file tersebut.

7 Telusuri layanan Web dengan memasukkan URL file .asmx (seperti misalnya http://localhostjappname/HelloService.asmx). ASP.NET akan menghasilkan sekumpulan halaman yang menerangkan metode-metode Web yang tersedia dan sintaks yang bisa dipakai client untuk mengakses metode-metode tersebut. Output permintaan awal ke layanan Web diperlihatkan dalam gambar berikut.



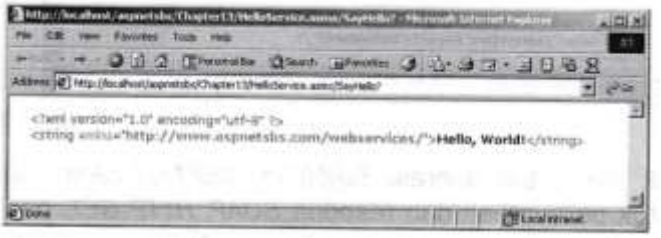
8 Klik link Service Description. ASP.NET akan mengembalikan kontrak WSDL yang menjelaskan layanan Web. Kontrak ini bisa dipakai oleh client untuk menentukan bagaimana berinteraksi dengan layanan Web. WSDL dibahas lebih lanjut dalam "Memahami File-File WSDL " pada halaman 457. Bagian dari output kontrak WSDL diperlihatkan dalam

gambar berikut. Klik Back dalam browser setelah Kita selesai melihat kontrak WSDL.



9 Klik link untuk operasi SayHello. ASP.NET akan menampilkan sintaks untuk permintaan dan respons SOAP, HTTP GET, dan HTTP POST, serta sebuah tombol yang memungkinkan Kita memanggil metode SayHello. Bagian dari output ini diperlihatkan dalam gambar berikut.

10 Klik tombol Invoke. Sebuah window browser akan terbuka ternpat hasil XML dari pemanggilan metode dikembalikan. Output ini diperlihatkan dalam gambar berikut.



Contoh sebelumnya memperlihatkan implementasi sebuah layanan Web, di mana kelas untuk layanan Web didefinisikan dalam file. asrnx. Listing berikuL HelloService_CB. asrnx dan listing HelloService_CB.vb, memperlihatkan kode untu k penerapan layanan

Web yang sama menggu nakan code-behind

Ingatlah bahwa ketika menggunakan code-behind, kelas yang menerapkan layanan Web harus secara manual dikompilasi sebelum layanan Web bisa digunakan. Perintah berikut bisa dipakai untuk mengkompilasi kelas code behind HelloService_CB.vb (perhatikan ketiga baris membentuk sebuah perintah kompilasi tunggal):

```
vbc/t:library/r:System.Web.dll/r:System.
dll
/r:System.Web.Services.dll/out:bin\Hello
ServiceCBdll HelloService_CB.vb
```

HelloService CB.asmx

```
<%@WebService
Class="ASPNETSBS.Hello_(B,HelloService_C
B"%>
```

HelloService CB.vb

```
Imports System.Web.Services
Namespace ASPNETSBS
<WebService(Namespace:="http://www.aspne
tsbs.com/webservicesl")>_Public Class
Hello
Inherits WebService
<WebMethod()>
Public Function Say Hello()As
StringReturn"Hello,World!"
EndFunction
```

```
EndClass End Namespace
```

Perhatikan dalam implementasi code-behind, namespace telah ditambahkan untuk melipat kelas Hello. Ini untuk menghindari segala konflik penamaan yang mungkin terjadi dengan assembly lain yang mungkin dipakai dalam aplikasi tersebut.

Menggunakan DataType Primitif

Contoh HelloService tidak memiliki parameter input, dan mengembalikan sebuah nilai bertipe String. Namun layanan Web berbasis-XML tidak sebatas pada string dan bisa menerima parameter-parameter input juga nilai-nilai kembalian.

Layanan Web berbasis-XML dalam ASP.NET menggunakan draft spesifikasi *XML Schema Definition* (XSD) untuk menentukan datatype yang didukung oleh layanan Web ASP.NET. Ini meliputi datatype primitif seperti misalnya kelas atau struct. Definisi untuk sebuah metode layanan Web yang menambahkan dua angka dan mengembalikan hasilnya akan tampak seperti yang berikut:

```
<WebMethod()>
Public Function  Add(Numl As
Integer,Num2 As Integer)As Integer
Return(Numl + Num2)
End Function
```

Mengakses Data dalam Layanan Web

Selain tipe-tipe primitif, kelas-kelas dan struct-struct, layanan Web ASP.NET bisa juga mengembalikan atau menerima dataset ADONET sebagai parameter input. Ini dimungkinkan karena kelas Dataset memiliki dukungan built-in untuk disejajarkan sebagai XML yang bisa dengan mudah dikirimkan lewat koneksi HTTP.

Untuk mengembalikan sebuah dataset dari layanan Web, ikuti langkah-langkah berikut ini:

1 Buatlah sebuah file baru bernama AuthorService.asmx.

2 Tambahkan direktif @ WebService ke file, yang menyebutkan Visual Basic.NET sebagai bahasanya dan Hello sebagai nama kelas yang menerapkan layanan Web tersebut.

```
<%@WebService
Language="VB"Class="Authors"%>
```

3 Tambahkan pernyataan Impor ts untuk mengimpor namespacename space System Data, System.Data.SqlClient, dan System .Web.Ser vices.

```
Imports System.Data
Imports System.Data.SqlClient Imports
System.Web.Services
```

4 Tambahkan sebuah definisi kelas untuk kelas Authors .

```
<WebServiceNamespace:="http://www.aspnet
sbs.com/webservices/'')>_PublicClassAuth
ors
Inherits WebService
methoddeclaration(s) End Class
```

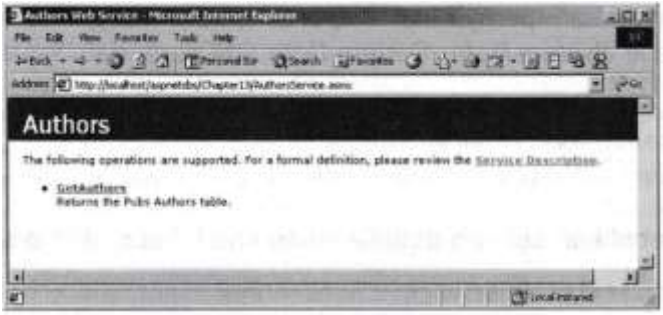
5 Tambahkan sebuah metode ke kelas yang mengembalikan tabel Authors

pada contoh database Pubs sebagai sebuah datasheet. Perhatikan kami telah menambahkan properti Description pada atribut WebMethod untuk menerangkan kegunaan metode tersebut. Lihatlah pada Bab 11 untuk lebih jelasnya mengenai penggunaan koneksi terpercaya SQL Server dalam ASP.NET.

```
<WebMethod(Description:="Returnsthe Pubs
Authors table.")>PublicFunction
GetAuthors()As Dataset
Dim Authors As New Dataset
Dim Sql Conn As New SqlConnection Dim
ConnStr As String
ConnStr="server=(local)\NetSDK;database=
pubs"ConnStr&=";Trusted_Connection=yes"
SqlConn.ConnectionString=ConnStr Dim Sql
Cmd As Sql DataAdapter
Sql Cmd= ew_
Sql DataAdapter("select*from
Authors",SqlConn)Sql
Cmd.Fi11(Authors."Authors")
ReturnAuthors End Function
```

6 Simpan file.

7 Telusuri layanan Web dengan memasukkan URL file .asmx. Output permintaan awal ke layanan Web diperlihatkan dalam gambar berikut. Perhatikan keterangan yang ditambahkan via properti Description pada atribut WebMethod yang ditampilkan sebagai bagian dari output.



8 Klik link GetAuthors. Kemudian klik tombol Invoke untuk melihat XM yang dikembalikan oleh layanan Web, yang merupakan sebuah datase serial berisi skema dan data untuk tabel Authors.

Mengiklankan Layanan Web

Setelah Kita membuat sebuah layanan Web menggunakan teknik-teknik dalam contoh-contoh terdahulu, layanan Web siap diakses oleh client. Tantangannya pada saat ini adalah bagaimana memberikan client informasi yang diperlukan agar mereka bisa mengakses layanan Web Kita.

Dalam sebuah skenario berbasis internet, ini relatif sederhana. Kita bisa memberikan client internal Kita yang potensial sebuah URL langsung ke layanan Web Kita, misalnya lewat e-mail ke mereka yang memerlukannya atau dengan membuat sebuah halaman Web terpusat yang bisa dibuka oleh kalangan intern organisasi Kita untuk menemukan URL yang dimaksud. Dalam sebuah skenario berbasis-Internet, Kita mungkin tidak tahu siapa client potensial Kita. Ini

menjadikannya jauh lebih sulit untuk memastikan bahwa mereka bisa menemukan dan menggunakan layanan Web Kita.

Ada dua solusi untuk dilema ini: penemuan layanan Web dan direktori-direktori layanan Web menggunakan Universal Description, Discovery, dan Integration.

Mengiklankan Layanan Web Melalui Dokumen Discovery

Dokumen discovery (temuan) adalah sebuah dokumen berbasis-XML berisi rujukan ke sebuah layanan Web dan/atau dokumen temuan lain. Melalui penggunaan dokumen temuan, Kita bisa dengan mudah mengkatalogkan semua layanan Web yang ada pada sebuah server Web di sebuah lokasi tunggal.

Dengan mengiklankan sebuah dokumen temuan ke sebuah URL yang tersedia secara umum, Kita bisa memungkinkan client untuk mencari dengan mudah dan menggunakan layanan-layanan Web yang telah Kita sediakan. Client bisa mengakses dokumen temuan menggunakan tool, seperti misalnya utilitas baris perintah wsdl.exe yang disertakan bersama NET Framework, atau bersama dialog Add Web Reference pada Visual Studio NET, untuk membuat client bagi layanan Web tersebut. Kita akan membahas proses ini lebih dalam pada "Menggunakan Layanan Web "

File discovery menggunakan ekstensi .disco dan berisi dua tipe simpul: simpul-simpul contractRef dan/atau discoveryRef Simpul-simpul

contractRef mengacu pada file-file WSDL yang bisa dipakai untuk membuat client-client untuk sebuah layanan Web, sementara simpul-simpul discoveryRef mengacu pada dokumen discovery/dokumen temuan tambahan. Dokumen discovery untuk kedua contoh layanan Web yang dibuat dalam bab ini diperlihatkan dalam listing, Chapterl3.disco. Atribut ref pada simpul contractRef berisi URL ke kontrak WSDL untuk layanan Web, dalam hal ini dihasilkan oleh ASP.NET dengan menambahkan argumen? WSDL ke URL file .asmx yang menerapkan layanan Web tersebut. Atribut docRef berisi URL ke dokumentasi layanan Web, dalam hal ini hanya URL ke file .asmx, berhubung ASP.NET secara otomatis membuat dokumentasi bagi layanan Web yang diterapkan dalam sebuah file .asmx. Atribut xmlns dalam simpul-simpul discoveryRef dan contractRef membentuk namespace-namespace XML tempat simpul-simpul ini berada.

Chapter 4 1.disco

```
<?xml version="l.O"encoding="utf-8"?>
<discovery xmlns="http://schemas.xml
soap.org/disco!">
<contractRefref="AuthorsService.asmx?wsd
l"docRef="AuthorsService.asmx"
xmlns="http:I/schemas.xml
soap.org/disco/scl/"I>
<contractRefref="HelloService.asmx?wsdl"
docRef="HelloService.asmx"
```

```
xmlns="http://schemas.xml
soap.org/disco/scl/"I>
<!discovery>
```

Agar mudah dibaca, beberapa URL relatif dipakai dalam listing ini. Dokumen discovery bisa berisi URL relatif maupun absolut. Apabila dipakai URL relatif mereka dianggap menjadi lokasi relatif dari URL dokumen discovery.

Mengiklankan Layanan Web Melalui UDDI

Opsi lain untuk mengiklankan layanan Web ke client potensial adalah UDDI Ini adalah sebuah inisiatif multivendor, yang dimotori oleh Ariba, IBM dan Microsoft, yang dimaksudkan untuk menyediakan sebuah registry bisnis berbasis-Internet di mana pembuat layanan Web bisa mendaftarkan layanan layanan Web mereka yang tersedia secara umum. Spesifikasi UDDI menjelaskan bagaimana layanan Web didaftarkan dalam suatu cara agar client bisa dengan mudah mencari tipe layanan Web yang diinginkan, atau menemukan layanan-layanan Web yang ditawarkan oleh mitra atau perusahaan tertentu.

Model untuk UDDI adalah bagi versi multi-identik dari registry bisnis UDDI yang akan ditampung oleh banyak partisipan, memberikan redundansi dan keseimbangan beban untuk registry. Penyedia layanan Web bisa mendaftarkan layanan Web mereka pada sembarang simpul, dan informasi akan diperbanyak ke semua registry setiap harinya,

sehingga para client potensial bisa menemukan layanan-layanan ini dari sembarang registry.

Saat ini, baik Microsoft maupun IBM menyediakan registry-registry UDDI secara online, dan keduanya bisa didapatkan dari http://www.uddi.org/. Kita bisa juga menemukan sejumlah makalah yang berguna tentang UDDI di sana.

Mengamankan Layanan Web

Satu tantangan dalam implementasi sebuah layanan Web muncul ketika Kita perlu membatasi siapa yang bisa mengaksesnya. Kita mungkin ingin membatasi akses hanya pada para client yang telah membayar iuran berlangganan. misalnya. Atau, jika layanan Web Kita menyimpan dan menggunakan data yang spesifik untuk user individual, dan Kita ingin menerapkan suatu proses login tertentu agar client bisa melihat dan memodifikasi data yang spesifik untuk mereka saja.

Satu masalah potensial dalam hal ini adalah protokol SOAP, yang berbasiskan layanan Web, tidak menyediakan suatu spesifikasi untuk keamanan layanan Web. Ini berarti semua terserah Kita memilih sarana untuk mengamankan layanan Web dengan penyediaan tingkatan pengamanan yang Kita perlukan, namun juga relatif mudah diterapkan, baik untuk Kita maupu n client Kita. Hal terakhir yang perlu dilakukan adalah bagaimana menjadikan para client Kita tidak kemana-mana lagi selain ke layanan Web Kita.

Mengeksplorasi Opsi-Opsi Otentifikasi

Pilihan pertama yang harus Kita buat sekarang adalah bagaimana mengesahkan para pengguna layanan Web Kita. Ada ban yak opsional, masing-masing dengan kelebihan dan kekurangannya.

Walaupun ASP.NET tidak mendefinisikan suatu metode otentifikasi dan otorisasi spesifik untuk layanan-layanan Web, Kita bisa memanfaatkan mekani sme otentifikasi yang ditawarkan IIS.

Kelebihan penggunaan salah satu mekanisme otentifikasi IIS adalah bahwa infrastruktur bagi mereka sudah ada. Kita tidak perlu membuat sendiri penyimpanan record bagi informasi pengguna Kita. Satu kekurangannya adalah bahwa setiap metode otentifikasi yang ditawarkan oleh IIS mengharuskan pembuat an akun-akun NT bagi setiap client aplikasi Kita. Ini bisa menyulitkan pengelolaannya, tergantung pada bagaimana seting server Web Kita (apakah bagian dari domain NT, apakah menggunakan Active Directory, dan sebagainya). Kekurangan lain adalah bahwa metode otentifikasi Integrated Windows, walaupun aman, mengharuskan client memiliki domain yang sama (atau domain terpercaya) sebagai servernya, yang menjadikannya tidak praktis untuk penggunaan Internet. Otentifikasi dasar cocok untuk penggunaan Internet, namun mengirimkan username dan password sebagai teks yang jelas, sehingga menjadikannya mudah diutakatik. Ia bisa menggunakan enkripsi SSL untuk mencegah munculnya masalah, namun bisa menambah sebuah overhead kinerja yang tidak bisa

diterima jika dokumen pengguna harus dikirimkan bersama setiap permintaan layanan Web.

Opsi lain untuk otentifikasi pengguna adalah mekanisme otentifikasi dari pihak ketiga, seperti misalnya Microsoft Passport. Untungnya, dukungan tool pengembang untuk Passport masih jarang ditemui, dan sebagian besar tool yang tersedia memfokuskan agar memungkinkan penggunaan Passport untuk otentifikasi pengguna dari situs Web, dan bukannya layanan Web walaupun ini sepertinya Passport dan mekanisme otentifikasi pihak-ketiga lainnya akan menambahkan dukungan bagi implementasi dalam layanan Web pada akhirnya, sekarang mekanisme-mekanisme ini mengharuskan terlalu banyak usaha pada pihak client.

Terakhir, Kita bisa "memutar sendiri" otentifikasi layanan Web dengan menerapkan sebuah fungsi login roll dalam satu atau semua layanan Web Kita, dengan metode login yang mengembalikan sebuah kunci unik (yang harus diacak atau disandikan) untuk menkitai user yang sedang login. Kunci ini kemudian akan dikirimkan sebagai sebuah parameter pada setiap permintaan layanan Web, yang menkitakan bahwa pengguna tersebut adalah pengguna yang telah disahkan. Agar bisa melindungi username dan password, metode layanan Web login harus diminta melalui sebuah koneksi SSL. Permintaan selanjutnya bisa dibuat lewat koneksi HTTP stkitar, jadi untuk menghindari overhead kinerja pada koneksi SSL. Untuk mengurangi resiko pengguna yang

sedang ditiru oleh seseorang yang telah mencuri kunci login tersebut, kunci kunci ini harus disimpan dengan suatu cara yang bisa dengan mudah membuatnya kadaluwarsa setelah nilai timeout yang ditentukan sebelumnya (dalam database back-end, misalnya).

Walaupun metode otentifikasi ini lebih sulit diterapkan dan diatur daripada sebagian yang yang lainnya, metode ini adalah yang paling sederhana untuk dipakai client Kita. Setelah Kita memberikan informasi kepada client tentang akunnya (username dan password), client tinggal membuat permintaan untuk login ke layanan Web (melalui koneksi SSL) dan menyimpan kunci loginnya. Pada setiap permintaan selanjutnya, client cukup memasukkan kunci tersebut sebagai salah satu parameter pada metode layanan Web yang dipanggil. Apabila nilai timeout kedaluwarsa antara satu permintaan dengan yang berikutnya, pengguna harus memanggil lagi metode login tersebut untuk menerima sebuah kunci baru. Oleh karena itu, nilai harus diseting agar memberikan keseimbangan antara keamanan dan kenyamanan. Di samping metode login, Kita bisa jug a menerapkan sebuah metode logout dalam layanan Web Kita untu k menjadikan kunci login kadaluwarsa secara eksplisit.

Menggunakan Layanan Web

Sekarang Kita telah memperoleh sebuah layanan Web yang efektif, Kita mungkin perlu mengetahui bagaimana menggunakannya, melalui

pengujian dengan memanggilnya dari sebuah browser. Walaupun memformat semua permintaan dan respons SOAP yang diperlukan untuk memanggil sebuah layanan Web bisa jadi terasa rumit, NET Framework telah mengabstraksikan banyak kerumitan ini.

Walaupun begitu, Kita perlu tahu bagaimana menemukan dan menggunakan sebuah layanan Web, termasuk yang berikut ini:

1 Menemukan sebuah layanan Web, menggunakan dokumen discovery atau UDDI

2 Menggunakan utilitas WSDL.EXE untuk membuat sebuah kelas proxy bagi layanan Web

3 Menggunak an kelas proxy dari sebuah halaman ASP.NET atau aplikasi konsol untuk mengakses layanan Web.

Menemukan Layanan Web

Langkah pertama dalam menggun akan layanan Web berbasis-XML adalah menemukannya. Apabila Kita berurusan dengan sebuah layanan Web yang dibuat oleh Kita atau seseorang dalam organisasi, Kita mungkin telah tahu lokasi layanan Web tersebut. Jika demikian, Kita bisa melompati langkah ini.

Menemukan sebuah layanan Web pada dasarnya adalah citra cermin dari mengiklankan layanan Web, jadi teknik yang sama bisa dipakai di sini. Ada dua teknik untuk menemukan sebuah layanan Web dalam ASP.NET: dokumen discovery dan UDDI.

Menemukan Layanan Web dengan Dokumen Discovery

Agar bisa menggunakan dokumen discovery untuk menemukan layanan Web, Kita perlu URL untuk dokumen discovery yang menerangkan layanan layanan yang menarik untuk Kita. Kita bisa menampilkan URL dokumen tersebut dari sebuah listing pada situs Web umum organisasi Kita, dari sebuah promosi e-mail, atau dari katalog.

Setelah Kita mempunyai URL untuk dokumen discovery, Kita bisa meng- gunakan utilitas baris perintah layanan Web untuk menghasilkan kontrak WSDL bagi penggunaan layanan Web yang telah dijelaskan dalam dokumen discovery. Sintaks untuk utilitas disco.exe adalah

```
disco[options]URL
```

Di sini, options menyatakan satu atau beberapa opsi baris-perintah untuk utilitas tersebut, dan URL menyatakan URL ke dokumen discovery. Opsi-opsi untuk utilitas disco.exe diperlihatkan dalam tabel berikut.

Opsi opsi disco.exe

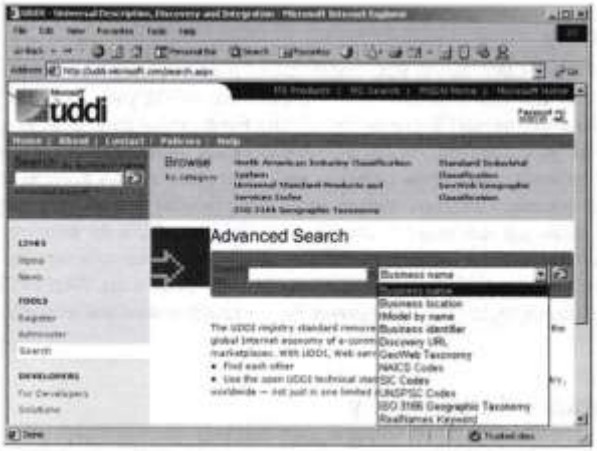
Opsi	Keterangan
Id:domain atau Id domain: domain /nosave	Menyeting domain yang dipakai saat menghubungkan ke sebuah situs yang meminta otentifikasi. Mencegah disco.exe dari

	menyimpan file file untuk WSDL yang ditemukan dan dokumen discovery .
/nologo / o:directory atau	Mencegah ditampilkannya banner pendahuluan Menyediakan nama direktori tempat output akan disimpan.
/out: directory / p: password atau Ipassword:password	Menyeting password yang dipakai saat koneksi ke situs yang meminta otentifikasi.
/proxy: url / pd: domain atau /proxy domain: domain	Menyeting proxy yang dipakai untuk permintaan HTTP. Menyeting domain yang dipakai saat berhubungan ke sebuah server proxy yang mengharuskan otentifikasi.
/ p p: password atau /proxypassword: password	Menyeting password yang dipakai saat berhubungan ke sebuah server proxy yang mengharuskan otentifikasi.
username atau/proxyusemame*:* username	Menyeting username yang dipakai saat berhubungan ke

	sebuah server proxy yang mengharuskan otentifikasi.
/u: username atau Iusername: username /?	Menyeting username yang dipakai saat berhubungan ke sebuah situs yang mengharuskan otentifikasi. Menyediakan bantuan untuk opsiopsi baris-perintah.

Menemukan Layanan Web dengan UDDI

Tidak seperti halnya menggunakan sebuah dokumen discovery untuk menemukan layanan Web, UDDI tidak mengharuskan Kita mengetahui lebih dulu URL dokumen yang menerangkan layanan Web yang diinginkan. Segala penyedia registry bisnis UDDI biasanya akan menyediakan sebuah tool untuk mencari instance registry tersebut untuk berbagai perusahaan atau layanan Web yang memenuhi keperluan Kita. Kriteria yang bisa Kita cari tergantung pada organisasi yang menampung instance tersebut. Sebagai contoh, registry UDDI Microsoft di http://uddi.microsoft.com mendukung pencarian berdasarkan nama perusahaan , lokasi perusahaan, URL discovery , kata kunci, dan beragam kriteria lainnya. Gambar berikut memperlihatkan daftar kriteria yang tersedia bagi pencarian registry UDDI Microsoft.



Sebuah teknik efektif untuk mencari layanan-layanan Web pada registry http://uddi.microsoft.com adalah mencari berdasarkan URL discovery dan mencari URL yang berisi "wsdl ". URL demikian hampir bisa dipastikan menunjuk ke kontrak-kontrak WSDL langsung yang bisa Kita gunakan untuk mengakses layanan-layanan Web yang tersedia.

Memahami File-File WSDL

WSDL adalah sebuah stkitar berbasis-XML untuk dokumen yang menerangkan layanan yang diekspos oleh sebuah layanan Web. Kita bisa menampilkan kontrak WSDL bagi seluruh layanan Web yang Kita buat dengan ASP.NET dengan cara memanggil URL sebuah layanan Web dalam browser dan menambahkan? WSDL pada layanan Web tersebut.

Kontrak-kontrak WSDL mirip dengan pustaka bertipe COM atau metadata assembly NET. Mereka dipakai untuk menerangkan anggota publik dari layanan Web, termasuk semua metode yang tersedia secara publik, parameter (dan datatype) yang diharapkan metode tersebut saat dipanggil, dan tipe kembalian dari segala nilai hasil. Oleh karena informasi ini dikodekan dalam format stkitar berbasis-XML, semua client yang bisa membaca XML dan bisa memahami WSDL dan SOAP pasti bisa dengan efektif menggunakan aplikasi Web yang diterangkan kontrak WSDL.

Membuat Kelas Proxy

Menggunakan sebuah layanan Web via SOAP sangatlah kompleks. Cara ini melibatkan pembuatan dan pemformatan sebuah amplop SOAP, seperangkat header SOAP, dan tubuh SOAP. Semua ini dilakukan beserta elemen-elemen dan atribut atribut yang benar bagi layanan Web yang sedang Kita targetkan. Untung saja, dengan adanya dukungan built-in untuk layanan Web dalam .NET Framework, semua plumbing yang diperlukan untuk mengakse layanan Web disediakan untuk Kita. .NET Framework menyediakan utama baris-perintah lain, wsdl.exe, untuk membantu Kita membuat sebuah kelas proxy .NET untuk layanan Web yang diterangkan dengan sebuah kontrak WSDL.

Menggunakan Utilitas wsdl.exe

Utilitas wsdl.exe menghasilkan sebuah kelas proxy dalam C#, Visual Basic.NET, atau JScript .NET untuk sebuah kontrak WSDL yang disebutkan dengan URL atau path. Kelas proxy ini bisa dimunculkan dan metodenya bisa dipanggil oleh sembarang client, persis seperti kelas.NET yang lain. Sintaks untuk utilitas wsdl.exe adalah sebagai berikut:

```
wsdl[options]{URL I path}
```

Di sini, options mewakili satu atau beberapa opsi baris-perintah untuk utilita tersebut, dan URL mewakili URL ke kontrak WSDL. Bisa juga, sebuah path file ke kontrak WSDL dapat disalurkan sebagai sebuah argumen path. Opsi-opsi untuk wsdl.exe diperlihatkan dalam tabel berikut.

Opsi opsi wsdl.exe

Opsi	Keterangan
/ appsettingurlkey: key atau / url key: key	Menyeting·kunci seting konfigurasi dari mana dilakukan pembacaan URL default untuk menghasilkan kode.
I appsettingbaseurl: baseurl atau /baseurl: baseurl	Menyeting URL basis yang dipakai untuk menghitung URL relatif saat kutipan URL

	disebutkan bagi argumen URL.
/ d: domain atau /domain:domain	Menyeting domain yang dipakai ketika berhubungan ke situs yang meminta otentifikasi
/! language atau /languange: language	Menyeting bahasa yang dipakai saat membuat kelas proxy. Nilai-nilai yang berlaku adalah: CS, VB, JS, atau nama yang memenuhi syarat bagi kelas yang menerapkan kelas System.CodeDom. Compiler.CodeDomProvider. Defaultnya adalah CS (C#).
/ n: namespace atau / namespace: namespace	Menyeting namespace yang akan dipakai saat membuat kelas proxy
/nologo	Mencegah ditampilkannya banner pendahuluan
/ofilename atau output./out filename	Menyediakan nama file (dan path) yang akan dipakai untuk disimpan
/ p password atau	Menyeting password yang akan

/password password	dipakai saat berhubungan ke sebuah situs yang meminta otentifikasi.
/protocol protocol	Menyeting protokol (SOAP, HTTP-GET, atau HTIP-POST yang dipakai untuk permintaan. Defaultnya adalah SOAP.
/ pd: domain atau /proxydomain: domain	Menyeting domain yang dipakai saat berhubungan ke penyedia yang mengharuskan otentifikasi.
/ p p password atau Iproxypassword password	Menyeting password yang dipakai saat berhubungan ke sebuah server proxy yang meminta otentifikasi.
/ pu: username atau /proxyusername: username	Menyetingu sername yang akan dipakai saat berhubungan ke server proxy yang meminta otentifikasi.
/server	Membuat sebuah kelas basis abstrak dengan mtrriacrs yang sama seperti yang disebutkan dalam kontrak WSDL
/ u: username atau	Menyetingu sername yang

/username :username	dipakai saat berhubungan ke sebuah situs yang mengharuskan otentifikasi.
/?	Menyediakan bantuan untuk opsi-opsi baris-perintah.

Ikutilah langkah-langkah berikut untuk membuat kelas-kelas proxy dalam Visual Basic .NET bagi contoh layanan-layanan Web HelloService dan AuthorsService yang dibuat sebelum bab ini.

1 Buka sebuah baris perintah dan arahkan ke direktori tempat Kita ingin membuat kelas-kelas proxy (atau, Kita bisa menggunakan opsi /o untuk menyebutkan path dan nama file untuk menyimpan kelas proxy tersebut).

2 Jalankan utilitas wsdl.exe, menggunakan opsi // untuk menyebutkan Visual Basic .NET sebagai bahasanya, dan menyalurkan URL ke file HelloService.asmx menggunakan parameter? WSDL, yang memberitah ASP.NET untuk mengembalikan kontrak WSDL bagi file .asmx (kedua baris di bawah akan membentuk sebuah perintah tunggal). URL ya ng Kita sebutkan harus pas dengan lokasi file .asmx pada server Web Kita, dan mungkin berbeda dari URL berikut:

```
wsdl/l:vb/n: ASPNETSBS
http://localhost/aspnetsbs/Chaterl3/Hell
oService.asmx?WSDL
```

3 Jalankan lagi utilitas wsdl.exe bagi AuthorsService menggunakan sintaks yang sama dengan yang diperlihatkan dalam langkah 2, namun

ubahla URL-nya untuk menunjuk AuthorsService.asmx.

4 Bisa juga, Kita membuat sebuah file batch (dengan ekstensi .bat) berisi kedua perintah pendahuluan. Jika Kita ingin melihat hasil eksekusi batch sebelum window perintah tertutup, tambahkan sebuah perintah pause di akhir file batch.

Setelah Kita membuat kelas proxy, Kita perlu mengkompilasi kelas kelas menggunakan compiler baris perintah untuk bahasa yang Kita pilih (dalam contoh ini, Visual Basic .NET). Listing berikut, MakeServices.bat, memperlihatkan sintaks compiler baris perintah untuk mengkompilasi kelas kelas ini dan menyalurkan assembly yang dihasilkan ke subdirektori bin dari direktori tempat perintah dijalankan. Perhatikan listing ini menampilkan sebuah perintah tunggal dan dilipat menjadi beberapa baris agar mudah dibaca.

MakeServices.bat

```
vbc/t:library/r:System.Web.dll/r:System.
dll
/r:System.Web.Services.dll/r:System.Xml.
dll/r:System.Data.dll
/out:bin\Services.dll Authors.vbHello.vb
```

Membuat Halaman Web Forms Client

Setelah Kita mengkompilasi kelas-kelas proxy untuk layanan Web yang ingin Kita gunakan, Kita menggunakan mereka dari sebuah aplikasi client atau halaman Web Forms persis seperti yang Kita lakukan dengan kelas .NET lain. Untuk halaman Web Forms, Kita cukup menambahkan

sebuah direktif @ Import ke halaman tersebut. Kemudian, dalam prosedur tingkat halaman (seperti event handler Page_L oad), buatlah sebuah instance kelas proxy dan panggillah metodenya.

Untuk membuat sebuah Web Forms client bagi layanan Web HelloService yang dibuat sebelum bab ini, ikuti langkah-langkah berikut:

1 Buatlah sebuah file baru bernama Hello.aspx dalam direktori yang berhubungan dengan aplikasi IIS. Aplikasi ini harus memiliki subdirektori bin yang berisi assembly yang dibuat oleh perintah MakeServices.bat.

2 Tambahkan tag-tag stkitar HTML, direktif @ Page, dan kontrol Label ASP.NET.

```
<%@Page Language="VB"%>
<html>
<head>
<!head>
<body>
<asp:labelid="mylabel"runat="server"I>
<!body>
</html>
```

3 Tambahkan sebuah direktif @ Import untuk mengimpor namespace ASP- NETSBS. Direktif ini harus berada pada baris setelah direktif @ Page.

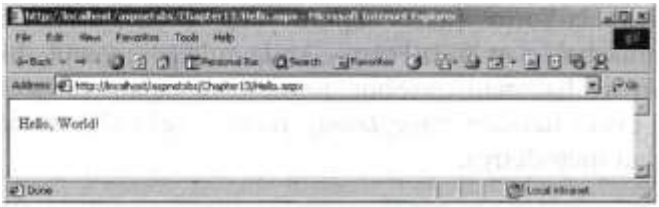
```
<%@ImportNamespace="ASPNETSBS"%>
```

4 Tambahkan sebuah blok <script> berisi event handler Page_Load. Dalam event handler, buatlah sebuah instance baru dari kelas Hello dan

panggil metode SayHellonya, teruskan hasilnya ke properti Text pada kontrol Label ASP.NET.

```
<scriptrunat="server">SubPage_Load()
DimHelloAsNewHello()
mylabel.Text=Hello.SayHello()EndSub
<!script>
```

5 Simpan dan telusuri halaman. Hasilnya akan mirip seperti gambar berikut.



Membuat Aplikasi Console Client

Perlu diperhatikan bahwa client-client layanan Web tidak sebatas Web Forms ASP.NET. Kita bisa dengan mudah menggunakan layanan Web dari aplikasi Windows Forms, aplikasi Console, atau bahkan layanan Web lain. Untuk mendemonstrasikan betapa sederhananya menggunakan layanan Web dari client semacam itu, ikuti langkah-langkah berikut untuk membuat sebuah aplikasi Console sederhana yang menggunakan AuthorsServic e. Aplikasi ini telah dibuat sebelumnya dalam bab ini.

1 Buatlah sebuah file kelas baru bernama AuthorsConsole.vb.

2 Tambahkan kode dalam AuthorConsole.vb ke file kelas. Kode ini membuat sebuah dataset lokal untuk menerima data dari layanan Web,

dan kemudian diubah pada setiap baris dan kolom dan menuliskan data yang terkandung dalam dataset ke konsol.

3 Kompilasi kelas menggunakan perintah berikut. (Sebagaimana dengan MakeServices.bat, kode ini akan dieksekusi sebagai sebuah perintah tunggal). Jika assembly Services.dll yang telah dikompilasi tidak berada dalam direktori bin dari aplikasi, pindahkan hasil executable ke direktori yang sama tempat assembly yang berisi kelas proxy AuthorsService berada.

```
vbc/r: system.dll/r: system.data.dll
/r: microsoft.visualbasic.dll/r:
system.xml.dll
/r: system.web.services.dll/r:
bin\services.dll
/out:bin\authorsconsole.exeauthorsconsol
e.vbpause
```

4 Buka sebuah baris-perintah dan arahkan ke folder berisi assembly dan jalankan menggunakan perintah berikut.

```
AuthorsConsole
```

Listing berikut, AuthorConsole.vb, memperlihatkan semua kode untuk aplikasi konsol.

AuthorConsole. vb

```
Imports ASPNETSBS Imports System Imports
System.Data
Imports System.Web.Services Imports
System.Xml
Imports Microsoft.Visual Basic
Public Class AuthorsConsole Public
Shared Sub Main()
```

```
Dim AuthorsDS As Dataset Dim this Row As
Data Row
Dim thisColumn As Data Column Dim
thisTable As DataTable
Dim AuthorsString As String Dim Authors
As New Authors()
AuthorsDS=Authors.GetAuthors()thisTable=
AuthorsDS.Tables(O)
For Each this Row In thisTable.Rows
AuthorsString&="----------------------"
For Each thisColumn in thisTable.Columns
AuthorsString&=vbCrLf&vbTab
If Not(this Row(thisColumn)Is
Nothing)Then If Not IsDbNull(this
Row(thisColumn))Then
AuthorsString&=CStr(this Row(thi
sColumn))
Else
AuthorsString&="NULL"End If
AuthorsString&=vbCrLf&vbTab
End If Next
AuthorsString&=vbCrLf
Next
Console.WriteLine(AuthorsString)End Sub
End class
```

Gambar berikut memperlihatkan sebagian output aplikasi Console.

C. PENUTUP

Ringkasan

Untuk	Lakukan ini
Membuat layanan Web berbasis-XML menggunakan ASP. NET	Buatlah sebuah file dengan ekstensi file .asmx dan tambahkan direktif @ WebService ke file. Buatlah sebuah kelas dalam file untuk menerapkan sebuah layanan Web, dengan menambahkan atribut WebMethod ke setiap metode yang ingin Kita ekspos dari layanan Web.
Menguji layanan Web	Telusuri (browse) file .asmx yang berisi layanan Web dari sebuah browser Weq. ASP.NET akan menghasilkan halaman-halaman dokumentasi bagi layanan Web,

	yang memungkinkan Kita memanggil layanan Web itu.
Mengiklankan Layanan Web	Buat sebuah dokumen discovery untuk layanan Kita dan publikasikan dokumen ke lokasi publik, seperti situs, Web organisasi Kita, atau daftarkan layanan Web Kita ke salah satu registry bisnis UDDI yang tersedia secara urhum, pastikan memberikan URL ke kontrak WSDL bagi masing-masing layanan Web yang Kita daftarkan.
Menggu nakan layanan Web dari ASP.NET	Gunakan utilitas wsdl.exe untuk membuat sebuah kelas proxy berdasarkan pada kontrak WSDL untuk layanan Web, kompilasi kelas proxy, dan benttiklah kelas proxy dari aplikasi client Kita seperti kelas .NET lainnya.

Pertanyaan

1. Apa yang anda ketahui tantang Layanan Web Berbasis XML?
2. Tuliskan perintah untuk mendeklarasikan layanan web?
3. Jelaskan maksud dari Pewarisan pada WebService?
4. Apa itu Atribut Metadata? dan berikan contohnya.
5. Sebutkan dan Jelaskan Atribut Properti.

E. DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.

13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 5

Menggunakan WSDL untuk Dokumen Layanan Web

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 5 akan di bahas penggunaan WSDL untuk Dokumen Layanan Web, yang mencakup diantaranya: Sintaks Dokumen WSDL, Elemen Definitions, Elemen Types, Elemen Message, Elemen Port Type, Elemen Binding, Elemen Services, Elemen Perluasan, Perluasan SOAP, Mengikat Elemen, Dokumentasi.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan Sintaks Dokumen WSDL, Elemen Definitions, Elemen Types, Elemen Message, Elemen Port Type, Elemen Binding, Elemen Services, Elemen Perluasan, Perluasan SOAP, Mengikat Elemen, Dokumentasi.

B. PENYAJIAN

Menggunakan WSDL untuk Dokumen Layanan Web

Kita telah belajar cara membuat skema untuk menjelaskan format pesan SOAP. Kita dapat menggunakan XML Schema untuk menerangkan layout pesan dan tipe data yg dimuatnya. Skema yang dihasilkan juga dapat digunakan untuk melakukan validasi pesan yang diterima oleh server Web. Namun, XML Schema sendiri tidak dapat menerangkan layanan Web secara lengkap.

Sebuah layanan Web Calculator, layanan Web akan memunculkan dua metode, add dan Subtract. Kedua metode itu menerima dua integer dan menghasilkan satu integer yang mengandung hasilnya, yaitu Add yang menghasilkan jumlah dua integer, dan Substract yang menghasilkan perbedaan antara dua angka.

Untuk menjelaskan bagaimana client berinteraksi dengan layanan Web, Penulis mendefinisikan skema untuk pesan-pesan yang akan dipertukarkan antara client dan server. Skema ini berisi definisi tipe yang kompleks untuk pesan permintaan dan jawaban, baik untuk metode Add maupun Substract. Kita perlu ingat bahwa tujuan utamanya bukan untuk mempersulit pengembang dalam definisi skema untuk memahami cara berinteraksi dengan layanan Web.

Selain informasi yang disediakan oleh skema, apa yang perlu diketahui oleh client untuk memanggil metode yang dimunculkan oleh layanan Web Calculator? Karena pesan SOAP dapat berisi apa saja yang tidak menolak XML, tiap pesan SOAP bisa dikombinasikan untuk mendukung banyak variasi pola pertukaran pesan. Pola pertukaran pesan untuk layanan Web Calculator tidak begitu sulit, tapi asosiasi formal antara pesan permintaan Add dan Subtract dengan pesan jawabannya akan menghapus setiap ambiguitas yang mungkin timbul.

Penjelasan formal pada pola pesan, bahkan lebih penting untuk layanan Web yang lebih kompleks. Sebagian layanan Web dapat menerima permintaan tapi tidak dapat mengirim balik jawabannya ke client. Sebagian lagi hanya bisa mengirim ke client.

Skema ini juga tidak berisi informasi tentang cara mengakses layanan Web.

Karena SOAP tidak bergantung pada protokol, pesan dapat dipertukarkan antara client dan server dengan banyak cara. Bagaimana Kita memutuskan harus mengirim pesan lewat HTTP, SMTP, atau protokol transpor lainnya. Selain itu, bagaimana Kita tahu ke alamat mana pesan itu harus dikirimkan?
WSDL (*Web Service Description Language*) adalah dialek berbasis XML yang lapisannya ada di atas skema yang menjelaskan layanan Web. Dokumen WSDL menyediakan informasi yang dibutuhkan client untuk berinteraksi dengan layanan Web. WSDL dapat diperluas dan dipakai untuk menjelaskan hampir semua layanan jaringan, termasuk SOAP lewat HTP. Bahkan, protokol yang bukan berbasis-XML, seperti DCOM lewat UDP.

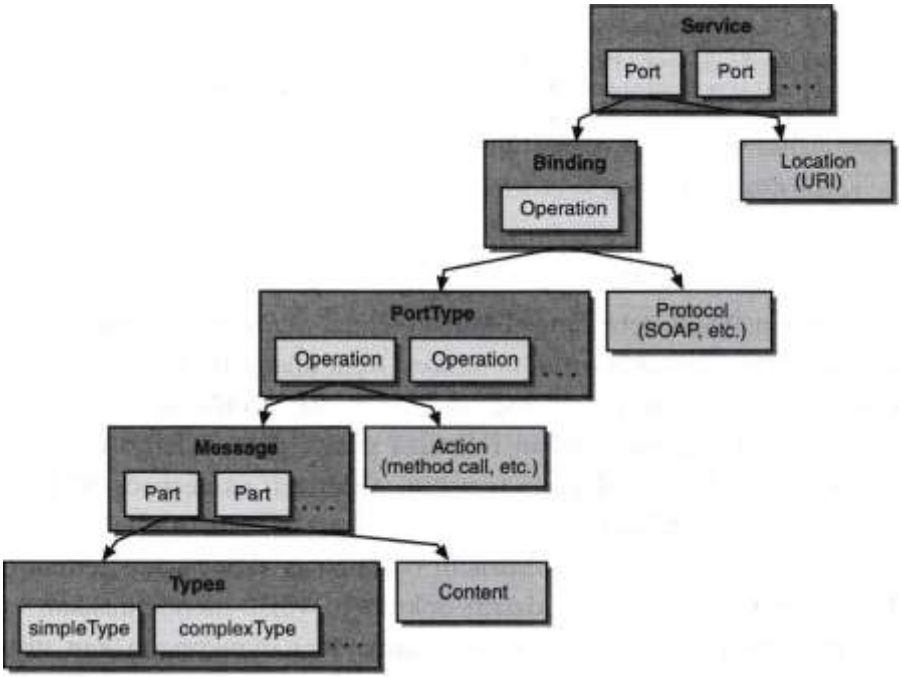
Sintaks Dokumen WSDL

Pada mulanya, dokumen WSDL akan membingungkan. Namun, sintaks pada dokumen WSDL tidak serumit dokumen XML Schema. Dokumen WSDL terdiri dari serangkaian asosiasi yang lapisannya ada di atas dokumen XML Schema yang menjelaskan layanan Web. Asosiasi ini menambah ukuran dan anggapan kerumitan di dalam dokumen WSDL. Namun, setelah melihat kedalamnya, dokuman WSDL ternyata tidak sulit.
Akar dari dokumen WSDL adalah elemen definitions. Di dalam elemen terdapat lima tipe elemen anak.

1. **Types**. Berisi definisi skema pesan yang dapat dikirim dan diterima oleh layanan ini. Cara yang biasa digunakan untuk mewakili skema ini adalah dengan menggunakan XML Schema.
2. **Message**. Berfungsi sebagai rujukan silang yang mengasosiasikan pesan dengan definisinya di dalam skema.

3. **PortType**. Mendefinisikan serangkaian yang dapat dimunculkan oleh layanan Web. Antar muka terkait dengan satu atau lebih pesan.
4. **Binding**. Mengasosiasikan definisi portType dengan protokol tertentu.
5. **Service**. Mendefinisikan sekelompok endpoint (ports) terkait yang diekspos oleh layanan Web.

Diagram berikut ini menggambarkan bagaimana lapisan kelima elemen di atas disusun di atas definisi skema untuk menjelaskan layanan Web.



Seperti yang Kita lihat, dokumen WSDL terdiri dari sejumlah asosiasi. Misalnya, bagian-bagian pesan digunakan untuk mengasosiasikan definisi

datatype dengan salah satu bagian dari isi pesan.

Elemen Definitions

Akar elemen di dalam dokumen WSDL, yaitu elemen definitions, memiliki peran yang sama dengan elemen schema dalam dokumen XML Schema. Akar elemen ini berisi elemen-elemen yang mendefinisikan layanan tertentu.

Sangat mirip dengan dokumen XML Schema, dokumen WSDL dapat mendefinisikan namespacenya sendiri dengan menambahkan atribut targetNamespace ke elemen definitions. Satu-satunya batasan adalah bahwa nilai pada atribut targetNamespace tidak boleh berisi URL relatif.

Namespace WSDL memungkinkan Kita mencocokkan referensi secara penuh dengan entitas yang didefinisikan di dalam dokumen WSDL. Misalnya, sebuah definisi pesan dirujuk oleh definisi portType. Selanjutnya dalam bab ini, akan merujuk entitas yang didefinisikan dalam namespace lain untuk mempermudah pewarisan antar muka.

Bagian dari WSDL berikut mendefinisikan elemen definitions untuk layanan Web Calculator.

```
<?xml version-"1.0 "encoding-"utf-8"?>
<definitionstargetNamespace-
"http://somedomain/Calculator/wsdl"xmlns:tns-
"http://somedomain/Calculator/wsdl"
xmlns-"http://schemas.xml soap.org/wsdl/">
<!--Definitions will go here -->
<! definitions>
```

Dokumen WSDL sebelumnya berisi elemen definitions. Namespace target diatur ke http://somedomain/Calculator. Kemudian , buatlah satu referensi ke

namespace target dan memasangkan prefiks tns:. Prefiks ini digunakan di dalam dokumen untuk mencocokkan referensi secara penuh dengan entitas yang didefinisikan di dalam dokumen tersebut. Pada akhirnya, namespace WSDL diatur dengan namespace default.

Elemen definitions mendefinisikan batas-batas name scope tertentu. Elemen-elemen yang dinyatakan di dalam dokumen WSDL mendefinisikan entitas, seperti port dan pesan. Entitas ini diberi nama dengan menggunakan atribut name. Semua atribut di dalam cakupan nama harus unik. Misalnya, jika dokumen WSDL berisi port dengan nama Foo, maka tidak boleh memiliki port atau pesan lain yang namanya Foo.

Mungkin, mendefinisikan sebuah URL berkualifikasi penuh yang unik untuk namespace tidak selalu praktis. Misalnya, diawal pengembangan atau jika Kita ingin membuat sejumlah layanan Web eksperimental. Dalam hal ini, Kita dapat menggunakan http://tempuri.org, URL khusus yang digunakan berdasarkan kesepakatan untuk mendefinisikan namespace yang tidak perlu diidentifikasi secara unik.

Elemen types

Elemen types berisi informasi skema yang dirujuk di dalam dokumen WSDL. Sistem tipe default yang didukung oleh WSDL adalah XML Schema. Jika XML Schema digunakan untuk mendefinisikan tipe yang ada di dalam elemen types, elemen schema akan langsung muncul sebagai elemen anak.

Kita dapat menggunakan sistem tipe lainnya berdasarkan perluasannya. Jika Kita menggunakan sistem tipe lainnya, sebuah elemen perluasan akan muncul di bawah elemen types. Nama elemen itu harus menyebutkan sistem tipe yang

digunakan. Dalam bab ini, dibatasi pembahasan pada XML Schema karena merupakan sistem tipe yang dominan di dalam dokumen WSDL, termasuk yang dibuat untuk layanan Web pada platform .NET.

Layanan Web Calculator akan memunculkan dua metode bergaya RPC, satu metode Add dan satu metode Substract. Pesan ini akan dikodekan dengan cara yang sama seperti yang ditunjukkan dalam Bab sebelumnya. Perbedaannya, skema itu akan dilekatkan di dalam dokumen WSDL, seperti di bawah ini:

```
<?xml version="l.O•encoding=="utf-8"?>
<definitionstargetNamespace=http://somedomain/Calculator/wsdl
xmlns:tns="http://somedomain/Calculator/wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"xmlns:s="http://somedomain/
Calculator/schema"xmlns="http://schemas.xml soap.org/wsdl/">
<types>
<schemaattributeFormDefault="qualified"elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://somedomain/Calculator/schema">
<!-- Definitions for both the Add and Subtract SOAP messages -->
<elementname="Add">
<complexType>
        <all>
        <elementname="x•type=="int"/>
        <elementname="y"type=="int"/>
        </all>
</complexType>
</element>
<elementname="AddResult">
<complexType>
<all>
<elementname="result•type=="int"/>
</all>
</complexType>
```

```
<!element>
<elementname="Subtract">
<complexType>
<a11>
<elementname"x"type--"int"/>
<elementname"y"type--"int"/>
</a11>
</complexType>
<! element>
<elementname"SubtractResult">
<complexType>
<a11>
<elementname"result"type--"int"/>
<!all>
</complexType>
<!element>
<!- - Common SOAP fault detail element used by Add and Subtract - - >
<elementname"CalculateFault">
<complexType>
<a11>
<elementname"x"type--"int"/>
<elementname"y"type--"int"/>
<elementname"Description"type--"string"/>
</a11>
</complexType>
<element>
<!schema>
<!types>
<! - - More definitions will go here. - - >
<!definitions>
```

Di dalam elemen types terdapat definisi skema untuk metode Add dan Substract yang menggunakan referensi untuk namespace skema yang muncul

sebelumnya di dalam elemen definitions.

WSDL tidak terbatas pada penjelasan format serialisasi berbasis XML. Kita dapat memanfaatkan WSDL untuk menjelaskan layanan yang menggunakan format lain, termasuk biner. Misalnya, Kita dapat menggunakan WSDL untuk menjelaskan layanan yang dimunculkan lewat DCOM. Dalam hal ini, Kita masih dapat menggunakan XML Schema untuk menjelaskan data yang dikirim lewat kabel. Spesifikasi WSDL menyediakan rekomendasi berikut ini untuk melakukannya:

1. Jelaskan data yang menggunakan elemen, bukannya atribut. Misalnya, tiap parameter harus dikodekan di dalam elemennya sendiri, dengan cara yang hampir sama dengan pengodean SOAP.
2. Jelaskan hanya data yang terkait dengan pesan dan tidak spesifik digunakan untuk kode koneksi. Misalnya, parameter yang diteruskan ke objek jarak jauh COM harus dijelaskan di dalam skema. Namun, 010 (Object Identifier/ Penunjuk Objek) DCOM adalah data protokol koneksi spesifik yang menunjukkan objek dan tidak perlu dijelaskan di dalam skema.
3. Tipe array harus bersumber dari tipe kompleks Array yang didefinisikan di dalam skema SOAP Encoding. Menurut kesepakatan, nama tipe harus merupakan tipe item di dalam array, dengan prefiks ArrayOf
4. Parameter yang dapat berisi data bertipe apapun harus didefinisikan oleh elemen bertipe xsd: anyType.

Elemen message

Elemen message menyediakan abstraksi umum untuk pesan yang diteruskan

antara client dan server. Karena Kita dapat menggunakan berbagai format definisi skema didalam dokumen WSDL, maka Kita perlu memiliki cara yang umum untuk mengidentifikasi pesan. Elemen message menyediakan abstraksi tingkat umum yang akan dirujuk dalam bagian-bagian lain dokumen WSDL.

Banyak elemen message dapat dan sering muncul dalam dokumen WSDL. Satu elemen untuk setiap pesan yang dikomunikasikan antara client dan server. Setiap pesan berisi satu atau lebih elemen parameter yang menjelaskan bagian-bagian dalam isi pesan tersebut. Contoh, sebuah bagian adalah tubuh pesan SOAP atau parameter yang ada di dalam string query. Sebuah parameter dikodekan di dalam tubuh pesan SOAP, atau seluruh tubuh pesan SOAP.

Tiap elemen part berisi atribut-atribut yang mengasosiasikan definisi type dan element di dalam elemen types. Karena bagian adalah definisi abstrak dari isi, informasi pengikatan harus diperiksa untuk menentukan makna bagian-bagian itu.

Dua atribut yang dapat muncul di dalam elemen part adalah atribut element dan type. Atribut element merujuk ke definisi elemen di dalam sebuah skema. Atribut type merujuk ke definisi tipe di dalam sebuah skema.

Karena layanan Web Calculator berisi dua metode, masing-masing dengan pesan permintaan dan jawaban, serta sebuah pesan yang telah didefinisikan, dokumen WSDL akan mengandung lima elemen message:

```
<?xml version-"1.0"encoding--"utf-8"?>
<definitionstargetNaespace-
"http://somedomain/Calculator/wsdl"xmlns:tns-
"http://somedomain/Calculator/wsdl"
xmlns:s-"http://somedomain/Calculator/schema"xmlns:xsd-
"http://www.w3.org/2001/XMLSchema"xmlns-"http://schemas.xml
soap.org/wsdl/">
```

```
<!- -Type definitions removed for clarity - - >
<message name="Add MsgIn">
< partname="parameters"element=="s:Add"/>
<!message>
<messagename="AddMsgOut">
<partname="parameters•element=="s:SubtractResult"/>
<!message>
<messagename="SubtractMsgln">
<partname="parameters"element=="s:Add"/>
<!message>
<messagename="SubtractMsgOut">
<partname="parameters•element=="s:SubtractResult"/>
<!message>
<messagename="CalculateFaultMsg">
<partname="fault•element=="s:CalculateFault"/>
<!message>
< !- - More definitions will go here. - - >
<!definitions>
```

Penulis telah membuat sebuah elemen message untuk pesan permintaan dan jawaban pada metode Add dan Substract. Penulis dapat saja menyebutkan satu elemen part untuk tiap parameter. Misalnya, pesan AddMsgIn dapat saja ditulis sebagai berikut:

```
<message name="Add MsgIn">
<partname="x"type=="xsd:int"/>
<partname="y"type=="xsd:int"/>
<!message>
```

Parameter x dan y disimpan di dalam bagian pesannya masing-masing. Pengikatan protokol akan banyak mempengaruhi representasi pesan-pesan itu. Dalam pembahasan tentang pengikatan akan menunjukkan tiap parameter

yang ada dalam string query HTTP sebagai bagian pesannya sendiri.

Karena setiap bagian dapat berfungsi sebagai definisi abstrak sepotong data, maka sebuah pesan dapat terdiri dari berbagai potongan data dari berbagai sumber. Meskipun tidak dianjurkan, Kita dapat menjelaskan sebuah pesan yang didalamnya sebagian parameter dikodekan dalam tubuh SOAP dan sebagian lagi dikodekan di dalam string query.

Elemen portType

Elemen portType berisi serangkaian operasi abstrak yang menerangkan tipe-tipe surat yang dapat terjadi antara client dan server. Untuk layanan Web bergaya RPC, portType dapat dianggap sebagai definisi antar muka yang setiap metodenya dapat didefinisikan sebagai suatu operasi.

Tipe port terdiri dari serangkaian elemen operation yang mendefinisikan tindakan tertentu. Elemen operation terdiri dari pesan yang didefinisikan di dalam dokumen WSDL. Dokumen WSDL mendefinisikan empat tipe operasi yang dikenal sebagai tipe operasi.

1. **Permintaan-jawaban**. Komunikasi bergaya RPC di mana client melakukan permintaan dan server mengirimkan jawabannya.
2. **Satu arah**. Komunikasi bergaya dokumen di mana client mengirimkan pesan, tapi tidak menerima jawaban dari server yang menunjukkan hasil dari pesan yang diproses.
3. **Solicit-respons**. Kebalikan dari operasi permintaan-jawaban. Server mengirimkan permintaan dan client mengirimkan jawabannya.
4. **Notiflkasi**. Kebalikan dari operasi satu arah. Server mengirimkan komunikasi bergaya dokumen ke client.

Sebuah operasi terdiri dari subset input, output, dan elemen fault. Tipe dan pengurutan elemen didalam operasi menentukan tipe operasinya. Misalnya, satu arah mendefinisikan pesan input, dan permintaan jawaban mendefinisikan pesan input dan output. Namun, tipe operasi solicit-repons dan notifikasi adalah kebalikan masing-masing permintaan-jawaban dan satu-arah. Operasi solicitrespons membuat daftar pesan output terlebih dulu kemudian pesan input. Operasi notifikasi berisi pesan output daripada pesan input.

Tabel 5-1 berisi daftar tipe dan urutan pesan untuk setiap tipe operasi.

Tabel 5-1 Urutan Pesan pada Tipe Operasi

Tipe Operasi	***input***	***output***	***fault***
Permintaan-jawaban	1	2	3*
Satu-arah	1		
Solicit-respons	2	1	3*
Notifikasi			
*) Pesan cacat adalah opsional. Semua pesan yang cacat dapat muncul dalam operasi			

Operasi yang menggunakan komunikasi dua arah secara opsional dapat menyebutkan satu atau lebih pesan yang cacat. Seperti definisi metode dalam Java pesan cacat memungkinkan Kita menyatakan tipe pengecualian yang dapat dilempar oleh aplikasi server. Namun, daftar yang cacat tidak boleh menyertakan kesalahan yang dinyatakan oleh protokol transpor yang mendasarinya. Misalnya, tidak perlu menerangkan kesalahan HTTP 500 di

dalam dokumen WSDL. Nama-nama elemen input, ouput, dan fault memiliki nilai default jika tidak disebutkan. Untuk tipe operasi satu arah dan dotifikasi, nama default adalah nama elemen operation yang menampungnya. Untuk tipe operasi permintaan jawaban, nama elemen input dan output adalah default untuk nama operasi dengan Request atau Response yang dilekatkan keujungnya. Untuk solicit-respon, nama elemen output adalah default untuk nama operasi dengan Solicit atau Response yang dilekatkan ke ujungnya.

Karena banyak elemen fault dapat didefinisikan di dalam operasi, maka tidak ada nama default untuk elemen fault. Oleh karena itu, setiap elemen fault harus memakai nama yang unik di dalam elemen operasi induknya.

Berikut ini definisi portType untuk layanan Web Calculator.

```
<?xml version-"1.0"encoding--"utf-8"?>
<definitionstargetNamespace-"http://somedomain/Calculator/wsdl"
xmlns:tns-"http://somedomain/Calculator/wsdl"xmlns:xsd-
"http://www.w3.org/2001/XMLSchema"xmlns-"http://schemas.xml
soap.org/wsdl/">
<! - - Type and message definitions removed for clarity - - >
<portTypename-"CalculatorPortType">
<operationname-"Add">
        <inputmessage-"tns:AddMsgln"/>
        <outputmessage-"tns:AddMsgOut"/>
        <faultmessage-"tns:CalculateFaultMsg"name--"Calculate Fault"/>
</operation>
<operationname-"Subtract">
        <inputmessage-"tns:SubtractMsgln"/>
        <outputmessage-"tns:SubtractMsgOut"/)
        <faultmessage-"tns:CalculateFaultMsg"name--"CalculateFault"/>
</operation>
</portType>
<!- - More definitions will go here. - - >
```

```
<!definitions>
```

Snippet sebelumnya pada dokumen WSDL Calculator mendefinisikan portType yang diberi nama CalculatorPortType. Di dalamnya, terdapat dua operasi permintaan-jawaban, Add dan Substract. Karena kedua operasi itu termasuk tipe permintaan-jawaban, maka keduanya mendefinisikan pesan input dan output. Keduanya juga berisi elemen fault dengan nama CalculateFault.

Operasi bergaya-RPC dapat saja menggunakan atribut parameterOrder untuk menyebutkan urutan parameter yang dikehendaki. Atribut ini bertipe nmTokens dan berisi daftar nama-nama parameter. Atribut ini jarang digunakan karena SOAP menyebutkan cara yang jelas untuk membuat serialisasi parameter dan nama serta urutan parameter dapat dijelaskan XML Schema.

Beberapa layanan yang dijelaskan dengan menggunakan WSDL dapat mendukung metode-metode overload, yaitu metode-metode yang namanya sama tapi menerima serangkaian parameter yang berlainan. Oleh karena itu, di dalam definisi portType, beberapa elemen operation dapat memiliki nama yang sama tapi menyebutkan pesan yang berbeda. Dalam kasus ini, elemen operation harus diidentifikasi oleh kombinasi antara nama operasi dengan nama elemen input, output, dan fault. Akibatnya, nama default untuk elemen input, output, dan fault tidak memastikan bahwa elemen-elemen operation yang namanya sama dapat diidentifikasi secara unik.

Elemen binding

Elemen binding berisi definisi pengikatan untuk mengikat protokol seperti

SOAP ke bindingType tertentu. Definisi binding menyebutkan pesan detail format pesan dan detail protokol. Misalnya, informasi pengikatan menetapkan apakah Kita dapat mengakses sebuah syarat portType dengan gaya seperti RPC.

Definisi binding juga menunjukkan jumlah komunikasi jaringan yang dibutuhkan untuk melakukan tindakan tertentu. Misalnya, panggilan SOAP RPC melalui HTTP dapat melibatkan pertukaran informasi, tapi panggilan yang sama melalui SMTP akan melibatkan dua pertukaran komunikasi SMTP yang berbeda.

Pengikatan dilakukan dengan menggunakan elemen perluasan. Setiap protokol memiliki sejumlah elemen perluasannya sendiri untuk menyebutkan detail perilaku serta format pesan. Untuk kontrol tertentu, elemen perluasan sering digunakan melengkapi tindakan di dalam sebuah operasi dan operasi itu sendiri dengan informasi pengikatan protokol. Kadang-kadang, elemen perluasan digunakan dalam tingkat portType sendiri.

Dokumen WSDL berikut ini menunjukkan pengikatan untuk layanan Calculator Web. Selain itu, terdapat placeholder untuk elemen perluasan yang menunjukkan tempat elemen tersebut dapat ditaruh dalam hal penyimpanan elemen binding. Dalam bab ini juga, akan membahas elemen perluasan yang didefinisikan oleh spesifikasi WSDL.

```
<?xml version-"1.0"encoding--"utf-8"?>
<definitionsxmlns:ext-"http://somedomain/MyBindingExt"xmlns-
"http://sch emas.xml soap.org/wsdl/">
<! -- Type,message,and port type definitions removed for clarity- - >
<!- - All attributes also removed for clarity - - >
<bindingname-"CalculatorBinding"type--"tns:CalculatorPortType>
<ext:SomeExtElement/>
```

```
<operationname-"Add">
<ext:SomeExtElement/>
<input>
<ext:SomeExtElement/>
<!input>
<output>
<ext:SomeExtElement/>
<!output>
<fault>
<ext:SomeExtElement/>
<!fault>
<!operation>
<operationname-"Subtract">
<ext:SomeExtElement/>
<input>
<ext:SomeExtElement/>
<!input>
<output>
<ext:SomeExtElement/>
<!output>
<fault>
<ext:SomeExtElement/>
</fault>
</operation>
<!binding>
< !- - More definitions will go here . - - >
<!definitions>
```

Elemen binding diasosiasikan dengan elemen portType melalui atribut type. Dalam dokumen WSDL terdahulu, Penulis mengasosiasikan pengikatan bernama CalculatorBinding dengan tipe port CalculatorPortType. Di dalam elemen pengikatan itu, Penulis membuat dua elemen operation untuk mengorelasikan dengan elemen-elemen yang didefinisikan di dalam elemen

portType.

Tiap operasi elemen harus memiliki elemen-elemen input, output, dan fault yang didefinisikan dalam elemen portType. Selain itu, nama-nama elemen operation serta elemen-elemen anak input, output, dan fault harus persis sesuai dengan nama pasangannya yang didefinisikan dalam elemen portType terkait.

Elemen service

Layanan adalah sekelompok port-port terkait dan didefinisikan oleh elemen service. Port adalah sebuah endpoint untuk layanan Web yang dirujuk oleh alamat tertentu. Port yang didefinisikan dalam layanan tertentu merupakan ortogonal. Misalnya, output untuk suatu port tidak dapat berfungsi sebagai input bagi port lain.

Berikut ini ringkasan definisi layanan untuk layanan Web Calculator. Dokumen tersebut berisi placeholder untuk elemen perluasan yang akan menunjukkan di mana elemen tersebut dapat ditaruh dalam hal penyimpanan elemen service.

```
<?xml version="l.O"encoding=="utf-8"?>
<definitions
xmlns:ext="http://somedomain/MyBindingExt"xmlns="http://schemas.xml
soap.org/wsdl/">
<! ··Type, message, port type. and binding definitions removed for
clarity>
<!··All attri butes also removed for clarity>
<servicename="CalculatorService">
<ext:SomeExtElement/>
<portname="CalculatorPort"binding=="tns:CalculatorBinding">
<ext:SomeExtElement/>
<!port>
```

```
</service>
<!··More definitions will go here.··>
</definitions>
```

Elemen sevice digunakan untuk mengelompokkan serangkaian port-port yang terkait. Dokumen WSDL sebelumnya mendefinisikan layanan bernama CalculatorService. Dokumen itu berisi satu port bernama CalculatorPort. Calculator Port diasosiasikan dengan elemen pengikat CalculatorBinding.

Sebuah port berisi elemen perluasan yang menyediakan alamat tempat menyimpannya. Jika harus menyebutkan lebih dari satu alamat, Kita harus membuat satu port untuk tiap alamat. Jika Kita mendefinisikan banyak port yang tipenya sama dan mungkin alamatnya berlainan di dalam satu layanan Web, maka semuanya harus dianggap sebagai alternatif.

Port-port tersebut harus menyediakan perilaku yang sama, tapi lewat protokol transpor yang berlainan. Client dapat mengulanginya melalui berbagai port untuk mencari pengikatan yang cocok dengan protokol dan portType yang ada.

Elemen Perluasan

Elemen perluasan digunakan untuk menetapkan teknologi tertentu. Misalnya, Kita dapat menggunakan elemen perluasan untuk menetapkan bahasa skem yang digunakan di dalam elemen types.

Skema untuk elemen perluasan harus didefinisikan dalam namespace berbeda, bukannya dalam WSDL. Definisi elemen sendiri dapat berisi atribut wsdl: required yang disebutkan dalam nilai Boolean. Jika atribut required diatur true dalam definisi elemen, pengikatan yang memjuk ke serangkaian elemen perluasan tertentu harus menyertakan elemen tersebut.

Seringkali, elemen perluasan digunakan untuk menetapkan informasi pengikatan. Spesifikasi WSDL mendefinisikan serangkaian elemen perluasan untuk mengikatkan ke SOAP, HTTP GET, HTTP POST, dan MIME. Namun, spesifikasi itu hanya mendefinisikan pengikatan untuk dua dari empat tipe operasi, satu arah dan permintaan-jawaban. Mari kita lihat ketiga pengikatan yang didukung oleh platform .NET: SOAP, HTTP GET, dan HTTP POST.

Perluasan SOAP

Perluasan SOAP menyediakan serangkaian elemen untuk mengikat tipe port ke pesan SOAP yang dikirim lewat protokol transpor tertentu. Misalnya, elemen perluasan SOAP digunakan untuk menunjukkan di mana tiap-tiap bagian disimpan di dalam pesan SOAP. Elemen-elemen ini juga digunakan untuk menunjukkan protokol transpor yang digunakan untuk mengirim pesan SOAP.

Elemen perluasan SOAP disimpan di dalam namespace http://schemas.xml soap.org/wsdl/soap/. Konvensi yang Penulis pakai adalah untuk mengaitkan referensi ke namespace dengan soap: moniker.

Mengikat Elemen binding

Elemen perluasan yang ditambahkan ke elemen binding menyediakan informasi tentang bagaimana parameter-parameter dibuat kodenya di dalam pesan SOAP. Elemen-elemen perluasan ditambahkan ke pesan-pesan bind, operation, input, output, dan fault. Elemen-elemen ini menyediakan informasi tentang protokol transpor yang digunakan untuk mengirim pesan SOAP dan bagaimana data itu dikodekan di dalam amplop SOAP.

Tujuan utama soap: binding adalah untuk memberi isyarat bahwa pengikatan SOAP diterapkan pada definisi pengikatan tertentu. Oleh karena itu, semua elemen pengikatan yang berisi pengikatan khusus SOAP harus mengandung elemen soap:binding. Elemen soap:binding juga dapat digunakan untuk menyebutkan style pesan dan protokol transpor yang akan digunakan untuk mengirimkan pesan SOAP. Berikut ini adalah bagian dari layanan Web Calculator dokumen WSDL yang menunjukkan penggunaan elemen soap: binding.

```
<?xml version="l.O"encoding=="utf-8"?>
<definitionstargetNamespace="http://somedomain/Calculator/wsdl"xmlns:tn
s="http://somedomain/Calculator/wsdl"
xmlns:soap="http://schemas.xml
soap.org/wsdl/soap/"xmlns="http://schemas.xml soap.org/wsdl/">
<! - - Type.message.and port type definitions removed for clarity- - >
<bindingname="CalculatorBinding"type=="tns:CalculatorPortType">
<soap:binding style="document"
transport-"http://schemas.xml soap.org/soap/http"/>
<! - - Operationel ements removed for clarity- - >
<! binding>
<! - - More definitions will go here. - - >
<!definitions>
```

Elemen soap: binding dapat berisi atribut transport untuk menyebutkan sebuah transpor. Atribut transport harus berisi URL yang secara unik mengidentifikasi transpor tersebut. Satu-satunya URL yang didefinisikan dalam spesifikasi itu adalah untuk transpor HTTP. http://schemas.xmlsoap.org/soap/http. Karena diterapkan ke semua definisi pengikatan, elemen soap:binding akan berlaku untuk semua operasi yang dirujuk oleh definisi pengikatan.

Gaya pesan ditunjukkan oleh atribut style. Nilainya bisa rpc atau document. Jika gaya diseting rpc, maka tiap bagian di dalam operasi akan menghadirkan suatu parameter. Parameter-parameter itu harus dibuat kodenya dalam tubuh pesan SOAP dengan cara yang seperti-struktur sebagaimana yang diperintahkan oleh spesifikasi SOAP. Nama elemen operation harus cocok dengan nama elemen yang mengandung parameter dalam pesan SOAP. Jika style diatur document, bagian-bagian pesan akan langsung tampil di dalam badan pesan SOAP.

Seperti yang nanti akan Kita lihat, style pesan juga dapat diatur pada tingkat operasi. Karena atribut style yang didefinisikan pada tingkat operasi merupakan preseden (pendahuluan), mengatur atribut sty le di dalam elemen soap:binding tidak menentukan style pesan; hanya mengatur nilai default. Jika atribut style tidak diatur, nilai defaultnya adalah document.

Elemen soap:binding menyediakan informasi pengikatan untuk seluruh operasi. Kita dapat memakainya untuk menyebutkan style dokumen serta nilai header HTTP pada SOAPAction untuk pengikatan HTTP. Berikut ini adalah bagian dari layanan Web Calculator dokumen WSDL yang menunjukkan penggunaan elemen soap:binding .

```
<?xml version="l.D• encoding=="utf-8"?>d
<definitionstargetNamespace-
"http://somedomain/Calculator/wsdl"xmlns:tns="http://somedomain/Calcula
tor/wsdl"
xmlns:soap="http://schemas.xml soap.org/wsdl/soap/"(
xmlns="http://schemas.xml soap.org/wsdl/">r
<! - - Type.message.and port type definitions removed for clarity- - >
<bindingname="CalculatorBinding"type=-"tns:CalculatorPortType">
<soap:bindingstyle="document"
transport="http://schemas.xml soap.org/soap/http"/>
```

```
<operationname="Add">
<soap:operationsoapAction=http://somedomain/Calculator/Add"/>
<input>
<soap:bodyuse="literal"/>
</input>
<output>
<soap:bodyuse-"literal"/>
<!output>
<fault>
<soap:faultname-"CalculateFault"use=="literal"/>
<!fault>
<!operation>
<operationname="Subtract">
<soap:operationsoapAction-http://somedomain/Calculator/Subtract"/>
<input>
<soap:bodyuse="literal"/>
<!input>
<output>
<soap:bodyuse-"literal"/>
</output>
<fault>
<soap:faultname="CalculateFault"use=="literal"/>
<!fault>
<!operation>
<!binding>
< !- - More definitions will go here. - - >
<!definitions>
```

Seperti yang telah disebutkan, atribut style dapat diatur dengan rpc atau document untuk menunjukkan style pada pesan.

Atribut soapAction menyebutkan nilai untuk header SOAPAction. Atribut soap Action dibutuhkan jika HTTP adalah protokol transpor. Nilainya dapat

kosong jika URL yang diminta HTTP sudah menjelaskan maksud dari pesan tersebut. Client harus meneruskan nilai pada atribut soapAction tanpa perubahan ketika mengirimkan pesan ke layanan Web. Jika protokolnya bukan HTTP, atribut soapAction dapat dihapus.

Elemen soap:body menyebutkan bagaimana bagian-bagian pesan dikodekan didalam badan pesan SOAP. Elemen ini digunakan untuk menyebutkan bagian mana dalam pesan yang muncul di dalam badan pesan SOAP. Elemen ini juga dapat digunakan untuk mendeklarasikan tipe pengodean yang digunakan untuk membuat serialisasi bagian-bagian di dalam badan pesan.

Jika ingin, Kita juga dapat menyebutkan daftar bagian yang dapat ditemukan di dalam badan pesan SOAP. Atribut parts dapat berisi daftar tKita yang diberi nama, di mana tiap tKita adalah nama bagian yang disimpan di dalam tubuh SOAP. Jika atribut parts tidak disebutkan, semua bagian yang didefinisikan oleh pesan tersebut akan dianggap sebagai bagian dari tubuh SOAP. Sebuah contoh penggunaan atribut parts adalah jika pesan HTIP mengandung pesan multipart yang menyertakan pesan SOAP disamping melekatkan MIME. Satu bagian pesan yang tidak dimasukkan dalam tubuh SOAP adalah yang dilekatkan.

Kadangkala, sebuah skema saja tidak cukup mewakili cara pembuatan serialisasi data. Misalnya, pengodean SOAP mendefinisikan banyak cara, sehingga serialisasi array dapat dilakukan: seluruh array, sebagian array, sekelompok array.

Atribut use adalah wajib dan akan berupa literal maupun encoded. ilailiteral artinya bagian-bagian di dalam tubuh SOAP harus sesuai dengan skemanya. Bagian di dalam definisi pesan harus merujuk ke skema dengan menggunakan

atribut type atau elemen.

Jika bagian-bagian di dalam tubuh pesan haru dibuat erialisasinya dengan menggun akan metode encoding, nilai untuk atribut use harus encoded. Setiap bagian pesan yang dibuat kodenya di dalam tubuh SOAP harus merujuk tipe abstrak dengan menggunakan atribut type. Misalnya, bagian yang ada dalam array SOAP akan merujuk tipe Array SOAP. Jika style kode mendukung variasi cara data dikodekan (seperti tipe Array SOAP), layanan itu pasti mendukung semua variasi ini.

Jika bagian pesan didasarkan pada definisi tipe abstrak daripada format konkret yang dinyatakan oleh definisi skema, style kode harus menjadi rujukan. Style kode dinyatakan oleh atribut wsdl:encodingStyle dan dapat mengandung daftar URL yang dipisahkan oleh spasi (sama dengan atribut encodingStyle yang didefinisikan oleh SOAP).

Jika atribut encodingStyle disebutkan untuk bagian pesan dan atribut use diatur literal, encodingStyle akan berfungsi sebagai petunjuk cara mengodekan data tersebut. Ini akan berguna jika Kita hanya menerima satu variasi datatype tertentu yang dikodkan SOAP. Misalnya, layanan Web hanya akan menerima array SOAP yang semuanya dibuat serialisasi.

Mengikat Elemen service

Satu-satunya elemen perluasan SOAP yang dinyatakan di dalam elemen service adalah soap:address. Elemen ini terdapat di dalam definisi port dan digunakan untuk menyebutkan URL di ujung atau port tempat layanan Web dapat dicapai.

```
<?xml version-"1.0" encoding--"utf-8"?>
```

```
<definitionstargetNamespace-
"http://somedomain/Calculator/wsdl"xmlns:tns-
"http://somedomain/Calculator/wsdl"
xmlns:soap-"http://schemas.xml soap.org/wsdl/soap/"xmlns-
"http://schemas.xml soap.org/wsdl/">
<! - - Type,message,port type.and binding definitions removed for
clarity- - >
<servicename-"CalculatorService">
<portname-"CalculatorPort"binding--"tns:CalculatorBinding">
<soap:address location-"http://somedomain/Calculator"!>
<!port>
<!service>
<!definitions>
```

Definisi layanan menyatakan bahwa layanan Web Calculator dapat dicapai di http://somedomain/Calculator. Jika alamat itu tidak dapat dinyatakan dengan URL, elemen soap:address dapat diganti elemen custom address yang menyatakan lokasinya.

Perluasan HTTP GET/POST

Kadangkala, kita ingin memanggil layanan Web dengan meneruskan parameter sebagai pasangan nama atau nilai dengan menggunakan mekanisme yang sama seperti pemakaian HTML . Parameter dapat diteruskan lewat query string atau form POST. Ini terkadang mempermudah client memanggil layanan Web tanpa harus membuat pesan SOAP yang lengkap.

Peluasan HTTP GET/POST menyediakan serangkaian elemen untuk mengikat tipe port ke protokol SOAP. Misalnya, elemen perluasan SOAP digunakan untuk menunjukkan tempat tiap-tiap bagiannya disimpan di dalam pesan SOAP. Elemen-elemen ini juga digunakan untuk menunjukkan protokol

transpor yang digunakan untuk mengirimkan pesan SOAP.

Dalam bagian ini, Penulis membuat dua pengikatan tambahan untuk layanan Web Calculator, satu untuk HTTP GET dan satu untuk HTTP POST. Penulis mempunyai kebebasan dalam menyebutkan pengikatannya, maka Penulis akan membuat pengikatan yang hampir sama dengan perilaku layanan Web yang dibuat pada platform .NET. Terutama, layanan Web yang dibuat dengan menggunakan platform .NET memakai pasangan nama/nilai yang dikodekan URL dan disertakan pada query string atau di POST kan di dalam tubuh pesan permintaan HTTP. Jika hasilnya muncul, maka akan diteruskan sebagai dokumen XML sederhana di dalam tubuh pesan respons HTTP.

Sebelum Penulis menspesifikasi informasi pengikatan, Penulis perlu membuat tambahan elemen pernyataan untuk menampung nilai parameter yang dihasilkan. Karena tiap parameter harus ditampilkan sebagai bagian-bagian pesan yang terpisah, Penulis juga perlu membuat sejumlah definisi untuk metode Add dan Substract, tempat setiap parameter disimpan di dalam bagian pesannya sendiri. Berikut ini adalah penambahannya:

```
<?xml version-"1.0" encoding--"utf-8"?>
<definitionstargetNamespace-
"http://somedomain/Calculator/wsdl"xmlns:tns-
"http://somedomain/Calculator/wsdl"
xmlns:s-"http://somedomain/Calculator/schema"xmlns:xsd-
"http://www.w3.org/2001/XMLSchema"
xmlns:http-"http://schemas.xml soap.org/wsdl/http/"xmlns:mime-
"http://schemas.xml soap.org/wsdl/mime/"xmlns-"http://schemas.xml soap.
org/wsdl/">
<!-- Note:Previously defined type definitions omitted for clarity - - >
<types>
```

```
<schemaattributeFormDefault-"qualified"elementFormDefault--
"qualified"xmlns-"http://www.w3.org/2001/XMLSchema
targetNamespace-"http://somedomain/Calculator/schema">
< !- - Common result element for HTTP GET/POST binding- - >
<elementname-"Result"type--"int"/>
</schema>
<!types>
<!-- Messages for HTTP GET/POST-based Web service - - >
<!- - Note:Previously defined messages omitted or clarity - - >
<messagename-"AddHttpMsgln">
<partname-"x"type--"xsd:string"/>
<partname-"y•type--"xsd:string"/>
</message>
<messagename-"AddHttpMsgOut">
<partname-"result"element--"s:Result"/)
<!message>
<messagename-"SubtractHttpMsgln">
<partname-"x•element--"xsd: string"/>
<partname-"y•element--"xsd:string"/>
</message>
<messagename="SubtractHttpMsgOut">
<partname="result•element=="s:Result"/>
<!message>
< !- - More definitions will go here. - - >
<!definitions>
```

Dokumen WSDL sebelumnya mendefinisikan elemen tipe baru int bernama Result yang digunakan untuk menampung hasil dari metode Add dan Substract yang dikirim kembali ke client. Elemen ini dirujuk oleh dua pesan diluar ikatan, satu pesan untuk satu metode. Elemen ini juga mendefinisikan pesan di dalam ikatan untuk metode Add dan Subtract. Pesan di luar ikatan mendefinisikan satu bagian untuk satu parameter. Karena pasangan nama/nilai

berisi parameter bukan bertipe kuat, tipe setiap bagiannya didefinisikan sebagai string.

