เว็บเซิร์ฟเวอร์สำหรับการวัดระยะไกล

นางสาวเบญจมาศ มานะทวีวัฒน์
นางสาวอรรถยา จุลประภา

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาฟิสิกส์ประยุกต์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2541

Telemetering Web Server

Miss. Benjamas Manataweewat

Miss.Attaya Julprapa

A Special Project Submitted in Partial Fulfillment fo the

Requirement for the Degree of Bachelor of Science

Department of Applied Physics

Faculty of Science

King Mongkut's Institue of Technology Ladkrabang

1998

หัวข้อโครงงานพิเศษ	เว็บเซิร์ฟเวอร์สำหรับการวัดระยะไกล
โดย	นางสาวเบญจมาศ มานะทวีวัฒน์
	นางสาวอรรถยา จุลประภา
ภาควิชา	ฟิสิกส์ประยุกต์
อาจารย์ที่ปรึกษา	ผู้ช่วยศาสตราจารย์วิชิต ศิริโชติ

ภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับโครงงานพิเศษฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต

..... หัวหน้าภาควิชาฟิสิกส์ประยุกต์
(รองศาสตราจารย์สุรพล รักวิจัย)

คณะกรรมการโครงงานพิเศษ

..... ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์วิชิต ศิริโชติ)

..... กรรมการ
(อาจารย์บดินทร์ ดำรงศักดิ์)

..... กรรมการ
(อาจารย์ภัทรียา กิติเดชาชาญ)

..... กรรมการ
(อาจารย์ภูมินทร์ จินดาจิธาวัฒน์)

ลิขสิทธิ์ของภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อโครงงานพิเศษ เว็บเซิร์ฟเวอร์สำหรับการวัดระยะไกล
โดย นางสาวเบญจมาศ มานะทวีวัฒน์
นางสาวอรรถยา จุลประภา
อาจารย์ที่ปรึกษา ผู้ช่วยศาตราจารย์วิชิต ศิริโชติ
ภาควิชา ฟิสิกส์ประยุกต์
ปีการศึกษา 2541

บทคัดย่อ

โครงงานพิเศษนี้ได้ทำการพัฒนาขึ้นสำหรับวัดค่าแรงดันไฟฟ้าภายในห้องทดลองที่เปลี่ยนแปลงไปในแต่ละวัน โดยแสดงผลบนเว็บบราวเซอร์ผ่านทางระบบเครือข่ายอินเตอร์เนต หลักการทำงานของโครงงานนี้เริ่มจากการเก็บรวบรวมค่าแรงดันไฟฟ้าที่วัดได้จากเครื่องดิจิตอลมัลติมิเตอร์ ไว้ในเว็บเซิร์ฟเวอร์ซึ่งเชื่อมต่อกับเครื่องมัลติมิเตอร์ทางพอร์ทอนุกรม โดยในส่วนนี้ได้ใช้ภาษาวิชวลเบสิคในการพัฒนาโปรแกรม สำหรับการแสดงผลสามารถเรียกดูได้จากทางเว็บบราวเซอร์ ซึ่งโปรแกรมที่ใช้ในการแสดงผลจากเว็บเซิร์ฟเวอร์บนเว็บบราวเซอร์นั้นพัฒนาจากโปรแกรมภาษาจาวา ซึ่งโปรแกรมจะให้ผลลัพธ์ออกมาในรูปของตารางแต่จะแสดงเพียงค่าโวลต์ที่เปลี่ยนแปลงไปจาก 220 โวลต์ ไปในช่วง +/-10 เปอร์เซ็นต์

Special Project Title	Telemetering Web Sever	
Name	Miss.Benjamas	Manataweewat
	Miss.Attaya	Julprapa
Special Project Advisor	Asst.Prof.Wichit	Sirichote
Department	Applied Physics	
Academic Year	1998	

Abstract

An Internet web server that interfaced with a measuring instrument, namely Telemetring Web Server, has been developed. The server is capable of providing recorded data and a public domain web page. The recorded data can be remotely retrieved through a web browser program. A digital multimeter, Fluke-45 with RS-232 interfacing , has been connected to the server. Demonstration of operation of the server was the measurement and recording of a main line voltage. Reading is performed every 1 minute, if there was a line fluctuation larger than $\pm$10% of RMS value, the server will save the reading and current time. The language that uses in this project was a Visual basic for interfacing with RS-232 and a JAVA for web browsering.

กิตติกรรมประกาศ

โครงงานพิเศษนี้สำเร็จลุล่วงได้ด้วยดี เนื่องจากความอนุเคราะห์จากบุคคลหลายฝ่ายดังนี้

ผศ.วิชิต ศิริโชติ	ผู้ให้โอกาสและแนวทางในการดำเนินการโครงงานพิเศษ และเครื่องมัลติมิเตอร์
ผศ.จิรวัฒน์ ปานกลาง	ผู้ให้คำปรึกษาและอำนวยความสะดวกในด้านต่างๆ
อาจารย์ยวิสันต์ ตั้งวงษ์เจริญ	ผู้ให้คำปรึกษาและแนวทางในการใช้โปรแกรมภาษาจาวา และ Java Servlet
คุณพิเศษ วินิจฉัยกุล	ผู้เอื้อเฟื้อสถานที่ และอุปกรณ์ในการทำงาน
คุณวิทิต อินทรหะ	นักศึกษาปริญญาโท คณะเทคโนโลยีสารสนเทศ ผู้ที่ช่วยเรียบเรียงถ้อยคำในการเขียนรายงาน
เพื่อนรุ่น14	ที่คอยให้กำลังใจตลอดมา

ท้ายสุดนี้ขอขอบคุณทุกท่าน ทั้งที่ได้กล่าวนามและไม่ได้กล่าวนามด้วยความจริงใจ

เบญจมาศ และ อรรถยา

สารบัญเรื่อง

สารบัญรูป

บทที่ 1
บทนำ

เว็บเซิร์ฟเวอร์สำหรับการวัดระยะไกล เป็นเว็บเซิร์ฟเวอร์ที่ติดตั้งกับเครื่องวัด Digital Multimeter (Fluke45) เพื่อรับข้อมูลที่ได้จากการวัดค่าแรงดันไฟฟ้าโดยส่งผ่านข้อมูลทางพอร์ทอนุกรม RS-232 เว็บเซิร์ฟเวอร์นี้ทำหน้าที่เก็บรวมรวมข้อมูลที่ได้จากการวัด การแสดงผลนั้นจะแสดงผ่านบริการของเวิลด์ไวด์เว็บ (WWW) ซึ่งจะต้องมีซอร์ฟแวร์ที่ทำหน้าที่อ่านข้อมูลที่เก็บไว้ในเว็บเซิร์ฟเวอร์โดยเรียกว่า บราวเซอร์(Browser) หรือ เว็บบราวเซอร์ (web Browser) เช่น Netscape Navigator หรือ Microsoft Internet Explorer ดังนั้นการเรียกดูข้อมูลที่ได้จากการวัดนั้นจึงไม่ถูกจำกัดอยู่แค่เฉพาะเครื่องวัดแต่สามารถเรียกดูจากที่ใดก็ได้เพียงแต่ต้องมีการเชื่อมต่อกับระบบเครือข่ายคอมพิวเตอร์ หรือ Internet

1.1 จุดประสงค์ของโครงงาน

1. เพื่อศึกษาพื้นฐานและลักษณะทั่วไปของ WWW และสามารถทำการส่งค่าแรงดันไฟฟ้า โดยใช้ความสามารถของระบบ WWW
2. เพื่อศึกษาการติดตั้งระบบ WWW เซิร์ฟเวอร์
3. เพื่อศึกษาหลักการทำงานของระบบสื่อสาร HTTP
4. เพื่อศึกษาการส่งผ่านข้อมูลทางพอร์ทอนุกรม RS-232
5. เพื่อสามารถทำการบันทึกค่าโวลต์ที่เปลี่ยนแปลงไปตามค่าที่กำหนด

1.2 วิธีดำเนินการ

1. ศึกษาการติดตั้งเซิร์ฟเวอร์
2. ศึกษาโปรแกรมภาษา JAVA
3. ศึกษาการรับส่งข้อมูลทาง RS-232
4. ศึกษาโปรแกรมภาษา Visual Basic
5. ทดลองติดตั้งเซิร์ฟเวอร์
6. ทดลองรับค่าแรงดันจากมัลติมิเตอร์ผ่านทาง RS-232
7. ทดลองเชื่อมต่อข้อมูลกับเซิร์ฟเวอร์
8. ทดลองเชื่อมเซิร์ฟเวอร์ที่ติดตั้งกับระบบเครือข่ายคอมพิวเตอร์
9. สรุปผลการทดลอง

1.3 ประโยชน์ที่ได้รับ

1. การประยุกต์ใช้ในการวัดค่าแรงดันไฟฟ้าจากระยะไกล โดยสามารถแสดงผลได้ทุกสถานที่ที่มีการเชื่อมต่อกับระบบเครือข่ายอินเตอร์เนต
2. เป็นพื้นฐานของการควบคุมอุปกรณ์ระยะไกล โดยผ่านระบบเครือข่ายอินเตอร์เนต

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 การติดต่อสื่อสารผ่านระบบเครือข่าย

เครือข่ายคอมพิวเตอร์ (Computer Network) เป็นกลุ่มคอมพิวเตอร์ที่นำคอมพิวเตอร์มาเชื่อมต่อกันหลายๆเครื่อง โดยใช้สัญญาณเป็นตัวเชื่อมต่อและคอมพิวเตอร์ที่เชื่อมต่อกันนี้สามารถสื่อสารและสามารถรับและส่งไฟล์ผ่านกันได้ โดยในปัจจุบันบริษัทและหน่วยงานต่างๆจะมีเครือข่ายคอมพิวเตอร์ใช้งานกันอยู่โดยสามารถแบ่งเครือข่ายคอมพิวเตอร์ได้เป็น 2 ประเภทดังนี้

1. เครือข่ายคอมพิวเตอร์ขนาดเล็ก หรือ LAN (Local Area Network) เป็นเครือข่ายคอมพิวเตอร์ที่ใช้งานในระดับท้องถิ่น

2. เครือข่ายคอมพิวเตอร์ระยะไกล หรือ WAN (Wide Area Network) เป็นการนำเอาเครือข่ายคอมพิวเตอร์ขนาดเล็กมาเชื่อมต่อกันหลายๆวงซึ่งเครือข่ายบางวงก็อยู่ใกล้กันบางวงก็อยู่ห่างกันคนละที่เช่นเชื่อมต่อเครือข่ายคอมพิวเตอร์ของบริษัทคนละสาขาในต่างจังหวัดเข้าด้วยกันโดย WAN นี้จะมีขนาดของเครือข่ายที่แตกต่างกันไป

2.1.1 เครือข่ายอินเตอร์เนต (internet network)

เครือข่ายคอมพิวเตอร์ในโลกนี้มีหลากหลายชนิด ซึ่งการนำเครือข่ายคอมพิวเตอร์เหล่านี้มาเชื่อมต่อกันโดยกำหนดข้อตกลงในการสื่อสารขึ้นมาตัวหนึ่งมีชื่อว่า TCP/IP เพื่อให้เครือข่ายคอมพิวเตอร์เหล่านี้สามารถสื่อสารกันได้ทั่วโลกนั้นเราเรียกว่า อินเตอร์เนต (internet)

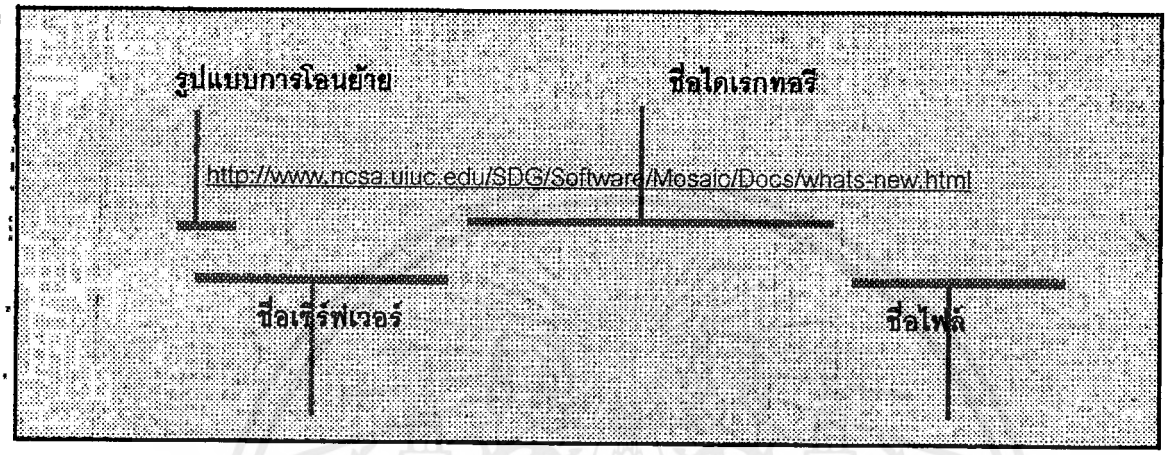
หลักการของTCP/IP (Transmission Control Protocol/Internet Protocol) คือการแบ่งข้อมูลออกเป็นชิ้นเล็กๆเรียกว่า แพคเกท (Packet) แต่ละแพคเกทจะมีข้อมูลเพื่อบอกจุดมุ่งหมายปลายทาง จากนั้นข้อมูลแต่ละแพคเกทจะถูกส่งไปยังเส้นทางต่างๆโดยไม่จำเป็นที่จะต้องไปในเส้นทางเดียวกัน โดยทุกตัวจะถูกส่งไปยังเส้นทางที่ใกล้ที่สุดดีที่สุดด้วยอุปกรณ์ที่เรียกว่า เราเตอร์ (Router) เป็นตัวจัดเส้นทางให้และเมื่อข้อมูลทุกแพคเกทถูกส่งไปถึงปลายทางมันจะถูกรวมเป็นข้อมูลชิ้นเดียวดังเดิมได้อีกครั้งหนึ่งโดยเมื่อเส้นทางใดถูกตัดขาด เราเตอร์จะหาเส้นทางใหม่ให้ข้อมูลในทันที และถ้าแพคเกทข้อมูลเกิดความเสียหายเราเตอร์ก็จะบอกให้คอมพิวเตอร์ต้นทางส่งแพคเกทไปให้ใหม่อีกครั้งหนึ่ง

รูปแบบการสื่อสารแบบ TCP/IP จะมีการกำหนดหมายเลขประจำตัวให้กับเครื่องคอมพิวเตอร์แต่ละเครื่องที่อยู่ในเครือข่าย โดยที่คอมพิวเตอร์ทุกตัวจะมีตัวเลขไม่ซ้ำกันเลย (Unique) โดยหมายเลขประจำตัวนี้เรียกว่า IP Address โดยจะมีรูปแบบเป็นตัวเลขสี่ชุดย่อยเรียงกันโดยใช้จุดคั่นเช่น 161.246.10.21 สำหรับตัวเลขในแต่ละชุดย่อยจะอยู่ในช่วง 0-255 จะไม่มีค่าเกินกว่านี้ซึ่ง IP Address นี้จะเป็นตัวกำหนดจุดเริ่มต้นและจุดปลายของการรับส่งข้อมูลแบบโปรโตคอล TCP/IP

2.1.2 World Wide Web (WWW) บริการเวิลด์ไวด์เว็บเป็นหนึ่งในบริการของอินเตอร์เน็ต โดยผู้ใช้สามารถอ่านและค้นหาข้อมูลโดยการคลิกเมาส์เปิดไปยังเว็บเพจที่ต้องการได้ซึ่งในเว็บเพจแต่ละหน้านั้น เราสามารถเชื่อมโยงกันเพื่อเข้าสู่เว็บเพจใดๆในโลกนี้ได้โดยการลิงค์ (Link) ซึ่งเป็นจุดเด่นในการนำเสนอข้อมูลของเวิลด์ไวด์เว็บ สำหรับเว็บเพจแต่ละหน้าที่แสดงขึ้นมาได้นั้นจะต้องถูกเขียนขึ้นด้วยภาษา HTML (HyperText

Markup Language)และไฟล์โค้ดของภาษาเหล่านี้จะถูกนำไปเก็บไว้ที่เครือข่ายคอมพิวเตอร์ที่สามารถให้บริการเวิลด์ไวด์เว็บได้ซึ่งเรียกว่า เว็บไซต์ (Web Site) หรือ เว็บเซิร์ฟเวอร์ (Web Server) และวิธีการเรียกใช้บริการเวิลด์ไวด์เว็บจะต้องมีโปรแกรมที่สามารถอ่านโค้ดภาษา HTML ที่เก็บในเว็บเซิร์ฟเวอร์ได้โปรแกรมนี้เรียกว่า เว็บบราวเซอร์ (Web Browser) ซึ่ง Netscape Navigator ก็คือเว็บบราวเซอร์ตัวหนึ่งนั่นเอง

สำหรับการเชื่อมโยงข้อมูลของโปรแกรมเว็บบราวเซอร์นี้ได้ถูกกำหนดให้เป็นแบบการสืบค้นแหล่งข้อมูลเรียกว่า URL (Umiform Resource Locators) โดยมีรูปแบบดังตัวอย่างที่ได้แสดงในรูปที่ 1.1



รูปที่ 2.1 แสดงรูปแบบของรหัสสืบค้นข้อมูล URL

รูปแบบของ URL เป็นรูปแบบมาตรฐานสำหรับระบบ WWW โดยกำหนดให้ขึ้นต้นด้วยคำว่า HTTP (Hypertext Transfer Protocol) ซึ่งหมายถึงการโต้ตอบเพื่อการสื่อสารกันแบบ ไฮเปอร์เท็กซ์ สำหรับคำจำกัดความของไฮเปอร์เท็กซ์คือ คำหรือวลีที่ประกอบด้วยคำอธิบายซ่อนอยู่เบื้องหลัง .หากใช้เมาส์ดับเบิลคลิกไปที่คำหรือวลีนั้น ๆ ก็จะปรากฏคำอธิบายหรือรายละเอียดของคำหรือวลีดังกล่าว ซึ่งคำอธิบายที่ปรากฏขึ้นนั้นได้มาจากการเชื่อมโยงไฟล์ข้อมูลจากแหล่งต่าง ๆ ซึ่งอาจจะเป็นข้อมูลบนคอมพิวเตอร์ของผู้ใช้เอง หรือข้อมูลจากเซิร์ฟเวอร์อื่น ๆ สำหรับรูปแบบ URL ที่แสดงในรูปที่ 1.1 นั้นมีความหมายว่าโปรแกรมจะทำการเชื่อมโยงข้อมูลโดยระบบโต้ตอบข้อมูลตามมาตรฐานของ WWW ซึ่งถูกกำหนดด้วยเครื่องหมาย http:// โดยเชื่อมโยงเข้ากับ WWW เซิร์ฟเวอร์ชื่อ www.ncsa.uiuc.edu เพื่อการโอนย้ายข้อมูลจากไฟล์ชื่อ whats-new.html ซึ่งอยู่ในไดเรกทอรี /SDG /Software /Mosaic /Docs /

2.2 โครงสร้างและการเขียน Home page ด้วยภาษา HTML

HTML (HyperText Markup Language) เป็นภาษาที่มีพื้นฐานมาจากภาษา SGML (Standard Generalized Markup Language) โดยมีความสามารถในการเชื่อมต่อกับเอกสารอื่น ๆ ในลักษณะไฮเปอรเท็กซ์ (Hypertext) ที่ทำให้การเข้าถึงเอกสารอื่น ๆ ในเครื่องเดียวกันหรือเอกสารบนเครื่องอื่น ๆ ในสถานที่ต่าง ๆ จากภายในเอกสารนั้นทำได้อย่างรวดเร็ว และเอกสารจะแสดงออกมาเป็นหน้า เอกสารในหน้าแรกซึ่งแสดงทุกครั้งที่เริ่มเข้าสู่โปรแกรม จะถูกเรียกว่า Home page ซึ่งเอกสารที่เขียนขึ้นมาตามรูปแบบของ HTML นั้น สามารถนำเสนอได้ในลักษณะของ ข้อความ ภาพนิ่ง ภาพเคลื่อนไหว หรือเสียงได้อีกด้วย แต่ในการที่จะแสดงเอกสารออกมาได้นั้นจำเป็นต้องอาศัย Application Program ที่สามารถ Run อยู่บนระบบ www ของอินเตอร์เน็ตได้ ที่เรียกว่า

บราวเซอร์ ซึ่งโปรแกรมบราวเซอร์นี้เป็นส่วนของเครื่อง Client เพื่อเข้าไปเรียกดูข้อมูลจาก web Server ต่าง ๆ โดยการใช้ URL เพื่อกำหนดส่วนที่อยู่ของข้อมูล

2.2.1โครงสร้างของ HTML

ลักษณะทั่วไปของ HTML ก็เช่นเดียวกับการเขียนโปรแกรมที่มีรูปแบบของคำสั่งหรือฟังก์ชั่นที่ใช้โดยทั่วไป ซึ่งคำสั่งที่ใช้ใน HTML จะอยู่ในรูป

<Tag_Name> affected text </ Tag_Name>

Tag แต่ละตัวประกอบขึ้นจาก เครื่องหมายน้อยกว่า (<) ตามด้วยชื่อของ Tag นั้นและปิดท้ายด้วยเครื่องหมายมากกว่า (>) การใช้ Tag จะใช้เป็นคู่ ๆ โดย Tag ตัวแรกจะเรียกว่า Tag เริ่มต้น (Starting Tag) ส่วนที่เหลือจะเรียกว่า Tag จบ (Ending Tag) โดย Tag จบจะต้องมีเครื่องหมาย "/" อยู่หน้าชื่อของ Tag จบ ต่อจากเครื่องหมายน้อยกว่า เช่น

< title> My Useful Document </title>

ซึ่งเป็นการบอกแก่ Browser ให้รู้ว่าข้อความ My Useful Document นี้เป็นชื่อเรื่องของเอกสารนี้เท่านั้น

หมายเหตุ

มีบาง Tag ที่ไม่ต้องใช้เป็นคู่ คือไม่ต้องมี Tag จบ เช่น Tag ซึ่งเป็นการขึ้นบรรทัดใหม่โดยที่ไม่ต้องใช้ Tag

2.2.2 รูปแบบและหน้าที่ของ Tag

Title

รูปแบบ : <Title> Name of Document </Title>

หน้าที่ : ใช้ในการกำหนดชื่อของเอกสาร

หมายเหตุ

ชื่อของ Title จะมีความยาวจำกัดประมาณ 6 คำ และจะถูกแสดงแยกต่างหากจากส่วนอื่น ๆ ของ Document โดยจะถูกแสดงไว้ในส่วน Windows title bar ของ Browser นั้น

Headings

รูปแบบ : <Hy> Text of Heading </Hy>

โดยที่ y เป็นตัวเลขตั้งแต่ตัวเลข 1 ถึง 6

หน้าที่ : ทำให้ข้อความที่เป็นหัวข้อ แตกต่างจากข้อความธรรมดา ซึ่งหัวข้อของ HTML Document มีทั้งหมด 6 ระดับ เริ่มจากระดับที่ 1 จนถึงระดับที่ 6 โดยในส่วนของ Tag Heading นี้จะแสดง text ออกมาในรูปแบบของอักขระขนาดใหญ่และหนา โดยที่ระดับที่ 1 จะมีขนาดใหญ่สุด และลดขนาดลงไปเรื่อย ๆ จนถึงระดับที่ 6 ซึ่งเป็นระดับที่เล็กที่สุด

ตัวอย่าง

<H1> Heading1 </H1>

<H2> Heading2 </H2>

ผลลัพธ์

Heading1

Heading2

Paragraph

HTML Document จะแตกต่างจาก Document ของ Word Processor ทั่วไปคือ HTML Document จะไม่ถือว่า Carriage return สำคัญ ดังนั้นการทำ Word Wrapping หรือการตัดคำเพื่อขึ้นบรรทัดใหม่โดยอัตโนมัติจะสามารถเกิดขึ้นได้ทุกจุดใน HTML Document

รูปแบบ : <p>

หน้าที่ : ใช้ในการขึ้นบรรทัดใหม่ โดยเว้นบรรทัด 1 บรรทัด

หมายเหตุ

ช่องว่าง (space) หลาย ๆ ช่องว่างจะถูก collapsed ให้เหลือเพียง 1 spece เท่านั้น

2.2.3การเชื่อมโยงกับ HTML Document อื่น ๆ

การทำให้ข้อความที่เราต้องการสามารถเชื่อมโยงไปหาเอกสารที่เกี่ยวข้องได้นั้น สามารถทำได้โดยใช้ Tag <a> ซึ่งข้อความเหล่านี้จะแตกต่างจากข้อความทั่วไปคือ เป็นข้อความสีฟ้า ซึ่งถ้าลากเมาส์ไปในบริเวณข้อความนี้จะทำให้สัญญลักษณ์ลูกศรของเมาส์เปลี่ยนเป็นสัญลักษณ์รูปมือแทน ซึ่งมีวิธีการใช้ดังนี้

1. พิมพ์ <a ตามด้วยช่องว่าง หนึ่งช่อง
2. พิมพ์ชื่อของ Doccment ที่ต้องการเชื่อมโยงไประหว่าง " " ดังนี้ HERF="File_Name และตามด้วย >
3. พิมพ์ข้อความที่ต้องการ
4. พิมพ์ Tag จบ

รูปแบบ : <a HERF="File_Name"> Text

ตัวอย่าง : <a HERF="Mainestats.html">Maine

จากตัวอย่างนี้ คำว่า Maine จะสามารถเชื่อมโยงไปยังแฟ้ม Mainestats.html ได้ ซึ่งแฟ้มนี้จะต้องอยู่ใน directory ปัจจุบัน แต่ถ้าแฟ้มนี้อยู่ใน directory อื่นๆ เราจะต้องบอกถึง path ที่ถูกต้องของแฟ้มนี้ด้วย เพราะฉะนั้นเพื่อความถูกต้องไม่ว่าเราจะอยู่ path ไหนก็ตามก็ควรบอก path แบบสมบูรณ์ของแฟ้มนั้นลงไป เช่น ถ้าแฟ้ม mainstats.html อยู่ใน directory AtlantioStates จะสามารถเขียนได้ดังนี้

<a HERF= "/AtlanticStates/Mainestats.html">Maine

Address

รูปแบบ : <address> Name of Address </address>

หน้าที่ : ใช้ในการกำหนดว่าข้อความในระหว่าง Tag address นี้เป็นที่อยู่ ซึ่งอาจจะเป็นที่อยู่ของผู้แต่ง ซึ่งเราสามารถติดต่อได้ โดย browser จะแสดงออกมาในลักษณะตัวอักขระเอียง (Italio)

ตัวอย่าง

<address> A beginner's Guide to HTML / NCSA / pubs@ncsa.uiuc.edu </address>

ผลลัพธ์

A beginner's Guide to HTML HTML / NCSA / pubs@ncsa.uiuc.edu

Character Formatting

หน้าที่ : เป็นการกำหนดรูปแบบของตัวอักขระที่จะให้ browser แสดงออกมาซึ่งมี Tag ที่ใช้กำหนดอยู่ด้วยกันหลายแบบ ดังนี้

 แสดงข้อความในลักษณะตัวเอียงและหนา

 แสดงข้อความในลักษณะตัวหนา

<b> แสดงข้อความในลักษณะเช่นเดียวกับ Tag

<I> แสดงข้อความในลักษณะตัวเอียง

<blink> แสดงข้อความในลักษณะที่สามารถกระพริบได้

ตัวอย่าง

 This is the first sentence.

 This is the second sentence.

<b> This is the third sentence. <b>

<I> This is the fourth sentence. <I>

ผลลัพธ์

This is the first sentence.

This is the second sentence.

This is the third sentence.

This is the fourth sentence.

Inline Image

คือ การแสดงภาพที่เราต้องการโดย Browser ต่าง ๆ จะแสดงภาพที่มีส่วนขยายเป็น .bmp (X bitmap) และ .gif ซึ่งมีรูปแบบในการใช้ ดังนี้ <IMG SRC = "image_URL"> และถ้าต้องการกำหนดตำแหน่งของภาพว่าจะให้อยู่ส่วนไหนของ page ก็สามารถที่จะทำได้ โดยเพิ่มข้อความ ALIGN = Left หรือ ALIGN = Right ลงไปใน Tag แต่ถ้าต้องการให้ข้อความที่อยู่ต่อจากรูปภาพอยู่ในระดับไหนของรูปภาพ ก็ให้เพิ่ม option top , option middle หรือ option bottom เข้าไปในข้อความ ALIGN และข้อความนี้จะมี default เป็น bottom

เส้นไต้

รูปแบบ : <hr>

หน้าที่ : ใช้ในการขีดเส้นตรง

ตัวอย่าง

This is the line. <hr>

ผลลัพธ์

This is the line.

2.2.4 คำสั่งพื้นฐานของ HTML

การทำให้ข้อความหรือรูปภาพอยู่ตรงกลาง Page

รูปแบบ : Text

หน้าที่ : ทำให้ข้อความหรือรูปภาพอยู่กลางหน้าเอกสาร

การสร้าง Black Ground **ของเอกสาร**

รูปแบบ : <body background = "name"> </body>

โดยที่ name เป็นชื่อของแฟ้มรูปภาพที่นำมาทำเป็น black ground

การขึ้นบรรทัดใหม่โดยไม่เว้นบรรทัด

รูปแบบ :

หน้าที่ : ใช้ในการขึ้นบรรทัดใหม่โดยที่ไม่ต้องการเว้นบรรทัด

ตัวอย่าง

Welcome to HTML Document.

This is the first paragraph.

This is the second paragraph.

ผลลัพธ์

Welcome to HTML Document. This is the first paragraph.

This is the second paragraph.

