การสื่อสารระบบมัลติมีเดียผ่านคู่สายโทรศัพท์

MULTIMEDIA COMMUNICATION VIA TELEPHONE LINE

โดย

นายธรรมรัตน์ วรรณราช

นางสาวศิริกานต์ ชูเชิด

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

การสื่อสารระบบมัลติมีเดียผ่านคู่สายโทรศัพท์

MULTIMEDIA COMMUNICATION VIA TELEPHONE LINE

โดย

นายธรรมรัตน์ วรรณราช 37014164

นางสาวศิริกานต์ ชูเชิด 37014435

อาจารย์ที่ปรึกษา

รศ.ดร.สุวิพล สิทธิชีวภาค

ผศ.เกรียงไกร วงศ์โรจนภรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การสื่อสารระบบมัลติมีเดียผ่านคู่สายโทรศัพท์**

MULTIMEDIA COMMUNICATION VIA TELEPHONE LINE

ผู้จัดทำ

1. **นายธรรมรัตน์ วรรณราช 37014164**
2. **นางสาวศิริกานต์ ชูเชิด 37014435**

สุวิพล

อาจารย์ที่ปรึกษา

(รศ.ดร. สุวิพล สิทธิชีวภาค)

อาจารย์ที่ปรึกษา

(ผศ.เกรียงไกร วงศ์โรจนกรณ์)

การสื่อสารด้วยระบบมัลติมีเดียผ่านคู่สายโทรศัพท์

MULTIMEDIA COMMUNICATION VIA TELEPHONE LINE

โดย นายธรรมรัตน์ วรรณราช 37014164
นางสาว ศิริกานต์ ชูเชิด 37014435

อาจารย์ที่ปรึกษา รศ.ดร.สุวิพล สิทธิชีวภาค
ผศ.เกรียงไกร วงศ์โรจนภรณ์

บทคัดย่อ

โครงงานนี้เป็นการเขียนโปรแกรมประยุกต์เพื่อรับ-ส่งข้อมูลที่เป็นทั้งภาพ เสียง และการพิมพ์ข้อความโต้ตอบระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง ผ่านทางพอร์ตสื่อสารแบบอนุกรมตามมาตรฐานของ RS-232 และผ่านโมเด็มไปตามสายโทรศัพท์ โดยใช้โปรแกรมเดลไฟล์ (Delphi) เป็นเครื่องมือสำหรับพัฒนาโปรแกรม

Abstract

This project is presented the program for sending and receiving data(text,voice and picture) between two computers by standard RS-232 serial port and MODEM via telephone line.The program is designed by Delphi 2.0.

สารบัญ

สารบัญรูปภาพ

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

ในยุคปัจจุบันนี้ได้มีการนำเอาเทคโนโลยีของคอมพิวเตอร์ ไปประยุกต์ใช้ในงานต่าง ๆ จนแพร่หลายมากขึ้นทั้งในงานอุตสาหกรรม การแพทย์ วิศวกรรม และในระบบสำนักงาน เมื่อมีการใช้งานมากขึ้น จำนวนข้อมูลที่ใช้ก็เพิ่มมากขึ้นตามไปด้วย จึงมีความต้องการข้อมูลจากที่อื่นมาใช้ อันเป็นที่มาของการสื่อสารข้อมูลระหว่างกัน

สมัยก่อนการติดต่อระหว่างกันมักใช้จดหมายหรือเอกสารส่งไปให้ปลายทาง ต่อมาพัฒนามาใช้โทรสารแทน ซึ่งรวดเร็วขึ้นและสะดวกกว่าเมื่อก่อน แต่ก็ยังไม่ทำให้คอมพิวเตอร์รับส่งข้อมูลกับที่ต่าง ๆ ได้โดยตรง ถ้าผู้รับต้องนำข้อมูลไปใช้งานในเครื่องคอมพิวเตอร์ ก็ต้องป้อนเข้าเครื่องอีกทีหนึ่ง นอกจากจะเสียเวลาแล้วยังอาจเกิดข้อผิดพลาดได้อีกด้วย ข้อจำกัดอันนี้ทำให้การสื่อสารข้อมูลมีความสำคัญขึ้นมาทันที