Elemen ektensi HTTP GET/POST disimpan di dalam namespace http: //schemas.xmlsoap.org/wsdllhttp/. Konvensi yang dipakai di bagian selanjutnya dalam bab ini adalah untuk mengasosiasikan referensi-referensi ke namespace dengan menggunakan http: moniker. Dalam beberapa skenario, pengikatan HTTP GET/POST menggunakan elemen perluasan MIME. Pengikatan tersebut didefinisikan di dalam namespace http://schemas.xmlsoap.org/wsdl/mime/ dan dirujuk dengan menggunakan mime: moniker.

Mengikat Elemen binding

Elemen perluasan yang ditambahkan ke elemen binding menyediakan informasi tentang bagaimana parameter dibuat kodenya di dalam pesan HTTP. Elemen-elemen perluasan ditambahkan ke pesan-pesan bind, operation, input, output, dan fault.

Elemen http:binding menetapkan apakah parameter-parameter itu diteruskan di dalam URL atau di dalam tubuh permintaan HTTP: Kata "kerja" pada atribut http: binding diseting GET atau POST Dokumen WSDL berikut ini menunjukkan penggunaan elemen htttp:binding di dalam definisi layanan Web Calculator.

```
<?xml version="l.O" encoding=="utf-8"?>
<definitionstargetNamespace="http://somedomain/Calculator/wsdl"xmlns:
tns="http://somedomain/Calculator/wsdl"
xmlns:http="http://schemas.xml
soap.org/wsdl/http/"xmlns="http://schemas.xml soap.org/wsdl/">
< !- - Type.message.and port type definitions removed for clarity - - >
```

```
<bindingname="CalculatorHttpGetBinding"type=="tns:CalculatorPortType"/>
<http:bindingverb-"GET"/)
<!- - Operati on elements removed for clarity - - >
<!bindig>
<!- - More definitions will go here. - - >
<!definitions>
```

Elemen http:operation menyebutkan alamat relatif untuk setiap operasi. Tiap pesan input dan output dilengkapi dengan elemen perluasan yang menunjukkan metode yang dipakai untuk mengodekan parameter yang diteruskan ke layanan Web. Tiga skenario di bawah ini untuk mengodekan parameter yang menunjukkan kode URL parameter pada string query, kode non di dalam URL dan kode URL parameter dalam tubuh POST.

Parameter dapat diteruskan ke layanan Web lewat URL yang dikodekan di dalam string query. Kode URL menyebutkan sisipan tKita? di ujung URL dan menyisipkan pasangan nama/nilai yang dipisah tKita =. Jika pasangan nama/ nilai yang disisipkan ke URL jumlahnya banyak, maka tiap pasangan harus dipisah tKita &. Misalnya, metode Add dapat dipanggil seperti kode berikut:

```
http://somedomain/Calculator/Add?x-2&y-3
```

Dalam hal ini, untuk operasi tertentu elemen input di dalam pengikatan itu akan dilengkapi dengan elemen http: urlEncoded. Berikut ini hasil dari definisi pengikatan HTTP GET untuk layanan Web Calculator.

```
<?xml version-"1.0" encoding--"utf-8"?>
```

```
<definitionstargetNamespace-
"http://somedomain/Calculator/wsdl"xmlns:tns-
"http://somedomain/Calculator/wsdl"
xmlns:http="http://schemas.xml soap.org/wsdl/http/"xmlns-
"http://schemas.xml soap.org/wsdl/">
<!- - Type.message.and port type definitions removed for clarity- - >
<bindingname-"CalculatorHttpGetBinding"type--"tns:CalculatorPortType"/>
<http:bindingverb-"GET"/)
<operationname="Add">
<http:operati on location-"/Add"/>
<input>
<http:url Encoded/ >
<!input>
<output>
<mime:mimeXmlpart="Body"/>
<!output>
<fault>
<mime:mimeXmlpart-"Fault"/>
</fault>
<!operation>
<operationname-"Subtract">
<http:operationlocation-"/Subtract"/>
<input>
<http:url Encoded/ >
<!input>
<output>
<mime:mimeXmlpart-"Body"/>
</output>
<fault>
<mime:mimeXmlpart-"Fault"/>
</fault>
</operation>
</binding>
```

```
<!- - More definitions will go here. - - >
</definitions>
```

Parameter juga dapat dikodekan di dalam URL dengan cara non . Dalam hal ini, lokasi atribut pada elemen http:operation berupa informasi tentang bagaimana parameter-parameter itu dibuat kodenya. Misalnya, parameter untuk metode Add dapat dibuat kodenya dalam URL, sebagai berikut:

```
http://somedomain/Calculator/Add/2plus3
```

Parameter 2 dan 3 dikodekan dalam informasi path URL tersebut di mana parameter-parameter itu dibatasi oleh plus. Elemen http:operation akan muncul dalam definisi pengikatan sebagai berikut:

```
<http: operationlocation-"Add/(xlplus(y)"/>
```

Setiap bagian pesan dimasukan dalam tKita kurung yang terlihat dalam posisi masing-masing di dalam URL relatif tersebut. Dalam hal ini, elemen input di dalam pengikatan itu untuk operasi tertentu akan dilengkapi dengan elemen http: urlReplacement.

Cara kedua dan terakhir pengodean parameter yang akan dibahas adalah pelekatan kode parameter URL di dalam tubuh pesan permintaan HTTP (HTTP POSD. Misalnya, parameter itu akan dibuat kodenya di dalam tubuh permintaan HTIP seperti berikut ini:

```
Add-"x-Z&y-3"
```

Dalam hal ini, elemen input dapat dijelaskan dengan menggunakan tipe MIME application/x-www-form-urlencoded. Oleh karena itu, elemen operation akan dilengkapi elemen mime:content. Berikut ini hasil dari definisi pengikatan HTIP POST untuk layanan Web Calculator.

```
<? xml version="l.O" encoding=="utf-8"?>
```

```
<definitionstargetNamespace="http://somedomain/Calculator/wsdl"xmlns:tn
s="http://somedomain/Calculator/wsdl"
xmlns:http="http://schemas.xml
soap.org/wsdl/http/"xmlns="http://schemas.xml soap.org/wsdl/">
<! - -Type.message.and port type definitions removed for clarity - - >
<bindingname="CalculatorHttpPostBinding"type=="tns:CalculatorPortType"/
>
<http:bindingverb="POST"/ >
<operationname="Add">
<http:operationlocation="/Add"/ >
<input>
<mime:contenttype="application/x-www-form-url encoded"/>
<!input>
<output>
<mime:mimeXmlpart="Body"/>
</output>
<fault>
<soap:faultname="CalculateFault"use=="literal"/>
</fault>
<operation>
<operationname="Subtract">
<http:operationlocation="/Subtract"/>
<input>
<mime:contenttype="application/x-www-form-url encoded"/>
<!input>
<output>
<mime:mimeXmlpart-"Body"/>
</output>
<fault>
<soap:faultname="CalculateFault"use="literal"/>
</fault>
<!operation>
<!bi n d i ng >
```

```
< !- - More definitions will go here. - - >
</definitions>
```

Atribut type berisi tipe MIME valid yang digunakan untuk menunjukkan tipe isi yang ada dalam tubuh pesan HTTP. Isi dari pesan HTTP tersebut juga dapat diberi label sebagai anggota keluarga tipe MIME dengan menggunakan wildcard. Berikut ini beberapa contohnya:

```
<! - - The content belongs to the MIME family of text types. - - >
<mime: contenttype="text/*"/>
<! - - Either declaration specifies all MIME types. - - >
<mime:contenttype="text/*"/>
<mime:contentl>
```

Elemen mime:content juga dapat berisi satu atribut part yang dipakai untuk menyebutkan bagian yang disimpan di dalam tubuh pesan HTTP.

Jika pesan itu memiliki tipe MIME multipart/ related, pesannya dapat berisi sekumpulan bagian yang diformat MIME. Misalnya, pesan yang terdiri dari banyak bagian dapat berisi pesan SOAP (text/xml) bersama gambar JPEG (image/ jpeg).

Pesan yang terdiri dari banyak bagian dapat diwakili di dalam definisi pengikatan dengan menggunakan elemen mime:multipartRelated. Elemen mime: multipartRelated berisi kumpulan elemen mime. parts. Setiap elemen mime.pa rt mewakili bagian yang diformat MIME di mana tipenya dinyatakan dengan menggunakan elemen mime:content.

Contoh berikut ini menunjukkan bagaimana pesan dengan banyak bagian yang berisi sebuah pesan SOAP dan satu gambar JPEG diwakili di dalam dokumen WSDL.

```
<mime:multipartRelated>
<mime:part>
```

```
<soap:bodyuse-"literal•part--"xyCoordinates"/>
<lmime:part>
<mime:part>
<mime:contenttype-"image/jpeg•part--"graph"/>
<!mime:part>
<lmime:multipartRelated>
```

Perhatikan bahwa Kita dapat menggunakan elemen soap: body untuk menunjukkan bahwa bagian MIME tertentu berisi pesan SOAP. Bagian pesan itu diasumsikan memiliki tipe MIME text/xml dan disimpan di dalam amplop SOAP valid.

Jika bagian pesan MIME mengandung XML, tapi tidak cocok dengan SOAP Kita dapat menggunakan elemen mime:mimeXml. Elemen part terkait akan mendefinisikan akar elemen XML daripada tubuh pesan SOAP. Elemen ini banyak digunakan dalam ASP.NET karena versi HTTP GET/POST pada layanan Web akan memberikan hasilnya dengan menggunakan XML yang sesuai dengan non SOAP.

Mengikat Elemen service

Satu-satunya elemen perluasan HTTP yang dinyatakan di dalam elemen service adalah http:address. Seperti juga SOAP, elemen ini pun terdapat di dalam definisi port dan digunakan untuk menyebutkan URL tempat layanan Web dapat dicapai. Berikut ini definisi layanan untuk layanan Web Calculator.

```
<?xm l versi on="l.0 " encod i ng=="ut f - 8"?>
<d efi n i ti ons ta rget Na mespace="http: //somedomai n/Cal cul ator
/wsdl " xml n s:tns= "http ://somed oma i n/C al c ul ator/wsd l "
xml n s:soa p="http://schemas.xml soa p.org/wsd l /soa p/" xml n
s="http://schemas.x m l soap.org/ws dl /">
```

```
<! --Type.message.porttype.and binding definitions removed for clarity-
->
<service name="CalculatorService">
<portname="CalculatorHttpGetPort"binding=="tns:CalculatorHttpGetBinding
">
<http:address location="http://somedomain/Calculator"/>
<!port>
<portname="CalculatorHttpPostPort"binding=="tns:CalculatorHttpPostBindi
ng">
<http::address location="http://somedomain/Calculator"/>
</port>
<!service>
<!definitions>
```

Dokumen WSDL sebelurnnya mendefinisikan dua port, satu untuk HTIP GET dan satu untuk HTIP POST. Kedua port itu dapat diperoleh di http://somedomain/Calculator.

Elemen import

Seperti dokumen XML Schema, dokumen WSDL dapat mengimpor dokumen lain. Kita dapat mencapai tingkat modular yang sama dengan dokumen XML Schema. Karena dokumen WSDL dapat cepat membesar, memecahnya ke dalam dokumen-dokumen kecil dapat mempermudah dokumen itu untuk dibaca, dipahami, dan mungkin lebih mud ah dikelola.

Cara yang biasa dipakai untuk memecah satu definisi layanan ke dalam banyak dokumen WSDL adalah dengan menaruh informasi pengik atan protokol di dalam dokumen terpisah. Dengan cara ini, Kita dapat menulis definisi-definisi antar muka sekali saja. Kemudian, mengimpornya kedalam

dokumen WSDL yang mendefinisikan protokol tertentu yang didukung oleh syarat layanan Web tertentu.

Tidak seperti XML Schema, elemen import harus berisi atribut namespace dan location. SeKitainya Penulis menggunakan dokumen WSDL untuk layanan Web Calculator dan memecahnya ke dalam tiga bagian. Penulis menaruh definisi skema di dalam Calculator.xsd, definisi antar muka dalam i_Calculator .wsdl, dan protokol dalam Calculator.wsdl. Calculator.wsdl berfungsi sebagai dokumen WSDL untuk layanan Web dengan mengimpor dua dokumen lainnya. Berikut ini adalah Calculator.wsdl:

```
<definitions xmlns-"http://schemas.xml soap.org/wsdl/">
<! - - First import the schema definitions. - - >
<importnamespace-"http://somedomain/myschema/"location-
"http://somedomain/Calculator.xsd">
<!- - Next import the port types and message definitions. - - >
<importnamespace-"http://somedomain/Calculator/"location-
"http://somedomain/i_Calculator.wsdl">
<!- - Finally provide the protocol-specific binding definitions. - - >
<!definitions>
```

Dokumentasi

Kita dapat menyisipkan dokumentasi di dalam dokumen WSDL dengan menggunakan atribut name pada elemen definitions atau elemen document. Atribut name dapat berisi penjelasan singkat tentang dokumen WSDL, dan elemen document dapat berisi teks ataupun elemen lainnya. Misalnya, Penulis dapat menggunakan elemen document untuk menyimpan metadata tentang dokumen itu.

```
<definitions xmlns="http://schemas.xml soap.org/wsdl/"
name-"The Calculator Webservice provides the results of adding and
subtracting two numbers.">
```

```
<document>
<author>ScottShort</author>
<version>l.O</version>
<!document>
<types>
<document>The following are defined usingXMLSchema.</d ocument>
<!- - Type definitions removed for clarity - - >
<!types>
<!- - Additional definitions removed for clarity - - >
<!definitions>
```

Seperti yang Kita lihat, elemen document dapat digunakan di dalam elemen bahasa WSDL apa pun.

Dokumen WSDL pada Layanan Web Calculator

Berikut ini adalah dokumen WSDL yang Penulis buat dalam keseluruhan bab ini:

```
<?xml version-"1.0 "encodin -="utf-8"?>
<definitionstargetNamespace-
"http://somedomain/Calculator/wsdl"xmlns:tns="http://somedomain/Calcula
tor/wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"xmlns:s-
"http://somedomain/Calculator/schema"
xmlns:soap-"http://schemas.xml soap.org/wsdl/soap/"

xmlns:http="http://schemas.xml
soap.org/wsdl/http/"xmlns:mime="http://schemas.xml
soap.org/wsdl/mime/"xmlns="http://schemas.xml soap.org/wsdl/">
<types>
<schemaattributeFormDefault="qualified"elementFormDefault=="qualified"x
mlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://somedomain/Calculator/schema">
```

```
< !- - Definitions for both the Add and Subtract SOAP messages - - >
<elementname="Add">
<complexType>
<a11>
<elementname="x"type=="int"/>
<elementname="y"type=="int"/>
<!all>
</complexType>
<!element>
<elementname="AddResult">
<complexType>
<a11>
<elementname="result"type=="int"/>
<!all>
</complexType>
</element>
<elementname="Subtract">
<complexType>
<a11>
<elementname="x"type=="int"/>
<elementname="y"type=="int"/>
<!all>
</complexType>
<!element>
<elementname="SubtractResult">
<complexType>
<all>
<elementname="result"type=="int"/>
<!all>
</complexType>
<!element>
< !- - Common SOAP fault detail element used by Add and Subtract - - >
<elementname="CalculateFault">
```

```
<complexType>
<all>
<elementname="x"type=="int"/>
<elementname="y"type=="int"/>
<elementname="Description"type=="string"/>
</a11>
</complexType>
<!element>
< !- - Common result element for HTTP GET/POST bindi ng - - >
<elementname="Result"type=="int"/>

<!schema>
<!types>
<!- - Messages for SOAP-based Web service - - >
<messagename-"Add Msgln">
<partname-"parameters"element--"s:Add"/>
<!message>
<messagename-"AddMsgOut">
<partname-"parameters"element--"s:SubtractResult"/>
</message>
<messagename-"SubtractMsgln">
<partname-"parameters"element--"s:Add"/>
<!message>
<messagename-"SubtractMsgOut">
<partname-"parameters"element--"s:SubtractResult"/>
</message>
<messagename-"CalculateFaultMsg">
<partname-"fault"element--"s:CalculateFault"/>
<!message>
<!- - Messages for HTTP GET/POST-based Web service- - >
<messagename-"AddHttpMsgln">
<partname-"x"type--"xsd:string"/>
<partname-"y"type--"xsd:string"/>
```

```
</message>
<me ssage na me-"AddHttpMsgOut">
<partname-"result"element--"s:Result"/>
<!message>
<messagename-"SubtractHttpMsgln">
<partname-"x"element--"xsd:string"/>
<partname-"y"element--"xsd:string"/>
<!message>
<messagename-"SubtractHttpMsgOut">
<partname-"result"element--"s:Result"/>
</message>
<portTypename-"CalculatorPortType">
<operationname-"Add">
<inputmessage-"tns:AddMsgln"/>
<outputmessage-"tns:AddMsgOut"/>
<faultmessage-"tns:CalculateFaultMsg"name--"CalculateFault"/)
</operation>
<operationname-"Subtract">
<input message-"tns:SubtractMsgln"/>
<outputmessage-"tns:SubtractMsgOut"/>
<faultmessage-"tns:CalculateFaultMsg"name--"CalculateFault"/>
<!operation>
</portType>
<!- - SOAP Binding- - >
<bindingname-"CalculatorBinding"type--"tns:CalculatorPortType">
<soap:bindingstyle-"document"
transport-"http://schemas.xml soap.org/soap/http"/>
<operationname-"Add">
<soap:operation soapAction-"http://somedomain/Calculator/Add"/>
<input>
<soap:bodyuse-"literal"/>
<!input>
<output>
```

```
<soap:bodyuse-"literal"/>
<!output>
<fault>
<soap:faultname-"CalculateFault"use--"literal"/>
</fault>
<!operation>
<operationname-"Subtract">
<soap:operationsoapAction-"http://somedomain/Calculator/Subtract"/>
<input>
<soap:bodyuse-"literal"/>
<!input>
<output>
<soap:bodyuse-"literal"/>
<!output>
<fault>
<soap:faultname-"CalculateFault"use--"literal"/>
</fault>
<!operation>
<!binding>

<!- - HTTP GET Binding - - >
<bindingname-"CalculatorHttpGetBinding"type--"tns:CalculatorPortType">
<http:bindingverb-"GET "/>
<operationname-"Add">
<http:operationlocation-"/Add"/>
<input>
<http:urlEncoded/>
<!input>
<output>
<mime:mimeXmlpart-"Body"/>
<!output>
<fault>
<mime:mimeXmlpart-"Fault"/)
```

```
<!fault>
<!operation>
<operationname-"Subtract">
<http:operationlocation-"/Subtract"/>
<input>
<http:urlEncoded/>
<!input>
<output>
<mime:mimeXmlpart-"Body"/>
<!output>
<fault>
<mime:mimeXmlpart-"Fault"/>
</fault>
<!operation>
<!binding>

<! - - HTTP POST Binding - - >
<bindingname-"CalculatorHttpPostBinding"type--"tns:CalculatorPortType">
<http:bindingverb-"POST"/>
<operationname-"Add">
<http:operationlocation-"/Add"/>
<input>
<mime:contenttype-"application/x-www-form-urlencoded"/>
<!input>
<output>
<mime:mimeXmlpart-"Body"/>
</output>
<fault>
<soap:faultname-"CalculateFault"use--"literar'/>
</fault>
<!operation>
<operationname-"Subt ract">
<http:operation location-"/Subtract"/>
```

```
<input>
<mime:contenttype-"application/x-www-form-urlencoded"/>
<!input>
<output>
<mime:mimeXml part-"Body"/>
<Ioutput>
<fault>
<soap:faultname-"CalculateFault"use--"literal"/>
</fault>
<!operation>
<!binding>
<servicename-"CalculatorService">
<portname-"CalculatorPort"binding--"tns :CalculatorBinding">
<soap:addresslocation-"http://somedomain/Calculator"/>
<!port>
<portname-"CalculatorHttpGetPort "binding--
"tns:CalculatorHttpGetBinding">
<http:addresslocation-"http://somedomain/Calculator"/>
<!port>
<portname-"CalculatorHttpPostPort"binding-
"tns:CalculatorHttpPostBinding">
<http:addresslocation-"http://somedomain/Calculator"/>
</port>
</service>
<!definitions>
```

C. PENUTUP

Ringkasan

WSDL menyediakan cara yang luwes dan dapat diperluas dalam membuat dokumen layanan jaringan. Dokumen WSDL terdiri dari lima elemen di bawah akar elemen definitions: types, message, portType, binding, dan seroice. Elemen-elemen ini digunakan untuk mendefinisikan layanan Web

melalui serangkaian asosiasi.

Elemen types berisi skema definitions untuk pertukaran data antara client dan server. Bahasa skema default adalah XML Schema. Namun, Kita dapat menyebutkan bahasa skema lainnya dengan menggunakan elemen-elemen perluasan.

Elemen message mengidentifikasi pesan tertentu yang dipertukarkan antara client dan server. Sebuah pesan terdiri dari satu atau lebih bagian. Tiap bagian diwakili oleh elemen part dan dapat erujuk ke definisi elemen atau tipe yang didefinisikan di dalam elemen types.

Elemen portTypes berisi satu atau lebih elemen operation. Kita dapat menya-makan operasi seperti antar muka, yaitu sebuah kontrak tentang bagaimana client dan server berinteraksi satu sama lain untuk melakukan sebuah tindakan. Sebuah operasi dapat berupa salah satu dari empat tipe berikut: permintaan respons, solicit-respons, satu-arah, atau pemberitahuan.

Elemen binding digunakan untuk mengasosiasikan tipe port dengan protokol tertentu. Hal ini dilakukan lewat elemen perluasan. Elemen perluasan adalah elemen yang didefinisikan di luar namespace WSDL. Spesifikasi WSDL mendefinisikan serangkaian elemen perluasan untuk menyebutkan informasi pengikatan: SOAP, HTTP GET/POST, dan MIME. Karena teknologi tertentu seperti SOAP dan HTTP diwakili oleh elemen perlu asan, WSDL dapat digunakan untuk menje laskan hampir semua layanan.

Elemen seroice berisi satu atau lebih elemen port. Elemen port digunakan untuk mendefinisikan alamat tempat layanan Web yang mendukung pengikatan tertentu dapat dicapai.

Pertanyaan

1. Sebutkan dan jelaskan elemen anak dari dokumen WSDL?
2. Jelaskan data yang menggunakan elemen, bukannya atribut?
3. Jelaskan hanya data yang terkait dengan pesan dan tidak spesifik digunakan untuk kode koneksi?
4. Praktikkan Dokumen WSDL pada Layanan Web Calculator.

DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.

10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 6

Membangun Layanan Web yang Aman

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 6 akan di bahas cara Membangun Layanan Web yang aman, yang mencakup diantaranya: Pengantar ke Threat Modeling, Brainstorming Ancaman, Memakai Model Ancaman STRIDE, Memilih Teknik untuk Menangani Ancaman, Contoh Layanan Web,Teknologi Keamanan Layanan Web, Otentikasi Layanan Web, Otentikasi Anonim, Otentikasi Dasar, Otentikasi Terekstrak, Otentikasi Windows, Otentikasi Berbasis Sertifikat, Otentikasi Berbasis Form, Otentikasi ,NET Passport, Otorisasi Layanan Web, Privasi dan Integritas Layanan Web, Teknologi Keamanan dalam .NET Framework, Teknologi Keamanan Layanan Web Masa Depan, Kesalahan Umum dalam Keamanan.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan Threat Modeling, Brainstorming Ancaman, Memakai Model Ancaman STRIDE, Memilih Teknik untuk Menangani Ancaman, Contoh Layanan Web,Teknologi Keamanan Layanan Web, Otentikasi Layanan Web, Otentikasi Anonim, Otentikasi Dasar, Otentikasi Terekstrak, Otentikasi Windows, Otentikasi Berbasis Sertifikat, Otentikasi Berbasis Form, Otentikasi ,NET Passport, Otorisasi Layanan Web, Privasi dan Integritas Layanan Web, Teknologi Keamanan dalam .NET Framework, Teknologi

Keamanan Layanan Web Masa Depan, Kesalahan Umum dalam Keamanan.

B. PENYAJIAN

Membangun Layanan Web yang Aman

Menurut sifatnya, banyak layanan Web berada dalam lingkungan yang paling rawan, yaitu internet. Oleh karena itu, layanan Web Kita harus memakai teknologi pengamanan yang sesuai. Kita dapat memakai suatu pendekatan yang dikenal sebagai threat-modeling (pemodelan-aman) untuk menentukan bagian mana dari aplikasi yang paling berisiko, serta tool dan teknik apa yang harus dipakai untuk menguranginya.

Di dalam bab ini, akan dijelaskan mengenai threat-modeling secara rinci dan cara menerapkannya untuk mengamankan layanan Web. Kemudian, akan dibicarakan teknologi keamanan yang disajikan oleh Microsoft Internet Information Services (IIS). Tidak ketinggalan, akan dibahas juga Teknologi Keamanan berbasis-XML yang penting, seperti XML Signature dan XML Encryption serta bagaimana Microsoft .NET Framework mendukungnya. Terakhir, kita akan mengamati pengamanan yang biasa dilakukan orang ketika membangun layanan Web dan bagaimana dapat menghindari kesalahan yang mengakibatkan layanan Web tidak aman.

Pengantar ke Threat-Modeling

Kita mungkin ingin segera membahas cara membangun sistem yang aman. Namun Penulisngnya, petama-tama kita harus membahas tahap desain karena ketika perangkat lunak dirancang secara acak, keamanan sering menjadi korban kekacauan itu. Salah satu cara untuk menyediakan struktur

sepanjang tahap desain adalah dengan menciptakan sebuah model ancaman.

Prinsip di belakang threat-modeling adalah bahwa Kita tidak dapat mengamankan sistem, kecuali jika memahami ancaman kepada sistem atau aplikasi itu. Threat modeling itu sederhana dan menyenangkan. Threat modeling dapat menjadi dasar spesifikasi desain dalam hal keamanan! Layanan yang dibangun dengan bantuan suatu model ancaman cenderung mempunyai fitur keamanan yang lebih baik sehingga menjadi sistem yang lebih aman.

Kita perlu meluangkan waktu melakukan threat-modeling karena lebih murah menemukan kesalahan desain keamanan pada tahap desain dan memperbaikinya sebelum tahap pengodean dimulai. Proses threat-modeling mempunyai tiga fase utama:

1. Brainstorming ancaman
2. Memilih teknik untuk mengurangi ancaman
3. Memilih teknologi yang tepat untuk menerapkan teknik.

Brainstorming Ancaman

Pertemuan brainstorming membutuhkan dua atau tiga jam bagi tim pengembang untuk membahas area sistem yang terancam. Sepanjang pertemuan, mintalah seseorang mempersiapkan arsitektur yang diusulkan di papan tulis. Yakinkan bahwa diagram itu mencakup semua aspek kritis layanan Web yang meliputi:

1. Teknologi penyimpanan data (Penyimpanan File, database SQL, file-XML dan registry)

2. Teknik komunikasi interproses (termasuk RPC, .NET Remoting, dan soket)
3. Teknik-teknik user-input (argumen SOAP dan pesan-HTTP)
4. Data Nonpersisten (seperti data on-the-wire)

Selanjutnya Kita akan melihat bagaimana masing-masing komponen ini dapat disatukan. Satu metode adalah memakai teknik threat-modeling STRIDE.

Memakai Model-Ancaman STRIDE

Sebelum membangun sistem Kita, seringkali berguna untuk menanyakan hal- hal berikut:

1. Bagaimana caranya seorang penyerang membajak kereta-belanja online?
2. Apa dampaknya jika seorang penyerang menghalangi para pengguna yang sah mengakses layanan?
3. Bagaimana caranya seorang penyerang melihat atau mengubah data yang berjalan dari layanan ke konsumen?

Salah satu cara untuk memastikan bahwa Kita telah menanyakan semua pertanyaan yang penting adalah memakai kategori ancaman. Dalam hal ini, kita akan memakai model ancaman STRIDE. STRIDE adalah suatu singkatan yang diperoleh dari enam kategori ancaman berikut:

1. **Spoofing identity** (menipu identitas). Penipuan identitas sering berarti secara tidak sah mengakses dan kemudian memakai informasi otentikasi pengguna lain, seperti username dan password.

2. **Tampering with data** (merusak data). Perusakan data di antaranya adalah memodifikasi data secara semena-mena. Contohnya meliputi membuat perubahan tidak sah pada data persisten, seperti yang tersimpan dalam suatu database dan mengubah data ketika mengalir antara dua komputer melalui jaringan terbuka seperti Internet.
3. **Repudiation** (penyangkalan). Repudiasi terjadi ketika pengguna menyangkal melakukan sebuah tindakan tanpa pihak lain dapat membuktikan sebaliknya. Sebagai contoh, seorang pengguna melakukan tindakan terlarang di dalam suatu sistem yang tidak memiliki kemampuan melacak tindakan itu. Nonrepudiasi adalah kemampuan suatu sistem untuk meyerang balik penyangkalan. Sebagai contoh, jika seorang pengguna membeli sebuah item, ia mungkin harus menKitatangani tKita terima. Penjual kemudian dapat memakai tKita terima yang ditKitatangani sebagai bukti bahwa pengguna menerima paket item itu. Seperti dapat Kita bayangkan, nonrepudiasi penting terutama untuk e-commerce. Cara paling sederhana untuk memikirkan repudiasi adalah dengan mengucapkan kata-kata “bukan Penulis!"
4. **Information disclosure** (pengungkapan informasi). Ancaman penyingkapan informasi di antaranya memunculkan informasi ke individu atau siapapun yang tidak diharapkan untuk mempunyai akses ke informasi tersebut. Sebagai contoh, seorang pengguna menjadi mampu membaca sebuah file padahal dia tidak memiliki akses, atau suatu kemampuan pengganggu membaca data di dalam pemindahan antara dua komputer.

5. Denial of Service (DoS) (penolakan layanan). Serangan DoS menyebabkan penolakan pemberian layanan kepada pengguna yang sah. Sebagai contoh dengan membuat sebuah Server Web untuk sementara tak tersedia atau tak dapat dipakai, atau Kita harus melindungi sistem dari ancaman DoS tertentu untuk meningkatkan ketersediaan sistem dan keKitalan.
6. **Elevation of privilege** (menambah hak istimewa). Dalam ancaman jenis ini seorang pengguna yang tidak memiliki hak khusus mendapatkan akses istimewa dan dengan demikian mempunyai akses cukup untuk berkompromi dengan sistem atau menghancurkan keseluruhan sistem. Ancaman peningkatan status ini meliputi situasi di mana suatu penyerang secara efektif menembus semua pertahanan sistem dan telah menjadi bagian dari sistem yang dipercayai benar-benar situasi yang berbahaya.

Jika Kita melihat ke belakang pada ketiga pertanyaan contoh sebelumnya. Kita akan memperhatikan bahwa yang menjadi perhatian pertanyaan pertama adalah suatu ancaman perusakan data (T), yang kedua adalah ancaman DoS (D) dan fokus pertanyaan ketiga adalah suatu ancaman penyingkapan informasi dan suatu perusakan data (I dan T).

Sampai saat ini, cara paling sederhana untuk menerapkan model STRIDE pada aplikasi Kita adalah dengan mempertimbangkan bagaimana tiap-tiap ancaman akan mempengaruhi setiap komponen-solusi, dan setiap koneksi komponen atau hubungan dengan komponen solusi yang lain. Prinsipnya, Kita memperhatikan masing-masing bagian dari aplikasi dan menentukan apakah ada ancaman S, T, R, I, D, atau E terhadap komponen atau proses

itu. Kebanyakan bagian akan mempunyai banyak ancaman dan Kita harus yakin telah mencatat semuanya.

Memilih Teknik untuk Menangani Ancaman

Langkah berikutnya adalah menentukan teknik penanganan yang sesuai. Tabel 6-1 menampilkan garis besar beberapa teknik dan teknologi untuk menangani ancaman di dalam kategori STRIDE.

Tabel 6-1 Teknik-teknik Penanganan Ancaman

Tipe Ancaman	**Teknik Menangani**
Spoofing identity	Buktikan keaslian dengan memakai teknologi seperti basic authentication, digest authentication, NTLM authentication, Kerberos authentication, X.509 certificates, .NET Passport authentication, dan otentikasi berbasis form. Ingat bahwa kadang-kadang Kita harus membuktikan keabsahan client dan pada kali lain keabsahan server. Kita dapat juga membuktikan bahwa data datang dari sebuah prinsipal dengan menKitatangani dan membuktikan tKitatangan digital, seperti yang yang dipekerjakan oleh XMLDSIG dan PKCS #7. Simpanlah data rahasia Kita dengan aman, terutama data otentikasi seperti password dan nomor-nomor PIN.

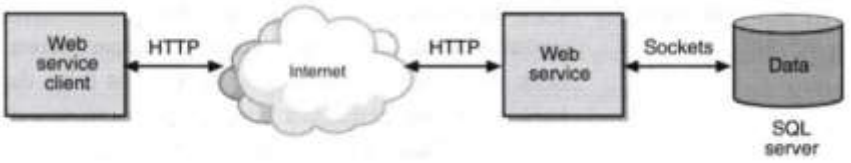
Tampering with data	Lindungi data dengan Daftar Kendali Akses sesuai (ACLS) atau ijin. Tentukan apakah data telah dirusak menggunakan hash atau kode otentikasi pesan. Lindungi data on-the-wire data memakai SSL/TLS atau IPSEC.
Repudiation	Perlindungan dari repudiasi sering melibatkan otentikasi kuat dan penKitatanganan data, juga pemakaian pembuatan catatan atau auditing yang ekstensif dan aman.
Information disclosure	Data dapat dilindungi dari penglihatan liar memakai ACLS dan perizinan yang sesuai. Juga, pertimbangkan bahwa jika Kita tidak menyimpan data, maka data tersebut tidak dapat disingkapkan. Teknik penjagaan privasi seperti enkripsi dapat membantu jika kunci untuk enkripsi dan dekripsi data juga dilindungi dari ancaman penyingkapan. SSL/TLS dan IPSEC menyediakan kerahasiaan on-the-wire, dan Sistem Enkripsi File (EFS) menyediakan privasi untuk file dan direktori.
Denial of service	Ancaman DoS sulit untuk dilindungi sebab tidak mudah dikatakan apakah sebuah server sekedar sibuk atau sedang diserang. Jika mengatur permintaan pengguna, Kita mungkin

	mengunci akses bagi para pengguna sah. Beberapa pertahanan sederhana meliputi pembatasan terhadap apa yang dapat dilakukan pengguna tanpa otentikasi. Sebagai contoh, Kita mungkin dapat mengalokasikan 10 persen sumber daya untuk para pengguna anonim dan 90 persen untuk para pengguna tervalidasi (Sumber-daya meliputi data cache , waktu CPU , ruang kosong dalam harddisk , bandwidth jaringan , dan koneksi ke database). Sebagian dari solusi ini berada di luar kendali langsung Kita dalam suatu aplikasi dan mungkin meliputi firewall dan router penyaring paket.
Elevation of privilege	Jangan meminta status atau ijin yang tidak diperlukan. Dengan cara itu, jika kode Kita mempunyai suatu kekurangan keamanan dan penyerang dapat mengeksekusi kode yang merusak atau menyebabkan kejadian yang membahayakan, dia tidak dapat menyebabkan banyak kerusak an sebab izinnya ditekan.

Contoh Layanan Web

Jika semua ini hanya sedikit memberi pemahaman pada Kita berikut Sebuah contoh akan membantu. Mari kita perhatikan suatu skenario

sederhana di mana suatu Client layanan Web berkomunikasi dengan suatu layanan Web, yang pada gilirannya berkomunikasi dengan suatu database dan mengembalikan isi database kepada klien, seperti dilukiskan Gambar 6-1. Kita akan berasumsi bahwa client itu adalah seorang pengguna, bukan proses peer.



Gambar 6-1 Contoh skenario layanan Web.

Skenario generik ini berlaku bagi banyak layanan Web. Tabel 6-2 berisi daftar sebagian dari ancaman terhadap sistem dan bagaimana cara menanganinya.

Tabel 6-2 Ancaman terhadap layanan Web dan Teknik Pengananan yang Tepat

Target	**ID**	**Jenis Ancaman**	**Uraian**	**Teknik Penanganan**
Layanan Web	1	S	Penyerang merobohkan layanan Web memakai serangan DoS tersebar dan menempatkan Web jahat miliknya di internet Aplikasi client tidak mengetahui bahwa	Gunakan koneksi SSL/TLS yang dipicu oleh client Untuk melakukan otentikasi server.

			ia sedang berkomunikasi dengan seorang penjahat.	
Data on the wire dari client ke layanan	2	T dan I	Penyerang memKitang atau memodifikasi data yang berjalan dari client kepada server dan sebaliknya.	Gunakan SSL/TLS atau IPSEC ke encrypt data ketika merambat ke dan dari layanan Web.
Data on the wire dari client ke layanan	3	E	Penyerang memKitang data password berjalan dari client ke server; Jika pengguna seorang administrator, penyerang dapat memakai username dan password	Gunakan suatu mekanisme otentikasi yang tidak melewatkan password secara murni bebas dari bahaya/kecurigaan ke seberang kawat, atau gunakan SSL/TIS atau IPSEC untuk melindungi saluran itu.
Layanan Web	4	D	Penyerang membanjiri layanan Web dengan beribu-ribu permintaan langsung dan memperlambat layanan Web itu.	Gunakan sebuah firewall untuk membatasi data yang diperbolehkan. Bangun logika yang membatasi data yang dapat dikirim oleh satu pengguna

				atau alamat IP. Pembatasan oleh alamat IP dapat bermasalah, karena banyak pengguna sah menggunakan ISP yang mempunyai jumlah alamat IP terbatas, maka banyak permintaan tampaknya datang dari IP yang sama ketika mereka ternyata berasal dari para pengguna di belakang sebuah proxy server.
Data SQL Server	5	T dan I	Penyerang mengakses data di dalam SQL Server secara langsung dibandingkan lewat layanan Web.	Batasi apa yang diperbolehkan dalam data yang dipakai untuk menyusun query SQL.
SQL Server	6	S, T, R, I, D dan E	Penyerang memakai prosedur tersimpan diperluas xp_cmdshell yang terdapat dalam SQL Server untuk memanggil kode	Batasi apa yang diperbolehkan dalam data yang dipakai untuk membangun SQL query. Buang prosedur tersimpan diperluas tak

			berbahaya di SQL database. Perintah ini dapat memanggil perintah manapun di server yang bersangkutan.	terpakai seperti xp_cmdshell. Jangan menghubungkan diri ke SQL Server sebagai sysadmin karena ia dapat melaksanakan Tugas SQL Server manapun, termasuk memanggil xm_cmdshell.

Skenario ini seharusnya membantu Kita memahami proses untuk menentukan teknik dan teknologi apa yang perlu dipekerjakan dalam membangun solusi yang aman. Pendekatan ini jauh lebih baik daripada sekedar menaburkan "serbuk keamanan ajaib" pada aplikasi dan berharap supaya aplikasi itu aman dari serangan.

Teknologi Keamanan Layanan Web

Tahap yang ketiga dari proses threaty-modeling adalah memilih teknologi yang sesuai untuk menerapkan teknik Penanganan ancaman itu yang sudah di pilih. Sebelum menguraikan teknologi tertentu, mari kita lihat apa yang sekarang ini disediakan oleh infrastruktur layanan Web dalam hal fitur-fitur keamanan. Protokol komunikasi yang banyak dipakai oleh layanan Web, yaitu SOAP, tidak mendefinisikan protokol keamanan. Protokol dibuat berdasarkan server Web dan mungkin aplikasi client untuk menyediakan jasa itu.

Alasan utamanya adalah SOAP tidak bergantung pada transpor, sedangkan layanan Web memakai HTTP sebagai transpor. Namun, layanan berbasis-SOAP lain mungkin memakai SMTP atau teknologi lain sebagai transpor. Ini dapat menjadi masalah jika model ancaman Kita menentukan bahwa data harus aman saat dalam perjalanan dari client ke semua server back-end.

Mari kita lihat suatu contoh pada Gambar 6-2. Client berkomunikasi dengan suatu layanan Web, dan layanan menentukan SSL atau TLS untuk melindungi data client itu selagi data berpindah antara client dan layanan Web. Ia kemudian akan mengirimkan data ke sebuah layanan back-end yang memakai soket penggunaan itu sebagai sebuah transpor.

Gambar 6-2 Skenario yang memakai SSL atau TLS untuk melindungi client dan data server.

Dapatkah Kita melihat letak persoalannya? Perlindungan yang disajikan oleh SSL/TLS berlaku hanya untuk mata rantai antara client dan server, bukan pada mata rantai antara server dengan server aplikasi back-end. Di dalam banyak kejadian, ini tidak sampai menjadi masalah. Namun, jika Kita menentukan dalam threaty-modeling Kita bahwa ancaman penyingkapan informasi itu ada saat data meninggalkan layanan Web dan server back-end, SSL/TIS tidak akan bekerja. Sekalipun data dilindungi kembali memakai SSL/TLS antara layanan Web dan server aplikasi, ia

akan tetap dalam teks murni dalam waktu tertentu di layanan Web dan boleh jadi dilihat oleh seorang pengguna di server (Tentu saja, harus punya kepercayaan kepada orang yang menjadi administrator layanan Web itu!). Sekali lagi, ini boleh jadi suatu risiko penyesuaian diri Kita dengannya. Namun, keputusan Kita harus didasarkan sepenuhnya pada threaty-modeling Kita.

Sekarang, setelah Kita mempunyai suatu pemahaman dasar tentang fitur keamanan yang disediakan infrastruktur layanan Web, mari kita memperhatikan beberapa cara untuk menerapkan teknologi keamanan yang sesuai.

Otentikasi Layanan Web

Otentikasi adalah kemampuan untuk membuktikan bahwa suatu entitas. Sebagai. contoh, seorang pengguna atau sebuah komputer sesuai dengan yang diklaimnya. Kita dapat mencocokkan klaim dengan meminta entitas itu, yang juga disebut principal, yaitu penyedia identitas rahasia. Identitas rahasia sering mengambil format sebuah username dan sebuah password. Ingatlah bahwa beberapa protokol otentikasi lebih aman dibandingkan yang lain dan dengan demikian Kita dapat lebih yakin bahwa identitas rahasia itu benar-benar datang dari pengguna yang benar dan tidak dimainkan lagi oleh seorang penyerang. Suatu layanan Web yang berjalan di atas IIS mempunyai sejumlah protokol otentikasi yang tersedia untuk itu, kebanyakan tercatat sebagai:

1. Otentikasi anonim
2. Otentikasi dasar

3. Otentikasi terekstrak
4. Otentikasi Windows
5. Otentikasi berbasis-sertifikat
6. Otentikasi berbasis form
7. Otentikasi Paspor .NET

Mari kita bahas satu per satu secara mendetail.

Otentikasi Anonim

Otentikasi anonim memang sederhana karena tanpa nama. Tidak ada otentikasi yang mengambil tempat. Ini adalah mekanisme otentikasi yang akan Kita pakai untuk data publik. Kita tidak harus memperbarui atau menambahkan kode apa pun ke client SOAP Kita karena tidak terjadi otentikasi.

Otentikasi Dasar

Otentikasi dasar mungkin menjadi format otentikasi yang paling umum di internet karena kesederhanaannya. Otentikasi ini disediakan oleh semua browser di semua platform. Otentikasi dasar juga tidak aman karena username dan password dikirimkan dalam setiap permintaan dari client ke server apa adanya karena data tidak dienkripsi. Dengan demikian, berarti Kita harus memakai semacam kanal enkripsi untuk melindungi username, sehingga password tidak dapat diketahui oleh pengguna jahat. Tentu saja, secara teknis Kita tidak perlu mengenkripsi saluran, jika Kita tidak peduli apakah username dan password disingkapkan. Sebetulnya bagus jika layanan Web Kita tidak menangani data rahasia seperti informasi saham

atau berita utama. Ada perbedaan antara memakai otentikasi sebagai alat untuk melindungi akses ke data dan memakai otentikasi sebagai alat untuk mengidentifikasi para pengguna untuk mengirimkan isi yang ditujukan bagi mereka. Sekali lagi, Kita seharusnya memakai threaty-modeling untuk menentukan apakah otentikasi dasar adalah cukup baik untuk aplikasi Kita.

Di dalam IIS 5 dan 6, account yang dipakai untuk otentikasi dasar haruslah account Windows yang sah. Namun, ketika otentikasi dasar dipakai oleh ASP. NET atau dengan demikian, dipakai oleh layanan Web yang ditulis memakai ASP.NET, Kita dapat memakai suatu database lookup untuk menentukan apakah identitas itu benar. Kita dapat mengatur otentikasi dasar di dalam tool administrasi IIS.

Jika Kita memakai otentikasi dasar, otentikasi terekstrak, atau otentikasi Windows, Kita dapat menetapkan username dan password pada client SOAP, seperti ditunjukkan kode VBScript berikut:

```
Setsc-CreateObject("MSSoap.SoapClient")
sc.mssoapinit("http://www.fabrikam.com/webse
rvice/service.asmx?wsdl")sc.ConnectorProper
ty("AuthName")-"username"
sc.ConnectorProperty("AuthPassword")-
"password"Status-
sc.GetShippingStatus("lOOOl")
```

Jika Kita menulis aplikasi sisi client Kita memakai C++ dan kelas SOAP ATL, Kita dapat menetapkan username dan password dalam metode CAt! Http Client: AddAuthObj. Jika kode client Kita ditulis dalam C# atau bahasa lain, seperti Visual Basic .NET, Kita dapat memakai kode seperti di bawah ini untuk membuat koneksi terotentikasi pada layanan Web

(Kode ini akan bekerja untuk otentikasi dasar, terekstrak, dan Windows) .

```
usingSystem;
usingSystem.Net;
usingSystem.Web.Services.Protocols;
ClientApp.localhost.Services-
newClientApp.localhost.Service();s.Credenti
als-
newNetworkCredential(username,password,doma
inl;stringshipped-
s.GetShippingStatus("lOOOl");
```

Dari semua skema, otentikasi dasar memang sangat lemah, terutama jika tidak dipakai SSL/TLS.

Otentikasi-terekstrak

Seperti halnya otentikasi dasar, otentikasi terekstrak adalah sebuah internet. Kedua-duanya diuraikan dalam RFC 2617 pada http://www.ietf.org/rfc/ rfc2617.txt. Namun, tidak sama dengan otentikasi dasar, otentikasi terekstrak tidak umum karena satu-satunya browser pendukungnya adalah Internet Explorer 5 dan versi yang lebih baru. Otentikasi terekstrak tidak mengirimkan password pengguna apa adanya, tetapi memakai hash, atau ekstrak dari password. Data yang disediakan oleh server dipakai untuk membuktikan keaslian pengguna. Jelaslah, otentikasi terekstrak sedikit lebih aman dibandingkan otentikasi dasar dan otentikasi ini memiliki kelebihan, yaitu tidak perlu memakai SSL untuk menyembunyikan password pengguna. Namun, seperti Penulis katakan, hanya sedikit lebih aman. Catat juga bahwa otentikasi terekstrak itu bekerja di IIS 5 dan 6 hanya jika Active Directory diinstall karena Active

Directory dapat diatur untuk menyimpan salinan password pengguna dalam bentuk teks murni yang diperlukan oleh otentikasi terekstrak.

Seperti halnya dengan otentikasi dasar, Kita dapat memakai tool administrasi IIS untuk mengatur server Web agar mengharuskan otentikasi terekstrak.

Otentikasi Windows

Seperti namanya, otentikasi Windows memakai mekanisme otentikasi dalam Windows, sejak Windows 2000 dan seterusnya yang berarti NTLM dan Kerberos. Otentikasi Kerberos hanya dapat diterapkan saat Active Directory dipakai. Kelemahan utama otentikasi Windows adalah tidak berjalan dengan baik lewat internet. Namun, hal ini akan lain jika dipakai untuk solusi intranet karena dapat memakai informasi logon pengguna secara langsung, tanpa perlu meminta pengguna memasukkan username dan password mereka. Password juga tidak pernah dikirim lewat kabel dalam bentuk teks biasa.

Otentikasi Windows dapat diatur sebagaimana protokol lain, tetapi dalam kasus layanan Web ASP.NET, Kita dapat memilih untuk otentikasi Windows dengan menambahkan kode berikut ke dalam file web.config:

```
<authenticationmode-"Windows">
<!authentication>
```

Kita dapat juga memaksa ASP.NET untuk mewakilkan pengguna yang melakukan panggilan dengan entri konfigurasi web.config berikut ini:

```
<system.web>
<identityimpersonate="true"I>
<!system.web>
```

Kita kemudian dapat memperoleh nama pemanggil dengan kode berikut

di dalam layanan Web Kita:

```
usingSystem.Security.Principal;
Windowsldentitywi=WindowsldentityGetCurrent(
);stringname=wi.Name;
```

Otentikasi Berbasis Sertifikat

Sertifikat memakai kunci pribadi yang lebih besar daripada password untuk menentukan identitas prinsipal. Sebelum membahas bagaimana dan mengapa Kita akan memakai sertifikat, mari sepintas kita: pelajari teknologi di belakang sertifikat. Berlainan dengan kunci enkripsi simetris yang memakai kunci tunggal untuk enkripsi dan dekripsi data, sertifikat memakai enkripsi tidak simetris yang juga disebut enkripsi kunci publik. Ketika suatu sertifikat dibuat, dua kunci besar dibuat, yaitu satu kunci pribadi dan satu lagi kunci publik. Kunci pribadi, seperti namanya, menyiratkan pribadi dan harus dilindungi. Kunci publik dapat dibuat publik dan ditempelkan ke sertifikat yang berisi informasi tentang pemilik kunci pribadi.

Kriptografi kunci publik juga mempunyai atribut penting berikut:

1. Data yang dienkripsi dengan kunci publik hanya dapat didekripsi memakai kunci pribadi.
2. Data yang dienkripsi dengan kunci pribadi hanya dapat didekripsi memakai kunci publik.

Jika Kita mempunyai sebuah kunci pribadi, maka Kita dapat mengirimkan pesan yang diotentikasi ke orang lain. Dengan kata lain, penerima dapat membuktikan bahwa Kita mengirim data. Ini dapat terjadi karena Kita melakukan enkripsi memakai kunci pribadi yang hanya Kita miliki; kunci

itu sifatnya pribadi. Hanya kunci publik yang dapat mendekripsi pesan itu. Kunci publik ada di dalam sertifikat yang meliputi detail Kita. Oleh karena itu, Kita sudah pasti mengirim data itu.

Proses enkripsi memakai kunci pribadi juga disebut signing. Jika Kita mempunyai kunci publik orang lain, Kita dapat mengirimi orang itu pesan terenkripsi. Sebenarnya, proses menKitatangani itu sedikit lebih rumit dari itu. Datanya sendiri tidak ditKitatangani, melainkan hanya suatu hash dari data yang ditKitatangani karena demi kecepatan.

Sertifikat sekarang kebanyakan memakai sebuah algoritma kriptografik yang disebut RSA, yaitu sebuah paten USA yang berakhir pada bulan September 2000.

Catatan: Beberapa hal yang sepele: algoritma RSA memakai bilangan prima besar ketika menciptakan kuncinya. Nomor paten untuk RSA yang sekarang telah kadaluwarsa adalah 4,405,829. Ini sebuah bilangan prima!

Ketika Kita memakai SSL/TLS, sesi antara client dan server dilindungi dari mata-mata liar memakai enkripsi simetris. Bagaimanapun juga, server juga diotentikasi. Ketika Kita mengunjungi toko buku Barnes & Noble di http://www.bn.com dan Kita pergi keluar, suatu sesi SSL/TLS terbentuk. Sepanjang proses jabat tangan, nama situs Web yang Kita kunjungi dibandingkan dengan nama dalam sertifikat untuk menentukan apakah server dengan tepat mendekripsi data acak yang disajikan oleh client setelah dienkripsi dengan kunci publik sertifikat itu. Jika situs itu dapat melakukannya, situs tersebut pasti mempunyai kunci pribadi terlindungi

yang terkait dengan kunci publik dalam sertifikat. Juga, kebenaran tanggal dibuktikan dalam sertifikat. Jika semua langkah-langkah ini berhasil, server diotentikasi dan pengguna mengetahui dia sedang berkomunikasi dengan situs Barnes & Noble yang sebenarnya, bukan yang palsu.

Kabar baik tentang SSL/TLS adalah bahwa ia juga mendukung suatu proses otentikasi client opsional. Dalam hal ini, server menceritakan kepada aplikasi client untuk menyediakan suatu sertifikat dan untuk mendekripsi sebuah data terkumpul yang dienkripsi memakai kunci publik dalam sertifikat. Setelah beberapa pengujian lain (kita berharap sukses) berhasil, client juga telah diotentikasi. Otentikasi client SSL/TLS adalah suatu langkah opsional dalam proses koneksi SSL/TLS dan kenyataannya jarang digunakan, kecuali dalam lingkungan sangat aman. Di dalam lingkungan luar biasa aman, sertifikat dan kunci pribadi dapat disimpan dalam sebuah kartu pintar.

Kita dapat memaksa client SOAP untuk memakai suatu sertifikat client spesifik ini dengan susunan berikut saat Kita memakai Microsoft SOAP Toolkit:

```
sc.Conn
ctorProperty("SSLClientCertificateName"l-
"mike@fabrikam.com "
```

Kita perlu menetapkan bagian nama umum (CN) dari nama subjek sertifikat. Perhatikan bahwa jika Kita mempunyai lebih dari satu sertifikat dengan nama umum subjek yang sama, yang pertamalah yang akan terpilih.

Jika kode client Kita ditulis memakai Visual Studio Web Reference Proxy code (yang dihasilkan setelah Kita membuat suatu Acuan Web), Kita dapat

menetapkan sertifikat client itu memakai koleksi ClientCertificate layanan itu. Perhatikan bahwa Kita sekarang ini tidak dapat mengakses sebuah sertifikat di dalam CAPI (CryptoApi), seperti yang dapat dilakukan kode SOAP Toolkit. Kita harus memakai sebuah file sertifikat.

Otentikasi Berbasis Form

Otentikasi berbasis form bukan cara industri untuk melakukan otentikasi pengguna, tetapi merupakan metode yang sangat populer. Otentikasi berbasis form biasanya mengacu pada suatu sistem di mana permintaan yang tidak disahkan diarahkan ke form HTML memakai HTTP redirection. Pengguna menyediakan identitas rahasia (username dan password) dan mengirimkan form tersebut. Jika aplikasi mengesahkan permintaan dengan melihat database atau XML file, pengguna akan diberi akses.

Otentikasi berbasis form memakai halaman HTML, maka tidak didukung oleh layanan Web karena layanan Web tidak mempunyai I. Bagaimanap un juga, ASP.NET mendukung otentikasi berbasis form.

Otentikasl .NET Passport

.NET Passport adalah suatu layanan otentikasi terpusat yang disajikan oleh Microsoft dan menawarkan sebuah layanan tunggal pendaftaran dan profil situs anggota. Ini bermanfaat bagi pengguna karena dia tidak lagi harus logon untuk mengakses situs atau sumber daya terlindung yang baru. Jika Kita ingin situs Kita cocok dengan otentikasi dan otorisasi Pasport, ini adalah penyedia yang Kita perlukan. Untuk informasi lebih jauh, lihat

Dokumentasi Passport di http://www.passport.com/ business.

Karena otentikasi Passport memakai cookies, sekarang ini ia belum didukung oleh layanan Web. Namun, ini akan berubah. ASP.NET mendukung otentikasi Passport, seperti halnya IIS 6.

Otorisasi Layanan Web

Setelah aplikasi Kita mengidentifikasi sebuah prinsipal memakai mekanisme otentikasi, ia harus menentukan apakah pengguna mempunyai akses kepada berbagai sumber daya yang dilindungi layanan Kita dan dapat memanggil fungsi-fungsi tertentu. Windows 2000 dan berikutnya menyediakan banyak jalan untuk memberi hak akses ke sumber daya; yang paling penting dan berhasil adalah ACLs.

Windows NT dan versi yang lebih baru melindungi sumber daya teramankan (seperti file) dari akses tidak sah dengan menerapkan pembedaan akses kontrol lewat discretionary access control lists (biasanya disingkat sebagai ACL dan bukan DACL). Sebuah ACL terdiri atas serangkaian ACE (Access Control Entries). Masing-Masing ACE mendaftar suatu prinsipal dan berisi informasi utama dan operasi yang dapat dilakukan prinsipal atau sumber daya. Sebagai contoh, sebagian orang diwarisi akses membaca, sedangkan yang lain mungkin mempunyai kendali penuh.

Secara harafiah, ACL adalah penghalang terakhir aplikasi Kita terhadap suatu serangan, dengan kekecualian enkripsi dan manajemen kunci yang baik. Jika seorang penyerang dapat mengakses suatu sumber daya, berarti pekerjaannya dilaksanakan. Oleh karena itu, sangat penting bahwa Kita

mempunyai ACLs sesuai pada sumber daya yang Kita lindungi. Sebagai contoh, perlukah semua pengguna (juga disebut Everyone) mempunyai akses penuh ke file sensitif? Mungkin tidak, tetapi para pengguna tertentu mungkin ya.

Keindahan yang sesungguhnya dari ACLs adalah bahwa mereka selalu dipaksakan tanpa terkait dengan mekanisme akses. Oleh karena itu, jika untuk beberapa alasan seorang penyerang dapat membypass layanan Kita dan dengan demikian melewati logika otorisasi rumit di tingkat aplikasi, sekaligus mengakses suatu sumber daya secara langsung, kebijakan otorisasi di dalam ACLs akan tetap dilaksanakan.

Sedikit di atas lapisan aplikasi adalah otorisasi di dalam ASP.NET dan di dalam aplikasi Kita, atau di dalam perijinan database (umpamakan Kita sedang memakai suatu database).

ASP. NET menawarkan teknik-teknik otorisasi yang kaya dengan menyertakan kebijakan-kebijakan itu, semudah mengatur sebuah properti XML. Sebagai contoh, Kita dapat membatasi akses kepada sebagian dari layanan Web Kita sedemikian rupa, sehingga para pengguna tanpa nama ditolak dengan menambahkan yang berikut kepada file web.config:

```
<authorization>
<denyusers-"?"I>
<!authorization>
```

Dalam hal ini, “? " berarti para pengguna tanpa nama dan " * " berarti semua pengguna. Kita juga dapat mengijinkan para pengguna tertentu memakai <allow users="name " />, dengan., name" adalah satu atau lebih nama yang dipisahkan oleh tKita koma.

Privasi dan lntegritas Layanan Web

Jika Kita memikirkan privasi dan integritas data SOAP, ingatlah bahwa ada dua aspek pada kedua jenis teknologi. Aspek pertama adalah pengamanan saluran antara client dan layanan. web dan yang kedua adalah pengamanan muatan SOAP atau data di dalam muatan. Bukan hanya ketika bepergian dari client ke layanan Web dan sebaliknya, tetapi juga ketika data disimpan dalam beberapa bentuk penyimpanan data. sekenario sebelumnya dapat dipahami dengan baik dan mudah ditangani tanpa kode modifikasi dengan memakai SSL/TLS atau IPSec. Karena protokol ini berada di bawah lapisan aplikasi (lapisan tempat layanan Kita berada), mereka sepenuhnya transparan. Cukup sebuah tombol sederhana di dalam Server Web. Kita mempunyai suatu koneksi terlindung dengan SSL/TLS. Sekarang ini, tidak ada cara untuk enkripsi pesan SOAP atau untuk menyediakan integritas data muatan. cara khusus tepat, jika Kita ingin mengijinkan client mana pun terhubung ke layanan Kita, dimana Kita harus didukung layanan untuk mencapai tujuan ini.

Lalu, apa yang terjadi jika Kita ingin meluncurkan enkripsi atau mekanisme integritas milik Kita sendiri? Kita dapat melakukan ini jika Kita mempunyai kendali atas kode client dan kode server karena Kita dapat menentukan format data seperti apa yang akan terlihat. Kita dapat menemukan suatu contoh sangat sederhana untuk melakukan ini dengan perluasan SOAP di situs MSDN (http://msdn.microsoft.com/library/enus/dnaspnet/html/asp09272001.asp). Bagaimanapun juga, kode memakai suatu kunci hard-coded untuk

enkripsi dan dekripsi memakai DE (Data Encryption d). Jika Kita merencanakan untuk memakai kode itu di dalam aplikasi SOAP, Kita perlu menyimpan kunci itu di tempat lain di komputer, seperti di registry atau di dalam suatu file konfigurasi XML yang terlindung. Maksud dari terlindung adalah bahwa file itu tidak dapat dengan mudah diakses melalui Server Web dan dilindungi oleh ACL yang baik dan bukan di dalam kode sumbernya sendiri. Ketika Kita mendesain sistem yang memakai enkripsi buatan sendiri, manajemen kunci adalah suatu masalah yang sangat sulit dipecahkan. Salah satu manajemen kunci ini adalah memindahkan kunci itu dari srver ke client memakai SSL/TLS. Jangan biarkan client itu menentukan kunci karena seorang penyerang mungkin memilih kunci yang lemah! Kemudian, laksanakan komunikasi di saluran yang terbuka, tapi lakukan enkripsi terhadap data SOAP yang sesuai.

Kita dapat menentukan apakah suatu metode SOAP dipanggil lewat kanal yang dijamin aman memakai SSL/TLS dengan penggunaan kode berikut:

```
ifCHttpContext.Current.Request.IsSecureConne
ction){
//Connection is using SSL/TLS.
J else{
//Connection is not us ing SSL/TLS.
```

Kode ini tidak akan melacak apakah IPSec dipakai untuk melindungi saluran antara client dan server. Sesungguhnya, saat ini sama sekali tidak ada cara yang mudah untuk menentukan hal itu dari lingkungan yang diatur.

Selagi kita sedang membahas tentang kunci, jika Kita perlu menghasilkan data acak berkualitas, seperti untuk menciptakan enkripsi kunci, jangan

memakai kelas Random karena sangat mudah ditebak. Sebaiknya, pakailah kode berikut ini yang menciptakan data acak 32 byte:

```
usingSystem.Security. Cryptography; byte []
b==newbyte [32];
newRNGCryptoServiceProvider (). GetBytes(b);
```

Teknologi Keamanan dalam .NET Framework

.NET Framework menawarkan dukungan untuk enkripsi dan tKita tangan data, kebanyakan, enkripsi semua arus data dan menKitatangani arus data dengan dukungan khusus untuk XML data. Kemudian ini disediakan melalui dukungan bagi World Wide Web Consortium (W3C) XMLDSIG .

Sekali dengan kunci yang dipakai untuk enkripsi dan dekripsi data antara kedua komputer, Kita dapat dengan mudah melakukan enkripsi dan dekripsi memakai kode sebagai berikut yang menggunakan kode simetris RC2:

```
staticstringEncrypt (stringplaintext. byte
[] key, byte [] IV)) {try {
MemoryStreamms=newMemoryStream ():
RC2rc2=newRC2CryptoServiceProvider ():
CryptoStreams=newCryptoStream (ms,
rc2.CreateEncryptor(key,
IVl.CryptoStreamMode. Write);
byte [] p==Encoding.UTFB. GetBytes
(plaintext. ToCharArray ()); s. Write (p,
O, p. Lengthl:
s. FlushFinalBlock (l:
returnConvert.ToBase64String (ms. ToArray
()): Jcatch (Exceptionl {
```

```
returnnul1;
staticstringDecrypt (stringciphertext, byte
[] key, byte [] IV)) {try {
MemoryStreamms=newMemoryStream ();
RC2rc2=newRC2CryptoServiceProvider ();
CryptoStreams=newCryptoStream (ms.
rc2.CreateDecryptor(key,
IVl.CryptoStreamMode. Writel;
byte []
c==Convert.FromBase64String(ciphertext); s.
Write (c, O, c. Lengthl:
s. FlushFinalBlock ();
returnEncoding.UTF8. GetString (ms.
GetBuffer ());
} catch(Exception) {returnnull:
```

Maka, bukan hanya sekedar mengirimkan data Metode Web itu ke seberang kawat sebagai teks murni, melainkan Kita dapat mengenkripsi data dan melewatkannya sebagai string terkode Base64 atau mengirimkan kembali data sensitif dari server dengan cara yang sama. Oleh karena itu, apa yang terbuka bagi "pemeriksaan," seperti SOAP berikut ini:

```
<? xml version="l.O" encoding=="utf-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3c.org/2001/XMLSchem
a-
instance"xmlns:xsd="http://www.w3c.org/2001
/XMLSchema"
xmlns:soap="http://schemas.xml
soap.org/soap/en v el ope /">
<soap:Body>
<GetMeetingResponsexmlns="http://www.fabrika
m.com/soap">
<GetMeetingResult>MeetatMidnight!
```

```
<!GetMeetingResult>
</GetMeetingResponse>
</soap:Body>
</soap:Envelope>
```

Lebih aman, jika menjadi seperti ini:

```
<? xml version="1.0" encoding=="utf-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema
-
instance"xmlns:xsd="http://www.w3.org/2001/
XMLSchema"
xmlns:soap="http://schemas.xml
soap.org/soap/envelope/">
<soap:Body>
<GetMeetingResponse
xmlns="http://www.fabrikam.com/soap">
<GetMeetingResult>
LslO+R09UUMziJd0104POPOzaFxqGHS=
<!GetMeetingResult>
<!GetMeetingResponse>
</soap:Body>
</soap:Envelope>
```

Kita mencapai hasil ini hanya dengan memanggil fungsi enkripsi pada metode jalan keluar. Ini adalah contoh cuplikan kode tersebut

```
[WebMethod J
PublicstringGetMeeting( )(returnmeetingdata;
```

Menjadi

```
[WebMethod
PublicstringGetMeeting()(
returnEncrypt(meetingdata.key,IV);
```

Diasumsikan bahwa kunci dan nilai vektor inisialisasi telah dinegosiasikan oleh kedua belah pihak.

Masalah dengan kode ini adalah mensyaratkan kedua pihak memiliki kode tertentu yang akan baik jika Kita mengontrol client dan layanan. Namun, tidak berlaku jika Kita menghendaki setiap orang terhubung ke layanan Kita dari client lain. Hal ini membawa kita pada apa yang ada di horizon keamanan layanan Web.

Teknologi Keamanan Layanan Web Masa Depan

Pada Oktober 2001, Microsoft memperkenalkan Global XML Web Services Architecture (GXA) yang didasarkan pada protokol layanan Web sekarang, seperti SOAP, WSDL, dan UDDI dan menambahkan blok bangunan tambahan, termasuk keamanan yang disebut WS-Security.

Apa yang dimaksud dengan IV?

Initialization Vector (IV) adalah suatu bilangan acak, umumnya dengan jumlah bit yang sama seperti ukuran blok algoritma enkripsi yang dipakai sebagai titik awal untuk enkripsi satu set data.

Jika IV tidak dipakai, dua pesan ciphertext identik dihasilkan saat dua pesan plaintext identik dienkripsi memakai kunci yang sama. Namun, jika masing-masing pesan ciphertext dienkripsi dengan IV berbeda, pesan chiper-text yang dihasilkan akan berbeda sama sekali.

Untuk keamanan yang lebih baik, Kita perlu mengenkripsi masing-masing pesan dengan IV berbeda, terutama sekali saat pesan berisi sejumlah besar duplikasi. Aplikasi Kita bertanggung jawab untuk menyebarkan IV bersama pesan yang terenkripsi itu. Tidak perlu mengenkripsi sebuah IV.

WS-Security menyediakan sebuah bahasa keamanan untuk layanan Web. Ia menguraikan peningkatan pesan SOAP untuk menyediakan tiga

kemampuan, yaitu pertukaran identitas rahasia, integritas pesan, dan kerahasiaan pesan. Kita dapat memakai tiga mekanisme ini secara bebas atau dengan kombinasi untuk mengakomodasi beragam teknologi otentikasi dan kriptografi.

Spesifikasi selengkapnya tersedia di http:!/msdn.microsoft.com/ws/2001 / 10/ Security.

Kesalahan Umum dalam Keamanan

Untuk membangun suatu layanan Web yang kebal serangan, Kita tidak dapat hanya menyebar fitur keamanan di sana-sini. Seperti telah Penulis nyatakan lebih awal, Kita harus mendesain aplikasi Kita dengan keamanan yang ada dalam pikiran Kita .

Berikut ini adalah tiga kesalahan keamanan paling serius yang dilakukan pengembang (termasuk pengembang layanan Web):

1. Penyimpanan data rahasia secara tidak aman
2. Terhubung ke SQL Server secara salah
3. Membangun string SQL tak aman

Meskipun runtime bahasa umum .NET dan .NET Framework membantu mengurangi ancaman keamanan yang ada saat ini, seperti banjirnya buffer (buffer overruns), mereka tidak dapat melawan keputusan desain yang buruk. Pada kenyataannya, ketiga jenis kekeliruan berlaku bagi semua lingkungan pemrograman dan sistem operasi.

Mari kita tinjau masing-masing jenis kekeliruan secara detail.

Kesalahan #1: Menyimpan Data Rahasia dengan Tidak Aman

Penyimpanan data rahasia secara aman di dalam software itu mustahil.

Kita hanya mempersulit seorang penyerang mendapatkan data. Penyimpanan data yang aman, sedikit banyak lebih mudah dilakukan di server karena penyerang tidak mempunyai akses secara fisik ke software Kita. Bagaimanapun, menyimpan kode rahasia di client juga sangat mustahil. Kita akan mengalami kesulitan jika berpikir bahwa Kita dapat melaksanakan suatu jabat tangan rahasia atau melaksanakan transaksi rahasia berdasarkan suatu rahasia yang melekat di aplikasi client Kita. Penyerang dapat dengan mudah merekayasa balik aplikasi client untuk memecahkan rahasia itu.

Data rahasia meliputi data pribadi atau personal seperti password, kunci enkripsi, nomor-nomor identifikasi, angka penjualan, dan nomor kartu kredit. Ketika Kita menyimpan data, pertimbangkanlah apa yang akan terjadi pada Kita, bisnis, dan client Kita, jika data telah dimunculkan kepada para pengguna jahat (I dalam model STRIDE) atau telah dirusak oleh seorang penyerang (T). Jika Kita mungkin sulit menangkap isu ini, bayangkan saja bawa ini adalah data personal Kita yang tersimpan!

Ancaman penyingkapan informasi mudah ditangani. Kita hanya tidak perlu menyimpan data itu sejak awal. Penulis benar-benar serius. Dalam beberapa situasi, sah-sah saja memberi pilihan kepada para pengguna untuk menyimpan data mereka, tetapi biarlah itu menjadi suatu pilihan. Para pengguna mungkin mendapatkan kemudahan penggunaan, tetapi mereka mungkin akan menyalahkan Kita jika datanya tidak cukup terlindung dari penyerang.

Sebagai tambahan, beberapa data tidak memerlukan penyimpanan. Data itu hanya dipakai untuk mengesahkan bahwa pemakai mengetahui data itu,

misalnya sebuah password. Kita dapat menentukan apakah seorang pengguna mengetahui password tanpa menyimpan password itu sendiri. Kita lakukan ini dengan menghashing data dan hanya menyimpan hash. Kemudian, ketika pengguna memberikan password, kode Kita melakukan hashing dan membandingkan hasilnya. Jika keduanya sama, berarti pengguna mengetahui passwordnya.

Kode contoh yang berikut menunjukkan bagaimana Kita dapat memperoleh password dari seorang pengguna dan membandingkannya dengan hash yang lain:

```
publicboolComparePasswordHash(stringpassword
byte[]hash))(SHAlManagedh=newSHAlManaged();
UTF8Encodinge=newUTF8Encoding();byte[J
p==e.GetBytes(password l;
ComputeHash(p);byte[]hr==h.Hash;
boolsame=true;
for(inti=0;i<hr.Length;i++)(
if(hr[iJ!!=hash[i])(
same=false;
break;
returnsame;
```

Namun, bagaimana jika Kita memakai data rahasia seperti password? Di mana Kita seharusnya menyimpan dan bagaimana cara mengamankannya? Satu tempat yang mungkin adalah di dalam file web.config. Namun, jika seorang penyerang dapat mengakse file itu secara langsung, berarti data itu dapat dibaca. Hal yang sama berlaku bagi menyimpan data dalam sebuah file .aspx atau .asmx. Dengan mengesampingkan isu deployment, Kita seharusnya menghindari penyimpanan data sensitif di dalam file-file ini. Sebagai gantinya, simpan

di luar ruang Web. Dengan demikian berarti menyimpan data di dalam sistem file, tetapi di luar root dari ruang sistem Web atau menyimpannya di luar sistem file, seperti di registry system.

Kesalahan #2: Menghubungkan ke SQL Ser ver Secara Salah

Banyak Pengembang salah menghubungkan ke database SQL, termasuk SQL Server, memakai account sysadmin. Mereka melakukan ini karena pengujian menjadi lebih mudah, sehingga semuanya dapat beroperasi. Penulisngnya, ini mungkin juga berarti bahwa segalanya juga akan bekerja untuk penyerang.

Account sysadmin SQL Server, sa, adalah account yang paling mampu dan istimewa yang tersedia dalam SQL Server. Account ini dapat melakukan apa pun kepada suatu server database SQL. Jika Kita harus memakai otentikasi SQL (daripada Otentikasi Windows), Kita harus terhubung memakai account yang hanya memiliki izin yang diminta dalam database SQL Server dan tidak punya akses ke object lain dalam database. Pastikan juga bahwa account itu memakai password yang sangat kokoh.

Kesalahan #3: Menyusun String SQL yang Tak Aman

Akuilah bahwa Kita pernah menyusun string SQL seperti ini:

```
string sql="select* from table where
name="'+name+'"";
```

Nama variabel diberikan oleh pengguna. Namun masalahnya, di SQL string ini penyerang dapat menyisipkan pernyataan SQL di dalam variabel nama.

Bayangkan input berikut, di mana name = "Blake ", yang membuat pernyataan SQL sederhana ini:

```
select*from table where name = 'Blake'
```

Namun bagaimana jika seorang penyerang memasukkan name = "Blake' delete from table where name = 'Lynne' -- ", yang membangun pernyataan merusak berikut?

```
select*from table where name ='Blake'
delete from table where name ='Lynne'
```

Pernyataan ini akan menghasilkan semua data di dalam tabel yang bernama Blake, dan kemudian akan menghapus semua baris yang bernama Lynne! Percayalah, banyak serangan lebih membahayakan dari pada ini. Bagaimana mungkin ini terjadi? Sebab koneksi SQL dibuat memakai account sysadmin yang dapat melakukan apa pun kepada database, termasuk menghapus data mana pun. Ingat penggunaan urutan '--', yaitu sebuah operator komentar yang membuat serangan itu lebih mudah disingkirkan.

Sekarang, mari kita perhatikan sebuah contoh dan beberapa perbaikan.

Contoh yang Terperinci

Setelah kita melihat beberapa praktek terbaik untuk membangun layanan Web dan beberapa kesalahan umum, mari kita perhatikan suatu contoh yang terperinci. Kita akan membahas suatu skenario tak aman yang umum dan kemudian kita akan mencari jalan untuk mengamankan layanan Web itu.

Versi Tak Aman (Jangan Mencobanya di Rumah!)

Lihat kode C# berikut, dan perhatikan baik-baik jika Kita dapat menyingkapkan kelemahannya:

```
[WebMethod(Description-
"DangerousShippingStatus")]publicstringGetS
hippingStatus(stringId)(
stringStatus-"No";stringsqlstring-"";try(
SqlConnectionsql-newSqlConnectionC
@"datasource-localhost;"+
"userid-sa;password-
password;"++"initialcatalog-Shipping");
sql.Open(l ;
sqlstring-"SELECTHasShipped"++"FROMdetail"++
"WHEREID-"'++Id+'"";
SqlCommandcmd-
newSqlCommandCsqlstring,sqll;if((int)cmd.Ex
ecuteScalar()!-0)
Status-"Yes";
}catch(SqlExceptionself
Status-
sqlstring+"failed\n\r";foreach(SqlErroreins
e.Errors){
Status+-e.Message+"\n\r";
}catch(Exceptionel(Status-e.ToString();
returnStatus;
}
```

Kita menemukan bug? OK, inilah kesalahan-kesalahan itu. Kesalahan yang pertama adalah membuat koneksi ke database SQL sebagai sa, account sysadmin. Kita tidak membutuhkan account yang demikian tinggi seperti itu hanya untuk query suatu tabel. Cukup sebuah kesalahan kecil, dan sa dapat melakukan apa pun yang dimauinya, seperti menghapus atau memodifikasi tabel, bahkan tabel master SQL Server.

Berikutnya, account sysadmin mempunyai password yang mudah ditebak. Ketiga, password tercantum di halaman layanan Web. Jika seorang

penyerang mengakses halaman ini, dia akan mengetahui detail koneksi dan mengetahui bahwa SQL Server ada di mesin layanan Web.

Barangkali, masalah yang paling berbahaya adalah bahwa kode tersebut rentan terhadap penembusan SQL karena penyerang dapat menetapkan ID dengan nilai sah dan diikuti oleh serangkaian pernyataan SQL berbahaya yang semuanya dieksekusi. Juga, seKitainya komunikasi SQL gagal untuk beberapa alasan, seperti pernyataan SQL yang cacat atau kegagalan koneksi, layanan Web akan mengirimkan banyak data kembali ke penyerang itu, mencakup teks yang menyusun SQL pernyataan. Ini sudah terlalu banyak untuk diceritakan kepada seorang penyerang. Sesungguhnya, yang ditampilkan itu tidak banyak gunanya untuk orang lain kecuali seorang pengembang.

Kode mempunyai sebuah kesalahan terakhir. Dapatkah Kita menemukan itu? Kita mungkin sulit menemukannya karena kesalahan itu sulit dipisahkan. Bayangkan si penyerang mengirimkan suatu string pada kode ini yang menyebabkan suatu pernyataan SQL cacat untuk dibangun. Kelas SQL akan melemparkan suatu eksepsi. Bagaimanapun juga, koneksi ke SQL Server tidak akan tertutup. Akhirnya, akan menjadi pengumpul sampah. Namun, apa akibatkan jika penyerang mengirimkan beribu-ribu permintaan cacat? Koneksi ke SQL Server akan menjadi lelah dan koneksi sah akan gagal. Ini bisa menjadi sebuah serangan DoS yang sangat bagus.

Solusi yang Aman

Sekarang, mari kita perhatikan sebuah versi yang mempunyai lapisan pertahanan bergKita. Jika satu mekanisme bertahan gagal, sedikitnya satu

yang lain akan melindungi aplikasi itu berikut datanya:

```
SqlClientPermissionAttribute(SecurityAction
PermitOnly,
AllowBlankPassword=false)J
[RegistryPermissionAttribute(SecurityAction.
PermitOnly,Read=@"HKEY_LOCAL_MACHINE\SOFTWA
RE\Shipping")]
publicstringSafeGetShippingStatus(stringId){
SqlCommandcmd=null;
stringStatus="No";try{

//Check for valid shipping
ID.Regexr=newRegex(@"A\d(10)$");if(!r.Match
(Idl.Success)
thrownewException("InvalidID");
//Get connection string from registry.
SqlConnectionsqlConn=newSqlConnectionCConnec
tionString);

//Add shipping ID parameter.
stringstr="sp_HasShipped";
cmd=newSqlCommand(str,sqlConn);cmd.CommandTy
pe=CommandType.StoredProcedure;cmd.Paramete
rs.Add("@ID",Id);
cmd.Connection.Open();
if((int)cmd.ExecteScalar()!=Ol

Status="Yes";
}catch(Exceptionel{
if(HttpContext.Current.Request.UserHostAddre
ss=="127.0.0.1")Status=e.ToStringCl;
else
Status="Error.";lfina11y(
```

```
//Shutdown connection - - even on
failure.if(cmd!=null)
cmd.Connection.Close();

returnStatus;
//Get connecti on string.
internal string ConnectionStringget(
return(string)Registry
LocalMachine
OpenSubKeyC@"SOFTWARE\Shipping\")
GetValueC"ConnectionSrng");
```

Sepintas, kode ini terlihat lebih rumit, tetapi sebenarnya tidak. Beri Penulis ke sempatan menjelaskan bagaimana kode ini lebih aman dibandingkan contoh yang pertama.

Pertama, kode ini memerintahkan bahwa suatu pengiriman nomor identitas harus persis 10 digit. Hal ini ditKitai dengan ekspresi reguler /\ \d {10} $, yang hanya mencari 10-digit angka-angka (\d{IOJ) dari awal (A) sampai akhir ($) data masukan. Dengan menyatakan apa yang merupakan masukan sah dan menolak segalanya selain itu, kita membuat berbagai hal lebih aman. Penyerang tidak dapat hanya menambahkan SQL pernyataan pada ID pengiriman. Ekspresi reguler dimunculkan melalui System.Text. RegularExpressions.)

Kode meliputi lebih banyak pertahanan lagi. Perhatikan bahwa objek Sql Connection dibangun dari sebuah string koneksi dari registry. Perhatikan juga fungsi pengakses ConnectionString. Dalam rangka menentukan string ini, suatu penyerang diharuskan tidak hanya mengakses kode sumber ke layanan Web, tetapi juga mengakses kunci registry yang sesuai.

Data di dalam kunci registry adalah string koneksi:

```
datasourcedb007a; useridshipuser;
password&ugv4!26dfA-+8;
initialcatalogShipping
```

Perhatikan bahwa database SQL kini ada pada komputer lain. Seorang penyerang yang berkompromi dengan layanan Web tidak akan secara otomatis memperoleh akses otomatis ke data SQL. Kode juga tidak terhubung sebagai sa. Sebagai gantinya, kode ini memakai suatu account spesifik, shipuser, dengan suatu password kuat. Account khusus ini hanya membaca dan melaksanakan akses kepada objek SQL yang sesuai. Jika koneksi dari layanan Web ke database disepakati, penyerang dapat menjalankan hanya segenggam penuh prosedur tersimpan dan query tabel yang sesuai, dia tidak dapat menghancurkan database master.

Pernyataan SQL tidak dibangun memakai teknik penggabungan string yang tak aman, melainkan kode memakai query berparameter untuk memanggil prosedur yang tersimpan. Pemanggilan prosedur tersimpan lebih cepat dan lebih aman dibandingkan memakai penggabungan string karena database dan nama tabel tidak dimunculkan dan prosedur tersimpan dioptimalkan oleh mesin database.

Catatlah bahwa saat kesalahan terjadi, pengguna (atau penyerang) tidak menerima pemberitahuan apa pun, kecuali jika permintaan sifatnya lokal atau pada mesin yang sama dengan tempat layanan Web berada. Jika Kita mempunyai akses fisik ke Komputer layanan Web, bagaimanapun juga Kita "memiliki" komputer itu!

Berikutnya, koneksi SQL selalu ditutup pada handler finally, sehingga jika suatu eksepsi dinaikkan dalam blok try/catch, koneksi itu dibebaskan

secara baik-baik. Dengan demikian, mengurangi ancaman DoS.