2.2.5 Image Map.

ตามปกติสามารถทำให้รูปภาพเชื่อมไปยังเอกสารอื่น ๆ ที่ต้องการได้โดยใช้ Tag <IMG> ใน Tag <a> ซึ่งสามารถที่จะคลิกเมาส์ในทุก ๆ จุดของรูปภาพ แต่ถ้าต้องการให้รูปภาพเชื่อมต่อไปยังเอกสารได้หลาย ๆ เอกสารโดยขึ้นอยู่กับตำแหน่งที่คลิกนั้น เราจะต้องใช้วิธี Image Map ซึ่งการใช้ Image Map นั้นก็จะต้องสร้างแผนที่ขึ้นมาแล้วบอกพิกัดของพื้นที่แต่ละส่วนที่จะใช้ในการคลิกเมาส์

หมายเหตุ

ในโปรแกรม Browser ที่เป็น Text อย่างเดียวเช่น Lynx ซึ่ง browser เหล่านี้ไม่สามารถแสดงในแบบ Graphic ได้ ดังนั้นจึงไม่สามารถใช้ Image Map ใน browser เหล่านี้ได้ แต่ก็จะสามารถแก้ไขได้โดย ทำ Text Anchor ที่สอดคล้องกับ Image Map ขึ้นเพื่อให้ browser เหล่านี้สามารถเชื่อมโยงไปยังเอกสารตามที่ต้องการได้

การสร้าง Image Map มี 3 ขั้นตอน คือ

1. สร้าง หรือ เลือกรูปที่จะนำมาทำ Image Maps
2. สร้าง Map file ซึ่งเป็น Text file ที่แสดงพิกัด (Coordinate) ของรูปภาพที่จะนำมาทำ Image Maps พร้อมกับ URL ที่ต้องการจะเชื่อมโยงไปถึง
3. เชื่อมต่อ Image และ Map file เข้าด้วยกันโดย program ที่เรียกว่า โปรแกรม gateway script ใน HTML

2.3 โปรแกรมภาษา Java

Sun Microsystem ได้แนะนำ Java ซึ่งมีลักษณะเป็นภาษา Object Oriented ซึ่งในปัจจุบันเป็นที่นิยมอย่างมาก Java ได้ถือกำเนิดขึ้นมาเพื่อเป็นภาษาสำหรับโปรแกรมที่ใช้งานได้จริงทางธุรกิจ

ประเภทของการ program ใน Java Programmingมีอยู่ 2 ประเภทคือ

1. Java Application เป็น standalone ที่สามารถ execute ได้ด้วยตัวเอง
2. Java Applet มีลักษณะคล้าย Java Application เพียงแต่ว่าไม่สามารถ run standalone ได้ ต้อง run ภายใต้ Virtual machine ของ Java โดยส่วนใหญ่โปรแกรม Java มักถูกเรียกใช้อยู่ใน homepage

2.3.1 ขั้นตอนพัฒนาโปรแกรมด้วย Java

หลังจากที่เขียน Program ด้วย Java เสร็จสิ้นแล้ว ทำการ compile ภายหลัง compile เสร็จจะใช้ชื่อ file class โดยจะอยู่ใน directory เดียวกับ Java source file (นามสกุลของ java source file คือ java) ใน class file นี้ เป็น java byte code ไปใช้ได้โดยแล้วแต่ละประเภท

2.3.2 ลักษณะเด่นของ Java

มีขนาดเล็กปลอดภัย Object oriented programming ทำให้สามารถแบ่งการโปรแกรมย่อย ๆ ได้ สะดวกเป็นลักษณะ byte coded มีความสามารถ Multithreaded programming language ทำงานได้ไม่ขึ้นกับ platform

2.3.3 มุมมอง Java กับการโปรแกรม

Syntax ของ Java ถูกพัฒนามาจาก C++ ทำให้ programmer ที่คุ้นเคยกับภาษานี้ (ซึ่งมีอยู่เป็นจำนวนมาก) สามารถเรียนรู้ Java ได้อย่างง่ายดาย

หากแต่ว่าบางส่วนของ C++ ก็ไม่ปรากฏใน Java เนื่องจากต้องการแก้ปัญหาในส่วนที่ยุ่งยาก และซับซ้อนของ C++ เช่น ในส่วนของ pointer และการจัดการ memory โดยส่วนเหล่านี้จะมีความสลับซับซ้อนในการใช้ และอาจใช้ผิดได้โดยง่าย การหาความผิดพลาดของ pointer ใน program ขนาดใหญ่ เป็นการหาที่ ยุ่งยากมาก ดังนั้นในการจัดการ memory Java จะจัดการให้อย่างอัตโนมัติ programmer ไม่ต้องมานั่งเขียน garbage collection เอาเอง

การออกแบบในส่วนของ data types และ object ใน Java นั้นง่ายมาก โดยจะเข้มงวดมากในเรื่องของตัวแปร นั่นคือ สามารถใช้ตัวแปรใน data type ที่กำหนดไว้แล้วเท่านั้น ซึ่งหากมีการใช้ตัวแปรผิดพลาดในตอน compile program จะไม่มีการปล่อยให้ข้อผิดพลาดนั้นหลุดรอดไปถึงขั้น run program ซึ่งทำให้เป็นการยากยิ่งขึ้นที่จะหา programmer ที่มีประสบการณ์ในการเขียน C++ แล้วเปลี่ยนมาเขียน Java จะมีปัญหาเกี่ยวกับการเปลี่ยนการลดของ C++ แต่ก็จะได้รับความสะดวกในการเขียนที่ง่ายขึ้น

2.3.4 ความเป็น Object Oriented Programming ของ Java

ภาษา Java นับเป็นภาษาแบบ Object oriented programming (OOP) ซึ่งลักษณะของภาษาเช่นนี้มีประโยชน์มากในการพัฒนา software OOP จะ organize program เป็น set ของ component ที่เรียกว่า objects โดย objectsเหล่านี้จะมีลักษณะเป็นอิสระต่อกัน และจะมีกฎในการติดต่อสื่อสารต่อกัน

Java จะได้รับ concept object oriented มาจาก C++ และภาษาอื่นอีกเช่น Smalltalk ในข้อดีของภาษาในแนว object oriented ก็คือว่า จะสามารถเข้าใจง่าย หาข้อผิดพลาดได้ง่ายและนำไปปรับปรุงใช้ใน project ได้ง่าย

2.3.5 ความปลอดภัยใน Java

ในการประสบความสำเร็จของ Java อีกประการหนึ่งก็คือ เป็นภาษาซึ่งมีความปลอดภัย ซึ่งการเน้นทางด้านความปลอดภัยเป็นสิ่งที่เหมาะสมสำหรับการพัฒนา Java ในโลกของ World Wide Web

Java จะมีความปลอดภัยในหลายระดับ อย่างแรกก็คือว่าเป็นภาษาที่ถูกออกแบบมาให้ยากที่จะ execute จาก code ที่ทำอันตราย program ได้ เช่น การไม่มี pointer ถึงแม้ว่า pointer จะมีประโยชน์อย่างมาก แต่ก็มีโทษมหันต์ เนื่องจาก pointer สามารถถูกใช้เพื่อเข้าถึงในส่วนพื้นที่ที่อาจทำลาย program ได้ และยังสามารถเข้าถึงพื้นที่ใน memory ได้ ดังนั้นการไม่มี pointer จึงเป็นความปลอดภัยอย่างหนึ่งของ Java ที่มีมากกว่าภาษาอื่น

ความปลอดภัยในระดับ byte code โปรแกรม Java ที่ทำการ compile แล้วจะอยู่ในรูปแบบของ byte codeก่อนที่จะมีการ run program Java จะมีการตรวจสอบในแต่ละ byte code ก่อนว่ามีสิ่งผิดปกติเกิดขึ้นหรือไม่

ความปลอดภัยในการใช้ Java applet ก็คือ Java applet จะไม่สามารถ เปิด อ่าน เขียน file บนระบบของuser เพื่อป้องกัน program จากการกระทำผิดของ disk dive ของ user

2.3.6 Multithread ของ Java

ในลักษณะของ Multithread ทำให้สามารถทำงานได้หลาย ๆ งานในเวลาเดียวกัน ซึ่งเหมาะกับ operating system ที่มีลักษณะเป็นแบบ multitasking เช่น Window 95 โดย Java ได้จัดเตรียม tool ในการเขียน program แบบ multithread และทำให้ program เหล่านี้มีการ execute ได้อย่างน่าเชื่อถือ

2.3.7 การทำงานของ Java ที่ไม่ขึ้นกับ platform

Software ของ computer ส่วนมากจะถูกพัฒนาเฉพาะสำหรับ operating system บางประเภทเท่านั้น การที่ไม่ขึ้นกับ platform ทำให้ program เดียวกันทำงานได้บน operating system ที่แตกต่างกัน ซึ่งช่วยลด

ความยุ่งยากในการพัฒนา program หลายรอบสำหรับแต่ละ operating system

ชนิดตัวแปรของ Java มีขนาดเดียวกันในทุก ๆ platform รวมถึง applet ที่พบบน Web byte code สามารถexecute ได้ในทุก ๆ platform โดยไม่มีการเปลี่ยนขนาด

2.3.8 จาวาสคริปต์เบื้องต้น

วิธีการเพิ่มโค้ดจาวาสคริปต์ในเว็บเพจ

จะเริ่มกันที่วิธีการแทรกโค้ดจาวาสคริปต์ลงไปในไฟล์ HTML ซึ่งแท็ก (tag) ที่ใช้บ่งบอกถึงจุดเริ่มต้นและจุดสิ้นสุดของโค้ดจาวาสคริปต์ก็คือ <SCRIPT> และ </SCRIPT> ซึ่งแท็กเริ่มต้นจาวาสคริปต์ ควรระบุ ว่าเป็นภาษาจาวาสคริปต์ด้วย ซึ่งจะเขียนได้ดังนี้

```
<SCRIPT language="JavaScript">
```

ส่วนที่เขียนว่า language="JavaScript" ก็เพื่อให้โปรแกรมบราวเซอร์รับรู้ว่าเป็นภาษาจาวาสคริปต์ ไม่ใช่ภาษาสคริปท์อื่นๆเช่น VBScript จะใส่โค้ดจาวาสคริปท์ตามแท็กดังกล่าว และสิ้นสุดโค้ดจาวาสคริปท์ด้วยแท็ก </SCRIPT> ซึ่งจะเขียนได้ดังนี้

```
<SCRIPT language="JavaScript">

.....โค้ดจาวาสคริปท์.....

</SCRIPT>
```

ในไฟล์ HTML ไฟล์หนึ่งสามารถจะมีแท็ก <SCRIPT> ได้หลายชุดตามที่ต้องการ ซึ่งก็เหมือนกับว่า มันเป็นHTML แท็กอย่างหนึ่งนั้นเอง แต่อย่าลืมแท็กสำหรับปิด </SCRIPT> และถ้าหากต้องการใช้ฟังก์ชัน จะต้องใส่ฟังก์ชันจาวาสคริปท์ไว้ในส่วน <HEAD> และ </HEAD> ของไฟล์ HTML ทั้งนี้ก็เพื่อให้ฟังก์ชันถูกเรียกขึ้นมาก่อนที่เว็บเพจนั้นจะปรากฏให้เห็นและจะได้ไม่ต้องพบกับปัญหาการเกิดความผิดพลาด (error) ตัวอย่างต่อไปนี้แสดงการเขียนฟังก์ชันจาวาสคริปท์

```
<HEAD>
<TITLE>My World</TITLE>
<SCRIPT language="JavaScript">
  function cool0 {
  .....ตัวโปรแกรม.....
}
</SCRIPT>
```

ตอนนี้ ยังมีอีกสิ่งหนึ่งที่ควรจะทราบก่อนที่จะเริ่มต้นการเขียนจาวาสคริปท์กันก็คือ เนื่องจากในโลกนี้ ยังมีผู้ใช้อินเทอร์เนตอีกเป็นจำนวนมากที่ยังใช้โปรแกรมบราวเซอร์รุ่นเก่าซึ่งยังไม่มีความสามารถในการแสดงจาวาสคริปท์ นั่นคือ ไม่รู้จักแท็ก <SCRIPT> ซึ่งมีผลทำให้โปรแกรมเหล่านั้นแสดงจาวาสคริปท์ออกมาเป็นตัวอักษรธรรมดา วิธีแก้ไขก็คือต้องทำให้โปรแกรมบราวเซอร์รุ่นเก่า ๆ ไม่สนใจสิ่งที่อยู่ในแท็ก <SCRIPT> โดยจะต้องเขียนเป็นคอมเม้นท์ (comment) เหมือนกับการเขียนคอมเม้นท์ในไฟล์ HTML ซึ่งทำได้ดังนี้

```
<SCRIPT language="JavaScript">
<!--hide from old browsers

.....โค้ดจาวาสคริปท์.....

//-->
</SCRIPT>
```

การสร้างตัวแปร (Variable)

การสร้างตัวแปรและชนิดต่าง ๆ ของตัวแปรเพื่อจะได้นำไปใช้กับฟังก์ชันจาวาสคริปท์ เริ่มจากการประกาศ (declare) ตัวแปร ซึ่งจะต้องอยู่ในส่วน HEAD ของไฟล์ HTML โดยการประกาศตัวแปรลงไประหว่างแท็ก SCRIPT ดังนี้

```
<HEAD>
<SCRIPT language="JavaScript">
<--hide from old browsers
   var name=value;
//-->
</SCRIPT>
</HEAD>
```

โดยที่สามารถอธิบายความหมายได้ดังนี้

var คือการบ่งชี้ว่าเป็นการประกาศตัวแปร

name ชื่อของตัวแปร จะกำหนดให้ตัวแปรที่ชื่อว่าอะไรก็ได้ ที่ไม่ซ้ำกับคำสงวนของการเขียนโปรแกรม เช่นให้ตัวแปรมีชื่อว่า function ไม่ได้

value ค่าเริ่มต้นของตัวแปรนั้น ซึ่งสามารถจะเป็นตัวเลข (number) ข้อความ (words)
ค่าบูลีน
(boolean(true , false)) หรือว่าง (null) ก็ได้ ซึ่งจะแยกอธิบายในส่วนต่อไปนี้

(boolean(true , false)) หรือว่าง (null) ก็ได้ ซึ่งจะแยกอธิบายในส่วนต่อไปนี้

การใช้ตัวเลข (Numbers)

สามารถกำหนดค่าที่เป็นตัวเลขให้กับตัวแปรโดยใช้เครื่องหมายเท่ากับ ดังนี้

```
var cars=3;
```

ตัวเลขดังกล่าวไม่เป็นจำนวนเต็ม เช่น เป็นทศนิยมก็ได้ เช่น

```
var cost=9.95;
```

การใช้สตริง (Strings)

สตริงคือกลุ่มของตัวอักษร เช่น คำหรือประโยคการกำหนดค่าตัวแปรเป็นสตริงต้องใช้เครื่องหมายคำพูดคู่ล้อมสตริงนั้นไว้ ดังนี้

```
var movie="The Lost World";
```

ถ้าใส่ตัวเลขไว้ระหว่างเครื่องหมายคำพูด ตัวเลขนั้นจะถูกใช้เหมือนกับว่ามันเป็นสตริง

การใช้ค่าบูลีน (Boolean Value)

เป็นการกำหนดให้ตัวแปรมีค่าเป็นจริง (true) หรือ เท็จ (false) ดังนี้

```
var story=true;
```

ค่าว่าง (Null Value)

ถ้ากำหนดให้ตัวแปรมีค่าเป็น null นั่นคือ เป็นค่าว่าง ไม่ใช่แม้แต่ 0 ซึ่งมีรูปแบบการใช้ดังนี้

```
var mymoney=null;
```

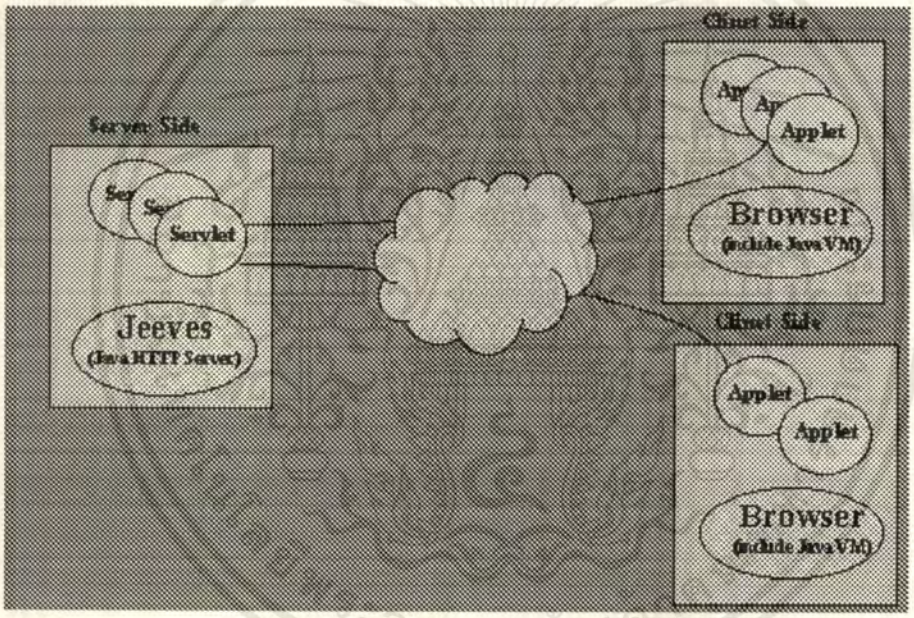
ความแตกต่างของตัวอักษรตัวเล็กหรือตัวใหญ่(Case Sensitivity)

ไม่เหมือนกับภาษา HTML จาวาสคริปท์ให้ความสำคัญกับความแตกต่างของตัวอักษรตัวเล็กหรือตัวใหญ่ จะใช้สลับกันไม่ได้ เช่น joHn กับ JOHN จะไม่ได้หมายถึงสิ่งเดียวกัน ดังนั้นต้องใช้ความรอบคอบ หากพิมพ์ผิด อาจทำให้โปรแกรมทำงานผิดพลาดได้

2.4 Java Servlet

Java Servlet คือ โปรแกรมภาษาจาวา ที่ถูกสร้างมาให้ทำงานได้บน HTTP Server โดยการทำงานของ Servlet มีลักษณะเป็น Server-side ซึ่งต่างจาก Java Applet โดยทั่วไปที่ทำงานในลักษณะ Client-side โดย Servlet เป็น modules ที่ทำงานอยู่ภายใต้กระบวนการ request/response-oriented servers เช่น Java-enabled web servers ตัวอย่างเช่น Servlet สามารถตอบสนองในการส่งข้อมูลโดยให้อยู่ในรูปของภาษา HTML และ Servlet ยังสามารถนำมาประยุกต์ใช้ในการพัฒนาฐานข้อมูล (database) ของบริษัทต่างๆได้

ประโยชน์ของ Servlet นั้นช่วยให้เราเขียนProgram ที่สั่งให้ HTTP Server ทำงานหรือโต้ตอบกับ Programของ Clientเหมือนที่ CGI (Common Gateway Interface) ทำได้ซึ่งServlet ถูกนำมาใช้แทน CGI scripts ได้อย่างมีประสิทธิภาพและ ให้ความสะดวกได้อย่างมากคือสามารถเขียนได้ง่ายกว่า และทำการ run ได้เร็วกว่า CGI scripts ซึ่ง Servlet ได้มีการจัดการในส่วนของการโปรแกรมทางด้าน server-side ไว้โดยเฉพาะ Servlet จะถูกพัฒนาโดยใช้ Java Servlet API ซึ่งเป็นส่วนเพิ่มเติมมาตรฐานของ Java



รูปที่ 2.2 แสดงความสัมพันธ์ระหว่าง Server ที่เป็น Servlet กับ Client ทั่วๆไป

2.4.1 ความแตกต่างของ Servlet กับโปรแกรมชนิดอื่น

เปรียบเทียบความแตกต่างระหว่าง Java Servlet กับ Java Applets

Java Servlet	Java Applets
ทำงานบน Java HTTP Server	ทำงานบน Browser (Client)
ไม่มีส่วนติดต่อกับผู้ใช้โดยตรง (User Interface)	มี AWT สำหรับทำส่วนติดต่อกับผู้ใช้
อนุญาติให้สามารถ access local file และ Network ได้	ไม่อนุญาติให้สามารถ access local file และ Network ได้นอกจากติดต่อกลับไปหาผู้เรียก

จุดเด่นของ Servlet ที่ CGI ไม่มี หรือด้อยกว่ามีดังนี้

1. Servlet สามารถที่จะฝังตัวทำงานอยู่บน server เมื่อทำการให้บริการ Client เสร็จสิ้นแล้วเพื่อจะคอยรับ Request ใหม่ที่อาจจะมีมากจาก Client อีกโดยไม่ต้องถูกสร้างขึ้นมาใหม่และตายไปทุกๆครั้งที่มี Request จาก Clientเหมือน CGI. Script ทำให้ Servlet รบกวนทรัพยากรของserverน้อยกว่า CGI.Script

2. Servlet สามารถจัดการกับงานประเภท Multiple Connection ได้ดี โดยอาศัยความสามารถของ Thread ที่มีอยู่ในตัว Java ตัวอย่างเช่น การ Broadcast ข้อมูลไปทุก Connection พร้อมๆกัน

3. Servlet สามารถติดต่อกับ Applet บน Client ได้อย่างต่อเนื่องเป็นผลให้การรับส่งข้อมูลระหว่างกันดำเนินไปอย่างต่อเนื่อง แต่สำหรับ CGI. SCript นั้นทำได้ไม่สะดวกนัก เนื่องจากการรับส่งข้อมูลอย่างต่อเนื่องนั้นต้องอาศัยการบันทึก State ก่อนหน้าเอาไว้ แต่ CGI Script จะสิ้นสุดลงทุกครั้งหลังจากให้บริการได้เพียงหนึ่ง Request เท่านั้น

4. Servlet สามารถถูก Upload จาก Client เพื่อสั่งให้ไปทำงานบน Serverใดๆบน Network ได้ ประโยชน์เช่น เราสามารถเขียน Program เพื่อค้นหาข้อมูลใน Wep Hostใดๆโดยไม่ต้องส่งข้อมูลมาประมวลที่เครื่องต้นทาง สามารถทำได้ที่ตัว Host นั้นๆเลย เสร็จแล้วค่อยส่งผลรับกลับคืน จึงช่วยลดเวลาและ Traffic บนระบบ Network ลงอย่างมาก

เปรียบเทียบความแตกต่างระหว่าง Java Servlet กับ CGI

	Java Servlet	CGI
ภาษาที่ใช้	JAVA	ภาษาทั่วๆไปเช่น Perl, C
การเรียกโปรแกรม	โดยการเรียก Methode call	โดยการรันโปรแกรม
การส่งผ่านค่า Parameters	ผ่าน ServletRequest methodes	ผ่าน Evironment Variables

ระบบความปลอดภัย	JAVA Security Manager	OS นั้นๆ
ความสิ้นเปลืองทรัพยากรบนServer	น้อย	สูง

2.4.2 ส่วนหนึ่งของ aplication ของ Servlet

1. การจัดการส่งผ่านข้อมูลบน HTTP โดยส่งให้อยู่ในรูปของภาษา HTML รวมไปถึงการสั่งซื้อสินค้าทาง on-line โดย Servlet จะทำหน้าที่ในการจัดการส่วนของการสั่งสินค้า การจัดการระบบ และการจัดการฐานข้อมูล รวมถึงการจ่ายเงินโดย credit card ทาง online

2 .อนุญาตให้มีการใช้งานได้พร้อมกัน Servlets สามารถตอบสนองต่อการร้องขอแบบ multiple ได้โดย สามารถตอบสนองต่อการร้องขอได้ในเวลาเดียวกัน

3. การตอบสนองต่อการร้องขอ Servlet สามารถตอบสนองต่อการร้องขอจาก server หรือ Servlet อื่นๆ ได้

4. เป็นการติดต่อสื่อสารแบบ active agents ผู้เขียน Servlet สามารถกำหนด active agents ซึ่งมีการแบ่งงานร่วมกันในแต่ละกลุ่ม โดยที่แต่ละ agent จะต้องเป็น Servlet และสามารถผ่านข้อมูลสู่กันภายในกลุ่มได้เองด้วย

2.4.3 สถาปัตยกรรมของ Servlet

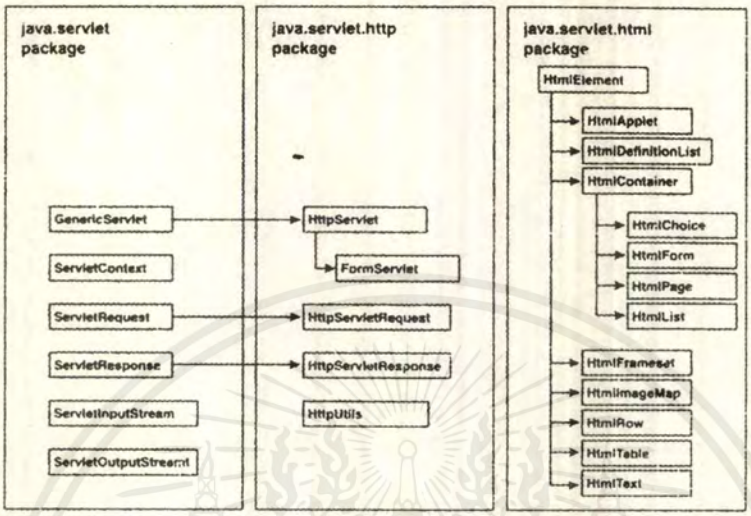
ใจความสำคัญของ Servlet API คือ การสนับสนุนการ interface ของ Servlet ทั้งแบบทางตรงหรือทางอ้อม โดยประกอบไปด้วยคลาสที่สนับสนุนการ interface เช่น HttpServlet ซึ่ง Servlet interface ประกอบด้วย method ที่ใช้เป็นตัวจัดการ Servlet และติดต่อสื่อสารกับ client ผู้เขียน Servlet จะกำหนด method นี้ทั้งหมดหรือบางส่วนเมื่อต้องการพัฒนา Servlet เมื่อ Servlet ได้รับการร้องเรียกจาก client จะมีการรับ object อยู่ 2 object ซึ่งอันแรกคือ ServletRequest และ อีกอันคือ ServletResponse โดยคลาส ServletRequest นั้นจะเป็นตัวติดต่อสื่อสารจาก client ไปยัง server ในขณะที่คลาส ServletResponse เป็นตัวติดต่อสื่อสารจาก Servlet กลับไปยัง client

ServletRequest interface อนุญาตให้ Servlet สามารถเข้าถึงข้อมูลได้ เช่น ชื่อของพารามิเตอร์ที่ผ่านเข้ามาโดย client , โปรโตคอล และ ชื่อ host ของclientที่เป็นตัวส่ง request และของserver ที่เป็นตัวรับ request ด้วยรวมไปถึงการกำหนดให้ Servlet เข้าถึง input stream ได้ด้วยโดย ServlerInputStream โดย Servlet ทำหน้าที่รับข้อมูลจาก client โดยผ่านทางโปรโตคอลเช่น HTTP, POST และ PUT method โดยคลาสย่อยของคลาส ServletRequest จะอนุญาตให้ Servletสามารถรับข้อมูลที่เป็นลักษณะเฉพาะของโปรโตคอลกลับมาได้

ServletResponse interface ประกอบด้วย method สำหรับการตอบกลับไปยัง client โดยอนุญาตให้ Servlet กำหนดขนาดและ mime type ในการตอบกลับ และเป็นตัวจัดเตรียม output streamด้วย โดยผู้เขียน Servlet สามารถใช้ ServletOutputStream ในการส่งข้อมูลกลับได้ subคลาส ของ ServletResponse ทำให้ Servlet มีโปรโตคอลที่แตกต่างกันได้หลายชนิด ตัวอย่างเช่น HttpServletResponse ประกอบด้วย method ที่อนุญาตให้ Servlet สามารถถ่ายเทข้อความจาก HTTP-specific header ได้

2.4.4 Servlet API

Servlet API สามารถใช้ได้บน server พื้นฐานทั่วไปที่มีการสนันสนุน Servlet และถูกกำหนดโดยคลาสที่อยู่ในแพคเกจดังนี้ java.Servlet ,java.Servlet.http และ java.Servlet.html รูปที่ 16.4 แสดงความสัมพันธ์ของแพคเกจ และ คลาสโดยรายละเอียดของแต่ละแพคเกจจะกล่าวถึงในลำดับต่อไป



รูปที่ 2.3 The Servlet API คลาส

2.4.5 Package: java.servlet

Java.Servlet เป็นแพคเกจที่กำหนดการทำงานของ Servlet จากสัดส่วนของ protocol-independent ในระดับนี้โปรโตคอลที่ถูกใช้ในการเข้าถึง Servlet ได้แก่ HTTP, โปรโตคอลของระบบไฟล์เครือข่าย หรือ โปรโตคอลที่ใช้ในการขนส่งอื่นๆ Servlet ถูกออกแบบให้ทำงานบน server-side โดยใช้คลาสในแพคเกจนี้ซึ่งประกอบไปด้วย interfaces class 4 คลาส และ implementation class 3 คลาส โดยจะกล่าวถึงต่อไป

1 Servlet Interface Class ประกอบไปด้วย method พื้นฐาน 4 method ดังนี้คือ

- init(ServletStub) ถูกเรียกโดยอัตโนมัติเมื่อระบบต้องการหาค่าเริ่มต้นของ Servlet โดยกำหนดตัวแปรเริ่มต้นของ Servlet ในไฟล์ servlet.properties ได้โดยการเรียก getInitParameter() หรือ getInitParameters()
- destroy() ถูกเรียกโดยระบบเพื่อคืนทรัพยากรให้ระบบเมื่อ Servlet ถูกยกเลิกการใช้งาน
- service(ServletRequest, ServletResponse) ถูกเรียกมาใช้ในกระบวนการรับ request ที่เข้ามาและสร้าง response ตอบกลับไป
- getServletInfo() ถูกเรียกเพื่อรับข้อมูล(information) ที่เป็น String ซึ่งบอกลักษณะของ Servlet

2 ServletContext Interface Class เป็นการกำหนด method เพื่อให้ Servlet สามารถทำงานแบบสลับฉาก(interact)ได้โดย method ต่างๆมีดังนี้

- getMimeType(String filename) ส่งคืน MIME type ของแต่ละไฟล์โดยใส่ชื่อของไฟล์(filename) ลงใน mime.properties file
- getRealPath(String path) การประยุกต์ alias rules ใน alias.properties file เพื่อเป็นการกำหนด virtual path และตอบสนองด้วยการส่ง real path หากการส่งข้อมูลเกิดการผิดพลาดค่าที่ส่งกลับคือ null

- getServerInfo() ส่งคืนค่าที่เป็น string ที่บ่งบอกชื่อ, version และฐานของ server software
- getServlet(String) ส่งคืน Servlet ที่ตรงกับชื่อที่เรียกมา
- getServlets() ส่งคืน Servlets ทั้งหมดที่ถูกโหลดใน server โดยรวมไปถึง Servlet ที่ถูกเรียกด้วย
- log(Servlet, String) เขียนข้อความลงใน servlet log ของ Servlet ที่ถูกกำหนด

3 ServletRequest Interface Class ประกอบไปด้วย method ที่ใช้สำหรับค้นหาข้อมูลที่เกี่ยวกับ servler request ซึ่งเป็นอิสระจากโปรโตคอลที่ใช้อยู่จริงในการส่งผ่านข้อมูลระหว่าง client และ server โดย request จะถูกนำพาโดย IP พื้นฐานของโปรโตคอลเช่น HTTP ซึ่งการส่งผ่านข้อมูลของ ServletRequest โดยอาศัยโปรโตคอลเช่น HTTP นั้นจะต้องมี method ที่มีความสัมพันธ์กับโปโตคอลดังนี้

- getRemoteAddr(). ส่งคืนค่า IP address ของ client ที่ส่ง request
- getRemoteHost(). ส่งคืน host name ของตัวแทนที่ส่ง request
- getServerName(). ส่งคืน host name ของ server ที่รับ request
- getServerPort(). ส่งคืนค่า port number ที่รับ request
- getProtocol(). ส่งคืนค่าที่เป็น string ซึ่งแสดง protocol และ version

request ประกอบไปด้วยพารามิเตอร์และข้อมูลต่างๆ โดย Servlet สามารถหาขนาดความยาวของ content ได้โดย getContentLength(). และหาชนิดของ content ได้จาก getContentType(). เมื่อหา content พบ Servlet สามารถอ่านได้โดยใช้ HttpInputStream ซึ่งได้จาก getInputStream(). โดย method ที่มีความสัมพันธ์กับ content มีดังนี้

- getParameter(String) ส่งคืนพารามิเตอร์ที่กำหนดมาสำหรับ request
- getParameters(). ส่งคืนพารามิเตอร์ทั้งหมดของ request
- getContentType() ส่งคืน MIME type ของข้อมูลที่รวมมากับ request หรือค่า null ถ้าไม่มีการกำหนf MIME type มากับ request
- getInputStream(). ส่งคืน input stream เพื่ออ่านข้อมูลที่มากับ request

4. ServletResponse Interface class เป็นตัวกำหนดการสร้าง response สำหรับตอบสนอง request โดยทั่วไปแล้ว response จะประกอบไปด้วยส่วนที่บอกลักษณะของ MIME และขนาดของ content โดยการเขียน content ใน response นั้น Servlet จะเรียก getOutputStream() เพื่อทำหน้าที่รับ ServletOutputStream จากนั้นเรียก print(). เพื่อส่ง response โดย method ที่ใช้ในการสร้าง response มีดังนี้

- setContentType(String). เซ็ท MIME type ของ response
- setContentLength(int). เซ็ทขนาดความยาวของ response
- getOutputStream().ส่งคืน ServletOutputStream เพื่อทำการเขียนข้อมูล response ให้กลับไปยัง client

5. GenericServlet Class เป็นคลาสพื้นฐานที่สนันสนุนการส่งผ่านข้อมูลของ Servlet และใช้สำหรับการปลุก Servlet จาก server-side include โดยประกอบด้วย method ดังนี้

- init(). ถูกเรียกหลังจาก Servlet ได้รับการโหลดให้เป็นขั้นเริ่มต้น โดยจะข้าม method นี้ไปได้เมื่อมีการรับพารามิเตอร์เริ่มต้นต่างๆ หรือมีการทำงานที่ฟังก์ชั่นอื่นๆ ซึ่งในแต่ละครั้งเมื่อ Servlet ถูกโหลด Servletจะมีค่าอยู่ที่ค่าเริ่มต้นหนึ่งครั้งเท่านั้น
- getInitParameter(String name) รับค่าพารามิเตอร์เริ่มต้นของ Servlet ทั้งแบบ servlet.properties file และ อยู่ในรูปของ<servlet> tag สำหรับ server-side includes servlets.
- getInitParameters(). ส่งคืนชื่อ และค่าพารามิเตอร์เริ่มต้นของ Servlet
- getServletContext(). ส่งคืน servlet context.
- Log(String) log ข้อความลงใน servlet log file\

6. ServletInputStream เป็น subclass ของInputStream ที่กำหนด readLine(byte[], int offset,int count) method ซึ่งทำการอ่าน byte ลงใน byte array จนกระทั่ง array เต็มหรือมีการอ่านบรรทัดใหม่เข้ามา

7. ServletOutputStream Abstract Class เป็น subclass ของ OutputStream ซึ่งคล้ายกับ PrintStream แต่เป็นเพียงการกำหนด method ที่ใช้ print ข้อมูลโดยทั่วไปชนิดของข้อมูลจะใช้ ints, longs และ strings การทำงานของ ServletOutputStream จะทำการเปลี่ยน code ที่ใช้ในโปรแกรม (internal) ให้เป็น US-ASCII ซึ่งเป็น code ที่ใช้กันทั่วไปในระบบ Internet protocol โดยประกอบด้วย method ดังนี้

- print(int), println(int)
- print(long), println(long)
- println(String), println(String)
- println()

โดยสามารถแปลงข้อมูลชนิดอื่นก่อนการสั่ง print ออกทาง ServletOutputStream ได้โดยใช้ toString ().