จะเห็นว่าคอมพิวเตอร์มีบทบาทอย่างมากคู่กับการสื่อสารข้อมูล โดยปกติการสื่อสารข้อมูลระหว่างคอมพิวเตอร์วิธีที่ง่ายที่สุดก็คือ การรับส่งข้อมูลผ่านสายเคเบิลโดยตรง ถ้าระยะทางไม่ห่างกันมากนัก หรืออาจรับส่งข้อมูลผ่านระบบเครือข่ายของคอมพิวเตอร์เอง เช่น LAN(Local Area Network) ,WAN(Wide Area Network) หรือ PDN(Public Data Network) ซึ่งเป็นการรับส่งข้อมูลระหว่างกันโดยผ่านโมเด็ม(MODEM) ไปตามสายโทรศัพท์ทำให้สามารถส่งข้อมูลไปได้ไกลทั่วโลกเท่าที่ระบบโทรศัพท์จะเข้าไปถึง

สำหรับโครงงานนี้เป็นการรับส่งข้อมูลรูปแบบต่าง ๆ ทั้งข้อมูลที่เป็นตัวอักษร เสียง และภาพ โดยผ่านโมเด็มไปตามสายโทรศัพท์ โดยที่ในภาคการศึกษาแรกได้ทำการทดลองการรับ-ส่งข้อมูลระหว่างคอมพิวเตอร์สองเครื่องผ่านทางพอร์ตสื่อสารแบบอนุกรม สำหรับในภาคการศึกษาที่สองนี้ได้ทำการทดลองการรับ-ส่งข้อมูลผ่านคู่สายโทรศัพท์โดยให้โมเด็มผ่านข้อมูลไปตามสายโทรศัพท์ต่อไป

บทที่ 2
ทฤษฎีหรือหลักการ

2.1 การสื่อสารแบบอนุกรม

2.1.1 การส่งข้อมูลแบบอนุกรม

ข้อมูลที่ถูกส่งออกไปแบบอนุกรมนั้น โดยปกติแล้วจะเป็นข้อมูล "0" หรือ "1" เรียงต่อกันเป็นชุด ๆ ซึ่งปกติแล้วข้อมูลที่ใช้สื่อสารกันทั่วไป จะประกอบไปด้วยข้อมูลจำนวนแปดบิตต่อหนึ่งตัวอักษร โดยข้อมูลแต่ละชุดนี้จะสามารถแทนตัวอักษรที่แตกต่างกันได้ถึง 256 แบบ ในเครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer : PC) ข้อมูลจะถูกเก็บอยู่ในรูปของข้อมูลขนาดแปดบิตหรือหนึ่งไบต์อยู่แล้วไม่ว่าจะใช้ระบบปฏิบัติการ DOS, OS/2 หรือใช้เครื่อง PC ชนิด 32 บิตก็ตาม

ข้อมูลบางแบบนั้นอาจจะมีขนาดเพียงเจ็ดบิต อย่างเช่นรหัส ASCII ที่ใช้กันในคอมพิวเตอร์ทั่ว ๆ ไป นั้นก็เป็นรหัสขนาดเจ็ดบิต ดังนั้นมันจึงใช้แทนตัวอักษรได้เพียง 128 ตัวเท่านั้น ยกตัวอย่างเช่นตัวอักษร A นั้นจะมีรหัส ASCII ที่ใช้แทนมัน คือรหัสหมายเลข 65

ในการส่งข้อมูล ผ่านพอร์ตแบบอนุกรมนั้น ข้อมูลแต่ละไบต์จะประกอบด้วย

1. บิตเริ่มต้น (start bit) 1 บิต
2. บิตข้อมูล (data bit) 7 หรือ 8 บิต
3. พาริตี้บิต (parity bit) จะมีหรือไม่มีก็ได้
4. บิตสิ้นสุด (stop bit) 1 หรือ2 บิต