Seperti dijanjikan, Penulis akan menjelaskan kedua atribut keamanan pada saat awal pemanggilan fungsi. Pertama, SQLClientPermissionAttribute mengijinkan SQL Server .NET Data Provider untuk memastikan bahwa seorang pengguna mempunyai suatu tingkat keamanan cukup untuk mengakses sumber data. Dalam kasus ini, dilarang memakai password kosong. Jika Kita mencoba menghubungi SQL Server memakai kode tanpa kehati-hatian, dan memakai password kosong, hal ini akan menaikkan suatu eksepsi. Atribut kedua, RegistryPermission Attribute, membatasi kunci atau kunci-kunci registry yang dapat diakses dan sampai tingkat apa (baca, tulis, dan seterusnya). Dalam hal ini, hanya satu kunci spesifik pemegang string koneksi yang dapat dibaca. Jika seorang penyerang mencoba membuat kode ini dengan mengakses bagian lain pencatatan, ia akan gagal.

Secara bersama-sama, semua mekanisme ini mendorong ke arah layanan Web yang sangat aman. Kita perlu selalu memakai mekanisme ini dan membuat lapisannya sedemikian, sehingga kode Kita aman dari serangan.

C. PENUTUP

Ringkasan

Di dalam bab ini, diuraikan sebagian dari fitur keamanan yang tersedia untuk Kita sebagai pengembang layanan Web. Fitur keamanan tidak didefinisikan dalam protokol SOAP sendiri karena SOAP tidak terbatas hanya untuk memakai HTTP. Oleh karena itu, aplikasi Kita harus meningkatkan fitur keamanan server Web yang telah ada. Akan menjadi

sangat penting jika semua fitur yang Kita pilih didasarkan pada data yang terkumpul dari sebuah latihan threaty-modeling. Sebagai contoh, Kita dapat memakai otentikasi dasar, otentikasi terekstrak, atau otentikasi .NET Passport untuk membantu mengurangi client dari ancaman spoofing. SSL/TLS dapat mengurangi ancaman spoofing server seperti juga ancaman perusakan data dan ancaman penyingkapan informasi dengan memanfaatkan enkripsi dan kode otentikasi pesan. SSL/TLS dapat juga menyediakan dukungan untuk otentikasi client memakai sertifikat otentikasi client opsional. Saat ini sedang diupayakan penyediaan fitur keamanan bagi pesan SOAP. Teknologi ini disebut Global XML Web Services Architecture.

Akhirnya, Penulis menguraikan beberapa kesalahan yang sangat umum dibuat oleh pengembang Aplikasi Web dan layanan Web, khususnya yang terfokus pada kepercayaan bahwa input dari pengguna bentuknya baik dan sederhana. Jika Kita memakai input tanpa melakukan pengesahan terlebih dulu, berarti Kita sedang menantikan bencana keamanan yang serius. Abaikan nasehat ini dan tanggung sendiri risikonya!

Pertanyaan

1. Bagaimana menurut pendapat anda cara membangun layanan Web yang Aman?
2. Sebutkan dan jelaskan tiga fase utama pada proses Threat-Modeling?
3. Sebutkan dan jelaskan model ancaman STRIDE?
4. Sebutkan dan jelaskan teknik teknik penanganan ancaman?

5. Sebutkan dan jelaskan jenis otentikasi layanan web?
6. Sebutkan dan jelaskan kesalahan umum dalam keamanan?

DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.

11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 7
Mendebug Layanan Web

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 7 akan di bahas Mendebug Layanan Web, yang mencakup diantaranya: Pendebugan Interaktif, Dasar Dasar Pendebugan, Pendebugan Jarak Jauh, Stack Pamggilan yang Sesuai dengan Layanan Web, Informasi yang Dibutuhkan Debugger, Metadata Assembly, Database Program, Informasi Pelacakan, Mendebug Kode Sumber, Instrumensasi Layanan Web, Pelacakan, Menegaskan Kesalahan, Direktif Praprosesor Kondisional, Log Pelacakan.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan Pendebugan Interaktif, Dasar Dasar Pendebugan, Pendebugan Jarak Jauh, Stack Panggilan yang Sesuai dengan Layanan Web, Informasi yang Dibutuhkan Debugger, Metadata Assembly, Database Program, Informasi Pelacakan, Mendebug Kode Sumber, Instrumensasi Layanan Web, Pelacakan, Menegaskan Kesalahan, Direktif Praprosesor Kondisional, Log Pelacakan.

B. PENYAJIAN

Mendebug Layanan Web

Setiap kali kita berusaha menuliskan blok kode yang baru, compiler selalu mengingatkan sejumlah kesalahan yang dibuat. Penulisngnya, beberapa

kesalahan tidak tertangkap oleh compiler dan akan muncul sebagai bug runtime.

Kebanyakan dari bug-bug tersebut akan terlihat oleh tester pada saat proses QA. Namun, tester juga manusia, sehingga kadang kala bug tetap ada sampai aplikasi diluncurkan.

Selama program ditulis dan diuji oleh manusia, maka tetap perlu mencari dan memperbaiki bug di seluruh tahap proyek. Untungnya, banyak sumber daya yang tidak dipakai untuk membantu dalam proses pencarian bug.

Dalam bab ini, akan membahas tool-tool upgrade mendebug yang disediakan oleh Microsoft Visual Studio.NET dan platform Microsoft .NET.

Pendebugan lnteraktif

Salah satu tool yang paling hKital ebagai amunisi pengembang adalah debugger. Debugger memungkinkan Kita menempelkannya ke suatu proses, diselipkan dalam status, dan bergantung pada debuggernya, bahkan mengontrol alur aplikasi. Jika debugger mendukung pendebugan jarak jauh, proses target dapat dicari dalam mesin yang lain.

Kita dapat memilih debugger untuk layanan Web pada .NET Kita. .NET Framework menyertakan dua debugger, CLR Debugger (DbgCLR.exe), yang berbasis Microsoft Windows, dan Runtime Debugger (CorDbg.exe), yang berbasis baris perintah. Visual Studio.NET juga memiliki dua debuggernya sendiri. Selain itu, Kita dapat memilih dari sejurnlah debugger pihak ketiga.

Dalam bagian ini, penulis akan membahas fungsionalitas yang didukung oleh kebanyakan debugger. Namun, contoh-contoh yang akan dipakai berbasis debugger Visual Studio .NET.

Penulis berasumsi bahwa kita sudah memiliki pengetahuan dasar cara mendebug aplikasi. Misalnya, penulis tidak membahas topik, seperti "Menelusuri kode, mengatur breakpoint ". Jika kita belum terbiasa dengan konsep ini, penulis sarankan untuk mempelajarinya lebih dulu sebelum meneruskan bagian ini.

Dasar-Dasar Pendebugan

Salah satu kelebihan Visual Studio .NET adalah integrasinya yang ketat dengan lingkungan runtime. Misalnya, jika kita membangun proyek layanan Web ASP.NET, maka Visual Studio.NET secara otomatis akan menerapkan aplikasi ke server Web.

lntegrasi ini mencakup dukungan untuk pendebugan layanan Web. Seperti yang sudah diketahui, kita dapat mulai mendebug dengan menekan F5. Sejumlah pekerjaan akan dilakukan secara otomatis. Visual Studio .NET secara otomatis akan melakukan tugas-tugas berikut ini:

1. Mengompilasi layanan Web. Hal ini akan memastikan bahwa aplikasi yang dikompilasi sesuai dengan kode sumbernya.
2. Menerapkan layanan Web. Visual Studio.NET akan menerapkan DLL yang baru dikompilasi dan file yang berubah ke dalam server Web. Kita dapat menerapkan layanan Web memakai Microsoft FrontPage Server Extensions dengan file yang dipakai bersama dan dipetakan ke sistem file yang menyimpan layanan Web tersebut.
3. Buka halaman .aspx di dalam browser. Hal ini akan menyebabkan proses pekerja ASP.NET memuat aplikasi layanan Web.
4. Lampirkan proses pekerja ASP.NET. Untuk melakukan hal ini, debugger

Visual Studio.NET akan mencari proses pekerja ASP.NET yang tepat yang menjalankan layanan Web, meskipun ada di dalam mesin yang berbeda.

Teknologi ini membuat Visual Studio.NET dapat secara otomatis mencari dan melampirkan ke dalam ASP.NET jarak jauh, proses yang menyimpan layanan Web. Teknologi ini memiliki derivatif yang hKital. Sementara kita mendebug client layanan Web, saat kita menemukan baris yang memanggil ke layanan Web lewat proxy, Kita dapat menekan Fl1 untuk memasuki pelaksanaan metode layanan Web. Visual Studio.NET akan menangkap panggilan ke layanan Web, menempelkannya ke dalam proses, lalu mengatur: breakpoint di bagian awal metode.

Fitur ini disebut sebagai kausalitas. Untuk memfasilitasi kausalitas, client layanan Web harus memiliki hak pengamanan yang memadai. Jika client layanan Web adalah sebuah aplikasi ASP.NET, maka account default yang menjadi dasar jalannya aplikasi tersebut belum memiliki izin yang memadai untuk memfasilitasi kausalitas. Oleh karena itu, untuk mengaktifkan fitur ini, kita harus memodifikasi atribut userName dan password pada elemen procesModel dalam file machine.config, sehingga aplikasi ASP. NET yang memanggil layanan Web dapat berjalan di bawah account dengan hak administratif.

Setelah selesai mendebug aplikasi kode yang Kita kelola, Kita dapat mengeluarkannya dari proses dan kode itu kembali dieksekusi. .NET. Namun, jika aplikasi juga memanggil kode yang tak dikelola dan kita ingin mendebugnya juga, kita tidak dapat mengeluarkannya dari proses tanpa menghentikannya.

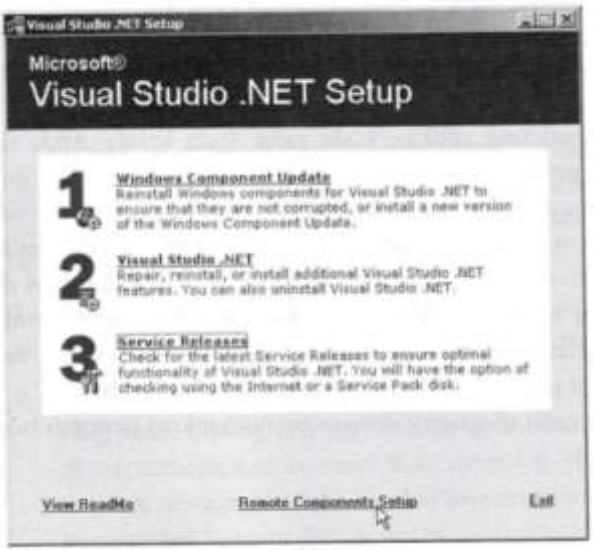
Cara mengatasi keterbatasan ini adalah dengan menginstal layanan dbgproxy. Setelah kita selesai mendebug kode yang tak dikelola, dbgproxy akan tetap membuka pendebugan terbuka untuk kita. Kemudian, kita dapat mengeluarkan kode dari proses tanpa harus menghentikannya lebih dulu.
Kita dapat menginstal dbgproxy dengan mengeksekusi perintah berikut ini:

```
Dbgproxy-install netstart dbgproxy
```

Perintah yang pertama akan menginstal layanan dbgproxy dan perintah kedua menjalankannya.

Pendebugan Jarak Jauh

Untuk memfasilitasi beberapa fitur pendebugan jarak jauh yang dijelaskan dalam bagian sebelumnya, Kita harus yakin bahwa lingkungannya sudah dikonfigurasi secara benar. Jika Kita kurang berpengalaman dalam mengkonfigurasi pendebugan jarak jauh dengan Visual Studio versi sebelumnya, Kita akan kagum karena begitu mudah dan cepatnya melakukan tugas ini dalam Visual Studio.NET.
Mesin yang menyimpan proses jarak jauh harus menginstal koleksi komponen COM. Cara termudah menginstal komponen ini adalah dengan menginstal Visual Studio .NET pada mesin tersebut. Dalam banyak kasus, terutama jika rnesin ada pada lingkungan produksi, solusi ini kurang optimal, sehingga setup Visual Studio .NET memberi Kita hanya untuk menginstal komponen-komponen untuk pendebugan jarak jauh saja.
Untuk menginstal komponen pendebugan jarak jauh, masukkan CD setup Visual Studio.NET dalam komputer target kemudian pilih link Remote Components Setup cli bagian bawah layar pembukaan, seperti gambar berikut ini'



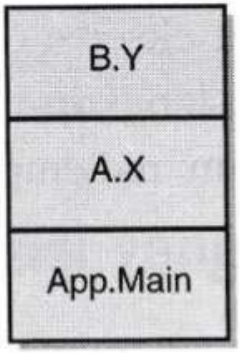
Pendebugan jarak jauh difasilitasi DCOM. Oleh karena itu, setelah menginstal komponen pendebugan jarak jauh, Kita harus mernastikan bahwa Kita memiliki izin memadai yang akan ditempelkan ke dan mendebug proses target pada mesin jarak jauh. Untuk meringkas proses ini, Setup membuat grup lokal pada mesin target yang disebLlt Debugger Users.

Pengguna yang ditambahkan ke grup Debugger Users akan memiliki izin yang memadai untuk melakukan sesi pendebugan interaktif jarak jauh. Namun, mereka juga membutuhkan izin untuk menempelkan ke proses itu sendiri. Secara default, proses pekerja ASP.NET yang menyimpan layanan Web akan jalan di bawah accoLrnt pengguna System. Dengan menambahkan diri Kita ke dalam grup Administrator lokal, Kita memiliki izin yang memadai untuk titempelkan ke dalam proses pekerja. .NET. Namun, jika hal ini tidak diizinkan oleh administrator sistem Kita, maka bicarakan dengannya cara mengkonfigurasi mesin agar diperoleh pembukaan pengamanan yang

dikehendaki.

Stack Panggilan yang Sesuai dengan Layanan Web

Stack panggilan adalah struktur data yang dipakai untuk menyimpan informasi panggilan tersarang yang dibuat oleh suatu thread di dalam aplikasi. Untuk setiap panggilan tersarang, bingkai stack dibuat dan ditambahkan ke stack panggilan. Misalnya, jika metode Main dalam suatu aplikasi memanggil metode X pada objek A dan sebaliknya metode X memanggil metode Y dari objek B, Kita akan mempunyai satu stack panggilan berisi tiga bingkai stack, seperti berikut ini:



Bingkai panggilan memberitahukan komputer cara mengembalikan kontrole sipemanggil setelah selesai mengeksekusi. Setelah tool metode kembali, bingkai stacknya turun dari stack panggilan.

Bingkai panggilan juga berisi parameter yang diteruskan ke metode dan ditambah data pemelihara lainnya. Visual Studio .NET dapat menangkap tiap bingkai stack pada stack tersebut untuk mengambil informasi tentang status terakhir suatu thread. Jika eksekusi suatu domain aplikasi ditangguhkan, maka

Kita dapat melihat informasi di bawah ini dari window Call Stack:

1. Nama modul yang berisi pelaksa naan metode.
2. Nama, tipe, dan nilai terakhir pada parameter metode.
3. Nomor baris yang sedang dieksekusi di dalam metode.
4. Bahasa yang dipakai oleh metode.

Dengan mengetahui urutan panggilan yang dilakukan, nilai tiap parameter yang diteruskan akan menjadi informasi penting saat Kita ingin mendebug aplikasi. .NET. Namun, jika salah satu metode memanggil layanan Web, kelanjutan stack panggilan akan terpotong. Karena dapat dieksekusi pada thread yang berbeda dan kemungkinan besar pada mesin yang berbeda, maka layanan Web akan memiliki stack panggilannya sendiri.

Visual Studio .NET mempermudah pendebugan aplikasi yang menjalankan banyak proses dan mesin dengan penyediaan tampilan berbagai stack panggilan yang membentuk satu thread eksekusi logis. Kita dapat melihat semua rantai panggilan dengan mengklik kanan dalam Window Call Stack Window kemudian memilih Include Calls to/From Other Threads.

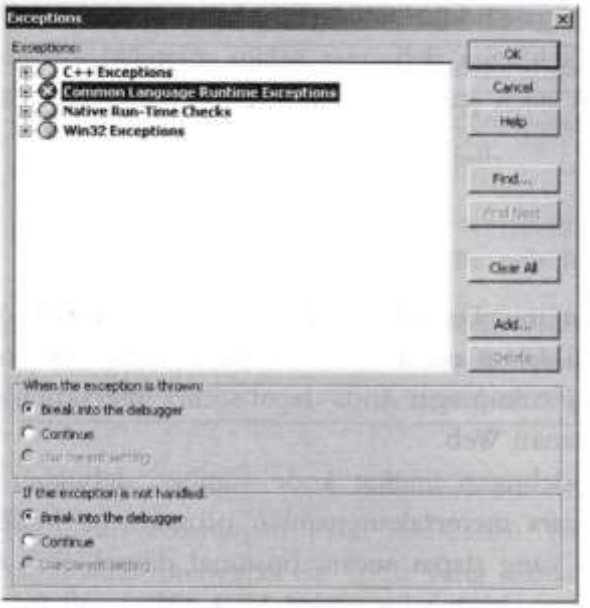
Untuk memastikan bahwa Kita punya stack panggilan yang lengkap, Kita harus melempar ulang eksepsi, tepat di dalam aplikasi Kita. Untuk melempar ulang sebuah eksepsi, Kita harus memanggil throw, tanpa parameter apa pun. Jika Kita meneruskan eksepsi sebagai parameter ke throw, maka stack akan mundur ke bingkai stack menuju metode yang melemparkan eksepsi dan penerima akhir eksepsi tidak akan mendapatkan jejak stack yang lengkap.

Berikut ini adalah contoh dua cara yang berlainan dalam melempar ulang eksepsi.

```
try
//Implementation.
```

```
catch(SystemExceptionse)
{
//Causes the stack to unwind to this method
call throwse;
catch(ApplicationExceptionae)
{
//The recipient of the exception will have a
full stack trace. throw;
```

Aplikasi Kita mungkin memakai kode yang tidak Kita miliki sumbernya dan salah melemparkan eksepsi. Untuk memfasilitasi perolehan jejak stack yang lengkap, Kita dapat mengkonfigurasi Visual Studio .NET untuk menangkap eksepsi yang pertama. Untuk membuka kotak dialog Exceptions, pilih Debug, Exceptions. Klik Common Language Runtime Exceptions, kemudian gunakan pilihan Break into The Debugger di dalam bagian When the Exception Is Thrown, seperti gambar berikut ini:



lnformasi yang Dibutuhkan Debugger

Debugger membutuhkan informasi tertentu untuk melakukan tugasnya, seperti mengatur breakpoint dan menampilkan stack panggilan. Informasi ini berasal dari tiga sumber utama: metadata yang ada di dalam assembly, database program, dan compiler JIT yang melacak informasi.

Dalam bagian ini, Penulis akan menjelaskan tipe informasi apa saja yang dibutuhkan debugger dan bagaimana memanfaatkan informasi itu. Penulis juga akan menjelaskan cara memastikan bahwa informasi tersebut tersedia untuk mendebug layanan Web. Terakhir, Penulis memberi rekomendasi untuk membuat bangun rilis dan debug untuk proyek layanan Web. Tujuan pembuatan bangun rilis adalah untuk membuat informasi yang dibutuhkan debugger, sehingga secara efektif mendiagnosa masalah yang mungkin muncul dalam lingkungan produk si.

Metadata Assembly

Dari metadata assembly. NET, debugger membutuhkan informasi tentang tipe yang didefinisikan di dalam assembly tersebut. Debugger memakai informasi ini untuk menampilkan nama-nama tipe yang mudah dipahami, metode yang dimunculkannya, dan nama tipe instance tipe-tipe tersebut, untuk menempati stack panggilan, window pengamat lokal, dan setemsnya. Metadata ini selalu ada di dalam assembly .NET, sehingga debugger selalu memiliki informasi yang cukup untuk menampilkan stack panggilan yang terdiri dari nama-nama yang dipahami.

Database Program

Beberapa fitur pendebugan membutuhkan informasi lebih banyak dari yang disediakan oleh metadata di dalam assembly. Misalnya, metadata assembly tidak mengandung informasi yang cukup agar Kita dapat secara interaktif menelusuri kode yang menerapkan layanan Web.

Untuk memfasilitasi pendebugan tingkat kode sumber, debugger membutuhkan informasi tentang cara memetakan gambar program ke kode sumber aslinya. Database program yang dapat secara opsional dibuat oleh compiler, mengandung pemetaan antara instruksi-instruksi MSIL (*Microsoft Intermediate Language*) di dalam assembly dengan baris-baris kode sumber yang dihubungkan.

Database program adalah file terpisah berekstensi .pdb dan biasanya memiliki nama yang sama sebagai file eksekusi (.dll atau .exe) yang jadi pasangannya. File .pdb sering satu direktori dengan file .dll atau .exe pasangannya.

File eksekusi dan file .pdb dibuat oleh compiler dan dianggap sebagai suatu pasangan yang tepat. Debugger tidak mengizinkan Kita memakai file .pdb yang lebih baru atau lebih lama daripada file eksekusi dan jala dalam proses target. Jika membuat file eksekusi dan file .pdb serta pasangannya, compiler akan memberi cap GUID pada keduanya yang akan dipakai debugger untuk memastikan bahwa file .pdb yang tepat telah dimuat.

Tidak ada mekanisme serupa untuk mengaitkan file .pdb dengan versi kode sumber yang menjadi sumber pembuatannya, sehingga mustahil untuk secara interaktif mendebug aplikasi Kita memakai versi kode sumber yang salah. Untuk mencegahnya, Kita harus menjaga kontrol versi yang ketat pada file eksekusi, file .pdb, dan kontrol sumber. Paling tidak, Kita harus mengecek

ketiga hal itu ke dalam database kontrol sumber sebelum menerapkan database pada mesin eksternal:

Compiler Visual C# (csc.exe) membuat file .pdb jika Kita menetapkan switch /debug. Tabel 7-1 menjelaskan semua variasi compiler switch /debug Visual C#.

Tabel 7-1 Switch-Switch Pendebugan Compiler Visual C#

Switch	Keterangan
/debug, /debug +, atau /debug.full	Menyatakan bahwa compiler akan membuat file .pdb
/debug	Menyatakan bahwa compiler tidak akan membuat file .pdb. Ini adalah pengaturan default
/debugpdbonly	Menyatakan bahwa compiler akan membuat file .pdb. Namun, pendebugan tingkat sumber secara default tidak aktif.

Dua item pertama pada tabel tersebut cukup ringkas. Item yang ketiga membutuhkan keterangan lebih mendalam. Dalam bagian berikutnya, Penulis akan membahas mengapa file .pdb yang dibuat oleh switch /debugpdbonly tidak dapat dipakai untuk pendebugan tingkat umber secara default.

Kita kemudian dapat memakai itch/optimize untuk menyatakan apakah kode Kita akan dioptimasi sebelum diekekusi. Secara default, optimasi tidak diaktifkan sama dengan menyatakan witch /optimize. Namun, akibatnya pada kinerja sangatlah besar.

Kita dapat mengaktifkan optimasi dengan pernyataan switch/optimize+. Namun, hal ini akan mengurangi ketepatan pendebugan kode sumber. Misalnya, kode dapat muncul untuk mengeksekusi di luar perintah atau tidak jalan sama sekali. Akibatnya, optimasi sering tidak diaktifkan saat pengembangan dan baru diaktifkan setelah aplikasi selesai.

Kita dapat menentukan apakah optimasi diaktifkan atau file .pdb akan dibuat untuk proyek Visual Studio .NET dengan memodifikasi pengaturan proyek Generate Debugging Information dan pengaturan Optimize Code di dalam kotak dialog Project Settings. Untuk membuka kotak dialog ini, pilih suatu proyek dalam Solution Explorer kemudian pilih Project, Properties, atau klik-kanan pada proyek lalu pilih Properties.

Visual Studio .NET akan secara otomati membuat dua konfigurasi untuk proyek Kita, yaitu Debug dan Release. untuk konfigurasi Debug, Generate Debugging Information diatur ke true dan Optimize Code diatur ke false. Untuk konfigurasi Release, Generate Debugging Information diatur ke false dan Optimize Code diatur ke true.

Kita akan melihat bahwa ternyata file .pdb sangat berguna untuk mendiagnosa masalah, terutama masalah yang muncul hanya dalam produksi. Penulis sangat menganjurkan Kita untuk membuat file .pdb bagi setiap assembly yang dirilis untuk produksi. Namun, sebelum Penulis memberi rekomendasi tentang pengaturan bangun tertentu, Penulis harus membuat gambaran yang lebih lengkap dulu.

lnformasi Pelacakan

Compiler Visual C# tidak membuat kode yang tidak dieksekusi dan dengan

demikian perlu didebug. Visual C# membuat MSIL, dan MSIL ini dikompilasi oleh compiler JIT ke kode asli sebelum dieksekusi oleh prosesor.

Jika Kita mendebug suatu layanan Web, maka Kita menempelkan debugger Kita ke proses yang mengeksekusi output dari compiler JIT. Oleh karena itu, compiler JIT memiliki pengaruh yang sama besarnya dengan compiler Visual C# terhadap kemampuan mendebug kode layanan Web secara interaktif.

Ingatlah bahwa database program yang dibuat oleh compiler Visual C# memetakan MSIL yang dibuatnya dengan kode sumber asli. Namun karena MSIL dikompilasi oleh compiler JIT sebelum dieksekusi, maka database program tidak mengandung informasi yang memadai untuk memfasilitasi pendebugan secara interaktif.

Untuk memfasilitasi pendebugan interaktif, debugger harus mampu memetakan kode asli yang eksekusinya di dalam proses dengan MSIL, lalu dengan kode sumber. Sebagian dari pemetaan ini, dari MSlL ke kode sumber, disediakan oleh file .pdb. Sebagian lagi, dari instruksi kode mesin asli ke MSIL, harus dibuat oleh compiler JIT pada saat runtime.

Pemetaan yang dibuat oleh compiler JIT disebut informasi pelacakan. Informasi pelacakan dibuat ketika MSIL dikompilasi ke kode asli oleh compiler JIT. Debugger memakai kombinasi dari informasi di dalam file .pdb dengan informasi pelacakan yang dibuat compiler JIT untuk mempermudah pendebugan interaktif kode sumber.

Dengan pelacakan yang tidak diaktifkan, Kita tidak dapat melakukan pendebugan tingkat sumber pada file eksekusi target. Jika kode sumber dikompilasi dengan memakai switch /debug, assembly yang dihasilkan akan diminta mengaktifkan pelacakan. Compiler JIT akan mempela jarinya karena

assembly tersebut dilengkapi oleh atribut Debuggable yang propertinya Is JIT TrackingEnabled diatur true. Jika compiler JIT memuat assembly, maka ia akar1 mencari atribut ini: nilai true untuk properti Is JIT TrackingEnabled akan menyingkirkan perilaku default.

Jika demikian, mengapa Kita harus repot-repot mengaktifkan pelacakan? Jika pelacakan aktif, akan berdampak pada kinerja saat aplikasi Kita dieksekusi. Terutama, pemanasan aplikasi agak lambat karena compiler JIT harus membuat informasi pelacakan di samping mengkompilasi MSIL ketika pertamakali metode dipanggil.

Setelah JIT dikompilasi, tidak ada lagi ongkos yang terkait dengan pelacakan. Oleh karena itu, dalam banyak kasus, manfaat dari dukungan pelacakan yang lebih baik untuk layanan Web akan menghilangkan ongkos yang terkait dengan pelacakan, terutama untuk layanan Web. Instance pada Web biasanya mendukung banyak permintaan dari banyak client, sehingga ongkos yang berkaitan dengan pembuatan informasi pelacakan akan cepat hilang.

Namun, dalam beberapa situasi, Kita mungkin ingin mengeluarkan ongkos untuk pelacakan, kecuali aplikasi tersebut mengalami masalah. Aplikasi Kita dapat dikompilasi dengan switch /debug. pdbondly, sehingga assembly yang dihasilkan akan mempunyai pasangan file .pdb yang dibuat untuk assembly tersebut tapi properti Is JIT TrackingEnabled pada atribut Debuggable tidak akan diatur true.

Perlu diingat bahwa Kita tidak boleh mengubah properti-properti pada bangun Visual Studio .NET untuk memanggil perilaku seperti yang dilakukan oleh switch /debug. pdbonly Jika Kita ingin membuat file .pdb dan tidak mengatur properti IsJIT TrackingEnabled di dalam assembly, Kita harus memakai cara

lain membangun aplikasi.

Jika Kita mencurigai adanya masalah dalam aplikasi yang dikompilasi dengan memakai switch /debug. pdbonly, Kita harus mengaktifkan pelacakan saat run- time. Dua cara utama untuk mengaktifk an pelacakan saat runtime adalah memakai debugger dan dengan mengkonfigurasi file.ini. Perlu diingat bahwa pada versi terakhir .NET, memodifikasi properti Is JITTrackingEnabled hanya akan berpengaruh saat aplikasi dimuat ulang oleh runtime bahasa umum. Kedua metode mengaktifkan pelacakan saat runtime tadi mengharuskan Kita merestart aplikasi Kita.

Metode pertama untuk mengaktifkan pelacakan saat runtime adalah dengan file .ini yang dipakai untuk mengatur pilihan pendebugan compiler JIT. File .ini harus memiliki nama yang sama dengan aplikasi dan harus disimpan dalam direktori yang sama. Misalnya, nama file .ini untuk MyRemotingWebService.exe adalah MyRemotingWebService.ini. lsi file .ini akan terlihat seperti berikut ini:

```
[.NET Framework Debugging Control] Generate
Trackinglnfo=l
AllowOptimize=O
```

Contoh tersebut mengkonfigurasi compiler JIT untuk membuat informasi pelacakan aplikasi. Seperti yang Kita lihat, Kita dapat memakai file .ini untuk mengontrol apakah compiler JIT membu at kode yang dioptimasi. Contoh ini melarang compiler JIT membuat kode asli yang dioptimasi.

Metode kedua untuk mengaktifkan pelacakan saat runtime adalah dengan debugger. Jika file eksekusi dijalankan di dalam debugger seperti Visual Studio.NET, debugger akan memastikan bahwa pelacakan aktif dan optimasi

tidak aktif. Kita dapat menjalankan file eksekusi dalam Visual Studio .NET dengan membuka proyek bertipe Executable Files (*.exe) yang ada. Pilih file eksekusi yang ingin Kita jalankan di dalam debugger. Ketika mulai mendebug, Kita akan diminta menyimpan file solusi Visual Studio .NET yang baru dibuat. Lalu, Visual Studio .NET akan menjalankan aplikasi dengan pelacakan aktif.

Kedua metode pelacakan saat runtime tersebut akan efektif untuk aplikasi .exe pada .NET, misalnya aplikasi yang menyimpan layanan Web Remoting dan client yang berinteraksi dengan layanan Web. Namun demikian, kedua metode itu tidak dapat dipakai untuk aplikasi yang disimpan oleh ASP.NET, terutama karena aplikasi ASP.NET disimpan dalam proses pekerja (aspnet_wp.exe). Proses pekerja ini tidak dapat dikelola dan menyimpan runtime-runtime bahasa umum. Proses penyimpanan runtime bahasa umum, misalnya ASP.NET, dapat secara program mengatur pilihan pendebugan untuk compiler JIT. Namun versi terakhir ASP.NET tidak menyediakan cara mengatur pilihan mendebug saat runtime, sehingga jika ingin secara interaktif mendebug layanan Web Dynamic Help hostting ASP.NET, Kita harus membangun komponen memakai pilihan /debug.

Untungnya, ongkos kerja yang terkait dengan pembuatan informasi pelacakan kurang relefan dalam hal layanan Web yang berhosting ASP.NET. Metode-metode yang dimunculkan oleh layanan Web cenderung dikompilasi sekali oleh JIT, setelah itu dieksekusi berkali-kali. Ongkos yang dikeluarkan dalam pembuatan informasi pelacakan menjadi kecil.

Penulis sarankan Kita mengkompilasi versi rilis layanan Web memakai switch /debug. Setelah kode Kita dikompilasi JIT, maka kinerja tidak akan terganggu

lagi. Dalam banyak kasus, kemampuan melakukan pendebugan tingkat sumber secara interaktif akan jauh mengur angi pengaruhnya pada kinerja saat pemanasan.

Jika overhead pelacakan menjadi masalah bagi layanan Web berhosting ASP. NET, maka Kita dapat mempertimbangkan untuk membangun dua versi rilis DLL, satu memakai /debug. pdbonly dan satu lagi memakaVdebug. Ahsan membangun file .pdb untuk kedua DLL tersebut adalah jika seKitainya versi runtime ASP.NET di masa depan memungkinkan Kita mengaktifkan pelacakan saat runtime.

Ringkasnya, Kita ham s mengkompilasi versi rilis aplikasi memakai switch/ optimize+. Optimasi yang dilaku kan oleh compiler JIT akan mengurangi ketepatan pendebugan interaktif kode umber. Namun demikian, ongkos pada kinerja karena optimasi tidak diaktifkan cukup besar dan dampaknya mempengaruhi seluruh masa hidup aplikasi.

Mendebug Kode Sumber yang Dikompilasi secara Dinamis

Perlu diingat bahwa pelaksanaan layanan Web juga dapat disimpan dalam file .asmx sendiri. Dalam hal ini, runtime ASP.NET menghasilkan MSIL. Kita hams memberitahu runtime ASP.NET untuk membuat informasi yang dibutuhkan supaya memfasilitasi pendebugan kode sumber secara interaktif.

Kita dapat mengaktifkan dukungan untuk pendebugan halaman .asmx tertentu, seluruh direktori, atau seluruh aplikasi. Hal ini akan mengakibatkan database program dan informasi pelacakan dibuat saat runtime. Selain itu, optimasi juga tidak diaktifkan.

Kita dapat mengaktifkan pendebugan tingkat halaman dengan mengatur

atribut Debug dalam direktif @Webservice. Berikut ini adalah contohnya:

```
<@WebServiceDebug"true"Language"Cfl"Class"My
WebService" »
using System;
using System.Web.Service;
public class MyWebService
WebMethod J
public string Hello( )
(
return"Helloworld.";
```

Kita dapat mengaktifkan pendebugan memakai file web.config. Kita dapat memakai file web .config untuk mengkonfigurasi file, baik yang ada di dalam suatu direktori atau dalam seluruh aplikasi bergantung pada tempatnya, seperti berikut ini:

```
<configuration>
<system.web>
<compilationdebug-"true"/>
<!system.web>
<configuration>
```

Mengaktifkan pendebugan juga berarti optimasi tidak aktif, sehingga layanan Web akan mengalami penurunan kinerja. Oleh karena itu, jika memungkinkan, pendebugan dalam produksi harus Kita matikan.

lnstrumentasi Layanan Web

Meskipun pendebugan tingkat sumber lebih hKital untuk mendebug aplikasi, tapi dalam banyak situasi cara ini tidak praktis. Misalnya, jika Kita secara interaktif mendebug layanan Web ASP.NET, maka Kita efektif memblokir semua thread dalam melayani permintaan lain. Hal ini tidak praktis jika layanan Web disimpan dalam lingkungan produksi dan Kita tidak mampu

mengisolirnya.

Dalam situasi seperti ini, instrumentasi akan sangat berguna. Instrumentasi adalah proses pembuatan output bagi pengembang atau administrator yang menyediakan informasi status jalannya layanan Web.NET Framework menawarkan pengembang dengan banyak pilihan instrumentasi layanan Web dan aplikasi yang memakainya. Dalam bagian ini, Penulis akan membahas tiga teknik yang dapat Kita manfaatkan untuk instrumentasi layanan Web, yaitu pelacakan, Event Log, dan counter (counter) kinerja.

Pelacakan

Pelacakan adalah proses merekam kejadian kunci dari suatu aplikasi yang dieksekusi selama periode yang terpisah. Informasi ini dapat membantu Kita dalam memahami jejak kode yang dilakukan dalam aplikasi. Informasi pelacakan dapat berisi informasi tentang perubahan yang dilakukan pada status aplikasi.

Tingkat pelacakan yang berbeda sering dibutuhkan untuk fase yang berbeda siklus hidup produknya. Misalnya, saat pengembangan, informasi ini dapat sangat panjang. Namun, ketika aplikasi dikirim, hanya subset pada informasi tersebut yang mungkin berguna.

Namespace System.Diagnostics berisi kelas Debug dan Trace yang menyediakan cara ringkas membuat informasi pelacakan dari aplikasi Kita. Kedua kelas ini menunjukkan perilaku yang serupa. Sebenarnya, secara internal keduanya meneruskan panggilan ke metode statis terkait yang dimunculkan oleh kelas pribadi TraceInternal. Perbedaan utama di antara

keduanya adalah bahwa kelas Debug harus dipakai saat pengembangan dan kelas Trace dipakai di seluruh siklus hidup aplikasi.

Tabel 7-2 menjelaskan properti dan metode yang dimunculkan oleh kelas Debug dan Trace.

Tabel 7-2 Properti dan Metode dari Kelas Debug dan Trace

Properti	**Keterangan**
AutoFlush	Menetapkan apakah metode Flush harus dipanggil setelah setiap kali menulis
Indentlevel	Menetapkan tingkat indentasi untuk penulisan
IndentSize	Menetapkan jumlah spasi di setiap indentasi
Listeners	Menetapkan koleksi pendengar yang memonitor output debug

Metode	**Keterangan**
Assert	Memberi nilai suatu ekspresi kemudian menampilkan stack panggilan dan pesan opsional yang ditentukan pengguna dalam kotak pesan jika ekspresi salah
Close	Membersihkan buffer output lalu menutup pendengar
Fail	Menampilkan stack panggilan dan pesan yang ditentukan pengguna di dalam kotak pesan
Flush	Membersihkan buffer output ke koleksi pendengar
Indent	Menambah satu nilai properti Indentl evel

Unindent	Mengurangi satu nilai properti IndentLevel
Write	Menuliskan informasi ke koleksi pendengar
Writeline	Menuliskan informasi dan linefeed ke koleksi pendengar
Writelinelf	Menuliskan informasi dan linefeed ke koleksi pendengar jika suatu ekspresi bernilai true

Setiap metode statis yang dimunculkan oleh kelas Debug dan Trace akan dilengkapi dengan atribut Conditional. Atribut ini mengontrol apakah panggilan ke suatu metode dieksekusi berdasarkan kehadiran simbol praproses tertentu.

Metode yang dimunculkan oleh kelas Debug dieksekusi hanya jika simbol DEBUG didefinisikan. Metode yang dimunculkan oleh kelas Trace dieksekusi hanya jika simbol TRACE didefinisikan.

Kita mendefinisikan simbol-simbol pada saat kompilasi dan Kita dapat mendefinisikannya di dalam kode sumber atau memakai switch compiler. Compiler ini akan membuat MSIL untuk memanggil metode yang dilengkapi atribut Conditional hanya jika simbol yang dibutuhkan itu didefinisikan. Misalnya, suatu panggilan ke Debug. WriteLine tidak akan dikompilasi ke dalam MSIL, kecuali jika simbol DEBUG didefinisikan.

Dengan Visual C#, Kita dapat memakai direktif #define untuk mendefinisikan simbol yang terbatas dalam file tertentu. Misalnya, kode berikut ini mendefinisi- kan simbol DEBUG maupun TRACE:

```
#define DEBUG
#define TRACE
```

Kita juga dapat mendefinisikan suatu simbol dengan compiler switch /define Visual C#. Simbol-simbol yang didefinisikan dengan cara ini akan terbatas

pada semua file kode sumber yang dikompilasi ke dalam file eksekusi. Perintah berikut ini mendefinisikan simbol DEBUG dan TRACE pada saat kompilasi :

```
csc/define:DEBUG;TRACE/target:libraryMyWebSe
rvicelmpl.cs
```

Simbol DEBUG dan TRACE didefinisikan saat Kita mengkompilasi bangun debug dan hanya simbol TRACE saja yang didefinisikan saat Kita mengkompilasi bangun rilis. Ini adalah default di dalam Visual Studio .NET. Kita dapat mengubah simbol mana yang didefinisikan saat kompilasi dengan mengkonfigurasi pengaturan proyek di bawah Configuration Properties, Build, kemudian Conditional Compilation Constants.

Sekarang, Kita sudah tahu cara mengatur simbol yang tepat. Mari kita lihat cara memakai beberapa metode kunci yang dimunculkan oleh kelas Debug dan Trace.

Menegaskan Kesalahan

Pengembang seringkali harus berusaha seimbang antara menulis kode yang besar dengan memaksimalkan kinerja aplikasi. Untuk menulis kode yang besar, Kita seringkali harus menulis kode dalam jumlah besar yang memberi nilai status aplikasi.

Kode validasi yang kaya akan sangat berguna untuk melacak permasalahan dengan cepat saat pengembangan. Namun, kode pengesahan yang telalu banyak dapat mempengaruhi kinerja aplikasi. Ringkasnya, layanan Web yang dimunculkan ke publik harus melakukan pengesahan parameter input yang diterima dari client. Namun, dalam keadaan tertentu tidak perlu melakukan

validasi variabel anggota yang dianggap sebagai detail pelaksanaan layanan Web.

Dalam kasus di mana lebih logis melakukan pengesahan hanya saat pengembangan, Kita dapat memakai metode Assert yang dimunculkan oleh kelas Debug dan Trace. Metode ini akan memberi nilai suatu ekspresi, dan jika ekspresi diberi nilai true, maka metode akan menghasilkan informasi tentang penegasan. Informasi kesalahan di antaranya adalah teks yang didefinisikan oleh aplikasi dan pengalihan stack panggilan.

Kemampuan untuk membuat informasi kesalahan secara program yang mencakup pengalihan stack panggilan tidak sulit untuk dilakukan. Mungkin Kita ingin melakukan hal ini dalam kode tertentu. Untuk melakukannya, Kita dapat memanggil metode Fail pada kela Debug dan Trace. Memanggil Fail sama dengan memanggil Assert di mana ekpresi selalu memiliki nilai false.

Mari kita lihat contohnya. Kode berikut ini menunjukkan penggunaan metode Assert dan Fail:

```
#define DEBUG
using
System.Web.Services;usingSystem.Diagnostics
:
public class Insurance
{
[WebMethod J
publicdoubleCalculateRate(intage,boolsmoker)
[
StreamReaderstream=File.OpenText("RateTable.
txt");Debug.Assert((stream.Peak( )==-1),
"Errorreading the rate table.",
"The rate table appears to be empty."):
```

```
try
[
//Implementation.
catch(Exceptionel
{
Debug Fail C "Unhandledexception.");throw;
```

Kode tersebut membuat penegasan jika file RateTable .txt kosong atau jika tertangkap eksepsi yang tak tertangani.

Karena metode Assert dan Fail dipanggil dari dalam layanan Web, ada satu persoalan dengan perilaku default kedua metode ini. Secara default, metode Assert dan Fail menampilkan kotak dialog jika ekspresi memberi nilai false. Namun, untuk kode serverside hal ini tentu saja tidak praktis. Kita dapat mengganti file web.config untuk mengarahkan output ke file log, seperti yang terlihat di bawah ini:

```
<configuration>
<system.diagnostics>
<assert assertuienabled-"false"
logfilename-"c:\Logs\Assert.log"/>
<!systemdiagnostics>

<!- - The rest of the
configurationinformation...- - >

</configuration>
```

Bagian dari file web.config ini menyatakan bahwa elemen assert harus mengganti perilaku default metode Assert dan Fail. Pertama, Penulis mengatur atribut assertuienabled dengan false untuk menyatakan bahwa suatu penegasan tidak boleh membuat kotak dialog modal ditampilkan. Penulis

kemudian menetapkan tempat file tersebut akan dituliskan memakai atribut logfilename. Penulis juga harus membuat direktori Logs dan memberi pengguna ASP. NET izin yang cukup untuk membuat dan menulis file Assert.log karena secara default, pengguna ASP.NET tidak diberi izin untuk menulis ke sistem file.

Akhirnya, perlu diingat bahwa perilaku default metode Assert dan Trace adalah untuk mengabaikan kesalahan dan melanjutkan. Oleh karena itu, jangan memakai metode Assert dan Fail sebagai pengganti untuk melemparkan eksepsi.

Direktif Praprosesor Kondisional

Perlu diingat bahwa atribut Conditional menyediakan cara untuk mendefinisikan metode yang harus dipanggil hanya jika simbol prapemrosesan didefinisikan. Namun, terkadang Kita ingin memiliki kontrol yang lebih ketat pada implementasi yang dikompilasi ke dalam aplikasi saat simbol prapernrosesan tertentu didefinisikan. Misalnya, Kita mungkin ingin memperluas rutin pengujian yang ditempelkan di dalam kode saat pengembangan. Kita dapat melakukan kontrol yang lebih ketat ini dengan menetapkan direktif prapernroses an kondisional di dalam aplikasi Kita.

Direktif prapemrosesan kondisional menKitai blok kode yang akan dikompilasi dalam MSIL hanya jika simbol tertentu didefinisikan. Tabel 7-3 menjelaskan kunci direktif prapernrosesan konditional yang dipakai untuk melakukan hal ini.

Tabel 7-3 Direktif Praprosesor Kondisional

Direktif	Keterangan
#if	Memulai blok kompilasi kondisional. Kode setelah direktif #if akan dikompilasi hanya jika kondisi bernilai true.
#else	Menetapkan pernyataan yang harus dikompilasi hanya jika kondisi yang disebutkan dalam direktif #if bernilai false.
#endif	Mengakhiri blok kompilasi kondisional.
#define	Mendefinisikan simbol prapemrosesan.
#undef	Membalikkan definisi simbol prapemrosesan.

Untuk layanan Web publik, biasanya tidak perlu mengembalikan jejak stack ke pengguna dalam kejadian eksepsi. Pelacakan stack menawarkan keuntungan minimal bagi pengguna eksternal layanan Web, lagi pula informasi yang disediakan oleh pelacakan stack dapat dipakai untuk mengamati keamanan yang rentan di dalam layanan Web Kita. amun, pada saat pengembangan, informasi tambahan ini dapat membantu pendebugan

Contoh berikut memakai direktif prapemrosesan kondisional untuk menghasilkan informasi pelacakan stack hanya jika aplikasi dikompilasi dengan simbol DEBUG yang didefinisik an:

```
#define DEBUG
using System.Web.Services;
using System.Web.Services.Protocols;

public class Insurance
(
```

```
[WebMethod J
public double
CalculateRate(intage,boolsmoker)
(
try

//Implementation.

catch(Exceptionel
(
#if DEBUG
thrownewSoapException
("Anunhandled exception was encountered.",
SoapException.ServerFaultCode,el;

#else
thrownewSoapException
("Anunhandled exception was encountered",
SoapException.ServerFaultCodel;
#endif
}
//Implementation.
```

Contoh tersebut melemparkan SoapException jika sebuah eksepsi yang tak ditangani tertangkap. Data yang dihasilkan di dalam SoapException bergantung pada simbol DEBUG didefinisikan atau tidak. Jika simbol DEBUG didefinisikan, sebuah instance baru dari kelas SoapException akan diinisialisasi dengan eksepsi yang tertangkap. Jika simbol DEBUG tidak didefinisikan, sebuah instance baru dari kelas tersebut hanya memiliki satu pesan kesalahan generik.

Log Pelacakan

Sampai sejauh ini, Penulis kebanyakan memfokuskan pada kondisi kesalahan. Namun demikian, memberi instrumen operasi aplikasi yang normal akan sama bergunanya. Kelas Debug dan Trace menyediakan serangkaian metode dan properti untuk membuat log informasi pelacakan di dalam aplikasi Kita.

Output klik dituliskan ke file log memakai metode Write, WriteLine, dan Write Linelf. Metode Write menghasilkan output teks pada file log, dan metode WriteLine menghasilkan output teks yang diikuti oleh linefeed. SeKitainya teks harus dituliskan ke dalam file log hanya jika kondisi tertentu terpenuhi, Kita dapat memakai metode WriteLinelf.

Kelas Debug dan kelas Trace juga memunculkan properti dan metode untuk mengontrol format output. Kita dapat memakai properti IndentLevel untuk mengatur berapa kali baris baru harus menjorok. Metode Indent dan Unindent masing-masing menambah dan mengurangi properti IndentLevel. Properti Indent Size menetapkan besarnya jarak spasi penjorokan.

Kita dapat menetapkan kapan buffer output akan diletakkan ke log pelacakan dengan memanggil metode Flush. Kita juga dapat mengatur properti AutoFlush dengan nilai true agar buffer output dibersihkan setelah dipindahkan ke log pelacakan.

Ingat bahwa kelas Debug dan Trace menangguhkan pelaksanaannya ke kelas Tracelnternal. Oleh karena itu, memodifikasi variabel statis memakai satu kelas akan berpengaruh pada kelas lain. Misalnya, mengatur setting Debug.IndentSize dengan nilai 4 juga mempengaruhi besarnya penjorokan kelas Trace.

Contoh berikut ini menunjukkan pemakai an metode pelacakan dalam konteks

layanan Web:

```
#define TRACE
using System.Diagnostics;using
System.Web.Services;
public class WebService
(
public WebService()
(
Trace.IndentSize 4;Trace.AutoFlush  true ;
[WebMethod
public void MyWebMethod(stringparam)
(
Trace.WritLine("MyWebMehod");Trace.Indent();
Trace.Writeline("Start:"++DateTime.ow);
//Implementation.
Trace.Writeline("End:" ++DateTime.Now);
Trace.Unindent();
```

Baik properti IndentSize maupun AutoFlush diatur di dalam konstruktor metode tersebut. Kita juga dapat mengaturnya saat runtime di dalam file web.config, seperti berikut ini:

```
<configuration>
<system.diagnostics>
<traceautoflush"true"indentsize"O"/>
<!system.diagnostics>
</configuration>
```

Kita dapat memakai elemen trace untuk mengatur nilai awal pada properti AutoFlush dan IndentSize. Setiap perubahan yang dilakukan aplikasi terhadap properti-properti tersebut akan menyingkirkan pengaturan default.

Kita perlu tahu tentang satu masalah saat Kita memanggil metode Write

Lineif. Perhatikan bagian kode berikut ini:

```
Trace.WriteLinelf(someCondition."Some
errormessage.",somelargeObject.ToString());
```

Karena teks yang akan dituliskan ke file log diteruskan ke metode WriteLineif, maka beberapa objek someLargeObject harus dibuat serialisasinya ke string, meskipun kondisinya memberi nilai false. Untuk mencegah pemrosesan yang tidak semestinya, kita dapat menulis ulang kode seperti berikut ini:

```
#if TRACE
if(someConditionl
{
Trace.Writeline("Some error
message.",somelargeO bject.ToString());
}
#endif
```

Objek someLargeObject akan diserialisasi ke string hanya jika variabel some Condition sama dengan true. Hal ini akan memastikan bahwa ongkos yang terkait dengan serialisasi someLargeObject hanya dikenakan jika teks yang dihasilkan akan dituliskan ke log pelacakan.

Pendengar Pelacakan

Kelas-kelas Debug dan Trace mendukung pengiriman output log pelacakan ke banyak pendengar. Suatu pendengar harus mewarisi kelas TraceListener. .NET Framework menyediakan tiga pendengar: Def aultTraceListener, EventLogTrace Listener, dan TextWriterTraceListener.

DefaultTraceListener ditambahkan ke sekelompok pendengar secara default. DefaultTraceListener menghasilkan output yang dapat ditangkap oleh debugger untuk kode yang terkelola maupun yang tidak. Informasi pelacakan

dikirim ke debugger kode terkelola lewat metode Debugger.Log dan dikirim ke debugger kode tak terkelola menggunakan API Win32 OutputDebugString. Jika memakai Visual Studio .NET, maka outputnya ditampilkan dalam window Output.

Kita dapat menambah ataupun menghapus pendengar dengan property Listeners pada kelas Debug dan Trace. Contoh berikut ini menghapus instance DefaultTraceListener dan menambahkan instance TextWriterTraceListener ke sekelompok pendengar:

```
//Remove instance of the
DefaultTracelistener.Debug.Listeners.Remove
(Debug.Listeners[0]);
//Add instance of the
TextWriterTracelistener.System.IO.FileStrea
mfs-
System.IO.File.OpenWrite(@"c:\Logs\Tracing.
log");Debug.Listeners.Add(new
TextWriterTraceLi stener(fs));
```

Kita juga dapat menambah atau menghapus pendengar saat runtime. Contoh berikut ini melakukan tugas yang sama dengan kode sebelurnnya, tapi dengan memodifikasi file web.config, seperti di bawah ini:

```
<configuration>
<system.diagnostics>
<trace>
<listeners>
<addname-"Text"
type-
"System.Diagnostics.TextWriterTracelistener
,System"initializeData-"c:\Logs\Tracing.log
"/>
```

```
<remove type-
"System.Diagnostics.DefaultTraceli
stener,System"/>
<!listeners>
</trace>
<!system.diagnostics>
<!configuration>
```

Dalam kedua kasus tadi, Kita harus membuat direktori Logs dan memberikan izin yang memadai kepada pengguna ASPNET untuk membuat dan menulis file Tracing.log karena secara default, izin yang dimiliki pengguna ASPNET tidak dapat menulis ke sistem.

Switch Pelacakan

Simbol prapemrosesan DEBUG dan TRACE memungkinkan Kita mengkonfigurasi tingkat pelacakan yang dibuat oleh aplikasi pada saat kompilasi. Namun kadang kala, Kita perlu tingkat pelacakan yang lebih ketat atau mengubah tingkat pelacakan saat runtime. Misalnya, Kita mungkin ingin menyimpan kesalahan dan peringatan hanya jika kondisi operasi normal. Namun jika muncul masalah, Kita ingin mengaktifkan pelacakan yang lebih panjang, tanpa harus mengkompilasi kembali kodenya.

Kita dapat memperoleh fungsionalitas ini menggunakan kelas yang mewarisi kelas Switch di dalam kode Kita. NET Framework menyertakan dua kelas seperti ini, BooleanSwitch dan TraceSwitch Kelas BooleanSwitch menyediakan satu mekanisme menyerupai kelas pemrosesan untuk menunjukkan apakah pelacakan harus diaktifkan. Namun demikian, Kita dapat menunjukkannya saat runtime dengan memodifikasi file konfigurasi aplikasi.

Misalnya, Penulis membuat sebuah instance pada kelas Boolean Switch yang memberi kemungkinan untuk mengontrol apakah informasi pelacakan ditampilkan di awal dan akhir metode.

```
using System;
using System.Diagnostics;
public class Application
{
privatestatic
BooleanSwitchprofileMethodsSwitch-new
BooleanSwitch("ProfileMethods","Controlswhe
therstart and endtimesaedisplayed for each
method.");
static public void Main(string[]args)J
{
Application.DoSomething("test",3);

privatevoidDoSomething(stringparaml,intparam
2)
(
Trace.Writelineif(profileMethodsSwitch.Enabl
ed,"StartDoSomething:"++DateTime.Now);

//Implementation...
TraceWritelineif(profileMethodsSwitch.Enable
d,"End DoSomething:"++DateTime.NowJ;
```

Penulis mendefinisikan BooleanSwitch untuk menentukan apakah informasi profil metode perlu dituliskan ke dalam log pelacakan. Mula-mula, Penulis membuat variabel statis bertipe BooleanSwitch dan mendefinisikan nama dan penjelasan switch tersebut di dalam konstruktor. Jika dipanggil, konstruktor switch itu akan membaca file konfigurasi aplikasi untuk menentukan nilainya (true atau false).

Kemudian, Penulis memakai profileMethodsSwitch sebagai kondisi pemanggilan ke WriteLineif yang menampilkan informasi profil metode. Perhatikan bahwa switch ini dapat dipakai oleh metode WriteLineif pada kelas Trace maupun Debug. Untuk masalah ini, switch tadi dapat ditentukan oleh pernyataan kondisional di dalam aplikasi.

Setelah switch didefinisikan, Kita dapat mengkonfigurasinya di dalam file konfigurasi aplikasi. File konfigurasi di bawah ini akan mengaktifkan switch ProfileMethods.

```
<configuration>
<system.diagnostics>
<switches>
<addname="TraceMethods"value=="l"II>
<!switches>
<!system.diagnostics>
<!configuration>
```

Penulis mengaktifkan switch TraceMethods dengan menyebutkan elemen add beserta atributnya, yaitu nanie dan diatur dengan string yang dipakai untuk menginisialisasi konstruktor switch tersebut. Jika switch TraceMethods tidak terdaftar di dalam file konfigurasi, nilai defaultnya akan diatur 0 atau false.

Jika ingin lebih pasti saat Kita mengkonfigurasi informasi pelacakan mana yang akan ditampilkan, Kita dapat memakai kelas TraceSwitch. Kita dapat mengatur instance pada kelas TraceSwitch dengan nilai numerik untuk menetapkan tingkat informasi pelacakan yang harus ditampilkan.

Kelas TraceSwitch mendukung lima tingkat pelacakan, dari 0 sampai 4. Tabel 7-4 menjelaskan tingkat-tingkat pelacakan tersebut.

Tabel 7-4 Properti-Properti dan Tingkat Pelacakan yang Terkait

Properti	Pelacakan	Keterangan
NIA	0	Pelaca kan tidak aktif.
TraceError	1	Hanya pesan kesalahan.
Trace Warning	2	Pesan kesalahan dan peringatan .
Tracelnfo	3	Pesan kesalahan, peringatan, dan informasi
Trace Verbose	4	Keterangan lengkap.

Aturlah instance pada kelas TraceSwitch dengan nilai kumulatif tertentu. Misalnya, jika nilainya diatur 3, buk an Tracelnfo saja yang diaktifkan, tapi juga TraceWarning dan TraceError.

Log Kejadian

Beberapa informasi pelacakan harus disimpan, apa pun pengaturannya pada switch atau apa pun simbol prapemrosesan yang didefinisikan. Misalnya, Kita harus melacak kesalahan kritis yang ditemui oleh layanan Web dan harus langsung ditangani oleh administrator sistem.

Informasi kritis tentang eksekusi sebuah aplikasi harus dituliskan ke Event Log. Event Log menyediakan penampungan umum untuk menyimpan kejadian-kejadian dari berbagai sumber. Secara default, sistem memiliki tiga log: Application Log, Security Log, dan System Log. Kejadian-kejadian yang ditimbulkan oleh aplikasi Kita, biasanya disimpan dalam Application Log.

Karena Event Log adalah komponen infrastruktur yang disediakan oleh sistem operasi, maka log ini disertakan dalam infrastruktur yang mendukungnya, atau Kita harus membuatnya sendiri. Misalnya, Event Log Service akan secara

otomatis mengontrol ukuran setiap log, sehingga Kita tidak perlu meringkas sendiri lognya. Kita dapat memakai Event Log Viewer untuk menampilkan dan mengurutkan entri-entri yang ada di dalam log. Kita juga dapat memperoleh tool yang mengoperasikan Event Log dan melaksanakan tugas-tugas seperti memberitahu administrator sistem jika ada aplikasi yang gagal.

Kita dapat memakai kelas EventLog untuk meletakkan pesan ke Event Log. Namun, sebelum menulis ke dalam Event Log, Kita terlebih dahulu harus mendaftarkan sumber kejadian. Sumber kejadian biasanya terkait dengan aplikasi Kita. Kode berikut ini menunjukkan cara mendaftarkan sebuah sumber kejadian:

```
if(!Eventlog.SourceExists C"MyWebService"))
{
Eventlog.CreateEventSourceC"My
WebService"."Application");
```

Kode di atas mula-mula memastikan apakah suatu sumber kejadian sudah didaftarkan atau belum. Jika belum, kode tersebut akan mendaftarkannya. Kemudian , Kita dapat menulis entri-entrinya ke dalam Event Log , seperti kode berikut ini:

```
Event log.WriteEntryC "My WebService",
"Unable toconnect to the database", E
ventlog Entry Type.Error);
```

Kode ini m nulis kejadian peringatan ke dalam Application Log. Tiga kategori kejadian adalah kejadian kesalahan, peringatan, dan informasional. Kita juga dapat memasukkan informasi tambahan bersama kejadian, di antaranya ID kategori dan ID kejadian yang didefinisikan aplikasi serta data biner mentah yang berguna saat Kita ingin mendiagnosa masalahnya.

Secara default, pengguna ASPNET tidak memiliki izin untuk menulis ke

dalam log kejadian. Untuk memberi izin, ubahlah pengaturan kunci registry:\ \ HKEY_ LOCAL_MACHINE \ SYSTEM \ CurrentControlSet \ Services\ Eventlog\Application \ RestrictGuestAccess menjadi 0 kemudian reboot mesin.

Secara default, pengguna ASPNET juga tidak memiliki izin untuk membuat sumber kejadian. Untuk mengatasi keterbatasan ini Kita dapat mendaftarkan sumber kejadian sebagai bagian dari prosedur instalasi pada layanan Web Kita. Jika Kita ingin mendaftarkan sumber kejadian saat runtime, pengguna ASPNET harus diberi izin read/write ke kunci registry \ \ HKEY_LOCAL _MA CHINE \ SYS- TEM\CurrentControlSet \ Services\ Eventlog beserta subkuncinya.

Kelas EventLog juga mendukung fungsi tambahan seperti menerima pemberitahuan ketika ada entri baru yang dibuat. Tabel 7-5 menjelaskan properti, metode, dan kejadian yang dimunculkan oleh kelas EventLog.

Tabel 7-5 Properti, Metode, dan Kejadian pada Kelas

Properti	Keterangan
EnableRaisingEvents	Menentukan apakah instance pada kelas Eventlog akan menerima pemberitahuan kejadian EntryWritten .
Entries	Mengambil kelas in pada kelas EventLogEntry.
Log	Menentukan nama log kejadian yang akan diakses.
LogDisplayName	Mengambil nama log kejadian yang mudah dipahami.
MachineName	Menentukan nama mesin tempat log kejadian

	target berada.
Source	Menentukan nama sumber kejadian yang dituliskan ke dalam log kejadian
Metode	Keterangan
Clear	Menghapus semua entri dari log kejadian target.
Close	Menutup handle pada log kejadian .
CreateEventSource	Mendaftarkan sumber kejadian baru di dalam registry sistem.
Delete	Menghapus log kejadian yang disebutkan.
DeleteEventSource	Menghapus pendaftaran sumber kejadian baru .
Exists	Menunjukkan apakah log kejadian yang disebutkan memang ada.
GetEventLogs	Mengambil array objek Eventlog dari mesin target.
LogNameFromSourceName	Mengambil nama Event Log yang terkait dengan sumber kejadian tertentu.
SourceExists	Menunjukkan apakah sumber kejadian yang disebutkan sudah terdaftar.
WriteEntry	Menulis entri ke log kejadian.
Entry Written	Timbul ketika kejadian dituliskan ke dalam Event Log pada mesin lokal

Counter Kinerja

Sampai saat ini, Penulis membatasi pembahasan metode instrumentasi pada bentuk komunikasi taksinkron. Aplikasi menuliskan data ke file teks atau

Event Log, kemudian client membuka sumber informasi dan membaca datanya. Namun demikian, kadang kala client ingin memonitor status aplikasi secara real-time.

Sebagai contoh, Penulis membuat layanan Web yang menerima order pembelian data pelanggan, Penulis mungkin ingin tahu jumlah permintaan perdetik yang diterima oleh layanan Web. Informasi seperti ini dapat diperoleh dengan counter kinerja.

Seperti yang sudah Kita ketahui, banyak aplikasi mempublikasikan banyak data menggunakan counter kinerja. ASP.NET juga tidak terkecuali. ASP.NET mempublikasikan banyak counter status runtime, termasuk jumlah aplikasi yang sedang berjalan dan jumlah proses pekerja yang jalan.

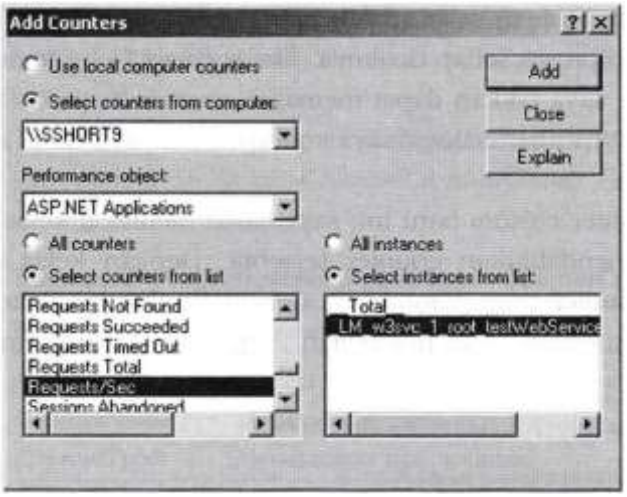
ASP.NET juga mempublikasikan banyak counter untuk status runtime masing-masing aplikasi yang ada di dalamnya. Counter-counter ini di antaranya jumlah permintaan per detik, jumlah permintaan dalam antrian, serta rata-rata waktu eksekusi permintaan.

Jika layanan Web yang baru Penulis jelaskan itu hanya menerima order pembelian, Penulis dapat memonitor jumlah permintaan yang diterima setiap detiknya, tanpa menulis sebaris kodepun. Penulis tinggal memakai aplikasi yang disertakan di dalam Windows bernama Performance Monitor. Langkah-langkah yang dibutuhkan untuk menjalankan Performance Monitor berbeda-beda, bergantung pada sistem operasi yang dipakai, atau lihat online help.)

Saat Performance Monitor jalan, Kita dapat menambahkan counter yang dibuat dalam bentuk grafik. Pertama, klik tombol bertKita plus untuk membuka kotak dialog Add Counters. Pilih ASP.NET Applications dari daftar drop-down Performance Object, kemudian pilih counter Requests/Sec. Lalu,

pilih instance yang terkait dengan aplikasi yang akan Kita monitor tersebut. Nama aplikasi tersebut akan terkait dengan nama direktori tempat aplikasi itu berada.

Kotak dialog Add Counters terlihat seperti di bawah ini:



Kita juga dapat membuat sendiri counter kinerja menggunakan kelas PerformanceCounterCategory dan kelas PerformanceCounter. Contoh berikut ini menunjukkan cara memakai kelas PerformanceCounterCategory untuk mendaftarkan sebuah counter kinerja baru:

```
if(!PerformanceCounterCategory
Exists("MyWebService"))
{
PerformanceCounterCategory.Create("My Web
Service","Performance counters published by
My Web Service.", "Total Purchase Orders
Processed",
"The total number of purchase orders
processed. ");
```

Kode di atas mendaftarkan sebuah kategori bernama My Web Service dan sebuah counter bernama Total Purchase Orders Processed jika kategori itu tidak ditemukan. Setelah counter itu didaftarkan, Kita dapat mempublikasikannya memakai sebuah instance pada kelas PerformanceCounter. Kode di bawah ini membuat sebuah counter kinerja dan menambah satu pada counter:

```
Performa nceCounter processedPOs=
new Performan ceCounter("My Web Service" ,
"Total Purchase Orders Processed",false);
processedPOs.Increment();
```

Penulis membuat instance pada kelas PerformanceCounter dan menginisialisasi instance tersebut agar dapat menulis ke dalam counter Total Purchase Orders Processed. Kemudian, Penulis menambah 1 pada counter dengan memanggil metode Increment pada objek.

Ini memang tepat, tapi tujuan Penulis adalah mempublikasikan angka rata-rata order pembelian yang diproses setiap detiknya. Jika layanan Web memunculkan lebih dari satu metode, Penulis takkan dapat memakai counter Requests/Sec yang dimunculkan oleh ASP.NET agar maksud Penulis itu terpenuhi. Penulis hams membuat sebuah counter custom.

Untuk membuat counter custom baru ini, Penulis hams memakai kelas Counter CreationData untuk mendaftarkan counter tersebut. Dengan kelas ini, Penulis dapat mengatur tipe counter yang dibutuhkan. Contoh berikut ini mendaftarkan counter untuk memonitor total order pembelian yang diproses dan jumlah yang diproses per detiknya.

```
if(!PerformanceCounterCategory.Exists("My
Web Service"))
{
```

```
CounterCreationData Collection counterCDC
new CounterCreation Data Collection();
counterCDC.Add(new
CounterCreationData("Purchase Orders
Processed/sec",
The number of purchase orders processed
persecond." ,
erformanceCounterType.RateOfCountsPerSecond
32ll; counterCDC.Add(new
CounterCreationData
("Total Purchase Orders Processed",
The total number of purcha e orders
processed.", Performance
CounterType.Number0fltems32ll;
Performance CounterCategory.Create("My Web
Service",
Performance counters published by My
WebService.",counterCDC);
```

Mula-mula, Penulis membuat instance pada kelas Counter Creation Data Collection yang akan dipakai untuk meneruskan counter yang akan didaftarkan. Kemudian, Penulis membuat dua instance pada kelas CounterCreationData untuk mendaftarkan counter. Perhatikan bahwa Penulis tidak perlu menulis kode apa pun untuk menghitung jumlah rata-rata permintaan order pembelian per detik. Ini akan ditangani oleh Performance Monitor.

Secara default, pengguna ASPNET memiliki izin untuk menulis ke dalam counter kinerja tertentu, tapi tidak dapat membuat kategori dan counter kinerja. Untuk mengatasi keterbatasan ini, Kita dapat mendaftarkan counter kinerja sebagai bagian dari prosedur instalasi pada layanan Web Kita.

Kadangkala, Kita mungkin ingin membuat counter kinerja saat runtime. Misalnya, Kita mungkin ingin mengasosiasikan sebuah instance pada counter kinerja dengan suatu instance dalam layanan Web atau mungkin dengan pengguna tertentu. Untuk mendaftarkan counter kinerja saat runtime, pengguna ASP.NET harus diberi izin read/write ke kunci registry \ \ HKEY_LOCAL_MACHINE \ SOFTWARE \ Microsoft \ Windows NT\ CurrentVersion\ Perflib beserta semu a sub- kuncinya.

Tabel 7-6, 7-7, dan 7-8 menjelaskan properti dan metode yang dimunculkan oleh kelas-kelas CounterCreationData, PerformanceCounter, dan PerformanceCounterCategory.

Tabel 7-6 Properti-properti pada kelas CounterCreationData

Properti	Keterangan
CounterHelp	Menetapkan string bantlian yang menj elaskan counter.
CounterName	Menetapkan nama counter.
CounterType	Menetapkan tipe counter

Tabel 7-7 Properti dan Metode pada Kelas PerformanceCounter

Properti	Keterangan
CategoryName	Menetapkan nama kategori di mana counter didaftarkan.
CounterHelp	Memanggil teks bantuan untuk counter.
CounterName	Memanggil teks bantuan untuk counter.
CounterType	Memanggil tipe counter.
InstanceName	Menetapkan nama instance yang terkait dengan counter.

MachineName	Menetapkan nama mesin yang terkait dengan counter.
Raw Value	Menetapkan nilai yang tak dihitung pada counter ini .
Read Only	Menetapkan apakah counter bersifat read-only.

Metode	Keterangan
Beginlnit	Dipakai oleh Visual Studio .NET untuk menjalankan inisialisasi counter.
Close	Menutup counter dan melepaskan semua sumber daya yang tertangkap.
Decrement	Mangurangi counter dengan satu di dalam operasi atomik.
Endlnit	Dipakai oleh Visual Studio .NET untuk mengakhiri inisialisasi counter.
Increment	Manambah counter dengan satu di dalam operasi atomik.
IncrementBy	Manambah counter dengan nilai yang disebutkan di dalam operasi Atomik.
NextSamp le	Memanggil nilai yang tak dihitung pada contoh counter.
Next Value	Memanggil nilai yang dihitung pada contoh counter.
Removelnstance	Menghapus instance kategori yang terkait dengan counter.

L

Tabel 7-8 Properti dan Metode pada Kelas PerformanceCounterCategory

Properti	Keterangan
CategoryHelp	Memanggil teks bantuan untuk kategori.
CategoryName	Menetapkan nama kategori.
MachineName	Menetapkan nama mesin tempat kategori berada.
CounterExists	Menyebutkan apakah counter spesifik terdaftar di bawah kategori tertentu.
Create	Mendaftarkan kategori dan satu atau lebih counter.
Delete	Menghapus kategori beserta counternya yang terdaftar.
Exists	Menunjukkan apakah suatu kategori sudah terdaftar.
GetCategories	Memanggil daftar kategori yang terdaftar.
GetCounters	Memanggil daftar kategori yang terdaftar untuk kategori tertentu.
GetlnstanceNames	Memanggil daftar instance untuk kategori tertentu.
InstanceExists	Menunjukkan apakah suatu instance pada kategori sudah terdaftar.
ReadCategory	Mengambil data instance yang terkait dengan tiap counter yang terdaftar di bawah kategori.

Tip dan Trik Mendebug

Beberapa informaasi yang dapat membantu Kita mendebug layanan Web tidak termasuk ke dalam bagian-bagian sebelumnya. Oleh karena itu, Penulis masukkan ke dalam daftar berikut ini:

1. Jika Kita memakai Microsoft Internet Explorer untuk menampilkan

dokumen yang secara otomatis dibuat oleh runtime (seperti WSDL serta hasil dari test harness ASP.NET), Kita harus mematikan pilihan untuk menampilkan pesan kesalahan yang mudah dipahami WSDL dan semua yang dihasilkan dari pesan kesalahan yang mudah dipahami dari tampilan. Dengan demikian, Kita dapat melihat pesan kesalahan aktual yang dihasilkan. Ikuti saja langkah-langkah berikut ini:

2. Pada Internet Explorer, pilih Tools, Internet Options.
3. Pada tab Advanced tab dari kotak dialog Internet Options, hilangkan tKita pada Show Friendly HTTP Error Messages.
4. Jika Internet Explorer menampilkan halaman kosong, buka sumbernya. Kadang kala, suatu pesan kesalahan yang dihasilkan oleh runtime tidak di- tampilkan di dalam browser. Untuk melihat pesan kesalahan tersebut, tampilkan kode sumbernya dengan mengklik menu View lalu pilih View Source.
5. Jika Kita mendebug suatu layanan Web yang diakses melalui proxy, Kita harus menambah nilai timeout proxy dengan angka yang besar. Untuk proxy ASP.NET yang bersumber dari kelas SoapHttpClientProtocol atau untuk proxy yang dibungkus Remoting yang bersumber dari kelas RemotingClient Proxy, aturlah seting property Timeout dengan nilai -1 (infinity). Sebelum Kita merilis aplikasi client untuk produksi, yakinkan bahwa Kita sudah mengatur kembali nilai timeout ke nilai yang dapat diterima.
6. Pada Visual Studio .NET Kita dapat mendebug banyak tipe kode di dalam suatu aplikasi, termasuk ASP, ASP. NET, kode takterkelola, dan SQL Server yang disimpan dalam prosedur. Pastikan bahwa Kita sudah

melakukan pilihan debug yang tepat pada pengaturan proyek Kita. Misalnya, jika Kita mempunyai suatu layanan Web ASP.NET yang memanggil kode takterkelola dan Kita ingin mendebug seluruh pelaksanaan layanan Web, pastikan bahwa Kita sudah mengaktifkan dukungan pendebugan untuk ASP.NET, maupun untuk kode tak terkelola.

C. PENUTUP

Ringkasan

Bab ini membahas pendebugan kode sumber secara interaktif, informasi yang dibutuhkan oleh debugger, dan bagaimana memberi instrumen pada aplikasi. Pertama, Penulis menjelaskan beberapa fitur dasar debugger Visual Studio .NET yang membantu mempermudah tugas pembuatan layanan Web. Salah satu syarat yang unik untuk mendebug layanan Web adalah dukungan yang kuat untuk mendebug dari jarak jauh. Fitur kunci yang disediakan Visual Studio .NET dalam mendukung pendebugan jarak jauh di antaranya:

1. Visual Studio .NET secara otomatis menempel pada proses ASP.NET jarak jauh yang merupakan hosting bagi layanan Web.
2. Dengan Visual Studio .NET Kita dapat mengkonfigurasi server target, sehingga dapat dilakukan pendebugan jarak jauh.
3. Visual Studio .NET dapat menampilkan stack panggilan logis yang beredar di antara thread-thread.
4. Visual Studio.NET memastikan bahwa Kita menerima stack panggilan secara lengkap jika eksepsi yang tak tertangani timbul di dalam aplikasi.

Kemudian, Penulis menjelaskan informasi seperti apa yang dibutuhkan debugger agar melakukan tugas-tugas penting. Terutama, debugger

membutuhkan informasi untuk membuat stack panggilan yang dapat dibaca. Informasi tersebut terkandung di dalam metadata dalam modul berisi tipe-tipe yang membentuk stack panggilan tersebut.

Pendebugan kode sumber secara interaktif mengharuskan pemetaan antara kode sumber dengan kode mesin yang dibuat oleh compiler JIT. Setengah bagian dari pemetaan ini, yaitu antara kode sumber dengan MSIL, disediakan oleh file database program (.pdb). Setengah bagian lagi, antara MSIL dengan kode mesin asli, disediakan oleh informasi pelacakan yang dibuat oleh compiler JIT.

Informasi pelacakan dibuat ketika MSIL dikompilasi oleh JIT ke dalam kode asli. Karena kode yang dikompilasi tersebut tidak terpengaruh oleh pembuatan informasi pelacakan, sedikit dampak oleh pelacakan pada kinerja akan terjadi hanya saat pemanasan aplikasi.

Kita dapat menetapkan apakah compiler JIT membuat kode yang dioptimasi atau tidak. Jika optimasi diaktifkan, Kita dapat kehilangan kehKitalan antara kode mesin yang dikompilasi dengan kode sumber asli. Karena mengorbankan kinerja dalam jurnlah besar akibat pembuatan kode mesin yang tidak dioptimasi, maka Penulis sarankan Kita mengaktifkan optimasi untuk bangun rilis.

Akhirnya, Penulis menjelaskan berbagai teknologi yang disediakan oleh .NET untuk instrumentasi layanan Web Kita dan aplikasi client yang berinteraksi dengannya. Penulis menjelaskan persamaan dan perbedaan antara kelas Debug dan kelas Trace serta menunjukkan bagaimana caranya menambah dan menghapus pendengar pada saat kompilasi dan saat berjalan.

Selain itu, Penulis juga menerangkan cara memakai Event Log untuk

mengomunikasikan informasi yang penting ke administrator sistem, serta menunjukkan cara memakai counter kinerja untuk mempublikasikan informasi real-time tentang status terakhir aplikasi.

Pertanyaan

1.Informasi apa saja yang dibutuhkan oleh seorang Debugger?

2.Sebutkan dan jelaskan properti dan metode dari kelas debug dan trace.

3.Apa fungsi dari Log Pelacakan?

4.Sebutkan tip dan trik dalam melakukan debug?

DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.

9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 8

Mengapa XML

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 8 akan di bahas XML (Extensible Markup Language) diantaranya: Kebutuhan Akan XML, Solusi XML, Menulis Dokumen XML, Menampilkan Dokumen XML, SGML, HTML dan XML, Apakah XML Menggantikan HTML, Sasaran Resmi XML, Aplikasi XML Sekitar, Aplikasi XML untuk Meningkatkan Dokumen XML, Penggunaan XML dalam Pekerjaan Nyata.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan Kebutuhan Akan XML, Solusi XML, Menulis Dokumen XML, Menampilkan Dokumen XML, SGML, HTML dan XML, Apakah XML Menggantikan HTML, Sasaran Resmi XML, Aplikasi XML Sekitar, Aplikasi XML untuk Meningkatkan Dokumen XML, Penggunaan XML dalam Pekerjaan Nyata.

B. PENYAJIAN

Mengapa XML

XML, yang merupakan singkatan dari *Extensible Markup Language*, didefinisikan oleh XML Working Group pada *World Wide Web Consortium* (W3C). Kelompok ini menjelaskan bahasa ini sebagai berikut:

"Extensible Markup Language (XML) adalah sebuah subset dari SGML.

Sasarannya adalah memungkinkan SGML generik dapat dilayani, diterima, dan diproses pada Web dengan cara yang sekarang dipakai untuk HTML. XML dirancang untuk kemudahan implementasi dan untuk interoperabilitas dengan SGML dan HTML."

Ini merupakan kutipan dari versi 1.0 pada spesifikasi resmi XML, yang diselesaikan oleh XML Working Group pada bulan Februari 1998. Kita bisa membaca seluruh dokumen pada situs Web W3C di http://w3.org/TR/REC xml.

Seperti yang kita lihat, XML adalah bahasa markup yang dirancang khusus untuk penyampaian informasi melalui World Wide Web, persis seperti HTML (*Hypertext Markup Language*), yang telah menjadi bahasa sekitar untuk membuat halaman Web sejak awal kehadiran Web. Berhubung kita telah memiliki HTML, yang dikembangkan untuk memenuhi setiap kebutuhan yang ada, Kita mungkin bertanya-tanya mengapa kita membutuhkan sebuah bahasa baru untuk Web. Apa yang baru dan yang berbeda dengan XML? Apa kelebihan khususnya dan kekuatannya? Apa hubungannya dengan HTML? Apakah XML dimaksudkan untuk menggantikan HTML atau untuk meningkatkannya? Dan terakhir, apakah yang dimaksud SGML-nya di mana XML menjadi subset daripadanya, dan mengapa kita tidak bisa hanya menggunakan SGML untuk halaman-halaman Web? Dalam bab ini Penulis akan berusaha menjawab semua pertanyaan ini.

Kebutuhan Akan XML

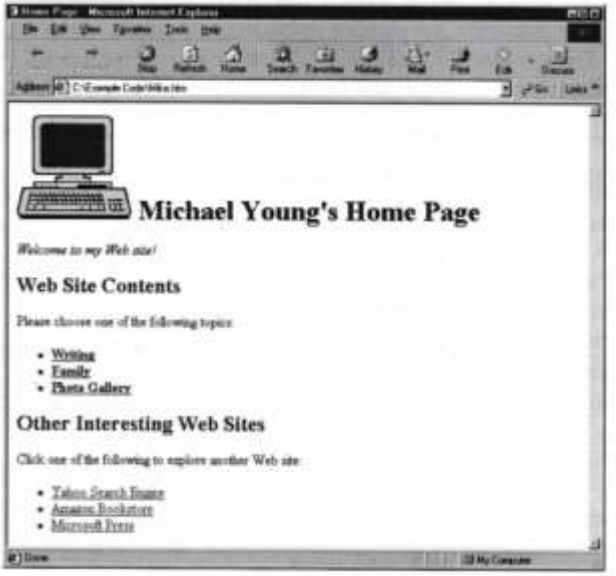
HTML memberikan sekumpulan elemen baku yang bisa Kita gunakan untuk mengitai komponen-komponen halaman Web serbaguna pada umumnya.