2.4.6 Package: java.servlet.http

java.servlet.http เป็นแพคเกจที่สนับสนุน Servlet ในการใช้ HTTP โดยเฉพาะ แพคเกจนี้ประกอบไปด้วย interface class 2 คลาส implementation class 2 คลาส และคลาสสนับสนุนที่เป็น subclass โดยจะกล่าาถึงในลำดับต่อไป

1.HttpServlet Abstract Class เป็นเครื่องมือที่ใช้ในการค้นหาและแปลงข้อมูลระหว่าง ServletRequest และ ServletResponse โดยประกอบด้วย method ดังนี้

- service(ServletRequest, ServletResponse) ทำหน้าที่ค้นหาและแปลงข้อมูลทั่วๆไประหว่าง HTTP-specific request กับ response โดยทั่วไปแล้วจะไม่มีการเขียนทับ method นี้
- service(HttpServletRequest, HttpServletResponse). เขียนทับ abstract method เพื่อรับ HTTP service request

2. *HttpServletRequest Interface Class* เป็น subclass ของ ServletRequest ซึ่งเป็นตัวจัดการข้อมูลของ extra request และแสดงลักษณะของ content โดยแบ่งประเภทของข้อมูลที่หาได้จาก HttpServletRequest เป็น 4 ลำดับดังนี้

2.1 General Request Information ประกอบด้วย method ที่ให้ข้อมูลทั่วไปเกี่ยบrequest รวมไปถึงชื่อของ method ที่ใช้ในการเข้าถึงข้อมูลและคำถามต่างๆมีดังนี้คือ

- getMethod().ส่งคืน string ที่บรรจุ method ที่ใช้ใน request เช่น "GET", "HEAD" หรือ "POST"
- getQueryString(). ส่งคืน query string ซึ่งเป็นส่วนหนึ่งของ URI หรือ ถ้าไม่มีจะส่งค่า null โดยที่ query string จะตามหลังเครื่องหมาย ? ใน URI

2.2 URL Information ประกอบด้วย method ที่ให้ข้อมูลเกี่ยวกับการร้องขอ URLดังนี้คือ

- getRequestURI(). ส่งคืน request URI
- getServletPath(). ส่งคืน request URI ส่วนที่อ้างอิงถึงการปลุก Servlet
- getPathInfo(). ส่งคืนข้อมูลของ extra path ที่ตามหลัง servlet path ซึ่งนำหน้า query string
- getPathTranslated(). ส่งคืนข้อมูลของ extra path ที่ส่งผ่านไปยัง real path โดยใช้ alias.properties rule
- getRequestPath(). ส่งคืน request URI ส่วนที่ตอบสนอง servlet path ที่เพิ่มเติมด้วยข้อมูลของ extra path

2.3 Authentication-Related Information การเข้าถึงข้อมูลที่เกี่ยวกับชื่อ และที่อยู่ของ client นั้น Servlet สามารถทำได้โดยใช้ getHostName() และ getHostAddress() method แต่สำหรับชนิดของ HTTP-level authenication สามารถแสดงได้โดยใช้ getAuthType() และการติดต่อในแบบ basic mode จะมีการส่งคืนชื่อผู้ใช้โดย getRemoteUser() method

- getAuthType(). ส่งคืนรูปแบบของการรัยรอง(Authenication) ของ request ในกรณีที่หาไม่พบค่าที่ส่งคืนคือ null
- getRemoteUser(). ส่งคืนชื่อของผู้ใช้ที่สร้าง request หรือถ้าไม่มีชื่อมากับ request นั้นจะให้ค่าเป็น null

2.4 Access to HTTP Request Header Fields สำหรับ HttpServletRequest มีการเปิดช่องทางให้มีการเข้าถึง HTTP request header fields โดยตรงเพื่อรับข้อมูลที่ได้รับการสนับสนุนเพียงเล็กน้อยจาก HttpServletRequest method ตัวอย่างเช่น include user-agent, referer, preferred languages และ cookie (สถานะของ client) ซึ่งสิ่งเหล่านี้มี header fields ที่ไม่แน่นอน

ต่อไปจะกล่าวถึง method 2 method ที่ใช้ในการเข้าถึง header เริ่มต้นเพื่อที่จะใส่ค่าดัชนีให้กับ header นั้นโดยค่าดัชนีที่เป็น 0 จะส่งคืน header field อันดับแรก และถ้าค่าดัชนีมีค่ามากกว่าจำนวนของ field ที่เป็นไปได้ค่าที่ส่งคืนคือ null

- getHeader(int). ส่งคืนจำนวนของ header field (n) ถ้ามีค่าน้อยกว่า header field (n) ค่าที่ส่งคืนคือ null
- getHeaderName(int) ส่งคืนชื่อของจำนวน (n) ถ้ามีค่าน้อยกว่า (n) ค่าที่ส่งกลับคือ null

ในกรณีต่อไปเป็นการค้นหา request header ที่ตรงกันกับชื่อของ header field และส่งคืนค่าของ header field นั้นกลับไป โดยเลือกใช้ method ที่กำหนดให้ต่อไปนี้ซึ่งขึ้นอยู่กับรูปแบบประเภทของข้อมูลที่ต้องการจะได้กลับคืนมา

- getHeader(String fieldname) ส่งคืนค่าของ header field ถ้าไม่มีการกำหนด method นี้ค่าที่ส่งคืนคือ null

- getIntHeader(String fieldname, int default) ส่งคืนค่าที่เป็น integer ซึ่งเป็นค่าของ header field หลังจากมีการแปลงข้อมูลที่เป็น string ให้เป็น int ถ้าไม่พบ field นี้ค่าที่ส่งคืนจะไม่ได้ถูกกำหนดไว้
- getDateHeader(String fieldname, long default) ส่งคืนค่าวันเดือนปีของ header field หลังจากแปลงข้อมูลที่เป็น string ให้เป็น date ถ้าไม่พบ field นี้จะไม่มีการกำหนดค่าที่ส่งกลับ

3. HttpServletResponse Interface Class เป็น subclass ของ ServletResponse และเป็นเครื่องมือในการสร้าง HTTP response header โดยแบ่งตามการนำไปใช้ประโยน์ได้ 4 แบบ

3.1 Setting the Response Status เป็นการเซ็ทสถานะของ response โดยค่าที่ส่งคืนคือ 200 หรือ OK. ถ้ามี error ส่งคืนหรือ OK.อยู่ในตำแหน่งที่ไม่ถูกต้อง Servlet สามารถระบุแยกรายละเอียดของทั้ง error code และ ข้อความที่บอกถึงลักษณะของ response ได้เอง ถ้ามีเพียง error แนกำ เกิดขึ้นเท่านั้น Servlet จะแปลง code ให้อยู่ในรูปข้อความที่เป็น string แทนซึ่ง method ต่างมีดังนี้

- setStatus(statusCode) เซ็ท status code ในการส่งคืน HTTP header โดย status code ถูกแปลงให้เป็นข้อความที่เป็น string
- setStatus(statusCode, messageString) เซ็ท status code ในการส่งคืน HTTP header

3.2 Specifying Non-Standard Header Fields การรวม field ต่างๆกับ response header นั้น Servlet ได้กำหนดให้ใช้ setHeader() method แต่สำหรับ non-standard header field ซึ่งไม่ได้รับการสนับสนุนจาก HttpServletResponse ซึ่งกำหนดให้ชื่อของ header fiald (ไม่มีเครื่องหมาย ;) อยู่ที่ argument แรก และข้อมูลอยู่ที่ argument ที่ 2 setHeader ที่ต่างกันนั้นแยกด้วยชนิดของพารามิเตอร์ที่เป็น String, int และ date โดยประกอบด้วย method ดังนี้

- setHeader(String fieldname, String) เซ็ทค่าของชื่อ header field เพื่อแสดงค่าออกมา
- setIntHeader(String fieldname, int) เซ็ทค่าของชื่อ header field ที่เป็น integer หลังจากมีการแปลงค่าให้เป็น string
- setDateHeader(String fieldname, long) เซ็ทค่าของชื่อ date field หลังจากแปลง long เป็น Internet-standard date-time string
- unsetHeader(String fieldname)ย้ายชื่อของ header field ออกจาก response

3.3 Returning Error Response การสร้าง และส่ง standard error response เพื่อส่งคืนerror นั้นมี method ที่ทำหน้าที่นี้อยู่ภายใต้ HttpServletResponse คือ sendError() method ซึ่งประกอบด้วย status code และข้อความที่อธิบายรายละเอียดของ error ซึ่งการส่งค่าคืนนี้จะส่งไปบน page โดยประกอบด้วย header ซึ่งแสดงค่า status code และ ข้อความมาตรฐานของ status code ในขณะที่ข้อความที่อธิบายรายละเอียดของ error นั้นจะอยู่บน body ของ page ที่ส่งคืน ตัวอย่างเช่น เมื่อมีการเรียก sendError(404,"Couldn't find it!") จะมีการสร้าง page ซึ่งประกอบด้วย header "404 Not Found" และ ข้อความคือ "Couldn't find it!" โดย method ที่กล่าวถึงมีดังนี้

- sendError(int) ส่ง error response ไปที่ client โดยระบุแต่ status code และไม่มีข้อความเพิ่มเติม
- sendError(int, String) ส่ง error response ไปที่ client โดยระบุทั้ง status code และข้อความที่อธิบายรายละเอียดของ error

3.4 Returning Redirect Response ในการสร้างและส่ง standard redirect response นั้น HttpServletResponse ได้กำหนดให้ใช้ sendRedirect() method โดยจะมีการสร้างและส่งค่า "302" เป็น response ค่าที่ส่งไปจะเป็น string ซึ่งเป็นตัวแสดงตำแหน่งใหม่ของ page โดย string ที่กล่าวถึงนี้ก็คือ URL นั่นเองเนื่องจาก URL คือที่อยู่ที่ต้องการส่ง page ไปเหมือนกับการ link นั่นเอง อย่างไรก็ตาม sendRedirect() ไม่สามารถตรวจสอบได้ว่า string ที่เข้ามานั้นตรงกับ URL ที่ถูกต้องหรือไม่ ถ้าผู้เขียนมี URL ที่ถูกต้องแล้วให้แปลง URL นั้นเป็น string โดยใช้ toString() ก่อนที่จะเรียก sendRedirect() โดยการเรียก method นี้จะอยู่ในรูป sendRedirect(String) ซึ่งจะส่ง redirect response ไปยัง client โดยการระบุที่อยู่ของ URL ที่ต้องการมาด้วย

4. FormServlet Class เป็นคลาสของ Servlet ที่ถูกระบุให้เป็นส่วนหนึ่งของ Servlet API และด้วยเหตุนี้มันจึงสามารถอยู่บน web server ทุกชนิดที่สนับสนุน Servlet โดยทำหน้าที่สร้างแบบฟอร์มการทำงานของ Servlet โดย subclass ของ FormServlet และเขียนทับบน sendResponse() method ซึ่ง method นี้ทำหน้าที่ควบคุมแบบฟอร์มของข้อความ และสร้าง response ซึ่งเป็น subclass ของ HttpServlet โดย method ต่อไปนี้ถูกกำหนดไว้ใน FormServlet Class

- getServletInfo() เขียนทับ method นี้เพื่อส่งคืน string ที่บรรจุอยู่ในข้อความของ Servlet เช่น ชื่อผู้เขียน หรือ เวอร์ชัน
- service(HttpServletRequest, HttpServletResponse) ผู้เขียนสามารถเขียนทับ method นี้เมื่อมีความต้องการ special request processingsendResponse(HttpServletResponse, Hashtable) เขียนทับ method นี้เมื่อต้องการแปลง code เพื่อจัดรูปแบบและสร้าง response *5. HttpUtils Class* เป็นคลาสที่ทำหน้าที่กำหนด static utility routine ในการสร้าง Servlet

2.4.7 Package: java.servlet.html

แพคเกจลำดับที่ 3 ซึ่งกำหนดใน Servlet API จะกล่าวถึงต่อไปนี้คือ java.servlet.html โดยแพคเกจนี้ประกอบด้วยคลาสที่สนับสนุน HTML 13 คลาส ซึ่งสนับสนุนการปฏิบัติการของ HTML เช่น ตาราง, การแทรกรูปภาพ และ ฟอร์ม ผู้เขียนจึงสามารถสร้าง HTML page โดยใช้ภาษา HTML จากนั้นจึงส่ง HTML page นั้นเขียนลง ServletOutputStream โดยคลาสต่างๆมีดังนี้

1. HtmlElement. สนับสนุนการกำหนด basic interface ประกอบด้วย 2 method ดังนี้

- wrap(). สำหรับรวม HTML tag pairs (เช่น <h1> และ </h1>) เข้ากับ element โดย tag ถูกระบุให้เป็น argument ใน wrap()
- write(). เขียน HTML ไปที่ OutputStream

2. HtmlText สนับสนุนข้อความ(text) ที่อยู่ภายใน tag ประกอบด้วย 2 method ดังนี้

- add(String), add(String tag). เพิ่มข้อความ(text) ซึ่งรวมไปถึงข้อความที่อยู่ใน tag
- addTag(String), addTag(String, String) เพิ่ม tag เดี่ยวที่ไม่เป็นคู่(non-paired tag) เช่น<p>

3. HtmlContainer Class สนับสนุน list ของ HtmlElement ที่อยู่ภายใน tags ซึ่ง HtmlContainer ประกอบด้วย beginning tags, list ของ HtmlElement และ ending tags HtmlContainer สนับสนุน wrap()., write() และ ชนิดต่างๆของ add() และ addTag() method โดย HtmlContainer ประกอบด้วย method ต่างๆดังนี้

- addImg(String). เพิ่ม <IMG> tag โดยที่ String คือ URLที่บรรจุรูปภาพที่ต้องการ

- addLink(String text, String URL) เพิ่ม link เมื่อ "text" อยู่ระหว่าง tags และ URL โดยเป็น URL ที่ต้องการจะ link ไป

4. HtmlApplet Class สนับสนุน applet HTML tag กับรายละเอียดของความกว้างและความสูง

5. HtmlChoice Class สนับสนุน HTML tag ที่ถูกเลือก

6. HtmlDefinitionList Class สนับสนุน dl, dt และ dd HTML tags

7.HtmlForm Class สนับสนุนการสร้าง HTML form ซึ่งประกอบด้วย input fields, check boxes, radio buttons, text area, selections และ submit buttons

8. HtmlFrameset Class สนับสนุน Frameset HtML tag

9. HtmlImageMap. Class สนับสนุน client-side image map HTML tags

10. HtmlList Class สนับสนุน ul และ ol HTML list tags

11. HtmlPages Class สนับสนุนการสร้าง web page ที่ประกอบด้วย header และ body areas

12. HTMLTable Class สนับสนุนการสร้างตารางจาก row element

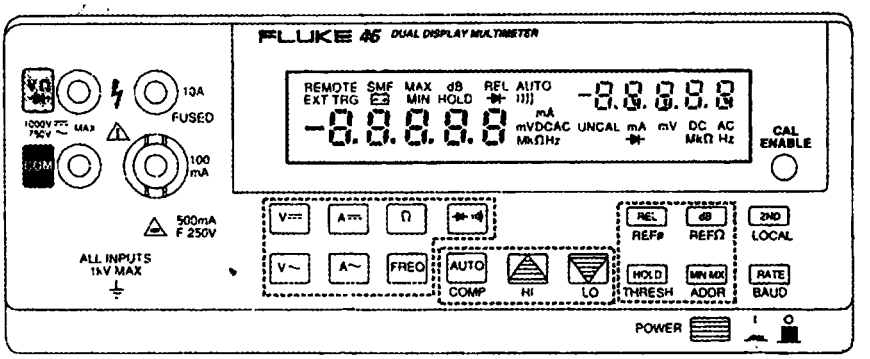
13. HtmlRow Class สนับสนุน HTML เพื่อกำหนดแถวของตาราง

2.5 Digital Multimeter (FLUKE 45)

คุณสมบัติของเครื่อง

- จอแสดงผลสามารถแสดงได้ 5 หลัก
- ความละเอียดวัดได้ถึง 1 μV DC , 10 μV AC, 1 m Ohm, 100nA DC Current, 100nA AC Current
- สามารถใช้มาตรฐานการส่งแบบ RS-232 เข้าสู่เครื่องคอมพิวเตอร์ส่วนบุคคล หรือเครื่องพิมพ์ได้
- ข้อมูลของสัญญาณแรงดันและกระแสแบบ RMS รวม AC +DC

เป็นเครื่องมือวัดที่สามารถวัดแรงดันไฟฟ้า, กระแส และความถี่ ที่แฝงอยู่ในสัญญาณไฟตรงและส่งเข้าสู่เครื่องคอมพิวเตอร์ส่วนบุคคลได้ โดยสามารถใช้มาตราฐานการส่งสัญญาณแบบ RS-232 หรือ IEEE-488.2 (GPIB) ก็ได้ ซึ่งสามารถตั้งค่าความเร็วในการส่งได้ และตั้งค่าความเร็วในการติดต่อสื่อสาร ได้ แบ่งเป็น 1200, 2400, 4800 หรือ 9600 bps ค่าสัญญาณที่วัดจะแสดงที่หน้าจอของเครื่อง สามารถแสดงได้ 2 สัญญาณพร้อมกัน (Dual Display)



รูปที่ 2.4 แสดงเครื่องดิจิตอลมัลติมิเตอร์ FLUKE 45

2.5.1 การติดต่อสื่อสารทางพอร์ทอนุกรมของเครื่อง FLUKE45

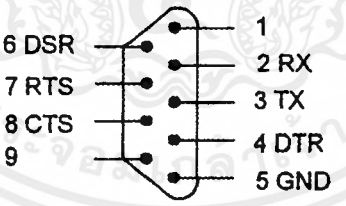
ใช้มาตรฐานการส่งแบบ RS-232 ซึ่งคอนเนคเตอร์ทางเครื่อง FLUKE 45 เป็นแบบ DB9 ตัวผู้ ขาที่จำเป็นต้องใช้ในการติดต่อคือ

ขา 2 ใช้ในการรับข้อมูล (RX)

ขา 3 ใช้ในการส่งข้อมูล (TX)

ขา 4 ใช้เป็นสัญญาณบอก DCE ว่าอุปกรณ์ DTE ที่อินเตอร์เฟสด้วยมีไฟเลี้ยงและพร้อมที่จะทำงาน

ขา 5 ใช้เป็นจุดอ้างอิงแรงดันสำหรับทุกสัญญาณในการอินเตอร์เฟส



รูปที่ 2.5 คอนเนคเตอร์แบบ DB9

2.6 การสื่อสารผ่านทางพอร์ทอนุกรม RS-232

ในการสื่อสารโดยใช้คอมพิวเตอร์นั้น มีรูปแบบที่ใช้ในการสื่อสารอยู่ 2 อย่างใหญ่ ๆ คือ การสื่อสารแบบอนุกรม และการสื่อสารแบบขนาน การสื่อสารแบบขนานนั้นสามารถส่งข้อมูลได้ทีละมากๆ เพราะมีสายสัญญาณหลายเส้น แต่ก็เป็นอุปสรรคในการสื่อสารในระยะทางที่ไกล การสื่อสารแบบอนุกรมจึงได้เปรียบในจุดที่ใช้สัญญาณเพียงแค่สองเส้น จึงเป็นการประหยัด และสะดวกในการใช้งาน

การสื่อสารข้อมูลหรืออินเทอร์เฟส ถึงกันนั้นย่อมต้องมีมาตราฐานในการสื่อสาร เพื่อให้เป็นที่เข้าใจกันทั้งฝ่ายส่งและฝ่ายรับ ที่ขื่อเรียกกันไปตามมาตราฐานต่างๆ เช่น อาร์เอส-232 (RS-232 ;Rercommemded Standard-232) , RS-422, RS-485 และอื่นๆ เป็นต้น

2.6.1ลักษณะสัญญาณที่ใช้ในการอินเทอร์เฟสตามมาตรฐานต่าง ๆ

มาตรฐาน RS-232-C ใช้สัญญาณเพียงเส้นเดียวในการส่งสัญญาณ โดยสัญญาณที่ส่งๆ ไปได้ทิศทางเดียว ในกรณีที่อัตราเร็วในการส่งข้อมูลมีค่าเท่ากับ 20k bps (กิโลบิตต่อวินาที) ซึ่งค่านี้เป็นค่าสูงสุดที่ใช้ในการส่งข้อมูล ระยะทางที่ใช้ส่งข้อมูลไม่ควรเกิน 50 ฟุต (ตามข้อกำหนดในมาตรฐาน) สำหรับการแทนแรงดันของระดับสัญญาณ (ในบทนี้เราแทนระดับสัญญาณด้วยลอจิกบวก) มีข้อกำหนดดังนี้ "1" แทนระดับแรงดันที่มีค่าระหว่าง +5 โวลต์ ถึง +15 โวลต์ "0" แทนระดับแรงดันที่มีค่าระหว่าง -5 โวลต์ ถึง -15 โวลต์

วงจรแปลงระดับแรงดัน

ปกติการอินเทอร์เฟสที่ออกแบบและจัดให้ใช้ควบคุมและตรวจสอบ (SENSE) ข้อมูล จะเป็นการอินเทอร์เฟสที่ใช้สัญญาณดิจิตอลเป็นหลักสัญญาณพวกนี้จะถูกส่งและรับมาจากรีจิสเตอร์ อินพุท/เอาท์พุท, อินพุทของ Interrupt-request, พอร์ทที่ใช้ในการทำ DMA และไทม์เมอร์/เคาน์เตอร์ การอินเทอร์เฟสที่กล่าวมาแล้วนี้ทุกตัวเป็นการใช้สัญญาณดิจิตอลที่มีระดับสัญญาณแบบ TTL ในการอินเตอร์เฟส ดังนั้นถ้าเรานำอุปกรณ์อื่นที่ใช้ระดับสัญญาณ TTL ในการอินเทอร์เฟสด้วยเช่นกัน การทำอินเทอร์เฟสเข้ากับเครื่อง PC จะทำได้โดยตรง แต่ในหลายกรณีสัญญาณที่ใช้ไม่ได้เป็นระดับสัญญาณ TTL หรือไม่ได้เป็นสัญญาณดิจิตอล และระยะทางที่ทำการอินเทอร์เฟสมีระยะห่างมากซึ่งก่อให้เกิดปัญหาหลายประการ เราสามารถแก้ปัญหาเหล่านี้ได้โดยการใช้การอินเทอร์เฟสแบบต่างๆ ดังที่จะกล่าวต่อไป

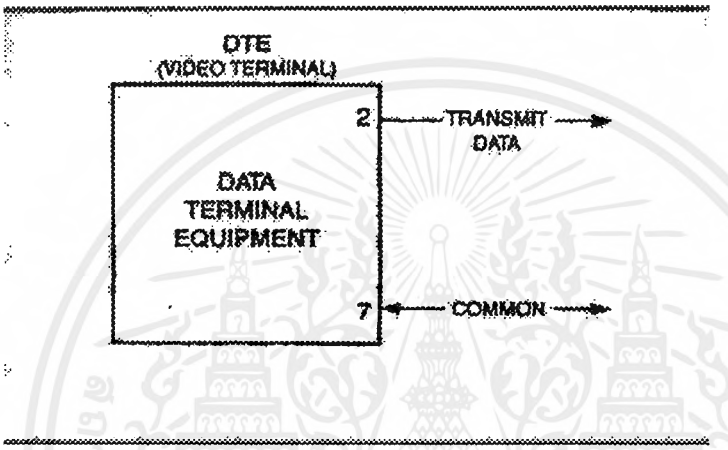
2.6.2 มาตรฐานการอินเทอร์เฟส RS - 232

จากเหตุผลที่ได้อธิบาย จะเห็นได้ว่าเรามีความจำเป็นที่จะต้องมีมาตรฐานการรับส่งข้อมูล ในปี 1969 EIA (Electronic Industries Association) ห้องวิจัย Bell และบรรดาผู้ผลิตอุปกรณ์สื่อสารได้ร่วมกันจัดตั้งมาตรฐาน EIA RS - 232 ซึ่งต่อมาไม่นานนักก็ได้มีการปรับปรุงแก้ไขอีกเล็กน้อยกลายเป็น RS-232-C และเมื่อไม่นานมานี้ก็ได้ออกมาตรฐาน RS-232-D และยังมีมาตรฐานคล้ายกันซึ่งออกโดยองค์กรระหว่างประเทศคือ Consultative Committee on International Telegraphy and Telephony (CCITT) เพื่อให้เราสามารถเข้าใจกับการอินเทอร์เฟส RS-232 ได้ดียิ่งขึ้น เราควรทำความเข้าใจกับวัตถุประสงค์หลักของ RS-232 ก่อน ซึ่งได้แสดงไว้อย่างชัดเจนในหัวข้อของเอกสารคือ

Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange (การอินเทอร์เฟสระหว่างอุปกรณ์เทอร์มินัลและอุปกรณ์สื่อสารข้อมูลที่ใช้วิธีการแลกเปลี่ยนข้อมูลไบนารีแบบอนุกรม)

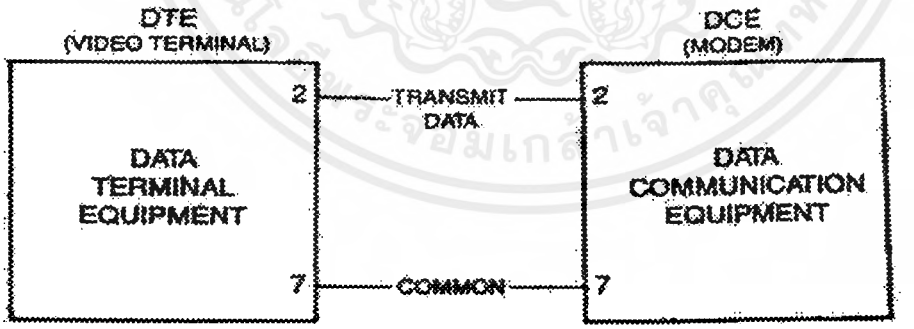
เราลองพิจารณาโครงสร้างเบื้องต้น การอินเทอร์เฟส RS-232 จะประกอบด้วยเส้นสายไฟเพียง 2 เส้น เส้นหนึ่งเป็นเส้นสำหรับรับส่งข้อมูล และอีกเส้นเป็นเส้นสำหรับอ้างอิงระดับแรงดันของวงจรอินเทอร์เฟส (Circuit common) ในส่วนแรงดันอ้างอิงนี้ มักมีผู้เข้าใจไขว้เขวว่าเป็นสิ่งเดียวกับกราวด์ (Ground) ซึ่งแท้ที่จริงแล้ววงจรอินเทอร์เฟสใช้สิ่งนี้อ้างอิงระดับแรงดันของวงจร เราควรจำไว้เสมอว่า ขาอ้างอิงแรงดัน (ขา 7) ในวงจรอินเทอร์เฟสเป็นสิ่งที่ขาดไม่ได้ไม่ว่าวงจรนั้นจะซับซ้อนหรือง่ายดายเพียงใด หากเราเริ่มด้วยการต่อขาอ้างอิงแรงดันนี้ เราก็ไม่ต้องพะวงกับเรื่องนี้อีกต่อไป

รูปที่ 2.6 แสดงอุปกรณ์ DTE เบื้องต้น (Data terminal Equipment) ซึ่งเป็นตัวเทอร์มินัลที่ประกอบด้วยคีย์บอร์ดและจอมอนิเตอร์ (ความแตกต่างระหว่าง DTE/DCE จะกระจ่างขึ้นในไม่ช้า) ตัวเลขแสดงภายในอุปกรณ์แสดงหมายเลขขาของตัวคอนเนคเตอร์ของอุปกรณ์



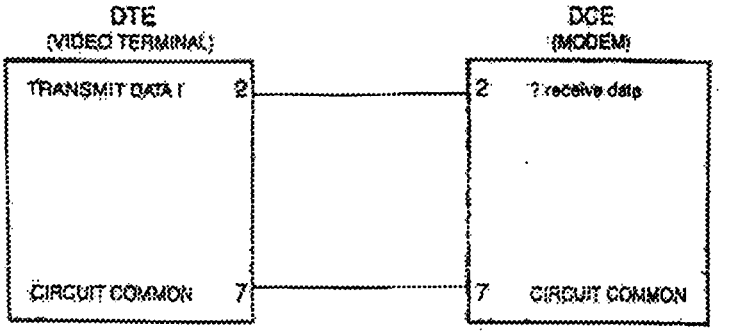
รูปที่ 2.6

การอินเทอร์เฟสใดๆ จะประกอบด้วยส่วนของตัวส่งข้อมูล และส่วนของตัวรับข้อมูล รูปที่ 2.6 เราสมมติให้ส่วนรับข้อมูลเป็นอุปกรณ์ DCE (Data Communication) ดั้งเดิมซึ่งก็คือ โมเด็มนั่นเอง



รูปที่ 2.7

หากเราพิจารณาตามรูป เราจะพบว่าข้อมูลที่ถูกส่งออกจากขา 2 ของอุปกรณ์ DTE จะรับเข้าไปยังอุปกรณ์ DCE จะเป็นข้อมูลที่รับได้ที่ขาของตัว DCE จะเป็นข้อมูลตัวเดียวที่ปรากฏบนขา 2 ของตัวมันเอง หรืออีกนัยหนึ่ง วลี "Transmit Data" มิได้มีส่วนในการกำหนดว่าอุปกรณ์ใดเป็นตัวต้นทางหรือปลายทาง ทั้งนี้ขึ้นอยู่กับมุมมองที่เราพิจารณา เราอาจกำหนดชื่อของขาคอนเนคเตอร์ไว้ในตัวอุปกรณ์ได้ดังรูปที่ 2.7



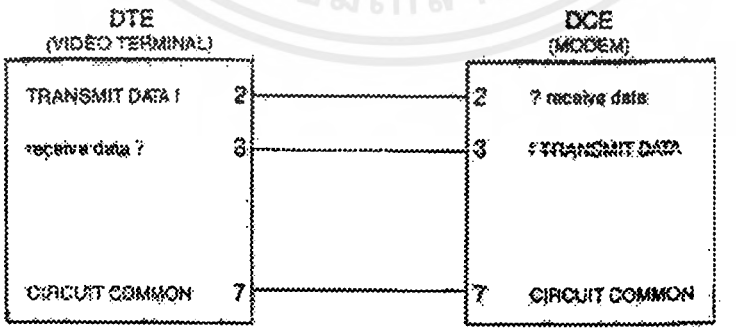
รูปที่ 2.8 อุปกรณ์ DTE และ อุปกรณ์ DCE เป็นคู่อุปกรณ์ร่วมที่ทำงานตรงข้ามกัน

จากรูปที่ 2.8 ข้อมูลที่ถูกส่งออกไปจากขา 2 ของ DTE เป็นข้อมูลตัวเดียวกันกับข้อมูลที่รับเข้ามาทางขา 2 ของ DCE ต่อไปนี้เราจะนิยามสัญญาณที่ส่งออกไปว่า "เอาท์พุท" และนิยามสัญญาณที่ส่งรับเข้ามาว่า "อินพุท" โดยถือว่าสัญญาณเป็นกิจกรรมทางไฟฟ้าที่เกิดขึ้นในกระบวนการอินเตอร์เฟส

สังเกตจากรูปที่ 2.8 ชื่อของอินพุทจะถูกแทนด้วยอักษรตัวพิมพ์เล็กกำกับด้วยเครื่องหมายคำถาม (?) และชื่อของเอาท์พุท แทนด้วยอักษรตัวพิมพ์ใหญ่กำกับด้วยเครื่องหมายอัศเจรีย์ (!) ทั้งนี้เพื่อให้ชัดเจนและง่ายต่อการเข้าใจ ซึ่งเป็นข้อตกลงที่ใช้เฉพาะในนี้เท่านั้น

2.6.3 รับส่งข้อมูลสองทิศทาง

ตามรูปที่ 2.8 เมื่อเทอร์มินัลรับข้อมูลจากคีย์บอร์ดแล้ว และส่งต่อไปยังโมเด็ม โมเด็มจะทำการส่งข้อมูลต่อออกไปทางสายโทรศัพท์ เราจะเห็นได้ว่าทั้งเทอร์มินัลและโมเด็มสามารถเป็นได้ทั้งอุปกรณ์ส่งและรับข้อมูล และยังสามารถส่งและรับข้อมูลในทิศตรงข้ามได้อีกด้วย ตัวอย่างเช่น เมื่อโมเด็มรับสัญญาณข้อมูลจากสายโทรศัพท์ ก็จะทำการส่งต่อไปให้เทอร์มินัล เมื่อเทอร์มินัลได้รับเอาท์พุทของโมเด็ม ก็จะทำการแสดงผลขึ้นที่หน้าจอมอนิเตอร์ เราจึงสามารถแทนการทำงานดังกล่าวได้ดังรูปที่ 2.9



รูปที่ 2.9 อุปกรณ์สามารถส่งและรับข้อมูลได้ทั้งสองทิศทาง

จากตัวอย่างที่กล่าวมาแล้ว การเดินทางของข้อมูลเป็นการแลกเปลี่ยนข้อมูลระหว่างอุปกรณ์สองตัว ซึ่งคล้ายกับการติดต่อกันระหว่างโทรศัพท์สองเครื่อง

ความแตกต่างระหว่างอุปกรณ์ DTE และ DCE ที่เราเห็นได้เบื้องต้นมีดังนี้

DTE ส่งเอาท์พุททางขา 2 และรับอินพุททางขา 3

DCE ส่งเอาท์พุททางขา 3 และรับอินพุททางขา 2

ในการเผชิญปัญหาการอินเทอร์เฟส ไม่ว่าจะง่ายดายหรือซับซ้อนเพียงใด เราจะเป็นต้องเริ่มต้นด้วยการตรวจสอบทิศทางของสัญญาณข้อมูลที่ขา 2 และ 3 เป็นอันดับแรก

(ตามคำจำกัดความโดย EIA อุปกรณ์ DTE หมายถึงอุปกรณ์ที่ข้อมูลมาสิ้นสุด และอุปกรณ์ DCE เป็นอุปกรณ์ที่ส่งผ่านข้อมูล ดังนั้นคอมพิวเตอร์ซึ่งสามารถทำได้ทั้งส่งผ่านข้อมูลและรับข้อมูล จึงไม่อาจระบุได้แน่ชัดว่าเป็นอุปกรณ์ DCE หรือ DTE)

2.6.4 การแฮนด์เชค

ถ้าหากการอินเทอร์เฟสเป็นแค่ดังที่กล่าวข้างต้น อาจจะงงว่าทำไมต้องมาสายสัญญาณมากมายถึง 21 เส้น ซึ่งที่จริงแล้ว การอินเทอร์เฟสยังมีเรื่องที่ซับซ้อนที่ยังต้องค่อยๆศึกษาอีกมาก จึงจะสามารถนำไปประยุกต์ใช้กับการอินเทอร์เฟสกับคอมพิวเตอร์ได้ สิ่งแรกที่เราจะขาดไม่ได้คือ วิธีการควบคุมการทำงานของอุปกรณ์สองตัวให้สัมพันธ์กันในการรับส่งข้อมูลในช่วงเวลาและใช้สัญญาณที่เหมาะสม ซึ่งก็คือกระบวนการแฮนด์เชค (Handshaking) นั่นเอง การแฮนด์เชคแบ่งได้เป็น 2 ประเภทคือ การแฮนด์เชคทางซอฟต์แวร์ และการแฮนด์เชคทางฮาร์ดแวร์

<u>การแฮนด์เชคทางซอฟต์แวร์</u> (Solfware Handshaking) เป็นวิธีหนึ่งในการควบคุมการทำงานของอุปกรณ์รับข้อมูลโดยส่งผ่านสัญญาณควบคุมไปพร้อมกับตัวข้อมูลที่ต้องการส่ง ตัวอย่างเช่น ในการพิมพ์ข้อความทางเครื่องพิมพ์ คอมพิวเตอร์จะส่งข้อมูลไปทีละบรรทัด มันจะแทรกอักษรควบคุมแสดงว่าสิ้นสุดแต่ละบรรทัด (End-Of-Line) เพื่อแจ้งเครื่องพิมพ์ว่าขณะนี้คอมพิวเตอร์กำลังรอสัญญาณตอบกลับอยู่ก่อนจะส่งข้อมูลบรรทัดถัดไป และเมื่อเครื่องพิมพ์รับข้อมูลเข้ามาและพิมพ์ไปจนจบบรรทัด ก็จะส่งสัญญาณไปบอกคอมพิวเตอร์ว่ามันพร้อมจะรับข้อความบรรทัดใหม่แล้ว ซึ่งเพียงเท่านี้ เราก็สามารถควบคุมการทำงานของเครื่องพิมพ์ได้ แต่ในความเป็นจริงแล้ว เครื่องพิมพ์บางรุ่นไม่สามารถแยกอักษรควบคุมออกจากข้อความที่ส่งเข้ามาได้

การใช้พาริตี้

พาริตี้เป็นวิธีการอย่างหนึ่งในการตรวจสอบความผิดพลาดในการส่ง และรับของข้อมูล เราจะใส่พาริตี้บิตเข้าไปในตอนท้ายของจำนวนชุดข้อมูล ซึ่งมีสองวิธีคือ

การใส่พาริตี้แบบคู่ (Even parity) เป็นการใส่ค่าบิตเพิ่มเข้าไปเพื่อให้จำนวนข้อมูลใน 1 ชุด เป็นจำนวนคู่

เช่น สมมติข้อมูลมี 7 บิต 1010010

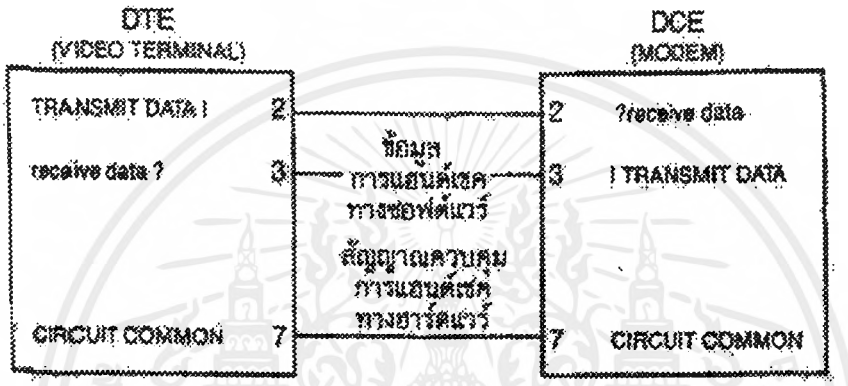
เมื่อเราใส่พาริตี้บิตเข้าไปจะกลายเป็น 10100101