ข้อมูลแบบอนุกรมจะถูกส่งไปตามสายนำสัญญาณโดยทยอยส่งออกไปทีละบิต ซึ่งบิตต่ำสุดจะถูกส่งออกไปก่อน แล้วติดตามด้วยบิตที่สูงขึ้นมาเรื่อย ๆ จนครบชุดข้อมูลหนึ่ง ๆ สมมติว่าถ้าต้องการที่จะส่งตัวอักษร A ออกไปแบบอนุกรม ซึ่งตัวอักษร A นั้นมีรหัส ASCII ที่เป็นเลขฐานสองคือ 01000001 ดังนั้นบิตที่ถูกส่งออกไปก่อน ก็คือบิตที่อยู่ทางขวาสุดซึ่งก็คือ 1 นั่นเองและหลังจากนั้นก็เป็น 0 ตัวถัดมาทางซ้ายเรียงต่อ ๆ กัน ตามลำดับไปจนครบข้อมูลหนึ่งชุด

2.1.2 การสื่อสารแบบ Synchronous และ Asynchronous

การสื่อสารของเครื่อง PC ส่วนใหญ่นั้นเป็นแบบ Asynchronous ซึ่งก็หมายความว่าไม่มีการกำหนดช่วงเวลาตายตัวที่ใช้ส่งข้อมูลหนึ่งตัวอักษรออกไปที่พอร์ตอนุกรม นั่นก็หมายความว่าการส่งข้อมูลแต่ละครั้งนั้นจะต้องมีบิตเริ่มและบิตหยุดเป็นตัวกำหนดขอบเขตของข้อมูล บิตที่แสดงขอบเขตนี้จะถูกสร้างขึ้นโดยอัตโนมัติจากวงจรสื่อสาร แต่สำหรับการสื่อสารแบบ Synchronous นั้นบิตเริ่มและบิตหยุดไม่มีความจำเป็น เนื่องจากว่า ได้กำหนดช่วงเวลาการส่งข้อมูลต่อหนึ่งตัวอักษรเอาไว้แล้ว

เมื่อ PC ได้ทำการเชื่อมต่อเข้าเครื่องคอมพิวเตอร์โฮสอื่น ๆ โมเด็มจะใช้การติดต่อแบบ Asynchronous เสมอ ส่วนการสื่อสารแบบ Synchronous นั้นจะใช้เฉพาะในงานพิเศษ ๆ ที่ต้องการความเร็วใน

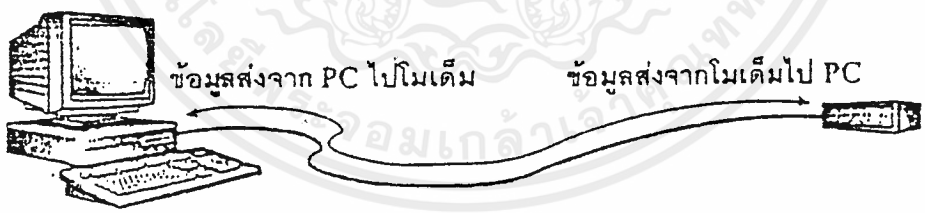
การส่งข้อมูลสูงมาก เช่นในการส่งข้อมูลระหว่าง LANหรือระบบเครือข่าย Packet switching ซึ่งวงจรสื่อสารที่พบในเครื่อง PC ทั่วไปนั้นไม่สามารถทำงานในแบบ Synchronous ได้

การสื่อสารแบบ Synchronous นั้นมีประสิทธิภาพการทำงานที่สูงกว่าแบบ Asynchronous มาก แต่จะต้องการวงจรพิเศษที่ต้องใช้สำหรับการสื่อสารแบบ Synchronous โดยเฉพาะข้อมูลที่ส่งแบบ Synchronous นั้นจะไม่มีการขาดตอน โดยข้อมูลจะถูกแยกออกเป็นชุด ๆ จากสัญญาณนาฬิกาที่ส่งไปพร้อม ๆ กันในสายนำสัญญาณอีกเส้นหนึ่ง ดังนั้นจึงมีความเร็วในการส่งข้อมูลที่สูงมาก