Contoh-contoh elemen-elemen tersebut berupa heading, paragraf, daftar, tabel, citra, dan link. Sebagai contoh, HTML bekerja dengan baik untuk membuat sebuah homepage pribadi, seperti dalam contoh berikut ini:

```
<HTML>
<HEAD>
<TITLE>Home Page<ITITLE>
<! HEAD>
<BODY>
<Hl><IMGSRC="MainLogo.gif">Michael Young's
Home Page</Hl>
<P><EM>Welcome to my Website! </EM><IP>
<H2>Web Site Contents</H2>
<P>Please choose one of the following
topics:</p>
<UL>
<LI><AHREF="Writing.htm"><B>Writi></A></LI>
<LI><AHREF="Family.htm"><B>Family</B</A>
</LI>
<LI><AHREF="Photos.htm"><B>PhotoGallery</B>
</A></LI>
<UL>
<H2>0ther Interesting Web Sites</H2>
<P>Click one of the following to explorea
nother Web site:</P>
<U L>
<LI>
<AHREF="http://www.yahoo.com/>yahoo Search
Engine</A>
</LI>
<LI>
<AHREF="http://www.amazon.com/>amazon
Bookstore</A>
```

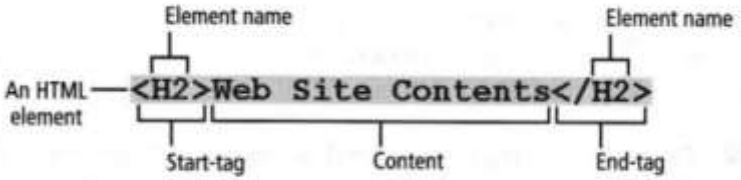
```
</LI>
<LI>
<AHREF="http://mspress
microsoft.com/">Microsoft Press</A)
</LI>
</UL>
</BODY>
</HTML>
```

Microsoft Internet Explorer 5 menampilkan halaman ini, seperti terlihat dalam gambar berikut:



Masing masing elemen dimulai dengan sebuah tag-awal sebuah blok teks yang didahului dengan tKita kurung siku kiri (<) dan diikuti dengan sebuah kurung siku kanan (>) yang berisi nama elemen dan mungkin pula informasi lain. Sebagian besar elemen diakhiri dengan tag-akhir, yang seperti pasangannya

tag-awal hanya saja mencantumkan sebuah karakter garis-miring (/) yang diikuti dengan nama elemen tersebut. Isi elemen adalah teks jika ada antara tag-awal dengan tag-akhir. Perhatikan sejumlah elemen dalam halaman contoh sebelumnya berisi elemen-elemen tersarang.



Halaman HTML contoh berisi elemen-elemen berikut:

Elemen HTML	Komponen halaman yang ditKitai
HTML	Seluruh halaman
HEAD	Informasi heading, seperti halnya judul halaman
TITLE	Judul halaman, yang muncul dalam baris judul browser
BODY	Tubuh utama dari teks yang ditampilkan browser
Hl	Heading tingkat teratas
H2	Heading tingkat kedua
p	Paragraf teks
UL	Daftar yang ditKitai (daftar tak berurutan)
LI	Item individual dalam sebuah daftar (List Item)
IMG	Citra
A	Link ke lokasi lain atau halaman lain (elemen Anchor)
EM	Blok dari teks miring (penekanan)
B	Blok teks tebal

Browser yang menampilkan halaman HTML mengetahui setiap elemen sekitar ini, dan tahu bagaimana memformat dan menampilkan mereka. Sebagai contoh, browser umumnya menampilkan sebuah heading Hl dalam huruf terbesar, heading H2 dalam huruf yang lebih kecil, dan sebuah elemen P dalam huruf yang lebih kecil lagi. Browser menampilkan elemen L1 dalam daftar takberurutan dengan bullet, dengan paragraf menjorok ke dalam, dan mengubah elemen A ke dalam sebuah link bergaris bawah yang bisa diklik user untuk pergi ke lokasi atau halaman lain.

Walaupun serangkaian elemen HTML terdefinisi telah ditambahkan sejak versi HTML yang pertama, HTML masih tidak cocok untuk mendefinisikan sejumlah tipe dokumen. Berikut ini adalah contoh-contoh dokumen yang tidak cukup dijelaskan dengan HTML:

1. Dokumen yang tidak berisi komponen biasa (heading, paragraf, daftar, tabel, clan seterusnya). Sebagai contoh, HTML kekurangan elemen penting untuk menandai skor musikal atau sekumpulan persamaan matematik.
2. Database, seperti halnya buku inventaris. Kita bisa menggunakan halaman HTML untuk menyimpan dan menampilkan informasi database yang statis (seperti halnya sebuah daftar penjelasan buku). Jika Kita ingin mengurutkan, menyaring, mencari, dan memakai informasi tersebut dengan cara lain, masing-masing potongan informasi individual perlu ditKitai (seperti terdapat dalam program database seperti Microsoft Access). HTML kekurangan elemen-elemen yang diperlukan untuk melakukannya.
3. Dokumen yang ingin Kita tata dalam bentuk hierarki pohon. Anggaplah,

sebagai contoh, Kita sedang menulis sebuah buku dan ingin mengitainya menjadi bagian, bab, bagian A, bagian B, bagian C, dan seterusnya. Sebuah program kemudian bisa memakai dokumen terstruktur ini untuk membuat sebuah tabel isi, untuk menghasilkan garis-besar dengan tingkat uraian yang beragam, untuk mengutip bagian-bagian tertentu, dan untuk memakai informasi tersebut dalam cara lain. Sebuah elemen heading HTML hanya mengitai teks pada heading itu sendiri. Misalnya:

```
<H2>Web Site Contents</H2>
```

Karena Kita tidak menyarangkan teks sesungguhnya dan elemen- elemen yang dimiliki sebuah bagian dokumen dalam suatu elemen heading, elemen-elemen ini tidak bisa dipakai untuk mengitai secara jelas struktur hierarki dokumen tersebut.

Solusi keterbatasan ini adalah XML.

Solusi XML

Definisi XML hanya berisi sintaks dengan kerangka polos. Saat Kita membuat sebuah dokumen XML, ketimbang menggunakan elemen-elemen yang telah ada, Kita membuat sendiri elemen Kita dan menamai mereka sesuka Kita seperti istilah extensible dalam Extensible Markup Language. Karenanya, Kita bisa menggunakan XML untuk menjelaskan secara virtual berbagai tipe dokumen, dari skor musik hingga database.

Sebagai contoh, Kita bisa menjelaskan sebuah daftar buku, seperti dalam dokumen XML berikut ini.

```
<? xml version=no 1.0"?>
<INVENTORY>
<BOOk>
```

```
<TITLli>The Adventures of Huckle berry Fjnn
</TITLE>
<AUTHOR>Mark Twain</AUTHOR>
<BINDING') mass market paper back</BINDING>
<PAGES>298</PAGES>
<PRICE>$5.49</PRICE>
</BOOK>
<BOOK>
<TITLE>Mobyt Di, ck</TITLE>
<AUTHOR>Herman Melville</AUTHOR>
<BINDING)trade paperback</BINDING>
<PAGliS>fr05</PAGliS>
<PRICE>$4.95</PRICE>,
</BOOK>
<BOOK>
<TITLE>The    Scarlet   Letter</TITLE>
<AUTHOR>Nathaniel   Hawthorne</AUTHOR>
<BINDING>trade paperback</BINDING>
<PAGES>253</PAGES>
<PRICE>$.4.2.5</PRLCE>
</BOOK>
</INVENTORY>
```

CATATAN Bila dipakai untuk menguraikan sebuah database, XML memiliki sepasang manfaat melalui format yang sesuai (seperti Access.mdb atau dBase .dbf): XML dapat dibaca oleh manusia, dan XML berdasarkan pada sekitar publik yang terbuka.

Perlu diingat bahwa nama nama elemen dalam sebuah dokumen XML (seperti halnya INVENTORY, BOOK, dan TITLE, pada contoh ini) bukanlah bagian dari definisi XML. Kita membuat nama-nama tersebut saat Kita membuat

sebuah dokumen tertentu. Kita bisa memilih nama yang valid untuk elemen tersebut (seperti misalnya LIST ketimbang menggunakan INVENTORY, atau ITEM ketimbang BOOK).

TIP Saat Kita menamai sebuah dokumen XML, cobalah pilih nama yang deskriptif. Sebagai contoh, BOOK atau ITEM, ketimbang menggunakan FOO atau BAR. Salah satu keuntungan dokumen XML adalah dapat menjelaskan dirinya sendiri yakni, masing- masing potongan informasi dapat memiliki label deskriptif yang disertakan.

Seperti yang bisa Kita lihat dari contoh sebelumnya, sebuah dokumen XML berstruktur seperti hierarki pohon, dengan elemen-elemen yang tersarang sepenuhnya dalam elemen-elemen lain, dan dengan sebuah elemen tingkat teratas (INVENTORY dalam contoh ini) <dikenal sebagai elemen dokumen atau elemen root-yang berisi semua elemen lainnya. Struktur contoh dokumen XML bisa digambarkan seperti ini:

Kita dengan demikian telah siap menggunakan XML untuk mendefinisikan

sebuah dokumen berstruktur hierarki. Seperti halnya sebuah buku dengan bagian-bagian, bab-bab, dan berbagai tingkat bagian, seperti dimaksudkan sebelumnya.

Menulis Dokumen XML

Karena XML tidak memasukkan elemen yang telah ada, XML tampaknya menjadi biasa secara relatif XML memiliki sintaks yang didefinisikan langsung. Misalnya, tidak seperti HTML, setiap elemen XML harus memiliki sepasang tag-awal dan tag -akhir (atau tag elemen-kosong khusus, yang akan Penulis jelaskan dalam bab-bab berikutnya). Berbagai elemen yang tersarang harus sepenuhnya termasuk dalam elemen yang menutupnya.

Pada kenyataannya, keluwesan dalam pembuatan elemen Kita sendiri membutuhkan sebuah sintaks langsung. Itulah sebabnya sifat alami dokumen XML membutuhkan program tersendiri untuk menangani dan menampilkan informasi yang terkandung di dalamnya. Sintaks XML baku memberikan dokumen XML sebuah form yang bisa diperkirakan, dan menjadikan program ini lebih mudah ditulis. Mengingat kembali kutipan dari awal bab ini bahwa "kemudahan implementasi" adalah salah satu sasaran utama bahasa ini.

Bagian ini membahas pembuatan dokumen XML yang tunduk pada aturan penulisan. Seperti yang akan Kita pelajari, Kita bisa menulis sebuah dokumen XML yang tunduk pada dua tingkat kebakuan penulisan yang berbeda. Dokumen tersebut dikenal sebagai "well-formed" atau "valid " bergantung pada tingkatan mana yang dipenuhinya.

Menampilkan Dokumen XML

Dalam sebuah halaman HTML, browser tahu bahwa elemen Hl, misalnya, adalah heading tingkat-teratas, dan akan memformat dan menampilkannya dengan tepat. Ini dimungkinkan karena elemen ini adalah bagian HTML. Namun bagaimana bisa sebuah browser atau program lain tahu cara menangani, atau menampilkan elemen dalam sebuah dokumen XML yang Kita buat (seperti halnya BOOK atau BINDING dalam contoh dokumen), karena Kita membuat sendiri elemen Kita?

Ada tiga cara mendasar untuk memberitahu browser (khususnya, Microsoft Internet Explorer 5) bagaimana menangani dan menampilkan setiap elemen XML Kita.

1. Pengaitan style sheet. Dengan teknik ini, Kita mengaitkan sebuah style sheet ke dokumen XML. Style sheet adalah sebuah file terpisah yang berisi instruksi pemformatan elemen XML individual. Kita bisa menggunakan juga style sheet bertumpuk (CSS, Cascading Style Sheet) yang juga dipakai untuk halaman HTML atau sebuah style sheet Extensible Stylesheet Language yang dianggap lebih ampuh dibandingkan CSS, dan dirancang khusus untuk dokumen XML.
2. Pengikatan data. Opsi ini mengharuskan Kita membuat sebuah halaman HTML, mengaitkan dokumen XML padanya, dan mengikat elemen XML dalam halaman tersebut, seperti misalnya elemen SPAN atau TABLE, ke elemen XML. Elemen HTML kemudian secara otomatis menampilkan informasi tersebut dari elemen XML yang diikatkan padanya.
3. Scripting. Dengan teknik ini, Kita membuat sebuah halaman HTML, mengaitkan dokumen XML padanya, dan mengakses serta menampilkan

elemen XML individual dengan menuliskan kode script (JavaScript atau Microsoft Visual Basic Scripting Edition [VBScript]). Browser menampilkan dokumen XML sebagai sebuah Document Object Model (DOM), yang menyediakan sekelompok besar obyek, properti, dan metode yang dapat dipakai kode script untuk mengakses, memanipulasi, dan menampilkan elemen XML.

SGML, HTML, dan XML

SGML, yang artinya Structured Generalized Markup Language, adalah ibu dari semua bahasa markup. Baik HTML maupun XML dikembangkan dari SGML (walaupun secara fundamental caranya berbeda). SGML mendefinisikan sebuah sintaks dasar, namun memungkinkan Kita membuat elemen Kita sendiri (itulah makanya disebut generalized). Untuk menggunakan SGML dalam menjabarkan dokumen tertentu, Kita harus membuat seperangkat elemen yang sesuai dan sebuah struktur dokumen. Misalnya, untuk menjelaskan sebuah buku, Kita bisa menggunakan elemen yang Kita namai BOOK, PART, CHAPTER, INTROD UCTION, A-SECTION, B-SECTION, C-SECTION, dan seterusnya.

Keserbagunaan sekumpulan elemen yang dipakai untuk menjelaskan tipe dokumen tertentu dikenal sebagai sebuah aplikasi SGML (aplikasi SGML juga termasuk aturan yang menentukan cara bagaimana elemen disusun sebagaimana fitur lainnya menggunakan teknik yang akan kita diskusikan). Kita bisa mendefinisikan sendiri aplikasi SGML Kita untuk melukiskan jenis dokumen tertentu yang Kita pakai, atau sebuah tubuh bisa mendefinisikan aplikasi SGML untuk menjelaskan tipe dokumen yang dipakai secara luas.

Contoh paling terkenal dari jenis aplikasi ini adalah HTML, yang merupakan sebuah aplikasi SGML yang dikembangkan pada tahun 1991 untuk menggambarkan halaman Web.

SGML tampaknya menjadi bahasa luas yang sempurna untuk menggambarkan dokumen Web. Anggota W3C yang membuat materi ini menganggap SGML terlalu rumit, dan sulit diatur untuk mengirimkan informasi pada Web dengan efisien. Fleksibilitas dan kelebihan fitur yang disediakan oleh SGML akan menyulitkan dalam menulis perangkat lunak yang dibutuhkan untuk memproses dan menampilkan informasi SGML dalam browser Web. Untuk itu diperlukan sebuah subset SGML ramping yang dirancang khusus untuk menyampaikan informasi pada Web. Pada tahun 1991, XML Working Group dari W3C mengembangkan subset tersebut, yang dinamai Extensible Markup Language. Seperti kutipan pada awal bab menyatakan, XML dirancang untuk "kemudahan implementasi, " sebuah fitur yang jelas tidak ada dalam SGML.

XML merupakan sebuah versi penyederhanaan SGML yang ditingkatkan bagi Web. Seperti pada SGML, XML memungkinkan Kita mengembangkan sendiri elemen yang dibutuhkan saat Kita menguraikan sebuah dokumen tertentu. Juga seperti SGML, sebuah tubuh atau individual bisa mendefinisikan sebuah aplikasi XML (juga dikenal sebagai sebuah perbendaharaan kata), yang merupakan sebuah kumpulan elemen serbaguna, dan sebuah struktur dokumen yang bisa dipakai untuk menguraikan dokumen dengan tipe tertentu (misalnya, dokumen yang berisi rumus matematika atau grafik vektor).

Sintaks XML menawarkan opsi yang lebih sedikit dibandingkan SGML,

sehingga lebih memudahkan manusia membaca dokumen XML, dan untuk para programmer untuk menulis browser, script, dan halaman Web yang mengakses dan menampilkan informasi dokumen.

Apakah XML Menggantikan HTML?

Saat ini, jawaban untuk pertanyaan itu adalah tidak. HTML masih menjadi bahasa utama yang dipakai untuk memberitahu browser bagaimana menampilkan informasi pada Web.

Dalam Internet Explorer 5, Kita bisa membuka sebuah dokumen XML dengan sebuah style sheet yang disertakan langsung dalam browser, tanpa menggunakan sebuah halaman HTML. Dua metode utama lainnya menampilka dokumen XML pengikatan data dan script DOM menggunakan halaman Web HTML sebagai mesin untuk menampilkan dokumen XML (bahkan dengan metode style sheet, jika Kita menggunakan XSL, Kita akhirnya memanfaatkan HTML untuk menyatakan pada browser bagaimana memformat data XML).

Ketimbang menggantikan HTML, XML saat ini dipakai bersama sama HTML dan sangat memperluas kapabilitas halaman Web untuk:

1. Menyampaikan secara virtual berbagai macam dokumen.
2. Mengurutkan, menyaring, mencari, dan memanipulasi informasi dengan cara yang lain.
3. Menghadirkan informasi yang sangat terstruktur.

Seperti kutipan pada awal bab menyatakan, XML dirancang untuk interoperabilitas dengan HTML.

Sasaran Resmi XML

Berikut ini adalah sepuluh sasaran XML seperti yang dinyatakan dalam spesifikasi XML resmi yang ditempatkan pada situs Web W3C (http://www.w3.org/TR/REC-xml).

1. XML harus dapat dipakai langsung pada Internet.

   Seperti yang Kita lihat dalam bab ini, XML dirancang terutama untuk menyimpan dan menyampaikan informasi pada Web.

2. XML harus mendukung berbagai macam aplikasi.

   Walaupun tujuan utamanya untuk menyampaikan informasi melalui Web via server dan program browser, XML juga dirancang untuk dipakai dengan jenis program lainnya. Misalnya, XML telah dipakai untuk pertukaran informasi antarprogram keuangan, untuk mendistribusikan dan mengupdate perangkat lunak, dan untuk menuliskan script voice bagi penyampaian melalui telepon.

3. XML harus kompatibel dengan SGML.

   Seperti Penulis jelaskan, XML adalah sebuah subset serbaguna dari SGML. Manfaat fitur ini adalah bahwa tool perangkat lunak SGML gampang diadaptasikan untuk menggunakan XML.

4. XML harus memudahkan penulisan program yang memproses dokumen XML.

   Jika XML bisa praktis, ia harus memudahkan penulisan browser dan program lain yang memproses dokumen XML. Pada kenyataannya, alasan utama untuk pembuatan subset XML dari SGML adalah kekakuan penulisan program untuk memproses dokumen SGML.

5. Jumlah fitur opsional dalam XML dibuat seminim mungkin, idealnya nol.

Dengan jumlah fitur opsional yang minim dalam SML menjadikannya lebih mudah untuk menuliskan program untuk mengakses dokumen XML. Besarnya fitur opsional pada SGML menjadi alasan utama mengapa ia dianggap tidak praktis untuk mendefinisikan dokumen Web. Fitur SGML opsional termasuk mendefinisikan kembali karakter pembatas dalam tag-tag (biasanya < dan >) dan pengecualian pada tag-akhir, bila prosesor bisa mengerti di mana sebuah elemen berakhir. Sebuah program yang mapan untuk pemrosesan dokumen SGML perlu memperhitungkan semua fitur opsional, bahkan yang jarang dipakai.

6. Dokumen XML harus bisa dipahami oleh manusia clan jelas.

   XML dirancang untuk menjadi lingua franca (bahasa pergaulan) bagi pertukaran informasi antarpengguna dan program di seluruh dunia. Keterbacaan oleh manusia mendukung sasaran ini dengan memungkinkan orang sebagaimana program perangkat lunak yang khusus untuk menulis dan membaca dokumen XML. Legibilitas manusiawinya membedakan XML dari format paling baku yang dipakai untuk dokumen database dan pengolah kata.

   Manusia dapat dengan mudah membaca sebuah dokumen XML karena ditulis dalam teks biasa, dan mempunyai sebuah struktur pohon yang logis. Kita bisa menambah legibilitas XML dengan memilih nama-nama yang bermakna bagi elemen, atribut, dan entitas dokumen Kita dengan menambahkan komentar yang berguna.

7. Desain XML harus dapat disiapkan dengan cepat.

   XML akan, tentu saja, menjadi sebuah yang layak, hanya jika komunitas programmer dan user mengadopsinya. Karenanya tersebut perlu

dituntaskan sebelum komunitas ini mulai mengadopsi alternatif, di mana perusahaan perangkat lunak cenderung berproduksi dengan cepat.

8. Desain XML harus bersifat formal dan ringkas.

   Spesifikasi XML yang ditulis dalam bahasa formal digunakan untuk mendefinisikan bahasa komputer, yang dikenal sebagai notasi Extended Backus Naur Form (EBNF). Bahasa formal ini, walaupun sulit dibaca secara biasa, memecahkan ambiguitas dan akhirnya menjadikannya lebih mudah untuk menulis dokumen XML dan khususnya perangkat lunak pengolah XML, yang selanjutnya mendorong pengadopsian XML.

9. Dokumen XML hams mudah dibuat.

   Agar XML bisa menjadi bahasa kode yang praktis bagi dokumen Web, program pengolah XML tidak. cuma harus mudah ditulis, namun juga dokumen XML sendiri harus mudah dibuat.

10. Keringkasan dalam markup XML tidak begitu dipentingkan.

    Sejalan dengan sasaran ke-6 (dokumen XML harus bisa terbaca oleh manusia dan jelas), kode XML tidak begitu ringkas seperti halnya bahasa sandi.

Aplikasi XML

Seperti yang telah Kita lihat, Kita tidak hanya bisa menggunakan XML untuk melukiskan sebuah dokumen individual, namun juga sesesorang, perusahaan, atau panitia bisa mendefinisikan seperangkat elemen XML serbaguna, bersama dengan sebuah struktur dokumen, untuk dipakai oleh kelas dokumen tertentu. Seperangkat elemen serbaguna dan struktur dokumen dikenal sebagai aplikasi XML atau perbendaharaan kata XML.

Sebagai contoh, sebuah organisasi bisa mendefinisikan sebuah aplikasi XML untuk membuat dokumen yang mendefinisikan struktur molekuler, dokumen yang menerangkan sumber daya manusia, dokumen yang menyajikan presentasi multimedia, atau dokumen yang menyimpan grafik vektor. Pada akhir bab ini, Penulis telah mencantumkan beberapa aplikasi XML yang tersedia secara luas yang telah dibuat atau diajukan.

Aplikasi XML biasanya didefinisikan dengan membuat sebuah document type definition (DTD), yang merupakan sebuah komponen opsional pada dokumen XML. DTD seperti skema database: mendefinisikan dan menamai elemen yang bisa dipakai dalam dokumen tersebut, dengan susunan dimana elemen bisa muncul, atribut yang bisa dipakai, dan fitur dokumen lainnya. Untuk menggunakan aplikasi XML tertentu, biasanya Kita memasukkan DTD nya dalam dokumen XML Kita; menjadikan DTD dalam dokumen tersebut membatasi elemen dan struktur yang bisa Kita gunakan sehingga dokumen Kita dipaksa untuk tunduk pada aplikasi. Contoh dokumen XML yang Kita lihat sebelumnya dalam bab ini tidak memasukkan sebuah DTD. Kita akan mempelajari cara mendifinisikan dan menggunakan DTD.

Keuntungan menggunakan aplikasi XML untuk mengembangkan dokumen Kita adalah Kita bisa berbagi dokumen dengan user lainnya dari aplikasi tersebut, dan dokumen tersebut bisa diproses dan ditampilkan menggunakan perangkat lunak yang telah dibuat bagi aplikasi tersebut.

Aplikasi XML untuk Meningkatkan Dokumen XML

Sebagai tambahan aplikasi XML untuk menggambarkan kelas spesifik pada suatu dokumen, sejumlah aplikasi XML telah didefinisikan di mana Kita bisa

menggunakannya dalam berbagai macam dokumen XML. Aplikasi ini memudahkan pembuatan dokumen dan memungkinkan Kita menambahkan perbaikan ke dokumen tersebut

Beberapa contoh adalah:

1. Extensible Stylesbeet Language (XSL) memungkinkan Kita membuat style sheet dokumen yang canggih menggunakan sintaks XML.
2. XML Schema memungkinkan Kita menulis skema yang rinci untuk dokumen XML Kita menggunakan sintaks XML, dan memberikan altenatif yang lebih mampu untuk menulis DTD.
3. XML Linking Language (XLink) mengizinkan Kita mengaitkan dokumen XML. Ia memungkinkan target multilink dan sejumlah fitur lainnya, dan dianggap lebih mampu ketimbang mekanisme pengaitan HTML.
4. XML Pointer Language (XPointer) memungkinkan Kita mendefinisikan target link yang luwes. Kita bisa menggunakan XPointer bersama dengan XLink dan mengaitkan ke berbagai lokasi dalam dokumen target tidak hanya ke target link yang telah ditKitai secara khusus saja.

Kita bisa melihat bahwa XML tidak hanya sebuah tool bermanfaat yang segera untuk mendefinisikan dokumen, namun juga bertindak sebagai kerangka kerja untuk membangun aplikasi dan peningkatan XML yang akan dibutuhkan untuk perkembangan Internet.

Penggunaan XML dalam Pekerjaan Nyata

Walau pun XML tampak seperti konsep yang menarik, Kita mungkin bertanya apa yang sebenarnya dapat Kita lakukan dengannya dalam pekerjaan nyata. Dalam bagian ini, Penulis telah mendaftarkan contoh penggunaan praktis

XML. Penulis memasukkan cara-cara yang sekarang dipakai XML, seperti halnya yang dipakai beragam kelompok yang telah disampaikan. Jika satu atau lebih aplikasi XML telah didefinisikan untuk penggunaan tertentu, Penulis memasukkan aplikasi semacam ini dalam tKita kurung. Misalnya, dari daftar Kita bisa ketahui bahwa MathML adalah sebuah aplikasi XML yang memungkinkan Kita memformat rumus matematika.

C. PENUTUP

Ringkasan

Untuk mendapatkan daftar yang lebih lengkap dari aplikasi XML yang ada dan yang diajukan, termasuk uraian detil masing-masing aplikasi, lihat halaman Web Oasis SGMUXML (http://www.oasis-open.org!coverlxm/ .html#applications) .

1. Penyimpanan database. Seperti format database baku, XML bisa dipakai untuk menamai setiap field informasi dalam masing-masing record database (misalnya, ia bisa mengitai setiap nama, alamat, dan nomor telepon dalam record database daftar alamat). Penamaan setiap potong informasi memungkinkan Kita menampilkan data tersebut dalam berbagai cara dan mencari, mengurutkan, menyaring, serta memproses data dalam cara lain.
2. Pembentukan dokumen. Struktur pohon dokumen XML menjadikan XML ideal untuk menKitai struktur dokumen, seperti halnya novel, buku nonfiksi, dan permainan. Sebagai contoh, Kita bisa menggunakan XML untuk mengitai sebuah permainan menjadi lakon, latar, pembicara, alur cerita, arahan panggung, dan seterusnya. XML memungkinkan perangkat

lunak menampilkan atau mencetak dokumen dengan format yang sesuai; untuk mencari, mengutip, atau memanipulasi informasi dokumen; untuk membuat tabel daftar isi, ringkasan, dan sinopsis; dan untuk menangani informasi dengan cara yang lain.

3. Penyimpanan grafik vektor. (VML, atau Vector Markup Language)
4. Menyajikan presentasi multimedia. (SMIL, atau Synchronized Multimedia Integration Language, dan HTML+TIME, atau HTML Timed Interactive Multimedia Extensions)
5. Mendefinisikan berbagai saluran. Saluran atau channel adalah halaman Web yang dikirim (dikirim otomatis) untuk pelanggan. (CDF, atau Channel Definition Format)
6. Menggambarkan paket perangkat lunak dan interdependensi mereka. Penjelasan ini memungkinkan perangkat lunak didistribusikan dan diupdate melalui jaringan. (OSD, Open Software Description)
7. Berkomunikasi antaraplikasi melalui Web dalam cara yang terbuka dan luwes, menggunakan pesan berbasis XML. Pesan ini tidak bergantung pada sistem operasi, model obyek, dan bahasa komputer yang dipakai. (SOAP, atau *Simple Object Access Protocol*)
8. Pengiriman kartu bisnis elektronik via e-mail.
9. Pertukaran informasi keuangan. Informasi tersebut dipertukarkan secara terbuka, dalam format yang terbaca, antarprogram keuangan (seperti misalnya Quicken dan Microsoft Money) dan lembaga keuangan (seperti bank, dan dana bersama). (OFX, atau Open Financial Exchange)
10. Pembuatan, pengaturan, dan penggunaan form digital yang rumit untuk transaksi perdagangan Internet. Form ini bisa memasukkan tKita digital

yang membuat form tersebut terikat secara legal. (XFDL, atau Extensible Forms Description Language)

11. Pertukaran uraian kerja dan biodata. (HRMML, atau Human Resource Management Markup Language)
12. Memformat rumus matematika dan kandungan ilmiah pada Web. (MathML, atau Mathematical Markup Language)
13. Menjelaskan struktur molekuler. (CML, atau Chemical Markup Language)
14. Pengkodean dan penampilan DNA, RNA, dan informasi sekuen protein. (BSML, atau Bioinformatic Sequence Markup Language)
15. Pengkodean data geneologis. (GedML, atau Genealogical Data Markup Language)
16. Pertukaran data astronomis. (AML, atau Astronomical Markup Language)
17. Penulisan lembar musik. (MusicML, atau Music Markup Language)
18. Penyimpanan script suara untuk penyampaian melalui telepon. Script suara dipakai untuk, sebagai contoh, menghasilkan instruksi voice-mail, pergerakan saham, dan laporan cuaca. (Vo: xML)
19. Pengiriman iklan surat kabar nasional dalam format digital. (AdMarkup)
20. Pengarsipan dokumen legal dan pertukaran informasi legalsecara elektronik. (XCI, atau XML Court Interface)
21. Pengkodean laporan pengamatan cuaca. (OMF, atau Weather Observation Markup Format)
22. Pertukaran informasi transaksi realestate. (RETS, atau Real Estate Transaction d)
23. Pertukaran data yang berhubungan dengan asuransi.

24. Pertukaran berita dan informasi menggunakan Web terbuka. (XMLNews)
25. Penyajian informasi keagamaan dan penKitaan teks liturgis. (ThML, Theological Markup Language, dan LitML, atau Liturgical Markup Language)

Pertanyaan

1. Apa itu XML?
2. Apakah XML menggantikan HTML?
3. Apa arti dari SGML, HTML dan XML?

DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1.

http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.

10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 9
SKEMA XML

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 9 akan di bahas Skema XML (Extensible Markup Language) diantaranya: Menerangkan Dokumen XML, Berbagai Datatype Built In, String, Data Biner, Namespace, Atribut targetNamespace, Atribut XMLNS, Atribut Schema Location, Atribut noNamespaceSvhemaLocation, Namespace XML, Definisi Elemen, Berbagai Datatype Custom, Tipe Kompleks, Group Elemen dan Atribut, Membatasi Pewarisan, .

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan Kebutuhan Akan XML, Solusi XML, Menulis Dokumen XML, Menampilkan Dokumen XML, SGML, HTML dan XML, Apakah XML Menggantikan HTML, Sasaran Resmi XML, Aplikasi XML, Aplikasi XML untuk Meningkatkan Dokumen XML, Penggunaan XML dalam Pekerjaan Nyata.

B. PENYAJIAN

SKEMA XML

SOAP menyediakan sebuah metode pengodean data ke dalam sebuah dokumen XML. Teknologi ini juga luar biasa luwesnya. Segalanya dapat dikodekan ke dalam tubuh pesan SOAP selama ia mensahkan XML. Tubuh pesan dapat berisi permintaan bagi informasi cuaca, order

pembelian, bagian dari sebuah rentetan pesan instan, citra satelit, dan segala yang dapat dibayangkan oleh seorang implementer layanan Web.

Dengan beragam isi dan tipe data yang dapat dimuat dalam sebuah pesan SOAP, Kita perlu sebuah cara untuk mengekspresikan struktur pesan tersebut. Kita juga perlu cara untuk menentukan tipe data yang akan muncul dalam sebuah pesan.

Barangkali, satu solusi potensial bagi pengembang untuk menyediakan sebuah contoh tentang seperti apa nantinya pesan SOAP yang valid akan terlihat. Sebagai contoh, anggaplah Kita ingin membuat antar muka dengan sebuah layanan Web untuk menempatkan sebuah order kepada suatu vendor, dan vendor tersebut telah menyediakan contoh pesan berikut:

```
<? xml version-"1 .0 "encoding-"utf- 8 "?>
<soap: Envelope xmlns: soap-"http://schemas.xml
soap.org/soap/envelope/" >
<soap: Body>
<PurchaseItem>
<Item>Apple</Item>
<Ouantity>l2</0ua nti ty>
<! Purchaseltem>
</soap: Body>
<lsoap: Envelope>
```

Dokumen XML-nya sangat sederhana. Dari contoh pesan tersebut, Kita dapat melihat bahwa layanan Web menerima dua parameter, Item dan Quantity. Kedua parameter ini merupakan elemen anak dari elemen Purchaseltem.

Masalahnya adalah bahwa pesan contoh tersebut menyisakan banyak ambiguitas. Sebagai contoh, Kita tahu bahwa Kita perlu menyalurkan

sebuah parameter Item, tapi haruskah ia berisi sebuah keterangan singkat item tersebut? Haruskah ia berisi salah satu dari sekian pemilihan nomor? Apakah nilainya sebatas pada jumlah karakter maksimum?

Ada banyak jawaban berkenaan dengan parameter Quantity. Adakah jumlah minimum yang harus dibeli? Adakah jurnlah maksimum yang dapat dibeli?

Sebuah cara untuk menjelaskan ambiguitas adalah membuat agar contoh pesan tersebut disertakan bersama sebuah dokumen yang menerangkan semua nuansa pesan tersebut, karena layanan Web setidaknya perlu mensahkan pesan sebelum Kita mengirimkannya ke layanan Web. Namun dengan pendekatan ini, Kita dan pengembang layanan Web barangkali akan kesulitan menulis sendiri kode pengesahannya.

Perlu sebuah cara untuk menjelaskan struktur dan tipe informasi yang harus dimuat dalam sebuah pesan XML yang dikirimkan ke layanan Web. Dengan kata lain, Kita perlu cara untuk menyatakan skema yang harus dipatuhi sebuah pesan XML supaya dapat diproses oleh layanan Web. Lebih jauh lagi, skema tersebut perlu diisasi dan disertakan dengan seperangkat API yang dapat dipakai untuk mensahkan sebuah dokumen XML secara programatikal terhadap skema itu.

Menerangkan Dokumen-Dokumen XML

Kita dapat memakai sebuah skema untuk menerangkan struktur sebuah dokumen XML dan informasi tipenya. Dua teknologi dominan untuk mendefinisikan skema XML adalah DTD (*Document Type Definitions*) dan XML Schema. Kita dapat memakai DTD untuk mendefinisikan

struktur sebuah dokumen XML, tapi tidak untuk menerangkan isi dokumen. Berikut ini sebuah contohnya:

```
<! - - RequestMessage - - >
<! ELEMENTPurchase! tem
<! ELEMENTQuantity
<! ELEMENTItem
<! -- ResponseMessage- - >
(Quantity.Item)> (#PCDATA)> (#PCDATAl >
<! ElementPurchase! temResult (Amount)>
<! ELEMENTAmount (#PCDATAl>
```

Sekilas, tampak bahwa sintaks DTD bukanlah berbasis XML. DTD tidak dapat diurai dengan memakai parser XML dan tidak dapat dengan mudah ditanamkan ke dalam dokumen XML lainnya.

DTD menerangkan struktur dokumen, namun tidak dapat mengekspresikan tipe data yang dimuatnya. Tidak ada konseptualisasi tipe-tipe fundamental dalam DTD, seperti halnya integer dan string, juga tidak ada dukungan untuk mendefinisikan tipe-tipe Kita sendiri.

Dalam contoh terdahulu, kedua elemen *Quantity* dan Item dideklarasikan sebagai #PCDATA. Ini tidak memberi Kita penjelasan apa pun mengenai tipe data yang dikandungnya. Sebagai contoh, DTD tidak menunjukkan apakah ia valid untuk menampilkan daftar kuantitas parsial, seperti misalnya 1,5 kasus. Ia juga tidak menunjukkan apakah elemen Item harus berisi sebuah ID produk numerik atau sekedar sebuah string yang berisi keterangan item.

Proposal yang telah diajukan ke W3C, bernama Oatatypes for DTDs (DT4DTD) 1.0 (http://www.w3 .org/TR/dt4dtd), menyediakan sebuah

sarana untuk menyisipkan informasi tipe ke dalam skema DTD. Saat penulisan ini, proposal tersebut telah dicantumkan sebagai catatan selama lebih dari setahun setengah dan kelihatannya tidak mendapat perhatian. DTD mestinya dianggap sebagai teknologi warisan untuk mendefinisikan berbagai skema XML karena keterbatasan mereka dan kurangnya dukungan industri.

Cara yang disarankan untuk mengekspresikan skema bagi layanan Web yang berbasis XML adalah via XML Schema. XML Schema mencakup dua spesifikasi yang telah diatur oleh W3C, XML Schema Part 1: Structures (http://www.w3.org/TR/xmlschema-1/) dan XML Schema Part 2: Oatatypes (http://www.w3.org/TR/ xmlschema-2/). Mulai 2, Mei 2001, kedua spesifikasi telah direkomendasikan W3C.

XML Schema menyediakan sebuah sintak yang kaya untuk mendefinisikan berbagai skema yang dipakai untuk mensahkan dokumen-dokumen instance XML. Skema ini tidak hanya memungkinkan Kita mendefinisikan struktur sebuah dokumen XML, tapi juga memungkinkan Kita mendefinisikan tipe data yang dikandung dokumen tersebut dan segala batasan pada data itu. Juga, ia memungkinkan Kita menyebutkan kunci asing dan batasan integritas referensial. Berikut ini adalah sebuah contoh yang sederhana:

```
<? xml version1.o'?>
<schema xmlns 'http://www.w3.org/2001/XML Schema')
<! - - Res ponse Message (work in-progress)- - >
<elementname'Amount'/)
<! schema >
```

Seperti yang dapat Kita lihat, contoh skema tersebut adalah sebuah

dokumen XML yang dapat dipakai oleh semua parser XML. Definisi skema dimuat dalam elemen schema root. Contoh skema tersebut mendefinisikan satu elemen bernama Amount.

Dokumen-dokumen XML yang dapat disahkan sebuah skema disebut dokumen instance. Berikut ini adalah dokumen instance bagi skema yang tadi telah dijelaskan:

```
<? xml version '1.0'?>
< Amount>351. 43</Amount>
```

Berbagai Datatype Built-In

Salah satu dari sekian fitur berguna pada XML Schema adalah karena ia mendefinisikan seperangkat datatype inti. Ini meliputi tipe-tipe pemrograman dasar seperti string, int, float, dan double, yaitu tipe-tipe matematis seperti misalnya integer dan decimal; dan tipe-tipe XML seperti misalnya NMTOKEN dan IDREF.

Salah satu kelebihan yang signifikan dari sistem tipe XML Schema adalah karena ia sepenuhnya independen terhadap platform. Nilai-nilai tipe dinyatakan secara konsisten tanpa melihat model perangkat keras, sistem operasi, atau perangkat lunak pemrosesan XML yang dipakainya. Sistem tipe XML Schema memungkinkan protokol yang berbasis XML, seperti misalnya SOAP mencapai kerja sama yang kuat dalam lingkungan komputasi heterogen.

Datatype berguna untuk mendefinisikan berbagai skema yang menerangkan tipe data yang harus dimuat dalam sebuah dokumen. la banyak mempengaruhi tipe data ketika Penulis nanti menjelaskan bagaimana membuat berbagai skema dalam bab ini. Cara lain supaya Kita

dapat mengatur sistem tipe XML Schema, yaitu dengan menjelaskan sebuah dokumen XML dengan tipe data yang dikandungnya. Ini membantu membuang ambiguitas mengenai maksud pembuat dokumen tersebut.

Dalam bagian sebelumnya, Penulis telah membuat sebuah pesan SOAP untuk mengirimkan sebuah permintaan PurchaseItem. Di sini, Penulis akan memakai berbagai datatype built-in untuk menunjukkan tipe-tipe parameter yang akan disalurkan.

```
<? xml version-"1 .0 "encoding-"utf-8"?>
<soap:Envelope xmlns :soap-"http://schemas.xml soap.org/soap/envelope/"
xmlns:xsi-"http://www.w3.org/2001/XMLSchema-instance">
<soap: Body>
<PurchaseItem>
<item xsi: type-"xsi: string">Apple</item>
<quantityxsi: type-"xsi: int">l</quantity>
</PurchaseItem>
</soap: Body>
</soap: Envelope>
```

Penulis telah menambahkan sebuah atribut xsi: type untuk masing-masing elemen parameter. Nilai atribut menyatakan datatype dari parameter yang telah dikodekan. Dengan menghias elemen-elemen dalam sebuah dokumen XML memakai informasi tipe akan mebuang semua ambiguitas pada tipe data yang telah dikodekan pengirim ke dalam pesan tersebut. Penerima dari pesan awal akan tahu bahwa quantity yang dinyatakan sebagai sebuah int dan item akan dinyatakan sebagai sebuah string.

Spesifikasi SOAP mendefinisikan sebuah polymorphic accessor sebagai sebuah elemen yang tipenya ditentukan pada saat runtime. Accessor

polimorfik secara konseptual mirip dengan tipe Object dalam Visual Basic.NET. Jika sebuah accessor polimorfik muncul dalam pesan SOAP harus berisi sebuah atribut type yang menunjukkan tipe data yang dikandung elemen tersebut.

Lampiran yang ada menyediakan sebuah daftar lengkap dari berbagai datatype built-in XML Schema. Perhatikan bahwa SOAP 1.1 telah didefinisikan pada draft kerja spesifikasi XML Schema yang diterbitkan pada tahun 1999. Sebagian dari tipe built-in ini telah mengalami perubahan nama selama direkomendasikan. Walaupun spesifikasi SOAP 1.1 dan berbagai skema yang berhubungan dengannya mengacu pada versi XML Schema terdahulu, sebagian besar implementasi SOAP yang memasukkan ASP.NET dan Remoting mengacu pada datatype built-in yang didefinisikan oleh spesifikasi XML Schema yang ada sekarang.

Dalam pengingat bagian ini, Penulis memasukkan beberapa data type menarik lainnya.

Bahasa XML Schema mendefinisikan sejumlah tipe yang dipakai untuk menerangkan sejumlah integer. Dua tipe yang seringkali membingungkan adalah integer dan int. Tipe int sebenarnya adalah sebuah turunan dari tipe integer dengan pembatasan tambahan.

Sekalipun keduanya menyatakan sebuah nilai integer, mereka melayani tujuan yang berbeda. Elemen dan atribut bertipe integer berisi sebuah nilai yang memenuhi definisi matematis dari sebuah integer. Jumlah tipe integer tak terbatas, maka dapat berisi nilai-nilai yang menyebabkan sebuah kondisi *overflow* jika disalin ke dalam register CPU. Karena elemen-elemen dan atribut-atribut dari tipe int dibatasi untuk memuat

sebuah integer 32-bit, mereka menjadi lebih cocok untuk aplikasi ilmu komputer.

Perbedaan yang sama berlaku untuk angka-angka desimal. Instance dari tipe float tunduk pada tipe titik-ambang presisi tunggal IEEE (*Institute of Electrical and Electronics Engineers*). Di lain pihak, tipe decimal menyatakan sebuah angka desimal presisi acak.

String

XML Schema mendefinisikan sebuah datatype string, namun tidak identik dengan tipe string dalam sejumlah bahasa pemrograman dan database. Secara khusus, tipe-tipe string memungkinkan karakter-karakter terlarang untuk muncul dalam datatype string XML Schema.

Karakter-karakter yang dapat membatalkan atau mengubah pengertian dari dokumen XML tidak dapat dimuat dalam sebuah elemen atau atribut bertipe string. Sebagai contoh, karakter-karakter yang dicadangkan, seperti misalnya "-" (kurang dari) dan "&" (dan) memiliki pengertian khusus dan tidak dapat muncul dalam sebuah dokumen XML. Karakter-karakter lain seperti "?" (tanda tanya) dan " " (tanda kutip), tidak dapat muncul dalam nilai suatu atribut. Karakter-karakter ini harus dihindari atau dikodekan dalam suatu gaya sebelum mereka diserialisasikan ke dalam sebuah dokumen XML.

XML mendefinisikan sebuah cara mengodekan karakter-karakter individual dalam sebuah dokumen XML memakai referensi karakter. Referensi karakter berisi sebuah tanda "&" diikuti sebuah identifier karakter, baru kemudian sebuah tanda ";". Identifier karakter dapat berupa

identifier numerik karakter unicode atau sebuah referensi entitas karakter. Referensi karakter numerik dipakai untuk mengenali karakter spesifik dalam kumpulan karakter Unicode (ISO/IEC 10646). Identifier karakter adalah nilai desimal atau heksadesimal dari karakter tersebut, didahului sebuah tKita pound. Misalnya, karakter A dapat dikodekan sebagai A atau &#x4l;.

Spesifikasi XML menyediakan referensi entitas yang merupakan identifier karakter yang dapat lebih terbaca untuk sebuah subset kecil dari karakter-karakter *Unicode*. Walaupun spesifikasi HTML 4 mendefinisikan ribuan referensi entitas karakter, spesifikasi XML hanya mendefinisikan lima, untuk karakter-karakter yang membatasi XML well-formed, seperti diperlihatkan dalam Tabel 9-1

Tabel 9-1 Referensi Entitas Karakter XML

Karakter	Referensi Karakter Numerik	Referensi Entitas Numerik
“	" atau "	&quot:
‘	' atau '	&apos;
&	& atau &	&amp;
<	< atau <	&lt;
>	> atau >	&gt;

TKita kutip dan apostrof mempunyai referensi entitas karakter yang didefinisikan untuk mereka karena pada kondisi tertentu tidak diperkenankan dalam nilai sebuah atribut. Jika nilai atribut diapit tKita kutip, tKita kutip tidak dapat muncul dalam nilai atribut tersebut. Hal yang

sama berlaku untuk apostrof. Sebagai contoh, elemen-elemen berikut berisi karakter-karakter ilegal:

```
<ea-"Scottsays,"This is illegal. "">
<ea-'Don't do this,either.'>
```

Elemen-elemen berikut adalah sah:

```
<ea-'Scott says,"This is perfectlyfine."' >
<ea-"This isn't aproblem.either.">
```

Cara lain untuk memasukkan string-string dengan karakter-karakter yang telah dicadangkan dalam sebuah dokumen XML adalah dalam seksi CDATA. XML mendefinisikan urutan karakter <! [CDATA [untuk memberitahu prosesor XML agar mengabaikan karakter-karakter khusus tersebut sampai menjumpai}} >.

Berikut ini adalah contohnya:

```
<myString><![CDATA[I can now use all five reserved
characters.(",'&.<.and>lJJ> </myString>
```

Saat Kita menyerialkan variabel-variabel string ke dalam sebuah dokumen XML, pastikan untuk mengodekan karakter-karakter khusus memakai referensi karakter atau hindari string tersebut dalam bagian CDATA.

Data Biner

Data biner harus dikodekan sebelum disisipkan ke dalam sebuah dokumen XML untuk memastikan bahwa ia tidak memasukkan segala karakter yang dapat menjadikan XML invalid. Spesifikasi XML Schema mendefinisikan dua datatype built in bagi data biner, base64Binary dan hexBinary.

Tipe hexBinary mengodekan setiap oktet biner ke dalam dua padanan heksadesimal karakternya. Sebagai contoh, nilai biner dari 11111111 akan dikodekan sebagai FF, FF, FF, atau FF.

Platform.NET menyediakan dukungan bagi pengodean dan decoding binhex. Kita dapat memakai metode *XmlTextReader.ReadBinHex* untuk mendekodekan binhex ke data biner dan metode untuk *XmlTextWriter*. WriteBinHex mengodekan data biner ke binhex.

Kita lebih terbiasa melihat data biner bertipe base64Binary. Hal ini khususnya berlaku pada layanan-layanan Web karena spesifikasi SOAP 1.1 merekomendasikan bahwa semua data biner yang ditanamkan dalam sebuah pesan harus dikodekan memakai algoritma yang didefinisikan oleh RFC 2045.

Elemen-elemen dan atribut bertipe base64Binary berisi data yang dikodekan memakai algoritma pengodean Base64 yang dijelaskan dalam RFC 2045. Seperti yang diperlihatkan Tabel 9-2, masing-masing kutipan 6 bit sebuah array oktet biner dikodekan ke dalam karakter XML yang kompatibel.

Tabel 9-2 Abjad Base64

Biner	Base64	Biner	Base64	Biner	Base64	Binary	Base64
000000	A	010000	Q	100000	g	110000	w
000001	B	010001	R	100001	h	110001	x
000010	C	010010	S	100010	i	110010	y
000011	D	010011	T	100011	j	110011	z
000100	E	010100	U	100100	k	110100	0
000101	F	010101	V	100101	l	110101	1
000110	G	010110	W	100110	m	110110	2
000111	H	010111	X	100111	n	110111	3

001000	I	011000	Y	101000	o	111000	4
001001	J	011001	Z	101001	p	111001	5
001010	K	011010	a	101010	q	111010	6
001011	L	011011	b	101011	r	111011	7
001100	M	011100	c	101100	s	111100	8
001101	N	011101	d	101101	t	111101	9
001110	O	011110	e	101110	u	111110	+
001111	P	011111	f	101111	v	111111	/

Base64 juga mendefinisikan sebuah karakter ke 65 untuk tujuan padding. Satu atau beberapa tKita = dapat muncul di akhir string yang dikodekan. Jika objek biner dimasukkan dengan persis ke dalam porsi 6 bit, maka tidak ada padding karakter yang diterapkan ke akhir string Base64. Semua kondisi lain perlu ditam- bahkan nol di akhir objek binernya yang sudah dikodekan. Sebuah karakter = akan ditambahkan ke akhir string yang dikodekan bagi setiap dua nol yang ditambahkan ke objek biner. Karena objek biner terdiri dari serangkaian byte (8 bit), maka ada tiga kemungkinan skenario, termasuk yang baru saja disebutkan:

1. **Sebuah byte tunggal tersisa akan dikodekan.** Dalam hal ini, empat nol akan ditambahkan, dua bagian (chunk) 6-bit yang dihasilkan akan dikode- ka n dan dua karakter = ditambahkan ke akhir string yang telah dikodekan.
2. **Dua byte tersisa akan dikodekan.** Dalam hal ini, dua nol ditambahkan, tiga bagian 6 bit hasilnya akan dikodekan dan sebuah karakter = tunggal akan ditambahkan ke akhir string yang telah

dikodekan.

3. **Tiga byte tersisa akan dikodekan.** Dalam hal ini, byte-byte yang tersisa dapat dipecah menjadi empat chunk 6 bit. Jadi, tidak ada karakter = yang ditambahkan ke akhir string yang dikodekan.

Mirip seperti binhex, XmlTextWriter dan XmlTextReader menyediakan metode WriteBase64 dan ReadBase64 bagi encoding dan decoding data Base64-encoded. Selain itu, platform .NET menjaga encoding dan decoding yang benar atas data biner bagi layanan-layanan Web yang dibangun pada kerangka kerja ASP.NET dan Remoting.

Namespace

Namespace berpengaruh besar dalam skema-skema XML. Jadi, Penulis tidak dapat melanjutkan tanpa membahas dulu apa yang dimaksud namespace dan bagaimana pendefinisiannya.

Namespace menyediakan bingkai logika untuk berbagai entitas yang didefinisikan dalam sebuah skema. Sebagai contoh, dalam bagian sebelumnya, Penulis telah membuat sebuah skema yang mendefinisikan elemen Amount. Bagaimanajika ada orang lain yang mendefinisikan sebuah elemen Amount? Jika sebuah elemen Amount muncul dalam sebuah dokumen instance, bagaimana Penulis dapat mengetahuinya; apakah ia sebuah instance dari yang telah didefinisikan oleh skema Penulis atau mereka? Jika setiap elemen Amount akan dikualifikasikan penuh dalam sebuah namespace terpisah, elemen Amount akan dikualifikasikan penuh pada namespace tertentu, sehingga tidak akan menjadi ambisius.

Atribut targetNamespace

Atribut targetNamespace dipakai untuk mengatur identifier bagi namespace. Nilai atribut ini adalah sebuah URI yang bertindak sebagai pointer opaque untuk meru juk namespace tersebut. Berikut ini contoh-contoh dari identifier yang sah bagi namespace-namespace:

```
http: //somed omain.com/
http://somedomain.com/Commerce
urn: Commerce-SomeDomain.Com
urn: Com:SomeDomain:Commerce
urn: WebService:SoapBased:Commerce
```

Dua URI pertama adalah URL yang menyebutkan sebuah nama domain yang terdaftar. Tiga URI yang terakhir adalah Uniform Resource Names (URNs) yang tidak bergantung pada lokasi (location-independent). Salah satu manfaat dari mendefinisikan sebuah namespace dalam konteks nama domain yang terdaftar adalah karena Kita menghindari kemungkinan tabrakan penamaan dengan namespace yang didefinisikan oleh yang lainnya.

Namespace dikenali dengan sebuah URI yang didefinisikan dalam sebuah dokumen skema. Entitas yang dapat dicakup oleh namespace, termasuk berbagai datatype, elemen, dan atribut. Dalam sebuah dokumen XML Schema, elemen schema dapat berisi sebuah parameter targetNamespace yang berisi sebuah URI bagi skema tersebut.

Kode berikut ini mendefinisikan sebuah namespace skema bagi layanan Web Commerce. Untuk saat ini, kode itu berisi definisi elemen Amount. Penulis akan meningkatkannya nanti.

```
<? xml1 version=·1.0?>
<schema xml ns='http://www.w3.org/2001/XMLSchema
'targetNamespace='urn:Commerce'>
<! - - Response Message (work-in-progress)- - >
<elementname-'Amount '/>
<! schema>
```

Penulis telah menambahkan. atribut targetNamespace ke elemen schema dan kemudian mengatur nilainya ke URN Commerce. Semua entitas yang didefinisikan oleh schema tersebut akan dimasukkan dalam namespace Commerce. Dalam hal ini, satu-satunya entitas yang didefinisikan adalah elemen Amount.

Atribut xmlns

Untuk mengualifikasi penuh entitas-entitas yang telah dirujuk dalam sebuah dokumen XML, Kita perlu merujuk satu atau beberapa skema. Dokumen XML yang merujuk skema adalah dokumen-dokumen instance dan skema-skema itu sendiri. Dokumen-dokumen instance harus mengacu pada URL namespace agar dapat mengualifikasikan penuh entitas-entitas yang diacu. Kita dapat melakukan hal ini dengan menambahkan sebuah atribut xmlns ke suatu elemen dalam dokumen tersebut. Berikut ini contohnya:

```
<? xml version-' 1.0 '?>
<Amount xmlns-'urn:Commerce'> 123.45<Amount />
```

Dalam dokumen instance, Penulis mengatur namespace default ke Commerce. Sebagai hasilnya, Amount dan subelemennya, jika ada, akan dikualifikasi penuh dalam namespace Commerce.

Ketika Kita merujuk sebuap namespace, Kita dapat memberikan referensi sebuah moniker. Pemberian moniker ke namespace yang dirujuk berwujud xmlns: moniker='SomeURJ '. Kita kemudian dapat memakai moniker itu untuk mengualifikasi penuh entitas-entitas yang muncul dalam dokumen XML yang didefinisikan dalam namespace yang diacu.

Untuk mengualifikasi penuh sebuah entitas seperti halnya sebuah definisi tipe atau sebuah deklarasi elemen, Kita mulai entitas tersebut dengan moniker yang diikuti sebuah tanda “: " (colon). Semua contoh pesan SOAP yang didefinisikan moniker namespace soap: dalam referensi ke skema SOAP, seperti yang diperlihatkan di sini:

```
<? xml versi on-"1.0 "encodi n g-"utf-8 "?>
<soap:Envelope xmlns:soap-"http://schemas.xml soap.org/soap/envelope/">
<soap: Body >
<! -- SOAP message - - >
</soap: Body>
</soap: Envelope >
```

Seringkali, moniker itu diperlukan untuk merujuk ke banyak skema sekaligus. Kita dapat melakukan ini dengan menambahkan banyak atribut xmlns sekaligus. Atribut xmlns sering ditambahkan ke elemen root pada dokumen supaya lebih mudah terbaca dan demi kenyamanan pengembang. Bagaimanapun juga, referensi skema dapat dibuat dalam suatu elemen di dokumen instance. Berikut ini sebuah pesan SOAP yang berisi dua elemen Amount dalam tubuh pesan:

```
<? xml version-"1.0 "encoding-"utf-8 "?>
<soap: Envelope xmlns: soap-"http://schemas.xml soap.org/soap/envelope/
">
<soap: Body>
<Amount xmlns-'urn: Commerce'> 123.45</Amount>
```

```
<soap: Amount xmlns: soap-'urn: Commerce'>123.45</soap: Amount>
</soap: Body>
</soap: Envelope>
```

Sekalipun Penulis memakai sintaks yang berbeda, kedua elemen Amount dalam dokumen terdahulu adalah sama. Mari kita perhatikan bagaimana Penulis memakai referensi namespace individual.

Tiga referensi namespace telah dibuat. Referensi pertama dibuat dalam elemen root dan mendefinisikan moniker soap: yang dipakai untuk mengualifikasi penuh entitas-entitas yang telah dirujuk dalam skema SOAP Envelope. Referensi kedua mengatur Commerce sebagai namespace default bagi elemen Amount pertama dan elemen-elemen anaknya (jika ada). Referensi terakhir menimpa moniker soap: yang telah didefinisikan sebelumnya daripada merujuk skema Commerce bagi elemen Amount kedua dan anaknya (jika ada).

Referensi-referensi namespace berlaku untuk elemen yang berisi atribut xmlns dan semua elemen anaknya. Oleh karena itu, jika sebuah pernyataan namespace dibuat dalam elemen Header SOAP, pernyataan tersebut berlaku untuk semua elemen dalam header tersebut. Karena referensi-referensi dimasukkan pada elemen di mana mereka dinyatakan, referensi yang dibuat dalam elemen Header SOAP tidak berlaku pada elemen Body SOAP atau elemen anaknya, jika ada.

Referensi-referensi namespace juga banyak dipakai dalam dokumen-dokumen skema. Dalam dokumen-dokumen skema, ia seringkali perlu merujuk entitas-entitas yang didefinisikan dalam dokumen. Kita dapat melakukan ini dengan membuat sebuah referensi di dalam dokumen

skema itu sendiri. Menurut aturannya, referensi ini biasanya diasosiasikan dengan moniker tns: yang merupakan singkatan dari "this namespace". Berikut ini kutipan dari skema Envelope SOAP yang memperlihatkan pemakaian moniker tns:

```
<? xmlversion='1.0 '?>
<! -XML Schema for SOAP v 1.1 En vel ope ->
<! - - Copyr i g ht 2000 Devel opMentor, I ntern ati onal Busi ness
Machi nes Cor porati on.
Lotus Development Corporation,Microsoft.User Land Software ->
<schema xmlns='http://www.w3.org/1999/XMLSchema'
xmlns:tns='http://schemas.xmlsoap.org/soap/envelope/targetNamespace=
'http://schemas.xml soap.org/soap/envelope/'>
<! - - Definition for the Envelopeelement - - >
<elementname= "Envelope"type="tns:Envelope"/>
<! - - Definition for the Envelope type - - >
<complexTypename='Envelope'>
<elementref='tns:Header'minOccurs= '0 ' ! >
<elementref='tns:Body 'minOccurs='l'!>
<anyminOccurs='0 'maxOccurs='*'I>
<anyAttribute/>
</complexType>
<! - - The rest of the SOAP Envelope schema ...- - >
<! schema>
```

Elemen skema mengatur namespace sasaran ke http://schemas.xmlsoap.org/soap/envelope/. Elemen ini juga berisi sebuah referensi ke dirinya sendiri yang memberikan moniker tns: Skema itu mendefinisikan elemen Envelope. Elemen Envelope dinyatakan bertipe Envelope. Karena definisi tipe dimuat dalam skema, maka ia didahului dengan tns:

Atribut schemaLocation

URI dari referensi namespace adalah sebuah pointer opaque. Jadi walaupun URI diberikan dalam bentuk URL, Kita tidak dapat bergantung padanya untuk memecahkan dokumen skema sesungguhnya. Bagaimanapun juga, Kita dapat memakai atribut schemaLocation untuk memberikan petunjuk kepada parser tersebut di mana dokumen skema mendefinisikan namespace-namesp ace yang dirujuk.

Nilai atribut schemaLocation adalah sebuah string berbatas spasi kosong. Ia berisi URI dari skema yang diikuti dengan URL yang menentukan dokumen skema yang dipakai untuk mendefinisikan namespace itu. Kita dapat memberikan banyak petunjuk sekaligus dalam sebuah atribut schemaLocation tunggal. Berikut adalah contohnya:

```
<? xml version='1.0'?>
<Amount xmlns= 'urn:Commerce·
xmlns:xsi='http://www.w3 .org/2001/XMLSchema-instance
xsi:schema Location='urn:Commerce http://somedomain/Commerce.xsd
http://www.w3.org/2001/XMLSchema-instance http://www.w3.org/2001/XM
LSchema.xsd'>
123 .45
<!Amount>
```

Untuk mengilustrasikan, dokumen instance merujuk dua namespace, yaitu namespace Commerce dan namespace XML Schema Instance. Karena URI bagi skema Commerce itu berbentuk sebuah URN, maka ia tidak dapat ditentukan. Bagaimanapun juga, walaupun URL bagi skema XML Schema Instance berbentuk URL, ia tidak langsung dapat ditentukan. Oleh karena itu, Penulis memakai atribut schemalocation untuk memberikan petunjuk tentang tujuan dokumen-dokumen skema untuk berbagai URL

namespace yang telah diasosiasikan tersebut ditempatkan.

Penulis akan membahas dua tema lagi yang menyangkut atribut schemalocation sebelum pindah ke topik selanjutnya. Atribut schemalocation dapat diterapkan pada suatu elemen dalam dokumen instance. Bagaimanapun juga, tidak seperti kebanyakan atribut XML Schema lainnya, atribut schemalocation tetap berlaku bagi dokumen selebihnya, tidak hanya bagi elemen anaknya. Terakhir, karena atribut schemalocation bertindak sebagai petunjuk, parser dapat memilih untuk mencari dokumen skema bagi namespace tertentu dengan memakai metode lain.

Atribut noNamespaceSchemaLocation

Skema-skema tidak dibutuhkan untuk mendefinisikan namespace. Kita dapat memakai Atribut noNamespaceSchemaLocation untuk merujuk ke skema-skema tanpa namespace. Berikut ini contohnya:

```
<? xml version='l.O'?>
<! -- Filenamed Commerce.xsd - - >
<schem a >
<! - - Response Message(workin - progress)- - >
<elementname='Amount'type=/>
<schema>
```

Penulis mendefinisikan sebuah skema tanpa definisi namespace yang dimuat dalam file Commerce.xsd. Selanjutnya, Penulis akan membuat sebuah dokumen instance yang merujuk skema tersebut:

```
<? xml version='l.O'?>
<Amount xmlns:xsi ='http://www.w3.org/2001/XMLSchema-
instancexsi:noNamespaceSchemalocation='file:Commerce.xsd'>
123.45
```

```
<!Amount>
```

Karena elemen Amount tidak didefinisikan dalam sebuah namespace, maka Penulis memakai atribut noNamespace-SchemaLocation untuk merujuk skema itu. Elemen Amount kemudian dikualifikasi penuh sehubungan dengan skema Commerce .xsd.

Sekalipun entitas-entitas yang tidak didefinisikan dalam sebuah namespace dapat dirujuk, sintaks ini rumit dan rentan. Oleh karena itu, Kita sebaiknya menghindari pendefinisian skema tanpa namespace. Jika Kita terpaksa harus merujuk sebuah skema yang tidak berisi definisi namespace, pertimbangkan untuk mengimpor skema tersebut ke dalam sebuah definisi namespace.

Namespace XML Schema dan XML Schema Instance

Spesifikasi XML mendefinisikan dua namespace mendasar, namespace XML Schema dan XML Schema Instance. Sekalipun mereka memakai bersama sebuah subset entitas seperti misalnya definisi tipe, elemen, dan atribut, masing-masing namespace melayani tujuan khusus. Namespace XML Schema harus dirujuk dalam dokumen skema dan namespace XML Schema Instance harus dirujuk dalam dokumen instance.

Namespace XML Schema berisi entitas-entitas yang dipakai untuk mendefinisikan skema-skema. Sebagai contoh, elemen element dan schema yang dipakai dalam skema Commerce didefinisikan dalam namespace XML Schema. URL untuk namespace XML Schema adalah http://www.wJ.org/2001/XM LSchema dan menurut ketentuan namespace seringkali dirujuk dengan moniker xsd:

Namespace XML Schema Instance harus dirujuk dengan dokumen

instance yang memakai entitas-entitas yang didefinisikan dalam namespace. Sebagai contoh, atribut schemaLocation yang didefinisikan dalam namespace XML Schema Instance. URI bagi namespace XML Schema Instance adalah http://www.w3.org/2001/XMLSchemainstance dan menurut ketentuan namespace tersebut seringkali dirujuk dengan moniker xsi:.

Definisi-Definisi Elemen

Seperti yang telah Kita lihat, elemen-elemen didefinisikan memakai elemen element. Atribut name dipakai untuk menyebutkan nama elemen yang akan muncul dalam dokumen instance. Kita dapat juga memakai atribut type untuk menunjukkan apa tipe data yang dapat dimuat elemen tersebut. Penulis akan memperluas definisi elemen Amount untuk menyatakan bahwa ia hanya dapat berisi data bertipe built-in double:

```
<? xml version-'1.0'?>
<schema xmlns- 'http://www.w3.org/2001/XMLSchematargetNamespace-
'urn:Commerce '>
<element name- 'Amount'type-'double'! >
<! schema >
```

Definisi-definisi skema dapat juga menyebutkan apakah sebuah elemen dapat berisi nilai null. Kita dapat juga mengatur atribut nillable elemen ke true atau false. Sebagai contoh, ketika sebuah client memanggil metode PurchaseItern pada layanan Web Commerce Penulis, Penulis dapat atau mungkin tidak dapat mempunyai akses ke informasi harga untuk menghasilkan biaya pembelian. Jika informasi harga tidak tersedia, Penulis tidak ingin memberikan produk secara cuma-cuma. Oleh karena itu, Penulis harus menghasilkan sebuah nilai null untuk menunjukkan

bahwa harga itu tidak tersedia, daripada nol. Berikut ini adalah versi modifikasi dari skema sebelumnya yang memungkinkan elemen Amount berisi nilai null:

```
<? xml version-'1 .0 '?>
<schema xmlns-'http://www.w3.org/2001/XMLSchema targetNamespace-
'urn:Commerce')
<! - - Response Message(work-in-progress) - - >
<elementname-'Amount'type-'double'nillable- 'true '/>
<! schema >
```

Untuk menyebutkan bahwa elemen dalam sebuah dokumen instance berisi nilai null, pakailah atribut xsi: nil:

```
<? xml version- '1.0'?>
<Amount xmlns:xsi-'http://www.w3.org/2001/XMLSchema-instancexmlns-
'urn: Commerce'xsi:nil-'true'/>
```

Karena elemen Amount diatur ke null, maka dokumen instance akan sah, walaupun elemen Amount tidak berisi nilai bertipe double. Juga perlu diperhatikan bahwa atribut xsi: nil hanya berlaku pada nilai elemen dan tidak berlaku pada atribut-atributnya, jika ada yang didefinisikan.

Berbagai Datatype Custom

Sistem tipe XML Schema sangat mungkin diperluas. la menyediakan sebuah mekanisme untuk mendefinisikan datatype-datatype baru yang mewarisi berbagai datatype built-in atau datatype custom. Datatype dibagi ke dalam dua kategori, yang bertipe sederhana dan bertipe kompleks. Tipe-tipe sederhana tidak dapat berisi subelemen atau atribut. Hanya tipe custom saja yang dapat berisi subelemen atribut.

Tipe-Tipe Sederhana

Tipe-tipe sederhana merupakan datatype-datatype yang dapat dipakai untuk menjelaskan tipe data yang dimuat dalam sebuah elemen atau sebuah atribut. Instance dari tipe sederhana tidak dapat memuat atribut-atribut atau elemen-elemen lain. Contoh dari tipe sederhana meliputi int, long, string, dan dateTime. Sebuah tipe sederhana dapat juga mendefinisikan sebuah enumerasi atau sebuah union.

Definisi tipe sederhana selalu dihasilkan dari tipe sederhana lain. Tiga tipe derivasi yang dimungkinkan oleh XML Schema adalah by restriction, by list, dan by union.

Tipe sederhana yang dihasilkan dari tipe-tipe basis by restriction dapat mendefinisikan pembatasan tambahan yang diberlakukan pada nilai-nilai yang dikenakan atas nilai-nilai yang dapat dimuat oleh instance bertipe sederhana. Oleh karena itu, instance bertipe sederhana yang dihasilkan lewat pembatasan (*by restriction*) hanya dapat berisi sebuah subset dari nilai-nilai yang dapat dimuat oleh tipe basisnya.

Sebagai contoh, tipe double built in adalah sebuah versi yang telah dibatasi dari tipe decimal. Instance tipe decimal tidak terikat (*unbounded*) dan instance bertipe double dibatasi pada nilai-nilai yang memenuhi tipe floating 64 bit presisi tunggal IEEE. Berikut ini adalah sepasang definisi tipe sederhana yang dihasilkan dengan pembatasan.

```
<? xml version='1.0 '?>
<schema xmlns="http://www.w3.org/2001/XMLSchema.xsd">
<simpleTypename="Mylnt">
<restrictionbase="int"/>
</simpleType>
<simpleTypename="GenericProductld">
```

```
<restrictionbase="string">
<minlength value="l"/>
<maxlengthvalue="20"/>
<! restriction>
<! simpleType>
<simpleTypename="Percent">
<restrictionbase="integer">
<minlnclusivevalue="0 "/>
<maxinclusivevalue="l 00 "/>
<! restriction>
</simpleType>
</schema>
```

Definisi tipe sederhana pertama, Mylnt, mendefinisikan sebuah datatype yang tak satu pun pembatasan tambahan berlaku. Nilai tipe Mylnt dapat memuat segala nilai yang didefinisikan oleh tipe basisnya, int. Definisi tipe sederhana kedua, GenericProductld, mendefinisikan sebuah versi yang telah dibatasi dari datatype string. Nilai-nilai tipe tersebut dapat berisi sebuah string dari 1 sampai 20 karakter. Definisi Percent adalah serupa, ia membatasi nilai-nilai tipe pada integer antara 0 dan 100.

XML Schema menerangkan keseluruhan host dari pembatasan (diperlihatkan dalam tabel 9-3) yang dapat diterapkan pada definisi-definisi tipe sederhana.

Tabel 9-3 Batasan Datatype XML Schema

Constraints	Definition
length	Instance tipe harus berisi jumlah
minLength	satuan berpanjang tetap. Instance
maxLength	tipe harus berisi jumlah satuan

pattern	minimum. Instance tipe hanya dapat berisi jumlah satuan maksimum. Instance tipe hanya dapat berisi data yang sesuai dengan pola tertentu yang didefinisikan melalui sebuah ekspresi biasa. Sebagai contoh, ekspresi biasa yang dapat digunakan untuk mendefinisikan Social Security Number dapat berupa [0-9]{3}-[0-9]{2}- {0-9]{4}.
enumeration	Instance tipe hanya dapat berisi kumpulan nilai yang disebutkan.
whitespace	Instance tipe dihasilkan dari string yang diproses oleh XML parser. Salah satu dari tiga cara bergantung pada nilai atribut. Nilai default preserve menyatakan bahwa nilainya dibiarkan tidak berubah. Nilai replace menyatakan bahwa semua kejadian tab, umpan-baris, dan paragraf baru akan diganti dengan spasi. Nilai collapse menyatakan bahwa

	segala sekuen spasi akan dijadikan spasi tunggal.
minlnclusive	Instance tipe tidak dapat berisi sebuah nilai kurang dari nilai yang telah disebutkan. Nilai minlnclusive setidaknya harus serestriktif tipe basisnya.
minExclusive	Instance tipe tidak dapat berisi sebuah nilai kurang dari atau sama dengan nilai yang disebutkan. Nilai minExclusive setidaknya harus serestriktif tipe basisnya.
maxlnclusive	Instance tipe tidak dapat berisi sebuah nilai kurang dari nilai yang disebutkan. Nilai maxlnclusive setidaknya harus serestriktif tipe basisnya.
maxExclusive	Instance tipe tidak dapat berisi sebuah nilai kurang dari atau sama dengan nilai yang disebutkan. Nilai maxExclusive setidaknya harus serestriktif tipe basisnya.
totalDigits	Menyebutkan jumlah digit maksimum yang dapat dimuat sebuah ins- tance tipe. Tipenya

	harus dihasilkan dari decimal.
fractionDigits	Menyebutkan jumlah digit maksimum di sebelah kanan titik desimal yang dapat dimuat sebuah instance tipe. Tipe harus dibuat dari decimal.

Salah satu batasan (*constraint*) menarik adalah pattern. Batasan pattern memungkinkan Kita mendefinisikan ekspresi biasa yang akan dipakai untuk membatasi nilai-nilai potensial dari tipe tertentu. Dengan mengontrol pembatasan pattern di dalam definisi tipe sederhana, Kita dapat secara signifikan mengurangi jumlah kode pengesahan yang Kita perlukan untuk menulis layanan Web Kita. Mari kita perhatikan sebuah contoh di mana pembatasan pola dapat membantu. Ingatlah bahwa metode Orderitem yang dimunculkan oleh layanan Web Commerce menerima sebuah parameter bernama Item. Dalam contoh terdahulu, Penulis telah mendefinisikan sebuah tipe bernama Productld untuk mendefinisikan tipe data yang dapat dimuat dalam elemen Item.

Di samping pembatasan panjang, anggaplah bahwa sebuah instance Productld tidak dapat berisi karakter-karakter berikut: I \ [ ] : ; _ = , + * < >. Jika Penulis memakai tipe Productid; seperti yang telah didefinisikan sebelumnya, Penulis perlu menuliskan kode untuk memastikan bahwa tidak ada karakter terlarang yang dimasukkan ke dalam elemen Item. Sebagai gantinya, Penulis akan menambahkan batasan pattern ke definisi Productld yang membatasi ripe karakter-karakter yang dapat dimuat oleh

nilai-nilai tipe itu.

```
<? xml version='l.O'?>
<schema xmlns="http://www.w3.org/2001/XMLSchema.xsd"
xmlns:tns="urn:Commerce"targetNamespace="urn:Commerce" >
<simpleTypename="Productld">
<restrictionbase="string">
<minlengthvalue="l"/>
<maxlengthvalue="20"/>
<patternvalue='[A/\&#x5B ;&#x50 ; : ; i =.+*?&gt;&l t ; J + '/>
<! restriction>
</simpleType>
<! - - Request Message (work- in - progress) - >
<elementname='Item'type='tns:Productld'/>
<! - - Response Message(work-in-progress) - >
<elementname= 'Amount'type='double'nillable='true'/>
<! schema >
```

Definisi tipe items membuat sebuah enumerasi tipe Productid dengan tiga kemungkinan nilai. Elemen Item yang didefinisikan bertipe items, sehingga hanya dapat berisi nilai Apple, Banana, atau Orange.

Tipe-tipe sederhana dapat juga diperoleh dari daftar. Pembuatan dari daftar menyatakan bahwa nilai tipe tersebut dapat berisi satu atau beberapa nilai tipe basis, di mana masing-masing nilai dibatasi oleh spasi kosong. Sebuah contoh adalah atribut encodingStyle SOAP. Ingatlah bahwa atribut ini dapat menerima sebuah daftar URI yang dibatasi-spasi kosong. Contoh berikut mendefinisikan atribut encodingStyle SOAP tersebut:

```
<simpleTypename= 'encodingStyle'>
<listbase'uri-reference'/>
<lsimpleType>
```

Tipe-tipe daftar bukanlah pengganti bagi array SOAP yang dikodekan. Array SOAP menyediakan sebuah metode untuk pengodean bagi berbagai instance tipe sederhana, juga tipe-tipe kompleks. SOAP Encoding juga mendefinisikan sintaks bagi serialisasi array parsial.

Tipe-tipe sederhana dapat juga dihasilkan lewat union (*by union*). Sebuah instance dari tipe yang dihasilkan lewat union (penggabungan) dapat berisi nilai dari salah satu tipe yang dimuat dalam union tersebut. Contoh berikut ini mendefinisikan dua union, MyUnion dan PhoneNumber:

```
<? xml version-'1.0 ?>
<schema xmlns-"http://www.w3.org/2001/XMLSchema.xsd">
<simpleTypename-"MyUnion">
<unionmemberTypes-"stringint"/>
<lsimpleType>
<simpleTypename-"PhoneNumber">
<union>
<simpleTypename-"UsPhone Number"/>
<restrictionbase-"string"/>
<patternvalue-"([0-9 ](3})[0-9 ]{3}- [0- 9 ]{4}"/>
<!restriction>
<lsimpl eType>
<s1mpleTypename-"UkPhoneNumber" >
<restrictionbase- "string">
<patternvalue-"+[0-9 ]{2)([0-9 J l[0- 9 ]{31[0- 9 ]{3}[0-9 ]1 41 "/>
<! restriction>
</simpleType>
<! union>
</simpleType>
<! schema>
```

Skema terdahulu memperlihatkan dua cara untuk mendefinisikan tipe-tipe union sederhana. Definisi pertama memakai atribut memberType untuk

menampilkan tipe-tipe yang dimuat dalam union tersebut. Elemen MyElement dapat berisi nilai-nilai string atau int. Definisi tipe kedua mendefinisikan sebuah union yang terdiri dari definisi-definisi tipe sederhana yang disisipkan. Kedua tipe sisipan tersebut mendefinisikan sebuah nomor telepon U. S dan nomor telepon U.K. PhoneNumber union dapat berisi nilai-nilai seperti misalnya (303) 555-1212 atau +44 (0)121 643 2345.

Definisi tipe dapat bernama atau anonim. Jika sebuah definisi tipe ditanamkan dalam definisi lain (sebuah definisi elemen, misalnya), Kita tidak perlu memberikan nama ke tipe tersebut. Berikut ini versi yang telah dimodifikasi dari skema layanan Web Commerce yang mendefinisikan enumerasi sebagai sebuah tipe anonim:

```
<? xml version-'1.0'?>
<schema xmlns-"http://www.w3.org/2001/XMLSchema.xsd "
xmlns:tns-"urn:Commerce"targetNamespace-"urn:Commerce">
<! - - Portions of the schema have been removed for clarity. - - >
<! - - Request Message(work-in-progress)- - >
<elementname-'Item'>
<simpleType>
<restrictionbase-"Productld">
<enumerationvalue-"Apple"/>
<enumerationvalue-"Banana"/>
<enumerationvalue-"Orange"/>
<! restriction>
</simpleType>
<element>
<! schema >
```

Enumerasi berisi nilai-nilai yang memungkinkan elemen Item

didefinisikan sebagai sebuah tipe anonim. Karena enumerasi didefinisikan dalam lingkup definisi elemen, maka atribut type tidak perlu disebutkan karena telah diterapkan. Karena tipe enumerasi hanya dapat dirujuk oleh elemen itu sendiri, maka tidak perlu menyebutkan nama bagi tipe tersebut. Kita perlu mendefinisikan tipe-tipe enumerasi dengan hati-hati. Tipe-tipe yang dipakai hanya sekali adalah kandidat yang baik bagi definisi-definisi tipe anonim. Bagaimanapun juga, jika datatype mungkin akan dipakai kembali dalam konteks lain, sebaiknya Kita menghindari untuk menyatakan definisi-definisi tipe anonim.

Kita juga harus berhati-hati dengan pemakai tipe-tipe sederhana, termasuk tipe-tipe built in dalam layanan-layanan Web bergaya RPC. Parameter-parameter yang disalurkan dengan nilai dapat didefinisikan memakai tipe-tipe sederhana. Namun, parameter-parameter yang disalurkan dengan referensi tidak dapat didefinisikan. Ingatlah bahwa SOAP Encoding menyebutkan cara mengodekan parameter-parameter yang disalurkan melalui referensi memakai atribut id dan href Karena elemen-elemen yang didefinisikan memakai tipe-tipe sederhana tidak dapat berisi atribut, maka mereka tidak dapat dikodekan dengan benar dalam pesan SOAP. Karena alasan ini, skema SOAP Encoding mendefinisikan tipe-tipe wrapper bagi tipe-tipe built in yang didefinisikan oleh XML Schema.

Tipe-Tipe Kompleks

Tipe kompleks adalah sebuah pengelompokan logis dari pernyataan elemen dan/atau atribut. Orang dapat mengatakan bahwa dokumen instance XML tidak benar-benar menarik atau berguna tanpa tipe-tipe

kompleks. Sebagai contoh, skema SOAP Envelope mendefinisikan banyak tipe kompleks. Envelope adalah sebuah tipe kompleks karena harus berisi elemen-elemen lain, seperti misalnya elemen Body dan mungkin juga berisi elemen Header. Penulis akan memakai tipe- tipe kompleks untuk mendefinisikan tubuh jawaban dan permintaan pesan pesan SOAP bagi layanan Web Commerce.

Tipe kompleks didefinisikan memakai elemen complexType. Elemen complex Type berisi pernyataan bagi semua elemen dan atribut yang dapat dimuat dalam elemen tersebut. Sebagai contoh, tubuh dari pesan-pesan permintaan dan respon Purchaseltem dapat diterangkan dengan membuat sebuah tipe kompleks. Berikut ini definisi skema bagi layanan Web Commerce:

```
<? xml version='1.0'?>
<schema xmlns="http://www.w3.org/2001/XMLSchema.xsd"
xmlns:tns="urn:Commerce"targetNamespace="urn:Commerce">
<! --TypeDefinitions -- >
<simpleTypename="Productld">
<restrictionbase="string">
<minlengthvalue="l"/>
<maxlengthvalue="20"/>
<patternvalue='[A/\&#x5B;& #x5D ; : ; 1 =.+*?&gt;&l t;J +'/>
<! restriction>
<lsimpleType>
<simpleTypename="Items">
<restrictionbase="Productld">
<enumerationvalue="Apple"/>
<enumerationvalue="Banana"/>
<enumerationvalue="Orange"/ >
<! restriction>
</simpl eTy pe>
```

```
<! - - Request Message(work-in-progress)- - >
<elementname='Purchase Item'>
<complexType>
<elementname='Item'type='tns:Productld'!>
<elementname='Ouantity'type='int'/.>
</complexType>
<! element>
<! - - Response Message (work-in-progress)- - >
<elementname='PurchaseltemResponse'>
<complexType>
<elementname='Amount'type='double'nillable='true '/>
</complexTy pe>
<! element >
<! schema>
```

Skema tersebut mendefinisikan dua tipe kompleks yang mendefinisikan tubuh pesan permintaan dan jawaban SOAP. Supaya sesuai dengan spesifikasi SOAP, Penulis telah mendefinisikan sebuah elemen Purchaseltem untuk memuat semua parameter yang disalurkan ke metode PurchaseItern pada layanan Web Commerce. Tubuh pesan jawaban akan berisi sebuah elemen bernama PurchaseItern Response dan akan berisi satu subelemen bagi tipe hasilnya.