ถ้าข้อมูล คือ 0001010 พาริตี้บิตที่ใส่เข้าไปจะเป็น "0" เพื่อที่จะทำให้ผลรวมของจำนวนบิตเป็นคู่ ดังเช่น 00010100

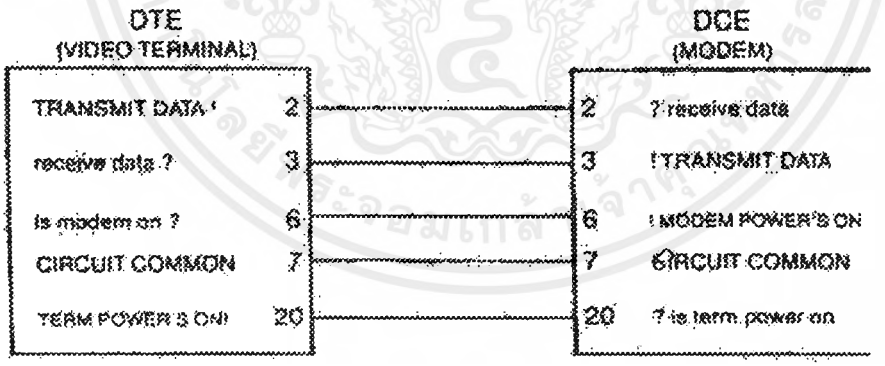
การใส่พาริตี้แบบคี่ (Odd parity) คล้ายคลึงกับพาริตี้แบบคู่ เพียงแต่เป็นการทำให้ผลรวมของจำนวนบิตของข้อมูลเป็นคี่

ในทางตรงกันข้าม การแฮนด์เชคทางฮาร์ดแวร์ (Hardware Handshaking) สามารถควบคุมเครื่องพิมพ์ได้ตั้งแต่ระดับฮาร์ดแวร์ โดยใช้การเปลี่ยนระดับแรงดันในสายสัญญาณควบคุมเป็นตัวระงับไม่ให้คอมพิวเตอร์ส่งข้อมูลเพิ่มเข้ามาอีก ซึ่งเป็นการหลีกเลี่ยงการใช้รหัสหรือโปรแกรม แต่การแฮนด์เชคทางฮาร์ดแวร์นั้นมีข้อจำกัด คือจำเป็นต้องมีสายสัญญาณควบคุมต่างหากสำหรับงานนี้โดยเฉพาะ ทำให้วิธีนี้ไม่เหมาะที่จะนำมาใช้ในการอินเทอร์เฟสกับโมเด็ม ตัวอย่างการแฮนด์เชคแสดงไว้ดังรูปที่ 2.10

ในรูปที่ 2.10 สัญญาณการอินเทอร์เฟสถูกแบ่งออกเป็น 2 ประเภทคือ สัญญาณข้อมูลและสัญญาณควบคุม เพื่อให้ง่ายต่อการเข้าใจ สัญญาณข้อมูลคือ สัญญาณที่เป็นอักษรข้อมูลหรือข้อความที่ต้องการรับส่งจริง ส่วนสัญญาณควบคุมจะหมายถึงสัญญาณอื่นๆ ที่เหลือทั้งหมด



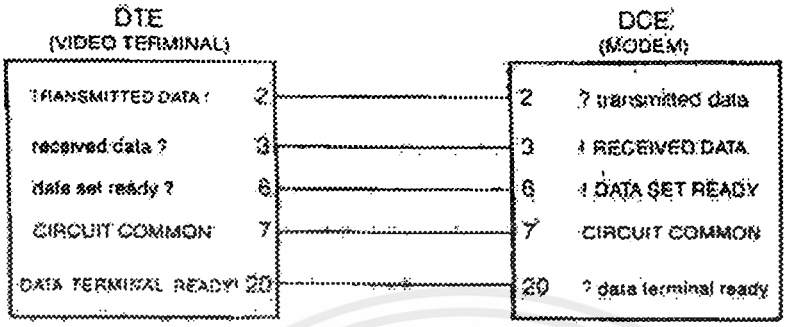
รูปที่ 2.10 สัญญาณควบคุมไม่มีข้อมูลเข้ามาเกี่ยวข้อง



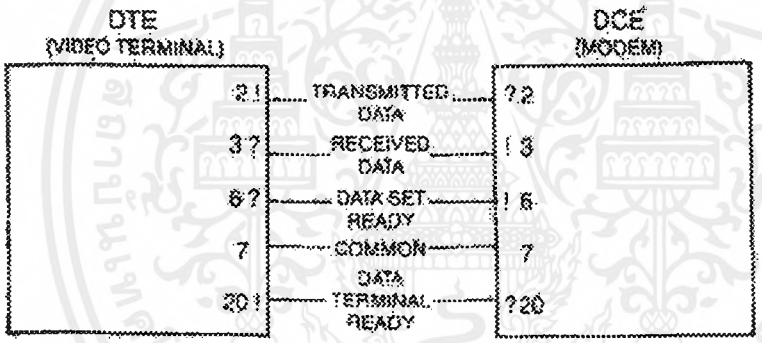
รูปที่ 2.11 คู่อินพุท/เอาท์พุทควบคุมสำหรับการแฮนด์เชค

การแฮนด์เชคระหว่างโมเด็มและเทอร์มินัลโดยการตรวจจับแรงดันไฟฟ้าของอีกอุปกรณ์หนึ่งเป็นเพียงวิธีการหนึ่งของมาตรฐานการอินเทอร์เฟส RS-232 ซึ่งในความเป็นจริง โมเด็มหรือเทอร์มินัลที่ใช้งานโดยทั่วไปไม่ได้มีความสามารถในการแฮนด์เชคแบบดังกล่าว ผู้ผลิตอุปกรณ์เหล่านั้นอาจจะใช้เทคนิคเฉพาะของตนขึ้นมาก็เป็นได้

แม้ว่ารูปแบบการอินเตอร์เฟสคอมพิวเตอร์อาจจะผิดแผกแตกต่างไปจากมาตรฐาน RS-232 บ้าง แต่สัญญาณข้อมูลและสัญญาณการแฮนด์เชคที่ปรากฏอยู่ยังเป็นมาตรฐานสากลอยู่ นิยามของแต่ละสัญญาณก็ยังคงใช้ได้ เริ่มต้น เราจะต้องนึกไว้เสมอว่า "DATA SET" ที่ใช้กันทั่วไปก็เป็นเพียงอีกชื่อเรียกของโมเด็มในรูปที่ 2.9 เป็นตัวอย่างการใช้ชื่อมาตรฐานในการเขียนวงจร



รูปที่ 2.12 ขาคอนเนคเตอร์ของ DTE และ DCE ต่างมีชื่อมาตรฐานเหมือนกัน



รูปที่ 2.13 การระบุชื่อขาคอนเนคเตอร์ไว้ระหว่าง DTE และ DCE

นอกจากชื่อมาตรฐานดังกล่าวแล้ว แต่ละสัญญาณยังมีตัวย่อดังต่อไปนี้

TRANSMITTED DATA	=	TD (หรือ TxD)
RECEIVED DATA	=	RD (หรือ RxD)
DATA TERMINAL READY	=	DTR
DATA SET READY	=	DSR

จากรูปที่ 2.13 สังเกตว่าชื่อสัญญาณและหมายเลขขาต่างๆ ของคอนเนคเตอร์ของ DTE และ DCE จะเหมือนกัน แต่มีการทำงานตรงกันข้ามกันคือ ถ้าขาหนึ่งของทางด้าน DTE เป็นอินพุทา ขาที่มีหมายเลขตรงกันของด้าน DCE จะเป็นเอาท์พุท ซึ่งในทางตรงกันข้ามก็เป็นสจริงเช่นเดียวกัน โดยชื่อสัญญาณจะคงเดิม ตัวอย่างเช่น ขา 2 ทางด้านโมเด็มแม้จะมีหน้าที่รับข้อมูล แต่ก็ยังคงเรียกว่าขา "TRANSMITTED DATA" (ข้อมูลส่ง) มีเพียงเครื่องหมาย ? เท่านั้นที่บ่งบอกหน้าที่อินพุทของขานั้น

ในการอินเทอร์เฟสอุปกรณ์ที่ทำงานเป็นคู่กัน (DTE/DCE) นั้นดูเป็นเรื่องธรรมดา แต่หากว่าอุปกรณ์ที่เราต้องการใช้งานกลับมีการทำงานที่เหมือนกัน (DTE/DCE หรือ DCE/DCE) เช่น พอร์ตอนุกรมบนเจ้าคอมพิวเตอร์ของคุณถูกวางสายไว้เป็นแบบ DCE (มีสัญญาณข้อมูลส่งออกทางขา 3) และโมเด็มที่คุณต้องการต่อก็เป็นแบบ DCE เช่นกัน ในสถานการณ์นี้ ถ้าไม่มีการดัดแปลงใดๆ ในการอินเทอร์เฟส อุปกรณ์ทั้งคู่จะพยายามส่งข้อมูลจากสายสัญญาณเดียวกัน และการแฮนด์เชคจะทำงานกลับกัน

ถ้าคุณตกอยู่ในเหตุการณ์ข้างต้น คุณอาจคิดว่าโมเด็มเสีย หรือคอมพิวเตอร์เสีย ทั้งที่จริงแล้วเกิดความบกพร่องในการอินเทอร์เฟส ดังนั้นในการอินเทอร์เฟสอุปกรณ์ที่ออกแบบมาลักษณะเดียวกันเข้าด้วยกันต้องคำนึงถึงหน้าที่และการทำงานของแต่ละสัญญาณในแต่ละขาของคอนเนคเตอร์มากกว่าที่จะไปยึดติดกับชื่อและหมายเลขของขานั้นๆ ซึ่งแม้ว่าจะมีการเปลี่ยนชื่อหรือเลขหมายของขาไป แต่หน้าที่การทำงานของสัญญาณจะยังไม่เปลี่ยนแปลง

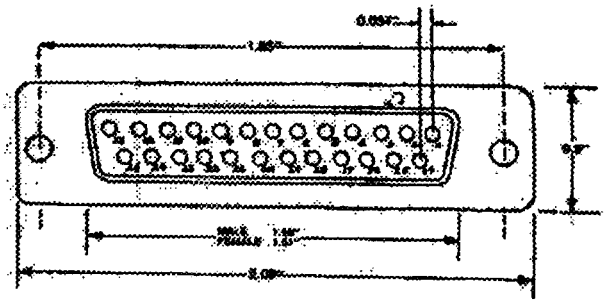
2.6.5 ขอบเขตความคอมแพติเบิลกับ RS-232

คุณสมบัติทางไฟฟ้า (ระดับแรงดัน ฯลฯ) ของการอินเทอร์เฟสได้รับการตรวจรับรอง ถ้าอุปกรณ์นั้นถูกอ้างว่าคอมแพติเบิลกับ RS-232 ย่อมหมายความว่าเราสามารถนำอุปกรณ์นั้นไปต่อกับอุปกรณ์อื่นที่คอมแพติเบิลกับ RS-232 ได้ โดยไม่ทำให้เกิดความเสียหายแก่อุปกรณ์ทั้งคู่ ซึ่งเงื่อนไขนี้ช่วยให้เรามั่นใจได้ว่าอุปกรณ์ทั้งคู่มีระบบไฟฟ้าที่ทำงานด้วยกันได้ โดยไม่ทำให้เกิดความบกพร่องในการรับส่งข้อมูล

ระดับแรงดันสำหรับค่า "ศูนย์" และ "หนึ่ง" ต้องเป็นไปตามที่ระบุไว้ในมาตรฐาน สัญญาณข้อมูลและสัญญาณควบคุมจะมีค่าระดับแรงดัน/ค่าทางตรรกะแตกต่างกัน ซึ่งอาจเป็นเหตุให้อุปกรณ์สับสนว่าแรงดันนั้นเป็น "0" หรือ "1" แต่ในขณะนี้แทนที่จะพิจารณาถึงระดับแรงดัน เราจะมองเพียงสถานะของขาอินพุทควบคุมนั้นว่าเป็นค่าที่ส่งออกไปเป็นค่าที่อนุญาตให้ทำงาน (Enabled) หรือค่าที่ใช้ห้ามการทำงาน (Disabled) ส่วนด้านขาเอาท์พุทควบคุมก็พิจารณาในทำนองเดียวกันว่าค่าที่ส่งออกมานั้นบอกให้มันเริ่มทำงาน (Asserted) หรือหยุดทำงาน (Inhibited)

คอนเนคเตอร์ที่เป็นปลั๊ก (ตัวผู้ : มีขา หรือ PIN) สามารถใส่พอดีกับที่เป็นแจ็ค (เต้ารับตัวเมีย) ในการผลิตคอนเนคเตอร์ตัวผู้ซึ่งมีขาสัญญาณยื่นออกมามันจะมีขอบโลหะล้อมโดยรอบเพื่อความปลอดภัยของผู้ใช้จากอันตรายไฟฟ้าดูดที่อาจเกิดระหว่างการใช้งานได้ โดยตัวคอนเนคเตอร์ตัวผู้นี้จะต้องต่อเข้ากับคู่คอนเนคเตอร์ตัวเมียของมันได้อย่างดี เช่น คอนเนกเตอร์ DB-25 ตัวผู้จะเสียบเข้าได้กับคอนเนคเตอร์ DB-25 ตัวเมียได้พอดี

แม้ว่าการกำหนดขาของคอนเนคเตอร์ได้รับการระบุไว้ในมาตรฐาน แต่รูปแบบตัวคอนเนคเตอร์นั้นยังขาดมาตรฐานที่จะมารองรับ อย่างไรก็ตาม คอนเนคเตอร์ DB-25 ซึ่งมีรูปร่างคล้ายอักษร D มีการนำไปใช้ในวงกว้างจนอาจจะสามารถกล่าวว่าเป็นมาตรฐานในเชิงปฏิบัติได้ ซึ่งผู้ผลิตอาจจะผลิตรูปทรงอื่นก็ไม่ผิดอะไร ตัวอย่างที่เห็นได้ชัดเจนคือ คอมพิวเตอร์ IBM PC/AT ออกแบบให้ใช้คอนเนคเตอร์แบบ 15 ขา โดยยังคงอ้างว่าคอมแพติเบิลกับ RS-232 ซึ่งในทางทฤษฎีแล้ว คอนเนคเตอร์ที่ถือว่าคอมแพติเบิลแท้จริง (Truly Compatible) จะต้องขารับส่งสัญญาณต่างๆ ทั้งหมด 21 สัญญาณ แต่ในความเป็นจริงแล้วผู้ผลิตได้มีการลดทอนขาที่ไม่จำเป็นต้องใช้งานออกไป



รูปที่ 2.14 คอนเนคเตอร์ แบบ DB-25

ตามมาตรฐานแล้ว สายเคเบิลที่ใช้ในการอินเทอร์เฟสจะมีปลายสายที่เป็นคอนเนคเตอร์ตัวเมียสำหรับต่อกับอุปกรณ์ DTE และปลายสายที่เป็นคอนเนคเตอร์ตัวผู้สำหรับต่อกับอุปกรณ์ DCE หรืออีกนัยหนึ่งก็คือ คอนเนคเตอร์สำหรับการอินเตอร์เฟสบนอุปกรณ์ DTE ต้องเป็นชนิดตัวผู้ และบนอุปกรณ์ DCE จะเป็นชนิดเต้ารับตัวเมีย เราจึงอาจสังเกตชนิดของคอนเนคเตอร์บนอุปกรณ์เพื่อระบุว่าเป็นอุปกรณ์ DTE หรือ DCE และยังสามารถเป็นลางบอกปัญหาที่อาจจะมีมากขึ้นหากตัวคอนเนคเตอร์ของทั้งสองอุปกรณ์ที่เราต้องการจะอินเตอร์เฟสเข้าด้วยกันเป็นชนิดเดียวกัน ซึ่งอาจจะไม่เคยตั้งใจสังเกตสิ่งเหล่านี้มาก่อน อย่างไรก็ตาม ในทางปฏิบัติตามปกติเราจะพบว่าคอนเนคเตอร์ทั้งของ DTE และ DCE มักเป็นแบบเต้ารับตัวเมียเหมือนกัน แต่หากเราพบอุปกรณ์ที่มีคอนเนคเตอร์เป็นตัวผู้ เราก็ค่อนข้างมั่นใจได้ว่าอุปกรณ์เป็น DTE

สาเหตุที่ผู้ผลิตมักนำมากล่าวอ้างในการหลีกเลี่ยงการใช้ คอนเนคเตอร์ตัวผู้กับฮาร์ดแวร์คือ ความวิตกว่าผู้ซื้ออาจจะต้องเปลี่ยนสายเคเบิลที่ใช้อินเทอร์เฟสบ่อยๆ เมื่อมีการอินเทอร์เฟสกับอุปกรณ์ใหม่ ซึ่งเราอาจเปรียบเทียบได้กับการที่เราซื้อเครื่องใช้ไฟฟ้ามาแล้วพบว่าปลายสายไฟแทนที่จะเป็นปลั๊กเสียบ มันกลับเป็นเต้ารับแทน ซึ่งคงทำให้เกิดปัญหากับผู้ซื้อในการที่จะนำเจ้าเครื่องใช้ไฟฟ้านั้นไปใช้งาน โดยอาจเสียทั้งเงินและเวลาในการเปลี่ยนให้เป็นปลั๊กไฟตามปกติ แต่ที่แน่นอน ผู้ซื้อคงต้องหงุดหงิดกับสินค้าที่ออกแบบโดยไม่ยึดหลักการหรือมาตรฐานสากลเช่นนี้

คุณสามารถมั่นใจกับสัญญาณบนขาคอนเนคเตอร์ได้บ้าง ขาที่พอจะอ้างอิงได้อย่างค่อนข้างมั่นใจได้แก่

ขา 2 ส่ง/รับข้อมูล

ขา 3 ส่ง/รับข้อมูล

ขา 7 ระดับแรงดันอ้างอิงของวงจร (CIRCUIT COMMON)

ถ้าอุปกรณ์ที่ซื้อมาไม่เป็นไปตามกติกาเรื่องของสัญญาณนี้ มันคงยากอุปกรณ์นั้นจะเป็นไปตามมาตรฐานข้ออื่นๆ

เครื่องพิมพ์เป็นอุปกรณ์ DTE จากการที่ในสมัยก่อนยังไม่มีจอมอนิเตอร์ใช้ เทอร์มินัลยุคนั้นจึงมีลักษณะเป็นเครื่องพิมพ์โดยตอบสนองการทำงานของคอมพิวเตอร์ด้วยการพิมพ์ข้อความออกมา ดังนั้นระบบอินเตอร์เฟสของเครื่องพิมพ์จึงยังคงยึดถือแนวทางเดิม และมักถูกออกแบบให้เป็นอุปกรณ์ DTE

ส่วนผู้ผลิตไมโครคอมพิวเตอร์ไม่สามารถรู้ล่วงหน้าได้ว่า คอมพิวเตอร์นั้นจะถูกนำไปใช้งานอะไร กับอุปกรณ์ประเภทไหน จึงจำเป็นต้องออกแบบให้มีพอร์ตอนุกรม 2 พอร์ตคือ พอร์ตสำหรับเครื่องพิมพ์ (วงจรเป็นแบบ DCE) และ พอร์ตสำหรับการสื่อสารหรือโมเด็ม (วงจรเป็นแบบ DTE)

โมเด็มเป็นอุปกรณ์ DCE โมเด็มถูกผลิตให้เป็นอุปกรณ์ DCE เพื่อสร้างมาตรฐานในการอินเตอร์เฟสโมเด็ม แต่ก็ยังมีโมเด็มบางรุ่นที่ผู้ผลิตได้เพิ่มสวิตช์สำหรับเลือกแบบการอินเทอร์เฟสไว้ด้วย ซึ่งหากคอมพิวเตอร์ที่ใช้บังเอิญมีพอร์ตโมเด็มต่อมาเป็นแบบ DTE คุณก็ยังสามารถนำไปใช้งานได้ทันที นอกจากนี้ ยังเห็นได้ว่าขนาดนิยามของโมเด็มที่เป็นอุปกรณ์สื่อสารข้อมูล (DCE) แท้ๆ ก็ยังอาจถูกบิดเบือนไปได้

สรุป คุณสมบัติเบื้องต้นของ RS-232 มีดังนี้

- *ข้อมูล* รับส่งข้อมูลระหว่างอุปกรณ์ทางคอนเนคเตอร์ขา 2 และ ขา 3
- *การแฮนด์เชคทางซอฟต์แวร์* เป็นการผนวกอักษรควบคุมการรับส่งข้อมูลเข้าไปกับชุดข้อมูลที่ต้องการส่ง
- *การแฮนด์เชคทางฮาร์ดแวร์* เป็นการผ่านสัญญาณควบคุมไปตามสายเคเบิลที่เชื่อมต่อระหว่างอุปกรณ์ทั้งสอง โดยจะต่อขาเอาท์พุท เข้ากับอินพุท
- แม้ว่าชื่อสัญญาณและหมายเลขประจำขาของคอนเนคเตอร์สำหรับการอินเทอร์เฟสของ DTE และ DCE จะเหมือนกัน แต่จะทำหน้าที่ตรงข้ามกัน โดยเมื่อมีด้านหนึ่งเป็นอินพุท อีกด้านที่ต่อกับขาก็จะเป็นเอาท์พุท

2.7 โปรแกรมภาษา Visual Basic 5

ในปัจจุบันระบบปฏิบัติการ (Operating System) ในลักษณะของ Window ได้เข้ามาแทนที่ระบบปฏิบัติการในลักษณะเดิม ซึ่งส่วนใหญ่ที่นิยมใช้กันอยู่คือ MS-DOS Windows ได้ทำการเปลี่ยนรูปแบบของคอมพิวเตอร์ให้มีความสามารถมากกว่าการเป็นเพียงคอมพิวเตอร์ส่วนบุคคล ด้วยการเพิ่มความสามารทางด้านการติดต่อระหว่างคอมพิวเตอร์และผู้ใช้ ซึ่งเรียกว่า "User Interface" เข้าไป โดยทำให้คอมพิวเตอร์มีการใช้งานที่ง่าย (User Friendly) มากขึ้น ด้วยการพัฒนาโปรแกรมต่างๆ ให้อยู่ในรูปแบบของ Graphic User Interface (GUI) ซึ่งแตกต่างจากรูปแบบของโปรแกรมในลักษณะเดิมที่ใช้งานอยู่บน MS-DOS แต่เดิมการแสดงผลจะอยู่ในรูปแบบของตัวอักษร ซึ่งค่อนข้างมีจำกัด โดยเฉพาะรูปแบบของคำสั่งที่ใช้จะเป็นแบบป้อนทีละบรรทัด หรือที่เรียกว่า *"Command line"* โดยผู้ใช้จะต้องทำการเรียนรู้และจดจำรูปแบบของแต่ละคำสั่งให้ถูกต้องแม่นยำ จึงจะใช้งานโปรแกรมนั้นๆ ได้เป็นอย่างดี

ในการพัฒนาโปรแกรมขึ้นใช้งานก็เช่นเดียวกัน แต่เดิมโปรแกรมเมอร์พัฒนาโปรแกรมอยู่บน MS-DOS จึงต้องเปลี่ยนแปลงรูปแบบและแนวความคิดมาทำการพัฒนาโปรแกรมบน Windows แทน ในยุคแรกของการพัฒนาโปรแกรม Windows นั้นค่อนข้างจะทำได้ยาก โดยอาจใช้ภาษา C หรือ Software Development Kit (SDK) มาเขียนโปรแกรมแต่ก็ต้องเขียน Routine ต่างๆ เป็นจำนวนมากเพื่อพัฒนาโปรแกรมหนึ่งๆ ให้แล้วเสร็จ ด้วยเหตุนี้ Microsoft จึงนำภาษาคอมพิวเตอร์ชื่อ "BASIC" ในรูปแบบเดิมมาพัฒนาขึ้นใหม่ ให้ชื่อว่า "Visual BASIC" โดยเริ่มจาก Visual Basic Version 1.0 และได้มีการพัฒนามาเป็นลำดับจนกระทั่งเป็น Version 5.0 ดังเช่นปัจจุบัน

2.7.1 ความเป็นมาของภาษา BASIC

ภาษา BASIC ถูกสร้างขึ้นในปี 1963 โดย Hohn Thomas Kurtz ที่วิทยาลัย Dartmouth ในเบื้องต้นพวกเขามีจุดมุ่งหมายในการพัฒนาภาษา BASIC ขึ้น เพื่อใช้ในการสอนแนวในการเขียนโปรแกรม (Programming Concept) โดยเน้นรูปแบบของภาษานั้นต่อทั้งการเข้าใจและการใช้งาน รวมทั้งทำงานในลักษณะของ Interpreter ซึ่งแตกต่างจากภาษาคอมพิวเตอร์อื่นๆ ในยุคนั้นที่จะอาศัย Job Control Language (JCL) และขั้นตอนในการ Compile และ Link ผลก็คือ ภาษา BASIC ได้กลายเป็นภาษาคอมพิวเตอร์ที่นิยมใช้กันอย่างกว้างขวาง โดยเฉพาะในหมู่ผู้ใช้คอมพิวเตอร์ส่วนบุคคล จึงอาจกล่าวได้ว่า ภาษา BASIC ได้รับการพัฒนาควบคู่กันไปกับการพัฒนาคอมพิวเตอร์ส่วนบุคคลในปี 1970 Microsoft ได้เริ่มผลิตตัวแปรภาษา BASIC ใน ROM ซึ่งเรียกว่า ROM-Based BASIC ขึ้น เช่น ชิป Radio Sheek TRS-80 เป็นต้น ต่อมาก็ได้พัฒนาเป็น GW-BASIC ซึ่งเป็น Interpreter ภาษาที่ใช้กับ MS-DOS และในปี 1982 Microsoft QuickBasic ได้รับการพัฒนาขึ้น โดยการเพิ่มความสามารถในการ Compile ให้เป็น Executed Program รวมทั้งทำให้ BASIC มีความเป็น "Structured Programming" มากขึ้น โดยการตัด Line Number ทิ้งไป เพื่อลบข้อกล่าวหาว่าเป็นภาษาคอมพิวเตอร์ที่มีโครงสร้างในลักษณะ Spaghetti Code (Logical Flow ของภาษาขาดโครงสร้าง) มาใช้รูปแบบของ Subprogram และ User Defined รวมทั้งการใช้ Structured Data Type และการพัฒนาการใช้งานด้านกราฟฟิกให้มีการใช้งานระดับที่สูงขึ้น รวมทั้งมีการใช้เสียงประกอบได้เหมือนกับภาษาคอมพิวเตอร์อื่น เช่น C หรือ Pascal

2.7.2 สาเหตุที่ต้องใช้ Visual Basic

สามารถใช้ Visual Basic สร้างโปรแกรมบน Windows โดยอาศัยการออกแบบโปรแกรมในลักษณะ Visualize ซึ่งใช้การกำหนดตำแหน่งของ Object เหล่านี้จะเปลี่ยนไปตามเหตุการณ์ (Event) ต่างๆ ที่เกิดขึ้น เช่น การเคลื่อนเม้าส์ หรือการรับข้อมูลจากคีย์บอร์ด ในการกำหนดขั้นตอนการทำงานให้กับ Object ภายใต้ Event ใดๆ จะใช้ภาษา BASIC เข้ามาช่วยในการเขียนโปรแกรม ดังนั้น อาจกล่าวได้ว่า การพัฒนาโปรแกรมบน Windows โดยใช้ Visual BASIC มีความง่ายและสะดวกในการใช้งาน รวมทั้งมีขั้นตอนน้อย เพียงแต่เลือก From และ Control ที่เหมาะสม แล้ววาดลงบนจอภาพเพื่อใช้ติดต่อกับผู้ใช้ จากนั้นจึงทำการเขียนภาษา BASIC เพื่อสร้างโปรแกรมด้วยตัวเอง ด้วยวิธีที่ง่ายแล้วเร็วกว่าที่คิด จึงทำให้ผู้ใช้เรียนรู้ได้ภายในเวลา 2-3 ชั่วโมง สามารถสร้างโปรแกรมบน Windows เป็นโปรแกรมแรกได้

นอกจากนี้ Visual Basic ยังใช้ได้ตั้งแต่ User ระดับต้นเพื่อใช้สร้างโปรแกรมง่ายๆ บน Windows หรือ โปรแกรมเมอร์ระดับกลางที่จะเรียกใช้ฟังก์ชันการทำงานต่างๆ ของ Visual Basic ได้อย่างมีประสิทธิภาพ ตลอดจนโปรแกรมเมอร์ระดับอาชีพที่จะพัฒนาโปรแกรมในระดับสูง โดยการใช้ Object Linking and Embedding (OLE) และ Windows Application Programming Interface (API) มาประกอบในการเขียนโปรแกรม

2.7.3 ขั้นตอนในการพัฒนาโปรแกรมของ Visual Basic

ขั้นตอนในการพัฒนาโปรแกรม ประกอบด้วยขั้นตอนหลัก 2 ขั้นตอน ดังนี้

<u>ขั้นตอนที่ 1</u> สร้างจอภาพ

ในขั้นตอนนี้ จะทำการออกแบบ From เพื่อใช้ในการติดต่อกับผู้ใช้ หรือที่เรียกว่า การออกแบบ „User Interface" ในการพัฒนาโปรแกรมเดิม ขั้นตอนนี้จะใช้เวลาและค่าใช้จ่ายสูง เนื่องจากจะต้องเขียนโปรแกรมเพื่อสร้างจอภาพต่างๆ จากนั้นต้อง Compile โปรแกรมนั้น แล้ว Run จึงจะเห็นจอภาพที่จัดทำขึ้น แต่สำหรับ Visual Basic ขั้นตอนนี้จึงสามารถทำได้อย่างง่าย เพียงแต่นำเอา Control ต่างๆ ใน Toolbox ที่ต้องการใช้งานมาวางไว้บน From ซึ่งทำให้ประหยัดเวลาและสามารถเห็นลักษณะจอภาพที่ออกแบบได้ในขณะนั้นเลย

ขั้นตอนที่ 2 เขียนโปรแกรม

เมื่อทำการวาง Control ต่างๆ ลงบน From เรียบร้อยแล้ว (Control ต่างๆ ที่นำมาวางบน From จะเรียกว่า "Obeject") ขั้นตอนต่อมาคือ การเขียนโปรแกรมเพื่อกำหนดการทำงานให้กับแต่ละ Oject ภายใต้เหตุการณ์ต่างๆ (Event) ที่จะเกิดขึ้นกับจอภาพนั้นๆ

บทที่ 3
การดำเนินการวิจัย

การดำเนินการวิจัยโครงงานนี้มีลำดับขั้นการดำเนินงานตามลำดับหัวข้อดังต่อไปนี้คือ

3.1 การติดตั้ง Telemetering Web Server

1. จัดเตรียมเครื่องคอมพิวเตอร์ที่จะนำมาใช้เป็นเครื่อง Server โดยได้จัดเตรียมเป็นเครื่องคอมพิวเตอร์ที่มีคุณสมบัติเบื้องต้นดังต่อไปนี้

- CPU AMD K6-2 3D 300@350 MHz.
- หน่วยความจำ SDRAM 64 MB
- Harddisk 1.7 GB
- LAN card 10 MBps

2. ติดตั้งระบบปฏิบัติการ(OS) สำหรับโครงงานนี้เลือกใช้ windows 95 มาเป็นระบบปฏิบัติการบนเครื่อง server

3. ติดตั้ง Java Development Kit เวอร์ชัน 1.1.6 (jdk 1.1.6) ซึ่งประกอบไปด้วยโปรแกรม และเครื่องมือ สำหรับการ complie และ run applets หรือ applications ซึ่งเขียนขึ้นโดยใช้ภาษาจาวา

4. ติดตั้ง Java Servlet Development Kit เวอร์ชัน 2.0 (jsdk 2.0) ประกอบด้วยเครื่องมือที่ใช้ในการ complie และ run Java Servlet และทำหน้าที่เป็น web server ให้กับ web ที่สร้างด้วย Java Servlet

5. ติดตั้งโปรแกรม Sambar Server เวอร์ชัน 4.2 ซึ่งเป็น multi-threaded HTTP,FTP และ Proxy server ที่มีการปฏิบัติงานบน windows 95

6. เซตค่า Configuration ของระบบ FTP Server ให้ Anonymous user สามารถใช้งานได้ เพื่อให้สามารถ Download ค่าแรงดันไฟฟ้าที่วัดได้ในวันต่างๆ