2.1.3 มาตรฐานการเชื่อมต่อแบบ RS-232-C

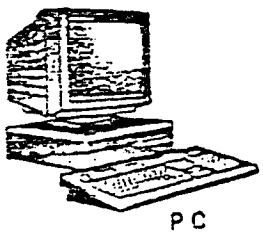
การเชื่อมต่อแบบอนุกรมนั้นเป็นวิธีการเชื่อมต่อสื่อสารทั่วไปของอุปกรณ์ทางอิเลคทรอนิคส์ต่าง ๆ องค์กรอุตสาหกรรมอิเล็คทรอนิคส์ของสหรัฐ (EIA) ซึ่งเป็นองค์กรที่เกี่ยวข้องโดยตรงกับอุตสาหกรรมอิเล็คทรอนิคส์ทั่วโลก ได้กำหนดมาตรฐานของการเชื่อมต่อและคุณสมบัติของพอร์ตอนุกรมสื่อสารขึ้นมา และมาตรฐานอันหนึ่งที่ได้รับการยอมรับโดย EIA คือมาตรฐานที่ 232 แบบ C หรือเป็นที่รู้จักกันในชื่อของ RS-232-C (Regulation Standard No.232 Type C) มาตรฐานการเชื่อมต่อแบบนี้มีความคล้ายคลึงกับมาตรฐาน CCITT V.24 และมีคุณสมบัติทางไฟฟ้าใกล้เคียงกับมาตรฐาน CCITT V.28 และนอกจากนั้นก็มีความใกล้เคียงกับมาตรฐาน Mil-Std-188C ของกระทรวงกลาโหมสหรัฐฯด้วย

มาตรฐาน RS-232-C นั้นแบ่งการเชื่อมต่อออกเป็น 2 ลักษณะ คือการต่อกับเทอร์มินัล (DTE : Data Terminal Equipment) และการต่อกับอุปกรณ์สื่อสารข้อมูล (DCE : Data Communication Equipment) ซึ่งในกรณีปกตินั้น DCE จะต้องต่อเข้ากับ DTE เสมอ ยกตัวอย่างเช่น การต่อโมเด็มเข้ากับเครื่อง PC โดยเครื่อง PC จะเป็นอุปกรณ์ DTE และโมเด็มจะเป็นอุปกรณ์ DCE



รูปที่ 2.1 รูปแสดงการต่ออุปกรณ์สื่อสารผ่านทางพอร์ตอนุกรม RS-232-C

จากรูปที่ 2.1นั้นเป็นรูปที่แสดงการเชื่อมต่อระหว่างเครื่อง PC กับโมเด็มโดยผ่านทางพอร์ตอนุกรม RS-232-C โดยที่เครื่อง PC จะส่งและรับข้อมูลจากโมเด็ม การเชื่อมต่อตามมาตรฐาน RS-232-C นั้นโดยปกติจะใช้คอนเน็กเตอร์รูปตัว D ชนิด 25 ขา กำหนดให้ปลายสายสัญญาณด้านหนึ่งที่เป็นคอนเน็กเตอร์ตัวผู้ใช้ต่ออยู่กับอุปกรณ์ DCE และปลายของสายสัญญาณอีกด้านหนึ่ง จะต้องเป็นคอนเน็กเตอร์ตัวเมียใช้ต่ออยู่กับอุปกรณ์ DTE แต่ต่อมา บริษัท IBM นั้นก็ได้ออกแบบคอนเน็กเตอร์แบบ 9 ขา ขึ้นมาใช้แทนคอนเน็กเตอร์แบบ 25 ขา