Tipe-tipe kompleks dapat dibagi ke dalam dua kategori, yaitu tipe-tipe yang berisi elemen-elemen lain dan tipe-tipe yang sebaliknya. Dalam sebuah definisi tipe kompleks, Kita dapat menyebutkan sebuah elemen complexContent atau simpleContent. Definisi-definisi datatype sebelumnya tidak berisi elemen-elemen ini. Jika tidak ada elemen yang dipakai dalam definisi tipe kompleks, maka dianggap complexContent. Oleh karena itu, definisi yang lebih panjang dari elemen Purchaceltern

berikut sama dengan definisi sebelumnya:

```
<? xml version='l.O'?>
<schema xmlns="http://www.w3.org/2001/XMLSchema.xsd "
xmlns:tns="urn:Commerce "targetNamespace="urn:Commerce">
<! - - Portions of the schema have been removed for clarity. - - >
<! -- Request Message(work - in - progress)- - >
<elementname='Purchaseltem'>
<complexType>
<complexContent>
<extension>
<elementname='Item'type='tns:Productld'!>
<elementname='Quantity'type='int'!>
<! extension>
</complexContent>
</complexType>
<! element>
<! schema>
```

Perhatikan bahwa skema tersebut juga berisi elemen extension. Jika disebutkan simpleContent atau complexContent, elemen anaknya pasti berupa elemen restriction atau extension. Secara default, tipe kompleks akan mendefinisikan sebuah versi perluasan dari tipe basisnya. Jika tipe basis tidak disebutkan, definisi tipe akan memperluas anyType. Dengan kata lain, sebuah definisi tipe kompleks yang tidak secara eksplisit menyatakan apakah isinya sederhana atau kompleks, maka secara default akan memuat isi kompleks dan dihasilkan dari anyType dengan ekstensi.

Seperti pada definisi tipe sederhana, Kita dapat membuat tipe-tipe kompleks yang lebih restriktif daripada tipe basisnya. Tidak seperti tipe-tipe sederhana yang membatasi nilai string sebuah instance tipe, tipe-tipe kompleks mempunyai pembatasan yang menyangkut definisi elemen dan

atribut dalam tipe tersebut.

Contoh berikut ini mendefinisikan tipe kompleks Family dan kemudian mendefinisikan beberapa tipe yang diturunkan dengan pembatasan:

```
<? xml version='l.O '? >
<schema xmlns='http://www.w3.org/2001/XMLSchema'>
<! - - Basetype - - >
<complexTypename='Family'>
<elementname='Parent'minOccurs='l'maxOccurs='2 I >
<elementname='Child'type='string'minOccurs='O'!>
<complexTypename='Children'>
< ! - - The number of parents is restricted to one. - - >
<complexType name='SingleParentFamily' >
<complexContent>
<restriction base='Family>
<elementname='Parent' type='string'minOccurs='l' maxOccurs='l'I>
<elementname='Child'type='string'minOccurs='O'!>
</restriction>
</complexContent>
</complexType>

<! - - No Child elements are all owed. - - >
<complexTypename='Childless Family'>
<complexContent>
<restriction base='Family'>
<elementname='Parent'type='string'minOccurs='l'maxOccurs='lI>
<elementname='Child'ty pe='string'minOccurs='O'maxOccurs='O'!>
<! restriction>
</complexContent>
<! complexType>

< ! - - The name of the children can only be George. - - >
<complexType name='Foreman Family'>
```

```
<complexContent>
<restriction base='Family'>
<elementname='Parent·type='string'minOccurs='0'maxOccurs='2'I>
<elementname='Child'type='string'minOccurs='0'fixed='George'!>
<! restriction>
</complexContent>
</complexType>

< ! - - Nota legal type declaration - - >
<complexTypename='OrphanedFamily>
<complexContent>
<restriction base='Family>
<elementname='Parent'type='string'minOccurs='0'maxOccurs='0'I>
<elementname='Child'type='string'minOccurs='0'!>
<! restriction>
</complexContent>
</complexType>
<! schema>
```

Penulis telah mendefinisikan tiga turunan terbatas yang sah dari tipe Family. Datatype SingleParentFamily membatasi jumlah elemen Parents yang dapat muncul dalam sebuah instance. Datatype ChildlessFamily melarang elemen Child opsional agar tidak muncul dalam sebuah instance. Kemudian, dalam gaya George Foreman tulen, datatype ForemanFamily mengijinkan elemen-elemen Child, asalkan masing-masing namanya adalah George.

Oleh karena itu, satu peringatan pada tipe-tipe terlarang dan pada tipe-tipe perluasan, adalah bahwa tipe-tipe turunan harus dapat menjadi pengganti bagi tipe basisnya tanpa mengeluarkan apa pun. Dua tipe turunan dalam contoh, SingleParentFamily dan ForemanFamily, sesuai dengan ketentuan

ini. Definisi tipe OrphanedFamily tidak memenuhi persyaratan ini karena tipe basis Family menyatakan bahwa Kita setidaknya harus mempunyai satu elemen Parent. Oleh karena itu, sebuah instance dari OrphanedFamily tidak dapat bertindak sebagai pengganti.

Ingatlah kembali bahwa SOAP Encoding menyediakan sebuah cara untuk mengatur identitas parameter-parameter yang disalurkan berdasarkan referensi. Ini dilakukan dengan atribut-atribut id dan href Atribut-atribut ini memungkinkan sebuah elemen merujuk data yang dikodekan pada lokasi lain di dalam atau bahkan di luar pesan SOAP. Contoh berikut ini mengilustrasikan perlunya mekanisme semacam itu:

```
I I Server Code:
public void TestReference(refintx. refinty)
[
x+=3;
y+=lO;


//Client Code: intz2;
Test Reference (ref z, ref z);
//zshould now equa l 20 (2 +3 +10).
```

Agar metode TestReference dapat dijalankan dengan benar, indentitas z harus dijaga. Oleh karena itu, elemen-elemen bagi parameter-parameter x dan y tidak dapat bertipe int yang didefinisikan oleh XML Schema karena elemen-elemen tidak akan dapat memuat atribut href dan id untuk didefinisikan. Dengan dernikian, skema SOAP Encoding memperluas tipe-tipe built-in. Contoh berikut menjalankan redefinisi yang sama:

```
<? xml version='l.O'?>
```

```
<schema xmlns-"http://www.w3.org/2001/XMLSchema.xsd"targetNamespace-
'urn: ExtendedBuiltinTypes'>
<complexTypenameint'>
<simpleContent>
<extensionbase-'int')
<attributename-'id 'type-'ID'!>
<attributename-·href'type-·urlReference'I>
<! extension>
<simpleContent>
</complexType>
</schema>
```

SOAP Encoding menyebutkan bahwa susunan di mana parameter-parameter sebuah pesan bergaya RPC muncul adalah signifikan. Oleh karena itu, Penulis memakai elemen sequence dalam skema tersebut untuk menyatakan bahwa elemen Item harus muncul pertama kali, diikuti elemen Quantity. Kita juga dapat menyebutkan suatu kombinasi atribut minOccurs dan maxOccurs. Dalam hal ini, tak satupun atribut yang disebutkan. Jadi, nilai default untuk 1 akan diberikan. Berikut ini adalah skema lengkap untuk layanan Web Commerce:

```
<? xml  version-'1.0  '?>
<schema  xml ns-"http://www.w3.org/2001/X
MLSchema .xsd "  xmlns:tns-"urn :Commerce
"ta rgetNa me space-"u rn:Commerce">
   <! - - Ty pe  Defi ni ti ons  - - >
   <s i mpl eTy pe  na me-"Producti d ">
< restri ct i on  base- "stri n g ">
         <m i n len g th  val ue-"1"/>
         <ma x length  v al u e-"20"/>
         <p atte rn  va l u e-'[A/\&flx5B; &flx5D;: ;[-,+*?&gt ;&l t ; J +'/>
<! rest ricti on>
   </s i mpl eTy pe>
   < si mpl eTy pe  na me-"Items">
```

```
< restri cti on  base-"Prod uct i d">
         <en umer a ti on  va l ue-"Appl e"/>
         <enumerat i on  v al u e-"Banana" />
         <en um er ati on val ue-"Ora nge "/>
      <! restriction>
   <ls i mpl eTy pe>

   <! - - Req uest  Message  (work- in - progress)>
   <element  name-'Purchasei tem'>
      <complexTy pe>
         <sequ ence>
            <elementna me-'Item  'type-'tns:Prod uct i d  '/>
            <el em en t name- 'Quantity
            'type-'int'!>
         <! sequence>
      <lcomplexTy pe>
   <! element >
   <! - - Response Message (work-in-progress)- - >
   <elementname-" PurchaseltemResponse'>
   <complexType>
   <elementname-'Amount 'type- 'double'nillable-'true '/>
   <!complexType>
   <! element>

   <! schema>
```

Elemen lain yang dapat dipakai untuk menghasilkan perilaku spesifik yang berkaitan dengan elemen-elemen yang didefinisikan meliputi elemen-elemen choice dan all. Elemen choice hanya memungkinkan salah satu elemen yang telah didefinisikan dalam tipe kompleks untuk muncul dalam sebuah instance tipe. Elemen all memungkinkan suatu subset elemen yang didefinisikan dalam tipe untuk muncul dalam sembarang urutan.

Ada satu perbedaan lagi antara elemen all dan elemen sequence serta choice. Pernyataan tipe kompleks yang dibuat dalam elemen-elemen yang disebutkan terakhir dapat berisi atribut maxOccurs dan minOccurs. Bagaimanapun juga, elemen-elemen yang didefinisikan dalam elemen all hanya dapat menyebutkan sebuah atribut maxOccurs dan sebuah atribut

minOccurs dengan nilai 0 atau 1. Secara default, datatype-datatype yang didefinisikan memakai elemen complexContent tidak mengijinkan isi campuran. lsi campuran berarti nilai-nilai berisi teks dan elemen-elemen anak. Kita dapat menimpa perilaku ini dengan menambahkan sebuah atribut mixed dan mengatur nilainya ke true. Pada umumnya, termasuk yang satu ini, lebih disukai karena melarang isi campuran.

Kadang kala, kita perlu menyebutkan apakah suatu elemen atau atribut dapat muncul dalam sebuah instance pada sebuah tipe kompleks. Sebagai contoh, tipe anyType menunjukkan bahwa suatu elemen atau atribut dapat muncul dalam sebuah elemen bertipe anyType. Kita dapat melakukan hal ini memakai elemen-elemen any dan anyAttribute dalam definisi tipe kompleks. Berikut ini definisi dari tipe anyType:

```
<? xml version-'1.0'?>
<sc hema xmlns- 'http://www.w3.org/2001/XMLSchema
xmlns:tns-'http://schemas.xml soap.org/soap/envelope/'targetNames
pace-'http://schemas.xml soap.org/soap/envelope/'>
<xs:complexTypename-"anyType"mixed-"true">
<xs:annotation>
<xs:doc umentati on>
Not the real urType, but as close an a pproximati on as we can
get in the XMLrepresentation</xs:documentation>
<lxs:annotation>

<xs:sequence>
<xs:anyminOccurs-"0 "maxOccurs-"unbounded"/>
<!xs:sequence>
<xs:anyAttribute/>
<lxs:complexType>
<!schema>
```

Porsi terdahulu dari skema untuk XML Schema sendiri mendefinisikan tipe kompleks anyType. Elemen any menyatakan bahwa suatu elemen dapat muncul dalam sebuah elemen bertipe anyType. Atribut-atribut minOccurs dan max Occurs juga dipakai untuk menunjukkan bahwa nol

atau beberapa elemen dapat muncul.

Kita juga dapat memberlakukan batasan tambahan atas atribut-atribut dan elemen-elemen yang dapat muncul dalam sebuah dokumen instance memakai atribut namespace. Atribut ini memungkinkan Kita menyatakan atribut-atribut dan elemen-elemen namespace-scoped apa yang dapat dan tidak dapat dimuat dalam sebuah dokumen instance. Tabel 9-4 mencantumkan nilai-nilai yang memungkinkan pada atribut namespace.

Tabel 9-4 Nilai-Nilai Atribut namespace

Atribut *namespace*	**Keterangan**
##any (default)	Elemen induk dapat berisi segala elemen/atribut XML well- formed dari segala namespa ce.
##focal	Elemen induk dapat berisi segala elemen/atribut XML well- formed yang tidak dimiliki sebuah namespace.
##targetNamespace	Elemen induk dapat berisi segala elemen/atribut well-formed yang didefinisikan dalam target namespace skema di mana tipe tersebut didefinisikan .
##other	Elemen induk dapat berisi segala elemen/atribut well-formed yang tidak didefinisikan dalam target

	namespa ce skema di mana tipe tersebut didefinisikan.
Daftar URI berbata s-spasi	Elemen induk dapat berisi segala elemen/atribut well-formed dari namespa ce yang disebutkan.

Atribut-atribut lain yang dapat disebutkan dalam elemen any atau anyAttribute adalah processContents. Atribut processContents menunjukkan bagaimana doku- men instance harus diproses oleh sistem. Tabel 9-5 memperlihatkan nilai-nilai yang memungkinkan dari atribut processContents.

label 9-5 Nilai-Nilai Atribut procesContent

Atribut process Contents	Keterangan
strict (default)	Sistem harus mensahkan semua elemen / atribut terhadap name-space mereka
skip	Sistem harus berusaha mensahkan semua element / atribut bagi namespace mereka. Jika usaha tersebut gaga!, tidak akan dihasilkan pesan kesalahan
lax	Sistem tidak akan berusaha mensahkan elemen / atribut

	namespace mereka

Grup-Grup Elemen dan Atribut

Kita mungkin suatu saat merasa perlu untuk menambahkan sekumpulan atribut atau elemen yang sama ke banyak definisi tipe yang kompleks sekaligus. XML Schema menyediakan elemen-elemen group dan attributeGroup bagi pengelompokan elemen dan atribut secara logis. Grup-grup atribut dan elemen menyediakan sebuah cara yang pas untuk mendefinisikan sekumpulan atribut atau elemen sekali, dan merujuk mereka berkali-kali dalam definisi-definisi tipe kompleks.

Satu contoh di mana grup atribut dipakai adalah dalam skema SOAP Encoding. Skema tersebut berisi definisi-definisi tipe kompleks yang memperluas tipe-tipe built-in XML Schema, sehingga mereka dapat disalurkan dengan referensi ke dalam tubuh pesan SOAP. Berikut ini sebuah definisi grup atribut yang didefinisikan oleh skema SOAP Encoding:

```
<attributeGroup name='commonAttributes'>
<attributename='id'type='ID'! >
<attributename='href'type='urlReference'!>
<anyAttributenamespace='##other'!>
<lattributeGroup>
```

Kutipan sebelumnya mendefinisikan grup atribut commonAttributes. la berisi definisi atribut bagi atribut-atribut id dan href yang diperlukan bagi pengodean parameter-parameter yang disalurkan dengan referensi. Berikut ini definisi tipe yang ditunmkan dari tipe data string yang merujuk grup atribut tersebut:

```
<elementname='string'type='tns:string'!>
<complexTypename='string'>
<complexContent>
<extensionbase='string'>
<attributeGroupref='tns:commonAttributes'/>
<! extension>
</complexContent>
</complexType>
```

Definisi tipe di atas sesungguhnya adalah sebuah versi update dari yang muncul dalam skema SOAP Encoding. Skema aslinya ditulis terhadap versi terdahulu dari spesifikasi XML Schema. Definisi tipe kompleks meru juk grup atribut memakai atribut ref yang berisi nilai definisi grup atribut yang ditargetkao. Elemen-elemen dapat dikelompokkan bersama memakai elemen group dan dapat dirujuk dengan afribut ref pula.

Namespace Scoping

Pernyataan-pernyataan elemen dan atribut yang secara lokal dimasukkan dalam sebuah definisi tipe kompleks dapat berupa qualified atau unqualified. Nilai defaultnya adalah unqualified yang berarti bahwa elemen atau atribut tersebut tidak berafiliasi dengan namespace apapun. Berikut ini contohnya:

```
<? xml version='l.O'?>
<schema xmln s= 'http://www.w3.org/2001/XMLSchema
xmlns:tns='urn:Example:Scoping'targetNamespace='urn:Example:Scopi
ng'>
<elementname='GloballyScoped'!>
<elementname='MyElement'>
<complexType>
<elementname='LocallyScoped'!>
<elementref='tns:GloballyScoped'!>
<lcomplexType>
<! element>
<! schema>
```

Skema ini mendefinisikan dua elemen dicakup secara global, GloballyScoped dan MyElement. Pernyataan elemen MyElement mendefinisikan sebuah tipe kompleks anonim yang berisi dua pernyataan elemen, yaitu elemen yang dicakup secara lokal bernama LocallyScoped dan sebuah referensi ke sebuah elemen yang dicakup secara global bernama - apa lagi kalau bukan-GloballyScoped. Berikutnya, Penulis akan membuat sebuah dokumen instance.

```
<? xml version='l.O'?>
<ex:MyElementxmlns:ex='urn:Example:Scoping'>
<LocallyScoped/>
<ex:GloballyScoped/>
</ex:MyElement>
```

Dokumen instance berisi sebuah elemen MyElement. Perhatikan bahwa elemen-elemen anak dari elemen MyElement dikualifikasikan berbeda. Elemen LocallyScoped tidak mempunyai prefiks karena tidak berafiliasi dengan namespace apapun. Elemen GloballyScoped dikualifikasikan penuh dalam prefix ex: Karena elemen GloballyScoped telah didefinisikan sebagai sebuah elemen global, ia berafiliasi dengan namespace urn: Example: Scoping.

Perhatikan bahwa, jika Kita mengatur namespace default dalam sebuah dokumen instance, maka elemen dan atribut yang dicakup secara lokal tidak akan dimiliki oleh namespace default. Sebagai contoh, dokumen instance berikut bukanlah bagian dari namespace urn: Example: Scoping:

```
<? xml version=1.0?>
<My Element xmlns='urn:Example:Scoping'>
<! - - Invalid because Local lyScoped is not affiliated with the
default namespace - - >
< LocallyScoped/>
<GloballyScoped/>
<! My Elemen t>
```

Ada dua cara untuk menghindari masalah ini. Pertama, memberikan sebuah prefiks ke referensi namespace daripada memberikan sebuah namespace default. Dalam contoh pertama, Penulis mengasosiasikan prefiks ex: dengan namespace urn: Example: Scoping. Solusi kedua adalah menimpa deklarasi namespace default dalam setiap elemen atau atribut lokal, seperti dalam contoh ini:

```
<? xml version='l.0'?>
<My Elementxmlns='urn:Example :Scoping'>
<! - - Validbecause LocallyScoped overrides the default namespace
- - >
<LocallyScoped xmlns=' ·! >
<GloballyScoped/ >
<! MyElement>
```

Kita dapat menghindari masalah dengan elemen dan atribut yang dicakup secara lokal ini dengan cara mengafiliasikan mereka dengan namespace di mana mereka didefinisikan. Kita dapat melakukannya dengan mengatur atribut form dalam pernyataan elemen atau atribut ke qualified. Ini mengharuskan elemen atau atribut yang dicakup secara lokal untuk dikualifikasikan sesuai dengan namespace-nya. Berikut ini sebuah versi yang telah diupdate dari skema tersebut;

```
<? xml version='l.0'?>
<schemaxmlns='http://www.w3.org/2001/XMLSchema
xmlns:tns='urn:Example:Scoping2'targetNamespace='urn:Example:Scop
ing2'>
<elementname='GloballyScoped'!>
<elementname='MyElement'>
<complexType>
<elementname='LocallyScoped'form='qualified'/>
<elementref='tns:GloballyScoped'!>
</complexType>
<! element>
<! schema>
```

Saat ini, Penulis menunjukkan bahwa elemen LocallyScoped harus

dikualifikasikan penuh dalam dokumen instance. Berikut ini adalah sebuah dokumen instance yang telah diupdate untuk menyesuaikan perubahan yang dilakukan pada skema:

```
<? xml version='1.0'?>
<MyElementxmlns='urn:Example:Scoping2'>
<! - - Valid since Locally Scoped element mustbe fullyqualified-
- >
<LocallyScoped/>
<GloballyScoped/>
<! MyElement>
```

Kita juga dapat menimpa nilai default untuk atribut form. Kita dapat melakukannya dengan mengatur dua atribut dalam elemen schema: elemenForm Default dan attributeFormDefault. Skema-skema otomatis akan dihasilkan oleh platform .NET bagi layanan-layanan Web yang secara umum mengatur elemen FormDefault dan attributeFormDefault ke qualify.

Polimorfisme

Polimorfisme adalah ketika berbagai instance dengan tipe berbeda dapat dianggap sama. XML Schema menyediakan dua mekanisme untuk memungkinkan perilaku polimorfisme, yaitu grup-grup pewarisan dan subtitusi.

Seperti yang telah Penulis peragakan dalam bagian-bagian terdahulu, XML Schema menyediakan sebuah model turunan yang kaya. Kita dapat membuat tipe-tipe sederhana baru yang dihasilkan melalui pembatasan dan Kita dapat membuat tipe-tipe baru yang diturunkan melalui ekstensi serta pembatasan.

Salah satu dari sekian aturan pada sebuah tipe turunan adalah bahwa ia

harus dapat menjadi pengganti bagi tipe basisnya. Akibatnya, sebuah instance dari tipe turunan dapat digantikan dalam sebuah dokumen instance bagi tipe basisnya. Sistem akan diberitahu bahwa dokumen instance tersebut berisi sebuah instance tipe turunan lewat atribut xsi: type. Sebagai contoh, anggaplah Kita ingin membuat sebuah sistem tipe umum untuk menerangkan ban dalam. Kita memerlukan suatu dealer atau manufaktur ban dalam supaya dapat memakai sistem tipe ini untuk membuat sebuah layanan Web agar mendapatkan kuota harga bagi ban dalam yang mereka jual. Berikut ini datatype-datatype umum yang dipakai untuk menerangkan ban dalam tersebut:

```
<? xml version=·1.0?>
<schema
xmlns='http://www.w3.org/2001/XMLSchematargetNamespace='urn:TireT
ypes'>
<complexTypename='Tire'abstract='true'>
<elementname='WheelDiameter'type='int'/>
<elementname='Width'type='int'!>
</complexType>
<complexTypename='AutoTire'>
<complexContent>
<extensionbase='Tire'>
<elementname='WheelDiameter'type='int'!>
<elementname='Width'type='int'!>
<elementname='AspectRatio'type='int'!>
<!extension>
</complexContent>
</complexType>
<elementname='MountainBikeTire'>
<complexContent>
<extensionbase='Tire'>
<elementname='WheelDiameter'type='int'!>
<elementname='Width'type='int'!>
<elementname='Position'!>
<simpleType>
<restrictionbase='string'>
<enumerationvalue='Front'/>
<enumerationvalue='Rear'/>
<! restriction>
```

```
</simpleType>
</element>
<! extension>
</complexContent>
<! element>
<! schema>
```

Skema ini mendefinisikan tipe basis Tire. la berisi dua elemen anak bagi ukuran rim dan lebar ban dalam tersebut. la kemudian menghasilkan dua tipe terpisah dari tipe basis Tire yang disebut Auto Tire dan MountainBike. Dalam kedua instance, tipe Tire diperluas untuk menambahkan elemen-elemen tambahan yang diperlukan untuk menerangkan tipe spefisik dari ban-dalam tersebut.

Instance-instance dari tipe basis Tire berisi informasi yang tidak memadai untuk menerangkan tipe ban dalam yang spesifik. Oleh karena itu, properti abstract dalam deklarasi tipe diatur ke true. Mengatur properti abstract dari definisi tipe kompleks Tire ke true menunjukkan bahwa tipe Tire tidak dimaksudkan untuk dapat dibuat secara langsung.

Sebuah perusahaan fiktif, The Round Rubber Tire Company, menjual semua tipe ban dalam dan ingin memunculkan sebuah layanan Web untuk menampilkan kuota harga atas ban dalam. Berikut ini skema untuk layanan Web NewTires yang mengontrol tipe-tipe ban dalam:

```
<? xml  version=' 1.0  '?>
<sc ema  xmlns='http://www.w3.org/2001/X M LSc hema'xml ns:
vt= 'urn :Ti reTypes  ta rgetNa me
space='http://roundrubbertire.com/N ewTi res
el ementFormDefa ult='q ua l ifi ed'>
<elementna me='GetOuote'>
      <complexType>
         <elementna me='T ire'type='vt:Tire' ! >
         <elementna me='Quantity'type='int' ! >
      <lcom pl exTyp e>
<! element>
```

```
<elementna me='GetQuoteResults'>
      <compl exTy pe>
         <elementna me='Result'type='double' ! >
      <! complexType>
<! elem ent>
</schema>
```

Metode GetQuote menerima informasi mengenai ban dalam yang diminta serta jumlahnya. Harga ban dalam baru yang kemudian dihasilkan berupa double. Bagaimanapun juga, karena datatype Tire adalah abstrak, maka layanan Web perlu menerima sebuah derivatif dari tipe Tire. Pesan SOAP berikut meminta sebuah kuota bagi ban dalam baru bertipe AutoTire:

```
<? xml  version="l.O  "encoding="utf-8  "?>
<soa p: En vel ope  xml ns:soap="htt p: //sch emas.xml soa
p.org/soap /envelope/ "xml ns:xsi ="http://www.w3.org/2001/X
MLSchema - i nstance  "
xmlns:ti res="http://bigrubbertire. com/ NewTires"xml ns:vt= "urn:
TireTypes''>
   <so a p:Body>
      <tires:GetOuote>
         <tires:Tirexs i:type="vt:AutoTire"/>
            <tires:Wheel Diameter>16</tires:WheelDia me ter>
            <tires:Width>225</tires:Tires>
            <tires:AspectRatio>SO</tires:AspectRatio>
         </tires:Tire>
         </tires:Quantity>4</tires:Quantity>
      </tires:GetQuote>
   </soap:Body>
</soap:Envelope>
```

Tubuh pesan SOAP berisi elemen GetQuote yang mengandung parameter Tire.

Parameter Tire berisi sebuah instance bertipe AutoType, seperti ditunjukkan oleh atribut xsi: type. Parameter adalah sebuah subtitusi legal karena AutoTire adalah turunan dari Tire.

XML Schema juga mendukung perilaku polimorfisme pada tingkat

elemen lewat konsep grup-grup subtitusi. Grup subtitusi adalah sebuah grup elemen yang dapat berlaku sebagai pengganti bagi elemen-elemen yang diberikan dalam sebuah dokumen instance. Kita dapat menambahkan definisi elemen ke grup pengganti memakai atribut subtitutionGroup.

Atribut subtitutionGroup berisi sebuah referensi ke elemen tempat ia berlaku sebagai pengganti. Semua definisi elemen dalam sebuah grup subtitusi harus bertipe sama atau turunan tipe elemen targetnya harus sama. Dalam contoh berikut, skema untuk layanan Web NewTires ditulis ulang supaya dapat memakai subtitusi grup sebagai ganti subtitusi tipe:

```
<? xml  version-'1 .0 '?>
<schemaxml ns-'http://www.w3.org/2001/X MLSchema'xml: v -
'urn:TireTypes'target Na mes pace-'http://rubbertire.com/NewTires'
xmlns:tns-"http://rubbertire.com/NewTires'elementForm Default-
'qualified'>
<elementname-'GetQuote'>
<complexType>
<elementref- 'tns:Tire'! >
<elementna me-'Quantity'type-'int' ! >
</complexType>
<! element>
<elementname- 'GetQuoteResults'>
<complexType>
<elementname-' Result'typ e- 'dou ble'/>
</com p l exTy pe>
<! element>
<!-- DeclaretheTireelement a nd its substitutes. - ->
<elementna me- 'Tire'type- vt:Tir e'abstract-'true' ! >
<elementname- 'AutoTire'typ e-'vt :AutoTire'subs titutionG rou p-
'tns:Ti re'/>
<elementname-'MountainBikeTiretype-'vt :MountainBikeTire
substitutionGroup-'tn s :Tire' !>
<! schema>
```

Dalam skema yang baru, definisi elemen Tire telah dipindah dari dalam tipe kompleks GetQuote (yang dicakup secara lokal) langsung di bawah elemen schema (yang dicakup secara global). Karena Penulis tidak ingin

elemen Tire muncul dalam dokumen instance, maka Penulis mengatur properti abstract ke true dalam definisi elemen. Penulis kemudian mendefinisikan dua elemen lain untuk bertindak sebagai subtitusi bagi elemen Tire. Berikut ini pesan SOAP hasilnya untuk pemesanan ban dalam mobil:

```
<? xml   version-"1.0   "encoding-"utf-8"?>
<soap:En velopexmlns:soap-"http://schemas.xml
soap.org/soap/en velope/ "xmlns:ti res-"http://bigru
bbertire.com/NewTires">
<soap: Body>
   <tires:GetQuote>
      <tires :AutoTire/>
         <tires:Whee1 Diameter>16</tires:Whee1 Diameter>
         <tires :Width>225</tires:Tires>
         <tires:AspectRatio>50</tires:AspectRatio>
      <ltires:AutoTire>
      <tires:Quantity>4</tires:Ouantity>
   </tires:GetQuote>
</soap:Body >
</soap:Envelope>
```

Dalam dokumen tersebut, Elemen Tire telah digantikan oleh elemen Auto Tire. Kita tidak perlu menghias elemen tersebut dengan atribut xsi: type karena elemen tersebut telah ditipekan kuat oleh skemanya sendiri.

Membatasi Pewarisan

Karena suatu tipe turunan dapat menjadi pengganti bagi tipe basisnya, maka Kita kadang kala perlu menyatakan bagaimana sebuah kelas basis dapat diwariskan. Sebagai contoh, namespace urn: TireTypes yang telah didefinisikan sebelumnya mendefinisikan datatype Tire. Datatype Tire didefinisikan sebagai abstrak karena sebuah instance dari tipe itu tidak akan berisi cukup informasi untuk menerangkan sebuah ban dalam dengan memadai. Bagaimanapun juga, mengatur tipenya ke abstract tidak

memberikan sebuah solusi tuntas.

Client dapat dengan mudah menghindari pemakaian tipe yang lebih kaya dengan menurunkan sebuah tipe baru dari Tire lewat pembatasan. Kemudian, client tersebut dapat memanggil metode GetQuote dan menyalurkan sebuah instance bertipe baru. Berikut ini contohnya:

```
<? xml  version='l.O  '?>
<sc hema  xmlns='http://www.w3.org/2001/XMLSchema'xml
ns:tire='urn:TireTypes targetNa
mespace='urn:DerivedTireTy pes'>
   <complexTypename='SkinnyTire'abstract= 'true' I >
      <complexContentbase='tir :Tire»
         <restriction>
            <elementname='Wheel  Dia meter'type='int'  ! >
            <elementna me='Width'type='int'fixed='l'/)
         </restriction>
      </complexContent>
   </complexType>
</schema>
```

Pertama-tama, Penulis menurunkan sebuah versi yang lebih terbatas dari tipe Tire. Penulis kemudian menyalurkan sebuah instance tipe baru ini ke metode GetQuote:

```
<? xml version-"1.0"encoding-"utf- 8"?>
<soap:Envelope xmlns:soap- "http://schemas.xml
soap.org/soap/envelope/ "xmlns:tires-
"http://bigrubbertire.com/ NewTires"
xml ns :vt-"urn :De r i vedTi reTy pes   "
xml ns :xsi -"http://www.w3.org/2001/XMLSchema - instance">
       <soap:Body>
     <tires:GetQuote>
<tires:Tirexsi:type-"dt: Ski nnyTire"/>
<tires :W heel Diameter>16</tires:Wheel Diameter>
     <tires:Width>l</tires:Tires>
       <!tires:Tire>
  <tires:Quantity>Z</tires:Ouantity>
     </tires:GetQuote>
       </soap: Body>
<!soap :Envelope>
```

Sebagaimana yang Penulis nyatakan sebelumnya, layanan Web tidak dapat menghasilkan harga ban dalam hanya dengan berdasarkan pada diameter dan lebar rodanya. Oleh karena itu, jika layanan Web menerima sebuah instance Skinny Tire, yaitu sebuah turunan yang telah dibatasi datatype Tire, maka ia tidak akan dapat memberikan sebuah kuota harga. Satu solusi adalah melarang pewarisan dengan pembatasan.

Sebuah contoh di mana Kita mungkin perlu melarang pewarisan dengan ekstensi adalah jika Kita memakai sebuah tipe yang menyatakan versi panjang dari form pajak pemerintah federal Amerika Serikat. Semua pengisi barangkali tidak perlu mengisi seluruh form yang panjang itu. Jadi, pemerintah mengeluarkan form EZ. Form EZ adalah turunan dari form panjang tersebut dengan pembatasan pada jumlah data yang dapat dimuatnya.

Kita dapat mendiktekan bagaimana sebuah datatype dapat diwarisi dengan mengatur atribut final dalam elemen complexType. Tabel 9-6 menerangkan nilai- nilai yang memungkinkan untuk itu.

label 9-6 Nilai-Nilai Atribut final

Atribut final	Keterangan
#all	Tipe tidak dapat bertindak sebagai tipe basis bagi tipe-tipe yang diturunkan dengan ekstensi atau pembatasan
restriction	Tipe dapat berupa sebuah tipe basis hanya untuk tipe-tipe yang

	diturunkan dengan ekstensi.
extension	Tipe dapat berupa sebuah tipe basis hanya untuk tipe-tipe yang dihasilkan melalui pembatasan.

Jika atribut final tidak diatur, nilai defaultnya adalah #all. Kita dapat menimpa nilai default dengan mengatur atribut Jina/Default dalam elemen schema.

Kadang kala, wajar saja mengijinkan yang lain untuk mewarisi dari datatype, tapi akan membatasi instance-instance dari tipe-tipe yang diturunkan untuk muncul dalam dokumen instance. Sebagai contoh, jasa konsultasi pajak seringkali mengumpulkan informasi melebihi apa yang dibutuhkan pada form panjang pajak penghasilan. Oleh karena itu, jasa konsultasi pajak mungkin perlu mengambil dari datatype long form melalui ekstensi untuk dipakai dalam sistem internalnya sendiri.

Jika tiba saatnya untuk memfilekan form pajak secara elektronis, skema bagi layanan Web tersebut perlu sebuah cara untuk melarang instance-instance versi yang telah diperluas dari datatype long form. Ini dilakukan dengan mengatur atribut block pada pernyataan elemen form pajak ke extension. Nilai-nilai lain yang memungkinkan pada atribut block dicantumkan dalam Tabel 9-7.

label 9-7 Nilai-Nilai Atribut block

Atribut bwck	Keterangan
restriction	Elemen tidak dapat berisi sebuah

	instance dari tipe yang diturunkan melalui pembatasan
extension	Elemen tidak dapat berisi sebuah instance dari tipe yang diturunkan melalui ekstensi.
substitution	Elemen tidak dapat digantikan untuk elemen lain dalam grup subtitu si.
#all	Elemen tidak dapat berisi sebuah instance dari tipe turunan dan tidak dapat digantikan untuk elemen lain dalam grup subtitusinya.

Jika atribut block tidak disebutkan, maka perilaku defaultnya adalah mengijinkan elemen untuk memuat sebuah instance dari tipe turunan atau digantikan dengan elemen lain dalam grup subtitusinya. Kita dapat menimpa nilai defaultnya dengan mengatur atribut Jina/Attribute dalam elemen schema ke nilai yang diinginkan.

C. PENUTUP

Ringkasan

XML Schema menyediakan sarana yang luwes dan tuntas untuk menjelaskan struktur serta tipe data yang akan muncul dalam sebuah dokumen instance. Ia superior terhadap bahasa skema DTD yang pertama

kali diperkenalkan bersama XML 1.0.

XML Schema menyediakan sebuah sistem tipe. Sistem tipe dipakai untuk mendefinisikan sebuah cara yang bersifat platform-independent untuk menerangkan tipe data yang dapat dimuat dalam sebuah elemen atau atribut. Sistem tipe juga menyediakan seperangkat tipe built-in yang menentukan data yang dapat dimuat oleh instance-instance tipe tersebut, seperti misalnya string, int, dan float. Karena SOAP adalah sebuah protokol berbasis-XML, pesan-pesan dapat dibuat dan dipakai tanpa mempedulikan perangkat keras, sistem operasi, atau perangkat lunak pemrosesan yang dipakai.

Sistem tipe juga dapat diperluas (extensible). XML Schema menyediakan sarana untuk mendefinisikan tipe-tipe sederhana dan tipe-tipe kompleks baru. Sebuah tipe sederhana tidak dapat memuat elemen anak. Tipe-tipe kompleks menyediakan sebuah cara yang logis untuk mengelompokkan elemen-elemen dan atribut-atribut yang berhubungan.

Tipe custom selalu mewarisi tipe custom lainnya atau tipe built in. Tipe-tipe sederhana dapat dihasilkan melalui pembatasan memakai sebuah sintaks yang kaya untuk mendefinisikan batasan tambahan. Mereka juga dapat dihasilkan dengan daftar atau union.

Tipe-tipe kompleks dapat dihasilkan melalui pembatasan atau ekstensi. Mereka juga hanya dapat berisi atribut-atribut isi sederhana atau atribut-atribut dan elemen-elemen isi kompleks. Atribut-atribut dan elemen-elemen ini dapat didefinisikan secara lokal atau dapat berupa referensi-referensi dari berbagai entitas yang didefinisikan secara global. Jika entitas-entitas dalam sebuah tipe kompleks didefinisikan secara lokal,

mereka akan diasosiasikan dengan namespace dengan menyeting atribut form mereka ke qualified. Ini memudahkan penulisan dokumen instance yang merujuk sebuah namespace default.

XML Schema memungkinkan perilaku polimorfisme dengan memperkenankan elemen-elemen untuk memuat berbagai instance bertipe turunan untuk muncul dalam dokumen. XML Schema juga memungkinkan elemen-elemen digantikan dengan elemen-elemen bertipe kompatibel lewat grup-grup subtitusi. Untuk memfasilitasi perilaku polimorfik, instance dari tipe turunan harus dapat menggantikan sebuah instance dari tipe basisnya.

XML Schema juga menyediakan mekanisme untuk membatasi pewarisan dan perilaku polimorfik. Definisi tipe kompleks dapat membatasi bagaimana tipe dapat diwariskan dengan mengatur atribut final. Definisi-definisi elemen dapat juga membatasi tipe subtitusi yang diperkenankan dengan mengatur atribut block.

Dokumen skema dapat berisi definisi-definisi elemen, atribut, dan tipe. Definisi-definisi ini dapat dicakup dalam sebuah namespace khusus dengan mengatur atribut targetNamespace dalam elemen schema. Skema tersebut kemudian dapat dirujuk oleh namespacenya dalam sebuah dokumen instance. Kita dapat merujuk sebuah namespace dengan menambahkan atribut xmlns ke sebuah elemen dalam dokumen tersebut. Referensi tersebut akan diperluas ke elemen yang berisi atribut xmlns dan segala elemen atau atribut yang dimuat dalam elemen tersebut.

Referensi ke namespace skema dapat diberikan sebuah moniker. Kemudian, segala entitas yang dirujuk dalam skema harus didahului

dengan moniker. Berdasar ketentuan, namespace XML Schema diberi moniker xsd dan namespace XML Schema Instance diberi moniker xsi. Skema tersebut berisi referensi ke definisinya sendiri. Referensi ke namespace-nya sendiri biasanya diberikan moniker tns.

Referensi ke sebuah namespace yang tidak diberi moniker dipakai untuk mendefinisikan namespace default. Atribut dan elemen yang tidak full-qualified dengan sebuah prefiks yang ada dalam lingkup deklarasi namespace default adalah qualified, berkenaan dengan namespace default.

XML Schema menyediakan sarana pembuatan skema yang terdiri lebih dari satu dokumen skema. Elemen include dipakai untuk memasukkan definisi skema lain ke dalam namespacenya. Skema-skema yang tidak mendefinisikan namespace dapat dimasukkan ke dalam suatu skema. Jika skema yang telah dimasukkan mendefinisikan sebuah namespace, ia harus sesuai dengan namespace target dari skema yang memasukkannya.

XML Schema merupakan cara yang disukai untuk menerangkan skema pesan yang dipertukarkan di antara client dan server. Ia menyediakan cara yang handal dan fleksibel untuk menjelaskan struktur dan tipe data yang dapat muncul dalam sebuah dokumen instance. Seperti yang akan Kita lihat nanti dalam bab berikutnya, platform .NET menyediakan kerangka kerja yang kaya untuk membuat dan memakai skema-skema XML Schema bagi layanan-layanan Web.

Pertanyaan

1. Tuliskan skema dari XML?
2. Tuliskan contoh dokumen XML.

3. Sebutkan dan berikan contoh jenis Datatype Built-In.
4. Sebutkan dan jelaskan batasan Datatype XML Schema?

DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.

11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 10

Membuat dan Menampilkan Dokumen XML Kita yang Pertama

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 10 akan di bahas Membuat dan Menampilkan Dokumen XML, yang mencakup diantaranya: Pembuatan Dokumen XML, Membuat Dokumen XML, Anatomi DokumenXML, Elemen Dokumen, Aturan Dasar XML, Menampilkan Dokumen XML, Menampilkan Dokumen Menggunakan Style Sheet Bertumpuk.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan Pembuatan Dokumen XML, Membuat Dokumen XML, Anatomi DokumenXML, Elemen Dokumen, Aturan Dasar XML, Menampilkan Dokumen XML, Menampilkan Dokumen Menggunakan Style Sheet Bertumpuk.

B. PENYAJIAN

Membuat dan Menampilkan Dokumen XML Kita yang Pertama

Dalarn bab ini, Kita akan rnendapatkan sebuah ringkasan seluruh proses pernbuatan dan penampilan sebuah dokurnen XML dalarn browser Web. Pertarna Kita akan rnernbuat sebuah dokumen XML sederhana, menjelajahi struktur dokumen, dan mempelajari sejumlah aturanfundamental untuk pembuatan dokumen XML yang well-formed atau rapi. Kemudian Kita akan mengetahui cara menampilkan dokumen itu alam browser Web Microsoft

Internet Explorer 5, dengan membuat dan menyisipkan sebuah style sheet sederhana yang menyatakan pada browser bagaimana memformat elemen-elemen dalam dokumen tersebut.

Pembuatan Dokumen XML

Karena sebuah dokumen XML ditulis dalam teks biasa, Kita bisa membuatnya menggunakan editor teks kesukaan Kita. Misalnya, Kita bisa memakai Notepad yang ada pada Microsoft Windows. Atau yang lebih baik lagi, Kita bisa menggunakan editor pemrograman dengan fitur yang memudahkan pengetikan kode sumber, semacarn editor teks Microsoft Visual Studio (editor teks disertakan pada Microsoft Visua l C++, Microsoft Visual InterDev, Microsoft Visual] ++, dan aplikasi Visual Studio lainnya).

Membuat Dokumen XML

1. Buka file teks baru yang kosong dalam editor teks Kita, dan ketikkan dalam dokumen XML seperti tampak dalam Listing 10-1. Jika mau, Kita bisa mengabaikan sejumlah elemen BOOK. Kita tidak perlu mengetikkan semuanya cukup tiga atau empat saja (elemen BOOK terdiri dari tag <BOOK> dan </BOOK> ditambah dengan semua teks di antaranya).
2. Gunakan perintah Save pada editor teks untuk menyimpan dokumen tersebut pada hard disk, namai dengan Inventory.xml.

lnventory.xml

```
<? xml version="l.O"?>
<! - - FileName:Inventory.xml - - >
<INVENTORY>
<BOOK>
```

```
<TITLE> The Adventures of Huckleberry
Finn</TITLE>
<AUTHOR>Mark Twain</AUTHOR>
<BINDING>mass market paperback</BINDING>
<PAGES>298</PAGES>
<PRICE>$5.49</PRICE>
</BOOK>
<BOOK>
<TITLE>Leaves of Grass</TITLE>
<AUTHOR>Walt Whitman</AUTHOR>
<BINDING>hardcover</BINDING>
<AGES>46 2</PAGES>
<PRICE>$7.75</PRICE>
</BOOK>
<BOOK)
<TITLE>The Legend of Sleepy Hollow</TITLE>
<AUTHOR>Washington Irving</AUTHOR>
<BINDING>massmarketpaperback</BINDING>
<PAGES>98</PAGES>
<PRICE>$2.95</PRICE>
</BOOK>
```

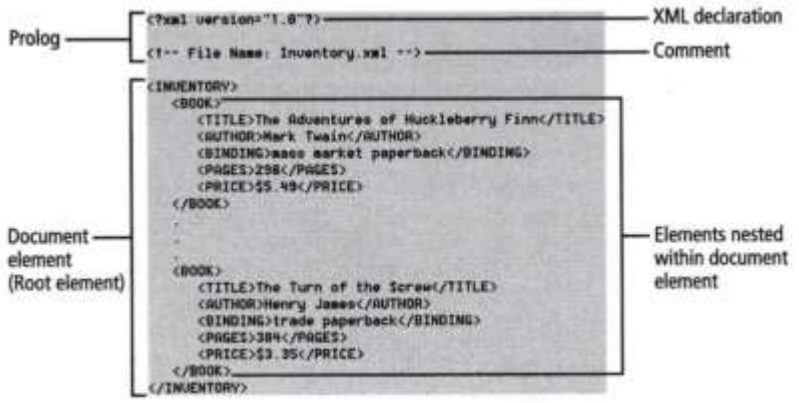
Listing 10-1 .

```
<BOOK>
<TITLE>The Marble Faun</TITLE>
<AUTHOR>Nathaniel Hawthorne,(/AUTHOR>
<BlNDING>tradepaperback</BINDING>
<PAGES>473</PAGES>
<PRICE>Sl0.95</PRICE>
</BOOK>
<BOOK>
<TITLE>Moby Dick</TITLE>
<AUTHOR>Herman Melville</AUTHOR>
```

```
<BINDING>hardcover</BINDING>
<PAGES>724</PAGES>
<PRICE>$9.95</PRICE>
</BOOK>
<BOOK>
<TlTLE>The Portrait of a Lady</TITLE>
<AUTHOR>Henry James</AUTHOR>
<BINDING>mass market paperback</BINDING>
<PAGES>256</PAGES>
<PRICE>$4.95</PRICE>
</BOOK>
<BOOK>
<TITLE>The Scarlet Letter</TITLE>
<AUTHOR>Nathaniel Hawthorne</AUTHOR>
<BINDING>trade paper back</BINDING>
<PAGES>253</PAGES>
<PRICE>$4.25</PRICE>
</BOOK>
< BOOK>
<TITLE>The Turn of the Screw</TITLE>
<AUTHOR>Henry James</AUTHOR>
<BINDING>trade paperback</BINDING>
<PAGES>384</PAGES>
<PRICE>$3.35</PRICE>
< BOOK>
</INVENTGRY>
```

Anatomi Dokumen XML

Sebuah dokumen XML, seperti pada dokumen contoh yang telah Kita ketik, terdiri dari dua bagian utama: prolog dan elemen dokumen (elemen dokumen juga dikenal dengan elemen root).



Prolog

Prolog pada dokumen contoh terdiri dari tiga baris :

```
< ? x l version-"1.0"? >
< ! - -   File Name:Inventory.xml    - - >
```

Baris pertama adalah deklarasi XML, yang menyatakan bahwa ini adalah sebuah dokumen XML dari menampilkan nomor versinya (pada saat penulisan ini, versi XML terbarunya adalah 1.0). Deklarasi XML bersifat opsional, walaupun spesifikasinya menyatakan bahwa ia harus dimasukkan. Jika Kita memasukkan sebuah deklarasi XML, ia harus muncul pada awal dokumen.

Baris kedua prolog terdiri dari spasi kosong. Untuk meningkatkan keterbacaannya (readibilitas), Kita bisa menyisipkan sejumlah spasi kosong antara item-item dalam prolog. Prosesor XML akan mengabaikannya.

Baris ketiga prolog adalah sebuah komentar. Penambahan komentar ke sebuah dokumen XML bersifat opsional, namun dengan melakukannya bisa

meningkatkan keterbacaan dokumen. Sebuah komentar diawali dengan karakter <! - dan diakhiri dengan karakter ->. Kita bisa mengetikkan sembarang teks yang Kita mau (kecuali -) di antara kedua kelompok karakter tersebut; prosesor XML akan mengabaikannya.

Prolog bisa juga berisi sejumlah komponen opsional berikut ini:

1. Sebuah deklarasi tipe dokumen, yang mendefinisikan tipe dan struktur dokumen tersebut. Jika dipakai, deklarasi tipe dokumen harus diletakkan setelah deklarasi XML.
2. Satu atau lebih instruksi pemrosesan, yang memberikan informasi bahwa prosesor XML menyalurkannya ke aplikasi tersebut. Selebihnya dalam bab ini, akan Kita lihat sebuah instruksi pemrosesan untuk pengaitan sebuah style sheet ke dokumen XML.

Elemen Dokumen

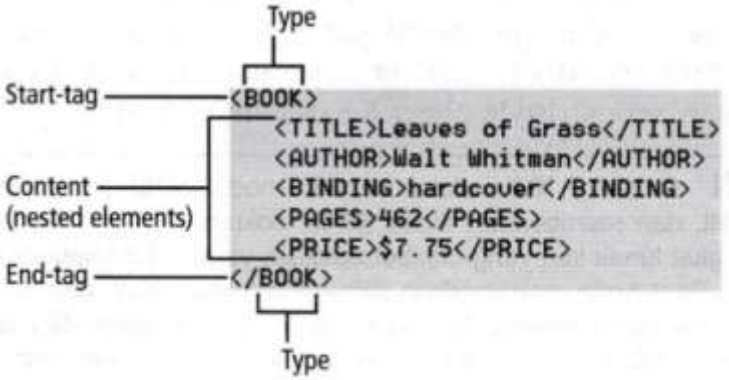
Bagian utama kedua sebuah dokumen XML adalah sebuah elemen tunggal yang disebut dengan elemen dokumen atau elemen root, yang bisa berisi elemen-elemen tambahan.

Dalam sebuah dokumen XML, elemen-elemen menKitakan struktur logika sebuah dokumen dan berisi isi informasi dokumen (yang dalam contoh dokumen adalah berupa informasi buku, sepeti judul, nama pengarang, dan harga). Kebanyakan elemen berisi sebuah tag-awal, isi elemen, dan sebuah tag-akhir. Isi elemen dapat berupa data karakter, elemen (tersarang) lain, atau kombinasi keduanya.

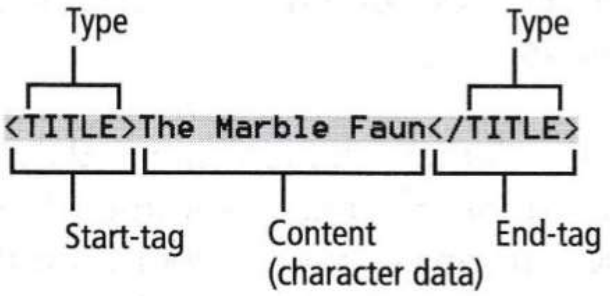
Pada contoh dokumen, elemen dokumen adalah INVENTORY. Tag-awalnya adalah <INVENTORY>, tag-akhirnya adalah </INVENTORY>, dan isinya

adalah delapan elemen BOOK yang tersarang.

Setiap elemen BOOK sepertinya berisi serangkaian elemen yang tersarang.



Setiap elemen yang tersarang dalam elemen BOOK, seperti halnya elemen TITLE hanya berisi data karakter.



Kita akan mempelajari tentang penambahan elemen ke dokumen XML Kita, dan memasukkan atribut dalam sebuah tag-awal elemen.

Sejumlah Aturan Dasar XML

Berikut ini adalah sebagian kecil aturan dasar untuk pembuatan dokumen XML yang rapi atau well-formed. Dokumen yang well-formed adalah dokumen yang tunduk pada seperangkat aturan minimal yang memungkinkan dokumen diproses oleh browser atau program lainnya. Dokumen yang telah Kita ketik sebelumnya (Listing 10-1) adalah sebuah contoh dokumen XML yang well-formed yang tunduk pada aturan ini.

1. Dokumen tersebut harus mempunyai persis satu elemen tingkat teratas (elemen dokumen atau elemen root). Semua elemen lainnya harus disarangkan.
2. Elemen-elemen harus disarangkan dengan tepat. Yakni, jika sebuah elemen diawali dalam elemen lain, ia harus juga berakhir dalam elemen yang sama.
3. Setiap elemen harus memiliki tag-awal dan tag-akhir. Tidak seperti HTML, XML tidak mengizinkan Kita mengabaikan tag-akhir bahkan tidak dalam situasi di mana browser tahu kapan elemen harus berakhir (dalam Bab ini Kita akan mempelajari sebuah notasi shortcut yang bisa Kita pakai untuk elemen kosong yakni, sebuah elemen tanpa isi).
4. Nama tipe-elemen dalam tag-awal hams persis berhubungan dengan nama dalam tag-akhir yang bersangkutan.
5. Nama-nama tipe-elemen bersifat case-sensitive. Pada kenyataannya, semua teks dalam markup XML adalah case-sensitive. misalnya, elemen berikut ini adalah ilegal karena nama tipe dalam tag-awal tidak sesuai dengan nama-tipe dalam tag-akhir:

```
<TITLE>Leaves of Grass</Title>
<! -illegal element->
```

TIP Kita akan menemukan instruksi yang lebih rinci untuk menulis tidak hanya dokumen XML yang well-formed juga dokumen XML yang valid, yang memenuhi sekumpulan persyaratan yang ketat.

Menampilkan Dokumen XML

Kita bisa membuka sebuah dokumen XML dalam browser Internet Explorer 5, seperti Kita membuka sebuah halaman Web HTML.

Jika dokumen XML tidak berisi link ke sebuah style sheet, Internet Explorer hanya akan menampilkan teks dari dokumen yang utuh, termasuk markup (tag-tag dan komentar, misalnya) dan data karakter. Internet Explorer 5 mengkodekan dalam warna perbedaan komponen dokumen untuk membantu Kita mengenali mereka, dan menampilkan elemen dokumen dalam bentuk pohon yang bisa diperluas/disempitkan untuk menKitai dengan jelas struktur logika dokumen, dan untuk memungkinkan Kita melihat berbagai tingkat detil.

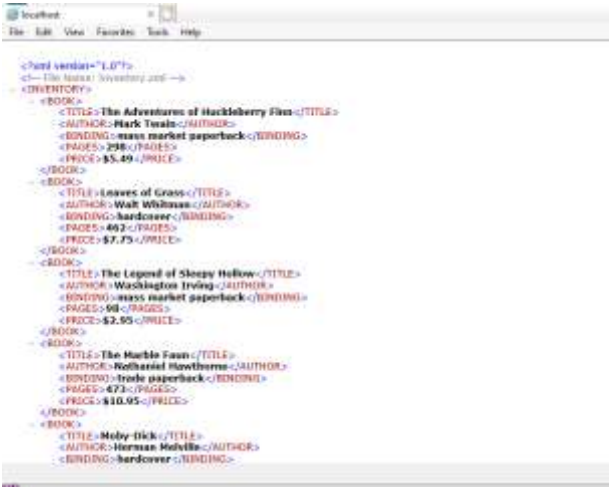
Jika dokumen XML berisi sebuah link ke sebuah style sheet, Internet Explorer 5 hanya akan menampilkan data dari elemen dokumen, dan ia akan memformat data ini sesuai dengan aturan yang Kita berikan dalam style sheet tersebut. Kita juga bisa menggunakan style sheet bertumpuk (CSS-style sheet yang sama yang dipakai untuk HTML), atau sebuah style sheet Extensible Stylesheet Language (XSL) (jenis style sheet yang lebih canggih yang memakai sintaks XML dan hanya bisa dipakai untuk dokumen XML).

Menampilkan Dokumen XML Tanpa Menggunakan Style Sheet

1. Dalam Windows Explorer atau dalam jendela folder, klik-ganda nama

Iventory.Xml

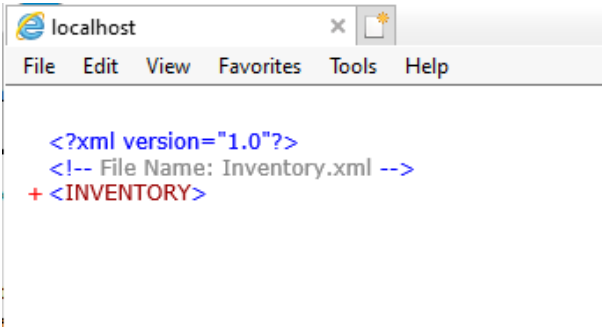
file Inventory.xml, yang telah Kita simpan dalam latihan sebelumnya. Internet Explorer 5 akan menampilkan dokumen tersebut seperti ini:



2. Bereksperimenlah dengan mengubah tingkatan detilnya seperti tampak dalam elemen dokumen. Klik simbol minus (-) pada bagian kiri tag-awal untuk meringkas elemen, sedangkan klik simbol (+) di depan elemen yang diringkas akan memperluasnya. Sebagai contoh, jika Kita mengklikck depan elemen INVENTORY, seperti tampak di sini:

```
<?xml version=111.011?>
<! -- file Name:Inventory.xml-->
<INVENTORY>
<BOOK>
<TITLE>The Adventures of Huckleberry
Finn</TITLE>
<AUTHOR>Mark Twain</AUTHOR>
```

seluruh elemen dokumen akan diringkas, sebagaimana terlihat di sini:



Menangkap Kesalahan XML dalam Internet Explorer 5"'

Sebelum Internet Explorer 5 menampilkan dokumen XML Kita, pengurai XML bawaannya akan menganalisa isi dokumen. Jika pengurai mendeteksi adanya kesalahan, Internet Explorer 5 akan menampilkan sebuah halaman dengan pesan kesalahan ketimbang berusaha menampilkan dokumen tersebut. Internet Explorer 5 akan menampilk an halaman kesalahan baik dokumen XML itu terhubung ke style sheet ataupun tidak.

Dalam latihan berikutnya, Kita akan mempela jari fitur pemeriksaan kesalahan Internet Explorer 5 dengan tujuan memperkenalkan sebuah kesalahan ke dalam dokumen Inventory.xml.

1. Dalam editor teks, buka dokumen Inventory.xml yang telah Kita buat pada latihan sebelumnya. Ganti elemen TITLE pertama dari

   ```
   <TITLE>The Adventures of Huekleberry Finn
   </TITLE>
   menjadi
   <TITLE>The Adventures of Huckleberry
   Finn</Title>
   ```

2. Simpan dokumen yang telah diganti tersebut.

3. Dalam Windows Explorer atau dalam jendela folder, klik-ganda nama file Inventory.xml.

```
Inventory xml
```

Ketimbang menampilkan dokumen XML, Internet Explorer 5 sekarang akan menampilkan halaman pesan kesalahan seperti berikut:

4. Berhubung Kita akan memakai Inventory.xml lagi dalam bab selanjutnya, Kita sebaiknya sekarang mengembalikan tag-akhir dalam elemen TITLE pertama menjadi bentuk aslinya (</TITLE>), dan kemudian menyimpan kembali dokumen tersebut.

Walaupun Kita tidak mengaitkan sebuah style sheet ke dokumen XML, Internet Explorer 5 menggunakan sebuah style sheet XSL default untuk menampilkan dokumen tersebut; dengan demikian halaman kesalahan menunjukkan "using XSL style sheet."

TIP Seperti yang Kita kerjakan sepanjang bab buku ini, ingatlah bahwa Kita bisa dengan cepat memeriksa apakah sebuah dokumen XML sudah well-formed hanya dengan membukanya langsung dalam Internet Explorer 5. (Jika Kita menampilkan sebuah dokumen XML melalui sebuah halaman HTML, seperti telah dijelaskan dalam Bagian sebelumnya, sebuah dokumen XML yang mengandung kesalahan akan gagal ditampilkan, namun Kita tidak akan melihat pesan kesalahan kecuali jika Kita secara eksplisit menuliskan kode script untuk menampilkannya.)

Menampilkan Dokumen Menggunakan Style Sheet Bertumpuk

1. Buka sebuah teks kosong yang baru dalam editor teks Kita, dan ketikkan dalam CSS seperti tampak dalam Listing 10-2.
2. Gunakan perintah Save pada editor teks Kita untuk menyimpan style sheet tersebut pada hard disk, namakan dengan InventoryOl.css.

   CSS yang telah Kita buat menyatakan pada Internet Explorer 5 untuk memformat data karakter elemen sebagai berikut:

   a. Tampilkan setiap elemen BOOK dengan spasi 12 point di atasnya (margin-top: 12pt) dan sebuah baris pemisah di atas dan di bawah-nya (display: blok) menggunakan huruf 10-point (font-size: I Opt).
   b. Tampilkan setiap elemen TITLE dalam huruf miring (font-style: italic).
   c. Tampilkan setiap elemen AUTHOR dengan cetak tebal Contoh weight: bold).

Inventory01.css

```
/* File Name: Inventory01.css */
BOOK
   {display:block;
    margin-top:12pt;
    font-size:15pt}
TITLE
   {font-style:italic}
AUTHOR
   {font-weight:bold}
```

Listing 10-2.

3. Dalam editor teks, buka dokumen Inventory.xml yang telah Kita buat dalam latihan sebelumnya. Tambahkan instruksi pemrosesan berikut ke bagian akhir prolog dokumen (sisipkan saja di atas elemen INVENTORY):

```
<? xml –style sheet
type="text/css"href="InventoryOl.css"?>
```

   Instruksi pemrosesan ini menghubungkan CSS yang Kita buat ke dokumen XML. Sebagai hasilnya, saat Kita membuka dokumen tersebut dalam Internet Explorer 5, browser akan menampilkan isi dokumen sesuai dengan instruksi pada style sheet.

4. Untuk mencerminkan nama file baru yang akan Kita berikan, gantilah komponen dekat awal dokumen dari

```
<! - File Name: Inventory.xml->
```

   menjadi

```
<! – File Name: InventoryOl.xml->
```

   Listing 10-3 memperlihatkan dokumen XML selengkapnya. Gunakan

perintah Save pada editor teks untuk menyimpan salinan dari dokumen yang telah dimodifikasi ini dengan nama file Inven tory01.xml. Pastikan untuk menyimpannya dalam nama folder yang sama dengan Kita menyimpan InventoryOl .css.

lnventory01.xml

```
<?xml version="1.0"?>
<!-- File Name: Inventory01.xml -->
<?xml-stylesheet type="text/css"
href="Inventory01.css"?>
<INVENTORY>
   <BOOK>
      <TITLE>The Adventures of Huckleberry
Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR>Walt Whitman</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>462</PAGES>
      <PRICE>$7.75</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Legend of Sleepy
Hollow</TITLE>
      <AUTHOR>Washington Irving</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
```

```
      <PAGES>98</PAGES>
      <PRICE>$2.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Marble Faun</TITLE>
      <AUTHOR>Nathaniel Hawthorne</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>473</PAGES>
      <PRICE>$10.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Moby-Dick</TITLE>
      <AUTHOR>Herman Melville</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>724</PAGES>
      <PRICE>$9.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Portrait of a Lady</TITLE>
      <AUTHOR>Henry James</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
      <PAGES>256</PAGES>
      <PRICE>$4.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Scarlet Letter</TITLE>
      <AUTHOR>Nathaniel Hawthorne</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>253</PAGES>
      <PRICE>$4.25</PRICE>
   </BOOK>
   <BOOK>
```

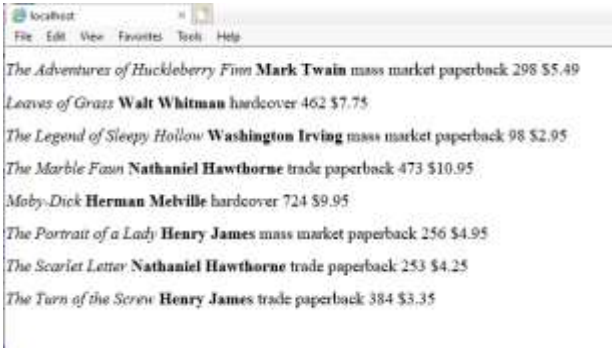
```
        <TITLE>The Turn of the Screw</TITLE>
        <AUTHOR>Henry James</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>384</PAGES>
        <PRICE>$3.35</PRICE>
    </BOOK>
</INVENTORY>
```

Listing 10-3.

5. Dalam Windows Explorer atau dalam jendela folder, klik-ganda InventoryOl.xml untuk membuka dokumen tersebut.

```
Iventory01.xml
```

Internet Explorer 5 akan membuka dokumen InventoryOl.xml dan menampilkannya sesuai dengan aturan yang terkait dengan style sheet InventoryOl.css, seperti diperlihatkan berikut ini:



6. Untuk mengetahui bagaimana melakukannya Kita bisa mengubah tampilan dokumen XML dengan mengubah style sheet yang dihubungkan, buka file teks kosong yang baru dalam editor teks Kita dan ketikkan CSS yang telah diubah seperti tampak dalam Listing 10-4.
7. Gunakan perintah Save pada editor teks untuk menyimpan style sheet yang baru pada hard disk dengan nama file Inventory02.css. Style sheet

hasil modifikasi yang telah Kita ketik menyatakan pada Internet Explorer 5 untuk memformat data karakter elemen sebagai berikut:

a. Tampilkan setiap elemen BOOK dengan spasi 12 point di atasnya (margin-top: 12pt) dan sebuah baris pemisah di atas dan di bawah-nya (display: blok) menggunakan huruf 10-point (font-size: lOpt).
b. Tampilkan setiap elemen TITLE, AUTHOR, BINDING, dan PRICE masing-masing pada baris terpisah (display-block).
c. Tampilkan setiap elemen TITLE dalam huruf 12-point (font-size: 12), bercetak tebal (jont-weight. bold), miring (jont-style: italic). (Perhatikan spesifikasi font-size 12-point dibuat untuk elemen TITLE menimpa spesifikasi 10-point yang dibuat untuk induk elemen, BOOK)
d. Inden elemen AUTHOR, BINDING, dan PRICE masing-masing 15- point (margin-left :15pt).
e. Tampilkan elemen AUTHOR dalam cetak tebal (jont-weight: bold).
f. Jangan tampilkan elemen PAGES (display-none).

Inventory02.css

```
/* File Name: Inventory02.css */
BOOK
   {display:block;
    margin-top:12pt;
    font-size:10pt}
TITLE
   {display:block;
    font-size:12pt;
    font-weight:bold;
    font-style:italic}
```

```
AUTHOR
   {display:block;
    margin-left:15pt;
    font-weight:bold}
BINDING
   {display:block;
    margin-left:15pt}
PAGES
   {display:none}
PRICE
   {display:block;
    margin-left:15pt}
```

Listing 10-4.

8. Dalam editor teks, buka dokumen Inventory.xrnl. tambahkan instruksi pemrosesan berikut ke akhir prolog dokumen (sisipkan tepat di atas elemen INVENTORY):

```
<?xml stylesheet
type="text/css"href="Inventory02.css"?>
```

Instruksi pemrosesan ini menghubungkan CSS baru yang Kita buat ke dokumen XML.

9. Untuk mencerminkan nama file baru yang akan Kita berikan, gantilah komponen dekat bagian awal dokumen tersebut dari

```
<! - File Name: Inventory.xml->
```

menjadi

```
<! – File Name: Inventory02.xml->
```

Listing 10-5 memperlihatkan dokumen XML selengkapnya.

10. Gunakan perintah Save pada editor teks Kita untuk menyimpan dokumen hasil modifikas i tersebut dengan nama Inventory02.xrnl. Pastikan Kita menyimpannya dalam folder yang sama dengan Kita menyimpan

Inventory02.css.

```
<?xml version="1.0"?>
<!-- File Name: Inventory02.xml -->
<?xml-stylesheet type="text/css"
href="Inventory02.css"?>
<INVENTORY>
   <BOOK>
      <TITLE>The Adventures of Huckleberry
Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
     <BINDING>mass market paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR>Walt Whitman</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>462</PAGES>
      <PRICE>$7.75</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Legend of Sleepy Hollow
</TITLE>
      <AUTHOR>Washington Irving</AUTHOR>
      <BINDING>mass market paperback</BINDING>
      <PAGES>98</PAGES>
      <PRICE>$2.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Marble Faun</TITLE>
      <AUTHOR>Nathaniel Hawthorne</AUTHOR>
```

```
      <BINDING>trade paperback</BINDING>
      <PAGES>473</PAGES>
      <PRICE>$10.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Moby-Dick</TITLE>
      <AUTHOR>Herman Melville</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>724</PAGES>
      <PRICE>$9.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Portrait of a Lady</TITLE>
      <AUTHOR>Henry James</AUTHOR>
      <BINDING>mass market paperback
</BINDING>
      <PAGES>256</PAGES>
      <PRICE>$4.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Scarlet Letter</TITLE>
      <AUTHOR>Nathaniel Hawthorne</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>253</PAGES>
      <PRICE>$4.25</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Turn of the Screw</TITLE>
      <AUTHOR>Henry James</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>384</PAGES>
      <PRICE>$3.35</PRICE>
   </BOOK>
```

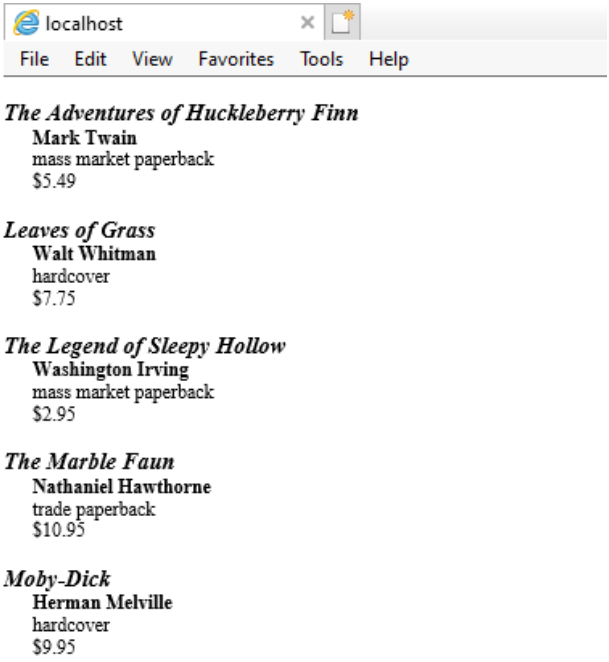
```
</INVENTORY>
```

Listing 10-5.

11. Dalam Windows Explorer atau dalam jendela folder, klik ganda file Inventory02.xml untuk membukanya.

    ```
    Iventory02.xml
    ```

    Internet Explorer 5 akan membuka dokumen Inventory02.xml dan menampilkannya sesuai dengan aturan yang terkait pada style sheet Inventory02.css, seperti tampak di sini :



TIP Bagian memberikan instruksi lengkap untuk menampilkan dokumen XML pada Web. Penulis akan membahas style sheet bertumpuk, seperti yang telah Kita buat di sini.

C. PENUTUP

Ringkasan

Semua komponen prolog yang dimaksudkan dalam bagian ini dijelaskan lebih lanjut dalam bab berikutnya.

Prosesor XML adalah sebuah modul perangkat lunak yang membaca dokumen XML dan memberikan akses ke isi dokumen. la menyediakan akses ini ke modul perangkat lunak lain yang disebut aplikasi, yang memanipulasi dan menampilkan isi dokumen. Saat Kita menampilkan sebuah dokumen XML dalam Internet Exlorer 5, browser memberikan prosesor XML dan sedikitnya sebagian dari aplikasi Jika Kita menulis kode HTML dan script untuk menampilkan sebuah dokumen XML, Kita memberikan bagian dari aplikasi Kita sendiri. Perhatikan istilah aplikasi yang dipakai di sini tidak sama dengan yang dipakai pada aplikasi XML (atau perbendaharaan kata), yang Penulis definisikan sebagai sebuah kumpulan elemen serbaguna dan strutur dokumen yang bisa dipakai untuk menjelaskan dokumen dengan tipe tertentu. Teks dalam dokumen XML terdiri dari markup majemuk dan data karakter. Markup adalah teks dalam kurung yang menjelaskan struktur dokumen tersebut yang dinamai, tag awal elemen, tag akhir elemen, komentar, deklarasi tipe dokumen, instruksi pemrosesan, bagian CDATA, acuan entitas, dan acuan karakter. Semua teks lain nya adalah data karakter isi informasi sesungguhnya dokumen tersebut (dalam dokumen contoh berupa judul, nama pengarang, harga, dan informasi buku lainnya).

Elemen dokumen dalam sebuah dokumen XML serupa dengan elemen BODY dalam sebuah halaman HTML, kecuali Kita bisa memberikannya nama yang sah.

Nama yang muncul pada awal tag-awal dan dalam tag-akhir dikenal sebagai tipe elemen.

Pengurai XML adalah bagian dari prosesor XML yang memeriksa seluruh dokumen XML, menganalisa strukturnya, dan mendeteksi berbagai kesalahan dalam penulisan (lihat untuk definisi dari prosesor XML).

Saat Kita membuka dokumen XML langsung dalam Internet Explorer 5, seperti yang Kita lakukan dalam bab ini, pengurai hanya memeriksa apakah dokumen tersebut sudah well-formed, dan kemudian menampilkan sebuah pesan bila menemukan kesalahan. la tidak memeriksa apakah dokumen tersebut valid.

Pertanyaan

1. Buatlah dokumen XML yang menampilkan nama Anda
2. Sebutkan dan jelaskan Anatomi XML
3. Praktekkan Listing program yang ada di bab ini

DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018

6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 11

Menambahkan Komentar, Instruksi Pemrosesan, dan Bagian CDATA

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 11 akan di bahas Menambahkan Komentar Instruksi Pemrosesan dan bagian CDATA, yang mencakup diantaranya: Menyisipkan Komentar, Bentuk sebuah Komentar, Dimana Anda Bisa Menempatkan Komentar, Menggunakan Instruksi Pemrosesan, Bentuk Instruksi Pemrosesan, Bagaimana Anda Bisa Menggunakan Instuksi Pemrosesan, Dimana Anda Bisa Menempatkan Instruksi Pemrosesan, Memasukkan Bagian CDATA, Bentuk sebuah Bagian CDATA, Dimana Anda Bisa Menenmpatkan Bagian CDATA.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan cara cara Menyisipkan Komentar, Bentuk sebuah Komentar, Dimana Anda Bisa Menempatkan Komentar, Menggunakan Instruksi Pemrosesan, Bentuk Instruksi Pemrosesan, Bagaimana Anda Bisa Menggunakan Instuksi Pemrosesan, Dimana Anda Bisa Menempatkan Instruksi Pemrosesan, Memasukkan Bagian CDATA, Bentuk sebuah Bagian CDATA, Dimana Anda Bisa Menenmpatkan Bagian CDATA.

B. PENYAJIAN

Menambahkan Komentar, Instruksi Pemrosesan, dan Bagian CDATA

Dalam bab ini Anda akan belajar cara menambahkan ketiga tipe markup XML ke dokumen Anda: komentar, instruksi pemrosesan, dan bagian CDATA. Bila ketiga item ini tidak diharuskan dalam dokumen XML yang well-formed (atau valid), mereka bisa bermanfaat. Anda bisa menggunakan komentar untuk menjadikan dokumen Anda lebih mudah dipahami saat dibaca oleh manusia. Anda bisa menggunakan instruksi pemrosesan untuk mengubah cara suatu aplikasi dalam menangani, atau menampilkan dokumen Anda. Anda bisa menggunakan bagian CDATA untuk mema- sukkan hampir semua kombinasi karakter dalam sebuah data karakter elemen.

Menyisipkan Komentar

Seperti yang telah di pelajari, enam sasaran spesifikasi XML adalah "Dokumen XML harus terbaca oleh manusia dan jelas dimengerti." Walaupun prosesor XML umumnya mengabaikan komentar, komentar yang ditempatkan dengan baik dan bermakna bisa menambah keterbacaannya dan kejelasan, dan dokumen XML tersebut ·bisa men- jadikan kode sumber program, seperti C atau BASIC, jadi lebih mudah dipahami.

Bentuk sebuah Komentar

Sebuah komentar diawali dengan karakter <! - dan diakhiri dengan karakter >. Di antara kedua pembatas ini, Anda bisa mengetikkan sembarang karakter yang Anda mau kecuali tanda minus ganda (--). Anda bahkan bisa mengetikkan karakter kurung siku kiri yang seringkali dilarang (<) dan ampersand (&). Berikut ini contoh sebuah komentar yang legal:

```
<! - - Here you can type any text except
adouble hyphen.
```

```
The<and & characters are OK! - - >
```

Di mana Anda Bisa Menempatkan Komentar

Anda bisa menyisipkan sebuah komentar di mana pun dalam sebuah dokumen XML di luar markup lain. Dengan kata lain, Anda bisa menempatkan mereka dalam prolog dokumen:

```
<? xml version="1.O.?>
<! -- Here is a comment in the prolog. - - >
<DOCELEMENT>
This is a very simpl XML document.
</DOCELEMENT>
```

Anda bisa menempatkan mereka dalam isi sebuah elemen:

```
<? xml version="1.O.?>
<DOCELEMENT>
This is a very simple XMLdocument
</DOCELEMENT>

< !- - This comment follows the document
element. - - >
```

Anda bisa menempatkan mereka dalam isi sebuah elemen:

```
< ?xml version="1.O"?>
<DOCLEMENT>
< !- - This comment is part of the content
of the root element. - - > This is a very
simple XMLdocument.
</DOCELEMENT>
```

Berikut ini contoh sebuah komentar yang ilegal karena ditempatkan dalam markup:

```
<?xml version="1.O"?>
<DOCELEMENT
```

```
<!- - This is an ILLEGAL comment ! - - > >
This is a very simple XM document.
</DOCELEMENT>
```

Anda bisa menempatkan sebuah komentar dalam sebuah definisi tipe dokumen (DTD document type definition) walaupun DTD adalah sebuah bentuk markup asalkan itu tidak dalam markup lain yang juga dalam DTD tersebut. Anda akan mempelajari semua tentang DTD dan bagaimana menempatkan sebuah komentar di dalamnya.

Menggunakan lnstruksi Pemrosesan

Tujuan instruksi pemrosesan adalah memberikan informasi bahwa prosesor XML yang disalurkan prosesor XML ke aplikasi.

Bentuk lnstruksi Pemrosesan

Sebuah instruksi pemrosesan memiliki bentuk umum sebagai berikut:

```
<? Target instruction?>
```

Di sini, target adalah nama dari aplikasi untuk mana instruksi diberikan. Sembarang nama diperkenankan, asalkan ia mengikuti aturan-aturan berikut ini:

1. Nama harus diawali dengan sebuah huruf atau garis bawah (_), diikuti dengan atau tanpa sejumlah huruf, · digit, titik (.) hyphen (-) atau beberapa garis bawah.
2. Nama "xml" dalam berbagai kombinasi huruf besar atau huruf kecil, dilarang (seperti yang Anda lihat, Anda menggunakan "xml " dalam huruf kecil untuk deklarasi dokumen XML, yang merupakan tipe instruksi pemrosesan).

Instruksi adalah sebuah informasi yang disalurkan ke aplikasi. la bisa berisi sembarang potongan karakter, kecuali sepasang karakter?> (yang dicadangkan untuk menghentikan instruksi pemrosesan).

Bagaimana Anda Bisa Menggunakan lnstruksi Pemrosesan

Instruksi pemrosesan tertentu yang bisa Anda pakai dalam sebuah dokumen XML bergantung pada prosesor yang akan membaca dokumen tersebut. Jika Anda menggunakan Internet Explorer 5 sebagai prosesor XML, Anda akan mendapatkan dua penggunaan utama instruksi pemrosesan:

1. Anda bisa menggunakan instruksi pemrosesan baku standar untuk memberitahu Internet Explorer 5 cara menangani atau menampilkan dokumen yang ada. Sebuah contoh akan Anda lihat dalam buku ini adalah instruksi pemrosesan yang memberitahu Internet Explorer 5 untuk menampilkan dokumen menggunakan sebuah style sheet khusus. Sebagai contoh, instruksi pemrosesan berikut menyatakan pada Internet Explorer 5 untuk menggunakan CSS yang ditempatkan dalam file InventoryO1.css:

   ```
   <? xml - styl esheet
   type="text/css"href="InventoryOl.css"?>
   ```

2. Jika Anda menulis sebuah script halaman Web untuk menangani dan menampilkan sebuah dokumen XML, Anda bisa menyisipkan sembarang instruksi pemrosesan yang tidak terlarang ke dalam dokumen tersebut, dan script Anda bisa membaca berbagai instruksi ini dan menjalankan aksi yang tepat. Sebagai contoh, Anda bisa menyisipkan instruksi pemrosesan berikut ke dalam sebuah dokumen untuk menyatakan pada script Anda tingkat detil yang ditampilkan:

```
<? MyScript detail="2"?>
```

Di Mana Anda Bisa Menempatkan lnstruksi Pemrosesan

Anda bisa menyisipkan sebuah instruksi pemrosesan di manapun dalam sebuah dokumen XML di luar markup lain yakni, Anda bisa menyisipkannya dalam tempat yang sama dengan tempat yang bisa Anda sisipkan komentar: dalam prolog dokumen, setelah elemen dokumen, atau dalam sebuah isi elemen. Berikut ini sebuah dokumen XML dengan sebuah instruksi pemrosesan pada setiap tempat yang sah:

```
<? xml version="l.O"?>
<! - - The following is a processing instruc
tion in the prolog: - - >
<? xml - stylesheet
type="text/css"href="InventoryOl.css"?>
<INVENTORY>
<BOOK>
<! - - Here's a processing instruction
within an element's content: - - >
<?Script Aemphasize="yes" ?>
<TITLE>The Adventures of Huckleberry Finn
</TITLE>
<AUTHOR>Mark Twain</AUTHOR>
<BINDING>mass market paper back</BINDING>
<PAGES>298</PAGES>
<PRICE>$5.49</PRICE>
<!BOOK>
<BOOK>
<TITLE>Leav s of Grass</TITLE>
<AUTHOR>Walt Whitman</AUTHOR>
<BINDING>hardcover</BINDING>
<PAGES>462</PAGES>
```

```
<PRICE>$7.75</PRICE>
<!BOOK>
<!INVENTORY>
< !- - And here's one following the document
element: - - >
< ?ScriptA Category="books"Style="formal"?>
```

Berikut ini contoh instruksi pemrosesan yang ditempatkan secara ilegal dalam markup:

```
< !- - The following element contains an
ILLEGAL processing instruction: - - >
<BOOK<? ScriptAemphasize="yes"?> >
<TITLE> eaves of Grass</TITLE>
<AUTHOR>Walt Whitman</AUTHOR>
<BINDING>hardcover</BINDING>
<PAGES>462</PAGES>
<PRICE>$7.75</PRICE>
<!BOOK>
```

Anda bisa menempatkan sebuah instruksi pemrosesan dalam sebuah DTD walaupun DTD adalah sebuah bentuk markup asalkan ia tidak dalam markup yang lain yang juga dalam DTD tersebut. Anda akan mempela jari semua tentang DTD dan cara menempatkan instruksi pemrosesan di dalamnya.