3.2 โปรแกรม Java Servlet สำหรับแสดงค่าแรงดันไฟฟ้า

โปรแกรมจาวาที่ได้ทำการพัฒนามีลำดับขั้นการทำงานดังนี้

3.4 การทดสอบการรับค่าแรงดันไฟฟ้าทางพอร์ทอนุกรมจากเครื่อง FLUKE 45

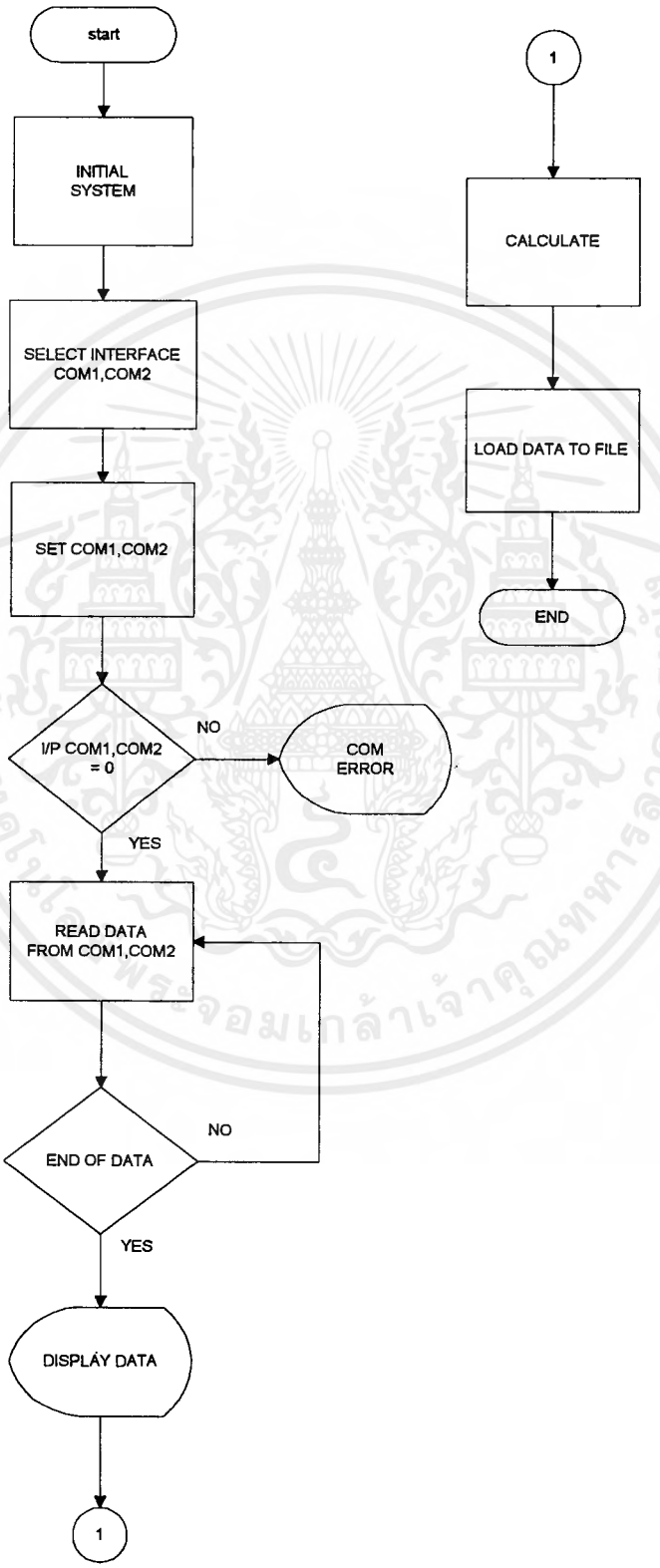
ในการทดสอบนี้ จะทำการต่อเครื่อง FLUKE 45 เข้ากับคอมพิวเตอร์ส่วนบุคคล และทำการวัดค่าแรงดันไฟฟ้าที่ศูนย์วิจัยและบริการคอมพิวเตอร์ ลาดกระบัง และใช้โปรแกรมวิชวลเบสิคที่ได้ทำการเขียนไว้ ในการรับค่าจากเครื่อง FLUKE45 ผ่านทางพอร์ท Com1 ของเครื่องคอมพิวเตอร์ส่วนบุคคล

จากการทดสอบพบว่า โปรแกรมที่ได้ทำการพัฒนานี้สามารถรับค่าจากเครื่องมัลติมิเตอร์ โดยจะแสดงผลขึ้นทาง TEXT BOX ของหน้าต่างของโปรแกรม ดังนี้ ทำให้ทราบว่าโปรแกรมที่เลือกใช้นี้สามารถรับค่าทางพอร์ทอนุกรมจากเครื่องมัลติมิเตอร์ FULKE 45 ได้

ในการทดลองรับและเก็บค่าจะได้กล่าวในบทต่อไป

3.5 โปรแกรม Visual Basic ที่ใช้รับค่าแรงดันไฟฟ้าจากเครื่อง FLUKE45

แสดงเป็นลำดับขั้นการทำงานได้ดังนี้



บางส่วนของโปรแกรมวิชวลเบสิคที่ได้ทำการพัฒนา ในการรับค่าแรงดันไฟฟ้า

```
Public Sub CommInput(thiscomm As dwComm, commdata As String)
    Dim use$
    Dim cpos%
    Dim filenum
    Dim Data!, Data1!

    If commdata <> "" Then
        ' Substitute the CR with a CRLF pair, dump the LF

        For cpos% = 1 To Len(commdata$)
            Select Case Asc(Mid$(commdata$, cpos%))
                Case 13
                    use$ = use$ + Chr$(13) + Chr$(10)
                Case 10
                    ' Dump the line feeds
                Case Else
                    use$ = use$ + Mid$(commdata$, cpos%, 1)
            End Select
Next cpos%

        TermText.SelStart = Len(TermText.Text)
        TermText.SelLength = 0
        TermText.SelText = use$
        TermText.SelStart = Len(TermText.Text)
    End If

    If carry% = 1 Then
            Data1! = 220
            filenum = FreeFile
        Open "c:\progra~1\project\backup\test.txt" For Append As filenum
        Print #filenum, Date, Time, "....................................", Data1!
        Close filenum
            carry% = 0
Else
            Data1! = 220
```

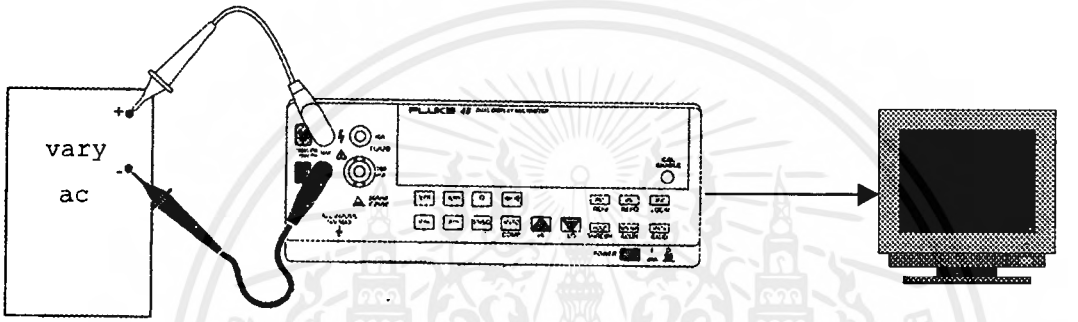
```
        Data! = Val(commdata$)

    If Mid$(Data!, 1, 3) < Mid$(Data1! * 0.9, 1, 3) Then
        filenum = FreeFile
    Open "c:\progra~1\project\backup\test.txt" For Append As filenum
    Print #filenum, Date, Time, "....................................", Data!
    Close filenum

Else
    Data1! = 220
    Data! = Val(commdata$)
    If Mid$(Data!, 1, 3) > Mid$(Data1! * 1.1, 1, 3) Then
    filenum = FreeFile
    Open "C:\progra~1\project\backup\test.txt" For Append As filenum
    Print #filenum, Date, Time, "....................................", Data!
    Close filenum
    End If
End If
  Image1.Visible = False
  If Mid$(use$, 1, 11) = Chr$(13) + Chr$(10) Then
  TermText.Text = " "
  Image1.Visible = True
  Beep
End If
  End If
End Sub
```

จากตัวอย่างโปรแกรมข้างต้นเป็นช่วงที่ใช้ทำงานในการรับค่าแรงดันจากเครื่องมัลติมิเตอร์ FLUKE 45 และนำมาคำนวณ เมื่อค่าแรงดันไฟฟ้าเปลี่ยนไปมากกว่า หรือน้อยกว่า 220 โวลต์ไป 10 เปอร์เซ็นต์ ก็จะทำการบันทึกค่าแรงดันไฟฟ้า พร้อมทั้งวันที่และเวลาลงไปในไฟล์ชื่อ test.txt ซึ่งสามารเรียกดูได้ทางเว็ปบราวเซอร์ โดยใช้ โปรแกรม Java ในดึงค่าข้อมูลมาแสดงผลที่หน้าจอโฮมเพจ ดังที่ได้กล่าวมาแล้วในหัวข้อ 3.2 และโปรแกรมทั้งหมดดูได้จากภาคผนวกท้ายเล่ม

บทที่ 4
การทดลองและผลการทดลอง

4.1 การทดลองโปรแกรมรับค่าแรงดันจากเครื่องวัดแรงดัน FLUKE 45

เนื่องจากโปรแกรมที่ได้ทำการออกแบบไว้ จะทำการรับค่าแรงดันไฟฟ้าจากเครื่องวัดแรงดัน FLUKE 45 ด้วยมาตราฐานการส่งข้อมูล RS 232 เข้ามาและทำการแปลค่าสัญญาณ RS 232 ออกเป็นตัวเลขแรงดันที่ส่งเข้ามา จากนั้น เมื่อค่าแรงดันที่ส่งมามีค่าเปลี่ยนแปลงไปจากค่าแรงดัน 220 โวลต์ 2 เปอร์เซ็นต์ โปรแกรมที่ทำการออกแบบไว้จะทำการบันทึกค่าแรงดันพร้อมทั้งเวลาและวันที่ที่แรงดันเกิดการเปลี่ยนแปลงดังกล่าวเอาไว้ การทดลองโปรแกรมที่ได้ทำการออกแบบไว้จึงทำการต่อวงจรดังรูปที่ 4.1



รูปที่ 4.1 แสดงระบบที่ทำการทดสอบโปรแกรมรับค่าแรงดัน

ระบบที่แสดงดังในรูปที่ 4.1 ประกอบด้วย

1. เครื่องวัดแรงดันไฟฟ้า FLUKE 45
2. เครื่องคอมพิวเตอร์ส่วนบุคคลที่มีข้อมูลจำเพาะดังนี้
   - 2.1 หน่วยประมวลผลกลาง เบอร์ AMD K-6 300
   - 2.2 ขนาดของหน่วยความจำบนบอร์ด 64 MB
   - 2.3 สัญญาณนาฬิกาของหน่วยประมวลผลกลาง 300 MHz
   - 2.4 สัญญาณนาฬิการของบอร์ด 100 MHz
3. หม้อแปลงเปลี่ยนแรงดันไฟฟ้า (VARY AC) ขนาด 300 วัตต์

การทดลองกระทำโดย

1. ทำการจ่ายค่าแรงดัน 220 โวลต์จากไฟฟ้าทั่ว ๆ ไปแก่หม้อแปลงเปลี่ยนแรงดันไฟฟ้า และทำการปรับค่าแรงดันที่ได้จากหม้อแปลงให้มีค่าเป็น 220 โวลต์
2. นำเครื่องวัดแรงดันไฟฟ้า FULKE 45 วัดแรงดันที่ได้จากหม้อแปลงเปลี่ยนแรงดันไฟฟ้า
3. ทำการเปิดเครื่องคอมพิวเตอร์ส่วนบุคคล จากนั้นทำการรันโปรแกรมเก็บค่าแรงดันไฟฟ้า เมื่อโปรแกรมรันถึงคำสั่งให้เก็บค่าแรงดันไฟฟ้าแล้ว ค่าแรงดันที่ปรากฏใน TEXT BOX จะปรากฏค่าแรงดันขนาด 220 โวลต์อยู่ตลอดเวลา

4. เมื่อค่าแรงดันที่ปรากฏที่ TEXT BOX ครบ 1 นาที ทำการเปลี่ยนแรงดันที่ได้จากหม้อแปลงเปลี่ยนแรงดันเป็นค่า 218 โวลต์ เป็นเวลานาน 10 วินาที จากนั้น ทำการลดค่าแรงดันไปเรื่อย ๆ ครั้งละ 2 โวลต์ และให้ค่าแรงดันคงที่อยู่ที่ค่าดังกล่าวเป็นเวลา 10 วินาที จนกระทั่งค่าแรงดันไฟฟ้ามีค่าลดลงถึง 210 โวลต์ ทำการสังเกตุค่าแรงดันที่ปรากฏบน TEXT BOX
5. ทำการเปลี่ยนแรงดันที่ได้จากหม้อแปลงเปลี่ยนแรงดันเป็นค่า 222 โวลต์ เป็นเวลานาน 10 วินาที จากนั้นทำการเพิ่มค่าแรงดันไปเรื่อย ๆ ครั้งละ 2 โวลต์ และให้ค่าแรงดันคงที่อยู่ที่ค่าดังกล่าวเป็นเวลา 10 วินาที จนกระทั่งค่าแรงดันไฟฟ้ามีค่าเพิ่มขึ้นถึง 230 โวลต์ ทำการสังเกตุค่าแรงดันที่ปรากฏปน TEXT BOX

6. บันทึกผลการทดลอง

ค่าแรงดันที่อ่านได้จากเครื่อง FLUKE45 เมื่อทำการเปลี่ยนค่าแรงดันจากหม้อแปลง	ค่าแรงดันที่ปรากฏบน TEXT BOX	ค่าแรงดันที่บันทึกลงบน File ที่ทำการเก็บค่า
220	220	3/6/99 10:33:25 AM.....220
220	220	-
220	220	-
220	220	-
220	220	-
220	220	-
220	220	-
220	220	-
220	220	-
220	220	-
220	220	-
220	220	-
218	218	-
218	218	-
218	218	-
218	218	-
218	218	-
218	218	-
216	216	-
216	216	-
216	216	-

ค่าแรงดันที่อ่านได้จากเครื่อง FLUKE45 เมื่อทำการเปลี่ยนค่าแรงดันจากหม้อแปลง	ค่าแรงดันที่ปรากฏบน TEXT BOX	ค่าแรงดันที่บันทึกลงบน File ที่ทำการเก็บค่า
216	216	-
216	216	-
216	216	-
216	216	-
214	214	3/6/99 10:35:25 AM.....214
214	214	3/6/99 10:35:30 AM.....214
214	214	3/6/99 10:35:35 AM.....214
214	214	3/6/99 10:35:40 AM.....214
214	214	3/6/99 10:35:45 AM.....214
214	214	3/6/99 10:35:50 AM.....214
212	212	3/6/99 10:35:55 AM.....212
212	212	3/6/99 10:36:00 AM.....212
212	212	3/6/99 10:36:05 AM.....212
212	212	3/6/99 10:36:10 AM.....212
212	212	3/6/99 10:36:15 AM.....212
212	212	3/6/99 10:36:20 AM.....212
210	210	3/6/99 10:36:25 AM.....210
210	210	3/6/99 10:36:30 AM.....210
210	210	3/6/99 10:36:35 AM.....210
210	210	3/6/99 10:36:40 AM.....210
210	210	3/6/99 10:36:45 AM.....210
210	210	3/6/99 10:36:50 AM.....210
*	*	*
222	222	-
222	222	-
222	222	-
222	222	-
222	222	-
222	222	-
224	224	-
224	224	-

ค่าแรงดันที่อ่านได้จากเครื่อง FLUKE45 เมื่อทำการเปลี่ยนค่าแรงดันจากหม้อแปลง	ค่าแรงดันที่ปรากฏบน TEXT BOX	ค่าแรงดันที่บันทึกลงบน File ที่ทำการเก็บค่า
224	224	-
224	224	-
224	224	-
224	224	-
226	226	3/6/99 10:45:15 AM.....226
226	226	3/6/99 10:45:20 AM.....226
226	226	3/6/99 10:45:25 AM.....226
226	226	3/6/99 10:45:30 AM.....226
226	226	3/6/99 10:45:35 AM.....226
226	226	3/6/99 10:45:40 AM.....226
228	228	3/6/99 10:45:45 AM.....228
228	228	3/6/99 10:45:50 AM.....228
228	228	3/6/99 10:45:55 AM.....228
228	228	3/6/99 10:46:00 AM.....228
228	228	3/6/99 10:46:05 AM.....228
228	228	3/6/99 10:46:10 AM.....228
230	230	3/6/99 10:46:15 AM.....230
230	230	3/6/99 10:46:20 AM.....230
230	230	3/6/99 10:46:25 AM.....230
230	230	3/6/99 10:46:30 AM.....230
230	230	3/6/99 10:46:35 AM.....230
230	230	3/6/99 10:46:40 AM.....230

4.2 สรุปผลการทดลอง

ในการทดลองใช้หม้อแปลงไฟในการเปลี่ยนค่าแรงดันไฟฟ้า เนื่องมาจากสถานที่ที่ทำโครงงานพิเศษนี้ทำการทดลองที่ศูนย์วิจัยและบริการคอมพิวเตอร์ ซึ่งมีแรงดันไฟฟ้าที่ค่อนข้างเสถียรมากๆ ทำให้ไม่เห็นการเปลี่ยนแปลงของค่าแรงดันไฟฟ้าสักเท่าไรนัก จึงต้องใช้หม้อแปลงไฟฟ้าช่วยในการปรับเปลี่ยนค่าแรงดัน เพื่อให้เห็นผลการทดลองที่ชัดเจนขึ้น

จากการทดลองจะเห็นว่าค่าแรงดันไฟฟ้าที่ปรากฏบน TEXT BOX ตรงกับค่าแรงดันไฟฟ้าที่แสดงบนหน้าจอของเครื่องมัลติมิเตอร์ และค่าที่ปรากฏในไฟล์ที่ทำการเก็บค่าแรงดันไฟฟ้า จะเก็บเฉพาะค่าที่เปลี่ยนแปลงไปจาก 220 โวลต์ (ซึ่งเป็นค่าเริ่มต้นที่ได้ทำการตั้งไว้เป็นแรงดันอ้างอิง) มากกว่า 2 เปอร์เซ็นต์ของ 220 โวลต์ (215.6 โวลต์ และ 224.4 โวลต์) ขึ้นไป เมื่อค่าแรงดันไฟฟ้าเปลี่ยนไปไม่มากกว่า 2 เปอร์เซ็นต์ของ 220

โวลต์ จะไม่ทำการเก็บค่าไว้ ซึ่งในการใช้งานจริง การเก็บค่าแรงดันไฟฟ้าที่เปลี่ยนแปลงจะตั้งไว้ที่ 10 เปอร์เซ็นต์ ของแรงดันไฟฟ้าปกติ เนื่องจากเราต้องการที่จะนำไปใช้ในโรงงานที่มีเครื่องจักรขนาดใหญ่ เพื่อที่จะสามารถเก็บค่าแรงดันไฟฟ้าที่ตกลง หรือเพิ่มขึ้นเมื่อทำการเปิดปิดเครื่องจักร

บทที่ 5
สรุป

เมื่อได้ทำการดำเนินการตามขั้นตอนที่ได้วางไว้ พบว่าสามารถทำการส่งข้อมูลแรงดันไฟฟ้าผ่านทางระบบเครือข่ายคอมพิวเตอร์ โดยค่าแรงดันไฟฟ้าที่แสดงผลนั้นเป็นไปตามช่วงที่กำหนดคือ ค่าแรงดันที่เปลี่ยนแปลงไปจาก 220 โวลต์ อยู่ในช่วง +/-10 เปอร์เซ็นต์

5.1 การประยุกต์ใช้งาน

1. สามารถนำแนวทางของโครงงานพิเศษนี้ไปประยุกต์ใช้ในรับ/ส่งค่าทางเครือข่ายคอมพิวเตอร์ โดยใช้มาตราฐานการส่ง RS-232
2. สามารถนำไปวัดค่าแรงดันในสถานที่โรงงานหรือในส่วนพื้นที่ที่ห่างไกลและอันตราย โดยที่ไม่ต้องใช้คนเข้าไปปฏิบัติงาน
3. สามารถนำไปประยุกต์ใช้ในการควบคุมอุปกรณ์ที่อยู่ห่างไกล

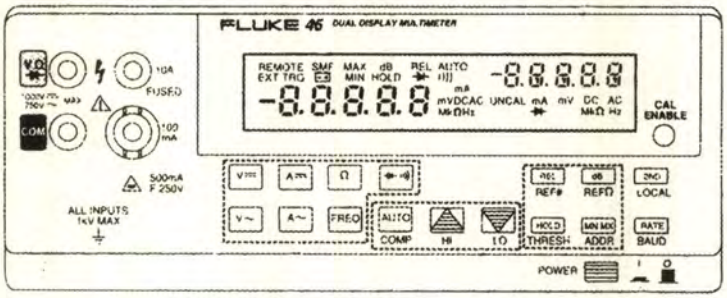
5.2 แนวทางในการพัฒนาโครงงาน

จากการทดลองรับค่าแรงดันแล้วส่งผลข้อมูล ให้ปรากฏบนเว็ปบราวเซอร์ สามารถนำข้อมูลแรงดันไฟฟ้ามาทำการเขียนกราฟเพื่อทำการวิเคราะห์ หาช่วงที่แรงดันไฟฟ้าตกลง ซึ่งแสดงให้ทราบถึงการใช้ไฟฟ้าในปริมาณที่มากกว่าปกติในช่วงเวลานั้นๆ

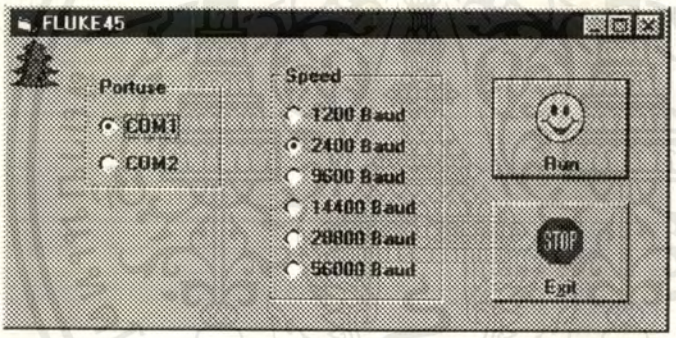
เมื่อทราบข้อมูลคำสั่งควบคุมเครื่องมัลติมิเตอร์ จะทำให้สามารถสั่งงานทางเว็ปบราวเซอร์ได้ ซึ่งเป็นการเพิ่มประสิทธิภาพของโครงงานพิเศษนี้

ภาคผนวก ก.
วิธีการใช้งาน

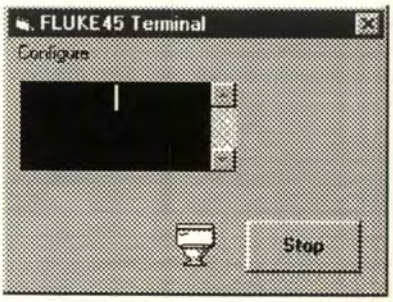
- **การใช้เครื่องมัลติมิเตอร์** FLUKE 45



1. เปิดเครื่องโดยกดปุ่ม POWER (สีเขียว)
2. กดปุ่มวัดแรงดันไฟสลับ (V~)
3. กดปุ่มให้ทำการส่งค่าแรงดันเข้าเครื่องคอมพิวเตอร์ โดยกดปุ่ม 2ND และ กดปุ่ม MIN MAX ตั้งค่าไว้ที่ 10 โดยทำการกดปุ่มสามเหลี่ยม (HI) และกดปุ่ม AUTO
4. ทำการเปิดไฟล์ชื่อ FLUKE.exe จะปรากฏหน้าจอดังรูป

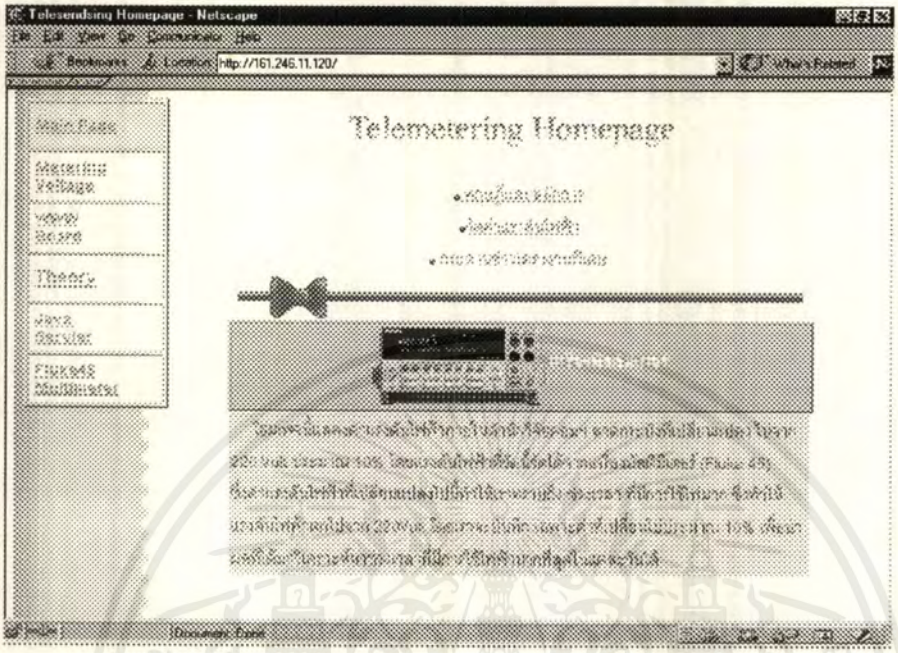


5. กดเลือกพอร์ตอนุกรมในการติดต่อ, เลือกความเร็วในการรับข้อมูล และทำกดปุ่มเริ่มทำงาน (RUN) เมื่อกด Exit จะเป็นการยกเลิกการทำงาน
6. เมื่อเลิกค่าตามที่ต้องการแล้วจะปรากฏหน้าจอใหม่ แสดงค่าแรงดันไฟฟ้าที่วัดได้จากเครื่องมัลติมิเตอร์ ปุ่มยกเลิก (STOP) กดเมื่อต้องการเลิกการทำงาน

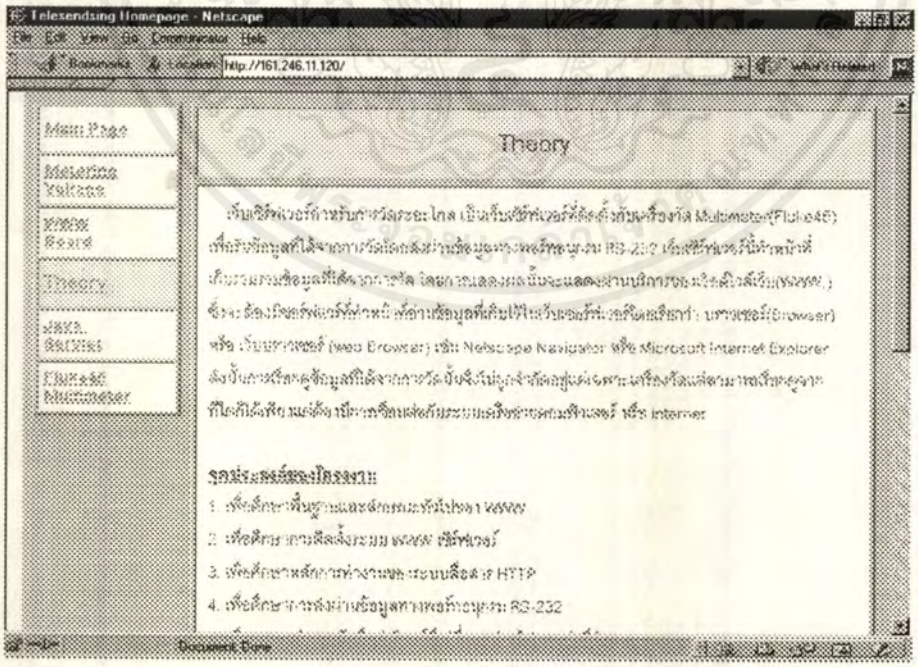


- การแสดงผลบน Home page

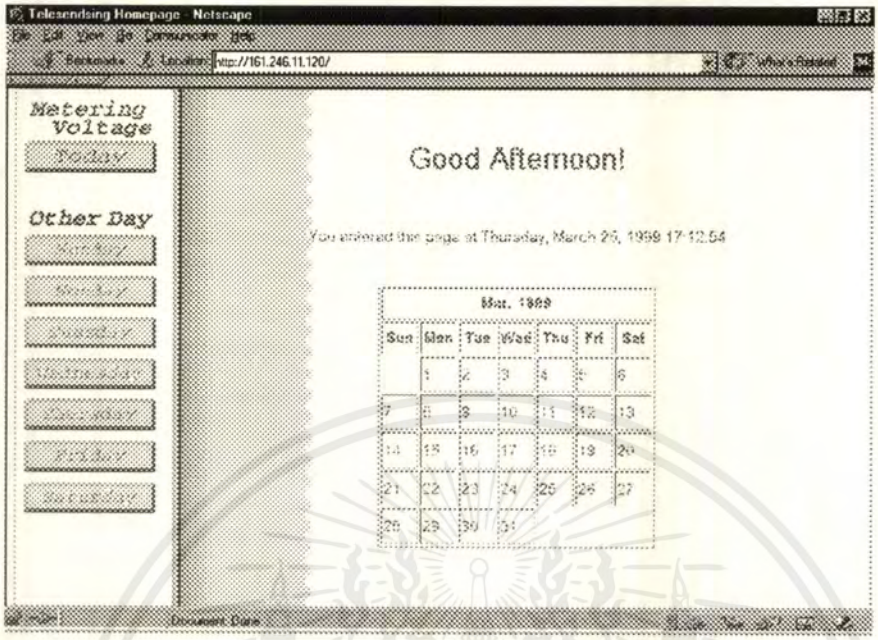
1. เมื่อทำการเปิดเว็ปบราวเซอร์สามารถเรียกดูค่าแรงดันไฟฟ้าที่วัดได้ โดยเข้าไปที่ http://161.246.11.120 จะปรากฏหน้าต่างดังรูป



2. เมื่อทำการคลิกเมาส์ที่ "ทฤษฎีและ หลักการ" หรือ "Theory"ในเมนู จะเข้าสู่หน้าต่างต่อไป ซึ่งมีเนื้อหาเกี่ยวกับจุดประสงค์ของโครงงาน และทฤษฎีที่เกี่ยวข้อง



3 เมื่อทำการคลิกเมาส์ที่ "Metering Voltage" ในเมนูก็จะเข้าสู่หน้าต่างต่อไป สามารถทำการเลือกดูแรงดันไฟฟ้าของวันนี้ หรือวันที่ผ่านมาภายในหนึ่งอาทิตย์ได้ โดยทำการกดที่ปุ่มต่างๆ ทางซ้ายมือ



4 หน้าต่างใหม่ที่เปิดขึ้นเพื่อแสดงค่าแรงดันไฟฟ้าในวันที่ผู้ใช้ต้องการทราบ

ภาคผนวก ข.
โปรแกรมจาวา

```
//Today.java

import java.io.*;
import java.math.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class OpenText extends HttpServlet
{
    public void service(HttpServletRequest req,HttpServletResponse res)throws IOException
    {

    String dout;
    DataInputStream din=new DataInputStream(new FileInputStream("d:/backup/project/test.txt"));
    res.setContentType("text/html");
    PrintWriter out =new PrintWriter(res.getOutputStream());
    out.println("<html>");
    out.println("<head><title>Voltage in my project room</title></head>");
    out.println("<body background=http://161.246.11.120/blstar.gif bgcolor=#ffffff text=#0f9bdf
link=#ff80c0> ");
    out.println("<center><img src=http://161.246.11.120/Vtoday.gif width=280 height=60
ALT\"df\"></center>");
    out.println("<center>");
    out.println("<img SRC=http://161.246.11.120/beau1.gif  width=540 height=50 ALT\"sfg\" >");
    out.println("</center>");
    out.println("<center><h2>แรงดันไฟฟ้าที่เปลี่ยนแปลงจาก 220 Volt ไป 10%</h2></center>");
    out.println("<center><b><u>หมายเหตุ</u> แรงดันไฟฟ้านี้วัดได้จากสำนักวิจัยและบริการคอมพิวเตอร์
(CRSC)พระจอมเกล้าฯ ลาดกระบัง</center>");
    out.println("<center><table cellpadding=2 cellspacing=0 width=60% border=0>");
    out.println("<tr bgcolor=#a7ddfe><font COLOR=></font>");
    out.println("<td align=left><font color=#ffffff size=3><b>วันที่ / เวลา </b></font></td>");
    out.println("<td align=right><font color=#ffffff size=3><b>แรงดันไฟฟ้า (Volt)</b></font></td>");
```

```
        out.println("</tr></table></center>");

    try{
        while((dout=din.readLine())!=null)
        {
        out.println("<center><table cellpadding=1 cellspacing=0 width=60%>");
        out.println("<tr bgcolor=#e0faff>");
        out.println(" <td width=50% align=left valign=top>");
        out.println("<ul>");
        out.println("<font size=3>");
        out.println(""+dout);
        out.println("</font></ul>");
        out.println("</td></tr></table></center>");
        }
    }
    catch(EOFException eof) {
        }
    catch(Exception e){
        return;
        }
        out.println("</body></html>");
        out.close();
        }
    }
```

ภาคผนวก ค.
โปรแกรมภาษาวิชวลเบสิค

```
Option Explicit
Dim commdata$
Dim indata$, Source$, Search$, Ans$, keep$
Dim carry%
Dim z%, q%, w%, X%, Y%, Found%


'Private Sub Command1_Click()

'------OLE Control--------
'Dim myobject As Object
'Dim mychart As Object
'Set myobject = CreateObject("Excel.application")
'myobject.Visible = True
'myobject.Workbooks.Add
'Dim a(8000) As String
'Dim i%, N%

'Source$ = TermText.Text
'Search$ = "#"
'Found% = InStr(1, Source$, Search$)
'X% = Found%

'Source$ = TermText.Text
'Search$ = ","
'Found% = InStr(1, Source$, Search$)
'q = Found%
'N = q
'z = q - 1
'indata$ = Mid$(TermText.Text, 1, z%)
'a(1) = Right$(indata$, z)
'myobject.Cells(1, 1).Value = a(1)

'For i = 2 To 8000
'Y% = N + 1
'If Y% = X% Then
'GoTo c:
'Else
'Source$ = TermText.Text
'Search$ = ","
'Found% = InStr(Y%, Source$, Search$)
'q = Found%
'z = q - 1
'indata$ = Mid$(TermText.Text, 1, z%)
'w = z - N
'a(i) = Right$(indata$, w)
'myobject.Cells(i, 1).Value = a(i)
'N = q
'GoTo D:
'myobject.Range("a1:a10000").Select
'c:
'CommonDialog1.Filter = "Microsoft Excel Workbook|*.XLS|All Files|*.*"
'CommonDialog1.Flags = &H4&
'CommonDialog1.FileTitle
'CommonDialog1.Action = 2
'End If
''D:
'Next i
'End Sub
```

```
Private Sub Command2_Click()
Comm.CloseComm
End Sub

'
' We open the port and get things started here
'

Private Sub Form_Load()
    ' We load the configuration form so that the default
    ' settings can be read by OpenThePort
    Commcfg.Show 1
    'TermText.Move 0, 0, ScaleWidth, ScaleHeight
    'CommDemo.WindowState = 2
carry% = 1
End Sub



Private Sub Form_Unload(Cancel As Integer)
    Set Comm = Nothing
End Sub


Private Sub MenuConfigure_Click()
    Commcfg.Show 1
End Sub

Private Sub TermText_KeyPress(KeyAscii As Integer)
    If Not (Comm Is Nothing) Then
        Comm.CommOutput (Chr$(KeyAscii))
    End If
    KeyAscii = 0
End Sub

Public Sub CommInput(thiscomm As dwComm, commdata As String)
    Dim use$
    Dim cpos%
    Dim filenum
    Dim Data!, Data1!

    If commdata <> "" Then
        ' Substitute the CR with a CRLF pair, dump the LF

        For cpos% = 1 To Len(commdata$)
            Select Case Asc(Mid$(commdata$, cpos%))
                Case 13
                    use$ = use$ + Chr$(13) + Chr$(10)
                Case 10
                    ' Dump the line feeds
                Case Else
                    use$ = use$ + Mid$(commdata$, cpos%, 1)
            End Select
Next cpos%

        TermText.SelStart = Len(TermText.Text)
        TermText.SelLength = 0
        TermText.SelText = use$
```

```
            TermText.SelStart = Len(TermText.Text)
        End If

          If carry% = 1 Then
                Data1! = 220
                filenum = FreeFile
            Open "d:\backup\project\test.txt" For Append As filenum
            Print #filenum, Date, Time,
    "....................................", Data1!
            Close filenum
                carry% = 0
    Else
                Data1! = 220
                Data! = Val(commdata$)
             If Mid$(Data!, 1, 3) < Mid$(Data1! * 0.98, 1, 3) Then
                filenum = FreeFile
             Open "D:\backup\project\test.txt" For Append As filenum
                  Print #filenum, Date, Time,
    "....................................", Data!
             Close filenum

     Else
             Data1! = 220
             Data! = Val(commdata$)
             If Mid$(Data!, 1, 3) > Mid$(Data1! * 1.02, 1, 3) Then
             filenum = FreeFile
             Open "D:\backup\project\test.txt" For Append As filenum
             Print #filenum, Date, Time,
    "....................................", Data!
             Close filenum
             End If
             End If
    End If
       Image1.Visible = False
       If Mid$(use$, 1, 11) = Chr$(13) + Chr$(10) Then
       TermText.Text = " "
       Image1.Visible = True

    End If

    End Sub

    Public Sub CommEvent(thiscomm As dwComm, ev As String)
        TermText.SelStart = Len(TermText.Text)
        TermText.SelText = vbCrLf & ev & vbCrLf
        TermText.SelStart = Len(TermText.Text)
    End Sub

    Private Sub Timer1_Timer()
        If Not (Comm Is Nothing) Then Comm.Poll
    End Sub
```

หนังสืออ้างอิง

1. กิตติ ภักดีวัฒนะกุล, จำลอง ครูอุตสาหะ, "Visual Basic 5 ฉบับ โปรแกรมเมอร์", พิมพ์ครั้งที่ 1, หจก. ไทยเจริญการพิมพ์, 2541.
2. ทวีชัย ภูริทรัพย์, "ไขปัญหา RS-232", พิมพ์ครั้งที่ 2, บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน), 2538.
3. Martin D. Seyer, "COMPLETE GUIDE TO RS232 AND PARALLEL CONNECTIONS", First Edition' Prentice-Hall, Inc., 1988.
4. Martin D. Seyer, "RS-232 MADE EASY", Second Edition, Prentice-Hall, Inc., 1991.
5. Lyle J Graham, "Your IBM PC", Osborne/McGraw-Hill, 1983.
6. TJ Byers, "Inside the IBM PC AT", International Edition, McGraw-Hill Book Co., 1987.
7. Nathan Gurewich and Ori Gurewich, "Teach Yourself Visual Basic in 21 Days", First Edition, Sams Publishing, 1993.
8 Cassady-Dorion, Luke, "Industrial strength Java", Indianapolis, IN : New Riders, 1997

- Talking to an IBM Series/1 computer that is running either the Real Time Programming System V5.1 or the Event Driven Executive V3.0 IBM operating systems
- Simple communications with another serial device.