Pin 1 - Protective Ground
→ Pin 2 - Transmitted Data (TxD)
← Pin 3 - Received Data (RxD)
→ Pin 4 - Request to Send (RTS)
← Pin 5 - Clear to Send (CTS)
← Pin 6 - Data Set Ready (DSR)
Pin 7 - Signal Ground
← Pin 8 - Carrier Detect (CD)
→ Pin 20 - Data Terminal Ready (DTR)
← Pin 22 - Ring Indicator (RI)

รูปที่ 2.2 สายนำสัญญาณที่ใช้ในมาตรฐาน RS-232-C

จากรูปที่ 2.2 แสดงสายนำสัญญาณที่ใช้ในมาตรฐาน RS-232-C นั้นจะมีทั้งหมด 9 เส้น ประกอบด้วยวงจรข้อมูล 2 วงจร คือ

วงจรชุดที่ 1 ประกอบไปด้วย

- ขานำสัญญาณหมายเลข 2 และ 3 ซึ่งจะใช้ในการส่งผ่านข้อมูลระหว่างเครื่อง PC และโมเด็ม
- ขานำสัญญาณหมายเลข 1 และ 7 เป็นกราวน์

วงจรชุดที่ 2 ประกอบไปด้วย

- ขานำสัญญาณหมายเลข 4 และ 5 เป็นวงจรควบคุมสัญญาณที่เรียกว่า RTS และ CTS ซึ่งจะใช้ในการควบคุมการไหลของข้อมูลระหว่าง PC และ โมเด็ม (Hardware Flow Control)
- ขานำสัญญาณหมายเลข 8 เป็นสัญญาณ Carrier Detect (CD) ใช้บอก PC ว่า ในขณะนั้นโมเด็มได้รับสัญญาณพาหะอยู่หรือไม่
- ขานำสัญญาณหมายเลข 20 เป็นสัญญาณ Data Terminal Ready (DTR) ใช้บอกโมเด็มว่าในขณะนั้นเทอร์มินัลพร้อมที่จะติดต่อกับโมเด็มหรือไม่
- ขานำสัญญาณหมายเลข 22 จะใช้เป็นสัญญาณแสดงว่าในขณะนั้นโมเด็มได้รับสัญญาณกระดิ่งหรือไม่ ซึ่งมักจะไม่ได้ถูกใช้งานมากนัก

ในการสื่อสารแบบอนุกรมนั้น การที่จะทำให้อุปกรณ์อิเล็กทรอนิกส์สื่อสารกันได้ จำเป็นจะต้องใช้สายไฟฟ้าเชื่อมต่อระหว่างอุปกรณ์ 2 ตัวไม่ต่ำกว่า 2 เส้น แต่จะเห็นว่ามาตรฐาน RS-232-C นั้นมีขานำสัญญาณถึง 25 ขา เหตุผลก็เพราะว่าอุปกรณ์สื่อสารแบบอนุกรมตามมาตรฐานนั้น จะต้องมีวงจรไฟฟ้าที่เกี่ยวข้องกับการรับ-ส่งข้อมูลอยู่ 3 วงจรด้วยกันก็คือ วงจรส่งข้อมูล วงจรรับข้อมูล และวงจรควบคุม ดังที่จะกล่าวถึงรายละเอียดแยกเป็นประเภทดังต่อไปนี้

1. วงจรส่งข้อมูล ทำหน้าที่ส่งข้อมูลจาก DTE ไป DCE
2. วงจรรับข้อมูล ทำหน้าที่รับข้อมูลจาก DCE เพื่อส่งไปให้ DTE

จากรูปที่ 2.2 จะเห็นว่าเครื่อง PC นั้นจะส่งข้อมูลไปให้โมเด็มทางขานำสัญญาณหมายเลข 2 และรับข้อมูลกลับจากโมเด็มทางขานำสัญญาณหมายเลข 3 โดยใช้ขานำสัญญาณหมายเลข 7 เป็นกราวน์ร่วมของทั้งสองวงจร