Memasukkan Bagian CDATA

Seperti yang telah Anda pelajari dalam Bab sebelumnya, Anda tidak bisa langsung menyisipkan karakter kurung siku kiri (<) ataupun ampersand (&) dalam data karakter pada isi elemen. Cara mengatasi keterbatasan ini adalah menggunakan sebuah acuan karakter (< atau &) atau sebuah acuan entitas umum yang telah didefinisikan (< atau &amp ;). Jika Anda ingin

menyisipkan sejumlah karakter < atau &, penggunaan acuan ini tidak menyenangkan dan menjadikan data sulit dibaca manusia. Dalam hal ini, lebih mudah untuk menempatkan teks yang berisi karakter terlarang dalam bagian CDATA.

Bentuk sebuah Bagian CDATA

Bagian CDATA diawali dengan karakter <! [CDATA [dan diakhiri dengan karakter]] >. Di antara kedua kelompok karakter pembatas, Anda bisa mengetikkan sembarang karakter (termasuk karakter < atau & kecuali]] > (tentu saja akan ditafsirkan sebagai akhir dari bagian CDATA). Semua karakter dalam bagian CDATA dianggap sebagai bagian literal dari data karakter elemen, bukan sebagai markup XML.

Berikut ini contoh sebuah bagian CDATA yang legal:

```
< ! [CDATA [
Here you can type any characters except two
right brackets followed by agreater - than
symbol.]] >
```

Jika Anda ingin memasukkan sebuah blok kode sumber, atau markup sebagai bagian dari sebuah data karakter sesungguhnya dari elemen yang akan ditampilkan dalam browser, Anda bisa menggunakan bagian CDATA untuk mencegah pengurai XML menafsirkan karakter < atau & sebagai markup XML. Berikut ini adalah contohnya:

```
<A-SECTION>
Teh following is an example of a very simple
HTML page :
<![CDATA[
<HTML>
```

```
        <HEAD>
             <TITLE> R. Jones & Sons</TITLE>
        </HEAD>
        <BODY>
             <P>Welcome to Our home page!</P>
        </BODY>
</HTML>
]]>
</A-SECTION>
```

Tanpa bagian CDATA, prosesor akan menganggap bahwa <HTML>, misalnya, sebagai bagian elemen tersarang ketimbang sebagai bagian dari data karakter elemen A-SECTION.

Di Mana Anda Bisa Menempatkan Bagian CDATA

Anda bisa menyisipkan bagian CDATA di mana pun yang terdapat data karakter-yakni, dalam sebuah isi elemen namun tidak dalam markup XML. Berikut ini bagian CDATA yang ditempatkan secara legal:

```
<MUSICAL>
<TITLE_PAGE>
     <![CDATA[
          <Oklahoma!>
               By
          Rogers & Hammerstein
     ]]>
</TITLE_PAGE>
<!-- Other elements here.......-->
</MUSICAL>
```

Dokumen XML yang buruk diperlihatkan pada halaman berikutnya berisi dua bagian CDATA yang ilegal. Yang pertama tidak dalam isi sebuah elemen,

yang kedua dalam isi sebuah elemen dokumen namun juga dalam markup tag-awal.

```
<? xml version="l.O"?>
<! [CDATA[ILLEGAL:not within element
content! JJ >
<DOCELEMENT>
<SUB_ELEMENT<![CDATA[ILLEGAL:inside of
markup! JJ>> sub - element content ...
</SUB_ELEMENT>
</DOCELEMENT>
```

C. PENUTUP

Ringkasan

Dalam Microsoft Internet Explorer 5, prosesor XML juga tidak menguraikan teks komentar untuk markup XML ataupun memprosesnya. la menjadikan teks sebuah komentar yang tersedia untuk sebuah script yang ditulis dalam suatu halaman Web HTML. Dalam Bab ini, Anda akan belajar cara menggunakan script untuk mengakses teks komentar seperti halnya komponen lain dari sebuah dokumen XML. Internet Explorer 5 juga menampilkan semua komentar dalam sebuah dokumen XML, jika Anda membuka dokumen tersebut secara langsung, dan bila dokumen tersebut tidak mempunyai style sheet yang disertakan padanya.

Lihat kembali, di mana prosesor XML adalah sebuah modul perangkat lunak yang membaca dan menyimpan isi sebuah dokumen XML. Aplikasi adalah sebuah modul perangkat lunak tersendiri yang mengambil isi dokumen dari prosesor, dan kemudian memanipulasi serta menampilkan isinya. Ketika Anda menampilkan XML dalam Internet Exlorer 5, browser menyediakan prosesor

XML dan paling sedikit sebagian dari aplikasi. Kata-kunci CDATA, seperti halnya kata-kunci XML lain yang akan Anda lihat berikutnya, harus ditulis dalam huruf besar semua.

Karena Anda bisa langsung menyisipkan karakter < dan & dalam bagian CDATA, Anda tidak perlu menggunakan acuan karakter (< atau &), atau acuan entitas umum yang telah didefinisikan &It; atau &. Pada kenyataannya, jika Anda menggunakan acuan semacam itu, pengurai akan menafsirkan setiap karakter dalam acuan secara literal dan tidak akan mengganti acuan dengan karakter < atau &.

Bagian CDATA tidak bersarang. Yakni, Anda tidak bisa menyisipkan satu bagian CDATA dalam bagian CDATA lainnya.

Pertanyaan

1. Pada bagian mana kita bisa menempatkan komentar?
2. Bagaimana cara menggunakan instruksi pemrosesan?
3. Pada bagian mana kita bisa menempatkan instruksi pemrosesan?
4. Pada bagian mana kita bisa menempatkan bagian CDATA?

DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016

5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 12

Menampilkan Dokumen XML Menggunakan Style Sheet Bertingkat

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 12 akan di bahas Cara Menampilkan Dokumen XML Menggunakan Style Sheet Bertingkat, yang mencakup diantaranya: Langkah Dasar Menggunakan Style Sheet Bertingkat, Masalah Insensitivitas dalam CSS, Pewarisan Setting Properti, Menggunakan Multi Elemen dan Multi Aturan, Menggunakan Selektor Kontekstual, MenggunakanAtribut Style, Mengimport Style Sheet Lain, Menentukan Nilai URL, Pengaliran dalam Style Sheet Bertingkat, Menyeting Properti Display, Menantukan Nilai Kata Kunci CSS, Menyeting Properti Font, Membuat Dokumen.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan cara Langkah Dasar Menggunakan Style Sheet Bertingkat, MAsalah Insensitivitas dalam CSS, Pewarisan Setting Properti, Menggunakan Multi Elemen dan Multi Aturan, Menggunakan Selektor Kontekstual, MenggunakanAtribut Style, Mengimport Style Sheet Lain, Menentukan Nilai URL, Pengaliran dalam Style Sheet Bertingkat, Menyeting Properti Display, Menantukan Nilai Kata Kunci CSS, Menyeting Properti Font, Membuat Dokumen.

B. PENYAJIAN

Menampilkan Dokumen XML Menggunakan Style Sheet Bertingkat

Dalam bab ini, Anda pertama akan mempelajari metode yang dicakup dalam buku ini untuk menampilkan dokumen XML pada browser Microsoft Internet Explorer 5; style sheet bertingkat (CSS-cascading style sheet). Style sheet bertumpuk /bertingkat adalah sebuah file yang berisi instruksi pemformatan elemen dalam dokumen XML.

Karena Anda membayangkan elemen Anda sendiri dalam XML, browser tidak punya cara sendiri bagaimana menampilkannya dengan tepat. Membuat dan mengaitkan style sheet bertingkat ke dokumen XML merupakan satu cara untuk memberitahu browser bagaimana menampilkan setiap elemen dokumen tersebut. Sebuah dokumen XML dengan style sheet bertingkat yang disertakan padanya bisa dibuka langsung dalam Internet Explorer 5. Anda tidak perlu menggunakan sebuah halaman HTML untuk mengakses dan menampilkan data tersebut.

Menjadikan instruksi tampilan dalam style sheet tersendiri dari dokumen XML sesungguhnya meningkatkan keluwesan dokumen XML, dan menjadikannya lebih mudah dikelola. Anda bisa, misalnya, dengan cepat menyesuaikan sebuah dokumen XML tunggal untuk beragam situasi tampilan yang berbeda (browser, aplikasi, konteks, dan device yang berbeda, dan sebagainya) cukup dengan menyertakan sebuah style sheet yang sesuai, tanpa harus membentuk ulang dokumen itu sendiri. Juga, Anda bisa dengan cepat memperbaharui format sekelompok dokumen XML yang sama cukup dengan merevisi style sheet bersamanya yang disertakan pada dokumen tersebut, tanpa harus membuat dan mengedit masing-masing dokumen.

Menggunakan style sheet bertingkat mungkin adalah metode termudah untuk menampilkan sebuah dokumen XML. Untuk satu hal, bahasa CSS telah akrab untuk para desainer halaman Web karena penggunaannya yang sekarang dengan halaman HTML. Juga, browser Web sekarang telah menyediakan dukungan tingkat tinggi untuk style sheet bertingkat, sementara metode lain untuk menampilkan XML masih dikembangkan dan browser baru mulai mendukungnya.

Bagaimanapun juga, dibandingkan dengan metode penampilan XML yang akan Anda pelajari nanti, style sheet bertingkat agak terbatas. Walaupun style sheet bertingkat memberikan kontrol yang cukup tinggi untuk cara browser memformat isi elemen dalam dokumen XML, ia tidak memungkinkan Anda mengubah isi elemen dalam dokumen XML. Ia juga tidak memungkinkan Anda mengakses atribut XML, entitas, instruksi pemrosesan, dan komponen lainnya-tidak juga untuk memproses informasi yang dimuat komponen ini.

Anda akan belajar cara mengaitkan sebuah dokumen XML ke sebuah halaman HTML dan menampilkan elemen XML dengan mengikat elemen HTML standar padanya. Anda akan belajar cara mengakse dan menampilkan elemen individual, atribut, dan komponen lain pada sebuah dokumen XML dengan cara menulis kode script dalam sebuah halaman HTML. Anda akan belajar cara menggunakan bahasa style sheet yang lebih mampu Extensible Stylesheet Language (XSL) yang memungkinkan Anda tidak hanya memformat isi elemen XML namun juga untuk mengirimkan isi dokumen dalam cara yang betul-betul luwes.

Langkah Dasar Menggunakan Style Sheet Bertingkat

Ada dua langkah dasar dalam menggunakan style sheet bertingkat untuk menampilkan sebuah dokumen XML:

1.Buat file style sheet.

2.Kaitkan style sheet ke dokumen XML

Langkah Pertama: Membuat File Style Sheet

Style sheet bertingkat adalah sebuah file teks biasa, umumnya dengan ekstensi .css, yang berisi sekumpulan aturan yang menyatakan pada browser bagaimana memformat dan menampilkan elemen dalam dokumen XML yang spesifik. Seperti dengan dokumen XML, Anda bisa membuat style sheet menggunakan editor teks kesukaan Anda.

Listing 12-1 berisi sebuah contoh style sheet bertingkat yang sederhana dengan nama file InventoryOl.css.)

InventoryOl.css

```
/* File Name: Inventory01.css */
BOOK
   {display:block;
    margin-top:12pt;
    font-size:15pt}
TITLE
   {font-style:italic}
AUTHOR
   {font-weight:bold}
```

Listing 12-1.

Style sheet ini disertakan pada dokumen XML yang tampak dalam Listing 12-

2. dengan nama file InventoryOl.css.) Anda bisa menempatkan catatan kecil pada halaman dengan Listing 12-2 tampak dalam contoh lain pada bab ini, dan Anda bisa melihatnya kembali.

InventoryOl.xml

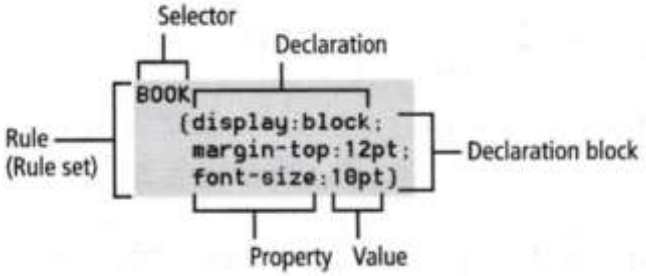
```
<?xml version="1.0"?>
<!-- File Name: Inventory01.xml -->
<?xml-stylesheet type="text/css"
href="Inventory01.css"?>
<INVENTORY>
   <BOOK>
      <TITLE>The Adventures of Huckleberry
Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR>Walt Whitman</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>462</PAGES>
      <PRICE>$7.75</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Legend of Sleepy
Hollow</TITLE>
      <AUTHOR>Washington Irving</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
```

```
      <PAGES>98</PAGES>
      <PRICE>$2.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Marble Faun</TITLE>
      <AUTHOR>Nathaniel Hawthorne</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>473</PAGES>
      <PRICE>$10.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Moby-Dick</TITLE>
      <AUTHOR>Herman Melville</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>724</PAGES>
      <PRICE>$9.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Portrait of a Lady</TITLE>
      <AUTHOR>Henry James</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
      <PAGES>256</PAGES>
      <PRICE>$4.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Scarlet Letter</TITLE>
      <AUTHOR>Nathaniel Hawthorne</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>253</PAGES>
      <PRICE>$4.25</PRICE>
   </BOOK>
   <BOOK>
```

```
        <TITLE>The Turn of the Screw</TITLE>
        <AUTHOR>Henry James</AUTHOR>
        <BINDING>trade paperback</BINDING>
        <PAGES>384</PAGES>
        <PRICE>$3.35</PRICE>
    </BOOK>
</INVENTORY>
```

Listing 12-2.

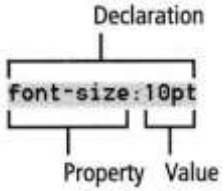
Sebuah style sheet berisi satu atau lebih aturan (kadang disebut dengan seperangkat aturan). Aturan ini memuat informasi tampilan u ntuk tipe elemen tertentu dalam dokumen XML. Contoh style sheet tersebut memuat tiga aturan: satu untuk elemen BOOK, satu untuk elemen TITLE, dan satu untuk elemen AUTHOR. Inilah aturan untuk elemen BOOK, dengan semua bagian yang dinamai:



selector adalah nama tipe elemen untuk mana informasi tampilan diterapkan. Mengikuti selector adalah declaration block, yang dibatasi dengan karakter kurung berkerut ({}) dan berisi satu atau lebih declarations yang dipisahkan dengan titik-koma (;).

Masing-masing deklarasi menyebutkan seting property tertentu, seperti misalnya ukuran font yang dipakai untuk menampilkan elemen tersebut.

Deklarasi terdiri dari properti, diikuti dengan tanda bagi (:), diikuti dengan value untuk properti itu. Misalnya, deklarasi berikut mengatur properti font-size ke nilai 10pt (10 point).



Style sheet bisa juga berisi comments (komentar). Sebuah komentar style sheet dimulai dengan karakter slash dan asteris (/*) dan diakhiri dengan karakter asteris dan slash (*/). Di antara pasangan karakter pembatas ini, Anda bisa mengetikkan sembarang teks yang diinginkan, dan ketika browser membaca style sheet tersebut untuk memformat dokumen, ia akan mengabaikan teks itu. Anda bisa menggunakan komentar untuk membantu dokumen atau menjelaskan style sheet agar bisa dimengerti orang. Sebagai con- toh adalah komentar pada awal contoh style sheet di Listing 7-1:

```
I * File Name: InventoryOl.css* I
```

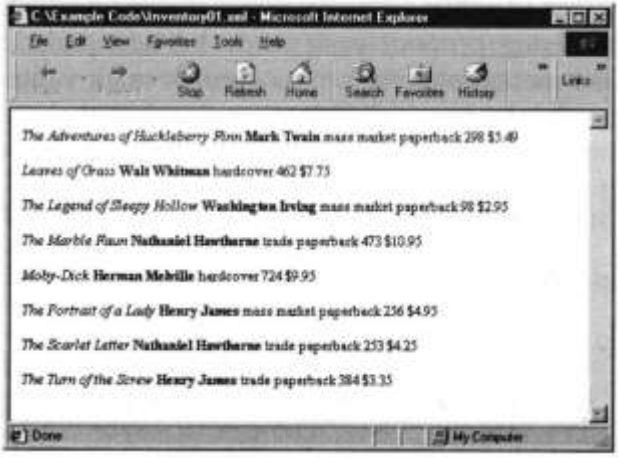
Anda juga bisa menggunakan komentar saat membangun sebuah style sheet untuk sementara waktu mematikan aturan atau sebagian aturan. Misalnya, jika Anda ingin melihat bagaimana elemen BOOK akan terlihat tanpa margin atas, Anda bisa sementara waktu menambahkan karakter komentar seperti dalam aturan BOOK berikut:

```
BOOK
{display: block;
/ * margin-top: l2pt; */
font - size: lO pt}
```

Contoh style sheet dalam Listing 12-1 memasukkan deklarasi berikut:

1. display: block. Menyisipk an sebuah baris pemisah sebelum dan setelah teks elemen (seting ini memiliki akibat lain yang akan Anda pelajari).
2. margin-top: l 2pt. Menambahkan margin selebar 12-point di atas teks elemen.
3. font-size: l Opt. Menyeting ukuran font yang dipakai untuk menampilkan teks elemen menjadi 10 point.
4. font-style: italic. Menampilkan teks elemen dalam karakter miring.
5. font-weight: bold. Menampilkan teks elemen dalam karakter tebal.

Baginilah caranya Internet Explorer 5 menampilk an dokumen XML yang menggunakan style sheet ini, mengikuti instruksi yang diberikan oleh deklarasi ini.



Sekumpulan properti yang tersedia dalam style sheet bettingkat mirip dengan yang bisa Anda terapkan pada teks dalam pengolah kata. Nanti dalam bab ini, Anda akan mempela jari tentang properti yang berbeda yang bisa Anda pakai dan nilai spesifik yang bisa Anda berikan padanya.

Masalah lnsensitivitas dalam CSS

Dengan Internet Explorer 5, style sheet bertingk at bersifat case-insensitive. Yaitu, bila Internet Explorer 5 memproses sebuah style sheet, ia mengabaikan besar kecilnya huruf. Sebagai contoh, Anda bisa mengetikkan aturan berikut dalam tiga cara:

```
TITLE
{font- style: italic}
Title
{FONT-STYLE: Italic}
title
{Font-Style: ITALIC}
```

Masalah insensitivitas dalam style sheet bertingkat mempunyai implikasi penting. Karena dokumen XML bersifat case-sensitive, Anda secara normal bisa memiliki dua tipe elemen berbeda yang namanya berbeda dalam besar-kecil hurufnya, seperti Book dan BOOK. Dalam style sheet bertingkat kedua nama ini akan dianggap sebagai tipe elemen yang sama, dan Anda tidak akan bisa memberikan mereka seting properti yang berbeda. Sehingga, jika Anda bermaksud menamp ilkan dokumen XML yang menggunakan style sheet bertingkat, Anda tidak boleh memiliki tipe elemen yang namanya hanya berbeda dalam besar-kecilnya dalam satu atau beberapa huruf.

Pewarisan Seting Properti

Secara umum, seting properti yang Anda berikan pada elemen tertentu (seperti BOOK) mempengaruhi semua elemen anak yang tersarang langsung ataupun tidak langsung di dalamnya, kecuali jika ia ditimpa dengan seting yang berbeda yang dibuat untuk elemen anak tertentu.

Properti berikut adalah perkecualian dan tidak diwa risi oleh elemen anak:

1. Properti display, dibahas dalam “Menyeting Property display’ dalam bab ini.
2. Properti latar (background color, background-image, background repeat, dan background-position), dijelaskan dalam "Menyeting Properti Latar" dalam bab ini.
3. Properti vertical-align, dijelaskan dalam “Menyeting Spasi Teks dan Properti Perataan" dalam bab ini.
4. Properti kotak, dibahas dalam "Menyeting Properti-Properti Box" dalam bab ini

Sebagai contoh, style sheet dalam Listing 12-1 memformat elemen BOOK (dalam dokumen pada Listing 12-2) seperti ini:

```
BOOK
{display: block;
margin-top: l2pt; font-size: lOpt }
```

Setiap elemen BOOK memiliki lima elemen anak. Karena font-size adalah sebuah properti yang diwariskan, semua elemen anak dalam elemen BOOK ditampilkan dalam font 10-point. Elemen anak tidak mewarisi seting properti display dan margin-top (margin-top adalah salah satu dari properti kotak).

Dengan sebuah properti yang tidak diwariskan, jika Anda tidak menyebutkan nilai properti untuk elemen tertentu, browser akan menggunakan nilai default properti itu. Misalnya, nilai default untuk properti display adalah inline. Bab ini memberikan nilai default bagi semua properti yang tidak diwariskan.

Lantaran sejumlah nilai properti diwariskan, bila Anda merancang sebuah style sheet, Anda bisa memulai dengan elemen tingkat atas, dan kemudian turun hingga elemen yang tersarang di bawahnya. Pada elemen-elemen ini Anda hanya akan membutuhkan beberapa penataan dan menambahkan sebuah

seting penimpa untuk waktu tertentu. Pendekatan ini akan mengurangi penyetingan properti yang tak perlu (dinamakan demikian di mana elemen anak mewarisi dan karenanya Anda tidak perlu menyebutkannya).

Anda akan mempela jari lebih lanjut tentang pewarisan dan bagaimana ia dimasukkan ke dalam seluruh mekanisme aliran pada bagian "Pengaliran dalam Style Sheet Bertingkat " dalam bab ini.

Menggunakan Multi-Elemen dan Multi-Aturan

Anda bisa menerapkan sebuah aturan tunggal pada sejumlah elemen dengan memasukkan semua nama elemen dalam selektor, dan memisahkan nama tersebut dengan koma. Sebagai contoh, aturan berikut diterapkan pada tipe elemen POEM, TITLE, AUTHOR, DATE dan STANZA:

```
POEM.TITLE. AUTHOR.DATE. STANZA
{display: block;
margin- bottom: l2pt}
```

Jika sebuah kelompok elemen bersama-sama memakai sekumpulan seting properti, Anda akan membuat style sheet Anda lebih pendek, dan mudah dimengerti dengan cara memasukkan semua elemen ini dalam sebuah aturan tunggal ketimbang harus menggandakan sering rersebut dalam aturan tersendiri.

Anda juga bisa memasukkan ripe elemen yang diberikan pada lebih dari satu aturan dalam style sheet yang sama. Misalnya, aturan berikur memasukkan elemen DATE:

```
POEM, TITLE.AUTHOR. DATE, STANZA
{display: block;
margin-bottom: l2pt}
DATE
```

```
{font-style: italic}
```

Aruran pertama berisi deklarasi bahwa DATE dipakai bersama dengan elemen lain yang tercantum, semenrara aturan kedua memperbaiki DATE yakni, ia menyebutkan sering properti yang diterapk an pada DATE sendiri.

Menggunakan Selektor Kontekstual

Dalam sebuah selektor, Anda bisa memperkenalkan nama elemen dengan nama-nama dari saru atau lebih elemen pendahulu (induk, induk dari induk dari induk, dan sererusnya), dan aruran tersebur hanya akan diterapkan pada elemen dengan nama itu yang tersarang. Selektor yang memasukkan saru arau lebih nama elemen pendahulu disebur selekror kontekstual. Selekror yang ridak memasukkan nama elemen pendahulu (seperti yang Anda lihat pada bagian sebelumnya) disebut selektor generik.

Jika sebuah properti khusus memiliki sebuah sering dalam suaru aruran dengan selektor konreksrual, dan memiliki sering lain dalam aruran dengan selekror generik untuk elemen yang sama, sering dalam aturan kontekstual akan didahulukan karena lebih spesifik.

Anggaplah, misalnya, yang berikut ini adalah elemen root dari sebuah dokumen XML:

```
<MAPS>
<CITY>
<NAME>Santa Fe</NAME>
<STATE>New Mexico</STATE>
<! CITY>
<STATE>California</STATE>
<! MAPS>
```

Aturan berikut dalam sebuah style sheet yang disertakan akan menyebabkan browser memformat "New Mexico " dalam font normal, namun "California" dalam bentuk miring:

```
CITY STATE
{font-style: normal)
STATE
{font-style: italic}
```

Walaupun elemen STATE New Mexico sesuai dengan selektor kontekstual pada aturan CITY STATE dan selektor generik dalam aturan STATE, selektor dalam aturan CITY STATE lebih spesifik dan karenanya didahulukan. (Anda nanti akan mempelajari lebih jauh tentang prioritas pada aturan yang berlawanan di "Pengaliran dalam Style Sheet Bertingkat " dalam bab ini.)

Menggunakan Atribut STYLE

Anda bisa menggunakan atribut STYLE dalam dokumen XML Anda ketimbang dalam style sheet yang ada-untuk memberikan satu atau lebih properti yang lebih spesifik pada sebuah elemen individual. Jika sebuah seting properti diberikan melalui sebuah atribut STYLE konflik dengan seting properti yang dibuat dalam style sheet yang disertakan, seting STYLE akan didahulukan. Dengan demikian, atribut STYLE adalah cara terbaik untuk menimpa-untuk elemen tertentu seting-seting properti umum yang dibuat untuk tipe elemen pada style sheet yang disertakan. Bagaimanapun juga, penggunaan STYLE tidak melanggar prinsip CSS dalam menjaga format informasi terpisah dari definisi isi dan struktur dokumen dalam file XML.

Untuk menyebutkan satu atau lebih seting properti, masukkan deklarasi tersebut dalam nilai atribut STYLE dalam tanda petik, pisahkan deklarasi

individual dengan titik-koma, seperti yang Anda lakukan dalam blok deklarasi CSS.

Sebagai contoh, style sheet dalam Listing 12-1 memformat elemen STYLE dalam font italic, 10-point. Jika Anda memasukkan atribut STYLE berikut dalam tag-awal untuk elemen TITLE spesifik pada dokumen, elemen itu sendiri akan ditampilkan dalam font roman (nonitalic) 14-point:

```
<TITLESTYLE'font-style: normal; font-size:
l4pt'>The Adventures of Huckleberry Finn
<! TITLE>
```

Beginilah caranya dokumen akan terlihat dalam Internet Explorer 5.

TIP Untuk dokumen XML yang valid, Anda perlu mendeklarasikan atribut STYLE dalam OTO sebelum Anda bisa memakai atribut tersebut. Anda harus mendeklarasikan nya seperti berikut:

```
<! ATTLIST TITLE STYLE CDATA #IMPLIED>
```

Mengimpor Style Sheet Lain

Anda bisa menggunakan perintah @import dalam style sheet bertingkat untuk memasukkan satu atau lebih style sheet. Opsi-opsi untuk mengimpor style sheet tersendiri memungkinkan Anda menyimpan aturan gaya dalam file tersendiri, dan kemudian mengkombinasikannya bila diperlukan untuk tipe dokumen tertentu.

Inilah bentuk umum perintah @import, di mana StyleSheetURL adalah URL (*Uniform Resource Locator*) lengkap atau relatif dari file yang berisi style sheet bertingkat yang ingin Anda impor:

```
@import url C styleSheetURL);
```

Untuk keterangan mengenai penyebutan nilai URL, lihat “Menentukan Nilai URL" pada halaman berikut. Sebagai contoh, perintah berikut (yang memakai URL relatif) ditempatkan pada awal style sheet dalam Listing 7-1 akan mengimpor style sheet yang terkandung dalam file Book.css (yang harus berada dalam folder yang sama dengan style sheet yang mengimpor):

```
I* File Name: InventoryOl.css*/
@import url C Book.css); BOOK
{display: block;
margin-top: l2pt; font-size: lOpt}

/* rest of style sheet. . . * I
```

Perintah @import harus berada pada awal style sheet, sebelum peraturan lainnya. Anda bisa memasukkan sejumlah perintah pada @import awal style sheet.

Ketika Anda mengimpor satu atau beberapa style sheet, browser menggabungkan aturan yang dimuat dalam style sheet yang diimpor dan yang mengimpor. Jika aturan yang konflik, sheet yang mengimpor akan

didahulukan dari style sheet yang diimpor. Dan jika Anda mengimpor beberapa style sheet, aturan dalam style sheet yang diimpor belakangan dalam file tersebut didahulukan dari yang ada dalam style sheet yang diimpor sebelumnya. Untuk lebih jelasnya mengenai prioritas, lihat "Pengaliran dalam Style Sheet Bertingkat" dalam bab ini.

Menentukan Nilai URL

URL merupakan alamat Internet standar, seperti misalnya http://msgress.microsoft.com/. Prihtah @import dan properti background image keduanya membutuhkan sebuah nilai URL yang menandakan lokasi somber daya yang bersangkutan (style sheet atau file citra). Anda menentukan URL menggunakan bentuk berikut, di mana URL dalam URL. Perhatikan Anda tidak bisa memasukan spasi di antara url dan pembuka karaktet (

```
url (URL)
```

Anda bisa menggunakan URL lengkap, seperti dalam contoh berikut ini:

```
@import
url(http://www.my_domain.com/stylesheets/My
Styles.css); INVENTORY
fbacg round-image: ur1 (file: IIE:\Example
Code\Background.gif)}
```

Atau, Anda bisa menggunakan sebagian URL yang menyebutkan lokasi yang menunjuk pada lokasi file style sheet berisi URL tersebut. URL relatif dalam style sheet bekerja persis seperti URL relatif dalam hala.man HTML. Misalnya; jika flle style sheet ditempa. tkan dalam folder Example Code, URL relatif berikut akan setara dengan URL lengkap dalam contoh terdahulu (dengan nama, file: //E: \Example Code\Background. gif):

```
INVENTORY (background-image; url
(Background.gif)}
```

Langkah Kedua: Mengaitkan Style Sheet ke Dokumen XML

Untuk mengaitkan style sheet bertingkat ke sebuah dokumen XML, Anda sisipkan instruksi pemrosesan xml-sty lesheet yang telah disediakan ke dalam dokumen tersebut. lnstruksi pemrosesan ini memiliki bentuk umum sebagai berikut, di mana CSSFilepath adalah URL yang menunjukkan lokasi file style sheet:

```
<? xml – style sheet
type="text/css"href=CSSFilePath?>
```

Anda bisa menggunakan URL utuh, seperti ini:

```
<? xml - stylesheettype="text/css"
href="http://www.my_domain.com/InventoryOl.c
ss"?>
```

Yang lebih umum Anda bisa menggunakan bagian URL yang menyebutkan sebuah lokasi yang menunjukkan lokasi dokumen XML yang berisi instruksi pemrosesan xml-stylesheet, seperti ini:

```
<? xml -
stylesheettype="text/css"href="InventoryOl.
css"?>
```

(URL lebih umum karena Anda biasanya menyimpan sebuah file style sheet dalam folder di mana Anda menyimpan dokumen XML, atau dalam salah satu subfoldernya.)

Biasanya Anda menambahkan instruksi pemrosesan xml-stylesheet ke prolog dokumen XML, mengikuti deklarasi XML, seperti Anda lihat dalam contoh dokumen XML pada Listing 12-2. (Untuk lebih jelasnya mengenai instruksi pemrosesan, dan penjelasan semua tempat di mana Anda bisa

menyisipkannya secara legal, lihat "Menggunakan Instruksi Pemrosesan".)

Kemampuan untuk menyertakan sebuah style sheet eksternal ke dokumen XML menjadikan pemformatan dokumen cukup dengan menyertakan sebuah style sheet yang berbeda. Untuk menyertakan sebuah style sheet yang berbeda, Anda hanya perlu mengedit URL dalam instruksi pemrosesan xml stylesheet tanpa melakukan perubahan apa pun dalam dokumen XML.

Ketika Anda mengaitkan sebuah style sheet ke sebuah dokumen XML, Anda bisa membuka dokumen itu secara langsung dalam Internet Explorer 5 misalnya, Anda bisa memasukkan URL dokumen atau lokasi file ke dalam Address Bar IE5 dan tekan Enter:

Ketik URL atau path file dokumen XML di sini dan tekan Enter.

Atau, dengan anggapan bahwa Internet Explorer 5 adalah browser default, Anda bisa cukup dengan mengklik-ganda nama file dokumen XML dalam Windows Explorer atau dalam jendela folder:

```
lnventory01.xml
```

Internet Explorer 5 akan membuka dokumen XML dan menampilkannya menggunakan instruksi dalam style sheet yang dikaitkan.

Anda bisa memasukkan lebih dari satu style sheet dalam sebuah dokumen XML dengan cara menyisipkan instruksi pemrosesan xml-stylesheet untuk masing-masingnya, seperti dalam contoh dari awal dokumen XML ini:

```
<? xml version="l.0"?>
```

```
<? xml - stylesheet
type="text/css"href="BookOl.css"?>
<? xml - stylesheet
type="text/css"href="Book02.css"?>

< INVENTORY>
<! - - contents of document element - - >
<! INVENTORY>
```

Opsi pengaitan sejumlah style sheet memungkinkan Anda menyimpan kelompok aturan yang berhubungan dalam file tersendiri, dan kemudian mengkombinasikanya jika diperlukan untuk tipe dokumen tertentu.

Bila Anda mengaitkan lebih dari satu style sheet, Internet Explorer 5 menggabungkan aturan sheet yang berbeda tersebut. Jika style sheet terpisah berisi aturan yang berkonflik, aturan dalam style sheet tersebut dikaitkan belakangan dalam dokumen yang lebih diutamakan daripada style sheet yang dikaitkan sebelumnya pada dokumen tersebut. Anda akan mempela jari lebih lanjut tentang prioritas antaraturan yang mengalami konflik pada bagian berikutnya.

Pengaliran dalam Style Sheet Bertingkat

Kata "cascading " dalam Cascading Style Sheet berarti bahwa Anda bisa memberikan nilai ke berbagai properti dalam sejumlah tingkatan yang berbeda (persis seperti aliran air terjun yang mengalir melalui sejumlah tingkatan). Daftar berikut menjelaskan tingkatan utama di mana Anda bisa memberikan sebuah nilai ke properti. Saya cantumkan tingkat-tingkat tersebut dalam susunan prioritas mereka dari yang tertinggi ke terendah. Ketika browser akan menampilkan sebuah elemen, jika properti yang diberikan seperti font-size

berisi nilai-nilai yang berkonflik untuk elemen itu pada tingkatan yang berbeda, browser akan menggunakan seting yang diberikan pada tingkat prioritas tertinggi.

1. Jika Anda memberikan sebuah nilai ke atribut STYLE dari elemen tertentu dalam dokumen XML, browser akan menggunakan nilai itu untuk menampilkan elemen tersebut. Misalnya, ia akan menampilkan elemen berikut dalam cetak tebal:

```
<TITLESTYLE-"font-weight: bold">Leaves of
Grass</TITLE>
```

2. Jika Anda tidak menyeting properti melalui atribut STYLE, browser akan memakai sebuah nilai properti yang dideklarasikan dalam aturan CSS dengan selektor kontekstual (yakni, selektor yang menyebutkan sebuah elemen bersama dengan satu atau lebih elemen pendahulu, seperti telah dibahas sebelumnya dalam "Menggunakan Selektor Kontekstual"). Anggaplah yang berikut ini adalah elemen dokumen dari sebuah dokumen XML:

```
<MAPS>
<CITY>
< NAME>Santa Fe</NAME>
<STATE>New Mexico</STATE>
<! CITY>
<STATE>California</STATE>
</MAPS>
```

Anggaplah juga bahwa style sheet yang disertakan berisi aturan berikut:

```
CITY STATE
{font-style: normall
STATE
{font-style: italic}
```

Browser akan menggunakan aturan CITY STATE untuk memformat

elemen STATE "New Mexico," karena ia memiliki sebuah elektor kontekstual, dan karenanya didahulukan dari aturan STATE, yang hanya memi- liki selektor generik. "New Mexico " akan muncul dalam font normal.

3. Jika Anda tidak mendeklarasikan nilai properti tertentu dalam sebuah aturan yang memiliki selektor generik yang bersangkutan, browser akan memakai nilai yang dideklarasikan dalam aturan dengan selektor generik (yakni, selektor yang hanya memasukkan nama elemen). Sebagai contoh, dalam contoh style sheet yang diberikan dalam item 2, browser tidak akan menemukan aturan kontekstual yang sesuai bagi elemen STATE "California", sehingga ia akan menggunakan aturan STATE generik, dan karenanya akan menampilkan "California" dalam font italic.
4. Jika Anda tidak mendeklarasikan suatu nilai dari properti tertentu dalam aturan generik untuk elemen tersebut, browser akan menggunakan seting properti yang dideklarasikan untuk elemen pendahulu yang terdekat (induk, induk dari induk, dan seterusnya). Sebagai contoh, dalam style sheet pada Listing 12-1, aturan untuk elemen TITLE tidak memberikan suatu nilai ke properti f ant-size:

```
TITLE
(font-style: italic}
```

Sehingga, browser akan memakai seting font-size dari induk elemen dari elemen ini, BOOK (BOOK adalah induk dari TITLE dalam dokumen XML yang menggunakan style sheet tersebut):

```
BOOK
{display: block;
margin-top: l2pt; font-size: lOpt}
```

Karenanya ia akan menampilkan teks elemen TITLE menggunakan

karakter 10-point.

Perhatikan proses ini akan terjadi hanya untuk properti yang diwariskan. Untuk properti yang tidak diwariskan, browser akan meng- gunakan nilai default properti (lihat "Pewarisan Seting Properti" sebelumnya dalam bab ini).

5. Jika style sheet tidak memasukkan sebuah seting properti untuk suatu elemen pendahulu, browser akan menggunakan setingnya sendiri. Seting ini bisa menjadi nilai default yang dimasukkan dalam browser atau yang diseting oleh. pengguna browser. Misalnya, karena contoh style sheet dalam Listing 12-1 tidak mengatur properti font family bagi suatu elemen, maka browser akan memakai nilai font family-nya sendiri untuk menampilkan semua elemen (dalam Internet Explorer 5, ini adalah Times New Roman kecuali jika pengguna browser memilih keluarga font yang berbeda melalui perintah Internet Options pada menu Tools).

Sekali lagi, proses ini hanya diterapkan pada properti yang diwariskan. Untuk properti yang tidak diwariskan, browser menggunakan nilai default propertinya.

Seperti yang bisa Anda lihat dari daftar ini, prinsip umumnya adalah: jika Anda memberikan sebuah properti yang nilainya mengalami konflik pada tingkat yang berbeda, browser akan memberikan preferensi untuk aturan yang lebih spesifik. Sebagai contoh, seting properti untuk elemen itu sendiri lebih spesifik dibanding seting untuk induk elemen, dan karenanya akan didahulukan. Anda bisa menggunakan prinsip ini untuk diterapkan pada kasus yang lebih rumit (misalnya, jika induk elemen anak memiliki aturan kontekstual dan aturan generik, aturan mana yang akan dipakai untuk elemen

anak? Anda benar: browser akan menggunakan aturan kontekstual!).

Apa yang terjadi jika properti tertentu yang diberikan mengalami konflik seting pada sejumlah tingkatan? Dalam hal ini, browser menggunakan seting terakhir yang diprosesnya. Sebagai contoh, jika dua aturan generik untuk elemen yang sama mengalami konflik seting untuk properti font- style, seperti dalam contoh berikut, browser akan menggunakan yang kedua karena ia memprosesnya terakhir:

```
TITLE.AUTHOR, BINDING, PRICE
{display: block; font-size: l2pt; font-
weight: bold; font-style: italic}
AUTHOR
{font-style: normall
```

Dengan demikian, dalam contoh ini, browser akan memformat elemen AUTHOR menggunakan font normal ketimbang italic.

Point-point berikut menjelaskan susunan di mana browser memproses aturan style sheet:

- Jika Anda mengaitkan seju mlah style sheet ke dokumen menggunakan instruksi pemrosesan xml-styles heet, browser akan memproses style sheet tersebut dalam urutan yang Anda cantumkan pada instruksi pemrosesan.
- Jika Anda mengimpor satu atau beberapa style sheet ke dalam style sheet yang sedang Anda pakai perintah @import (seperti diterangkan dalam bagian sebelumnya "Mengimpor Style Sheet Lain"), browser akan memproses mereka dalam urutan Anda mengimpor mereka.
- Dalam style sheet tertentu, aturan diproses dalam urutan mereka dicantumkan.

Menyeting Properti display

Properti display mengontrol cara dasar browser menampilkan sebuah teks elemen. Anda bisa memberikan salah satu dari tiga kata-kunci CSS berikut:

1. Block. Browser selalu menyisipkan sebuah baris pemisah sebelum dan sesudah teks elemen (yang memasukkan teks yang dimiliki oleh suatu elemen anak). Sebagai hasilnya, teks elemen ditampilkan dalam "blok" terpisah dengan teks pendahuluan di atasnya, dan teks selanjutnya di bawahnya. Memberikan nilai block juga memungkinkan Anda memformat teks dengan menerapkan beragam properti box ke blok teks, seperti halnya margin, border yang tampak, dan padding. Elemen block dengan demikian serupa dengan sebuah paragraf pada program pengolah kata, yang memisahkan teks pendahulu dengan teks selanjutnya menggunakan pemisah baris, dan biasanya bisa berupa margin, border yang diberikan, dan lain sebagainya.
2. Inline (default). Browser tidak menyisipkan baris pemisah sebelum atau sesudah teks elemen (kecuali jika teks pengantar atau teks elemen telah mencapai sisi kanan jendela, dan browser perlu melipat teks ke baris di bawahnya). Browser akan menyisipkan baris pemisah dalam teks elemen hanya jika dibutuhkan untuk menjadikan teks sesuai dalam jendela. Teks elemen dengan demikian bisa berada pada baris yang sama dengan teks pengantar atau berikutnya. Elemen inline serupa dengan sekuen karakter dalam sebuah paragraf pada program pengolah kata.
3. None. Browser tidak menampilkan elemen tersebut. Anda bisa menggunakan seting ini untuk elemen-elemen yang menyimpan informasi yang ingin Anda sembunyikan pada layar.

Untuk keterangan lebih lanjut tentang pemberian kata-kunci CSS pada properti, lihat "Menentukan Nilai Kata-Kunci CSS".

Anggaplah, misalnya, Anda menggunakan style sheet berikut untuk menampilkan contoh dokumen XML dalam Listing 12-2 (ingatlah bahwa untuk mengganti style sheet yang dipakai untuk menampilkan sebuah dokumen XML, Anda perlu mengedit instruksi pemrosesan xml-stylesheet dalam dokumen tersebut):

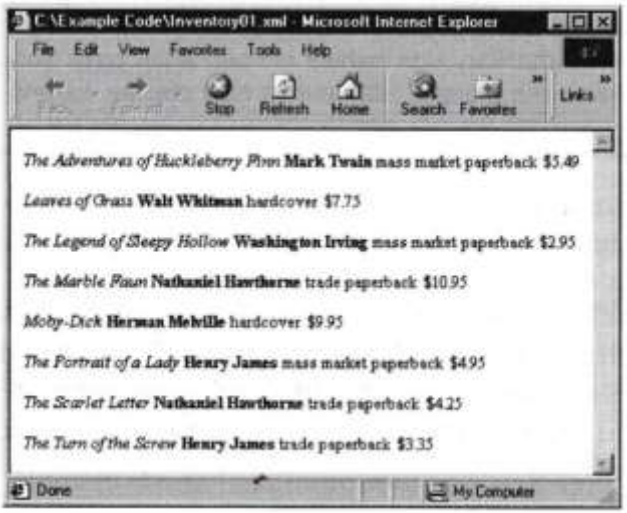
```
BOOK
{display: block;
margin-top: l2pt; font- size: lOpt
TITLE
(font-style: italicl
AUTHOR
(font-weight: bold}
PAGES
{display: nonel
```

Karena properti display elemen BOOK diberikan nilai block, browser selalu menyisipkan sebuah baris pemisah sebelum dan sesudah teks elemen (BOOK memiliki isi elemen. Teksnya berisi teks yang dimiliki oleh semua elemen anaknya).

Karena style sheet tersebut tidak memberikan nilai-nilai properti display untuk elemen TITLE, AUTHOR, BINDING, dan PRICE (dan elemen ini tidak mewarisi nilai display dari elemen induk mereka), browser menganggap mereka sebagai elemen inline, yang merupakan nilai default. Karena itu, browser tidak menyisipkan baris pemisah di antara elemen ini, dan sepanjang jendela browser cukup lebar ia akan menampilkan semuanya pada baris yang sama.

Karena properti display elemen PAGES diberikan nilai none, maka browser tidak menampilkan elemen itu.

Inilah hasil seluruhnya:



Menentukan Nilai Kata-Kunci CSS

Dengan sejumlah properti CSS, Anda bisa-atau harus-memberikan sebuah nilai menggunakan kata-kunci CSS yang telah ada. Kata-kunci spesifik yang bisa Anda pakai bergantung pada properti tertentu. Misalnya, Anda bisa memberikan properti display salah satu dari tiga kata-kunci: block, inline, atau none. Anda bisa memberikan pfoperti color salah satu dari 16 kata-kunci yang menggambarkan warna warna dasar, seperti halnya red, green, yellow, atau fuchsia, seperti dalam contoh ini:

```
PARA {color: fuchsia}
```

Anda bisa memberikan properti border-style salah satu dari Sembilan kata kunciyang memungkinkan: solid, dotted, dashed, double, groove, ridge, inset,

outset, atau none seperti tampak di sini:

```
SECTION {border-style: solid
```

Menyeting Properti Font

Standar CSS menyediakan properti berikut untuk mengubah font yang dipakai dalam menampilkan teks elemen:

1. Font Family
2. font-size
3. font-style
4. font-weight font- variant

Semua properti ini diwarisi oleh elemen anak.

Menyeting Properti font-family

Properti font-family menyebutkan nama font yang dipakai untuk menampilkan teks elemen, seperti ditunjukkan dalam contoh ini:

```
BOOK (font-familiy: Arial}
```

Anda bisa memasukkan sembarang nama font yang diinginkan (semua itu tidak ada dalam kata-kunci CSS yang disediakan). Jika browser tidak bisa menemukan font yang diminta, ia akan menggantinya dengan font yang tersedia.

TIP Jika sebuah nama font yang berisi spasi, kelilingi seluruhnya dengan tanda petik, seperti dalam contoh ini: BOOK {font-familiy:"Times New Roman "}.

Anda bisa menambah kesempatan Anda untuk menampilkan tipe font yang

diinginkan dengan mencantumkan pilihan alternatif, dipisahkan dengan koma, dalam urutan yang diinginkan. Inilah contohnya:

```
BOOK {font-family: Arial.Helvetica}
```

Jika font yang dinamai Arial tidak ada, browser akan menggunakan Helvetica. Jika Helvetica tidak ada, ia akan menggantinya dengan font apa pun yang tersedia.

Selanjutnya Anda bisa menambah kesempatan Anda untuk menampilkan font yang diinginkan dengan memasukkan sebuah kata-kunci CSS-biasanya pada akhir daftar-yang menandakan tipe umum dari font yang Anda inginkan, seperti dalam contoh ini:

```
BOOK {font-family: Arial.Helvetica,
sansserif)
```

Di sini, jika browser tidak bisa menemukan Arial atau Helvetica, ia akan menggantinya dengan beberapa font sans serif (yakni, font tanpa serifs, juga dikenal sebagai font gothic).

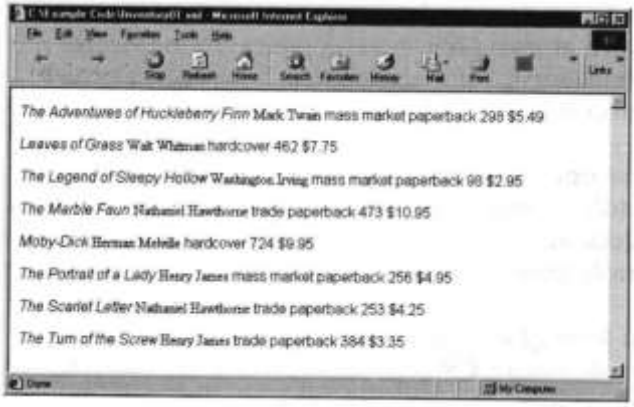
Tabel berikut mencantumkan berbagai kata-kunci yang bisa Anda pakai untuk menandakan tipe umum font yang Anda inginkan. Spesifikasi CSS memanggil nama-nama keluarga generik ini. Untuk setiap nama keluarga generik, tabel juga memberikan nama dari font spesifik yang dimiliki oleh keluarga itu. Seperti pada contoh yang ditampilkan Internet Explorer 5 ketika Anda meminta keluarga itu (font tertentu yang ditampilkan Internet Explorer 5 bergantung pada kumpulan font yang saat itu diinstal dalam Microsoft Windows, sehingga font-font tersebut bisa jadi Anda lihat berbeda).

Kata-kunci nama keluarga generik fontfamily	Contoh dari font spesifik	Contoh teks
serif	Times New Roman	The Adventures of Huckleberry Finn

sans-serif	Arial	THE ADYUTURJKS *OY* HUCKLIKBURYfINN
cursive	Zapf Chancery	The Adventures""of Huckleberry Finn
fantasy	Western	*%e 5'1.aventures of J-[ucRJebemt :Finn*
monospace	Courier New	The Adventures of Huckleberry Finn

Sebagai contoh, jika Anda menyertakan style sheet berikut ke contoh dokumen XML dalam Listing 12-2, Internet Explorer 5 akan menampilkan dokumen tersebut seperti berikut:

```
BOOK
{display: block;
margin-top: l2pt;
font-family: Arial, sans-serif; fontsize:
l2pt
TITLE
{font-style: italic}
AUTHOR
{font-family:"Times NewRoman", serif)
```



Font Arial diberikan pada pro perti font-jamily elemen BOOK diwarisi oleh semua elemen anak kecuali AUTHOR, yang mempunyai nilai font-family penimpanya sendiri (" Times New Roman ", serif>.

Menyeting Properti font-size

Properti font-size mengatur tinggi font yang dipakai untuk menampilkan teks elemen. Anda bisa memberikan properti ini empat macam tipe nilai:

1. Sebuah nilai yang menyangkut ukuran font pada browser. Anda bisa menyebutkan ukuran font yang menyangkut ukuran font yang ada pada browser dengan memberikan properti font-size salah satu nilai kata-kunci dalam tabel berikut. Dengan Internet Explorer 5, nilai small menyebabkan browser memakai ukuran font terpilihnya yang ada saat itu; nilailain diskalakan menurun atau naik dari sini.

Kata-kunci font-size :	xx-small
Contoh aturan CSS:	TITLE {font-size:xx-small}
Penjelasan:	Ukuran font terkecil yang bisa diatuur dengan kata-kunci
Contoh teks:	The Adventures of Huckleberry Finn
Kata-kunci font-size:	*x-sma ll*
Contoh aturan CSS:	TITLE{font-size:x-sma11}
Penjelasan:	Sekitar 1,5 kali dari xx-small
Contoh teks:	The Adventures of Huckleberry Finn

Kata-kunci font-size:	*small*
Contoh aturan CSS:	TITLE {font-size: small}
Penjelasan:	Sekitar 1,5 kali dari xx-small
Contoh teks:	The Adventures of Huckleberry Finn
Kata-kunci font-size:	*medium*
Contoh aturan CSS:	TITLE {font-size: medium)
Penjelasan:	Sekitar 1,5 kali dari small
Contoh teks:	The Adventures of Huckleberry Finn
Kata-kunci font-size:	*large*
Contoh aturan CSS:	TITLE {font-size: large}
Penjelasan:	Sekitar 1,5 kali dari medium
Contoh teks:	The Adventures of Huckleberry Finn
Kata-kunci font-size:	*x-large*
Contoh aturan CSS:	TITLE (font-size: x-large}
Penjelasan:	Sekitar 1,5 kali dari large
Contoh teks:	The Adventures of Huckleberry Finn
Kata-kunci font-size:	*xx-large*
Contoh aturan CSS:	TITLE {font-size: x-largel}

Penjelasan:	Sekitar 1,5 kali dari xx-large
Contoh teks:	The Adveores of Huckleberry Finn

2. Persentasi ukuran font induk. Ketimbang menggunakan kata-kunci smaller atau larger, Anda bisa menyebutkan ukuran font yang menyatakan uku ran font pada elemen induk saat itu dengan presisi yang lebih besar dengan cara memberikan nilai persentasi ke properti font- size. Sebagai contoh, aturan berikut menginginkan ukuran font yang satu dan satu setengah kali ukuran font induk.

```
TITLE (font-size: l50%}
```

(Jika browser menggunakan rasio penskalaan yang direkomendasikan pada 1,5, aturan ini akan setara dengan aturan TITLE (jont-size: larger}.)

```
TITLE {font-size: l60%}
```

Perhatikan untuk elemen root, persentasinya didasarkan pada ukuran font browser. Untuk keterangan lebih lanjut mengenai pemberian nilai persentasi, lihat "Menentukan Nilai Persentasi" pada halaman berikut).

3. Nilai ulfuran spesifik. Anda jug a bisa menyebutkan ukuran font elemen dengan memberikan nilai ukuran pada font -size. (Saya menjelaskan tipe yang berbeda dari nilai tipe ini dalam "Menentukan Nilai Ukuran " dalam bab ini.) Sebagai contoh, aturan berikut menyebutkan ukuran font 12-point:

```
TITLE {font-size: l2pt}
```

Aturan berikutnya menyebutkan sebuah font yang dua kali lebih besar dari font elemen induk:

```
TITLE {font - size:2em}
```

(Contoh kedua sama dengan 1111E (jont-size:200%}.)

Menyeting Properti color

Properti color mengatur warna dari teks elemen. Anda bisa memberikan properti ini nilai warna yang dipakai pada format yang telah dibahas dalam "Menentukan Nilai Warna" pada halaman berikut. Sebagai contoh, aturan berikut menyeting warna pada teks elemen AUTHOR menjadi biru cerah:

```
AUTHOR {color: blue}
```

Aturan berikut ini menyeting teks elemen AUTHOR menjadi merah cerah:

```
AUTHOR (color: rgb (255,0,0))
```

Properti color diwarisi oleh elemen. Dengan demikian, jika Anda menyertakan style sheet berikut ke contoh dokumen XML dalam Listing 12-2, semua teks akan menjadi biru cerah kecuali teks PRICE, yang akan menjadi merah cerah karena style sheet memasukkan sebuah seting warna penimpa untuk elemen ini:

```
BOOK
{display: block;
margin-top: l2pt; font-size: lOpt; color:
blue}
TITLE
{font-style: italic}

AUTHOR"'
{font-weight: bold}

PRICE
{color: red}
```

TIP Properti color mengatur warna pada huruf individual dalam teks (kadang disebut warna latardepan teks (text foreground color). Untuk

mengatur warna latar belakang teks, gunakan properti background-color yang telah dibahas dalam "Menyeting Properti background-color" dalam bab ini.

Menentukan Nilai Warna

Properti-properti untuk mana Anda memberikan nilai-nilai warna termasuk color, background-color, dan border-color. Anda bisa memberikan sebuah nilai warna menggunakan salah satu dari empat format yang berbeda, yang diilustrasikan oleh contoh aturan berikut. Aturan ini adalah sama-masing-masing memberikan warna merah cerah ke properti color.

```
PARA {color: red}
PARA (color: rgb (255,0, Cl)} PARA {color:
#FFOOOO}
PARA (color: rgb (l00%,0%,0%))
```

Format pertama memakai kata-kunci CSS (red), sementara yang lainnya memberikan sebuah warna dengan menyebutkan intensitas relatif dari komponen warna red, green, blue, dalam urutan itu. Pada format kedua, setiap intensitas warna disebutkan melalui nilai desimal yang berkisar dari 0 hingga 255. Pada format ketiga, warna disebutkan menggunakan angka heksadesimal enam digit yang berkisar dari 000000 hingga FFFFFF, di mana dua digit pertama meriyatakan intensitas merah, dua digit kedua intensitas hijau, dan dua digit terakhir menyatakan intensitas biru. Dalam format terakhir, setiap intensitas warna disebutkan melalui nilai persentasi yang berkisar dari 0% hingga l00%.

Tabel berikut mencantumkan nilai warna yang bisa Anda berikan

menggunakan kata kunci CSS, dan untuk setiap warna memperlihatkan spesifikasi warna yang setara dalam empat format (katakunci CSS menggunakan nama grafis untuk warna, seffientara kolom pertama dalam tabel ini menggunakan nama warna standar yang dipakai dalam fotografi dan optik.

Jika Anda menggunakan salah satu format RGB, Anda tentu saja bisa membuat lebih banyak lagi warna selain yang diperlihatkan dalam tabel ini. Pada kenyataannya, karena Anda bisa memberikan salah satu dari tiga komponen warna 256 nilai yang berbeda, Anda bisa menyebutkan seluruhnya 16.777.216 warna yang berbeda (256.256.256). Jika Anda menampilkan pas sebuah sistem dengan kedalaman warna 24-bit atau lebih, monitor bisa menampilkan warna sesungguhnya.

Menyeting Properti Background

Standar CSS menyediakan properti berikut yang memungkinkan Anda mengubah latar elemen:

1. background-color
2. background-image
3. background-repeat
4. background-position

Background atau latar belakang adalah area yang mengitari karakter individual dari teks elemen. Anda bisa memberikan pula warna solid atau sebuah citra pada sesuatu latar elemen.

Secara teknis, elemen-elemen anak tidak mewarisi apa pun dari properti properti ini. Bagaimanapun juga, secara default, background elemen

adalah transparan. Artinya bahwa jika Anda membuang semua properti background dari elemen anak, warna atau citra background elemen induk (atau browser) terus ditampilkan, dengan efektif memberikan elemen anak background yang sama dengan induknya (atau browser).

Menyeting Properti background-color

Anda bisa menerapkan sebuah warna background solid ke sebuah elemen dengan memberikan sebuah nilai warna pada properti background colornya. Sebagai contoh, aturan berikut mengatur warna background pada elemen TITLE menjadi kuning cerah:

```
TITLE (background-color: yellow)
```

Ingatlah bahwa properti color mengatur warna foreground (latar-depan) elemen yakni, warna karakter itu sendiri. Dengan demikian, aturan berikut membuat huruf biru dengan latar kuning.

```
TITLE
[color: blue;
background-color: yellow}
```

Jika Anda tidak ingin menyebutkan warna background yang solid untuk suatu elemen, Anda bisa memberikan properti background-color dengan nilai transparent, seperti diperlihatkan di sini:

```
TITLE (background-color: transparentl
```

Atau, karena transparent adalah nilai default, Anda bisa saja menghilangkan properti background-color dari elemen itu. Kecuali jika Anda memberikan sebuah citra background ke elemen tersebut, seting

transparent menyebabkan background elemen induk tersebut ditampilkan.

Menyeting Properti background-image

Anda bisa menambahkan sebuah citra background ke sebuah elemen dengan memberikan URL dari file citra tersebut ke properti background image. (Untuk informasi mengenai pemberian URL, lihat "Menentukan Nilai URL" sebelumnya.) Sebagai contoh, aturan berikut memberikan citra background yang dimuat dalam file Leaf.bmp ke elemen STANZA:

```
STANZA{background-image:url(Leaf.bmp)}
```

Untuk mengilustrasikan lebih lanjut, perhatikan style sheet yang ditampilkan dalam Listing 12-3, yang disertakan ke dokumen XML seperti tampak dalam Listing 12-4.

Leaves.css

```
/* File Name: Leaves.css */
POEM
   {font-size:145%}
POEM, TITLE, SUBTITLE, AUTHOR, SECTION,
NUMBER, STANZA, VERSE
   {display:block}
SECTION, STANZA
   {margin-top:1em}
STANZA
   {background-image:url(Leaf.bmp)}
```

Listing 12-3.

Leaves.xml

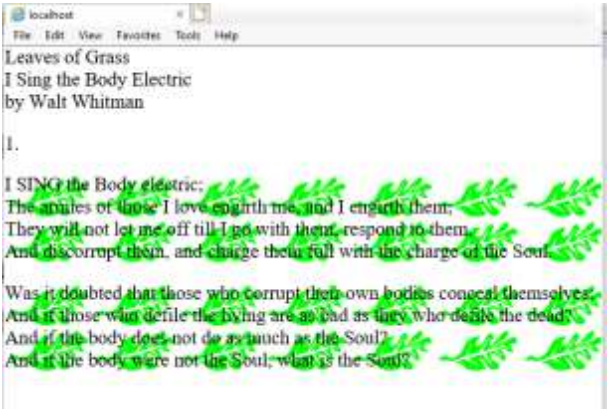
```
<?xml version="1.0"?>
<!-- File Name: Leaves.xml -->
<?xml-stylesheet type="text/css"
href="Leaves.css"?>
```

```
<POEM>
<TITLE>Leaves of Grass
   <SUBTITLE>I Sing the Body
Electric</SUBTITLE>
</TITLE>
<AUTHOR>by Walt Whitman</AUTHOR>
<SECTION>
<NUMBER>1.</NUMBER>
<STANZA>
   <VERSE>I SING the Body electric;</VERSE>
   <VERSE>The armies of those I love engirth
me, and I engirth them;</VERSE>
   <VERSE>They will not let me off till I go
with them, respond to them,</VERSE>
   <VERSE>And discorrupt them, and charge
them full with the charge of the
Soul.</VERSE>
</STANZA>
<STANZA>
   <VERSE>Was it doubted that those who
corrupt their own bodies conceal
themselves;</VERSE>
   <VERSE>And if those who defile the living
are as bad as they who defile the
dead?</VERSE>
   <VERSE>And if the body does not do as
much as the Soul?</VERSE>
   <VERSE>And if the body were not the Soul,
what is the Soul?</VERSE>
</STANZA>
</SECTION>
</POEM>
```

Listing 12-4.

Inilah isi dari file grafis Leaf.bmp:

Internet Explorer 5 akan menampilkan Leaves.xml seperti ini:



Perhatikan citra tersebut diulangi (berupa tile) untuk mengisi seluruh area yang ditempati oleh isi elemen, menyebar ke sisi kanan jendela browser (bagian berikutnya menjelaskan bagaimana mengontrol tiling). Perhatikan juga ada bagian citra yang menempati bagian atas atau bawah teks elemen yang dipotong (yakni, yang dibatasi). Dalam contoh tersebut, hanya sebagian kecil citra pada baris bawah masing-masing elemen STANZA yang dipotong.

Jika Anda tidak ingin menyebutkan suatu citra background untuk sebuah elemen, Anda bisa memberikan properti background-image dengan nilai none, seperti ini:

```
STANZA {background - image: none)
```

Atau, karena none adalah nilai default, Anda bisa mengabaikan dengan

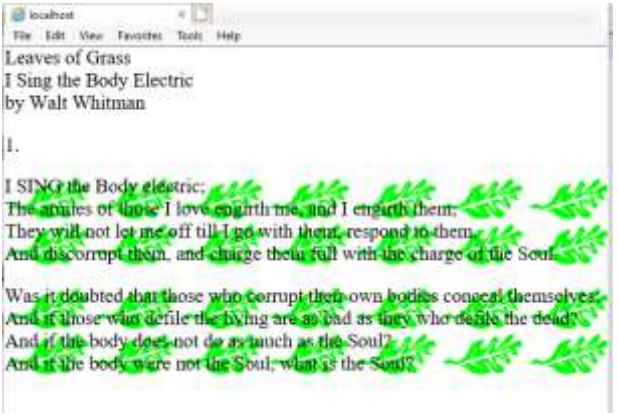
membuang properti background image elemen itu. Kecuali jika Anda memberikan warna background solid ke elemen tersebut, seting none menyebabkan background induk (atau browser) ditampilkan seluruhnya.

Menyeting Properti background repeat

Jika Anda telah memberikan sebuah file citra ke propetti background image, Anda bisa mengontrol cara pengulangan citra tersebut dengan memberikan properti background repeat salah satu dari nilai kata-kunci berikut:

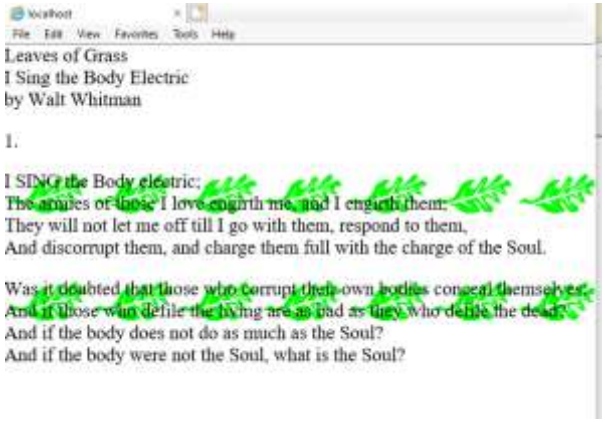
1. repeat (default). Mengulang citra tersebut baik horizontal maupun vertikal. Karena ini merupakan nilai default, penambahan background repeat. repeat ke aturan STANZA dalam style sheet pada Listing 7-3, seperti terlihat di sini, tidak akan berpengaruh apa-apa pada tampilan dokumen, seperti terlihat berikut ini:

```
STANZA
{background - image:url(Leaf.bmp);
background - repeat: repeat)
```



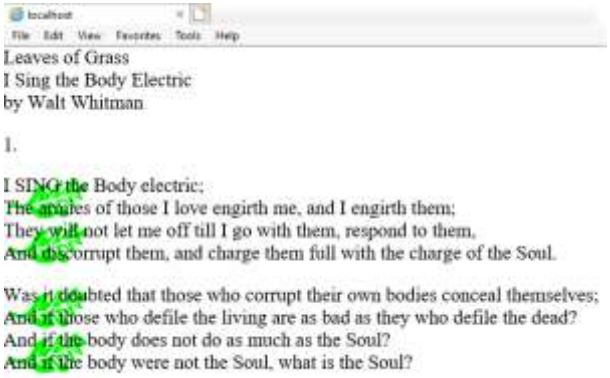
2. repeat-x. Mengulang citra dalam arah horizontal saja. Sebagai contoh, aturan STANZA berikut akan menampilkan dokumen seperti tampak dalam gambar berikut:

```
STANZA
[background - image:url(Leaf.bmp);
background-repeat: repeat-x}
```



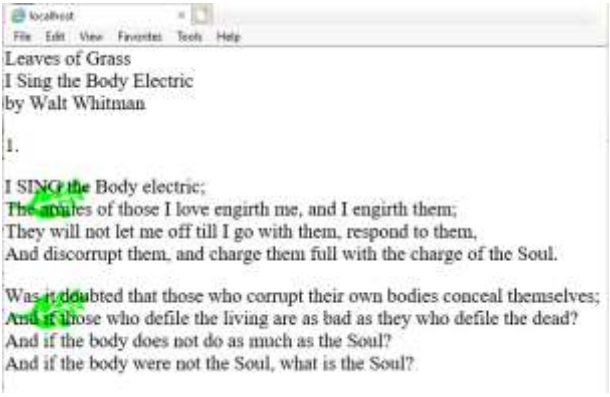
3. repeat-y. Mengulang citra dalam arah vertikal saja. Sebagai contoh, aturan STANZA berikut akan menampilkan dokumen seperti tampak dalam gambar berikut:

```
STANZA
[background - image:url(Leaf.bmp);
background - repeat: repeat-y}
```



4. no-repeat. Menyebabkan citra ditampilkan hanya sekali. Sebagai contoh, aturan STANZA berikut akan menampilkan dokumen seperti tampak dalam gambar berikut:

```
STANZA
[background - image:url(Leaf.bmp);
background - repeat:no-repeat}
```



Menyeting Properti background position

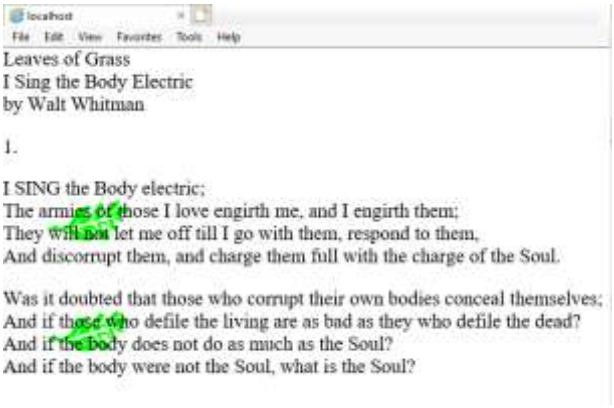
Secara default, sudut kiri atas citra latar (atau sudut kiri atas dari kiri atas salinan citra tersebut jika ia diulang) diratakan dengan sudut kiri atas

elemen. Anda bisa mengganti perataan ini memberikan sebuah nilai ke pro- perti background-position. Anda bisa memberikan properti ini salah satu dari tiga tipe nilai:

1. Nilai ukuran horizontal dan vertikal. Anda bisa memberikan properti background position dua nilai ukuran. Nilai pertama menandakan posisi horizontal citra tersebut dalam elemen, dan yang kedua menandakan posisi vertikal citra tersebut dalam elemen. Anda bisa memberikan sembarang tipe nilai ukuran seperti digambarkan dalam "Menentukan Nilai Ukuran" sebelumnya. Sebagai contoh, aturan berikut menempatkan sudut kiri atas dari citra 0,5 ini ke kanan dan 0,25 inci ke bawah dari sudut kiri atas dari elemen STANZA:

```
STANZA
{background - image:url(Leaf.bmp);
background - repeat:no - repeat: background
- position: 0.5in 0.25in}
```

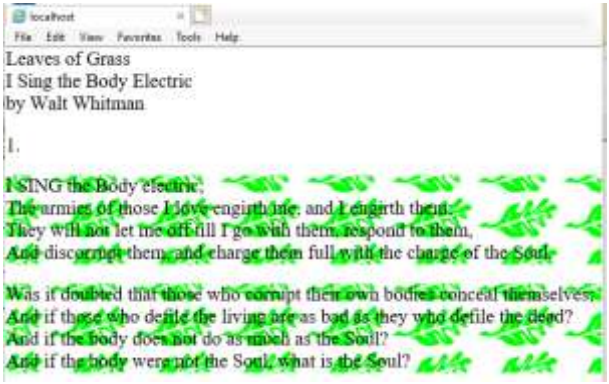
Inilah hasil dari aturan ini:



Jika sebuah citra diulang seperti disebutkan dalam aturan berikut, seluruh pola citra yang diulang tersebut ditempatkan berdasar jumlah yang

disebutkan, seperti tampak dalam. gambar ini:

```
STA NZA
{background - image:url(Leaf.bmp};
background - repeat: repeat;
background - position: 0.5in 0.25in}
```



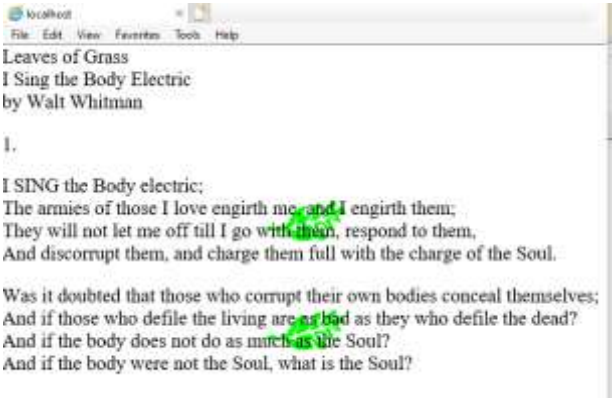
2. Nilai-nilai persentasi horizontal dan vertikal. Anda bisa memberikan properti background position dua nilai persentasi. Yang pertama menandakan posisi horizontal citra tersebut dalam elemen, di mana 0% menempatkannya pada bagian kiri (dalam posisi horizontal defaultnya), 50% menempatkannya pada horizontal tengah elemen, dan 100% menempatkannya pada bagian kanan elemen. Persentasi kedua menunjukkan posisi vertikal citra, di mana 0% menempatkannya pada bagian atas (dalam posisi vertikal defaultnya), 50% menempatkannya pada vertikal tengah dari elemen, dan 100% menempatkannya pada bagian bawah elemen.

   Sebagai contoh, aturan berikut ini akan menempatk an citra tersebut di tengah elemen:

   ```
   STANZA
   ```

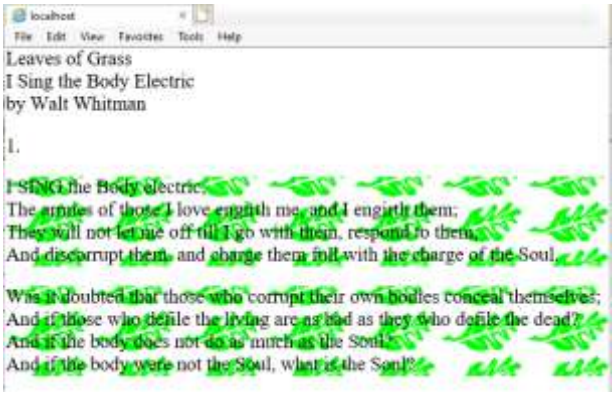
```
{background - image:url(Leaf.bmp);
background - repeat:no-repeat;
background - position:50% 50%}
```

Inilah hasil aturan tersebut:



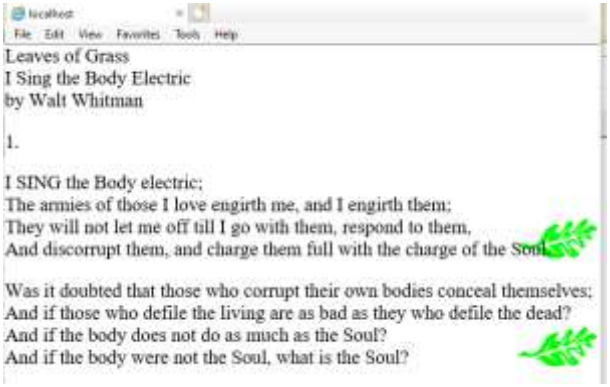
Jika citra diulang seperti disebutkan dalam aturan berikut, seluruh pola citra yang diulang ditempatkan berdasar jumlah yang disebutkan, seperti tampak dalam gambar ini:

```
STANZA
{background - image:url(Leaf.bmp);
background - repeat: repeat;
background - position:50% 50%)
```



3. Nilai Kata-kunci. Anda bisa menyebutkan posisi citra background dengan memberikan properti background position satu, atau dua kata kunci CSS. Misalnya, memasukkan kata-kunci right dan bottom, seperti dalam aturan berikut, menempatkan citra tersebut pada bagian kanan bawah elemen, seperti tampak dalam gambar ini:

```
STANZA
(background - image: url (Leaf.bmp);
background - repeat: no - repeat;
background - position: right bottom}
```



Gambar berikut memperlihatkan posisi citra yang dihasilkan dari masing-masing kombinasi kata-kunci.

Susunan kata-kunci tidaklah penting. Misalnya, background position: bottom right sama dengan background position: right bottom.

Menyeting Properti Text Spacing dan Alignment

Standar CSS menyediakan properti berikut yang mengubah spasi, perataan, dan fitur lain pada teks:

1. letter-spacing
2. vertical-align
3. text-align
4. text-indent
5. line-height
6. text-transform
7. text-decoration

Elemen anak mewarisi semua properti ini kecuali vertical-align.

Menyeting Properti letter-spacing

Anda bisa menggunakan properti letter-spacing untuk menambah atau mengurangi spasi antarkarakter pada teks elemen. Anda bisa memberikan letter-spacing sebuah nilai ukuran positif untuk menambah spasi karakter dengan jumlah yang disebutkan. Sebagai contoh, aturan berikut menambah spasi karakter dengan jumlah yang sama dengan seperempat tinggi teks:

```
TITLE (letter - spacing:.25em)
```

Anda bisa memberikan letter-spacing nilai ukuran negatif untuk mengurangi spasi karakter dengan jumlah yang disebutkan. Sebagai contoh, aturan berikut menambah spasi karakter dengan jumlah yang sama dengan setengah tinggi teks:

```
TITLE (letter - spacing: - .5pt}
```

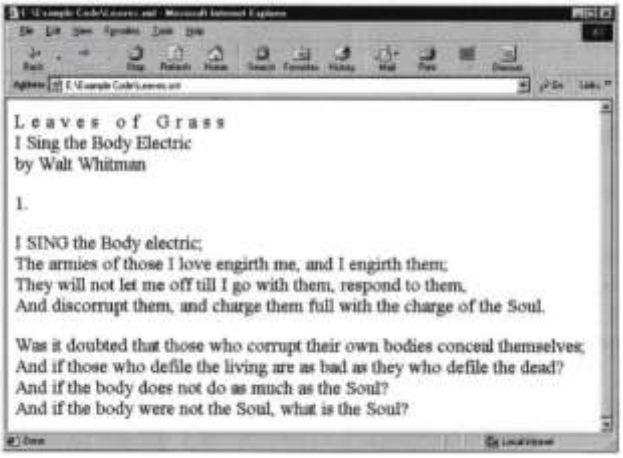
(Untuk penjelasan mengenai macam macam tipe nilai yang bisa Anda berikan, lihat "Menentukan Nilai Ukuran" sebelumnya.)

Atau, Anda bisa memilih spasi karakter normal dengan memberikan letter-spacing nilai normal. Sebagai contoh, style sheet berikut, disertakan ke dokumen XML dalam Listing 12-4, memberikan spasi karakter tambahan ke elemen TITLE, dan ia memberikan spasi karakter normal ke elemen SUBTITLE (pemberian ke dua perlu untuk menimpa spasi karakter tambahan yang akan diwarisi SUBTITLE dari elemen induknya TITLE):

```
POEM
(font-size: l45%}
POEM , TITLE, SUBTITLE,AUTHOR, SECTION ,
NUMBER, STANZA, VERSE
{display: blockl
```

```
SECTION, STANZA
{margin - top: lem}

TITLE
(letter - spacing: .5emJ

SUBTITLE
(letter - spacing: normal}
```

Dengan aturan tersebut dalam style sheet ini, beginilah caranya Internet Explorer 5 menampilkan dokumen XML:



Menyeting Properti vertical-align

Anda bisa memakai properti vertical-align untuk membuat teks superscript atau subscript. Properti ini hanya mempengaruhi elemen inline (elemen inline adalah elemen yang properti displanya diseting jadi inline, seperti dibahas dalam bagian sebelumnya "Menyeting Properti displaj' dalam bab ini).

Anda bisa memberikan vertical-align salah satu kata-kunci CSS dalam

tabel berikut. Untuk membuat setiap contoh teks dalam kolom terakhir, saya memberikan pada setingnya properti vertical-align pada elemen CHILD saja, yang merupakan sebuah elemen inline yang tampak dalam dokumen seperti ini:

```
<PARENT> PARENTELEMENT
<CHILD>CHILDELEMENT</CHILD>
<! PARENT>
```

Menyeting Properti text-align

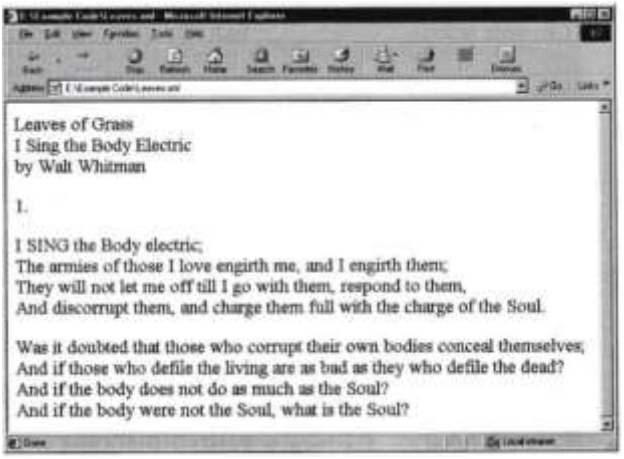
Anda bisa memakai properti text-align untuk mengontrol perataan horizontal teks elemen. Properti ini hanya bekerja jika Anda secara eksplisit memberikannya ke elemen block. Ia akan mempengaruhi elemen itu sendiri, plus semua elemen anak yang dikandungnya, apakah elemen anak tersebut adalah block maupun inline (elemen block dan inline dijelaskan dalam bagian sebelumnya "Menyeting Properti displaj').

Properti text-align mempengaruhi perataan teks dalam area isi teks. Secara default, area isi teks menempati hampir seluruh lebar jendela browser. Seperti yang akan Anda pelajari nanti dalam bab ini (pada "Menyeting Properti Properti Box"), Anda bisa mengubah lebar dan posisi area isi teks elemen.