The ACS program supports the following applications:

- Talking to an IBM computer that is running either the IBM VM/370 System Control Program or the IBM MVS TSO System Control Program
- Sophisticated communications with another PC
- Sophisticated communications with another serial device.

Later we will look at an application example of these programs: the COMM program for accessing on-line data and the ACS program for PC-to-PC file transfer.

SERIAL COMMUNICATIONS PARAMETERS

There are several parameters governing how serial communications data is formatted. It will be useful for you to familiarize yourself with these parameters since the communications support program as well as the external device may require you to specify these values prior to using the communications link.

What parameters you must be aware of and how you can change them will depend on the program or device you are using. At the PC end, a given parameter may be set automatically by the communications program, or else the program will prompt you for this information. With a modem or external device, parameters may be set either by the manufacturer or by you with a switch. The device or program manuals should tell you how to set up these parameters.

Many problems with serial communications are caused by a discrepancy between the settings at the PC and the serial device. If you do have problems, double-check these parameters.

The most common serial communications parameters that you will encounter are the baud rate, the number of data bits per word, parity, and the number of stop bits.

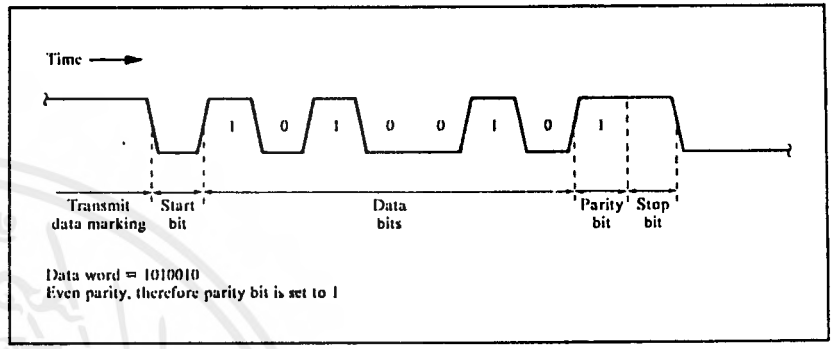


FIGURE 10-8. A typical serial data word with seven data bits, one stop bit, and even parity

Baud Rate

"Baud rate" is the speed at which data is transmitted. This parameter, also referred to as bits per second, bit rate, or data rate, typically ranges from 300 to 9600 baud.

Number of Data Bits

The "number of data bits" refers to the number of bits that constitutes one data word. In serial communications, each data word is transmitted as a sequence of bits. This is shown in Figure 10-8. For example, if you specified a data-word length of 7, then an individual word sent on the communications line would consist of seven data bits, one right after another. This word may be immediately preceded or followed by other bits (parity or start and stop bits) that are used for controlling the communication. The number of stop bits is variable; there is always just one start bit.

Parity

Parity is a method for detecting errors in data communications. The parity bit is added at the end of a data word. The value of this bit is a function of the rest of the data word. There are several ways that the

parity bit can be calculated. "Even parity" means that the parity bit is set so that the sum of all the bits in the data word (including the parity bit) is even. For example, suppose a seven-bit data word were specified with even parity. If the data word were

1010010

then the parity bit would be set to 1, since this would make the number of 1's in the complete data word even. The final data word, with parity, would therefore be

10100101

If the original data word were

0001010

then the parity bit would be a 0, making the sum of all the bits in the complete data word even. The complete data word would now be

00010100

"Odd parity" is similar; the parity bit is set so that the sum of all the data bits in a word, including the parity bit, is an odd number. "Mark parity" means that the parity bit is always a 1; "space parity" means that the parity bit is always a 0. "No parity" means that no parity bit is added to the end of a data word.

Stop Bits

In asynchronous serial communications, either one, one and a half, or two stop bits are added to the end of a data word. These bits tell the receiver where the end of each data word is. Figure 10-8 shows a seven-bit serial data word with even parity and one stop bit.

Full-Duplex and Half-Duplex Serial Communications

An additional characteristic of a communications link is whether or not the connection is a full- or half-duplex line. Often this parameter is set within the communications program. In some cases, you may have to set switches on any modems in the link.

Full- and half-duplex refer to the number of channels in a serial link. "Full" means there are two, and "half" means there is one. With half-duplex, only one device in a link can talk at a time. With full-duplex communication, information can flow in both directions at once. Most serial communications with the PC will use the full-duplex scheme.

ACCESSING ON-LINE INFORMATION WITH THE PC

Now consider how to hook up the PC with an on-line database service using the COMM program on the DOS disk. In this case, the on-line subscription service called The Source will be accessed.

The Parts

The parts you will need to access on-line information are

- A PC with at least one disk drive and an ACA
- A modem (full-duplex; acoustic or direct-coupled)
- An RS-232 cable between PC and modem
- A telephone (if a direct-coupled modem is used, then the phone must be a modular type)
- A Source account
- A copy of the DOS disk.

The Process

The first step is to attach the cables from the PC to the modem, and, if a direct-connect modem is used, from the modem to the telephone. One end of the RS-232 cable plugs into the ACA's 25-pin D connector on the back of the PC; the other end plugs into the D connector on the modem. If the modem is a direct-connect type, then it may be plugged into the phone line now.

The Serial/Parallel Adapter will add or remove start, stop, and parity bits. It also contains a programmable baud-rate generator that allows operation from 50 baud to 9600 baud. Data bytes of five, six, seven, and eight characters are supported by this adapter, along with 1, 1.5, or 2 stop bits. All in all, it is a very versatile serial interface adapter.

Like the parallel ports, the AT supports more than one serial communications port. Two, to be exact. As before, these channels exist as two separate addresses on the I/O bus. The Serial/Parallel Adapter may access either serial port by setting a jumper plug. The serial port jumper plug, J1, is located just above J2 in the lower left corner of the board. In the lower position it selects Serial Port 1 as its communications channel. Incidentally, Serial Port 1 is the primary serial channel, and the default mode of the AT. Reversing the jumper sets the serial interface to Serial Port 2.

SERIAL INTERFACE CONNECTOR

In most applications, the setup and configuration of the Serial/Parallel Adapter serial port is hidden in your program's software. Whatever parameters are needed will be established long before you ever see anything on the screen. You are required, however, to provide the right mechanical interface between the IBM AT and the peripheral device.

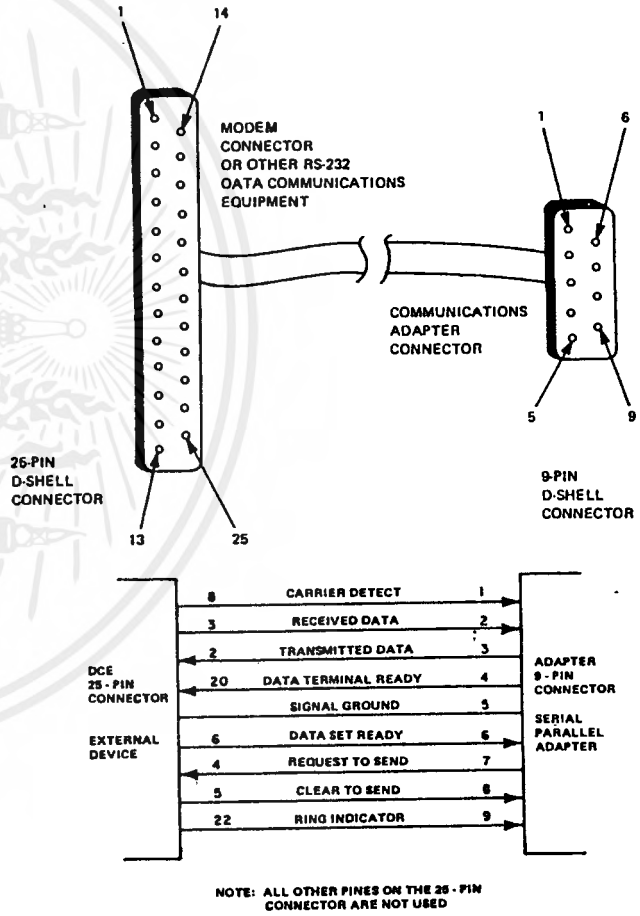
Serial interface to the IBM AT is provided through a 9-pin, D connector that is classified as an RS-232C port. Basically, the RS-232C standard is a defined way of mechanically establishing a serial interface between two pieces of equipment. Both the mechanical and electrical requirements, including connector style, size, and wiring order, are specified.

The RS-232C standard was originally derived from a CCITT telecommunications standard. Since its adoption, however, it has gone through no less than three revisions which you may or may not see labeled: RS-232A, RS-232B, and finally RS-232C. The sad part is that the three are not compatible. In fact, differences are sometimes found even within the same subset. That notwithstanding, it is one of the more stable factors in the highly volatile computer industry, and a wise choice on IBM's part. The CCITT V.24 interface standard, so popular in the European countries, is equivalent to the RS-232C standard, and all references made to the description of RS-232C in the following discussion apply to V.24.

The original RS-232C standard specified a 25-pin, D-shell connector with 25 wires. Of the twenty-five, only five signals are of any importance. The IBM AT supports eight signal lines. They are the same eight signals that have been universally accepted as RS-232C standards.

But, IBM chose to make a mechanical revision to the standard by substituting a 9-pin connector for the venerable 25-pin interface. The connector is mounted on the rear apron of the adapter. This connector is more than a bit unusual in that it establishes yet another standard for RS-232. This change was made to allow more connectors to occupy the same panel space behind the AT. There is one vendor which manufactures a single serial adapter card with four serial adapter ports. Without this modification, all four connectors would have never fit in a single slot. It remains to be seen what impact this change will have on the industry as a whole.

To make the AT's 9-pin connector compatible with standard 25-pin RS-232C equipment, IBM offers a Communications Cable. At one end of the cable is the IBM 9-pin RS-232C connector; at the opposite end is a standard 25-pin RS-232C plug. Basically what the cable does is make the conversion between the two formats by matching pin functions. You can do the same thing with your own hardware using the following pinout listing.



Reprinted from the *IBM PC AT Technical Reference Manual* by permission of IBM.

Computer Interface

The Fluke 45 has a built-in RS-232 interface that can be used to remotely program the meter from a computer, or readings can be sent directly to a serial printer for datalogging applications. An optional GPIB interface is available providing IEEE-488.2 capability. Closed-case calibration is possible with either the RS-232 or GPIB interface.

Flexible Operating Modes

The Fluke 45 has selectable reading rates of: 2.5, 5, or 20 readings per second. Other measurement functions include: limits testing for HI / LO / PASS comparison, Touch Hold, audible continuity and diode test. An optional battery is available that gives the Fluke 45 eight hours of portable operation.

Dual Display Multimeter

This popular meter has dual 5 digit displays allowing 2 parameters to be viewed simultaneously. For example, AC volts can be shown on one display, while the frequency of the same AC signal is shown on the other.

Dual 5-Digit Vacuum Fluorescent Display

Resolution to: 1µV DC, 10µV AC, 1m Ohm,

100nA DC Current, 100 nA AC Current

Standard RS-232 interface...Connects to PC or printer!

0.02% basic DC Voltage accuracy

0.05% basic DC Current accuracy

True RMS voltage and current measurement,including. AC+DC

1 MHz Frequency Counter, 10 mHz min resolution

dB with 21 reference impedances from 2 to 8,000 Ohms

Audio power calculations from 2 to 16 Ohms

Compare (Hi/Lo/Pass) function for quick in-tolerance testing

Min/Max recording and Touch Hold GPIB

interface and internal rechargeable battery pack available

One year warranty. \

Chapter 10

File Handling

All the Visual Basic programs you have created thus far have accessed information typed in by an end user or hard-coded into your forms and modules. Although this may be sufficient for some programs, you will inevitably need to read data from or write data to a disk file.

Unless you use data files, your programs will never be able to *remember* what the user did the last time he ran the program. Data files are used to store information so the information can be retrieved at some time in the future when you run your Visual Basic program again.

Sometimes files are used to store configuration information such as the user's name and their preferred options. Other times, files are used to *remember* aspects such as the last file that was open and the position of all the windows that were on-screen. Other times, data files are used to store *database information* such as data about customers and orders. All these reasons and many more are valid reasons to use data files in a Visual Basic program.

In this chapter, you will learn the types of data files that you may need to access from within a Visual Basic program, and cover how to do so using the Visual Basic programming language.

File Types

Files are used for a wide variety of purposes on computers. Almost every commercial program uses files in one way or another. These programs create and maintain files with information for their own use or to share with other programs.

There are numerous types of files that you can have on your computer. Typically the file's extension (the last three characters of the filename; the ones after the period) identifies what type of file it is, although nothing forces this identification. You could easily rename .BAS files to have an .EXE extension, but that would not make it run when you click it in the File Manager!

Visual Basic uses quite a few file types: .FRM files for your form's source code, .BAS files for your module's source code, .MAK files for your project files, .VBX files for custom controls, and so on.

Although each file extension typically identifies the type, most file types are only understood by the program that uses them. DOS, which provides all the file support for Windows, does not know one type of file from the next—at least not from the filename alone. Typically, DOS only sees a file as being a series of characters that can be interpreted in any number of ways. The *format* of a file is usually determined by the program that uses it.

For example, DOS does not know or care what a .FRM file is; only Visual Basic knows and cares. This is not true for all file types, of course. Most notably, DOS knows that an .EXE file is an executable file that can be run from Windows and possibly from the DOS command line.

Although DOS does not typically care about specific types of files, there are different formats that can be used by and are used by various programs. You will need to read and update each of these formats differently. The remainder of this section will discuss these different kinds of formats:

- ASCII text files
- Foreign file formats
- Import/Export formats
- Initialization (.INI) files
- Simple multiple record databases

ASCII Text Files

ASCII stands for American Standard Code for Information Interchange, and it defines standard characters for interchanging information on a computer. In ASCII, each character is assigned a numeric code that is used to represent and store that character in a file. The printable characters defined by ASCII are #32 (a space) through #126 (a tilde: "~").

A true *ASCII text file* is a file that contains a series of printable characters separated by a carriage return (ASCII value of 13) and a line-feed character (ASCII value of 10). Each of these carriage return/line-feed pairs denotes the end of a line and the beginning of another line. Figure 10.1 illustrates what a four-line ASCII text file looks like conceptually.

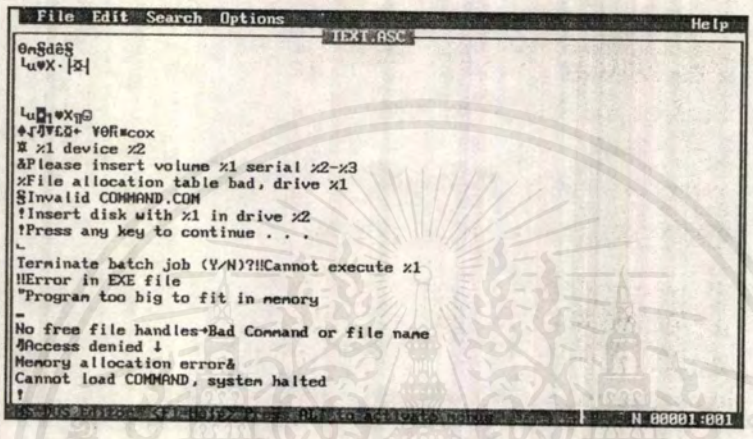


Fig. 10.1
A four-line ASCII text file.

ASCII text files are probably the most common file format used by your PC. Virtually every program that can read and write files can read and write ASCII text files. Both the AUTOEXEC.BAT and CONFIG.SYS on your computer are ASCII text files. The TYPE command in DOS displays an ASCII text file. Windows NotePad allows you to edit and save ASCII text files. Figure 10.2 shows Notepad with CONFIG.SYS loaded.

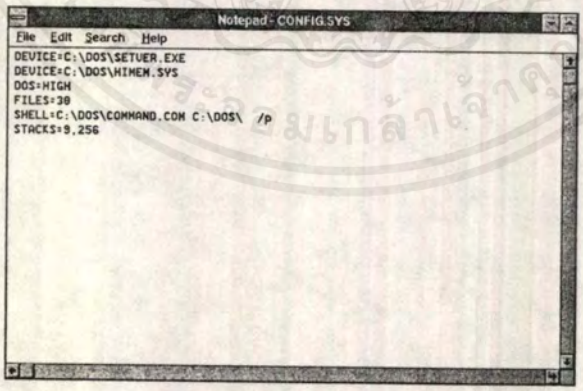


Fig. 10.2
Windows Notepad with CONFIG.SYS loaded.

In addition, Visual Basic project files (those ending in .MAK), form files (those ending in .FRM), and module files (those ending in .BAS) can be stored

as ASCII text files. Windows .INI files are ASCII text. Every Windows word processor on the market can import an ASCII text file into their document format, and all of them can create one.

Visual Basic provides statements and functions that make reading and writing ASCII text files line by line a breeze. Those statements will be covered in detail later in this chapter.

Foreign File Formats

Many programs maintain information in their own special file formats. These formats are simply layouts that were created for use with the program because the program's developers decided it was best for them to store their data in that manner.

There are many examples of different types of data formats. For example, database programs like dBASE and FoxPro store their data in .DBF files. Lotus 1-2-3 originally stored its spreadsheets in .WKS files. Word for Windows stores its documents in .DOC files and its templates in .DOT files. Ami Pro stores its documents in .SAM files. And the list goes on and on.

Each of these file formats was originally designed by the developer(s) of the program. Some of these formats have been published so that anyone who wants to read and update them can do so. Other formats are considered trade secrets and the only way to figure them out (short of hiring a spy) is to spend a lot of time with a program that lets you look at each character in the file—although this is much like trying to learn French by picking up a book written in French and simply trying to read it. Some people can do it, but it will probably take a long time and a lot of perseverance!

Provided you know the format of a specific file, you can write routines in Visual Basic to read, write, and update any foreign file type you want. This writing is done using the *binary file* processing capability of Visual Basic, which is covered later in this chapter.

Import/Export Formats

Although many programs maintain information in files of their own format, there are several common file interchange formats used for importing and exporting into and out of programs. These common formats are understood by many programs and facilitate the transfer of information from one program to the next.

Examples of these format types are the Rich Text Format (.RTF) for transferring word processing files from one word processor to the next. Another example is an older format known as the Data Interchange Format (.DIF) which is used to transfer information from one spreadsheet program to another. There are many other special formats in use today, and there will be many more designed and used in the future.

Often these import/export file formats are simply specially formatted ASCII text files. These types of files can be easily viewed and edited with a text editor such as Notepad. Even so, these files contain additional information needed by programs during the import of the data.

Visual Basic provides direct support for an import/export format often referred to as a *delimited text file* or a *quotes-and-commas* format because each character field in the file is quoted, and all fields are delimited with commas. Listing 10.1 provides a sample of a quotes-and-commas format file.

Listing 10.1 Sample delimited text file.

```
"Mike Schinkel",31,"Atlanta","GA"
"Michelle Brookshire",26,"Clermont","GA"
"Matt Adams",30,"Buford","GA"
"Traci Detchon",36,"Smyrna","GA"
```

Using either Visual Basic's ASCII or binary file processing capability, you have the tools you need to process any type of import/export file that you will find.

Initialization (.INI) Files

Another use of ASCII text files is the special .INI format used by Windows and so many of its programs. These programs create and maintain their own .INI files, and most likely your Visual Basic programs should too.

Windows uses WIN.INI and SYSTEM.INI, which are configuration files that specify Windows current configuration. Figure 10.3 shows SYSTEM.INI and WIN.INI loaded in SysEdit, a program found in the \WINDOWS\SYSTEM directory that allows an experienced Windows user to modify his or her configuration files.

Although you could write routines to read and update .INI files using Visual Basic's ability to process ASCII files, Windows provides specific functions (known as *API functions*) that VB can call to manipulate .INI files.

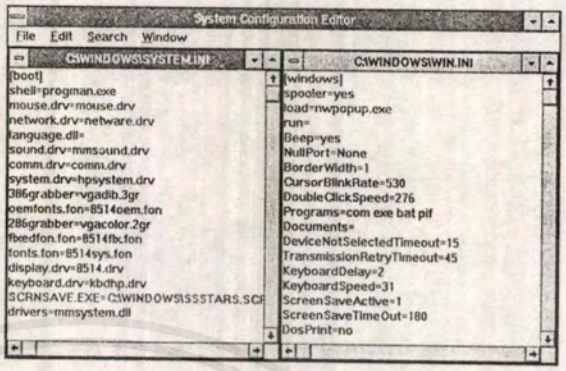


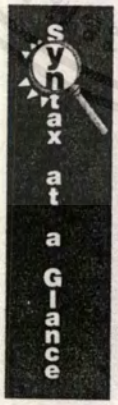
Fig. 10.3
SysEdit showing WIN.INI and SYSTEM.INI.

File Basics

Because data files are external to your programs, you must take certain steps to allow your program to access the information contained in the files. Essentially, your Visual Basic program must establish a line of communication with a file before it can process it. When you finish using a file, you should cut off that line of communication because each open line of communication requires resources to maintain.

Opening a File

In the same way you open the door of a filing cabinet before you can browse through its files, your Visual Basic program must open a file before it can read or update its contents. The statement to do this is `Open`. Using only its most basic options, you can open files for reading (`Input`), writing (`Output`), or appending (`Append`).



Opening Data Files

The basic format for the `Open` statement is

```
Open filename$ [for mode] As [#]filenum
```

You must use `Open` before attempting to read or write from the file. *`filename$`* must be a valid DOS filename. *`mode`* can be any of `Input`, `Output`, or `Append`. *`filenum`* can be any integer from 1 to 255.

Examples

```
Open "C:\AUTOEXEC.BAT" For Input As #1
Open "ERRORLOG.TXT" For Append As 2
```

When you open a file, you must specify a *file number*. This file number is used to refer to the line of communication between your program and the file on disk. Whenever you need to read from, write to, or otherwise manipulate an open file, you will need to use the file number you specified when you opened the file.

Note

Although it is okay in small programs to specify a literal number when you open a file, such as 1, 2, or 7, it is much better to use the `FreeFile` function in larger programs to grab the lowest unused file number for use with your `Open` statement. To use `FreeFile`, store the number it returns into a variable and then use that variable whenever you need to refer to the file in the `Open` statement or any other statement or function that expects a file number as a parameter. For example:

```
Dim FileNum
FileNum = FreeFile()
Open "C:\AUTOEXEC.BAT" For Input As FileNum
```

The mode that you specify when you open the file is important as well. If you open a file for `Input`, then you will only be able to read from the file. If you open it for `Output`, you will only be able to write to the file, and Visual Basic will start writing to the file at the beginning. If you instead specify `Append`, you will only be able to write to the file, but Visual Basic will start writing to the file at the end if the file already exists on disk. You will learn how to both read *and* write to the same open file later in this chapter.

Caution

If you open a file for `Output` and that file already exists, all the data previously in the file will be erased. Be very careful with this and only open a file for `Output` when you know it does not exist, or if you definitely want to start over with it from scratch. Use the following function to determine if a file exists:

```
Function FileExists(FName As String) As Integer
   If Dir$(FName,0) = "" Then
      FileExists = False
   Else
      FileExists = True
   End If
End Function
```

Handling File Errors

When you open a file, there is unfortunately a good chance that something could go wrong—for example, the file could be missing. Because of this fact, you must be sure to provide error handling for any routines that open files.

The easiest way for you to handle an error on the `Open` statement is to use `On Error Resume Next` and then check the `Err` function afterwards. The `Err` function will return a nonzero number identifying the error if an error did occur on the `Open` statement. The values returned by the `Err` function are listed under "Trapable Errors" in Visual Basic's Help system.

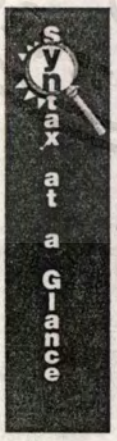
Listing 10.2 illustrates how to trap and identify file errors; you yourself will have to determine what to do in case of an error, as the error handling required for each is likely to be different.

Listing 10.2 Handling file errors.

```
Dim FileNum As Integer
FileNum = FreeFile
On Error Resume Next
Open "C:\AUTOEXEC.BAT" For Input As FileNum
If Err <> 0 Then
   MsgBox ("File Error: " + Str$(Err))
End If
```

Reading Data Files

After you have opened a file, the easiest way to read from it is to read the entire file. You can do this using a combination of the two functions `Input$` and `Lof`. The `Input$` function reads data from a file and returns it as a string.



Reading Data from Files

Use the `Input$` function with a file opened using the `Input` mode of the `Open` statement. The basic format for the `Input$` function is

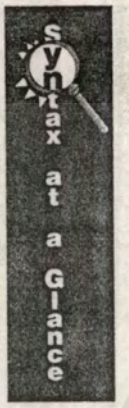
```
data$ = Input$(bytes, [#]filenum)
```

Where *`bytes`* is the number of bytes to read from the file into `data$`, and *`filenum`* specifies which file to read. This *`filenum`* should be the same number used on the `As` clause of the `Open` statement.

Example

```
Open "C:\CONFIG.SYS" For Input As #1
text = Input$(1024, #1)
```

The Lof function returns the total number of characters contained in an open file (that is, its size in bytes).



Obtaining the Length of a Data File

The Lof function returns the number of bytes in a file (its length). The basic format for the Lof function is

```
length = Lof(filenum)
```

Where *filenum* specifies which file to read and should be the same number used on the As clause of the Open statement.

Example

```
FileSize = Lof(1)
```

You can use the Input$ function with the Lof function to retrieve an entire file into a string variable. This technique is one of the simplest ways of reading a file and care should be taken not to read a very large file in this manner. Listing 10.3 shows how to read an entire file into a variable using Input$ and Lof (note that it ignores error handling for the sake of brevity).

Listing 10.3 Reading files with *Input$* and *Lof*.

```
Dim FileNum As Integer
Dim Text As String
FileNum = FreeFile
Open "C:\AUTOEXEC.BAT" For Input As FileNum
Text= Input$(Lof(FileNum), FileNum)
```

Caution

Although it is difficult to say exactly how large a file you can read in this manner, a good rule for a maximum may be 32K, or 32,768 bytes. This is half the maximum string size and is likely to work without a problem. You could load files larger than this, but it would be unreliable. Processing the file in smaller segments would be a better solution, as you will see later in this chapter.

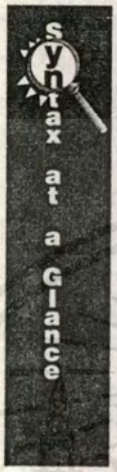
Closing Files

Just as it is a good idea to close your file cabinet drawers after you are through with them (so you don't bang your shin!), you should also close your program's data files as soon as you no longer need to have them open. A simple

`Close` statement is all that is required to release the line of communication between the program and the file.

Tip

It is a good idea for you to close a file as soon as possible to avoid corruption. Because Visual Basic does not always write information to the disk when you ask it to (it waits until an internal output buffer is full), you cannot be sure that what you sent to the file is *actually* written to the file until you close it. If users get in the habit of turning off their computers before exiting your program, or if they have frequent power outages, any open files may become corrupt. Because it is better to be safe than sorry, close the file as soon as you can.



Closing a Data File

The `Close` statement releases the file opened with the `Open` statement and forces all data written to that file to disk. The basic format for the `Close` statement is

```
Close [[#]filenum]
```

Where *`filenum`* specifies which file to close and should be the same number used on the `As` clause of the `Open` statement. If *`filenum`* is omitted, then all the open files are closed.

Examples

```
Close #1
Close 2
```

Loading a File

In the interest of providing a complete example, Listing 10.4 contains the function `LoadFile` which, when passed a filename and a numeric variable for an error code, will open a file, read it completely into a string variable, close the file, and then return the file contents. The sample `Sub Main` in Listing 10.4 illustrates how to call the `LoadFile` function:

Listing 10.4 Loading a file with *LoadFile*.

```
Sub Main
   Dim iErr As Integer
   Dim Text As String
   Text= LoadFile("C:\AUTOEXEC.BAT",iErr)
   If iErr <> 0 Then
      MsgBox("Error Reading C:\AUTOEXEC.BAT")
   End If
End Sub

Function LoadFile(FName As String, iErr As Integer)
   Dim FileNum As Integer
   FileNum = FreeFile()
   On Error Resume Next
   Open FName For Input As #FileNum
   IErr = Err
   If IErr <> 0 Then
      LoadFile = ""
   Else
      LoadFile = Input$(LOF(FileNum), #FileNum)
      Close (FileNum)
   End If
End Function
```

Types of File Access

There are three modes for Visual Basic programmers to access disk files:

- Sequential-access
- Random-access
- Binary-access

As discussed earlier, each of these three methods of accessing files has its own particular purpose.

Sequential Files

A *sequential file* is one written and read from the beginning to the end, in order, much like how tape is played in a tape player. The typical tape player cannot skip to the next song on the tape; instead, it must play or fast-forward through the entire song.

So, a sequential file is one that you can only read in the order that it was written. The reason for this is there is no predetermined structure, and there are no clues for what data is located where.

Reading ASCII Text Files. A perfect example of a sequential file is an ASCII text file. A text editor such as Notepad reads an ASCII text file line by line into memory, and then later writes the entire file back out to disk, typically overwriting the previous copy of the file.

In your Visual Basic programs, you may want to read ASCII text files so you can modify them. For example, you may want to update your AUTOEXEC.BAT `PATH` statement. Or, you may want to write a utility to extract information like the names of subroutines and functions from your Visual Basic .FRM and .BAS files. Or, you can choose from a multitude of other tasks you can accomplish.

To read the data in an ASCII text file, you must first open the file for input and then you can use the `Line Input#` statement.

As mentioned earlier, you can use the `Line Input#` statement to read a line in an ASCII text file, but ASCII text files often contain more than one line. Obviously, you may need to read more than one line so what do you do? Do you use two `Line Input#` statements to read two lines, three statements to read three lines, and so on? Of course not! You read lines inside a loop such as a `While...Wend` loop.

Reading a Line from a File

The basic format for the `Line Input#` statement is

```
Line Input #filenum, line$
```

The `Line Input#` statement reads a line from the file identified by *`filenum`* into the *`line`*`$` string variable.

Examples

```
Dim TextLine As String
Line Input #1, TextLine
```

But that creates another problem. How do you know when you've reached the last line in the file? Easy. Visual Basic provides an `Eof` (end-of-file) function that tells you when you have read the last line in the file (but not before).

As an example, assume you want to write a program that would display all the `Sub` and `Function` declarations in an .FRM file (assuming you save your .FRM's in ASCII).

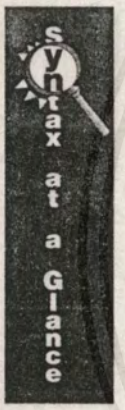
To write this program, you would need a function that would open the .FRM file, read the file line by line to determine which lines had a `Sub` or `Function` declaration in them, and then return only those lines separated with carriage return-line feeds. `LoadSubDecl`, shown in Listing 10.5, performs exactly this task. The CR/LFs make it possible for you to display the string returned by `LoadSubDecl` using a multiline text box.

Listing 10.5 Loading subroutine and function declarations from an .FRM file.

```
Function LoadSubDecl (FName As String, ErrNum As Integer)
   Dim FileNum As Integer
   Dim TextLine, TempLine As String
   Dim OutLine As String
   Dim CRLF As String
   CRLF = Chr(13) + Chr(10)
   FileNum = FreeFile
   On Error Resume Next
   Open FName For Input As FileNum
   ErrNum = Err
   If ErrNum = 0 Then
      OutLine = ""
      Do Until EOF(FileNum)
```

```
            Line Input #FileNum, TextLine
            TempLine = Left(UCase$(TextLine), 4)
            If TempLine = "SUB " Then
               OutLine = OutLine + TextLine + CRLF
            ElseIf TempLine = "FUNC" Then
               OutLine = OutLine + TextLine + CRLF
            End If
         Loop
         Close FileNum
      End If
      LoadSubDecl = OutLine
   End Function
```

Writing to ASCII Text Files. Now that you have read ASCII text files in, you may want to learn how to write them out as well. You write ASCII text files out by using the `Print#` statement.