1. left. Meratakan setiap baris ke kiri. Anggaplah, misalnya, Anda menerapkan aturan berikut ke dokumen XML pada Listing 7-4 (selain untuk aturan lainnya yang tampak dalam style sheet pada Listing 7-3, kecuali seting background-image, yang saya hapus untuk lebih jelas):

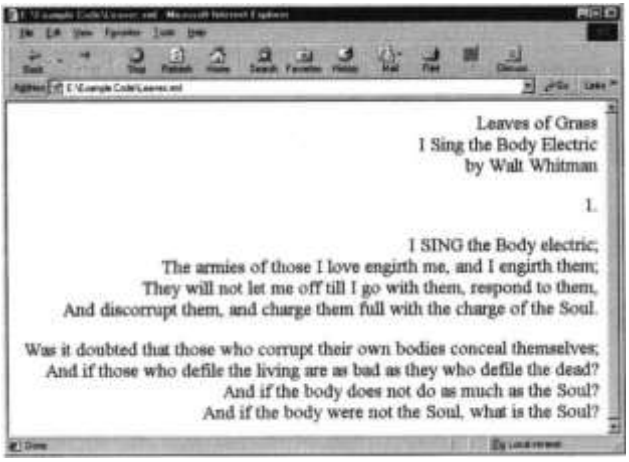
   ```
   POEM {text - align: leftl
   ```

   Puisi tersebut akan diratakan seperti ini:



2. right. Meratakan setiap baris ke kanan. Misalnya, aturan berikut meratakan puisi tersebut ke kanan, seperti tampak dalam gambar ini:

```
POEM (text - align: right}
```



3. center. Menengahkan setiap baris secara horizontal. Misalnya, aturan berikut menengahkan seluruh puisi, seperti bisa Anda lihat dalam gambar ini:

```
POEM {text - align: center}
```

Menyeting Properti text-indent

Anda bisa menggunakan properti text-indent untuk menginden baris pertama pada teks elemen. Anda bisa memberikan text indent berbagai macam nilai ukuran yang dijelaskan dalam sebelumnya "Menentukan Nilai Ukuran " dalam bab ini. Sebagai contoh, aturan berikut menginden baris pertama dari elemen VERSE sebanyak tiga kali tinggi font tersebut:

```
VERSE {text - indent:3em}
```

Beginilah elemen VERSE akan terlihat:

```
It is his walk, the carriage of his
neck, the flex of his waist and kness-dress
does not hide him;
```

Cara lain, Anda bisa menyebutkan indentasi sebagai sebuah persentasi dari total lebar teks elemen. Sebagai contoh, aturan ini menginden baris pertama dari elemen VERSE sebanyak setengah lebar elemen:

```
VERSE {text- indent:50%}
```

Beginilah elemen VERSE akan terlihat:

```
It is his walk, the carriage of his neck,
the flex of his waist and kness - dress
does not hide him;
```

Anda bisa memberikan sebuah nilai negatif-juga nilai ukuran atau persentasi-untuk memindah baris pertama, ke luar ke kiri baris lainnya. Jika Anda hanya memberikan sebuah nilai negatif ke properti text-indent, bagian pertama baris tersebut akan disembunyikan, seperti tampak di sini:

```
in his walk, the carriage of his neck, the
flex of his waist and knees-dress does not
hide him;
```

Untuk menghindari penyembunyian teks, Anda harus menerapkan sebuah margin kiri ke elemen yang bersangkutan. Sebagai contoh, aturan berikut

menerapkan margin kiri 4em (margin-left :4em) dan kemudian memindahkan baris pertama keluar 2em (text-indent: -2em), membuat sebuah inden gantung, seperti tampak dalam gambar ini:

```
VERSE
{margin - left:4em; text- indent: - 2em}
```

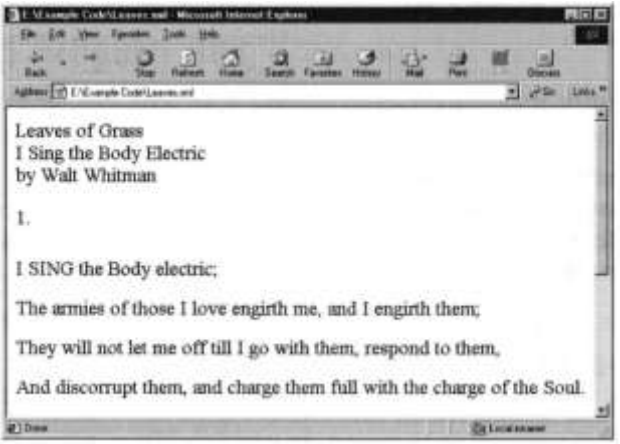
Menyeting Properti line-height

Properti line-height mengontrol jarak antarbaseline dari baris berikutnya pada sebuah teks elemen. Anda memakainya untuk menyesuaikan spasi baris vertikal pada teks.

Anda bisa memberikan line-height berbagai macam nilai yang dijelaskan dalam "Menentukan Nilai Ukuran" sebelumnya. Anggaplah, misalnya, Anda menerapkan aturan berikut pada dokumen XML dalam Listing 7-4 (di samping aturan dalam Listing 7-3, kecuali seting background-image, yang saya hapus agar lebih jela s):

```
STANZA {line - height: 2em}
```

Teks STANZA akan ditampilkan dalam baris dua-spasi, seperti terlihat di sini:



Sebagai alternatif, Anda bisa menyebutkan tinggi-baris sebagai sebuah persentasi dari tinggi teks elemen. Aturan berikut, misalnya, akan sama. dengan aturan yang diberikan di atas, dan akan menghasilkan baris dengan spasi ganda:

```
STANZA {line - height: 200%}
```

Menyeting Properti Box

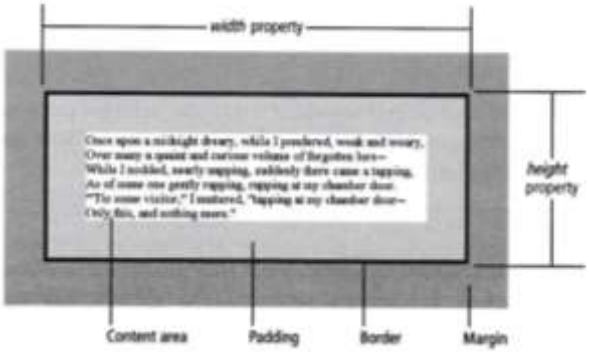
Spesifikasi CSS menyediakan sekumpulan properti yang dikenal dengan properti box yang bisa Anda gunakan untuk memformat sebuah blok teks yang dimiliki sebuah elemen. Macam-macam tipe properti box adalah sebagai berikut:

1. Properti margin menambahkan sebuah margin transparan di sekitar elemen. Di luar border yang tampak (jika ada).
2. Properti border menampilkansebuah border yang tampak dalam berbagai gaya di sekitar elemen, di luar padding (jika ada).
3. Properti padding menambahkan spasi tepat di luar isi elemen, dalam

border (jika ada).

4. Properti size, height dan width, mengontrol dimensi area isi elemen plus semua padding atau border yang termasuk (seperti dalam gam- bar berikut).
5. Properti positioning, float dan clear, mengontrol posisi elemen yang berkenaan dengan penggabungan elemen.

Gambar berikut mengilustrasikan empat kelompok pertama dari properti box yang diterapkan pada elemen block



Ingatlah kembali dari bagian sebelumnya "Menyeting Properti displaj' di mana elemen block adalah elemen yang properti displaynya diseting ke nilai block, sementara elemen inline adalah elemen yang properti displaynya diseting jadi inline. Pada Internet Explorer 5.0 dan 5.01, tiga kelompok pertama dari properti (margin, border, dan padding) hanya berpengaruh pada elemen block saja. Pada Internet Explorer 5.5, Anda bisa menggunakan properti ini baik dengan elemen block maupun inline. Pada semua versi Internet Explorer (5.0 sampai 5.5), Anda bisa menerapkan properti ukuran/size atau posisi baik pada elemen inline maupun block; properti positioning, lebih efektif dan dapat diperkirakan

bila dipakai dengan elemen block.

Elemen anak tidak disertakan dalam properti box.

Menyeting Properti Margin

Secara default, lebar margin sekeliling elemen adalah nol. Untuk menambahkan satu atau lebih sisi pada suatu elemen, Anda bisa memberikan sebuah nilai bukan-nol ke satu atau lebih properti berikut ini:

1. margin-top
2. margin-right
3. margin-bottom
4. margin-left

Anda bisa memberikan prope1ti ini ke berbagai macam nilai ukuran yang dijelaskan dalam "Menentukan Nilai Ukuran " yang dibahas sebelumnya pada bab ini. Sebagai contoh, aturan berikut ini menambahkan sebuah margin ke kiri dan kanan elemen STANZA. Lebar margin tersebut adalah dua kali tinggi teks elemen:

```
STANZA
{margin - left:2em; margin - right:2em}
```

Anda juga bisa menyebutkan ukuran margin sebagai persentasi lebar induk elemen. Misalnya, aturan berikut membuat margin kiri sama dengan 1/4 lebar induk elemen:

```
STANZA {margin - left:25%}
```

Sebagai sebuah shortcut, Anda bisa menambahkan sebuah margin yang sama ukurannya ke semua (empat) sisi elemen dengan memberikan sebuah nilai tunggal sebuah nilai ukuran atau persentasi pada property margin. Untuk mengilustrasikan, pertama perhatikan style sheet yang diberikan

dalam Listing 12-5, yang disertakan ke dokumen XML yang di- berikan dalam Listing 12-6, dan menampilkan teks tersebut tanpa margin. (Anda bisa temukan salinan kedua listing ini dalam CD penyerta dengan nama Raven.css dan Raven.xml.)

Raven.css

```
Raven.css
/* FileName: Raven.css*/
POEM
{font- size: smal l}
POEM, TITLE. AUTHOR, DATE, STANZA. VERSE
{display: block}
```

Listing 12-5.

Raven.xml

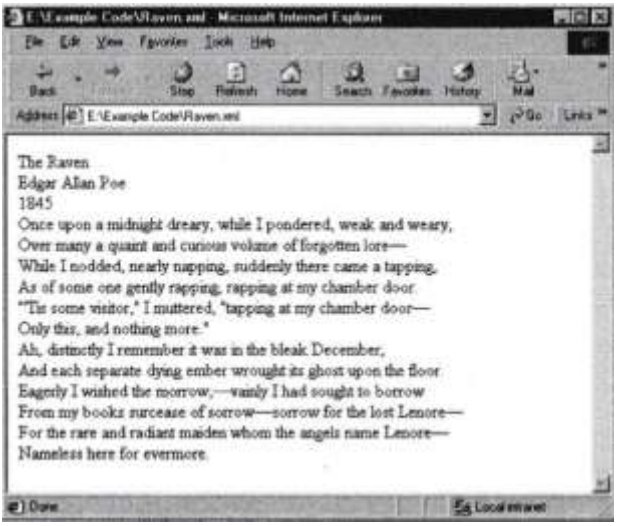
```
<?xml version="1.0"?>
<!-- File Name: Raven.xml -->
<?xml-stylesheet type="text/css"
href="Raven.css"?>
<POEM>
<TITLE>The Raven</TITLE>
<AUTHOR>Edgar Allan Poe</AUTHOR>
<DATE>1845</DATE>
<STANZA>
   <VERSE>Once upon a midnight dreary, while
I pondered, weak and weary,</VERSE>
   <VERSE>Over many a quaint and curious
volume of forgotten lore&#8212;</VERSE>
   <VERSE>While I nodded, nearly napping,
suddenly there came a tapping,</VERSE>
   <VERSE>As of some one gently rapping,
rapping at my chamber door.</VERSE>
```

```
   <VERSE>"'Tis some visitor," I muttered,
"tapping at my chamber door&#8212;</VERSE>
   <VERSE>Only this, and nothing
more."</VERSE>
</STANZA>
<STANZA>
   <VERSE>Ah, distinctly I remember it was
in the bleak December,</VERSE>
   <VERSE>And each separate dying ember
wrought its ghost upon the floor.</VERSE>
   <VERSE>Eagerly I wished the
morrow;&#8212;vainly I had sought to
borrow</VERSE>
   <VERSE>From my books surcease of
sorrow&#8212;sorrow for the lost
Lenore&#8212;</VERSE>
   <VERSE>For the rare and radiant maiden
whom the angels name Lenore&#8212;</VERSE>
   <VERSE>Nameless here for
evermore.</VERSE>
</STANZA>
</POEM>
```

Listing 12-6.

CATATAN &#8212 adalah acuan karakter untuk karakter em dash (-).

Beginilah bagaimana elemen tersebut terlihat tanpa margin:



Penambahan aturan berikut ke style sheet menyisipkan margin 2, Sem di sekeliling sisi dua elemen STANZA, seperti dalam gambar ini:

CATATAN Area margin adalah selalu transparan, artinya warna atau citra back- ground elemen induk ditampilkan keseluruhan.

Menyeting Properti Border

Anda bisa menggunakan properti berikut untuk menggambarkan border yang terlihat di sekeliling elemen:

1. border-style
2. border-width
3. border-color

Menyeting Properti border-color

Secara default, border yang Anda buat dengan properti border-style memiliki warna yang sama dengan seting properti color pada elemen yang ada. Anda bisa mengganti warna semua border dengan memberikan properti border-color sembarang nilai warna yang disebutkan dalam "Menentukan Nilai Warna," yang telah dibahas sebelumnya. Sebagai contoh, aturan berikut menambahkan border merah solid ke semua sisi elemen TITLE:

```
TITLE
{border - style: solid; border - color: red)
```

Anda bisa mengganti warna border individual di sekeliling elemen dengan memberikan empat nilai warna yang berbeda pada border-color. Nilai tersebut menunjukkan border atas, kanan, bawah, dan kiri dari elemen TITLE, dan sebuah border hijau solid pada elemen kanan dan kiri:

```
TITLE
```

```
(border - style: solid; border - color: red}
```

Menyeting Properti Padding

Ingatlah kembali pada bagian awal "Menyeting Properti Box, " bahwa properti padding menambahkan spasi persis mengelilingi isi elemen, dalam border elemen. Tanpa padding, border akan terlihat terlalu dekat dengan teks elemen. Anda bisa meningkatkan tampilan border dengan menambahkan padding.

Secara default, lebar area padding sekitar elemen adalah nol. Untuk menambahkan padding pada satu atau beberapa sisi elemen, Anda bisa memberikan sebuah nilai bukan-nol pada properti berikut:

1. padding-top
2. padding-right
3. padding-bottom
4. padding-left

Anda bisa memberikan properti ini berbagai macam nilai ukuran yang dijelaskan dalam "Menentukan Nilai Ukuran," yang telah dibahas sebelumnya. Sebagai contoh, aturan berikut menambahkan padding pada bagian atas dan bawah elemen STANZA. Lebar area padding adalah dua kali tinggi teks elemen:

```
STANZA
(padding - top:2em; padding- bottom :2em}
```

Anda juga bisa menyebutkan lebar area padding sebagai sebuah persentasi dari lebar induk elemen. Misalnya, aturan berikut menambahkan padding ke bagian kiri elemen STANZA. Ketebalan padding sama dengan 1/4 lebar induk elemen:

```
STANZA{padding-left:25%}
```

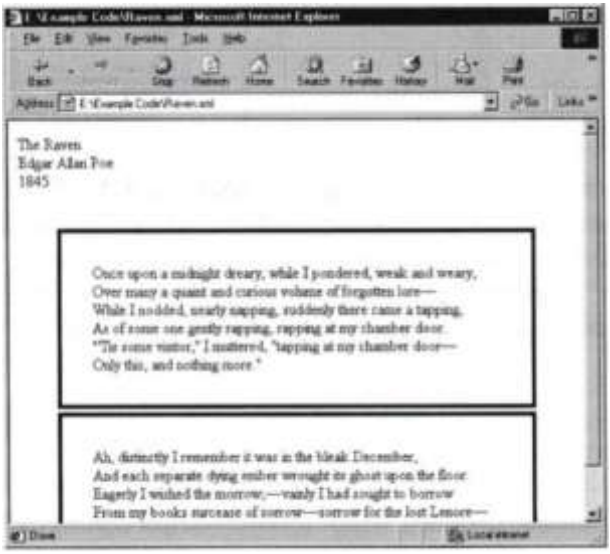
Sebagai shortcut, Anda bisa menambahkan padding dengan ketebalan yang sama pada semua sisi elemen dengan memberikan sebuah nilai tunggal-nilai ukuran atau persentasi-ke properti padding. Anggaplah, misalnya, bahwa Anda menambahkan aturan berikut ke contoh style sheet dalam Listing 7-5:

```
STANZA
{margin:2.5em;
border - style: solid; padding:2em}
```

Aturan ini akan menampilk an fitur pemformatan berikut di sekeliling masing-masing elemen STANZA:

1. Padding, setebal 2em, tepat sekeliling elemen.
2. Border solid di luar padding
3. Margin di luar border

Seperti inilah dokumen tampak dalam Internet Explorer 5:



CATATAN Seperti area isi elemen lainnya, area padding memperlihatkan warna atau citra background yang Anda berikan pada elemen tersebut. (lngatlah bahwa, sebaliknya, area margin memperlihatkan background pada elemen induk.)

Menyeting Properti Size

Properti size (ukuran), width dan height, mengontrol dimensi area isi elemen, plus semu a padding atau border yang dimasukkan. (Lihat gambar pertama dalam bagian “Menyeting Properti Box" pada bab ini.)

Anda bisa memberikan pada properti width dan height tipe-tipe nilai berikut:

1. Semua jenis nilai ukuran yang dijelaskan dalam "Menentukan Nilai Ukuran" pada bah ini. Sebagai contoh, aturan berikut mengatur properti width elemen STANZA menjadi 3 inci, dan properti heightnya 2 inci:

   ```
   STANZA
   (width:3in; height:2in)
   ```

2. Persentasi lebar atau tinggi elemen induk. Sebagai contoh, aturan berikut mengatur properti width dan height elemen STANZA menjadi setengah dari lebar dan tinggi elemen induk:

   ```
   STANZA
   {width :50%; height:50%)
   ```

3. Nilai kata kunci auto, yang merupakan default. Nilai ini menyebabkan browser menyesuaikan properti width dan height untuk mengakomodasi ukuran sesungguhnya dari teks. Misalnya, aturan berikut mengatur properti width dan height ke auto (yang punya akibat

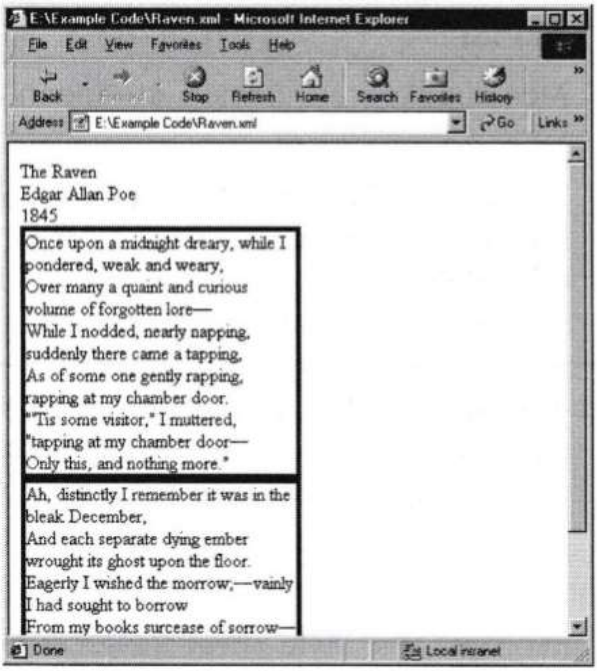
sama dengan membuang seting properti width dan height):

```
STANZA
{width: auto;
height: auto}
```

Jika Anda membuat properti width terlalu sempit sehingga baris teks tidak pas dalam area isi yang dihasilkan, browser akan melipat teks tersebut untuk membuatnya sesuai. Jika teks tidak bisa dimuat secara vertikal dalam area isi yang dihasilkan dari seting height Anda, browser akan menambah seting height untuk mengakomodasi teks tersebut, seperti ketika Anda menjadikan height ke auto.

Sebagai contoh, jika Anda menambahkan aturan berikut ke style sheet yang tampak dalam Listing 7-5, browser akan menampilkan dokumen XML dalam Listing 7-6 (untuk mana style sheet disertakan), seperti tampak dalam gambar berikut:

```
STANZA
{border - style: solid; width:2.5in;
height: lin}
```



Perhatikan teks melipat untuk menyesuaikan dalam lebar 2,5 inci yang diminta, namun tingginya bertambah 1 inci dari yang diminta untuk meng-akomodasi isi teks seluruhnya.

Menyeting Properti Positioning

Anda bisa menggunakan properti positioning, float dan clear, untuk mengontrol posisi elemen block yang menunjukkan teks dokumen selebihnya.

Menyeting Properti float

Secara default, isi elemen block memiliki seluruh lebar jendela browser, dengan teks dokumen awal yang ditempatkan di atasnya, dan teks

dokumen yang mengikutinya ditempatkan di bawahnya, mirip seperti paragraf dalam dokumen pengolah kata pada umumnya. Anda bisa menggunakan properti float untuk menampilkan sebuah isi elemen block berja jar (yakni, kiri ke kanan) pada teks elemen berikut ini.

Anda bisa memberikan properti float salah satu dari tiga nilai kata-kunci berikut.

Kata-kunci float	Akibat
left	Menampilkan isi elemen ke kiri teks dokumen yang mengikutinya.
right	Menampilkan isi elemen ke kanan teks dokumen yang mengikutinya
none (default)	Mematikan fitur float, sehingga elemen tersebut ditempatkan sebagai elemen block biasa. Yakni, teks dokumen awal ditempatkan di atasnya dan teks dokumen yang mengikutinya di bawahnya.

Dalam latihan pada dua bagian berikut, Anda akan belajar cara menggunakan properti float untuk membuat sebuah catatan margin, dan untuk menampilkan sebuah citra mengambang di depan teks elemen.

Membuat Catatan Margin

1. Dari editor teks Anda, buka file style sheet Raven.css yang diberikan dalam Listing 12-5.
2. Ubahlah style sheet tersebut sehingga ia terlihat seperti ditunjukkan dalam Listing 12-7. Ini adalah fitur-fitur baru yang Anda tambahkan ke style sheet semula:

a. Anda memberikan elemen STANZA mengirim kiri satu inci.
b. Elemen NOTE yang telah Anda format (di mana Anda menambahkan ke dokumen sebelumnya) sebagai sebuah tampilan catatan margin dalam area margin kiri elemen STANZA pertama. Khususnya:
    - Anda memberikan elemen STANZA margin kiri satu inci.
    - Anda menengahkan teksnya.
    - Anda memberikan pada properti width dan height sebuah nilai satu inci.
    - Anda membuatnya mengambang pada bagian kiri teks elemen yang mengikutinya

3. Gunakan perintah Save As pada editor teks untuk menyimpan salinan dokumen hasil modifikasi ini dengan nama RavenOl.css.

Raven0l.css

```
/* File Name: Raven01.css */
POEM
   {font-size:12pt}
POEM, TITLE, AUTHOR, DATE, NOTE, STANZA,
VERSE
   {display:block}
DATE
   {margin-bottom:.25in}
STANZA
   {margin-left:1in;
    margin-bottom:.25in}
NOTE
   {border-style:solid;
    border-width:1px;
    text-align:center;
    width:1in;
    height:1in;
```

```
        float:left}
```

Listing 12-7.

4. Dalam editor teks Anda, bukalah dokumen Raven.xml yang diperlihatkan dalam Listing 12-6.
5. Dalam Raven.xml, editlah instruksi pemrosesan xml-stylesheet pada bagian awal file sehingga menunjukkan style sheet baru yang Anda buat-raveOl .css-seperti ini:

```
(? xml - stylesheet type="text/css"
href="RavenOl.css"?>
```

6. Dalam Raven.xml, tambahkan elemen baru berikut ini tepat di atas elemen STANZA pertama:

```
< NOTE> This is a floating margi nnote.
</NOTE>
```

   Karena Anda memberikan NOTE seting properti float: left dalam style sheet tersebut, ia akan mengambang pada bagian kiri teks dokumen yang mengikutinya yakni, pada bagian kiri elemen STANZA pertama.

7. Gunakan perintah Save As pada editor teks untuk menyimpan salinan dokumen basil modifikasi ini dengan nama RavenOl.xml.

   Anda bisa melihat dokumen selengkapnya dalam Listing 12-8. Dokumen ini disediakan

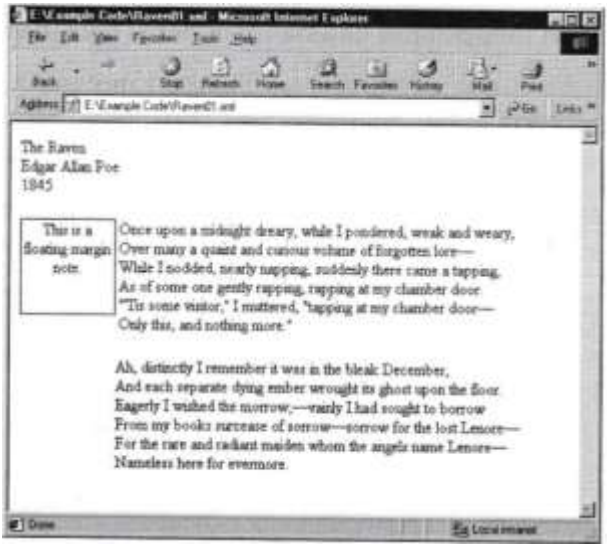
8. Buka RavenOl.xml dalam Internet Explorer 5, yang akan menampilkannya seperti ini:

Raven0l.xml

```
<?xml version="1.0"?>
<!-- File Name: Raven.xml -->
<?xml-stylesheet type="text/css"
href="Raven.css"?>
<POEM>
<TITLE>The Raven</TITLE>
<AUTHOR>Edgar Allan Poe</AUTHOR>
<DATE>1845</DATE>
```

```
<STANZA>
   <VERSE>Once upon a midnight dreary, while
I pondered, weak and weary,</VERSE>
   <VERSE>Over many a quaint and curious
volume of forgotten lore&#8212;</VERSE>
   <VERSE>While I nodded, nearly napping,
suddenly there came a tapping,</VERSE>
   <VERSE>As of some one gently rapping,
rapping at my chamber door.</VERSE>
   <VERSE>"'Tis some visitor," I muttered,
"tapping at my chamber door&#8212;</VERSE>
   <VERSE>Only this, and nothing
more."</VERSE>
</STANZA>
<STANZA>
   <VERSE>Ah, distinctly I remember it was
in the bleak December,</VERSE>
   <VERSE>And each separate dying ember
wrought its ghost upon the floor.</VERSE>
   <VERSE>Eagerly I wished the
morrow;&#8212;vainly I had sought to
borrow</VERSE>
   <VERSE>From my books surcease of
sorrow&#8212;sorrow for the lost
Lenore&#8212;</VERSE>
   <VERSE>For the rare and radiant maiden
whom the angels name Lenore&#8212;</VERSE>
   <VERSE>Nameless here for
evermore.</VERSE>
</STANZA>
</POEM>
```

Listing 12-8.



Menampilkan Citra Mengambang

1. Dalam editor teks Anda, bukalah file style sheet Raven.css yang diperlihatkan dalam Listing 12-5.
2. Ubahlah style sheet tersebut sehingga terlihat seperti ditunjukkan dalam Listing 12-9.

   Tambahan utama pada style sheet yang asli adalah aturan untuk elemen IMAGE:

```
IMAGE
! background - image: url (Raven.bmp);
background - repeat: no- repeat;
background - position: center;
width :89px; height :58px; float: left}
```

   IMAGE adalah sebuah elemen kosong (yang nanti akan Anda tambahkan pada dokumen XML) yang dirancang untuk menampilkan

citra mengambang. Elemen tersebut tidak berisi teks, namun diberikan sebuah citra background (ketiga seting properti dalam aturan tersebut) untuk ditampilkan sebagai gantinya.

Anda berikan pada properti width dan height lebar dan tinggi eksak citra tersebut. Karena file citra tersebut berupa bitmap, penting untuk memberikan ukurannya dalam satuan pixel, sehingga seluruh citra bisa ditampilkan pada semua monitor dan berbagai mode grafis. Perhatikan, jika Anda tidak memberikan nilai-nilai width dan height, ukurannya akan menjadi no! karena tidak berisi teks, dan akan disembunyikan.

Terakhir, Anda memberikan properti float nilai left sehingga citra tersebut mengambang pada bagian kiri teks dokumen yang mengikutinya.

Raven0l.css

```
/* File Name: Raven02.css */
POEM
   {font-size:12pt}
POEM, TITLE, AUTHOR, DATE, IMAGE, STANZA,
VERSE
   {display:block}
DATE, STANZA
   {margin-bottom:.25in}
IMAGE
   {background-image:url(Raven.bmp);
    background-repeat:no-repeat;
    background-position:center;
    width:89px;
    height:58px;
    float:left}
```

Listing 12-9.

3. Dalam editor teks Anda, bukalah dokumen Raven.xml yang diperlihatkan dalam Listing 7-6.
4. Dalam Raven.xml, editlah instruksi pemrosesan xml-sty lesheet pada awal file sehingga menunjukkan style sheet baru yang Anda buat Raven02.css seperti ini:

```
<? xml - stylesheet type-"text /css"href-
"Raven02.css"?>
```

5. Dalam Raven .xml, tambahkan elemen IMAGE kosong berikut ini tepat di atas setiap elemen STANZA:

```
< IMAGEI>
```

6. Karena Anda memberikan elemen IMAGE seting properti jloat: left dalam style sheet tersebut, mereka akan mengambang pada bagian kiri setiap elemen STANZA (yang berisi teks dokumen yang mengikutinya).
7. Gunakan perintah Save As pada editor teks untuk menyimp an salinan dokumen hasil modifikasi ini dengan nama Raven02.xml.

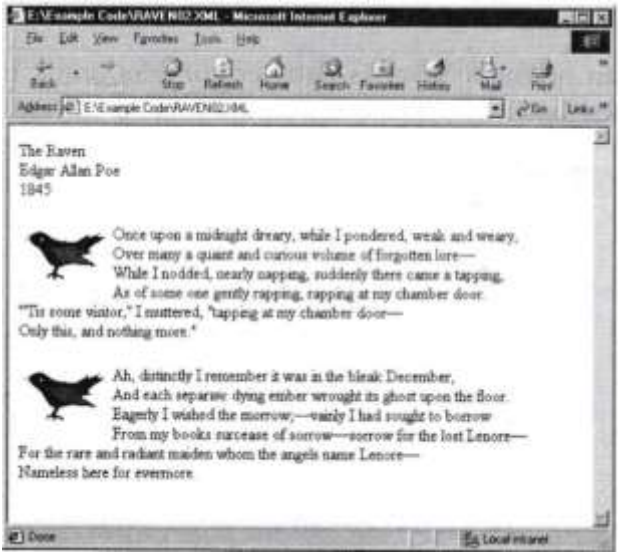
Raven02.xml

```
<?xml version="1.0"?>
<!-- File Name: Raven02.xml -->
<?xml-stylesheet type="text/css"
href="Raven02.css"?>
<POEM>
<TITLE>The Raven</TITLE>
<AUTHOR>Edgar Allan Poe</AUTHOR>
<DATE>1845</DATE>
<IMAGE />
<STANZA>
   <VERSE>Once upon a midnight dreary, while
I pondered, weak and weary,</VERSE>
```

```
   <VERSE>Over many a quaint and curious
volume of forgotten lore&#8212;</VERSE>
   <VERSE>While I nodded, nearly napping,
suddenly there came a tapping,</VERSE>
   <VERSE>As of some one gently rapping,
rapping at my chamber door.</VERSE>
   <VERSE>"'Tis some visitor," I muttered,
"tapping at my chamber door&#8212;</VERSE>
   <VERSE>Only this, and nothing
more."</VERSE>
</STANZA>
<IMAGE />
<STANZA>
   <VERSE>Ah, distinctly I remember it was
in the bleak December,</VERSE>
   <VERSE>And each separate dying ember
wrought its ghost upon the floor.</VERSE>
   <VERSE>Eagerly I wished the
morrow;&#8212;vainly I had sought to
borrow</VERSE>
   <VERSE>From my books surcease of
sorrow&#8212;sorrow for the lost
Lenore&#8212;</VERSE>
   <VERSE>For the rare and radiant maiden
whom the angels name Lenore&#8212;</VERSE>
   <VERSE>Nameless here for
evermore.</VERSE>
</STANZA>
</POEM>
```

Listing 12-10.

8. Buka Raven02.xml dalam Internet Explorer 5, yang akan menampilkannya seperti ini:



Menyeting Properti clear

Secara default, sebuah elemen mengambang (yakni sebuah elemen yang properti float diberikan left atau right) akan ditampilkan pada sisi kiri atau kanan teks dokumen yang mengikutinya. Anda bisa menggunakan properti clear dengan elemen tertentu untuk mencegah sebuah elemen mengambang ditampilkan di sisinya.

Anda bisa memberikan properti clear sebuah elemen dengan salah satu nilai kata-kunci berkut:

Kata-kunci clear	Pengaruhnya
left	Elemen akan ditampilkan di bawah bukan di sisi elemen mengambang terdahulu dengan seting properti float :left.

right	Elemen akan ditampilkan di bawah bukan di sisi elemen mengambang terdahulu dengan seting properti jloat: rigbt.
botb	Elemen akan ditampilkan di bawah bukan di sisi elemen mengambang terdahulu dengan seting properti float: left atau float: right.
none (default)	Elemen tersebut akan ditampilkan di samping elemen mengambang terdahulu.

Sebagai contoh, jika Anda menambahkan aturan berikut ke contoh style sheet pada Listing 12-9, setiap elemen STANZA akan ditampilkan di bawah, bukan di samping elemen IMAGE terdahulu (mengambang), seperti terlihat dalam gambar di bawah:

```
STANZA
{clear: left}
```

Menggunakan Pseudo-Elemen (hanya untuk Internet Explorer 5.5)

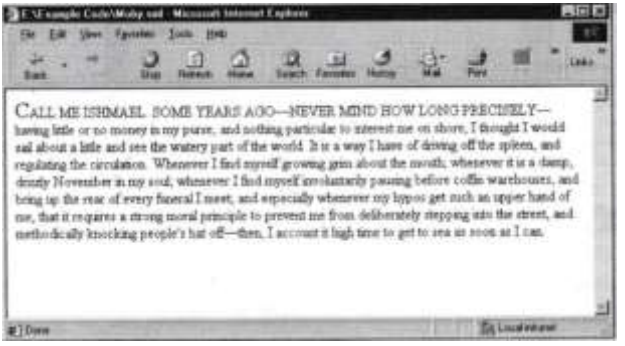
Jika Anda ingin menampilkan sebuah dokumen XML dalam Internet Explorer 5.5, Anda bisa membuat sebuah aturan yang hanya diterapkan pada huruf pertama elemen block dengan menambahkan. first letter ke nama elemen dalam selektor. Begitu juga, Anda bisa membuat sebuah aturan yang hanya diterapkan pada baris perta ma elemen block dengan menambahkan. first-line pada nama elemen dalam selektor. Penggunaan kedua ekspresi ini membuat apa yang dikenal dengan sebuah pseudo-elemen "pseudo " dalam artian bahwa aturan tersebut diterapkan pada sebuah blok teks yang bukan merupakan elemen terpisah.

Sebagai contoh, jika sebuah dokumen XML berisi serangkaian elemen PARAGRAPH, aturan berikut akan menampilkan huruf pertama dari

masing-masing elemen dalam font yang lebih besar, dan akan mengubah baris pertama menjadi huruf besar semuanya:

```
PARAGRAPH
(display: block; font- size: small}
PARAGRAPH: first - letter
{font-ze: large}
PARAGRAPH: first- line
{text- transform: uppercase}
```

Seperti inilah Internet Explorer 5 akan menampilkan elemen PARAGRAPH:



Menyisipkan Elemen HTML ke Dokumen XML dan Menggunakan Namespaces

Walaupun Anda bisa menggunakan style sheet be1tingkat untuk menambahkan fitur pemformatan dasar ke elemen XML dalam dokumen Anda, ia akan lebih cocok untuk menambahkan elemen standar XML seperti halnya hyperlink, citra, dan form-sehingga dokumen Anda bisa diuntungkan dari fitur-fitur built-innya. Untungnya, ketika Anda menampilkan dokumen melalui penyisipan style sheet bertingkat, Anda bisa menyisipkan berbagai elemen HTML standar ke dalam dokumen

Anda, dan menjadikan browser menampilkan elemen itu menggunakan nama elemen khusus yang disediakan untuk keperluan ini.

Anda mungkin mengira bahwa Anda bisa menyisipkan sebuah elemen HTML hanya dengan memberikan elemen XML nama yang sama. Sebagai contoh, ini tampaknya memungkinkan untuk menyisipkan sebuah elemen IMG HTML hanya dengan membuat sebuah elemen yang dinamai IMG, seperti berikut:

```
<IMGSRC="Raven.bmp"I>
```

Bagaimanapun juga, browser tidak punya cara untuk mengetahui bahwa ini adalah sebuah elemen HTML, dan bukan sekedar elemen XML biasa yang Anda bayangkan. Agar mekanisme semacam ini bisa bekerja, semua nama elemen HTML (dan ada banyak macamnya) akan disediakan secara eksklusif untuk penyisipan elemen HTML. Seperti sebuah aturan akan bertentangan dengan semangat XML, menurut mana semestinya Anda diperkenankan untuk memakai berbagai nama legal yang Anda inginkan untuk elemen Anda.

Untunglah, Anda bisa menggunakan sebuah konvensi XML yang dikenal sebagai namespace untuk membedakan nama yang berlawanan. Dua elemen berbeda bisa memiliki nama yang sama asalkan merek a ada dalam namespace yang terpisah.

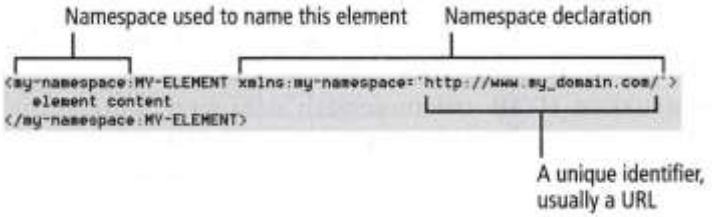
Nama namespaces ditambahkan ke awal nama elemen dan dipisahkan dari nama yang mengikutinya dengan tanda bagi (:), seperti dalam contoh ini:

```
my - namespace: MY - ELEMENT
```

Sebuah elemen yang dinamai my-namespace: MY-ELEMENT dan sebuah elemen yang dinamai MY-ELEMENT bisa muncul dalam dokumen yang sama, dan akan dianggap elemen berbeda karena mereka dimiliki oleh

namespace tersendiri: my-namespace:MY-ELEMENT adalah dalam namespace my-namespace, sementara MY-ELEMENT ada dalam namespace dokumen default.

Sebelum Anda bisa menggunakan sebuah namespace, Anda harus mendeklarasikannya secara eksplisit. Walau pun ada sejumlah tempat di mana Anda bisa mendeklarasikan sebuah namespace, tempat termudah adalah dalam tag-awal dari elemen di mana Anda ingin menggunakan namespace. Misalnya, beginilah caranya Anda bisa mendeklar asikan mynamespace:



Perhatikan dengan bentuk deklarasi namespace ini, Anda bisa menggunakan namespace hanya dalam elemen di mana Anda mendeklarasikannya, atau dalam berbagai elemen anaknya.

Jika sebuah elemen XML memiliki nama sebuah elemen XML (seperti misalnya IMG, A, atau HR), dan jika ia ada dalam namespace html, Internet Explorer 5 akan menganggapnya sebagai sebuah elemen HTML, dan akan menyisipkan elemen HTML ini ke dalam halaman yang ditampilkan. Jika elemen tersebut tidak dalam namespace html, Internet Explorer 5 akan mengangapnya sebagai sebuah elemen XML biasa.

Namespace html bersifat khusus, merupakan namespace yang dicadangkan, yakni dideklarasikan seperti berikut:

```
xmlns: html= ' http://www.w3c.org/TR/REC -
htm1 40/ '
```

Untuk menempatkan ini semua secara bersama-sama, di sini sebuah element XML menyebabkan Internet Explorer 5 menyisipkan sebuah elemen HTML IMG (image), di mana sumber image adalah file Raven.bmp.

```
< html: IMG xmlns: html= ' http ://www.
w3c.org /TR/REC - html40/' SRC = ' Raven.
bmp'I>
```

Ini merupaka n elemen XML kosong yang well-formed. Ia memiliki nama yang memasukk an sebuah namespace, dan memiliki dua atribut. Atribut pertama mendeklarasikan namespace, sementara atribut kedua adalah salah satu dari atribut HTML standar yang bisa Anda masukkan dalam tag-awal elemen IMG HTML.

Ingatlah baik-baik bahwa untuk menyisipkan HTML sepe1ti dijelaskan di sini, dokumen XML harus memiliki sebuah lembatr gaya yang disertakan, dan Anda harus membukanya langsung dalam browser (seperti semua dokumen XML yang dijelaskan dalam bab ini).

TIP Anda bisa menemukan spesifikasi namespace resmi pada halaman Web W3C

http://www.w3c.org!TRIREC-xml-names!

Sebuah Contoh

Versi dokumen RAVEN yang terlihat dalam Listing 12-11 mengilustrasikan teknik untuk memasukkan HTML dalam sebuah dokumen XML. Perhatikan dokumen ini menyertakan versi asli style sheet

tersebut, Raven.css, yang bisa Anda temukan dalam Listing 7-5.

Dokumen tersebut memasukkan tiga elemen HTML standar:

1. la memasukkan sebu ah citra melalui elemen HTML berikut:

```
< html: IMG xml ns: html
='http://www.w3c.org/TR/REC - html 40/'
SRC= 'Raven.bmp ' ALIGN= 'LEFT' >
```

   Elemen ini menyisipkan sebuah elemen (citra) IMG HTML standar. Atribu t HTML ALIGN= 'LEFT ' membuat citra tersebut meng ambang di bagian kiri teks dokumen yang mengikutinya. Ini merupakan sebuah alternatif untuk metode menampilkan sebuah citra yang telah Anda pelajari dalam bagian sebelumnya “Menampilkan Citra yang Mengambang ".

2. la menjadikan nama penulis (sebelumnya tersimpan dalam elemen AUTHOR) sebuah hyperlink dengan memasukkan elemen XML berikut (menggantikan AUTHOR):

```
< html: A xmlns:
html='http://www.w3c.org/TR/REC - html 40/;
HREF='http: //www.edgar.com'>
Edgar Allan Poe
<! html: A >
```

   Elemen ini menyisipkan sebuah elemen (jangkar) HTML standar.

3. Ia menyisipkan dua garis pembagi horizontal menggunakan elemen XML berikut pada ke dua tempat:

```
< html: HR
xmlns:html='http://www.w3c.org/TR/REC -
html 40/'I >
```

   Elemen ini menyisipkan sebuah elemen (aturan horizontal) HR HTML standar.

Beginilah caranya Internet Explorer 5 menampilkan dokumen tersebut:

Raven03.XML

```
<?xml version="1.0"?>
<!-- File Name: Raven03.XML -->
<?xml-stylesheet type="text/css"
href="Raven.css"?>
<POEM>
<html:IMG
xmlns:html='http://www.w3c.org/TR/REC-
html40/'
   SRC='Raven.bmp' ALIGN='LEFT' />
<TITLE>The Raven</TITLE>
<html:A
xmlns:html='http://www.w3c.org/TR/REC-
html40/'
   HREF='http://www.edgar.com'>
```

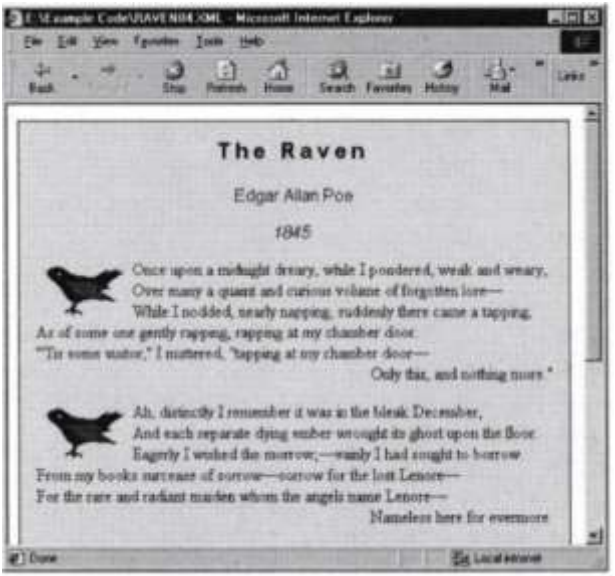
```
   Edgar Allan Poe
</html:A>
<DATE>1845</DATE>
<html:HR
xmlns:html='http://www.w3c.org/TR/REC-
html40/' />
<STANZA>
   <VERSE>Once upon a midnight dreary, while
I pondered, weak and weary,</VERSE>
   <VERSE>Over many a quaint and curious
volume of forgotten lore&#8212;</VERSE>
   <VERSE>While I nodded, nearly napping,
suddenly there came a tapping,</VERSE>
   <VERSE>As of some one gently rapping,
rapping at my chamber door.</VERSE>
   <VERSE>"'Tis some visitor," I muttered,
"tapping at my chamber door&#8212;</VERSE>
   <VERSE>Only this, and nothing
more."</VERSE>
</STANZA>
<html:HR
xmlns:html='http://www.w3c.org/TR/REC-
html40/' />
<STANZA>
   <VERSE>Ah, distinctly I remember it was
in the bleak December,</VERSE>
   <VERSE>And each separate dying ember
wrought its ghost upon the floor.</VERSE>
   <VERSE>Eagerly I wished the
morrow;&#8212;vainly I had sought to
borrow</VERSE>
```

```
   <VERSE>From my books surcease of
sorrow&#8212;sorrow for the lost
Lenore&#8212;</VERSE>
   <VERSE>For the rare and radiant maiden
whom the angels name Lenore&#8212;</VERSE>
   <VERSE>Nameless here for
evermore.</VERSE>
</STANZA>
</POEM>
```

Listing 7-11.

Membuat dan Menggunakan Style Sheet Bertingkat Full-Fitur

Dalam latihan berikut, Anda akan membuat sebuah dokumen XML yang berisi empat stanza pertama dari "The Raven. " Anda kemudian akan membuat sebuah style sheet bertingkat yang secara luas memformat dokumen ini, dan dalam prosesnya menggunakan hampir semua properti yang telah dibahas dalam bab ini. Gambar berikut memperlihatkan bagaimana puisi tersebut tampak dalam Internet. Explorer 5.



Membuat Dokumen

1. Buka sebuah file teks kosong yang baru dalam editor teks Anda, dan ketikkan dalam dokumen XML seperti tampak dalam Listing 7-12.
   a. Ia menyertakan style sheet bertingkat Raven04.css, yang akan Anda buat dalam latihan berikutnya.
   b. la memiliki sebuah elemen IMAGE kosong tepat sebelum masingmasing elemen STANZA. Anda akan menggunakan elemen IMAGE untuk menampilkan citra yang mengambang pada raven di awal masing-masing stanza.
   c. Bait terakhir dalam setiap stanza ditempatkan dalam elemen khusus yang dinamai LASTVERSE. Ini memungkinkan Anda memformat bait terakhir berbeda dengan bait yang lainnya (ia akan diratakan ke kanan bukannya ke kiri).

2. Gunakan perintah Save As pada editor teks untuk menyimpan dokumen ini pada hard disk dengan nama Raven04.xml.

Raven04.xml

```
<?xml version="1.0"?>
<!-- File Name: Raven04.xml -->
<?xml-stylesheet type="text/css"
href="Raven04.css"?>
<POEM>
<TITLE>The Raven</TITLE>
<AUTHOR>
   Edgar Allan Poe
   <AUTHOR-BIO>Edgar Allan Poe was an
American writer who lived from 1809 to
1849.</AUTHOR-BIO>
</AUTHOR>
<DATE>1845</DATE>
<IMAGE/>
<STANZA>
   <VERSE>Once upon a midnight dreary, while
I pondered, weak and weary,</VERSE>
   <VERSE>Over many a quaint and curious
volume of forgotten lore&#8212;</VERSE>
   <VERSE>While I nodded, nearly napping,
suddenly there came a tapping,</VERSE>
   <VERSE>As of some one gently rapping,
rapping at my chamber door.</VERSE>
   <VERSE>"'Tis some visitor," I muttered,
"tapping at my chamber door&#8212;</VERSE>
   <LASTVERSE>Only this, and nothing
more."</LASTVERSE>
</STANZA>
<IMAGE/>
```

```
<STANZA>
   <VERSE>Ah, distinctly I remember it was
in the bleak December,</VERSE>
   <VERSE>And each separate dying ember
wrought its ghost upon the floor.</VERSE>
   <VERSE>Eagerly I wished the
morrow;&#8212;vainly I had sought to
borrow</VERSE>
   <VERSE>From my books surcease of
sorrow&#8212;sorrow for the lost
Lenore&#8212;</VERSE>
   <VERSE>For the rare and radiant maiden
whom the angels name Lenore&#8212;</VERSE>
   <LASTVERSE>Nameless here for
evermore.</LASTVERSE>
</STANZA>
<IMAGE/>
<STANZA>
   <VERSE>And the silken sad uncertain
rustling of each purple curtain</VERSE>
   <VERSE>Thrilled me&#8212;filled me with
fantastic terrors never felt
before;</VERSE>
   <VERSE>So that now, to still the beating
of my heart, I stood repeating:</VERSE>
   <VERSE>"'Tis some visitor entreating
entrance at my chamber door&#8212;</VERSE>
   <VERSE>Some late visitor entreating
entrance at my chamber door;</VERSE>
   <LASTVERSE>This it is, and nothing
more."</LASTVERSE>
</STANZA>
<IMAGE/>
```

```
<STANZA>
   <VERSE>Presently my soul grew stronger;
hesitating then no longer,</VERSE>
   <VERSE>"Sir," said I, "or Madam, truly
your forgiveness I implore;</VERSE>
   <VERSE>But the fact is I was napping, and
so gently you came rapping,</VERSE>
   <VERSE>And so faintly you came tapping,
tapping at my chamber door,</VERSE>
   <VERSE>That I scarce was sure I heard
you"&#8212;here I opened wide the
door;&#8212;</VERSE>
   <LASTVERSE>Darkness there, and nothing
more.</LASTVERSE>
</STANZA>
</POEM>
```

Listing 12-12 .

Membuat Style sheet

1. Buka sebuah file teks kosong yang baru dalam editor teks Anda, dan ketikkan dalam style sheet bertingkat seperti tampak dalam Listing 12-13.

   Berikut ini beberapa hal yang perlu diingat tentang style sheet ini:

   a. Style sheet yang ada mendemonstrasikan hampir semua properti yang diberikan dalam bab ini.
   b. Saya telah menerangkan semua teknik dalam style sheet tersebut pada bagian sebelumnya dalam bab ini.
   c. File citra (RavShade.bmp) yang ditampilkan menggunakan elemen IMAGE adalah sama dengan citra yang ditampilkan oleh

versi sebelumnya dari dokumen Raven.: xml, hanya saja ia memiliki background berarsir yang sesuai dengan warna background dari elemen POEM.

d. Style sheet tersebut menyembunyikan isi elemen AUTHOR BIO dengan memberikan none pada properti displaynya.

2. Gunakan perintah Save As pada editor teks untuk menyimpan dokumen ini pada harddisk dengan nama Raven04.css.

Raven04.css

```
/* File Name: Raven04.css */
POEM
   {font-size:12pt;
    width:5.5in;
    padding:1em;
    border-width:1px;
    background-color:rgb(225,225,225)}
POEM, TITLE, AUTHOR, DATE, STANZA
   {display:block;
    margin-bottom:1em}
AUTHOR-BIO
   {display:none}
TITLE, AUTHOR, DATE
   {font-family:Arial,sans-serif;
    text-align:center}
DATE
   {font-style:italic}
TITLE
   {font-size:16pt;
    font-weight:bold;
    letter-spacing:.25em}
IMAGE
   {background-image:url(RavShade.bmp);
```

```
    background-repeat:no-repeat;
    background-position:center;
    width:89px;
    height:58px;
    float:left}
STANZA
   {color:navy;
    line-height:1.25em}
VERSE
   {display:block}
LASTVERSE
   {display:block;
    text-align:right}
```

Listing 12-13.

3. Tampilkan dokumen tersebut dengan membuka file Raven04.xml langsung dalam Internet Explorer 5:

```
Ravenven04.xml
```

C. PENUTUP

Ringkasan

Bab ini mencakup hampir seluruh properti CSS yang didukung Internet Explorer 5, dan merupakan bagian dari versi asli CSS, yang telah didefinisikan oleh World. Wide Web Consortium (W3C), dan dikenal sebagai Cascading Style Sheet Level 1 atau CSS1. W3C juga telah mendefinisikan versi yang lebih tinggi dari CSS, yang secara luas merupakan pengembangan CSS1, dan dikenal sebagai Cascading Style Sheet Level 2 atau CSS2. CSS2 hanya didukung sebagian saja oleh browser yang ada dan di luar cakupan buku ini. Anda bisa melihat spesifikasi W3C selengkapnya untuk CSS1 pada

http://www.w3c.org/TR/REC-CSS1 dan spesifikasi CSS2 di http://www.w3c.org!TRIREC- CSS2.

Contoh style sheet dalam Listing 7-1 dan contoh dokumen XML dalam Listing 12-2 merupakan salinan file yang Anda buat dalam latihan yang diberikan pada "Menampilkan Dokumen XML Menggunakan Style Sheet Bertingkat "

Karakter spasi kosong (spasi, tab, dan baris pemisah) memisahkan komponen CSS yang berbeda, seperti misalnya deklarasi individual dalam blok deklarasi tersebut. Cara saya memakai spasi kosong dalam buku ini hanya satu gaya yang memungkinkan. Anda bisa menggunakan spasi kosong dalam cara apa pun yang membantu Anda mengelola dan menjelaskan style sheet Anda sendiri. Sebagai contoh, Anda bisa menempatkan semua deklarasi yang dimiliki oleh aturan pada baris yang sama, ketimbang pada baris tersendiri seperti yang Anda lihat dalam contoh ini.

Pastikan Anda tidak menyisipkan koma di antara nama elemen dalam selektor kontekstual. Jika tidak, aturan yang ada akan diterapkan pada semua elemen (seperti diterangkan dalam bagian sebelumnya) ketimbang hanya pada elemen anak yang terakhir dicantumkan.

Jika browser tidak bisa menemukan file style sheet yang disebutkan dalam instruksi pemrosesan xml-stylesheet, ia akan menampilkan teks dokumen menggunakan seting propertinya (misalnya, seting font dan font-size yang ada). Jika sebuah dokumen XML tidak dikaitkan ke sebuah style sheet (yakni, dokumen tidak berisi instruksi pemrosesan xml-stylesheet), Internet Explorer 5 akan menampilkan sumber XML untuk dokumen tersebut ketimbang isinya. Susunan prioritas dalam daftar sebelumnya tidaklah kaku. Seting properti pada browser bisa saja didahulukan atas seting properti dalam style sheet yang

Anda sertakan pada dokumen XML. lni memungkinkan user dengan kebutuhan khusus untuk mengontrol semua pemformatan (misalnya, user dengan penglihatan yang kabur bisa menggunakan font ekstra-besar). Pada Internet Eksplorer 5, misalnya, user bisa memberikan pada browser seting properti yang diprioritaskan terhadap seting style sheet Anda dengan memilih perintah Internet Options dari menu Tools, klik tombol Accessibility dalam tab General pada kotak dialog Internet Options, dan pilih opsi yang sesuai.

Aturan di mana browser menggunakan seting properti style terakhir yang diprosesnya adalah berlawanan dengan aturan yang dipakai prosesor XML ketika ia menemukan multiatribut atau sejumlah deklarasi entitas. Lihatlah kembali dalam bab sebelumnya bahwa prosesor XML menggunakan atribut atau deklarasi entitas pertama dan mengabaikan selebihnya.

Spesifikasi CSS Menandakan bahwa properti display tidak diwarisi oleh elemen anak. lni betul sekali jika Anda memberikan seting block ke properti display elemen tersebut. Elemen anak secara efektif tidak mewarisi seting apa pun, karena saat Anda memberikan seting ini ke properti display induknya, Anda juga menyembunyikan semua elemen anak. Perhatikan juga bahwa elemen anak pada sebuah elemen inline juga akan menjadi inline jika mereka tidak punya nilai display, karena inline adalah seting default.

Jika Anda memberikan citra background dan juga warna background solid (menggunakan properti background-color), citra tersebut akan menutupi warna solid.

Pertanyaan

1. Sebutkan langkah dasar dalam menggunakan Style-Sheet Bertingkat?

2. Bagaimana cara menuliskan penggunaan multi-elemen dan multi-aturan dalam CSS?
3. Bagaimana cara menuliskan perintah untuk mengimport style sheet lain?
4. Sebutkan dan jelaskan cara menyeting properti display?
5. Jelaskan perbedaan Dont Family, font-size, font-style, font-wight, font-variant?
6. Jelaskan perbedaan Background-color, background-image, background-repeat, dan background-position?
7. Jelaskan perbedaan letter-spacing, vertical-align, text-align, text-indent, line-height, text-transform, dan text-decoration?

DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.

9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 13

Membuat Dokumen XML yang Valid

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 13 akan di bahas Membuat Dokumen XML yang Valid, yang mencakup diantaranya: Kriteria Dasar sebuah Dokumen XML yang Valid, Keuntungan Membuat Dokumen XML yang Valid, Menambahkan DTD, Bentuk DTD, Membuat DTD, Mendeklarasikan Tipe Elemen, Bentuk Deklarasi Tipe Elemen, Spesifikasi isi Elemen, Menyebutkan isi Elemen, Penyebutan isi Campuran, Mendeklarasikan Atribut, Bentuk Deklarasi Daftar, Tipe Atribut.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan cara cara Kriteria Dasar sebuah Dokumen XML yang Valid, Keuntungan Membuat Dokumen XML yang Valid, Menambahkan DTD, Bentuk DTD, Membuat DTD, Mendeklarasikan Tipe Elemen, Bentuk Deklarasi Tipe Elemen, Spesifikasi isi Elemen, Menyebutkan isi Elemen, Penyebutan isi Campuran, Mendeklarasikan Atribut, Bentuk Deklarasi Daftar, Tipe Atribut.

B. PENYAJIAN

Membuat Dokumen XML yang Valid

Dokumen XML yang valid memenuhi serangkaian kriteria yang lebih ketat daripada sekedar dokumen yang well-formed yang telah Anda pelajari pada

bab-bab sebelumnya. Dalam bab ini, pertama Anda akan mempelajari persyaratan dasar bagi sebuah dokumen XML yang valid, dan menjela jahi keuntungan menggunakan dokumen yang valid. Kemudian, Anda akan belajar cara menambahkan deklarasi tipe dokumen yang diperlukan dalam semua dokumen XML yang valid. Saya nanti akan menyajikan instruksi detil untuk mendefinisikan elemen dan atribut dalam sebuah dokumen yang valid, dan akan saya akhiri dengan sebuah latihan di mana Anda akan mengubah dokumen contoh yang dibuat dalam Bab ini menjadi sebuah dokumen yang valid. Dalam bab berikutnya, Anda akan belajar cara mendeklarasikan dan menggunakan berbagai entitas, yang bersifat opsional namun merupakan komponen berguna pada dokumen XML yang valid.

Kriteria Dasar sebuah Dokumen XML yang Valid

Setiap dokumen XML harus dibuat well-formed, dalam arti memenuhi persyaratan minimum untuk sebuah dokumen XML. Jika sebuah dokumen tidak well-formed, tidak bisa dianggap sebagai sebuah dokumen XML.

Sebuah dokumen XML yang well-formed bisa juga menjadi valid. Sebuah dokumen XML yang valid adalah dokumen well-formed yang memenuhi dua persyaratan selanjutnya:

1. Prolog pada dokumen harus memasukkan deklarasi tipe dokumen yang sesuai, yang berisi sebuah definisi tipe dokumen (DTD document type definition) yang mendefinisikan struktur dokumen.
2. Dokumen selebihnya harus tunduk pada struktur yang didefinisikan dalam DTD.

Pada bagian berikut bab ini, Anda akan belajar cara membuat dokumen yang

tunduk pada dua persyaratan umum ini.

Keuntungan Membuat Dokumen XML yang Valid

Membuat sebuah dokumen XML yang valid tampaknya tidak begitu meng-ganggu. Anda pertama hams sepenuhnya mendefini sikan struktur dokumen dalam sebuah DTD, dan kemudian membuat dokumen itu sendiri, ikuti semua spesifikasi DTD. Ini tampaknya lebih mudah hanya dengan segera menambahkan elemen apa pun dan atribut yang Anda perlukan, seperti yang Anda lakukan dalam contoh dokumen well-formed pada ba b terdahulu.

Jika Anda ingin memastikan bahwa dokumen Anda sesuai dengan struktur spesifik, atau serangkaian standar, pencantuman pada sebuah DID yang mendefinisikan bahwa struktur tersebut memungkinkan sebuah prosesor XML (seperti yang ada pada Microsoft Internet Explorer 5) untuk memeriksa apakah dokumen Anda sesuai dengan struktur tersebut. Dengan kata lain, DTD memberikan sebuah cetak biru standar pada prosesor untuk pemeriksaan validitas dokumen, ia bisa memaksakan struktur yang diinginkan dan menjamin bahwa dokumen Anda memenuhi standar yang diinginkan. Jika ada bagian dokumen yang tidak memenuhi spesifikasi DTD, prosesor dapat menampilkan sebuah pesan kesalahan sehingga Anda bisa mengedit dokumen tersebut dan menjadikannya sesuai dengan aturan.

Menjadikan dokumen-dokumen XML valid sangat berguna sekali untuk memastikan keseragaman antardokumen yang serupa. Pada kenyataannya, standar XML mendefinisikan sebuah DTD sebagai "tata-bahasa untuk sekelompok dokumen".

Bayangkanlah, misalnya, sebuah perusahaan penerbit Web yang

mengharapkan semua editornya membuat dokumen XML yang tunduk pada sebuah struktur bersama. Pembuatan DTD tunggal dan memasukkannya dalam semua dokumen bisa menjamin bahwa dokumen ini tunduk pada struktur yang diinginkan, dan para editor tidak menambahkan elemen baru yang saling berbeda, menempatkan informasi pada susunan yang keliru, memberikan tipe data yang salah pada atribut, dan seterusnya. Tentu saja, dokumen harus dijalankan melalui sebuah prosesor yang memeriksa validitasnya.

Memasukkan DTD dan memeriksa validitas adalah sangat penting, khususnya jika dokumen tersebut akan diproses oleh perangkat lunak yang sama yang menginginkan struktur dokumen tertentu. Bila semua user perangkat lunak memasukkan sebuah DTD bersama yang sesuai dalam dokumen XML mereka, dan jika dokumen tersebut diperiksa validitasnya, para user bisa memastikan bahwa dokumen mereka akan dikenali oleh perangkat lunak pemroses. Sebagai contoh, jika sebuah kelompok ahli matematika sedang membuat dokumen matematis yang akan ditampilkan menggunakan program khusus, mereka bisa memasukkan dalam dokumen mereka sebuah DTD bersama yang mendefinisikan strnktur, elemen, atribut dan fitur lain yang diinginkan.

Pada kenyataannya, sebagian besar aplikasi XML 'nyata', seperti misalnya MathML, terdiri atas sebuah DTD standar yang dimasukkan oleh semua user aplikasi ke dalam dokumen XML mereka. Dengan demikian pemeriksaan validitas dokumen akan memastikan bahwa mereka tunduk pada struktur aplikasi dan akan dikenali oleh berbagai perangkat lunak yang dirancang untuk aplikasi itu.

TIP Jika Anda membuka sebuah dokumen (baik dengan atau tanpa style sheet) langsung dalam Internet Explorer 5, prosesor Internet Explorer 5 akan memeriksa kerapian seluruh dokumen (termasuk deklarasi tipe dokumen, jika dimasukkan) dan menampilkan sebuah kesalahan fatal untuk pelanggaran yang ditemukannya. Bagaimanapun juga, prosesor Internet Explorer 5 tidak memeriksa validitas dokumen. Untuk menguji validitas dokumen, Anda bisa menggunakan script pemeriksa validitas yang diberikan dalam "Pemeriksaan Validitas Dokumen". Anda mungkin ingin membaca instruksi yang diberikan dalam bagian itu sekarang, dengan demikian Anda bisa mulai memeriksa validitas dokumen XML yang Anda buat.

Menambahkan DTD

Deklarasi tipe dokumen adalah sebuah blok markup XML yang harus Anda tambahkan pada prolog dokumen XML yang valid. la bisa di mana saja dalam prolog di luar markup lainnya setelah deklarasi XML (ingatlah jika Anda memasukkan deklarasi XML, ia harus ada pada awal dokumen).

Deklarasi tipe dokumen mendefinisikan struktur dokumen. Jika Anda

membuka sebuah dokumen tanpa deklarasi tipe dokumen dalam Internet Explorer 5, prosesor Internet Explorer 5 hanya memeriksa apakah dokumen itu sudah well-formed. Jika Anda membuka sebuah dokumen dengan deklarasi tipe dokumen dalam Internet Explorer 5, prosesor akan memeriksa validitas dokumen di samping kerapiannya, dan dokumen Anda harus tunduk pada semua deklarasi dalam deklarasi tipe dokumen. Anda, misalnya, tidak akan bisa memasukkan suatu elemen atau atribut dalam dokumen tersebut yang belum Anda deklarasikan dalam deklarasi tipe dokumen. Setiap elemen dan atribut yang Anda masukkan harus sesuai dengan spesifikasi (misalnya isi yang diperke- nankan pada suatu elemen atau tipe yang diizinkan pada suatu atribut) yang dinyatakan dalam deklarasi bersangkutan.

Bentuk DTD

Deklarasi tipe dokumen memiliki bentuk umum sebagai berikut:

```
<! DOCTYPE Name DTD>
```

Di sini, Name menyat akan nama elemen dokumen. Nama elemen dokumen sesungguhnya harus sesuai dengan nama yang Anda masukkan di sini (untuk penjelasan ketentuan yang mengatur nama-nama elemen, lihat kembali "Anatomi sebuah Elemen "). Misalnya, jika Anda sedang membuat sebuah deklarasi tipe dokumen untuk dokumen contoh yang tampak dalam bagian sebelumnya, Anda akan menggunakan nama INVENTORY:

```
<! DOCTYPE INVENTORY DTD>
```

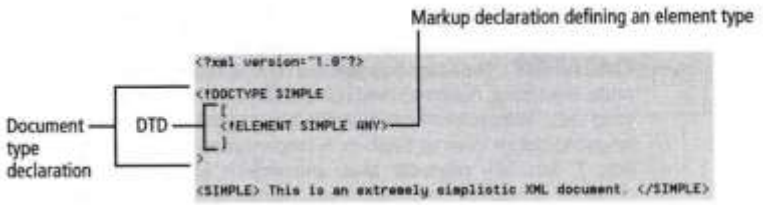
Cini masih belum merupakan deklarasi tipe dokumen yang utuh, DTD masih harus diganti dengari. isi sesungguhnya.)

DTD adalah deklarasi tipe dokumen, berisi deklarasi yang mendefinisikan berbagai elemen, atribut dan fitur lain dari dokumen. Pada bagian berikut

Anda akan lihat bentuknya.

Membuat DTD

DTD berisi sebuah karakter kurung persegi kiri CD, diikuti dengan serangkaian deklarasi markup, dikuti dengan karakter kurung persegi kanan 0). Deklarasi markup menjelaskan struktur logika dokumen; yakni; ia mendefinisikan elemen, atribut, dan fitur lain dari dokumen. Berikut ini sebuah dokumen XML valid yang lengkap berisi sebuah DTD dengan sebuah deklarasi markup tunggal yang mendefinisikan satu tipe elemen dokumen, SIMPLE:



DTD dalam contoh dokumen ini menyatakan bahwa dokumen tersebut dapat berisi hanya elemen dari tipe SIMPLE (yakni tipe elemen satu-satunya yang didefinisikan), dan elemen SIMPLE bisa memiliki sembarang tipe isi yang memungkinkan (kata kunci ANY).

DTD bisa berisi tipe deklarasi markup berikut ini:

1. Deklarasi tipe elemen. Deklarasi ini mendefinisikan tipe-tipe elemen yang bisa dimuat dokumen, serta isi dan susunan elemen. Deklarasi tipe elemen dijelaskan dalam bagian selanjutnya.
2. Deklarasi daftar atribut. Setiap deklarasi daftar atribut mendefinisikan nama atribut yang bisa dipakai bersama tipe elemen khusus, juga dengan tipe data dan nilai default dari atribut ini. Saya akan membahas deklarasi

ini nanti.

3. Deklarasi Entitas. Anda bisa menggunakan berbagai entitas untuk menyimpan blok teks yang paling sering dipakai, atau untuk memasukkan data non-XML dalam dokumen Anda.
4. Deklarasi notasi. Notasi menjelaskan format data, atau menentukan program yang dipakai untuk memproses format tertentu.
5. Instruksi pemrosesan. Saya telah menerangkannya dalam bagian "Menggunakan Instruksi".
6. Komentar Saya telah membahasnya pada bagian "Menyisipkan Komentar".
7. Acuan entitas parameter. Sebagian item di atas pada daftar ini dapat dimuat dalam entitas parameter dan disisipkan melalui acuan entitas parameter. Pernyataan ini tidak akan benar-benar dimengerti hingga Anda membaca, namun saya telah memasukkannya di sini untuk keutuhannya.

Mendeklarasikan Tipe Elemen

Dalam sebuah dokumen XML yang valid, Anda harus secara eksplisit mendeklarasikan tipe setiap elemen yang Anda pakai dalam dokumen tersebut pada sebuah deklarasi tipe elemen dalam DTD. Deklarasi tipe elemen menandai nama tipe elemen, dan isi yang diperkenankan untuk elemen (seringkali menyebutkan susunan di mana terdapat elemen anak). Ingat pula, deklarasi tipe elemen dalam DTD mirip dengan sebuah skema database memetakan seluruh struktur logika pada dokumen. Yaitu, deklarasi tipe elemen menandai tipe elemen yang dimuat dokumen, susunan elemen, dan spesifikasi isi elemen.

Bentuk Deklarasi Tipe Elemen

Deklarasi tipe elemen mempunyai bentuk berikut:

```
<! ELEMENT Name contentspec>
```

Di sini, Name adalah nama tipe elemen yang akan dideklarasikan. (Untuk melihat kembali nama-nama elemen yang sah, lihat "Anatomi Elemen ". Dan contentspec adalah spesifikasi isi, yang mendefinisikan apa yang dimuat elemen. Bagian berikutnya menjelaskan perbedaan tipe- tipe spesifikasi isi yang Anda pakai.

Berikut ini adalah sebuah deklarasi tipe elemen bernama TITLE, yang diizinkan hanya memuat data karakter saja (tidak ada elemen anak yang diperkenankan):

```
<! ELEMENT TITLE(ffPCDATAl>
```

Dan yang ini adalah deklarasi untuk sebuah tipe elemen bernama GENERAL, yang bisa memuat sembarang tipe isi:

```
<! ELEMENT GENERALANY>
```

Sebagai contoh akhir, berikut ini dokumen XML selengkapnya dengan dua tipe elemen. Deklarasi tipe elemen COLLECTION menandai bahwa ia bisa berisi satu atau lebih elemen CD, dan deklarasi tipe elemen CD menyebutkan bahwa ia hanya bisa berisi data karakter. Perhatikan dokumen patuh pada deklarasi ini dan karenanya adalah valid:

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

<!DOCTYPE COLLECTION
   [
   <!ELEMENT COLLECTION (CD)+>
   <!ELEMENT CD (#PCDATA)>
   <!-- You can also insert a comment in a DTD. -->
   ]
>

<COLLECTION>
   <CD>Mozart Violin Concertos 1, 2, and 3</CD>
   <CD>Telemann Trumpet Concertos</CD>
   <CD>Handel Concerti Grossi Op. 3</CD>
</COLLECTION>
```

Spesifikasi isi Elemen

Anda bisa menyebutkan isi sebuah elemen yakni, mengisi bagian contentspec pada deklarasi tipe elemen dalam empat cara:

1. Isi EMP'IY. Anda menggun akan kata kunci EMPTY untuk menandai bahwa elemen tersebut harus kosong yaitu, tidak bisa memiliki isi. Inilah contohnya:

   ```
   <! ELEMENT IMAGE EMPTY>
   ```

   Berikut ini akan menjadi elemen IMAGE valid yang bisa Anda masukkan dalam dokumen:

   ```
   <IMAGE> </IMAGE>
   <IMAGEI>
   ```

2. Isi ANY. Anda menggunakan kata-kunci ANY untuk menandai bahwa elemen tersebut bisa memiliki sembarang tipe isi yang sah. Yakni, elemen tipe ini bisa berisi nol atau lebih elemen anak, dalam sembarang susunan atau sejumlah pengulangan, dengan atau tanpa data karakter yang tersebar. Ini merupakan spesifikasi isi yang paling longgar, dan membuat suatu tipe elemen tanpa batasan isi. Berikut contoh deklarasinya:

```
<! ELEMENT MISCANY>
```

3. Isi elemen (juga dikenal sebagai isi anak). Dengan tipe spesifikasi isi ini, elemen bisa berisi elemen anak, namun tidak bisa langsung berisi data karakter. Saya akan jelaskan opsi ini dalam bagian selanjutnya.
4. Isi campuran. Dengan tipe isi spesifikasi ini, elemen bisa berisi sembarang jumlah data karakter, secara opsional tersebar dengan elemen anak pada tipe yang disebutkan.

Menyebutkan lsi Elemen

Jika sebuah elemen memiliki isi, ia bisa langsung berisi elemen anak yang disebutkan saja, dan bukan data karakter. Dalam dokumen tersebut, Anda bisa saja memisahkan elemen anak dengan karakter spasi kosong untuk meningkatkan keterbacaannya, namun prosesor akan mengabaikan karakter tersebut dan tidak akan mengirimkannya ke aplikasi.

Perhatikan contoh dokumen XML berikut, yang menggambarkan sebuah buku tunggal:

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

<!DOCTYPE BOOK
   [
   <!ELEMENT BOOK (TITLE, AUTHOR)>
   <!ELEMENT TITLE (#PCDATA)>
   <!ELEMENT AUTHOR (#PCDATA)>
   ]
>
<BOOK>

   <TITLE>The Scarlet Letter</TITLE>

   <AUTHOR>Nathaniel Hawthorne</AUTHOR>

</BOOK>
```

Dalam dokumen ini, tipe elemen BOOK dideklarasikan untuk memiliki isi elemen. (TITLE, AUTHOR) yang mengikuti nama elemen dalam deklarasi dikenal sebagai model isi. Model ini menyatakan tipe elemen anak yang diperkenankan dan susunannya. Dalam contoh ini, model ini menyatakan bahwa elemen BOOK harus tepat memiliki satu elemen anak bernama TITLE diikuti dengan persis satu elemen anak AUTHOR. Dalam dokumen, prosesor akan mengabaikan tiga baris kosong yang dipakai untuk memi- sahkan elemen anak dalam elemen BOOK.

Model isi bisa memiliki juga dua bentuk dasar berikut ini:

1. Sekuen. Bentuk sekuen pada model isi menandakan bahwa elemen harus memiliki sebuah sekuen elemen anak yang spesifik. Anda pisahkan nama tipe elemen anak dengan koma. Misalnya, DTD berikut ini menandakan bahwa elemen dokumen MOUNTAIN harus memiliki sebuah elemen anak NAME, diikuti dengan satu elemen anak HEIGHT, diikuti dengan satu elemen anak STATE:

```
<!DOCTYPE MOUNTAIN
   [
   <!ELEMENT MOUNTAIN (NAME, HEIGHT, STATE)>
   <!ELEMENT NAME (#PCDATA)>
   <!ELEMENT HEIGHT (#PCDATA)>
   <!ELEMENT STATE (#PCDATA)>
   ]
>
```

Maka, elemen dokumen berikut akan menjadi valid:

```
<MOUNTAIN>
   <NAME>Wheeler</NAME>
   <HEIGHT>13161</HEIGHT>
   <STATE>New Mexico</STATE>
</MOUNTAIN>
```

Elemen dokumen berikut akan menjadi valid karena susunan elemen anak tidak dideklarasikan:

```
<MOUNTAIN> <!-- Invalid element! -->
   <STATE>New Mexico</STATE>
   <NAME>Wheeler</NAME>
   <HEIGHT>13161</HEIGHT>
</MOUNTAIN>
```

Membuang atau memasukkan tipe elemen anak yang sama lebih dari sekali juga akan menjadi tidak sah atau invalid. Seperti yang bisa Anda lihat, ini merupakan tipe deklarasi yang paling ketat.

2. Pilihan. Bentuk pilihan model isi menandakan bahwa elemen bisa memiliki salah satu dari elemen anak yang memungkinkan, yang dipisahkan menggunakan karakter I. Misalnya, DTD berikut menyatakan bahwa elemen FILM bisa berisi satu elemen anak STAR, atau satu NARRATOR, atau satu elemen anak INSTRUCTOR:

```
<!DOCTYPE FILM
   [
   <!ELEMENT FILM (STAR | NARRATOR | INSTRUCTOR)>
   <!ELEMENT STAR (#PCDATA)>
   <!ELEMENT NARRATOR (#PCDATA)>
   <!ELEMENT INSTRUCTOR (#PCDATA)>
   ]
>
```

Dengan demikian, elemen dokumen berikut akan menjadi valid:

```
<FILM>
<STAR>Robert Redford</STAR>
<! FILM>
```

begitu pula elemen ini:

```
<FILM>
<NARRATOR>Sir Gregory Parsloe</NARRATOR>
<! FILM>
```

dan juga yang ini:

```
<FILM>
<INSTRUCTOR>Galahad Threepwood</INSTRUCTOR>
<! FILM>
```

Elemen dokumen berikut akan menjadi tidak valid karena Anda bisa memasukkan hanya salah satu tipe elemen anak:

```
< FILM> <! -- Invalidelement! - - >
< NARRATOR>Sir Gregory Parsloe</NARRATOR>
< INSTRUCTOR>Galahad Threepwood</INSTRUCTOR>
<! FILM>
```

Anda juga bisa mengubah bentuk model isi ini menggunakan karakter tanda tanya (?), tanda tambah (+), dan asteris (*), yang memiliki arti seperti dalam tabel berikut ini:

Karakter	Pengertian
?	Nol atau lebih item pengantar
+	Satu atau lebih item pengantar

* Nol atau lebih item pengantar

Sebagai contoh, maksud deklarasi berikut bahwa Anda bisa memasukkan satu atau lebih elemen anak NAME, dan bahwa elemen anak HEIGHT bersifat opsional:

```
<!ELEMEN MOUNTAIN (NAME+, HEIGHT? STATE)>
```

Dengan demikian, elemen berikut menjadi legal:

```
<MOUNTAIN>
<NAME>Pueblo Peak</NAME>
<NAME>Taos Mountain</NAME>
<STATE>New Mexico</STATE>
</MOUNTAIN>
```

Seperti contoh lainnya, maksud deklarasi berikut bahwa Anda bisa memasukkan nol atau lebih elemen anak STAR, atau satu elemen anak NARRATOR, atau satu elemen anak INSTRUCTOR:

```
<! ELEMENT FILM (STAR* INARRATORI
INSTRUCTOR)>
```

Dengan demikian, masing-masing elemen berikut menjadi legal:

```
<FILM>
<STAR>Tom Hanks</STAR>
<STAR>Meg Ryan</STAR>
</FILM>
<FILM>
<NARRATOR>Sir Gregory Parsloe</NARRATOR>
</FILM>
<FILM>
```

Anda juga bisa menggunakan karakter? +, atau * untuk mengubah seluruh model ini dengan mengganti karakter setelah kurung tutup. Misalnya, deklarasi berikut memungkinkan Anda memasukkan satu atau lebih elemen anak dari ketiga tipe, dalam sembarang susunan:

```
<!ELEMENT FILM (STAR INARRATORI
INSTRUCTOR)+>
```

Deklarasi ini menjadikan elemen berikut legal:

```
<FILM>
<NARRATOR>Bertram Wooster</NARRATOR>
<STAR>Sean Connery</STAR>
<NARRATOR>Plug Basham</NARRATOR>
</FILM>
<FILM>
<STAR>Sean Connery</STAR>
<STAR>Meg Ryan</STAR>
</FILM>
<FILM>
<INSTRUCTOR>Stinker Pike</INSTRUCTOR>
</FILM>
```

Terakhir, Anda bisa membentuk model isi yang lebih kompleks dengan penyarangan sebuah model isi pilihan dalam model sekuen, atau sebuah model sekuen dalam sebuah model pilihan. Sebagai contoh, DTD berikut menyebutkan bahwa elemen dokumen FILM harus memiliki satu elemen anak TITLE; diikuti dengan satu elemen anak CLASS; diikuti satu elemen anak, STAR, NARRATOR, atau INSTRUCTOR:

```
<!DOCTYPEFILM
[
<!ELEMENT FILM (TITLE, CLASS, (STAR |
NARRATOR | INSTRUCTOR))>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT CLASS (#PCDATA)>
<!ELEMENTSTAR (#PCDATA)>
<!ELEMENTNARRATOR(#PCDATA)>
<!ELEMENT INSTRUCTOR(#PCDATA)>
]>
```

Menurut DTD ini, elemen dokumen berikut adalah legal

```
<FILM>
<TITLE>The Net</TITLE>
<CLASS>fictional</CLASS>
<STAR>Sandra Bullock</STAR>
</FILM>
```

seperti yang satu ini:

```
<FILM>
<TITLE>How to Use XML</TITLE>
<CLASS>instructional</CLASS>
<INSTRUCTOR> Penny Donaldson</INSTRUCTOR>
</FILM>
```

Penyebutan lsi Campuran

Jika sebuah elemen memiliki isi campuran, ia bisa berisi data karakter. Jika Anda menyebutkan satu atau lebih tipe elemen anak dalam deklarasi tersebut, ia bisa berisi sembarang elemen anak dalam sembarang susunan, dan dengan sembarang pengulangan (nol atau lebih). Dengan kata lain, dengan isi campuran Anda bisa membatasi tipe elemen anak, namun Anda tidak bisa membatasi susunan atau jumlah pengulangan tipe elemen anak, Anda juga tidak bisa mewajibkan elemen anak tertentu.

Untuk mendeklarasikan sebuah tipe elemen pada isi campuran, Anda bisa menggunakan pula dua bentuk model berikut:

1. Hanya data karakter. Untuk mendeklarasikan sebuah tipe elemen yang bisa berisi data karakter saja, gunakan model isi (#PCDATA). Deklarasi berikut, misalnya, memungkinkan sebuah elemen SUBTITLE untuk memuat data karakter saja:

   ```
   <!ELEMENT SUBTITLE (#PCDATA)>
   ```

Dua elemen berikut valid sesuai dengan deklarasi ini:

```
<SUBTITLE>A New Approach </SUBTITLE>
<SUBTITLE> </SUBTITLE>
```

Perhatikan dalam contoh kedua bahwa sebuah elemen yang dideklarasikan untuk memuat data karakter bisa berisi nol karakter-yakni, Anda bisa membiarkannya kosong. (Dengan bentuk mode l ini, istilah isi campuran secara teknis tidak cocok.)

2. Data karakter plus elemen anak opsional. Untuk mendeklarasikan sebuah tipe elemen yang bisa memuat data karakter ditambah nol atau lebih elemen anak, tampilkan setiap tipe elemen mengikuti #PCDATA dalam model isi, pisahkan item-item nya dengan karakter I, dan sisipkan sebuah asteris (*) mengikuti seluruh model isi. Setiap nama elemen bisa muncul hanya sekali dalam model isi tersebut. Misalnya, deklarasi berikut memungkinkan sebuah elemen TITLE memuat data karakter plus nol atau beberapa elemen anak SUBTITLE:

```
<!ELEMENT TITLE (JfPCDATA I SUBTITLE) *>
```

Berikut ini adalah elemen TITLE yang valid, sesuai dengan dengan deklarasi ini:

```
<TITLE>
Moby - Dick
<SUBTITLE>Or, the Whale</SUBTITLE>
</TITLE>
<TITLE>
<SUBTITLE>Or, the Whale</SUBTITLE>
Moby- Dick
</TITLE>
<TITLE>
Moby - Dick
</TITLE>
```

```
<TITLE>
<SUBTITLE>Or, the Whale</SUBTITLE>
<SUBTITLE>Another Subtitle</SUBTITLE>
</TITLE>
<TITLE> </TITLE>
```

Mendeklarasikan Atribut

Oalam sebuah dokumen XML yang valid, Anda juga harus mendeklarasikan secara eksplisit semua atribut yang ingin Anda gunakan dengan elemen dokumen. Anda mendefinisikan semua atribut yang berhubungan dengan elemen tertentu menggunakan sebuah tipe markup OTO yang dikenal dengan deklarasi daftar atribut. Oeklarasi ini melakukan hal berikut:

1. Mendefinisikan nama-nama atribut yang berhubungan dengan elemen tersebut. Oalam sebuah dokumen yang valid, Anda bisa memasukan hanya dalam sebuah elemen tag-awal semua atribut yang telah Anda definisikan untuk elemen itu.
2. Menyebutkan ripe data pada masing masing atribut.
3. Menyebutkan bagi masing masing atribut apakah atribut itu dibutuhkan. Jika atribut itu tidak diperlukan, deklarasi daftar atribut juga menandai prosesor apa yang harus melakukan jika atribut tersebut dibuang. (Oeklarasi tersebut bisa, misalnya, memberikan nilai atribut default yang akan akan dipakai prosesor.)

Bentuk Deklarasi Daftar-Atribut

Deklarasi daftar-atribut berbentuk sebagai berikut:

```
<! ATTLIST Name AttDefs>
```

Di sini, Name adalah nama tipe elemen yang berhubungan dengan atribut atau

atribut-atribut. AttDefs adalah serangkaian atau lebih definisi atribut, masing-masing mendefinisikan satu atribut.

```
Definisi atribut berbentuk:
Name AttType DefaultDecl
```

Di sini, Name adalah nama atribut (untuk melihat kembali aturan penamaan atribut yang legal, lihat "). AttType adalah tipe atribut, yakni semacam nilai yang bisa diberikan pada atribut (saya akan jelaskan tipe atribut dalam bagian berikutnya). Dan DefaultDecl adalah deklarasi default, yang menandai apakah atribut ter- sebut dibutuhkan dan memberikan informasi lain.

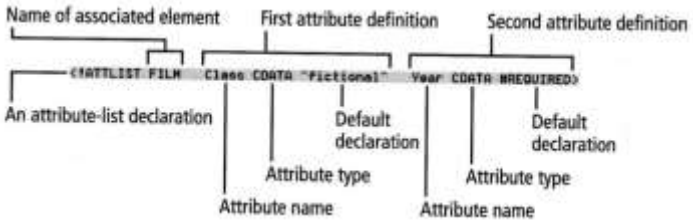
Katakanlah Anda telah mendeklarasikan sebuah tipe elemen bernama FILM seperti ini:

```
<!ELEMENT FILM (TITLE, (STAR I NARRATOR I
INSTRUCTOR))>
```

Berikut ini contoh sebuah deklarasi daftar-atribut yang mendeklarasikan dua atribut-bernama Class dan Year-untuk elemen FILM.

```
<!ATTLIST FILM Class CDATA"fictional" Year
CDATA #REQUIRED>
```

Inilah bagian deklarasi yang berbeda:



Anda bisa memberikan atribut Class sembarang potongan string yang legal (kata-kunci CDATA); jika Anda mengabaikan atribut dari elemen tertentu, ia akan otomatis diberikan nilai default "fictional". Anda bisa memberikan atribut Year sembarang string kutipan yang legal; atribut ini harus diberikan

sebuah nilai pada setiap elemen FILM (kata-kunci #REQUIRED), dan karenanya tidak memiliki nilai default.

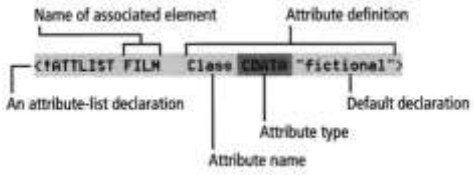
Dokumen XML berikut ini memasukkan deklarasi daftar-atribut serta FILM:

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

<!DOCTYPE FILM
   [
   <!ELEMENT FILM (TITLE, (STAR | NARRATOR | INSTRUCTOR))>
   <!ATTLIST FILM   Class CDATA "fictional"   Year CDATA #REQUIRED>
   <!ELEMENT TITLE (#PCDATA)>
   <!ELEMENT STAR (#PCDATA)>
   <!ELEMENT NARRATOR (#PCDATA)>
   <!ELEMENT INSTRUCTOR (#PCDATA)>
   ]
>

<FILM Year="1948">
   <TITLE>The Morning After</TITLE>
   <STAR>Morgan Attenbury</STAR>
</FILM>
```

Dalam elemen FILM, atribut Year diberikan nilai "1948". Atribut Class diabaikan; karena atribut ini memiliki sebuah nilai default ("fictional"), ini diberikan di mana nilai default hanya jika Anda telah memasukkan atribut dan mengetikkan nilainya.

Tipe Atribut

Tipe atribut adalah komponen kedua yang dibutuhkan pada sebuah definisi atribut. la menyatakan jenis nilai yang Anda berikan ke atribut dalam dokumen.



Anda bisa memberikan tipe atribut dengan tiga cara berbeda:

1. Tipe string. Atribut tipe string bisa diberikan dengan sembarang string kutipan (juga dikenal dengan literal) yang tunduk pada aturan umum yang dijelaskan pada bagian "Aturan untuk Nilai Atribut yang Legal". Anda mendeklarasikan sebuah atribut tipe string meng- gunakan kata-kunci CDATA, seperti dalam definisi atribut pada Class dalam contoh berikut:

   ```
   <!ATTLIST FILM Class CDATA"fictional">
   ```
2. Tipe bertanda. Nilai-nilai yang Anda berikan ke atribut tipe bertanda dibatasi dengan sejumlah cara, seperti yang akan saya jelaskan dalam bagian selanjutnya.
3. Tipe terhitung. Anda bisa berikan sebuah atribut tipe terhitung salah satu daftar nilai yang telah disebutkan.

Penyebutan Tipe Bertoken

Seperti nilai atribut, nilai yang Anda berikan ke tipe bertoken harus berupa string kutipan yang tunduk atruran umum, seperti dijelaskan dalam bagian "Aturan Nilai Atribut yang Legal ".

Selain itu, nilai tersebut harus tunduk pada batasan khusus yang Anda sebutkan dalam definisi atribut menggunakan kata-kunci yang sesuai. Sebagai contoh, dalam contoh dokumen XML berikut, atribut StockCode didefinisikan sebagai tipe bertanda menggunakan kata-kunci ID. (ID hanya salah satu kata kunci yang bisa Anda pakai untuk mendeklarasikan tipe yang ditandai.) Kata kunci ini berarti bahwa untuk masing-masing elemen atribut harus diberikan sebuah nilai yang unik (misalnya, pemberian kode stok “5021” ke dua elemen ITEM akan menjadikannya tidak valid).

```
<?xml versi on="l.O"?>
<!DOCTYPE INVENTORY
```

```
[
<!ELEMENT INVENTORY (ITEM)*l>
<!ELEMENT ITEM (#PCDATA)>
<!ATTLIST ITEM StockCode ID#REQUIRED>
]
>
<INVENTORY>
<! - - Each ITEM must have a different
StockCode value. - - >
<ITEM StockCode="S02l">Peach TeaPot</ITEM>
<ITEM StockCode="S034">ElectricCoffee
Grinder</ITEM>
< TEM Stock Code="S086">Candy
Thermometer</ITEM >
</INVENTORY>
```

Berikut ini daftar selengkapnya kata-kunci yang bisa Anda pakai untuk mendefinisikan atribut tipe bertanda dan batasan yang mereka kenakan pada nilai atribut:

1. ID. Dalam setiap elemen atribut harus memiliki sebuah nilai unik. Nilai tersebut harus dimulai dengan sebuah huruf atau garis-bawah (_) diikuti dengan nol atau sejumlah huruf, digit, titik, tanda kurang (-), atau beberapa garis-bawah. la bisa juga berisi sebuah tanda bagi (:) kecuali dalam posisi karakter pertama. Tipe elemen tertentu bisa hanya memiliki satu atribut tipe ID, dan deklarasi default atribut harus berupa #REQUIRED atau #IMPLIED (dijelaskan nanti dalam bab ini). Anda bisa melihat contoh tipe atribut ini dalam dokumen INVENTORY yang diberikan di atas.
2. IDREF. Nilai atribut harus sesuai dengan nilai beberapa atribut tipe dalam sebuah elemen dalam dokumen. Dengan kata lain, tipe atribut ini

mengacu pada pengenal unik atribut lain. Misalnya, Anda bisa menambahkan sebuah atribut IDREF yang dinamai GoesWith untuk elemen ITEM:

```
<!ELEMENT ITEM(#PCDATA>
<!ATTLIST ITEM StockCode ID#REQUIREDGoesWith
ID REF#IMPLIED>
```

Anda kemudian bisa menggunakan atribut ini untuk mengacu ke ele- men ITEM lain, seperti ditunjukkan di sini:

```
<ITEM StockCode="S034">Electric Coffee
Grinder</ITEM>
<ITEM StockCode="S047"
GoesWith="S034">Coffee Grinder Brush
</ITEM>
```

3. IDREFS. Tipe atribut ini persis seperti tipe IDREF, hanya saja nilainya bisa memasukkan sejumlah acuan ke sejumlah pengenal dipisahkan dengan karakter spasi kosong semua dalam string kutipan. Misalnya, jika Anda memberikan atribut GoesWith tipe IDREFS seperti ini:

```
<!ATTLIST ITEM StockCode ID #REQUIRED
GoesWith IDREFS #IMPLIED>
```

Anda bisa menggunakannya untuk menunjuk ke sejumlah elemen lain:

```
<ITEM StockCode="S034">Elctri Coffee
Grinder</ITEM>
<ITEM StockCode="S039">
1-pound Breakfast Blend Coffee Beans
</ITEM>
<ITEM StockCode="S047" GoesWith=" S034 5039"
>Coffee Grinder Brush
</ITEM>
```

4. ENTITY. Nilai atribut harus sesuai dengan nama pada entitas tak-terurai yang dideklarasikan dalam DTD. Sebuah entitas tak-terurai meng- acu

pada file eksternal, umumnya yang menyimpan data non-XML.

Sebagai contoh, dalam DTD Anda bisa mendeklarasikan sebuah elemen bernama IMAGE untuk menyatakan sebuah citra grafis, dan sebuah atribut tipe ENTITY bernama Source untuk menandakan sumber data grafis, seperti ini:

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE Source ENTITY #RQU IRED>
```

Jika Anda telah mendeklarasikan sebuah entitas tak-terurai bernama Logo (menggunakan teknik yang akan Anda pelajari dalam Bab 6) yang berisi data grafis untuk sebuah citra, Anda bisa memberikan entitas itu ke atribut Source pada sebuah elemen IMAGE dalam dokumen, seper- ti ini:

```
<IMAGE Source="Logo" I >
```

5. ENTITES. Tipe atribut ini persis seperti sebuah tipe ENTITY, hanya saja nilai tersebut bisa memasukkan nama-nama sejumlah entitas tak terurai dipisahkan dengan karakter spasi kosong-semua dalam string kutipan. Misalnya, jika Anda mendefinisikan atribut Source untuk memiliki tipe ENTITIES, seperti ini:

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE Source ENTITIES #REQUIRED>
```

Anda bisa menggunakannya untuk merujuk sejumlah entitas tak-terurai (mungkin saja entitas yang menyimpan data grafis dalam format alternatiD, seperti ini:

```
< IMAGE Source="LogoGif LogoBmp" I >
```

Contoh ini beranggapan bahwa LogoGif dan LogoBmp adalah nama entitas takterurai yang telah dideklarasikan dalam DTD).

6. NMTOKEN. Nilainya adalah token nama, yaitu nama yang terdiri dari satu atau lebih huruf, digit, titik (.), tanda-kurang (-), atau garis-bawah (_).

Token nama bisa juga berisi sebuah tanda bagi (:) kecuali dalam posisi karakter pertama. Misalnya, jika Anda memberikan atribut ISBN tipe NMTOKEN, seperti ini:

```
<!ELEMENT BOOK(#PCDATA)>
<!ATTLIST BOOK ISBN NMTOKEN #REQUIRED>
```

Anda bisa memberinya sebuah nilai yang diawali dengan angka (pada karakter digit permulaan diizinkan untuk tipe NMTOKEN dan NMTOKENS, namun tidak untuk tipe bertoken lainnya):

```
<BOOK ISBN="9-99999-999-9">The Portrait of a
Lady</BOOK>
```

7. NMTOKENS. Tipe atribut ini persis seperti tipe NMTOKEN hanya saja nilainya bisa memasukkan sejumlah token nama dipisahkan dengan karakter spasi kosong semua dalam string kutipan. Misalnya, jika Anda memberikan atribut Codes tipe NMTOKENS, seperti ini:

```
<!ELEMENT SHIRT(#PCDATA)>
<!ATTLIST SHIRT Codes NMTOKENS #REQUIRED>
```

Anda bisa memberinya sejumlah nilai token nama:

```
<SHIRT Codes="38 21 97">long sle ve Henley
</SHIRT>
```

Penyebutan Tipe Terhitung

Seperti semua nilai atribut, nilai yang Anda berikan pada sebuah tipe terhitung harus berupa string dalam kutipan yang tunduk pada aturan umum, seperti disebutkan dalam "Aturan Nilai Atribut yang Legal". Selain itu, nilai tersebut harus sesuai dengan salah satu nama yang Anda cantumkan dalam spesifikasi tipe-atribut, yang bisa juga memiliki dua bentuk berikut:

1. Sebuah kurung buka, diikuti dengan daftar token nama yang dipisahkan

dengan karakter I, diikuti dengan kurung tutup. Ingatlah bahwa token nama adalah nama yang terdiri dari satu atau lebih huruf, digit, titik (.), tanda kurang (-), atau garis bawah U, dan yang juga bisa berisi sebuah tanda bagi (:) kecuali dalam posisi karakter pertama. Misalnya, jika Anda ingin membatasi nilai-nilai atribut Class pada "fictional ", "instructional", atau "documentary ", Anda bisa mendefinisikan atribut ini sebagai sebuah tipe terhitung, seperti ini:

```
<!ATTLIST FILM
Class (fictional | instructional | document
any)"fictional">
```

Berikut ini dokumen XML selengkapnya yang memperlihatkan penggunaan atribut Class:

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

<!DOCTYPE FILM
   [
   <!ELEMENT FILM (TITLE, (STAR | NARRATOR | INSTRUCTOR))>
   <!ATTLIST FILM
      Class (fictional|instructional|documentary) "fictional">
   <!ELEMENT TITLE (#PCDATA)>
   <!ELEMENT STAR (#PCDATA)>
   <!ELEMENT NARRATOR (#PCDATA)>
   <!ELEMENT INSTRUCTOR (#PCDATA)>
   ]
>

<FILM Class="instructional">
   <TITLE>The Use and Care of XML</TITLE>
   <NARRATOR>Michael Young</NARRATOR>
</FILM>
```

Jika Anda menghilangkan atribut Class, ia akan diberi nilai default "fictional ". Pemberian nilai lain pada Class selain "fictional ", "instructional", atau "documentary " akan menghasilkan kesalahan validitas.

2. Kata-kunci NOTATION, diikuti dengan spasi, diikuti dengan kurung

buka, diikuti daftar nama notasi dipisahkan dengan karakter I, diikuti kurung tutup. Setiap nama ini hams persis sama dengan nama pada notasi yang dideklarasikan dalam DTD. Notasi menjelaskan format data atau mengenali program yang dipakai untuk memproses format tertentu.

Sebagai contoh, anggaplah bahwa notasi HTML, SGML, dan RTF di-deklarasikan dalam DTD Anda, Anda bisa membata si nilai-nilai atribut Format pada salah satu nama notasi dengan mendeklarasikannya seperti ini:

```
<!ELEMENT EXAMPLE_DOCUMENT(#PCDATA)>
<!ATTLIST EXAMPLE_DOCUMENT
Format NOTATION (HTML | SGML | RTF)
#REQUIRED>
```

Kemudian Anda bisa menggunakan elemen Format untuk menandai format pada elemen EXAMPLE_DOCUMENT tertentu, seperti dalam contoh ini:

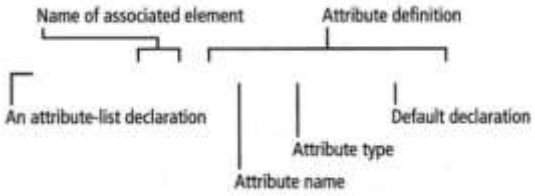
```
<EXAMPLE_DOCUMENT Format="HTML">
<![CDATA[
<HTML>
<HEAD>
<TITLE>Mike's Home Page</TITLE>
</HEAD>
<BODY>
<P>Welcome!</P>
</BODY>
</HTML>
 ]
<IEXAMPLE_DOCUMENT>
```

Memberikan pada Format nilai selain "HTML", "SGML", atau "RTF" akan menghasilkan kesalahan validitas. (Perhatikan penggunaan bagi- an

CDATA di sini, yang memungkinkan Anda menggunakan karakter kurung siku kiri (<) secara bebas dalam data karakter elemen.)

Deklarasi Default

Deklarasi default adalah komponen ketiga dan terakhir yang diperlukan pada sebuah definisi atribut. la menyebutkan apakah atribut diperlukan, dan jika tidak diperlukan, menandai prosesor apa yang harus melakukannya, jika atribut ditinggalkan. Deklarasi tersebut bisa, misalnya, memberikan nilai atribut default yang dipakai prosesor, jika tidak ada atribut.



Deklarasi default memiliki empat bentuk yang memungkinkan:

1. #REQUIRED. Dengan bentuk ini, Anda harus menyebutkan sebuah nilai atribut bagi setiap elemen pada tipe yang bersangkutan. Misalnya, deklarasi berikut ini menandakan bahwa Anda harus memberikan sebuah nilai pada atribut Class dalam tag-awal pada setiap elemen FILM dalam dokumen tersebut:

   ```
   <!ATTLIST FILM Class CDATA #REQUIRED>
   ```

2. #IMPLIED. Bentuk ini menandakan bahwa Anda bisa juga memasukkan atau membuang atribut dari sebuah elemen pada tipe yang bersangkutan. Jika Anda membuang atribut tersebut, tidak ada nilai default yang diberikan pada prosesor (bentuk ini "berarti" ketimbang "menyatakan " sebuah nilai, menyebabkan aplikasi menggunakan nilai defaultnya sendiri

begitulah namanya). Sebagai contoh, deklarasi berikut menandakan bahwa pemberian sebuah nilai pada atribut Class dalam sebuah elemen FILM bersifat opsional, dan dokumen tidak memberikan nilai Class default:

```
<!ATTLIST FILM Class CDATA #IMPLIED>
```

3. AttValue, di mana AttValue adalah nilai atribut default. Dengan bentuk ini, Anda bisa memasukkan atau membuang atribut dari sebuah elemen pada tipe yang bersangkutan. Jika Anda membuangnya, prosesor akan menggunakan nilai default; bila Anda telah mendeklarasikan atribut tersebut dan mengetikkan nilainya.

   Nilai default yang Anda berikan harus, tentu saja, patuh pada tipe atribut yang dinyatakan. Sebagai contoh, deklarasi berikut memberikan nilai default "fictional" pada atribut Class:

   ```
   <!ATTLIST FILM Class CDATA "fictional">
   ```

   Dengan deklarasi ini, dua elemen berikut adalah sama:

   ```
   <FILM>The Graduate</FILM>
   <FILM>Class="fictinal">The Graduate</FILM>
   ```

4. #FIXED AttValue, di mana AttValue adalah nilai atribut default. Dengan bentuk ini, Anda bisa memasukkan atau membuang atribut dari sebuah elemen pada tipe yang bersangkutan. Jika Anda membuangnya, prosesor akan menggunakan nilai default; jika Anda memasukkannya, Anda harus menyebutkan nilai defaultnya. (Karena Anda hanya bisa menyebutkan nilai default, tidak ada alasan memaksa untuk memasukkan sebuah spesifikasi atribut dalam suatu elemen, kecuali mungkin saja untuk menjadikan dokumen lebih jelas untuk dibaca orang.) Sebagai contoh, deklarasi berikut memberikan sebuah nilai default baku pada atribut

Class:

```
<!ATTLIST FILM Class CDATA
#FIXED"documentary">
```

Dengan deklarasi ini, dua elemen berikut sama validnya:

```
<FILM>The Making of XML</FILM>
<FILM> Class="documentary">The Making of
XML</FILM>
```

sementara elemen berikut tidak valid:

```
<!- Invalid elemen! ->
<FILM>Class="instructional">The Making of
XML</FILM>
```

Menggunakan Subset OTO Eksternal

Definisi tipe dokumen yang telah Anda lihat dalam bab ini dimuat seluruhnya dalam deklarasi tipe dokumen pada dokumen tersebut. Tipe DTD ini dikenal sebagai subset DTD internal.

Sebagai alternatif, Anda bisa mengganti semua atau sebagian dokumen DTD dalam sebuah file tersendiri, dan kemudian merujuk pada file itu dari deklarasi tipe dokumen. DTD-atau sebagian dari DTD-yang termuat dalam file tersendiri dikenal sebagai sebuah subset DTD eksternal.

Menggunakan Subset OTO Eksternal Saja

Untuk hanya menggunakan sebuah subset DTD eksternal, buanglah blok deklarasi markup dalam karakter kurung persegi (0), dan sebagai gantinya masukkan kata-kunci SYSTEM diikuti dengan petikan uraian pada lokasi file terpisah yang berisi DTD. Anggaplah, sebagai contoh, dokumen SIMPLE yang Anda lihat sebelumnya dalam bab ini, memiliki sebuah subset DTD

internal:

```
<?xml version= "l.O"?>
<!DOCTYPESIMPLE
[
<!ELEMENTSIMPLEANY>
>
<SIMPLE> This is an extremely simpli stic
XML document. </SIMPLE>
```

Jika dokumen ini menggunakan sebuah subset DTD eksternal, ia akan tampak seperti ini:

```
<?xml version="l.O"?>
<!DOCTYPE SIMPLE SYSTEM "Simple.dtd">
<SIMPLE> This is anextremely simplistic XML
document<!SIMPLE>
```

File Simple .dtd akan memiliki isi berikut ini:

```
<!ELEMEN SIMPLE ANY>
```

File yang berisi subset DTD eksternal tersebut bisa memasukkan sem- barang deklarasi markup yang bisa dimasukkan dalam sebuah subset DTD internal. Saya mencantumkannya sebelumnya dalam bagian "Membuat DTD", pada bab ini.

Penjelasan lokasi file ("Simple.dtd" dalam contoh ini) dikenal sebagai lite- ral sistem. Ia bisa dibatasi menggunakan tanda petik tunggal (') atau petik- ganda ("), dan ia bisa memuat sembarang karakter kecuali karater petik yang dipakai untuk membatasinya.

Literal sistem menyebutkan uniform resource identifier (URL) pada file yang memuat subset DTD eksternal. Saat ini, URI sangat pentingnya dengan alamat Internet standar, yang dikenal dengan Uniform Resource Locator atau URL. Anda bisa menggunakan URI lengkap, seperti:

```
<!DOCTYPE SIMPLE SYSTEM
"http://bogus.com/dtds/Simple.dtd">
```

Atau, Anda bisa memakai. sebagian URL yang menyebutkan lokasi yang menunjuk pada lokasi dokum n XML yang berisi URL, seperti:

```
<!DOCTYPE SIMPLE SYSTEM "Simple.dtd">
```

URL relatif dalam dokumen XML bekerja seperti URL relatif dalam halaman HTML. Dalam contoh kedua, jika URL lengkap pada dokumen XML berupa http://bogus.com/documents/Simple .xml, "Simple.dtd" akan merujuk http://bogus.com/documents!Simple.dtd. Seperti halnya, jika dokumen XML ditempatkan pada file :///C:\XML Step by Step\Example Code\Simple. xml, "Simple.dtd ' akan merujuk file :///C:\ XML Step by Step\Example Code\ Simple .dtd.

Menggunakan Subset DTD Eksternal Bersama Subset DTD Internal

Untuk menggunakan subset DTD eksternal bersama subset DTD internal, masukkan kata-kunci SYSTEM bersama dengan literal sistem yang menampilkan lokasi file subset DTD eksternal, diikuti dengan deklarasi markup subset DTD internal dalam karakter kurung persegi ([]).

Berikut ini contoh dokumen XML sederhana dengan subset DTD eksternal dan subset DTD internal:

```
<?xml version-"1.0"?>
<!DOCTYPEBOOK SYSTEM "Book.dtd"
[
<!ATTLIST BOOK ISBN CDATA #IMPLIED Year
CDATA"2000">
<!ELEMENT TITLE (#PCDATA)>
]
>
```

```
<BOOK Year-"1998">
<TITLE>The Scarlet Letter/TITLE>
<!BOOK >
```

Berikut ini adalah isi file yang memuat subset DTD eksternal, Book.dtd:

```
<!ELEMENT BOOK ANY>
<!ATTLIST BOOK ISBN NMTOKEN #REQUIRED>
```

Bila Anda memasukkan subset DTD eksternal dan subset DTD internal, beginilah prosesor XML mengkombinasikan isinya:

1. Secara umum, ia menggabungkan isi kedua subset untuk membentuk DTD yang utuh. Misalnya, hasil penggabungan DTD mendefinisikan dua elemen, TITLE dan BOOK, dan dua atribut untuk elemen BOOK, ISBN dan Year.
2. Jika sebuah atribut dengan nama yang sama dan tipe elemen yang sama dideklarasikan lebih dari sekali, prosesor XML akan memakai deklarasi yang pertama, dan mengabaikan selebihnya (ini juga diterapkan pada pendeklarasian-ulang entitas).
3. Subset DTD internal didahulukan dari subset DTD eksternal (walaupun acuan subset eksternal muncul pertama dalam deklarasi tipe dokumen). Sehingga, semua atribut (atau entitas) yang didefinisikan dalam subset internal mendahului yang lain dengan nama yang sama dan tipe elemen sama yang dideklarasikan dalam subset eksternal. Dalam contoh, prosesor XML menganggap atribut ISBN memiliki tipe CDATA dan karenanya elemen berikut (yang meninggalkan ISBN) adalah valid:

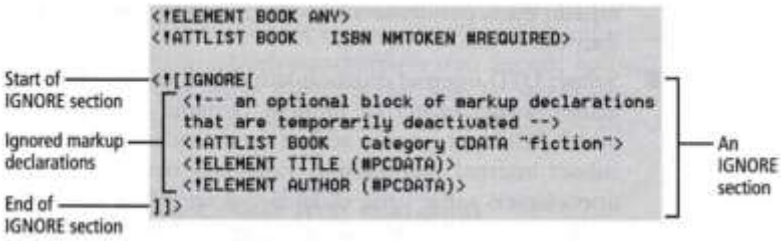
```
< BOOK Year-"1850">
<TITLE>The Scarlet Letter</TITLE>
</BOOK>
```

Cara prosesor XML mengkombinasikan subset DTD eksternal dan internal memungkinkan Anda memakai DTD bersama (seperti yang diberikan pada aplikasi XML, seperti mathML) sebagai sebuah subset DTD eksternal, namun kemudian disesuaikan (atau subclass, dalam istilah pemrograman) DTD untuk dokumen yang ada dengan memasukkan subset internal. Subset internal Anda bisa menambahkan berbagai elemen, atribut, atau entitas-dan ia bisa menggan ti definisi atribut atau entitas.

Mengabaikan Bagian Subset DTD Eksternal

Anda bisa membuat prosesor XML mengabaikan bagian dari subset DTD eksternal menggunakan bagian IGNORE. Anda bisa, misalnya, menggunakan sebuah bagian IGNORE, bila Anda sedang mengembangkan sebuah dokumen untuk tidak mengaktifkan sementara waktu sebuah alternatif atau blok deklarasi markup opsional. Dengan demikian Anda tidak perlu menghapus baris tersebut dan bisa menyisipkan mereka kembali nantinya (jika Anda seoarang programmer, Anda akan mengerti bahwa teknik ini sama dengan "Pemberian tanda komentar " pada blok kode yang ingin Anda abaikan sementara waktu). Bagian IGNORE dimulai dengan karakter <! IGNORE {dan diakhiri dengan karakter [[>.

Berikut ini contoh subset DTD eksterna l lengkap yang memasukkan sebuah bagian IGNORE:



Jika Anda ingin sementara waktu mengaktifkan kembali sebuah blok deklarasi markup dalam sebuah bagian IGNORE, Anda cukup mengganti kata kunci IGNORE dengan INCLUDE, tanpa harus menghapus semua karakter pembatas (< [[, [, dan [[>), seperti dalam contoh ini:

```
<! [INCLUDE [
<! - - anoptional block of markup
declarations that are temporarily
reactivated - - >
<! ATTLIST BOOK Category CDATA "fiction">
<! ELEMENT TITLE(#PCDATA)>
<! ELEMENT AUTHOR(#PCDATA)>
]] >
```

Anda bisa dengan cepat mematikan bagian itu kembali dengan menempatkan lagi IGNORE. Bagian INCLUDE tersarang dalam sebuah bagian IGNORE tetap diabaikan.

Mengubah Dokumen Well-Formed menjadi Dokumen Valid

Pada bagian ini, Anda akan memperoleh pengalaman berlatih dengan konsep yang disajikan dalam bab ini melalui konversi dokumen well-formed menjadi dokumen yang valid. Anda akan mengubah dokumen Inventory. xml yang Anda buat untuk menjadikan dokumen tersebut valid. Anda juga akan

menambahkan sebuah elemen baru dan dua atribut untuk mencoba teknik tambahan yang telah Anda pelajari dalam bab ini.

Menjadikan Dokumen Valid

1. Dalam editor teks, buka dokumen Inventory.xml yang telah Anda buat dalam Bab 2 (dokumen tersebut terdapat dalam Listing 2-1).
2. Tepat di atas elemen dokumen-bernama INVENTORY ketikkan dalam deklarasi tipe dokumen yang berikut ini:

```
<!DOCTYPE INVENTORY
[
<!ELEMENT INVENTORY (BOOK)*>
<!ELEMENT BOOK (TITLE, AUTHOR, BINDING,
PAGES, PRICE)>
<!ATTLIST BOOK  InStock (yes|no) #REQUIRED>
<!ELEMENT TITLE (#PCDATA | SUBTITLE)*>
<!ELEMENT SUBTITLE (#PCDATA>
<!ELEMENT AUTHOR (#PCDATA>
<!ATTLIST AUTHOR Born CDATA #IMPLIED>
<!ELEMENT BINDING (#PCDATA)>
<!ELEMENT PAGES (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
]
>
```

TIP Dalam melakukan modifikasi yang diberikan dalam latihan ini, Anda bisa merujuk pada dokumen yang telah dimodifikasi dalam Listing 13-1 pada akhir bab ini.

Perhatikan nama kata-kunci DOCTYPE berikut terkait dengan nama pada elemen dokumen, INVENTORY, seperti yang disyaratkan. DTD tersebut terdiri atas sebuah subset internal saja, yang mendefinisikan elemen dan

atribut dokumen sebagai berikut:

1. Elemen dokumen, INVENTORY, memiliki isi elemen. Ia bisa berisi nol atau sejumlah elemen anak BOOK.
2. Elemen BOOK juga memiliki isi elemen. Ia harus memuat persis salah satu dari masing-masing elemen berikut, dalam urutan yang tercantum pada deklarasi elemen: TITLE, AUTHOR, BINDING, PAGES, dan PRICE.
3. Elemen TITLE memiliki isi campuran. Ia bisa memuat data karakter yang berbaur dengan nol atau lebih elemen SUBTITLE.
4. Elemen-elemen AUTHOR, BINDING, PAGES, dan PRICE juga masing-masing memiliki isi campuran. Elemen-elemen ini bisa memuat data karakter saja dan tidak elemen anak.
5. Elemen BOOK memiliki atribut tipe terhitung bernama InStock, yang merupakan sebuah atribut diperlukan yang padanya bisa diberikan "yes" atau "no".
6. Elemen AUTHOR memiliki sebuah atribut tipe string bernama Born, yang bersifat opsional dan tidak memiliki nilai default.

3. Tambahkan elemen anak SUBTITLE berikut ke elemen TITLE untuk buku Moby-Dick:

```
<BOOK>
<TITLE>Moby- Dick
<SUBTITLE> Or, the Whale</SUBTITLE>
<! TITLE>
```

4. Tambahkan atribut InStock pada masing-masing elemen BOOK, berikan padanya "yes" atau "no" seperti yang tampak di sini:

```
< BOOK InStock="yes">
```

```
<TITLE>The Adventures of Huckleberry
Finn</TITLE>
<AUTHOR> Mark Twain</AUHOR >
< BINDING>mass marketpaperback</BINDING>
<PAGES>298</PAGES>
<PRICE>$5.49</PRICE>
<! BOOK>
```

5. Tambahkan elemen Born pada satu atau beberapa elemen. Walaupun Anda bisa memberikan pada atribut ini sembarang string kutipan yang legal, tujuannya adalah untuk menyimpan tanggal lahir penulis. Inilah contohnya:

```
<Author Born="l835">mark Twain</AUTHOR>
```

6. Untuk mencerminkan nama file baru yang akan Anda berikan, gantilah komentar pada awal dokumen dari:

```
<! - File Name: Inventory.xml->
```

menjadi:

```
<! - File Name: Inventory Valid.xml->
```

7. Gunakan perintah Save pada editor teks untuk menyimpan salinan modifika si dokumen ini dengan nama Inventory Valid.xml.

Dokumen XML selengkapnya diperlihatkan dalam Listing 13-1.

Inventory Valid.xml

```
<?xml version="1.0"?>
<!-- File Name: Inventory Valid.xml -->
<!DOCTYPE INVENTORY
   [
   <!ELEMENT INVENTORY (BOOK)*>
   <!ELEMENT BOOK (TITLE, AUTHOR, BINDING,
PAGES, PRICE)>
   <!ATTLIST BOOK   InStock (yes|no)
#REQUIRED>
```

```
   <!ELEMENT TITLE (#PCDATA | SUBTITLE)*>
   <!ELEMENT SUBTITLE (#PCDATA)>
   <!ELEMENT AUTHOR (#PCDATA)>
   <!ATTLIST AUTHOR   Born CDATA #IMPLIED>
   <!ELEMENT BINDING (#PCDATA)>
   <!ELEMENT PAGES (#PCDATA)>
   <!ELEMENT PRICE (#PCDATA)>
   ]>
<INVENTORY>
   <BOOK InStock="yes">
      <TITLE>The Adventures of Huckleberry
Finn</TITLE>
      <AUTHOR Born="1835">Mark
Twain</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK InStock="no">
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR Born="1819">Walt
Whitman</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>462</PAGES>
      <PRICE>$7.75</PRICE>
   </BOOK>
   <BOOK InStock="yes">
      <TITLE>The Legend of Sleepy
Hollow</TITLE>
      <AUTHOR>Washington Irving</AUTHOR>
```

```
      <BINDING>mass market
paperback</BINDING>
      <PAGES>98</PAGES>
      <PRICE>$2.95</PRICE>
   </BOOK>
   <BOOK InStock="yes">
      <TITLE>The Marble Faun</TITLE>
      <AUTHOR Born="1804">Nathaniel
Hawthorne</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>473</PAGES>
      <PRICE>$10.95</PRICE>
   </BOOK>
   <BOOK InStock="no">
      <TITLE>Moby-Dick
         <SUBTITLE>Or, the Whale</SUBTITLE>
      </TITLE>
      <AUTHOR Born="1819">Herman
Melville</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>724</PAGES>
      <PRICE>$9.95</PRICE>
   </BOOK>
   <BOOK InStock="yes">
      <TITLE>The Portrait of a Lady</TITLE>
      <AUTHOR>Henry James</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
      <PAGES>256</PAGES>
      <PRICE>$4.95</PRICE>
   </BOOK>
   <BOOK InStock="yes">
      <TITLE>The Scarlet Letter</TITLE>
```

```
            <AUTHOR>Nathaniel Hawthorne</AUTHOR>
            <BINDING>trade paperback</BINDING>
            <PAGES>253</PAGES>
            <PRICE>$4.25</PRICE>
         </BOOK>
         <BOOK InStock="no">
            <TITLE>The Turn of the Screw</TITLE>
            <AUTHOR>Henry James</AUTHOR>
            <BINDING>trade paperback</BINDING>
            <PAGES>384</PAGES>
            <PRICE>$3.35</PRICE>
         </BOOK>
      </INVENTORY>
```

Listing 13-1.

8. Jika Anda ingin menguji validitas dokumen Anda, gunakan script pemeriksaan validitas XML yang diberikan pada "Memeriksa Validitas Dokumen XML

C. PENUTUP

Ringkasan

Prosesor Internet Explorer 5 memeriksa validitas dokumen hanya jika Anda membuka dokumen melalui sebuah halaman Web HTML. Jika Anda membuka sebuah dokumen XML langsung dalam Internet Explorer 5, prosesor akan memeriksa kerapian dokumen (termasuk berbagai deklarasi tipe dokumen di dalamnya), namun ia tidak akan memeriksa validitas dokumen bahkan seandainya ia berisi sebuah deklarasi tipe dokumen.

Seperti semua kata kunci XML, DOCTYPE harus semuanya dalam huruf besar.

Anda bisa mendeklarasikan tipe elemen tertentu hanya sekali pada dokumen

yang ada.

Kata-kunci PCDATA adalah singkatan dari parsed character data. Anda telah mempelajari bahwa prosesor XML menguraikan data karakter dalam sebuah elemen yakni, ia memeriksa elemen tersebut untuk mencari markup XML. Karenanya Anda tidak bisa menyisipkan kurung siku kiri (<) atau ampersand (&) atau string]] > sebagai bagian dari data karakter, karena pengurai akan menafsirkan setiap karakter atau sekuen ini sebagai markup. Anda bisa menyisipkan sembarang karakter menggunakan sebuah acuan karakter atau acuan entitas yang telah ada atau sebuah bagian CDATA.

Jika Anda memasukkan lebih dari satu deklarasi daftar-atribut untuk tipe elemen yang diberikan, isi kedua deklarasi akan digabungkan. Jika sebuah atribut dengan nama yang diberikan dideklarasikan lebih dari sekali untuk elemen yang sama, deklarasi pertama dipakai dan yang kedua diabaikan. (Deklarasi daftar multi-atribut lebih umum dipakai ketika sebuah dokumen memiliki subset DTD internal dan eksternal,)

Menggunakan subset OTO eksternal sangat menguntungkan sekali bagi OTO bersama yang dipakai oleh seluruh kelompok dokumen. Masing-masing dokumen bisa merujuk file OTO tunggal (atau salinan file itu) sebagai sebuah subset OTO eksternal. lni menghindari keharusan menyalin isi OTO ke dalam setiap dokumen yang memakainya, dan juga memudahkannya dalam mengelola OTO (Anda hanya perlu mengubah file OTO tunggal dan salinan file tersebut ketimbang harus mengedit semua dokumen yang menggunakannya). Lihatlah kembali bahwa beberapa aplikasi XML standar yang berdasarkan pada OTO bersama yang dimasukkan dalam semua dokumen XML tunduk pada aplikasi tersebut. Untuk mengulang, perhatikan

"Aplikasi XML Standar" dan "Penggunaan XML untuk Pekerjaan Nyata".

URL adalah sistem notasi baru yang sangat luwes bagi pengalamatan sumber daya. Salah satu tipe URL adalah URL (Uniform Resource Locator) biasanya dipakai pada Internet (sebagai contoh, http://mspress .microsoft.com/). Di masa men- datang, URL akan memasukkan tipe-tipe notasi lain untuk pengalamatan sumber daya. Semua ini masih dalam tahap pengembangan.

Walaupun prosesor XML hanya mengabaikan deklarasi ulang pada atribut dan entitas, deklarasi ulang pada sebuah elemen (bahkan jika ia dideklarasikan dengan cara sama) adalah ilegal.

Anda bisa menggunakan bagian IGNORE dan INCLUDE hanya dalam subset DTD eksternal, atau dalam entitas parameter eksternal (seperti yang akan Anda pelajari, entitas parameter eksternal merujuk sebuah file tersendiri yang seperti sebuah subset DTD eksternal berisi deklarasi markup).

Pertanyaan

1. Apa keuntungan membuat dokumen XML yang valid?
2. Apa kriteria dasar sebuah dokumen XML yang valid?
3. Sebutkan dan jelaskan Spesifikasi isi dari Elemen?
4. Tuliskan bentuk dari deklarasi daftar atribut?

DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie

4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018
6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

BAB 14

Membuat Dokumen XML yang Well- Formed

A. PENDAHULUAN

1. Deskripsi Singkat*:*

Pada mata kuliah ini khususnya pada BAB 14 akan di bahas Membuat Dokumen XML yang Well Formed, yang mencakup diantaranya: Bagian Dokumen XML yang Well Formed, Dokumen XML Minimalis, Menambahkan Elemen ke Dokumen, Anatomi Elemen, Tipe isi Elemen, Elemen Kosong, Membuat Tipe Elemen yang Berbeda, Menambahkan Atribut ke elemen, Aturan untuk Pembuatan Atribut, Aturan untuk Nilai Atribut yang Sah, Mengubah isi menjadi Atribut.

2. Kemampuan Akhir yang diharapkan:

Melalui buku ajar ini diharapkan mahasiswa dapat menjelaskan cara cara Bagian Dokumen XML yang Well Formed, Dokumen XML Minimalis, Menambahkan Elemen ke Dokumen, Anatomi Elemen, Tipe isi Elemen, Elemen Kosong, Membuat Tipe Elemen yang Berbeda, Menambahkan Atribut ke elemen, Aturan untuk Pembuatan Atribut, Aturan untuk Nilai Atribut yang Sah, Mengubah isi menjadi Atribut.

B. PENYAJIAN

Membuat Dokumen XML yang Well- Formed

Dalam bab ini, Anda akan belajar teknik-teknik dasar untuk membuat dokumen XML yang baik. Dokumen yang baik adalah dokumen yang

memenuhi sejumlah kriteria bagi sebuah dokumen XML. Bila Anda membuat dokumen XML yang menarik, Anda bisa menempatkan dan menambahkan elemen yang Anda perlukan, dan memasukkan data dokumen Anda, persis seperti yang Anda lakukan ketika membuat halaman Web HTML. (Walaupun, seperti yang Anda pelajari dalam bab sebelumnya, dalam sebuah dokumen XML Anda membuat elemen sendiri ketimbang menggunakan yang telah ada.) Anda tidak akan bermasalah dalam menangani dan menampilkan dokumen XML yang baik tersebut pada Internet Explorer 5.

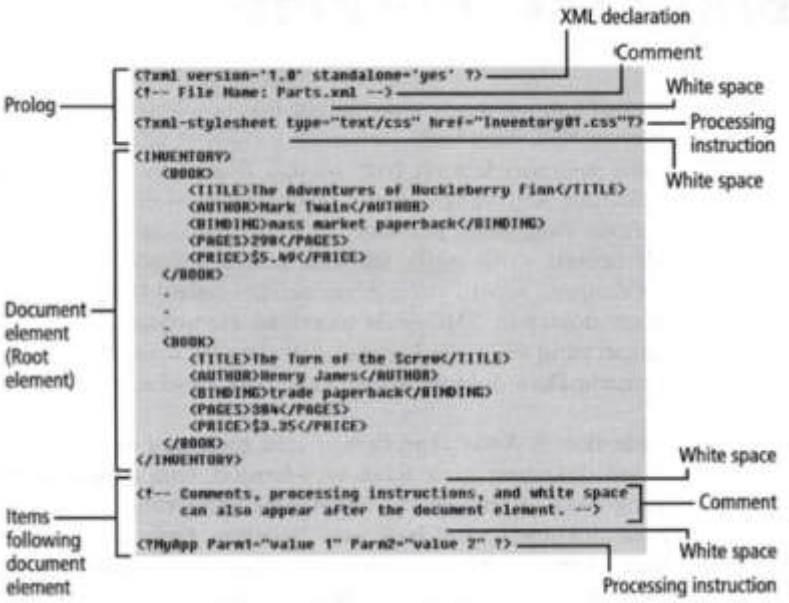
Pada Bab ini, Anda akan belajar cara membuat dokumen XML yang valid: yakni dokumen yang tidak well-formed, tetapi juga tunduk pada aturan yang baku. Membuat dokumen XML yang valid tidaklah semudah membuat dokumen yang baik saja. Sebelum Anda mulai menambahkan elemen-elemen dan data ke dokumen yang valid, Anda harus mendefinisikan dengan lengkap struktur dokumen tersebut dalam sebuah deklarasi tipe dokumen yang telah ditambahkan ke pengantar dokumen. Dalam Bab ini akan Anda pelajari bahwa ada beberapa manfaat dalam membuat dokumen yang valid, khususnya jika Anda atau yang lainnya membuat sejumlah dokumen yang mirip.

Dalam bab ini, Anda akan belajar tentang semua bagian yang diperlukan dan yang bersifat opsional pada dokumen XML yang menarik. Kemudian Anda akan temukan bagaimana cara menambahkan informasi ke dokumen XML dengan mendefinisikan elemen dokumen. Anda akan mempelajari cara memberikan informasi dokumen tambahan dengan menambahkan atribut-atribut ke elemen-elemen tersebut.

Bagian Dokumen XML yang Well-Formed

Seperti yang telah Anda pelajari pada Bab sebelumnya, sebuah dokumen XML terdiri atas dua bagian pokok: prolog dan elemen dokumen (yang juga dikenal sebagai elemen root). Sebagai tambahan, mengikuti elemen dokumen, sebuah dokumen XML yang menarik bisa memasukkan berbagai komentar, instruksi pemrosesan, dan spasi kosong. Berikut ini sebuah contoh dokumen XML yang well-formed yang menunjukkan perbedaan bagian bagian dokumen dan item-item yang bisa Anda tambahkan pada masing- masing bagian.

Listing 14-1 memperlihatkan versi selengkapnya dari dokumen contoh ini. (Anda akan temukan salinannya pada CD penyerta dengan nama file Parts. xml.)



Parts.xml

```
<?xml version='1.0' standalone='yes' ?>
<!-- File Name: Parts.xml -->
<?xml-stylesheet type="text/css"
href="Inventory01.css"?>
<INVENTORY>
   <BOOK>
      <TITLE>The Adventures of Huckleberry
Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR>Walt Whitman</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>462</PAGES>
      <PRICE>$7.75</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Legend of Sleepy
Hollow</TITLE>
      <AUTHOR>Washington Irving</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
      <PAGES>98</PAGES>
      <PRICE>$2.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Marble Faun</TITLE>
      <AUTHOR>Nathaniel Hawthorne</AUTHOR>
```

```
      <BINDING>trade paperback</BINDING>
      <PAGES>473</PAGES>
      <PRICE>$10.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Moby-Dick</TITLE>
      <AUTHOR>Herman Melville</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>724</PAGES>
      <PRICE>$9.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Portrait of a Lady</TITLE>
      <AUTHOR>Henry James</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
      <PAGES>256</PAGES>
      <PRICE>$4.95</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Scarlet Letter</TITLE>
      <AUTHOR>Nathaniel Hawthorne</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>253</PAGES>
      <PRICE>$4.25</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>The Turn of the Screw</TITLE>
      <AUTHOR>Henry James</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>384</PAGES>
      <PRICE>$3.35</PRICE>
   </BOOK>
```

```
</INVENTORY>

<!-- Comments, processing instructions, and
white space
     can also appear after the document
element. -->
<?MyApp Parm1="value 1" Parm2="value 2" ?>
```

Listing 14-1.

Nomor versi dalam deklarasi XML pada bagian awal pengantar dokumen dapat ditutup dengan tanda petik tunggal atau ganda. Secara umum, tanda petik yang dalam kode XML dikenal sebagai literal dapat menggunakan pula tanda petik tunggal atau ganda. Dengan demikian, kedua ini adalah sah:

```
<? xml version='l.O'?>
<? xml version="l.O"?>
```

Deklarasi XML dalam dokumen contoh pada Listing 3-1 juga memasukkan sebuah deklarasi deklarasi dokumen standalone (standalone ='yes). Deklarasi ini dapat dipakai pada sejumlah dokumen XML untuk menyederhanakan pemrosesan dokumen.

Dokumen contoh tersebut berisi sebuah komentar dalam penga ntarnya dan komentar lain setelah elemen dokumen.

Dokumen tersebut juga berisi sebuah baris kosong yang dinamai "spasi kosong " pada peng antarnya, dan lainnya yang mengikuti elemen dokumen tersebut. Spasi kosong terdiri atas dua atau lebih spasi, tab, paragraf baru, atau karakter umpan baris. Agar dokumen XML lebih mudah dibaca orang, Anda dapat bebas menambahkan spasi kosong antara tanda XML (misalnya tag mulai, tag akhir, komentar, dan instruksi pemro sesan) dan juga pada sejumlah tempat dalam tanda (sebagai contoh, spasi antara "yes' dan? pada bagian akhir deklarasi XML dalam dokumen contoh). Prosesor cukup mengaba ikan saja

spasi kosong kecuali jika ia dalam sebuah elemen yang langsung berisi data karakter. (Dalam hal ini, prosesor melewatkan spasi kosong ke aplikasi sebagai bagian dari data karakter elemen.)

Dokumen contoh mempunyai sebuah instruksi pemrosesan dalam pengantar dan lainnya yang mengikuti elemen dokumen.

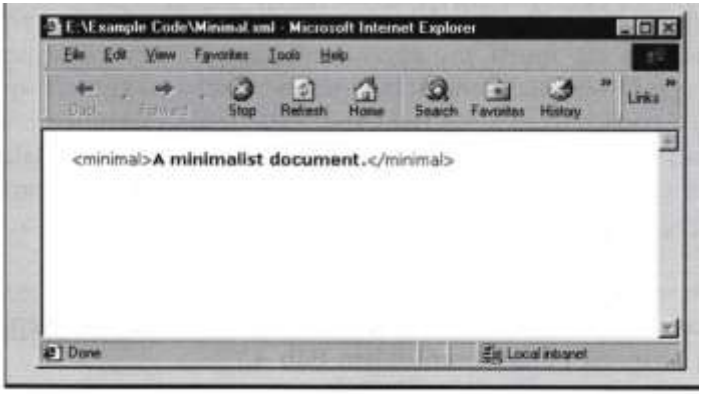
Terakhir, dokumen tersebut mema sukkan sinequanon dari sebuah dokumen XML: elemen dokumen. Pembuatan elemen dokumen dan elemen tersarang yang dikandungnya adalah fokus bab ini.

Dokumen XML Minimalis

Pengantar dalam. dokumen XML contoh pada Listing 14-1 berisi sel; mah contoh masing. masing item yang diizinkan dalam sebuah pengantar. Perhatikan item item ini semuanya bersifat opsional (walaupun menyebutkan spesifikasi XML yang Anda "haruskan" termasuk deklarasi XML). Pengantar itu sendiri adalah opsional sifatnya, dan dokumen minimalis berikut, yang hanya berisi sebuah elemen dokumen sederhana, tunduk pada standar XML bagi sebuah dokumen yang baik.

```
<minimal>Aminima list document</minimal>
```

Dokumen ini akan ditampilkan dalam Internet Explorer 5 seperti tam- pak di sini:



Menambahkan Elemen ke Dokumen

Elemen-elemen dalam dokumen XML berisi informasi dokumen aktual (dalam Listing 14-1, sebagai contohnya, judul, penulis, harga, dan informasi lainnya pada buku dalam inventaris) dan mereka menandakan struktur logika informasi ini.

Elemen-elemen tersebut disu sun dalam sebuah struktur hierarki berbentuk pohon, dengan elemen yang tersarang dalam elemen lain. Dokumen tersebut harus tepat di atas satu elemen-dokumen elemen tingkat-atas atau element root-dengan semua elemen lainnya yang bersarang di dalamnya. Dengan demikian, yang berikut ini adalah sebuah dokumen XML yang rapi:

```
<?xml version="1.0"?>
<!-- A well-formed XML document. -->
<INVENTORY>
   <BOOK>
      <TITLE>The Adventures of Huckleberry Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
      <BINDING>mass market paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK>
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR>Walt Whitman</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>462</PAGES>
      <PRICE>$7.75</PRICE>
   </BOOK>
</INVENTORY>
```

Sedangkan yang berikut ini bukanlah dokumen yang well-formed:

```
<?xml version="1.0"?>
<!-- This document is NOT well-formed. -->
<BOOK>
   <TITLE>The Adventures of Huckleberry Finn</TITLE>
   <AUTHOR>Mark Twain</AUTHOR>
   <BINDING>mass market paperback</BINDING>
   <PAGES>298</PAGES>
   <PRICE>$5.49</PRICE>
</BOOK>
<BOOK>
   <TITLE>Leaves of Grass</TITLE>
   <AUTHOR>Walt Whitman</AUTHOR>
   <BINDING>hardcover</BINDING>
   <PAGES>462</PAGES>
   <PRICE>$7.75</PRICE>
</BOOK>
```

Elemen juga harus disarangkan dengan benar. Yakni, jika sebuah elemen (dikurung dengan sebuah tag-awal dan tag-akhir, seperti yang akan saya jelaskan nanti) mulai dalam elemen lain, ia ju ga harus berakhir dalam elemen yang sama. Sebagai contoh, elemen ini ditulis dengan baik:

```
<BOOK>
```

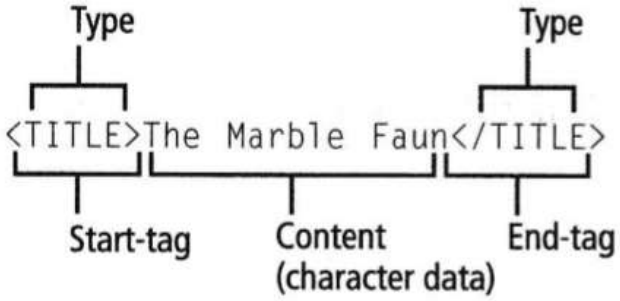
```
<TITLE>Leaves of Grass</TITLE>
<AUTHOR>Walt Whitman</AUTHOR>
<!BOOK >
```

Elemen berikut ini tidak dibuat dengan rapi:

```
<! - NOT well- formed: ->
<BOOK><TITLE>Leaves of Grass</BOOK></TITLE>
```

Anatomi Elemen

Seperti yang Anda lihat, sebuah elemen biasanya terdiri dari tag-mulai, isi, dan tag-akhir.



Tidak seperti HTML, XML mengharuskan Anda selalu memasukkan kedua tag-awal dan tag-akhir (perkecualian satu-satunya adalah elemen tanpa isi, di mana Anda bisa menggunakan tag element-kosong khusus yang akan di jelaskan kemudian dalam bab ini).

Nama yang muncul pada awal tag-mulai dan tag-akhir (TITLE dalam contoh di atas) dikenal sebagai tipe atau pengenal generik elemen (GI-generic identifier). Nama tipe mengenali sebuah tipe atau kelas khusus, bukan elemen yang spesifik (misalnya elemen BOOK atau TITLE dalam Listing 14-1).

Saat Anda menambahkan sebuah elemen ke dokumen XML, Anda bisa

memilih suatu nama tipe yang diinginkan, asalkan Anda mengikuti ketentuan berikut ini:

1. Nama harus dimulai dengan sebuah huruf atau garis-bawah (_), diikuti dengan atau tanpa huruf, digit, periode (.), tanda kurang (-), atau beberapa tanda garis-bawah.
2. Spesifikasi XML menyatakan bahwa nama-nama tipe-elemen diawali dengan awalan 'xml" (dalam berbagai kombinasi huruf besar atau kecil) dijadikan sebagai "standarisasi". Walaupun Internet Explorer 5 tidak memaksakan pembatasan ini, lebih baik tidak menggunakan awalan ini untuk menghindari masalah-masalah di masa mendatang.

Berikut ini adalah nama-nama tipe-elemen yang sah:

```
Part
_1stPlace
A
B-SECTION
Street.Address.1
```

Sedangkan yang berikut ini adalah nama-nama tipe-elemen yang tidak sah:

```
1stPlace   <!-- Digit not allowed as first character -->
B Section  <!-- Space not allowed within a name -->
B/Section  <!-- Slash not allowed within a name -->
:Chapter   <!-- Colon not allowed as first character in IE5 -->
A:Section  <!-- In IE5, allowed only if you've declared A
                as a namespace -->
```

Juga, nama dalam tag-mulai harus persis sesuai nama dalam tag-akhir, termasuk besar-kecilnya huruf. Sehingga, elemen berikut ini bukanlah elemen yang ditulis dengan baik:

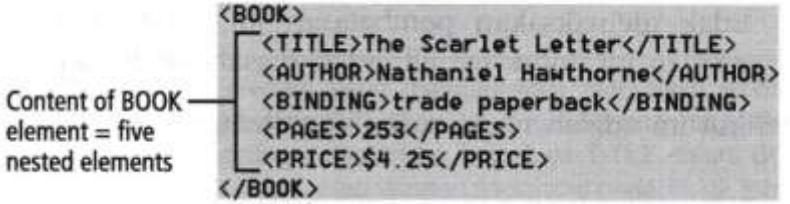
```
<Title>Chapter One</title><!- NOT wel1-
formed->
```

Besar-kecilnya adalah penting pada nama-nama elemen, sebagaimana ia dalam sebuah teks pada tanda. Sehingga, sebuah tipe elemen yang dinamai Ace bukanlah tipe elemen yang sama dengan ace atau ACE.

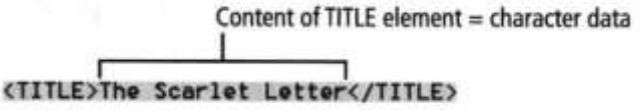
Tipe lsi Elemen

Isi elemen adalah teks antara tag-mulai dan tag-akhir. Anda bisa memasukkan tipe item berikut ini dalam isi sebuah elemen:

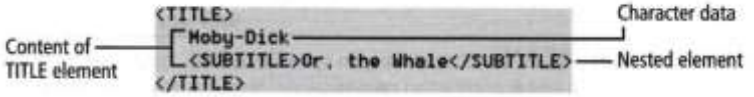
1. Elemen tersarang. Pada Listing 14-1, baik elemen INVENTORY maupun elemen BOOK berisi elemen tersarang sebagai isinya:



2. Data karakter. Data karakter adalah teks yang menyatakan isi informasi pada sebuah elemen, seperti misalnya judul buku tertentu pada elemen TITLE:



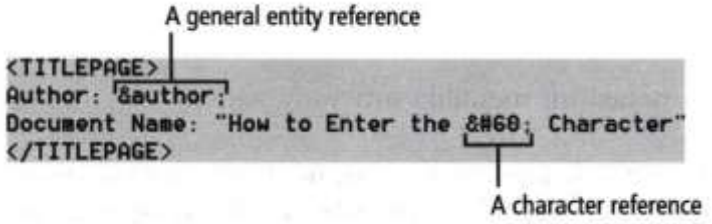
Berikut ini sebuah contoh isi elemen yang berisi data karakter dan sebuah elemen tersarang:



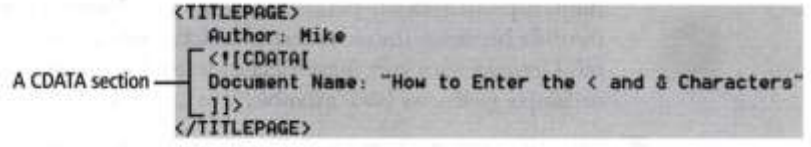
Saat menambahkan data karakter ke sebuah elemen, Anda bisa

menyisipkan sembarang karakter kecuali kurung siku kiri (<) , ampersand (&) atau string]]>.

3. Acuan entitas um.um atau acuan karakter. Ini adalah sebuah elemen yang berisi salah satunya, masing-masing:

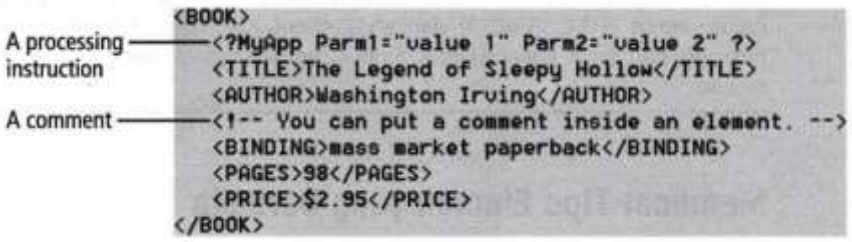


4. Bagian CDATA. Bagian CDATA adalah sebuah blok teks di mana Anda bebas menyisipkan berbagai karakter selain string]] >. Berikut ini contoh bagian CDATA pada sebuah elemen:



5. lnstruksi pemrosesan. Instruksi pemrosesan memberikan informasi pada aplikasi XML.
6. Komentar. Komentar adalah sebuah catatan-kaki untuk dokumen XML Anda yang bisa dibaca orang namun diabaikan oleh prosesor XML.

Berikut ini sebuah elemen yang berisi sebuah instruksi pemrosesan dan sebuah komentar.



Elemen Kosong

Anda juga bisa memasukkan sebuah elemen kosong, yakni, elemen tanpa isi ke dalam dokumen Anda. Anda bisa membuat sebuah elemen kosong dengan menempatkan tag-akhir tepat setelah tag-mulai, seperti pada contoh ini:

```
<HR></HR>
```

Atau, Anda bisa menyimpan ketikan dengan menggunakan tag elemen-kosong khusus, seperti tampak di sini:

```
<HR/>
```

Kedua notasi ini memiliki arti yang sama.

Karena sebuah elemen kosong tidak ada isinya, Anda mungkin bertanya apa gunanya. Dalam hal ini ada dua kemungkinan penggunaannya:

1. Anda bisa menggunakan sebuah elemen kosong untuk menyuruh aplikasi XML melakukan tindakan, atau menampilkan sebuah obyek. Contoh HTML elemen kosong BR, yang memberitahu browser untuk menyisipkan sebuah pemisah baris, dan elemen kosong HR, yang menyuruh browser untuk menambahkan sebuah garis pembagi horizontal. Dengan kata lain, keberadaan sebuah elemen dengan nama tertentu tanpa suatu isi bisa memberikan informasi penting ke aplikasi.
2. Sebuah elemen kosong bisa menyimpan informasi melalui atribut, yang

akan Anda pelajari nanti dalam bab ini. (Anda belum melihat elemen dengan atributnya.) Sebuah contoh HTML adalah elemen kosong IMG (image), berisi atribut yang memberitahu prosesor di mana mencari file grafis dan bagaimana menampilkannya.

TIP Seperti yang akan Anda pelajari dalam Bab ini, style sheet bertumpuk bisa memakai sebuah elemen kosong untuk menampilkan sebuah citra. Pada Bab ini, akan Anda pelajari cara menggunakan pengikatan data untuk mengakses atribut sebuah elemen kosong atau tidak kosong. Dalam Bab selanjutnya akan Anda pelajari cara menggu nakan script HTML dan style sheet XSL untuk mengakses elemen (kosong dan tidak-kosong) dan atribut mereka, serta kemudian melakukan aksi yang sesuai.

Membuat Tipe Elemen yang Berbeda

1. Buka sebuah file teks baru yang kosong dengan editor teks Anda, dan ketikkan dokumen XML yang tampak dalam Listing 14-2. Jika Anda mau, Anda bisa menggunakan dokumen Inventory.xml yang Anda buat sebelumnya sebagai titik mulai.
2. Gunakan perintah Save pada editor teks Anda untuk menyimpan dokumen tersebut ke hard disk, namai dengan Inventory03.xml.

   Inventory03.xml

```
<?xml version="1.0"?>
<!-- File Name: Inventory03.xml -->
<?xml-stylesheet type="text/css"
href="Inventory02.css"?>
<INVENTORY> <!-- Inventory of selected 19th
Century
```

```
                American Literature -->
   <BOOK>
      <COVER_IMAGE Source="Huck.gif" />
      <TITLE>The Adventures of Huckleberry
Finn</TITLE>
      <AUTHOR>Mark Twain</AUTHOR>
      <BINDING>mass market
paperback</BINDING>
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK>
      <COVER_IMAGE Source="Leaves.gif" />
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR>Walt Whitman</AUTHOR>
      <BINDING>hardcover</BINDING>
      <PAGES>462</PAGES>
      <PRICE>$7.75</PRICE>
   </BOOK>
   <BOOK>
      <COVER_IMAGE Source="Faun.gif" />
      <TITLE>The Marble Faun</TITLE>
      <AUTHOR>Nathaniel Hawthorne</AUTHOR>
      <BINDING>trade paperback</BINDING>
      <PAGES>473</PAGES>
      <PRICE>$10.95</PRICE>
   </BOOK>
   <BOOK>
      <COVER_IMAGE Source="Moby.gif" />
      <TITLE>
         Moby-Dick
         <SUBTITLE>Or, the Whale</SUBTITLE>
      </TITLE>
```

```
            <AUTHOR>Herman Melville</AUTHOR>
            <BINDING>hardcover</BINDING>
            <PAGES>724</PAGES>
            <PRICE>$9.95</PRICE>
        </BOOK>
    </INVENTORY>
```

Listing 14-2.

3. Dalam Windows Explorer atau dalam jendela folder, klik ganda nama file yang Anda simpan tersebut, Inventory03.xml:

   ```
   Inventory03.xml
   ```

   Internet Explorer 5 akan menampilkan dokumen tersebut seperti tampak di sini:

Dokumen yang Anda masukkan berisi tipe-tipe dari sejumlah elemen dan isi elemen:

1. Sebuah elemen dengan komentar sebagai bagian dari isinya (INVENTORY). Perhatikan browser tidak menampilkan teks komentar

tersebut.

2. Sebuah elemen kosong bernama COVER_IMAGE pada awal masing-masing elemen BOOK. Tujuan elemen ini adalah menyuruh aplikasi XML menampikan image yang disebutkan pada cakupan buku (atribut Source berisi nama dari file citra). Untuk bisa menggunakan elemen semacam itu Anda perlu menampilkan dokumen XML melalui sebuah script dalam halaman HTML, atau sebuah style sheet XSL, ketimbang menggunakan CSS biasa seperti dalam contoh ini.
3. Sebuah elemen (elemen TITLE untuk Moby-Dick) yang berisi data karakter dan elemen anak (SUBTITLE). Perhatikan browser menampilkan baik data karakter maupun elemen anak dalam sebuah garis tunggal, menggunakan format yang sama. (Format CSS yang diberikan pada elemen TITLE diwarisi oleh elemen SUBTITLE.)

Menambahkan Atribut ke Elemen

Dalam tag-awal pada sebuah elemen, atau dalam sebuah tag elemen kosong, Anda bisa memasukkan satu atau beberapa spesifikasi atribut. Spesifikasi atribut adalah pasangan nilai nama yang berhubungan dengan elemen tersebut. Sebagai contoh, elemen PRICE berikut ini memasukkan sebuah atribut bernama Type, yang menunjuk nilai retail:

```
<PRICE Type="retail">$l0.95</PRICE>
```

Untuk buku lainnya, atribut ini, sebagai contoh, bisa dijadikan wholesale.

Elemen BOOK berikut ini memasukkan dua atribut, Category dan Display

```
<BOOK Category="fiction"Display="emphasize">
<TITLE>The MarbleFaun</TITLE>
<AUTHOR>Nathaniel Hawthorne</AUTHOR>
```

```
<BINDING>trade paperback</BINDING>
<PAGES>473</PAGES>
<PRICE>$10.95</PRICE>
<!BOOK>
```

Elemen kosong berikut ini memasukkan sebuah atribut bernama Source, yang menandakan nama file yang berisi citra yang akan ditampilkan:

```
<COVER_IMAGE Source="Faun.gif"I>
```

Penambahan sebuah atribut memberikan satu cara alternatif untuk memasukkan informasi dalam sebuah elemen. Pada umumnya, Anda menempatkan sekumpulan data elemen yang ingin Anda tampilkan dalam isi elemen. Anda menggunakan sejumlah atribut untuk menyimpan berbagai macam properti elemen, tidak harus dimaksudkan untuk ditampilkan, seperti halnya sebuah kategori atau instruksi menampilkan. Spesifikasi XML tidak membedakan secara kaku tentang tipe informasi yang mesti disimpan dalam atribut atau isi.

Aturan untuk Pembuatan Atribut

Seperti yang bisa Anda lihat, sebuah spesifikasi atribut terdiri atas nama atribut diikuti dengan sebuah tanda sama-dengan, kemudian sebuah nilai atribut. Anda bisa memilih sembarang nama atribut yang diinginkan, asalkan Anda mengikuti aturan berikut ini:

1. Nama tesebut harus dimulai dengan sebuah huruf atau tanda garis- bawah (_), diikuti dengan atau tanpa huruf apa pun, digit, periode (.), hyphen (-), atau beberapa garis-bawah.
2. Spesifikasi XML menyatakan bahwa nama tipe-elemen diawali dengan awalan 'xml" (dalam berbagai kombinasi huruf besar atau kecil) dijadikan sebagai "standarisasi ". Walaupun Internet Explorer 5 tidak memaksakan pembatasan ini, lebih baik tidak menggunakan awalan ini untuk

menghindari masalah di masa mendatang.

3. Nama atribut khusus bisa muncul hanya setelah elemen tag-mulai dan tag-kosong yang sama.

Sebagai contoh, nama atribut dalam tag mulai berikut ini adalah sah:

```
<NATION FleName="Waldo.ani">
<LIST lstPlace="Sam">
<ENTRYZip.Code="94941" >
```

Nama atribut berikut adalah ilegal:

```
< !- - Duplicated attribute name in same
tag: - - >
<ANIMATION
FileName="Waldol.ani"FileName="Waldo2.ani">
<LISTlstPlace="Sam"><!- - Digitnot allowed
as first character- - >
<ITEMA:Category="cookware"><!- -
InIES.allowed only if you’ve decla red A as
anamespace - - >
```

Aturan untuk Nilai Atribut yang Sah

Nilai yang Anda berikan ke suatu atribut adalah sebuah rangkaian karakter yang ditutup dengan tanda petik, yang dikenal sebagai quoted string atau literal. Anda bisa memberikan sembarang nilai literal ke suatu atribut, asalkan Anda membaca aturan ini:

1. String dapat ditutup menggunakan dua tanda petik tunggal (') atau petik-ganda (").
2. String tidak bisa memuat karakter petik sama yang dipakai untuk menutupnya.
3. String bisa memu at acuan karakter atau acuan ke entitas internal umum
4. String tidak bisa memasukkan karakter < (pengurai akan menolak

karakter ini saat menjalankan tanda XML).

5. String tidak bisa memasukkan karakter &, kecuali untuk mengawali sebuah karakter atau acuan entitas.

Anda telah mengetahui contoh spesifikasi atribut yang legal. Spesifikasi atribut berikut ini adalah yang ilegal:

```
<EMPLOYEE Status=""downsized"">
<! -- Can't use delimiting quote within
string. - - >
<ALBUM Type="<CD>"><! - - Can't use<with in
string. - - >
<WEATHER Forecast=Cold&Windy">
<! -- Can't use&except to start areference.
- - >
```

Jika Anda ingin memasukkan tanda petik ganda (") dalam nilai atribut, Anda bisa menggunakan petik tunggal (') untuk menutup string tersebut, seperti dalam contoh ini:

```
<EMPLOYEE Status='"down sized"'>
< !- Legal attribut evalue.->
```

Begitu pula, untuk memasukkan tanda petik tunggal dalam nilai yang ada, tutup menggunakan petik ganda:

```
<CANDIDATE name="W.T.'Bill'Bagley">
< !- Legal attribute value ->
```

TIP Anda bisa menukar ketentuan karakter dan memasukkan sembarang karakter ke dalam sebuah nilai atribut menggunakan sebuah acuan karakter, atau jika tersedia sebuah acuan entitas umum yang telah ada. Saya akan jelaskan karakter dan acuan entitas umum.

Jika Anda membuat dokumen rapi yang tidak mempunyai deklarasi tipe dokumen (seperti yang telah Anda kerjakan dalam bab ini), Anda bisa

memberikan pada atribut sembarang nilai yang tunduk pada aturan yang telah dijelaskan di atas. Anda bisa mendefini sikan sebuah atribut yang da- pat diberikan jika nilai tersebut "yes" atau "no." Sehingga, satu keuntungan penyimpanan sejumlah tipe informasi dalam atribut elemen dibandingkan dalam isi adalah Anda bisa mencoba lebih jauh kontrol atas tipe data yang bisa diberikan ke atribut tersebut, dan menyuruh pengurai menerapkan aturan tipe ini. (Seperti yang akan Anda lihat dalam Bab sebelumnya, spesifikasi dasar XML tidak menyediakan cara untuk membatasi tipe data karakter dalam suatu elemen.)

Mengubah isi menjadi Atribut

1. Buka sebuah file teks baru yang kosong dengan editor teks Anda, dan ketikkan dokumen XML yang tampak dalam Listing 14-2. Jika Anda mau, Anda bisa menggun akan dokumen Inventory.xml yang Anda buat sebelumnya sebagai titik mulai.
2. Gunakan perintah Save pada editor teks Anda untuk menyimpan dokumen ke harddisk, namai dengan Inventory04.xml.

   Inventory04.xml

```
<?xml version="1.0"?>
<!-- File Name: Inventory04.xml -->
<?xml-stylesheet type="text/css"
href="Inventory02.css"?>
<INVENTORY>
   <BOOK Binding="mass market paperback">
      <TITLE>The Adventures of Huckleberry
Finn</TITLE>
      <AUTHOR Born="1835">Mark
Twain</AUTHOR>
```

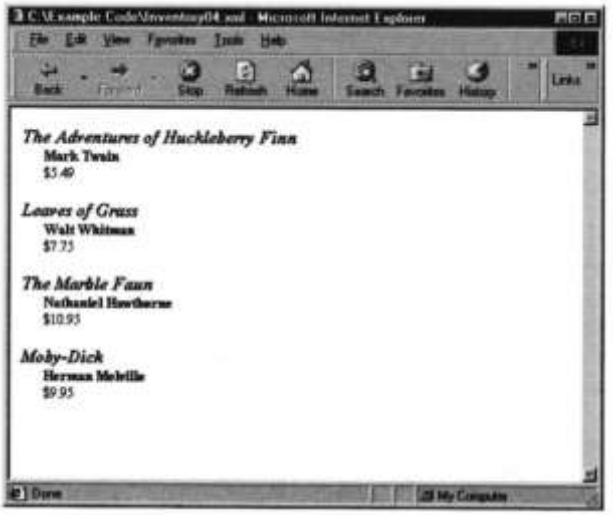
```
      <PAGES>298</PAGES>
      <PRICE>$5.49</PRICE>
   </BOOK>
   <BOOK Binding="hardcover">
      <TITLE>Leaves of Grass</TITLE>
      <AUTHOR Born="1819">Walt
Whitman</AUTHOR>
      <PAGES>462</PAGES>
      <PRICE>$7.75</PRICE>
   </BOOK>
   <BOOK Binding="trade paperback">
      <TITLE>The Marble Faun</TITLE>
      <AUTHOR Born="1804">Nathaniel
Hawthorne</AUTHOR>
      <PAGES>473</PAGES>
      <PRICE>$10.95</PRICE>
   </BOOK>
   <BOOK Binding="hardcover">
      <TITLE>Moby-Dick</TITLE>
      <AUTHOR Born="1819">Herman
Melville</AUTHOR>
      <PAGES>724</PAGES>
      <PRICE>$9.95</PRICE>
   </BOOK>
</INVENTORY>
```

Listing 14-3.

3. Dalam Windows Explorer atau jendela folder, klik-ganda nama file yang telah Anda simpan tersebut, Inventory04.xml:

   ```
   Inventory04.xml
   ```

   Internet Explorer 5 akan menampilkan dokumen tersebut seperti ter lihat di sini:



Dokumen yang telah Anda ketik berdasarkan pada Inventory.xml, yang Anda buat dalam latihan sebelumnya. Sebagai tambahan agar elemennya jadi lebih kecil dari Inventory .xml, dokumen baru itu mempunyai dua modifikasi yang menggambarkan penggunaan atribut:

1. Pada setiap elemen BOOK, informasi pengikatan buku disalin dari isi (dalam bentuk elemen tersarang BINDING) menjadi sebuah atribut bernama Binding. Anda bisa melakukan konversi ini jika, misalnya, Anda ingin menyimpan tipe pengikatan, namun tidak ingin memperlihatkannya bersama informasi buku lainnya saat menampilkan dokumen menggunakan CSS. (Dalam gambar di atas, lihatlah Internet Explorer 5 tidak menampilkan nilai atribut.)
2. Sebuah atribut yang bernama Born telah ditambahkan pada masing-masing elemen AUTHOR untuk menyimpan tanggal lahir pengarangnya Ini adalah sebuah contoh informasi kurang penting yang mungkin ingin Anda simpan namun tidak perlu ditampilkan. Satu cara untuk

menyembunyikan informasi semacam ini dan untuk menandai bahwa ia tidak penting adalah memberikannya ke sebuah atribut ketimbang menempatkannya dalam isi dari elemen.

Ini semua hanyalah sebagian kecil dari beberapa kemungkinan penggunaan atribut.

C. PENUTUP

Ringkasan

Dokumen XML yang valid perlu memasukkan sebuah komponen tambahan yang tidak tercantum dalam dokumen contoh pada Listing 14-1: sebuah deklarasi tipe dokumen, yang bisa Anda tempatkan di mana saja pada pengantamya, di luar tanda lainnya, setelah deklarasi XML. Deklarasi tipe dokumen mendefinisikan struktur dokumen XML yang valid.

Suatu elemen yang memuat satu atau lebih elemen yang dikenal sebagai elemen induk. Sebuah elemen yang terkandung secara langsung dalam induk (misalnya TITLE dalam BOOK) dikenal seba- gai elemen-anak, subelemen, atau elemen tersarang pada induk tersebut.

Sesuai dengan spesifikasi XML, colom (:) dalam suatu nama elemen ditetapkan untuk pemberian namespaces. Namespaces membedakan elemen dengan nama yang sama; saya akan bahas mereka dalam "Menyisipkan Elemen HTML ke dalam Dokumen XML dan Menggunakan Namespaces". Internet Explorer memungkinkan Anda menyisipkan sebuah colon dalam suatu nama elemen hanya jika ia mengikuti sebuah namespace yang telah Anda deklarasikan dalam dokumen tersebut. Sebagai contoh, A: Section akan menjadi sah hanya jika Anda telah mendeklarasikan A sebagai sebuah

namespace.

Pengurai XML memeriksa sebuah data karakter elemen untuk tanda XML. Anda tidak bisa menyisipkan < atau & atau string]] > sebagai bagian dari data karakter, karena pengurai akan menafsirkan < sebagai awal sebuah elemen tersarang, & seba- gai awal sebuah entitas atau acuan karakter, dan]] > sebagai akhir dari bagian CDATA. Jika Anda ingin menyisipkan < atau & sebagai bagian dari data karakter, Anda bisa menggunakan sebuah bagian CDATA. Anda juga bisa menyisipkan sembarang karakter (termasuk yang tidak ada pada keyboard Anda) menggunakan acuan karakter, dan Anda bisa menyisipkan sejumlah karakter (seperti misalnya < atau &) menggunakan acuan entitas umum yang telah ada.

Dokumen yang Anda ketik menggunakan CSS yang dinamai lnventory02.css yang Anda buat dalam latihan terdahulu. (lni tampak dalam Listing 2-4 dan pada CD penyerta.) Pastikan file style sheet ini ada dalam folder yang sama dengan lnventory03.xml.

Saat Anda menampilkan sebuah dokumen XML menggunakan CSS, browser tidak menampilkan atribut atau nilai mereka. Menampilkan sebuah dokumen XML menggunakan pengikatan data, script dalam halaman HTML, atau sebuah style sheet XSL, memungkinkan Anda mengakses atribut dan nilai mereka, dan untuk menampilkan nilai atau menjalankan aksi yang sesuai lainnya.

Sesuai dengan spesifikasi XML, penggunaan tanda bagi dalam sebuah nama atribut dicadangkan untuk perancangan namespaces. Namespaces dipakai untuk membedakan atribut yang mempunyai nama sama; mereka dibahas dalam "Menyisipkan Elemen HTML ke dalam Dokumen XML dan

Menggunakan Namespaces". Internet Explorer 5 memungkinkan Anda menyisipkan sebuah tanda bagi dalam suatu nama atribut, hanya jika ia mengikuti namespace yang telah Anda deklarasikan dalam dokumen tersebut. Sebagai contoh, A: Category akan menjadi sah hanya jika Anda telah mendeklarasikan A sebagai sebuah namespace.

Dokumen yang telah Anda ketikkan menggunakan CSS yang dinamai Inventory02.css yang telah Anda buat dalam latihan sebelumnya (ini diberikan dalam Listing 2-4). Pastikan file style sheet ini ada dalam folder yang sama dengan lnventory04.xml.

Pertanyaan

1.Tuliskan bentuk dokumen XML Minimalis?

2.Sebutkan dan jelaskan anatomi Elemen?

3.Sebutkan dan jelaskan tipe isi elemen?

4.Apa itu elemen kosong?

5.Sebutkan aturan untuk pembuatan atribut?

DAFTAR PUSTAKA

1. XML, Step by Step, Michael J. Young,
2. Bulding XML Web Service for the Microsoft .NET Platform, by Microsoft Corporation Published by Microsoft Press
3. Microsoft ASP.NET, Step by Step, G. Andrew Duthie
4. Pemrograman Web Service dan SOA, Wiranto Herry Utomo, Penerbit Andi 2016
5. Service-Oriented Architecture Second Edition by Thomas Erl, 2018

6. Buku Adobe Flash + XML = Rich Multimedia Application, Andy Sunyoto, 2010
7. Web Dengan HTML & XML, Aji Supriyanto, Graha Ilmu, 2018
8. Definisi Atau Pengertian Komunikasi Data Lengkap. http://www.pengertianku.net/2014/09/definisi - atau - pengertian komunikasi – data - lengkap.html. Diakses 19 Agustus 2016.
9. Wahli, U., Burroughs, O., Cline, O., Tung, L. 2006. Service Handbook for WebSphere Application Server 6.1. http://www.redbooks.ibm.com/redbooks/pdfs/sg247257.pdf. Diakses 19 Agustus 2016.
10. Gottshchalk, Petter .2002. A Stages of Growth Model for Knowledge Management Technology in Law Firms. Jurnal Of Information, Law and Technology (JILT) 2002.
11. Dykes, Lucinda. Tittel. Ed. 2005. XML for Dummies. 4th Edition. Wiley Publishing, Inc. Canada.
12. JSON. https://id.wikipedia.org/wiki/JSON. Diakses 19 Agustus 2016.
13. Pengenalan JSON, http://www.json.org/json-id.html. Diakses 19 Agustus 2016.
14. Nurseitov, N., Paulson, M., Reynolds,R., Izurieta, C. 2009. Comparison of JSON and XML Data Interchange Formats: A Case Study. Department of Computer Science, Montana State University.Bozeman, USA .

ISBN 978-623-6141-27-4