Writing a Line to a File

The basic format for the `Print#` statement is

```
Print #filenum, line$
```

The `Print#` statement as shown writes the information contained in the *`line$`* string variable to the file identified by *`filenum`*.

Examples

```
Dim TextLine As String
TextLine= "Hello World!"
Print #1, TextLine
```

When you use the `Print#` statement, it writes your string to the file followed by a CR/LF.

Note

If you want to write to a file but you don't want `Print#` to write the CR/LF characters that define an ASCII file, or if you want to write several values before `Print#` outputs another CR/LF, you can force the `Print#` statement to omit the CR/LF by placing a semicolon after the string. For example:

```
Print #1, "This ";
Print #1, "creates ";
Print #1, "one ";
Print #1, "line."       ' Note no semi-colon here

Print #1, "This " ; "also " ; "creates " ; "one " ; "line."
```

Suppose you want to write a routine that updates the PATH setting in AUTOEXEC.BAT. Writing this routine can be useful if your application consists of multiple .EXE files and you want it to be able to find the .EXEs no matter what the current directory.

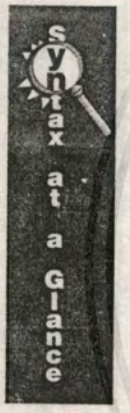
To write such a routine, you would need a function that would open up both AUTOEXEC.BAT for input and another temporary file for output. The function FixAutoExec, shown in Listing 10.6, creates a file called AUTOEXEC.NEW with the updated file information.

Listing 10.6 Fixing the *PATH* statement in AUTOEXEC.BAT.

```
Sub Main()
   Dim Fixed As Integer
   Dim iErr As Integer
   Fixed= FixAutoExec("C","C:\MYAPP",iErr)
   If Fixed Then
      MsgBox("AUTOEXEC.NEW Created!")
   Else
      MsgBox("Error creating AUTOEXEC.NEW")
   End If
End Sub

Function FixAutoExec (Drive As String,
                      ➥NewDir As String,
                      ➥ErrNum As Integer) As Integer
   Dim InpFile As Integer
   Dim OutFile As Integer
   Dim TxtLn As String
   Dim Fixed As Integer
   Dim Temp As String
   Dim FileCnt As Integer
   Fixed = False
   On Error Resume Next
   InpFile = FreeFile
   Open Drive + ":\AUTOEXEC.BAT"
   ➥For Input As InpFile
   ErrNum = Err
   If ErrNum = 0 Then
      OutFile = FreeFile
      Open Drive + ":\AUTOEXEC.NEW"
      ➥For Output As OutFile
      ErrNum = Err
      If ErrNum = 0 Then
         Do Until EOF(InpFile)
            Line Input #InpFile, TxtLn
            Temp = Left(UCase$(TxtLn), 5)
            If Temp = "PATH=" Then
               TxtLn = "PATH=" + NewDir + ";" + Mid$(TxtLn,6)
            End If
            Print #OutFile, TxtLn
```

```
            Loop
            Close OutFile
         End If
         Close InpFile
         Fixed = True
      End If
      FixAutoExec = Fixed
   End Function
```

Of course, the function `FixAutoExec` does not actually update AUTOEXEC.BAT, it simply creates a new file called AUTOEXEC.NEW with the updated `PATH` statement. If you want `FixAutoExec` to actually update AUTOEXEC.BAT, you will need two statements that have not been covered yet; `FileCopy` and `Kill`.



Copy a File

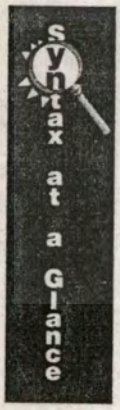
The basic format for the `FileCopy` statement is

```
FileCopy SourceFile$, DestinationFile$
```

The `FileCopy` statement copies the file you specify for *`SourceFile$`* to the filename you specify for *`DestinationFile$`*.

Example

```
FileCopy "AUTOEXEC.NEW", "AUTOEXEC.BAT"
```



Delete a File

The basic format for the `Kill` statement is

```
Kill File$
```

The `Kill` statement deletes the file you specify from the disk.

Example

```
Kill "AUTOEXEC.NEW"
```

Now that the `FileCopy` and `Kill` statements have been discussed, you can add the source code in Listing 10.7 to the `FixAutoExec` function in place of the

line that contained "`Fixed = True.`" This line was located near the bottom of the `FixAutoExec` function. Be sure that you indent your code neatly.

Listing 10.7 Updating AUTOEXEC.BAT with AUTOEXEC.NEW.

```
FileCopy "C:\AUTOEXEC.BAT", "C:\AUTOEXEC.OLD"
If Err = 0 Then
   FileCopy "C:\AUTOEXEC.NEW", "C:\AUTOEXEC.BAT"
End If
If Err = 0 Then
   Kill "C:\AUTOEXEC.NEW"
   Fixed = True
End If
```

Reading Delimited Text Files. As stated earlier, a delimited text file is an ASCII file, but it is an ASCII file for which Visual Basic provides some special processing capabilities. Often you may use these file formats to import from or export to database or spreadsheet programs.

To read delimited text files, you can use the `Input#` statement whose format is shown in the Syntax at a Glance boxes:

Reading a Line from a Delimited Text File

The basic format for the `Input#` statement is

```
Input #filenum, var1 [, var2] [..., varN]
```

The `Input#` statement reads a line from the file identified by *`filenum`* and parses the information in the file into the variables *`var1`* through *`varN`* that you specify.

Example

```
Dim FullName As String * 25
Dim Age As Integer
Dim City As String * 15
Dim State As String * 2
Input #1, FullName, Age, City, State
```

When you use the `Input#` statement, you must be sure that the file you are reading and the variables you are using coincide in number and in type. If you do not specify enough variables to hold all the data to be read in, or if you specify too few, Visual Basic will get confused and you will end up getting garbage in your variables.

Writing Delimited Text Files. The converse of the `Input#` statement is the `Write#` statement. The `Write#` statement will create a file that can be read back

into numerous other programs, or into Visual Basic using the `Input#` statement.

Write a Line to a Delimited Text File

The basic format for the `Write#` statement is

```
Write #filenum, value1 [, value2] [..., valueN]
```

The `Write#` statement writes a delimited text file line using the values *value1* through *valueN* to a file identified by *filenum*.

Example

```
Write #1, "John Deere", 95, "Des Moines","Iowa"
```

Random-Access Files

As compared to a sequential file, a *random-access file* is one that can be read and written in any order. You may want to think of a random-access file as being like a compact disc instead of a cassette tape. Just like a compact disc where you can select and play specific songs (or tracks) in any order, you can read, write, and even update portions of a random-access file in any order.

Random-access files were originally provided to allow the BASIC programmers an easy way to create *databases*. What is a database? A database is simply a collection of related information organized in a structured way. In earlier days, BASIC programmers created relatively sophisticated database applications using random-access files as the tables in a relational database. Today, however, you would probably not want to use random-access files to implement a complex database program. Visual Basic provides much greater database capabilities via the powerful Access Database Engine and through ODBC. (Both of these topics are beyond the scope of this book. For more information, refer to Que's *Using Visual Basic 3*.)

Random-access files are useful if you need to create a relatively simple database or if you need to read and update data that was created or is maintained by an older BASIC application.

Random-Access vs. Sequential Files. Assuming you have decided that using a random-access file is the best solution for your needs, be aware that

random-access files provide both benefits and drawbacks when compared to sequential files. Table 10.1 discusses the pros and cons of random-access files.

Table 10.1 Pros and Cons of Random-Access Files

PROS	CONS
Information can be accessed much quicker using random-access files because your program can request to read a specific record only instead of being required to read the entire file.	You will not be able to read random-access files by other programs unless they have been specifically written to recognize your record structures.
You can update the records of a random-access file without having to create a new file.	Records must be fixed-length and thus take up more space than the same information stored in a sequential file.
You can read and write records in a random-access file in any order.	Every record in a random-access file must be exactly the same structure.

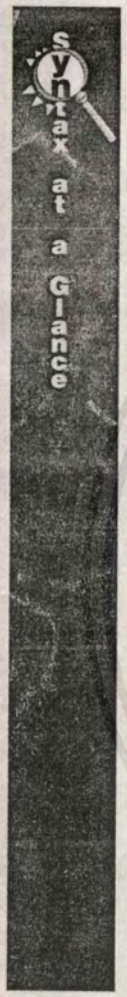
Record Variables: User-Defined Types. Processing random-access files means processing records. As such, you must define the records in your file, which is done by using the `Type...End Type` block statement. After you have defined your structure with `Type...End Type`, you can create variables using that definition and then store the values contained in these variables directly into the records of your random-access file.

You must be aware of the size of the structure (in bytes) that you create using the `Type...End Type` statement. This awareness will be important when you open the random file so you can determine how much disk space will be required by your random-access file.

The size in bytes of a record is determined by the sum of the sizes in bytes for each member of your structure. String variables are easy: each character in a string takes one byte, so a string defined as `String * 25` occupies 25 bytes in your structure. The number of bytes required by each of the numeric types is a little more difficult to determine. You may want to refer to the information in Chapter 4 where data types are discussed more fully.

As an example, assume you want to add a file of user information to your Visual Basic program. This use would be well-suited for a random-access file

because you would likely have no need to read this information with another program; on the contrary, you would probably want this information to be somewhat secure.



Define Record Structures for Random-Access Files

This format for the `Type...End Type` statement is

```
Type rec_struct
    field1 AS type1
    field2 AS type2
    ...
    fieldN AS typeN
End Type
...
Dim var As rec_struct
```

Use the `Type...End Type` statement to define the structure of your records for random-access files. `rec_struct` becomes the name of the new data type, *field1* through *fieldN* are the names of the fields in the structure, and *type1* through *typeN* are their corresponding data types.

The structure name `rec_struct` then becomes a valid data type and can be used to declare *var* variables on the `Dim` statement.

Example

```
Type GolfCart
   Model String * 10
   Color String * 15
   Seats Integer
   Horses Single
End Type

Dim MyCart As GolfCart
MyCart.Model = "Bogie XL"
MyCart.Color = "Fairway Green"
MyCart.Seats = 2
MyCart.Horses = 10
```

Your user information file would likely include the user's full name, user id, user level, password, and birthday so that you can wish him a happy birthday on the appropriate day. In addition, you may want to keep some statistics such as the date of his first use of the system, and the total time that he has spent using the system. The latter necessitates that you maintain the time and date which he started running the program last. Finally, you may want to maintain a field to indicate which records are deleted. Listing 10.8

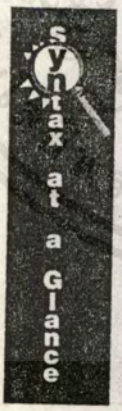
provides a structure definition for UserStruct that provides this type of information and a declaration for the variable User which is of type UserStruct. Note that you will need to declare your structures in one of your modules (.BAS files), not in your forms (.FRM files).

Listing 10.8 Structure of user info record.

```
Type UserStruct
   UserId As String * 10      'User's ID
   FullName As String * 25    'User's Full Name
   Password As String * 10    'User's Password
   Birthday As String * 10    'User's BirthDay
   UserLevel As String * 1    'User Level 0..3
   FirstUse As String * 10    'Date of first use
   StartTime As Long          'Seconds this run
   StartDate As String * 10   'Date of this run
   TotalTime As Double        'Cum time since start
   UseCount As Integer        'Times in the system
   Deleted As String * 1      'Delete Flag: *
End Type
Dim User As UserStruct
```

The size of the User variable of type UserStruct is 91 bytes: 10 + 25 + 10 + 10 + 1 + 10 + 4 + 10 + 8 + 2 + 1.

Opening Random-Access Files. To open a file for random-access, you should specify For Random on the Open statement instead of Input, Output, Append. In addition, you should specify the record size in bytes using the Len= clause.



Opening Data Files for Random Access.
This format for the Open statement is

```
Open filename$ For Random As [#]filenum Len=reclen
```

You must use Open before attempting to read or write from the file. *filename$* must be a valid DOS filename. *filenum* can be any integer from 1 to 255. *reclen* should be the size of the file's records.

Example

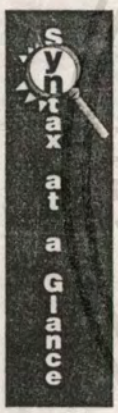
```
Open "USERS.DAT" For Random As #1 Len=55
```

Note

When you open a random-access file, you specify the record length using the Len= clause. Although you can hard-code the length, it is more maintainable to let Visual Basic calculate the length itself. By passing the record variable to the Len function (discussed in Chapter 5), you can determine the size in bytes of the record. If you change the record structure as your program evolves, you will not need to modify the Len= clause of the Open statement. For example:

```
Dim User As UserStruct
Open "USERS.DAT" For Random As #1 Len=Len(User)
```

Reading Records from Random-Access Files. When you have a file open for random access, you can read any record simply by using the Get statement.



Reading Records from Random-Access Files

This format for the Get statement is

```
Get #filenum, [recnum], recvar
```

The Get statement reads the record identified by *recnum* into the *recvar* variable from the file identified by *filenum*. *recnum* is optional and if omitted, the Get statement will read the next record in the file.

Examples

```
Get #1, 5, User      'Get 5th user record
Get #1, , User       'Get next user record
```

You can omit the record number in the Get statement if you want. Listing 10.9 provides a simple example of reading records from a file sequentially using the Get statement.

Listing 10.9 Example of using the *Get#* statement.

```
'From MODULE1.BAS
Type NameStruct
   First As String * 10
   MI As String * 1
   Last As String * 15
End Type

'From FORM1.FRM
Dim RecCount As Long
Dim i As Integer
```

(continues)

Listing 10.9 Continued

```
Open "names.dat" For Random As #1 Len = 26
RecCount = LOF(1) / 50
ReDim Names(RecCount) As NameStruct
For i = 1 To RecCount
   Get #1, , Names(i)  'No need for recnum
Next I
Close #1
```

Although you have seen how to read records sequentially using the `Get` statement, you can read them in any order. You only need be careful that you do not read a record that does not yet exist in the file; if you attempt to do so, Visual Basic will trigger an error.

Writing Records to Random-Access Files. Conversely to the `Get` statement, the `Put` statement is used to write records to a random-access file.

Writing Records to a Random-Access File

This format for the `Put` statement is

```
Put #filenum, [recnum], recvar
```

The `Put` statement writes the record contained in the *recvar* variable to the record identified by *recnum* in the file identified by *filenum*. *recnum* is optional, and if omitted, the `Put` statement will write to the *next* record in the file.

Examples

```
Put #1, 5, User     'Put user var into 5th record
Put #1, , User      'Put user var into next record
```

Just like the `Get` statement, you can omit the record number when you call the `Put` statement.

As an example, assume that you had a form with three text controls for entering first name, middle initial, and last name. Listing 10.10 shows how to write that information to the *current* record (stored in the global variable `RecNo`) of a random-access file.

Listing 10.10 Example of using the *Get#* statement.

```
'From MODULE1.BAS
Type NameStruct
   First As String * 10
   MI As String * 1
   Last As String * 15
```

```
End Type
Global RecNo As Long

'From FORM1.FRM
Dim FullName As NameStruct
Open "names.dat" For Random As #1 Len = 26
FullName.First = txtFirst.Text
FullName.MI = txtMI.Text
FullName.Last = txtLast.Text
Put #1, RecNo, FullName
Close #1
```

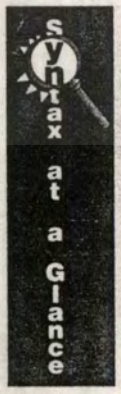
Binary Files

As compared to both sequential and random-access files which access files essentially at the record level, *binary files* are accessed byte by byte. As such, binary files have no limitations whatsoever; you can process a binary file any way you like (assuming that you designed the format of the file). On the other hand, nothing is done for you automatically with binary files; you must determine exactly where to place data into the file and then specify that information to Visual Basic.

Note

Actually, there is no such thing as a *binary* file; any file can be opened for binary access. You will be able to read, write, and update information for every type of file ever created on a personal computer using binary mode, assuming that you know the file's format!

Opening Files for Binary Access. To open a file for binary access, you should specify For `Binary` on the `Open` statement. Because binary files are accessed byte by byte, you do not need to specify a `Len=` clause as you did for random-access files.



Opening Data Files for Binary Access

This format for the Open statement is

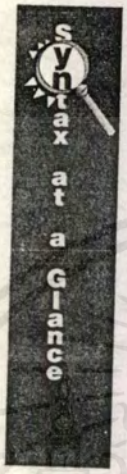
```
Open filename$ For Binary As [#]filenum
```

You must use Open before attempting to read from or write to the file. *filename*$ must be a valid DOS filename. *filenum* can be any integer from 1 to 255.

Example

```
Open "CUSTOMER.DBF" For Binary As #1
```

Reading from Binary Access Files. When you have opened a file for binary access, you can use the `Get` statement to read information from the file into variables.



Reading Records from Binary Files

This format for the `Get` statement is

```
Get #filenum, [bytepos], var
```

The `Get` statement reads data from the file identified by *`filenum`*, starting at the position *`bytepos`*, into the *`var`* variable. The size of *`var`* (in bytes) determines how many bytes will be read from the file. *`bytepos`* is optional and if omitted, the `Get` statement will read from the file starting at the next byte in the file.

Examples

```
Get #1, 33, FldInfo     'Get to FldInfo from byte 33
Get #1, , FldInfo       'Get to FldInfo from current
```

Writing to Binary-Access Files. Writing to a binary file is as easy as reading; you simply use the `Put` statement to write information from a variable into the file.

Writing Records to Binary Files

This format for the `Put` statement is

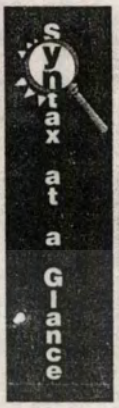
```
Put #filenum, [bytepos], var
```

The `Put` statement writes the contents of *`var`* to the file identified by *`filenum`*, starting at the position *`bytepos`*. The size of *`var`* (in bytes) determines how many bytes will be written to the file. *`bytepos`* is optional and if omitted, the `Put` statement will write to the file starting at the next byte in the file.

Examples

```
Put #1, 50, FldInfo     'Put to file starting at byte
                        ➥50
Put #1, , FldInfo       'Put to file immediately
```

Updating the Current Position in a Binary File. If you use binary-access files, you may need to move to a specific position in a file so you can begin accessing a portion of the file in a sequential manner. You can use the `Seek` statement to update the current file position.



Updating the Current Position in a Binary File

The format for the Seek statement is

```
Seek #filenum, bytepos
```

The Seek statement updates the current file pointer position (for *filenum*) so that the next byte, read or written, is the byte you specified with *bytepos*.

Examples

```
Seek #1, 50    'Move current position to byte 50
```

When you use the Get or Put statements and specify a byte position to start at, it is much like calling the Seek statement and then calling the Get or Put statements without a starting byte position. In other words, the source code in the following two lines of code:

```
Seek #1, 33
Get #1, , FldInfo
```

is identical in effect to this line of code:

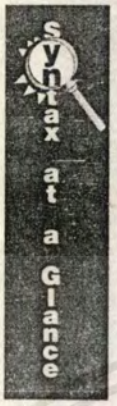
```
Get #1, 33, FldInfo
```

You may use the Seek statement just prior to entering a loop. Listing 10.11 shows how to use the Seek statement in this manner for bypassing the file header on a dBASE .DBF file prior to scanning for the first deleted record in the file (one whose record data begins with an asterisk).

Listing 10.11 Using *Seek* with binary-access files.

```
Dim RecData As String
RecData= String$(63)    'Set buffer to 63 bytes
Open "customer.dbf" For Binary As #1
Seek #1, 161            'Move to byte 161
Do
   Get #1, , RecData
Loop Until Eof(1) Or Left$(RecData,1)="*"
Close #1
```

Determining the Current Position in a Binary File. You also can determine the byte position that was last processed and the next byte to process using the two functions Loc and Seek, respectively. By using either of these you can save and later restore your position in a file.



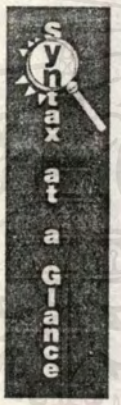
Retrieve the Last Byte Position Processed in a Binary File
The format for the Loc function is

```
bytepos = Loc(filenum)
```

The Loc function returns (into *bytepos*) the last byte position, read or written, in the file identified by *filenum*.

Examples

```
Put #2, 161, "*"    'Write asterisk at pos 161
MsgBox(Loc(2))      'Displays 161
```



Retrieve the Next Byte Position to Process in a Binary File
The format for the Seek function is

```
bytepos = Seek(filenum)
```

The Seek function returns (into *bytepos*) the next byte position, read or written, in the file identified by *filenum*.

Examples

```
Put #2, 161, "*"    'Write 6p6
splays 162
```

You can use Seek (or Loc) to save and later restore your positions in a binary file as shown in Listing 10.12.

Listing 10.12 Saving and Restoring Byte Positions.

```
Dim SavePos As Integer
Open "orders.dbf" For Binary As #3
SavePos = Seek(3)
...                     'Do something here
Seek #3, SavePos        'Return to previous position
```

The Wide Variety of Uses for Binary Files. You have now seen the statements used to access a file on the byte-by-byte level. You can use binary access to allow you to open and read files created by any application, assuming you know the format.

In addition, you can define your own file formats and then write Visual Basic routines to create and then load the information stored in these files.

Binary-access files provide more low-level power than any of the other types of access, and you will likely find yourself using this type of access when the type of information you need to store becomes more and more complex.

Updating .INI Files

Windows and most Windows applications make use of a special type of ASCII file with an extension of .INI (pronounced *Innie*). These .INI files are used to store information about program configuration, and to save information about the state of an application between successive runs of the program.

Windows uses two special files called SYSTEM.INI and WIN.INI in which it stores Windows system-level configuration information and user-level configuration information, respectively. Listing 10.13 shows a portion of SYSTEM.INI so that you can see the format of this file.

Listing 10.13 A portion of SYSTEM.INI.

```
[boot]
386grabber=mach32.3gr
oemfonts.fon=vgaoem.fon
286grabber=vgacolor.2gr
fixedfon.fon=VGAFIX.FON
fonts.fon=VGASYS.FON
display.drv=aw5xga.drv
shell=C:\APPS\DASH\dash.exe
mouse.drv=aw5mouse.drv
network.drv=NETWARE.DRV
language.dll=
comm.drv=comm.drv
keyboard.drv=aw5kbd.drv
system.drv=system.drv
drivers=mmsystem.dll winmm16.dll
SCRNSAVE.EXE=C:\WINDOWS\SSSTARS.SCR
oemfonts=VGAOEM.FON

[keyboard]
subtype=
type=4
keyboard.dll=
oemansi.bin=

[boot.description]
aspect=100,96,96
display.drv=mach32 Driver
displayinf=OEM68800.INF

...much more stuff
```

As you may have noticed, .INI files have various sections that begin with their section name enclosed in square brackets, followed by lines that set identifiers to values using an equal (=) sign.

Although you can process .INI files using the sequential file I/O, the Windows API provides a much better alternative. There are four functions in the Windows API that read and write .INI files; two of them work specifically with WIN.INI and the other two will support any .INI file.

Tip

Storing information about your application in an .INI file is a very good idea because it is a common approach under Windows; as such, many Windows users and support professionals are familiar with them. On the other hand, it is a very *bad* idea to add application-specific items to the WIN.INI file, although many applications do so. Microsoft applications are the worst violators of this rule.

Every time an item is added to WIN.INI, it slows Windows down slightly, and it makes it much more difficult for a user to remove all traces of a program from their computer after they do not need it any longer.

Note

The Windows API (application programmer interface) is a collection of dynamic link libraries (.DLL files) that contain Windows system functions you can call if you provide a declaration for Visual Basic in a .BAS file. These declarations are available for you in the *Win 3.1 API Help* Windows Help file that comes with Visual Basic Professional 3.0.

To create or update a *private* .INI file, one that you name yourself, you call the `WritePrivateProfileString` function. To read the values from a *private* .INI file, you call the `GetPrivateProfileString` function.

To update a WIN.INI file, you call the `WriteProfileString` function and to read the values from WIN.INI, you call the `GetProfileString` function.

The declarations for the four profile string functions are contained in Listing 10.14.

Listing 10.14 Declarations for Windows API functions for processing .INI files.

```
'Write to Private .INI file
Declare Function WritePrivateProfileString
➥Lib "Kernel" (ByVal AppName$, ByVal
➥KeyName$, ByVal KeyValues$, ByVal
➥FileName$) As Integer

'Read from Private .INI file
Declare Function GetPrivateProfileString Lib
➥"Kernel" (ByVal AppName$, ByVal KeyName$,
➥ByVal Defaults$, ByVal ReturnedKeyValues$,
➥ByVal nSize As Integer, ByVal FileName$) As Integer

'Write to WIN.INI file
Declare Function WriteProfileString Lib
➥"Kernel" (ByVal AppName$, KeyName$,
➥KeyValues$) As Integer
```

```
'Read from WIN.INI file
Declare Function GetProfileString Lib
➥"Kernel" (ByVal AppName$, KeyName$, ByVal
➥Default$, ByVal ReturnedString$, ByVal nSize
ÂAs Integer) As Integer
```

Listing 10.15 shows you how to call the two functions `WritePrivateProfileString` and `GetPrivateProfileString`.

Listing 10.15 Calling the Windows API functions for processing .INI files.

```
Dim n As Integer
Dim TestValue As String * 5
n = WritePrivateProfileString("myapp",
➥"test", "ABCDE", "myapp.ini")
n = GetPrivateProfileString("myapp",
➥"test", "", TestValue, 5, "myapp.ini")
MsgBox(TextValue) 'Should display ABCDE
```

From Here...

As you have learned, you can store information to disk files and later read that information back into your programs using Visual Basic's file commands. As far as Visual Basic is concerned, there are three types of files: sequential, random, and binary.

You also learned the following information in this chapter:

- To gain access to a disk file, you use the `Open` statement.
- Whenever you open a file, you should include error-handling code in case an error occurs.
- You can read information from a file using the `Input$` function.
- You can determine the length of an open file using the `Lof` function.
- The `FreeFile` function returns the next unused file number and makes your routines that open files more generic.
- You read sequential files with the `Line Input` or `Input` statements, and you write to them using `Print#` or `Write#`.
- You declare records for random-access files using `Type...End Type`.

Chapter 16

Jeeves and Java Servlets

This chapter covers the newest aspect of Java networking—servlets. Servlets are the server-side equivalents of applets. They are Java programs designed to execute in an environment provided by an HTTP server. Intended as a Java-style replacement for Common Gateway Interface (CGI) programs, servlets are supported as an extension package to JDK 1.1 and use a Java-based HTTP server that implements the newly defined Servlet API. Like applets, servlets can be automatically downloaded to the server, and untrusted servlets run in a secured environment. Unlike applets, servlets have no user interface and do not use the AWT classes. Servlets and applets are compared in table 16.1. In this chapter, you are introduced to the basic features of the Jeeves HTTP server and learn how to construct your own servlets.

TABLE 16.1
Comparison of Servlets and Applets

Servlet	*Applet*
Subclass of GenericServlet	Subclass of Applet
Runs in a server	Runs in a browser
Must be multithreaded or thread safe	Generally single thread per applet
No direct user interface	Uses AWT for user interface
If downloaded to server, controlled access to files and network	If downloaded to browser, no access to files and network access back only to serving host
If local to server, full access to files and network	n/a

Undoubtedly the biggest weakness of CGI (and similar techniques) is a lack of formal security mechanisms. Without a secure runtime system, web masters must closely inspect CGI programs for programming flaws and security loopholes that could be exploited to disrupt operations or obtain sensitive information. Through experience, a list of defensive programming techniques have been compiled for CGI programs to prevent known security problems. (For example, always limit user-entered data to the standard printing character set to prevent entry of special shell command escape characters.) Despite these countermeasures, many shared-user web hosts restrict the use of CGI programs. Dynamic content may often be limited to forms processing; to go beyond this, a safe and secure environment is required for creating dynamic content. A Java-based solution can provide the needed security.

Servlet API Goals

The Servlet API is designed to replace CGI as the standard interface between HTTP servers and HTML-processing programs, although implementation is initially limited mostly to the JavaSoft-developed Jeeves HTTP server. Java programs implementing the Servlet API are called *servlets*—the server-side equivalent of applets. Servlets execute in a special environment provided by a web server that supports the new Servlet API. This environment supplies all the resources needed for servlets to run and enforces the security model to maintain system integrity. This approach is similar to the controlled execution environment provided by web browsers for applets. The Servlet API has the following attributes:

- **Ease of use and understanding.** The Servlet API is similar to CGI in terms of HTTP information available, making the conversion of existing CGI programs to servlets feasible.
- **High performance.** The Servlet API eliminates the overhead of starting an external program to serve each request; it enables servlets to practically perform all server operations, including basic file serving.
- **Platform independence.** Like applets, servlets run on any server platform that provides a Java Virtual Machine supporting the Servlet API.
- **Transport independence.** The Servlet API has base classes that are independent of the transport protocol used to carry web content and subclasses that are specific to HTTP. Specific versions of the Servlet API can be developed for future web transport protocols.
- **Secure and safe.** Untrusted servlets run in a secured environment, as described in the "Security" section later in the chapter, that protects access to the network and other local resources. Trusted servlets have full access to local resources.

Java's Servlet API has a significant advantage over CGI in terms of performance, transport independence, and security. The differences between servlets and CGI are highlighted in table 16.2. The use of servlets for all web server operations makes a very powerful and flexible web platform. For example, server administrators can easily augment the file serving servlet to collect statistics on files requested or track user information, or augment the error response servlet to specialized error messages.

TABLE 16.2
Comparison of Servlet API and CGI

	Servlet	*CGI*
Language	Java	Any programming language
Invocation	Direct method call	Run program
Parameters	ServletRequest methods	Environment variables
Runtime Checks	Java Virtual Machine	Language specific
Security	Java Security Manager	Operating system specific

Servlet API Overview

Briefly, the Java Servlet API is comprised of three packages that define high-level functionality, HTTP-specific support, and dynamic HTML creation support. You can assume these packages are available on any server platform supporting servlets.

The java.servlet package contains classes that define the high-level servlet functionality independent from the details of using HTTP for transport. These classes are as follows:

- Servlet
- GenericServlet
- ServletContext
- ServletInputStream
- ServletOutputStream
- ServletRequest
- ServletResponse

The java.servlet.http package contains subclasses of the generic classes in java.servlet and an HTTP-specific utilities class. You use these classes in building servlets that specifically support HTTP. These classes are as follows:

- HttpServlet
- HttpServletRequest

- HttpServletResponse
- FormServlet
- HttpUtils

The java.servlet.html package contains 13 HTML support classes that provide operations for assembling HTML elements such as tables, imagemaps, and forms. These classes are as follows:

- HtmlApplet
- HtmlChoice
- HtmlContainer
- HtmlDefinitionList
- HtmlElement
- HtmlForm
- HtmlFrameset
- HtmlImageMap
- HtmlList
- HtmlPage
- HtmlRow
- HtmlTable
- HtmlText

Together, these three packages define the Servlet API and form the basis for building servlets. These packages are covered in more detail in "The Servlet API," later in this chapter. Of course, web servers that support servlets are likely to offer vendor-specific classes in addition to these standard ones. The HTTP server included in the Jeeves package has its own set of JavaSoft-specific classes (such as the sun.server.* and sun.security.* packages). These vendor-specific packages provide additional support for implementing servlets beyond the basic API. The JavaSoft classes provide support for additional security mechanisms, regular expression parsing, and cache file handling, among others.

File Suffix to MIME Type Translation

Most servlets know the type of content that they are producing and can correctly set the contentType attribute of the response. However, the FileServlet has no such knowledge and thus needs a means to determine the likely content type from the file name. The convention is to use a consistent set of file name suffixes that can be mapped to MIME types. This mapping is stored in the mime.properties file. A small portion of this table is as follows:

```
*=text/plain
java=text/plain
html=text/html
htm=text/html
gif=image/gif
jpeg=image/jpeg
mpeg=video/mpeg
qt=video/quicktime
shtml=java-internal/parsed-html
```

Each line in the file indicates a given file name suffix representing a particular MIME type. The first line (*=text/plain) is the default rule that is applied when all other rules fail to match. The order of entries in this file is not important; the * character indicates the default content type if the file matches none of the suffixes. Note that the last line of the sample file defines a mapping of file names with suffixes of .shtml to the private MIME type java-internal/parsed-html. The File Servlet invokes the server-side include servlet to process any file with a content type of "java-internal/parsed-html" (which is converted to MIME-type text/plain when the file is sent). You only need to add entries to this table to reflect new applications or file types.

The Servlet API

The Servlet API is defined by the classes in the java.servlet, java.servlet.http, and java.servlet.html packages. Figure 16.4 shows the packages, classes, and their relationships. Together, these packages specify the following:

- The generic behavior of servlets independent from the web transport protocol
- Additional facilities available to HTTP-specific servlets
- Support for programmatic creation of HTML pages as a hierarchical structure of HTML elements

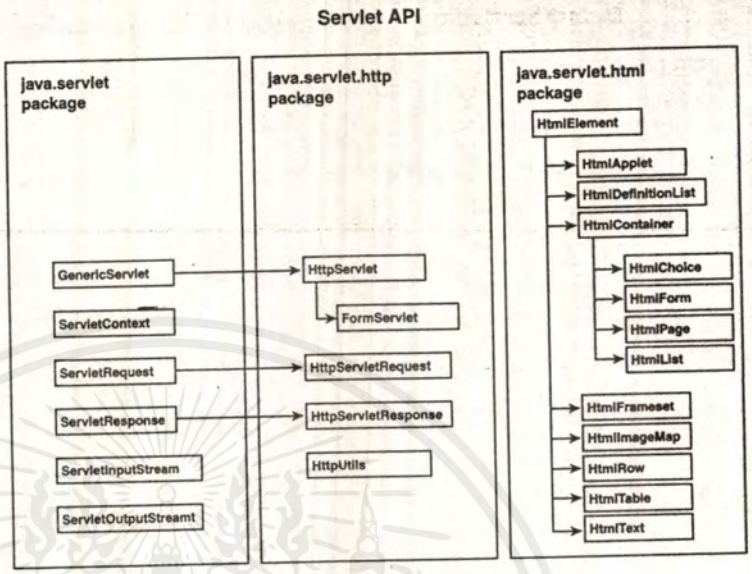


FIGURE 16.4
The servlet API classes.

The following sections cover the methods that comprise the Servlet API. Where appropriate, the equivalent CGI variable is indicated. In addition to the Servlet API's availability on any server platform that supports servlets, developers may have access to vendor-specific classes and servlets that are proprietary to a given server and not part of the Servlet API. Any such extensions would be covered by server-specific documentation.

The java.servlet Package

The java.servlet package defines the operation of servlets from a protocol-independent perspective. At this level, the protocol used to access the servlets could be HTTP, a network file system protocol, or other transport protocol. Servlets that are designed to operate as a server-side include use the classes in this package. The four interfaces and three implementation classes in this package are described in more detail in the following sections.

Servlet Interface Class

Servlet is the highest level, generic interface class. It defines only four basic methods to accomplish the following: initialize the servlet, destroy the servlet, service a request, and return an information string:

- **init(ServletStub).** Automatically called by the system to initialize a servlet. Subclass this method if your servlet needs to perform any initialization after it is loaded. Access the initialization parameters specified for your servlet in the servlet.properties file by calling getInitParameter() or getInitParameters().
- **destroy().** Called by the system to free allocated resources when the servlet class is unloaded.
- **service(ServletRequest, ServletResponse).** Called to process the incoming request and create the outgoing response. You must code your service() method in a thread-safe manner because concurrent requests for the same servlet result in multiple threads executing your service() method. The easiest way to be thread-safe is to not modify any object instance variables and only use parameters of the service() call. Alternatively, you can add the "synchronized" modifier to your declaration of service() to restrict access to one thread at a time.
- **getServletInfo().** Called to get an information string that describes the servlet.

A subclass of Servlet uses a ServletContext object, described in the next section, to obtain information about the server's environment and to call various server utility routines.

ServletContext Interface Class

The ServletContext interface defines methods that the servlet can use to interact with its operating environment. The relationship between a servlet and the ServletContext is similar to how an applet is related to an AppletContext. The facilities provided to a servlet is limited to getting MIME types, translating path names, finding other servlets, and logging information. These methods are as follows:

- **getMimeType(String filename).** Returns the MIME type of the specified file by matching the file name suffix against entries in the mime.properties file.
- **getRealPath(String path).** Applies the alias rules in the alias.properties file to the specified virtual path and returns the corresponding real path; null is returned if the translation fails.
- **getServerInfo().** Returns a string identifying the name, version, and platform of the server software and is equivalent to the CGI variable SERVER_SOFTWARE.
- **getServlet(String).** Returns a servlet of the given name.
- **getServlets().** Returns an enumeration of all the servlets currently loaded in the server. This enumeration includes the calling servlet.
- **log(Servlet, String).** Writes a message to the servlet log for a given servlet.

ServletRequest Interface Class

The ServletRequest interface class defines method used to obtain information about the servlet request independent of the actual transport protocol used between the client and server. The request must be carried by an IP-based transport protocol, such as HTTP. Implementations of ServletRequest for specific transports, such as HTTP, must provide the following protocol-related methods:

- **getRemoteAddr().** Returns the IP address of the client that sent the request (its equivalent CGI variable is REMOTE_ADDR).
- **getRemoteHost().** Returns the fully qualified host name of the agent that sent the request (its equivalent CGI variable is REMOTE_HOST).
- **getServerName().** Returns the host name of the server that received the request (its equivalent CGI variable is SERVER_NAME).
- **getServerPort().** Returns the port number on which this request was received (the CGI variable is SERVER_PORT).
- **getProtocol().** Returns a string identifying the protocol and version (the CGI variable is SERVER_PROTOCOL).

The request can include parameters or content information. The servlet can determine the length of any content sent with the request using getContentLength(). The servlet can also obtain the content encoding type using getContentType(). When content is available, the servlet can read it using a HttpInputStream obtained from getInputStream(). Implementations of ServletRequest for specific transports, such as HTTP, must provide the following content-related methods:

- **getParameter(String).** Returns the value of the specified parameter for the request.
- **getParameters().** Returns a hashtable of all the parameters of this request.
- **getContentType().** Returns the MIME type of any data included with the request, or null if the MIME type is not specified in the request header.
- **getContentLength().** Returns the size of the data included with the request, or –1 if a length is not specified in the request header.
- **getInputStream().** Returns an input stream for reading any data included with the request.

ServletResponse Interface Class

The ServletResponse interface class defines a general way to create the response to the service request. At this generic level, the response is just content with a MIME identifier and optional length. To write the actual content of the response, the servlet calls getOutputStream() to get a ServletOutputStream and then makes the necessary print() calls to compose the response. The methods to create the response are as follows:

- **setContentType(String).** Sets the MIME type of the response.
- **setContentLength(int).** Sets the content length for the response.
- **getOutputStream().** Returns a ServletOutputStream used for writing response data to the client.

GenericServlet Class

In general, the GenericServlet class is a base implementation class for any transport-independent servlet, and specifically it is used for any servlet that can be invoked from a server-side include. GenericServlet implements the Servlet interface and provides convenient access to information from the servlet's ServletContext. In addition to implementing init(), destroy(), service(), and getServletInfo() methods required by the Servlet interface, GenericServlet provides the following methods:

- **init().** Called after the servlet is loaded to initialize the servlet. Override this method to accept initialization parameters or to perform other initialization functions. A servlet is initialized only once, when it is loaded.
- **getInitParameter(String name).** Gets the value of an initialization parameter of the servlet as specified in either the servlet.properties file or as an attribute in the <servlet> tag for server-side includes servlets.
- **getInitParameters().** Returns a name/value pair hashtable of the initialization parameters of the servlet.
- **getServletContext().** Returns the servlet context.
- **log(String).** Logs a message into the servlet log file.

ServletInputStream

ServletInputStream is an abstract subclass of InputStream that defines the readLine(byte[], int offset, int count) method, which reads bytes into the byte array starting at the offset until the array is filled or a newline character is read.

ServletOutputStream Abstract Class

ServletOutputStream is an abstract subclass of OutputStream that is similar to a PrintStream, but only defines methods to print the most commonly used network data types (ints, longs, and strings). The implementation of ServletOutputStream handles conversion from internal Unicode encoding of characters to US-ASCII, which is generally used by Internet protocols. This class also defines methods to print and terminate the line with the Internet-standard CR/LF combination. The output methods are as follows:

- print(int), println(int)
- print(long), println(long)
- print(String), println(String)
- println()

You must convert other data types using toString() before printing to the ServletOutputStream.

The java.servlet.http Package

The java.servlet.http package supports servlets that specifically use HTTP. This package contains two interface classes, two implementation classes, and the implementation of a servlet class intended to be subclassed. As the API matures, other standard servlets may be included into the Servlet API definition. The HTTP-specific classes are HttpServlet, HttpServletRequest, HttpServletResponse, FormServlet, and HttpUtils. All are described in the following sections.

HttpServlet Abstract Class

Subclass HttpServlet, an abstract subclass of GenericServlet, to implement your HTTP-specific servlets. HttpServlet handles the cast conversion between the generic ServletRequest and the ServletResponse objects and their HTTP-specific subclasses. You override the service(HttpServletRequest, HttpServletResponse) method in your servlet class to perform the servlet processing function. The HttpServlet methods are as follows:

- **service(ServletRequest, ServletResponse).** Performs the cast conversion between generic and HTTP-specific request and response objects. Normally do not override this method, but instead override the following HTTP-specific method.
- **service(HttpServletRequest, HttpServletResponse).** Override this abstract method to accept HTTP service requests.

HttpServletRequest Interface Class

HttpServletRequest is a subclass of ServletRequest that handles the extra request information, other than content identification, contained in an HTTP request header. The information available from HttpServletRequest, beyond that of the base ServletRequest, falls into four general categories:

- General information about the request
- Information concerning the URL
- Authentication-related information
- Access information to the original HTTP header

The exact information available using these calls is described in the following sections.

General Request Information

These methods provide general information about the request, including the name of the method used to access the data and any query information:

- **getMethod()**. Returns a String containing the method used in the request, for example, "GET," "HEAD," or "POST" (the equivalent CGI variable is REQUEST_METHOD).
- **getQueryString()**. Returns the query string part of the URI, or null if none; the query string follows the ? in the URI (the CGI equivalent is QUERY_STRING).

URL Information

These methods provide information about the requested URL, its major components, and its local translation:

- **getRequestURI()**. Returns the request URI.
- **getServletPath()**. Returns the part of the request URI that refers to the servlet being invoked.
- **getPathInfo()**. Returns optional, extra path information following the servlet path, which immediately precedes the query string.
- **getPathTranslated()**. Returns extra path information translated to a real path using the alias.properties rules (CGI's variable is PATH_TRANSLATED).
- **getRequestPath()**. Returns the part of the request URI that corresponds to the servlet path plus the optional, extra path information.

Authentication-Related Information

The servlet can provide some very weak access control by using name and address information about the client as provided by getHostName() and getHostAddress(). The type of HTTP-level authentication in use is available using getAuthType(). Finally, for connections running in basic mode, getRemoteUser() returns the user identification:

- **getAuthType().** Returns the authentication scheme of the request, or null if none (CGI variable AUTH_TYPE).
- **getRemoteUser().** Returns the name of the user making this request; null is returned if the name is not supplied with the request (CGI's equivalent is REMOTE_USER).

Access to HTTP Request Header Fields

HttpServletRequest provides a means to get direct access to the HTTP request header fields to get information not explicitly supported by an HttpServletRequest method. Examples include user-agent, referer, preferred languages, accepted content encodings, and client-state (*cookie*) information, as well as all non-standard header fields.

Instead of using an enumerator, the following two methods to access the raw header take an index into the header. An index of 0 retrieves the first header field; null is returned if the index is greater than the number of fields available:

- **getHeader(int).** Returns the value of the nth header field; null is returned if there are fewer than *n* fields.
- **getHeaderName(int).** Returns the name of the nth header field; null is returned if there are fewer than *n* fields.

You can also access request header fields by name, which happens to be the easiest approach. In this case, the request header is searched for a matching field name and returns the field value. Use one of the following methods, depending on the desired data conversion:

- **getHeader(String fieldname).** Returns the value of a header field; if this method is not available, null is returned.
- **getIntHeader(String fieldname, int default).** Returns the value of an integer header field after converting the string to an int. If the field is not found, it returns the default value.
- **getDateHeader(String fieldname, long default).** Returns the value of a date header field after converting the string to a date. If the field is not found, it returns the default value.

HttpServletResponse Interface Class

The HttpServletResponse is a subclass of ServletResponse and provides the mechanisms to create an HTTP response header. These mechanisms are used for the following four purposes:

- To set the response status.
- To add arbitrary response header fields.
- To create an error response.
- To redirect the request to another URL.

Creating normal, error, and redirect responses using the HttpServletResponse object is described in the following sections.

Setting the Response Status

The return status defaults are 200 or OK. If an error is returned or OK is not the correct status, the servlet can specify both an error code and some descriptive text. If only an error code is given, the code is translated into a standard status text string. The two methods are as follows:

- **setStatus(statusCode).** Sets the status code in the returned HTTP header. The status code is translated into a text string using a standard definition of arguments.
- **setStatus(statusCode, messageString).** Sets the status code in the returned HTTP header.

Specifying Non-Standard Header Fields

Your servlet can add any field to the response header by using one of the setHeader() methods, leting you add non-standard header fields not supported by HttpServletResponse. Provide the name of the header field (without a colon) as the first argument and the data as the second argument. Variants of setHeader are defined for String, int, and date parameters. Successive calls to set the same header field overwrite the previous value and do not add multiple copies of the field. The four methods are as follows:

- **setHeader(String fieldname, String).** Sets the value of the named header field to the indicated value.
- **setIntHeader(String fieldname, int).** Sets the value of the named integer header field after converting the value to a string.

- **setDateHeader(String fieldname, long).** Set the value of the named date field after converting the long into an Internet-standard date-time string.
- **unsetHeader(String fieldname).** Removes the named header field from the response.

Returning Error Responses

HttpServletResponse provides two methods to create and send standard error responses. To return an error, your servlet calls sendError() with a status code and possibly an explanatory message. The page is titled and has a header that indicates the status code and a standard text message based on the status code. The optional explanatory message is inserted into the body of the returned page. Calling sendError(404, "Couldn't find it!") would create a returned page that has a title and H1 header of "404 Not Found" and "Couldn't find it!" as the text. These methods are

- **sendError(int).** Sends an error response to the client using the specified status code and no explanatory message.
- **sendError(int, String).** Sends an error response to the client using the specified status code and detailed explanatory message.

Returning Redirect Response

HttpServletResponse also provides a method to create and send a standard redirect response. Calling sendRedirect() creates and sends a "302" (Moved Temporarily) response. Pass a string that indicates the new location of the page. The string should be a URL because it is placed in the returned page as a link. However, sendRedirect() does not check to see whether the string is a correctly formatted URL. If you have a URL object already, convert it to a string using toString() before calling sendRedirect(). This method, sendRedirect(String), sends a redirect response to the client using the specified redirect location URL.

FormServlet Class

The FormServlet class is the only servlet class that is specified as part of the Servlet API, and hence it is available in all web servers that support servlets. Build forms processing servlets by subclassing FormServlet and overriding the sendResponse() method. The sendResponse() method handles form information and generates a response. As a subclass of HttpServlet, the following methods are defined for the FormServlet:

- **getServletInfo().** Override this method to return a String containing information (such as author and version) about your servlet.
- **service(HttpServletRequest, HttpServletResponse).** If special request processing is required, override this method in your servlet.
- **sendResponse(HttpServletResponse, Hashtable).** Override this method to supply code to process the form and generate a response.

An example construction of your own form processing servlet is provided in "Handling HTML Forms."

HttpUtils Class

The HttpUtils class is a set of static utility routines useful in building servlets. In Jeeves Alpha 2, the only method defined is parseQueryString(String). This method builds a Hashtable of name/value pairs based on parsing a query string. Your servlet can use the name (as the key) in the Hashtable.get() method to retrieve the value string.

The java.servlet.html Package

The third package that defines the Servlet API is java.servlet.html. This package includes 13 HTML support classes that provide operations for assembling HTML elements such as tables, imagemaps, and forms. From these elements, you can build an HTML page as a hierarchical list of HTML commands and text elements, and then you can write out the completed page to a ServletOutputStream. The classes are as follows:

- **HtmlElement.** Supports the basic interface definition
- **HtmlText.** Supports text enclosed by tags
- **HtmlContainer.** Supports a list of HtmlElements enclosed by tags
- **HtmlApplet.** Supports the applet HTML tag with width and height specification
- **HtmlChoice.** Supports the select HTML tag
- **HtmlDefinitionList.** Supports the dl, dt, and dd HTML tags
- **HtmlForm.** Supports building HTML forms comprised of input fields, check boxes, radio buttons, text areas, selections, and submit buttons
- **HtmlFrameset.** Supports frameset HTML tag
- **HtmlImageMap.** Supports client-side image map HTML tags
- **HtmlList.** Supports the ul and ol HTML list tags
- **HtmlPage.** Supports building a web page comprised of a header and body areas

- **HtmlTable.** Supports building of tables from row elements
- **HtmlRow.** Supports the HTML to define a row of a table

The methods in this package do not automatically add double quotes to enclose literal strings used as HTML attributes. You must add double quotes if your string has embedded spaces or other special control characters.

All the classes in this package implement the HtmlElement interface, which lets you specify text for the HTML tag and write the resulting HTML to an output stream. By creating objects that represent the various HTML tags and calling methods to add data and attributes to the tags, you can generate the most common HTML elements and avoid some simple errors, such as ending tags in the wrong order. However, it may be simpler to hand-generate the HTML, particularly for simple, static pages. The examples in the section "Writing Servlets" make use of the classes from the java.servlet.html package to generate dynamic response pages.

The parameter layout of these methods is similar for all classes. The first parameter is the information that is to be added to the HtmlElement. Methods are often defined to handle the addition of Strings, HtmlElements, or Vectors.

The second parameter is eitherthe tag or attributes for a tag implied by the method. For example, addLink() implies an "a" tag, so the second parameter is interpreted as attributes of the tag. For tags, do not include the <, >, or / characters in the tag string; these characters are automatically supplied where needed. Multiple tags are specified using a comma-delimited string.

These HTML generation classes are new to the Jeeves Alpha 2 release. The contents of the java.servlet.html package and the API are likely to evolve in future releases. The final API may be significantly different from that discussed in this section.

These HTML support classes do not check the actual HTML to verify that the tags and keys are legal and used correctly. It only ensures that tags are terminated in the correct order; for example, it verifies whether </body> occurs before the </html> command.

HtmlElement Interface

The HtmlElement interface defines two methods provided by all classes in the java.servlet.html package. These are as follows:

- **wrap().** Adds HTML tag pairs (such as <h1> and </h1>) to the element such that it follows previously added tags and encloses the text of the element. The tag or tags are specified as the argument to wrap(). Conceptually, an HtmlElement has beginning tags,text, and ending tags.
- **write().** Writes the HTML to an OutputStream.

HtmlText Class

HtmlText is a fundamental class in the package because it supports text enclosed by HTML tags.

- **add(String), add(String, String tag).** Adds the text, which can be optionally wrapped in the tag.
- **addTag(String), addTag(String, String).** Adds a single, non-paired tag, such as <p>, with optional tag attributes.

HtmlContainer Class

HtmlContainer is a base class for elements that herarchically contain other HTML elements. Conceptually, an HtmlContainer has beginning tags, a list of HtmlElements, and ending tags. For example, an HtmlPage is a subclass of HtmlContainer.

HtmlContainer supports wrap(), write(), and the various types of add() and addTag() methods. Unique to HtmlContainer are the following methods:

- **addImg(String).** Adds an <IMG> tag where the String is the URL of the image's source (SRC attribute).
- **addLink(String text, String URL).** Adds a link (anchor tag) where "text" is the text occuring between the tags and "URL" is the URL for the link (HREF attribute).

HtmlPage Class

The HtmlPage class represents a complete HTML page that has a header, title, and body sections. Create a new HtmlPage with a String that is the page's title. HtmlPage automatically creates the <html><head><title>, </title></head><body>, and </body></html> commands. All HtmlElements added to an HtmlPage are added into the body area of the page.

Writing Servlets

Together the Servlet API and the Jeeves HTTP server provide an environment that makes writing servlets very easy. All servlets must implement the java.servlet. Servlet interface by subclassing either GenericServlet or HttpServlet. When an HTTP request is received, the server translates the URL into a reference to a specific servlet class, as was previously

described in the "Jeeves HTTP Server Overview" section. If this is the first request for the servlet, the servlet class is dynamically loaded, and its init() method is called to perform any necessary servlet-specific initialization. In most cases, the servlet has already been loaded in which case its service() method is just called.

Note that Jeeves is a multithreaded server and several different threads can concurrently executing the service() method of the same servlet object. For example, a single instance of the FileServlet object serves all requests to load files. Consequently, servlets must be designed to operate in multithreaded environment. The easiest way to ensure this is to store all information about the request being processed in local variables and not store it as instance variables in the object. For cases where access to a single resource must be controlled, you must use the standard Java synchronization techniques based on the "synchronized" keyword. In most cases, you can just add the "synchronized" modifier to the declaration for service() or sendResponse() and effectively modify the servlet to serve only one client at a time.

The following sections contain examples of a simple servlet, form processing servlets, an error response servlet, and a server-side include servlet.

Simple Servlet

To learn how to write a servlet, build a simple servlet that displays a table of the various HTTP information available from the request. Subclass HttpServlet and override the service() method as shown in the following code:

```
public class SimpleServlet extends HttpServlet {
  public void SimpleServlet() {
  }

  public void service (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
```

Your servlet can create its response using either the HttpServletResponse passed to the service() method or by resetting the HttpServletResponse and writing the response header directly to the ServletOutputStream.

Before calling service(), Jeeves sets the HttpServletResponse status to 200 or "OK," the "Date:" header field is set to the current date and time, and the "Server:" header field is set to identify the server. If there is an error or "OK" is not the correct status, the servlet can specify both an error code and some descriptive text. If only an error code is given, the code is translated into a standard status text string. Normally you would not change these values.

You do need to identify the type of data contained in the response by specifying the MIME type; and you may optionally specify the content length. To write the content, the servlet calls getOutputStream() to get a ServletOutputStream as follows:

```
res.setContentType("text/html");
ServletOutputStream out = res.getOutputStream();
```

You can create your HTML either manually by using print() statements or by using the HTML support package defined as part of the Servlet API. Use the HTML classes and create a new HtmlPage. The information from the request is displayed as a table. First, display the information available only through method calls:

```
HtmlPage page = new HtmlPage("HTTP Request Display");
page.add("HTTP Request Information", "h1");

HtmlTable tbl = new HtmlTable("Border");
tbl.addHeader("Parameter");
tbl.addHeader("Value");
nextRow(tbl, "Server name", req.getServerName());
nextRow(tbl, "Server port", Integer.toString(req.getServerPort()));
nextRow(tbl, "Client host", req.getRemoteHost());
nextRow(tbl, "Client addr", req.getRemoteAddr());
nextRow(tbl, "Method", req.getMethod());
nextRow(tbl, "Request URI", req.getRequestURI());
nextRow(tbl, "Path info", req.getPathInfo());
nextRow(tbl, "Path Xlated", req.getPathTranslated());
nextRow(tbl, "Query String", req.getQueryString());
```

Next, get a Hashtable containing all of the HTTP request header parameters and iterate through them, as follows:

```
Hashtable params = req.getParameters();
Enumeration enum = params.keys();
while (enum.hasMoreElements()) {
  String name = (String) enum.nextElement();
  String value = (String) params.get(name);
  tbl.newRow();
  tbl.addData("<strong>" + name + ":</strong>");
  tbl.addData("<em>" + value + "</em>");
}
```

Finally, write the HTML for the page to the ServletOutputStream. The response header that you create is automatically written to the ServletOutputStream before your first write is handled. This code completes the service() method:

```
    page.add(tbl);
    page.write(out);
    }
```

The nextRow() method formats each line of the response as follows:

```
    void nextRow(HtmlTable tbl, String name, String value) {
    tbl.newRow();
    tbl.addData("<strong>" + name + ":</strong>");
    tbl.addData("<em>" + value + "</em>");
    }
}
```

This completes the SmpleServlet class definition. The servlet can also use ServletResponse to return a standard error or redirect response.

Handling HTML Forms

Many of the functions performed by CGI programs center around HTML forms handling. When using HTML forms, the browser extracts the information entered into the form by the user and sends it to the server as either a GET or POST request. Because form processing is such a common servlet activity, the Servlet API includes a built-in Servlet class specific for forms handling: The FormServlet extracts the form information from the query string for GET requests or from the content for POST requests. The differences between GET and POST are invisible to your subclass of FormServlet.

In the following example, FormServlet is subclassed to build a form-processing servlet that simply reads and displays the contents of a form:

```
public class FormDisplayServlet extends FormServlet {
  public void FormDisplayServlet() {
  }
```

FormServlet's service() method has already accomplished the following:

- Verified the HTTP method as GET or POST
- Retrieved the form information from either the URL for a GET request or from the content of the request for a POST
- Stored the form information as a Hashtable of name/value pairs

Your subclass should override the sendResponse() method to do any request processing as follows:

```
public void sendResponse(HttpServletResponse res, Hashtable params)
  throws IOException {
ServletOutputStream out = res.getOutputStream();
```

You must always specify the content-type on responses that include data. Allocate a HtmlPage to help construct the HTML response page and add a level-one header as shown in the following:

```
res.setContentType("text/html");
HtmlPage page = new HtmlPage("Form Display");
page.add("Form Display", "h1");
```

Because the FormServlet superclass has already stored the form information in a Hashtable, performing forms handling is a simple matter of retrieving data from the Hashtable and processing it. For this example, processing is simply formatting the forms data as a table that is returned to the browser. Access the data using an Enumerator and add a row to the table for each name/value pair in the form request as shown in the following:

```
HtmlTable tbl = new HtmlTable("Border");
tbl.addHeader("Input Element");
tbl.addHeader("Value");
int count = 0;
Enumeration enum = params.keys();
while (enum.hasMoreElements()) {
  String name = (String) enum.nextElement();
  String value = (String) params.get(name);
  count++;
  tbl.newRow();
  tbl.addData("<strong>" + name + "</strong>");
  tbl.addData(new HtmlText(value, "em"));
}
```

Note the two different ways to include HTTP format information in the table cells. The first involves manually creating a string that contains the desired HTML commands. The second uses the HTML support classes to generate a text element that is added to the table. The use of one approach instead of using the other is primarily a style issue, except in two cases. The HTML classes are a simplification of the complete language; you are required to manually create HTML code for attributes not supported by the classes. Alternatively, when your HTML processing is structured or repeating as in this example, it is easier to use the support classes because they reflect the hierarchical nature of HTML.

After you finish building the table, write a summary of the form information, add the table to the page, and then write the HTML for the complete page to the output stream. The first write to the ServletOutputStream triggers a write of the HTTP response header to the output stream. Even though the ServletOutputStream is buffered, you do not need to call the flush() method; the server automatically calls the stream's flush() method. The following code completes the sendResponse() method. Remember that the ➥ symbol indicates a line that normally fits on the previous line.

```
    page.add("<p>This page contains " + Integer.toString(count) + " input
    ➥elements.\n");
    page.add(tbl);
    page.write(out);
    }


}
```

This completes the definition of the FormDisplayServlet class.

Extending the FormServlet

One limitation of the standard FormServlet is that the subclasses that implement the forms processing do not have access to the request information contained in the HTTP header. Only the name/value pairs from the form are provided to FormServlet subclasses when performing form processing. The main reasons why the request information is needed is to obtain state information from the client (using cookies), to verify the client's ability to accept certain MIME types or to obtain the user's name when performing additional access control checks. The following modified version of FormServlet passes the HttpServletRequest information as an argument of the sendResponse() method:

```
public abstract class NewFormServlet extends HttpServlet {
  public void NewFormServlet() {
  }

  public void service (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
```

First determine the HTTP method being used. For GET requests, the form information is encoded into the URL after the servlet reference and can be obtained by calling getQueryString(). For POST methods, the processing is a bit more complex; the form information follows the request header and must be identified as being URL encoded. The getContents() method, which follows, reads the request content and returns a string.

The following code retrieves the query information:

```
String formdata;
String method = req.getMethod();
if ("GET".equals(method))
  formdata = req.getQueryString();
else if ("POST".equals(method)) {
  if (!"application/x-www-form-
urlencoded".equalsIgnoreCase(req.getContentType()))
    throw new IOException("illegal content type");
  formdata = getContents(req);
}
else
  throw new IOException("invalid method");
```

After you have the query information, call parseQueryString() to construct a Hashtable of name/value pairs. It decodes the special URL encoding where " " (the space character) is represented as "+," and non-printing characters are represented as a number preceded by a percent sign (%). Finally, call the servlet using the newly defined service() method as shown in the following:

```
Hashtable fd = HttpUtils.parseQueryString(formdata);
sendResponse(req, res, fd);
}
```

The NewFormServlet class defines a variant of the sendResponse() method that accepts information about the query in addition to information from the form. This new method is defined as follows:

```
public abstract void sendResponse(HttpServletRequest req,
 ➥HttpServletResponse res,
  Hashtable formdata);
```

The getContents() method reads the request content and returns it as a string. Note the loop structure around the read() call; this loop guarantees that the read() fully completes with all request information. The code to read the request content is as follows:

```
  public String getContents(ServletRequest req) throws IOException {
  InputStream in = req.getInputStream();
  int count;
  byte buf] = new bytereq.getContentLength()];
  for (int index = 0; index < buf.length; index += count) {
    count = in.read(buf, index, buf.length - index);
    if (count == -1)
```

```
        throw new IOException();
    }
    return new String(buf, 0);
    }
}
```

This completes the definition of the NewFormServlet class.

Building Server-Side Includes Servlets

Jeeves provides support for a form of server-side includes that uses the newly defined <servlet> tags. Files whose MIME type is "java-internal/parsed-html" are further processed by the SSIncludeServlet to resolve any references to servlets embedded in the document. By default, files with the .shtml suffix are scanned.

As the document is written to the client, the SSIncludeServlet scans for the newly defined <servlet> HTML tag. When this tag is encountered, the servlet identified in the attributes of the tag is invoked and its output is sent to the client in place of the <servlet> </servlet> HTML element.

The server-side includes supported by SSIncludeServlet are not the same as those supported by many other HTTP servers, such as NCSA and Netscape; these embed server-side includes commands inside HTML comments.

The syntax for invoking servlets is as follows:

```
<servlet name=Name code=Code.class codebase=CodeBase
initParam1=initArg1 initParam2=initArg2 ...>
<param name=param1 value=val1>
<param name=param2 value=val2>
.
.
.
</servlet>
```

The server first tries to invoke the servlet with the name provided in the name field. If this fails or if no name is provided, the server then attempts to load the servlet based on the code and codebase fields. Any remaining attribute fields within the <servlet> tag are passed as initialization parameters to the servlet. If the server loads the servlet and a name was specified, the server keeps the servlet loaded so that the next time the servlet is accessed, it is not reloaded. If no name is specified, the server reloads the servlet each time it is accessed.

After the servlet is ready to execute, the server calls the service() method of the servlet. The name/value pairs specified in the HTML file with the <param> tag are accessible to the servlet by using the standard getParameter() and getParameters() methods on the ServletRequest object that is passed to the servlet in the service() method. Everything the servlet writes to ServletResponse.getOutputStream() is also written to the client as part of the document that the client requested.

The following section contains an example server-side include servlet that inserts the current time (at the server) into a document.

DateSSIServlet

The DateSSIServlet is a servlet designed to work as a server-side include. This outputs the current date and time into the document that referenced this server-side include servlet. Servlets that can function as server-side include the GenericServlet subclass rather than the more common HttpServlet because HTTP information is not available. The class definition for DateSSIServlet and its service() method is as follows:

```
public class DateSSIServlet extends GenericServlet {
  public void service(ServletRequest req, ServletResponse res)
    throws ServletException, IOException {
  Date today = new Date();
  res.setContentType("text/plain");

  ServletOutputStream out = res.getOutputStream();
```

To enable the user to control whether to display the time in the local time-zone or in GMT, use an initialization parameter. For HttpServlets, the initialization parameters are obtained from the configuration file servlet.properties. For server-side includes, the user specifies initialization parameters as additional name/value pairs within the servlet tag. The following code determines the time format (local or GMT) and performs the indicated operation:

```
String zone = getInitParameter("date");
  if ("local".equalsIgnoreCase(zone))
    out.println(today.toString());
  else
    out.println(today.toGMTString());
  }

}
```

This completes the definition of the DateSSIServlet class. The HTML code to invoke this servlet to output local time to a document is as follows:

```
<servlet name=dateSSIServlet .date=local></servlet>
```

The ending </servlet> tag is required even though this servlet does not take parameters. Note that this servlet can be invoked directly rather than just as a server-side include.

Subclassing ErrorServlet

When calling HttpServletResponse's sendError() method with only a status code to return an error to the requester, the ErrorServlet is actually called to create and send the error reply. The built-in ErrorServlet returns either a simple error reply message generated by translating the status number into a standard error text string, or it returns the contents of a file from the preconfigured error page directory. Calling sendError() with a status code and message string bypasses the ErrorServlet and returns a simple page that contains the error text message.

Assume that you have a busy site that is the target of many hypertext links. When files are moved to new directories, you will have many old links that will cause "not found" errors. By changing the entry in the servlet.properties file that specifies the class to be loaded for the "error" servlet, you can have your subclass of ErrorServlet invoked rather than the standard built-in version. By overriding the service() method, you can dynamically create custom error reply pages containing links to possible alternates to the requested URL.

In the following example, the service() method is overridden, but not init(), so that the original ErrorServlet will be properly initialized.

```
public class NewErrorServlet extends ErrorServlet {
  public NewErrorServlet() {
  }
```

Your service() method is called with the original request's HttpServletRequest object which has information about the requester; obtain the status code from the HttpServletResponse object. First determine whether the error is "not found," code 404. If not, just call super.service() to get the standard error reply page. The service() method is defined as follows:

```
public void service(HttpServletRequest req, HttpServletResponse res)
  throws IOException {
if (((HttpServletResponse)res).getStatusCode() != 404) {
  super.service(req, res);
  return;
}
```

If the status is 404, you use the URL request string to lookup a list of potential matches by calling getAlternates(). If no alternates are found, call super.service() for the standard error reply as shown in the following code:

```
String request = req.getRequestURI();
Vector alternates = getAlternates(request);
if (alternates.isEmpty()) {
  super.service(req, res);
  return;
}
```

The getAlternates() method returns a Vector of URL strings that are similar to the requested URL. This method might reference a hand-generated table, consult an automatically generated site index, or perform a real-time search of the file system; the exact mechanism is not defined for this example. The intent is to present a list of candidate links to the user. You can then construct the reply page with this information as shown in the following code:

```
res.setContentType("text/html");
res.setDateHeader("Date", System.currentTimeMillis());
ServletOutputStream out = res.getOutputStream();
HtmlPage page = new HtmlPage("404 File not Found");
page.add("404 - File not found", "h1");
page.add("The file <strong>" + request + "</strong> was not found. ");
page.add("You may have misspelled the URL or the file has moved. ");
page.add("Here is a list of possible links that are similar to your
➥original request:");
```

After adding the initial message to the error reply page, construct a table of links to the candidate documents. When the table is complete, add it to the page and write the page's HTML to the ServletOutputStream. The first write to the ServletOutputStream causes the response header to be written to the ServletOutputStream. The following code completes the definition of the service() method:

```
HtmlTable tbl = new HtmlTable();
Enumeration enum = alternates.elements();
while (enum.hasMoreElements()) {
  String link = (String)(enum.nextElement());
  tbl.newRow();
  tbl.addData(new HtmlText(link, "a href=" + link));
}
page.add(tbl);
```

```
page.write(out);
}
```

The getAlternates() method is called to build a list of alternate URLs for the indicated file. The exact technique for constructing this list is undefined in this example. Depending on your application, this method could reference a manually constructed file or perform a search for similar file names. The following code is only the shell of the method:

```
  Vector getAlternates(String file) {
  Vector v = new Vector();
     .
     .
     .
  return v;
  }

}
```

This completes the definition of the NewErrorServlet class.

Summary

The Jeeves Alpha 2 release covered in this chapter is just a beginning step toward defining a radically new architecture for server software. Taken to its logical conclusion, servlets can evolve into the server-side equivalent of application components, and Jeeves can become the future of server software component architecture—a Java Beans for the server. This new architecture goes beyond simply replacing CGI programs to completely changing server implementations. Jeeves shows how an HTTP server can be remolded to use servlets to significantly increase flexibility and customization. As the servlet technology matures, more standard servlets will be defined, such as standard servlet bridges to databases, and new server-side technologies, such as JDBC, will be first introduced as servlets.