สัญญาณทางไฟฟ้าที่ขานำสัญญาณหมายเลข 2,3 และ 7

ตามมาตรฐาน RS-232-C รูปแบบของสัญญาณทางไฟฟ้าที่ใช้แทนข้อมูลนั้นเป็นรูปแบบสัญญาณง่าย ๆ คือสัญญาณสี่เหลี่ยมที่ถูกสร้างจากไฟฟ้ากระแสตรงระดับแรงดันประมาณ 3 ถึง 25 โวลท์ แทนข้อมูล "0" และ -3 ถึง -25 โวลท์จะแทนข้อมูล "1" สำหรับช่วงแรงดัน -3 ถึง 3 โวลท์นั้นจะเป็นช่วงระดับแรงดันที่ใช้ในการแบ่งแยกระดับสถานะของสัญญาณระหว่างสถานะ "0" และสถานะ "1"

คุณสมบัติของสัญญาณแบบสี่เหลี่ยมนี้ เมื่อถูกส่งออกมาในสายไฟฟ้าแบบธรรมดา มักจะเกิดความผิดเพี้ยนได้ง่าย เนื่องจากข้อจำกัดของสายนำสัญญาณ หรืออาจจะมีสาเหตุมาจากสัญญาณรบกวนจากภายนอกอื่น ๆ ทำให้การรับส่งข้อมูลมีระยะใช้งานอยู่ในช่วงที่จำกัด โดยเฉพาะการรับส่งข้อมูลที่ความเร็วสูงนั้นคลื่นสี่เหลี่ยมจะเรียงชิดติดกัน ยิ่งทำให้มีโอกาสเกิดความผิดเพี้ยนได้ง่าย ดังนั้นมาตรฐาน RS-232-C จึงได้กำหนดความยาวของสายนำสัญญาณสำหรับการใช้งานที่ความเร็วต่าง ๆ ไว้

3. วงจรควบคุมของ RS-232-C นั้นมีอยู่ทั้งหมด 5 วงจร ซึ่งแต่ละวงจรจะมีหน้าที่ในการสร้างสัญญาณควบคุมต่าง ๆ ขึ้นมาเพื่อทำให้เครื่อง PC และโมเด็มทราบสถานะในการทำงานของกันและกันว่าอยู่ในสถานะใด โดยที่สัญญาณในวงจรควบคุมนั้นจะมีลักษณะทางกายภาพเช่นเดียวกันกับสัญญาณที่ปรากฏอยู่ในวงจรข้อมูล แต่โมเด็มส่วนใหญ่นั้นไม่ได้ใช้วงจรควบคุมครบทุกวงจร

Request to Send และ Clear to Send (ขานำสัญญาณหมายเลข 4 และ 5)

วงจร RTS และ CTS นั้นเป็นวงจรที่ใช้สำหรับการควบคุมการส่งผ่านข้อมูลระหว่างเครื่อง PC และโมเด็ม โดยโมเด็มจะส่งสัญญาณ CTS สถานะ "ON" ให้แก่ PC เมื่อโมเด็มพร้อมที่จะรับข้อมูล และส่งสัญญาณ CTS สถานะ "OFF" เมื่อโมเด็มนั้นไม่พร้อมที่จะรับข้อมูลจากเครื่อง PC ได้ ในทำนองเดียวกันเครื่อง PC ก็จะให้สัญญาณ RTS สถานะ "ON" เมื่อ PC พร้อมที่จะรับข้อมูล และส่งสัญญาณ RTS สถานะ "OFF" เมื่อ PC ไม่พร้อมที่จะรับข้อมูลจากโมเด็ม สัญญาณ RTS และ CTS นี้จะมีประโยชน์มากในการสื่อสารข้อมูลด้วยความเร็วสูง โดยสามารถป้องกันการล้นและการสูญหายของข้อมูลได้เป็นอย่างดี

Data Terminal Ready (ขานำสัญญาณหมายเลข 20)และ Data Set Ready (ขานำสัญญาณหมายเลข 6)

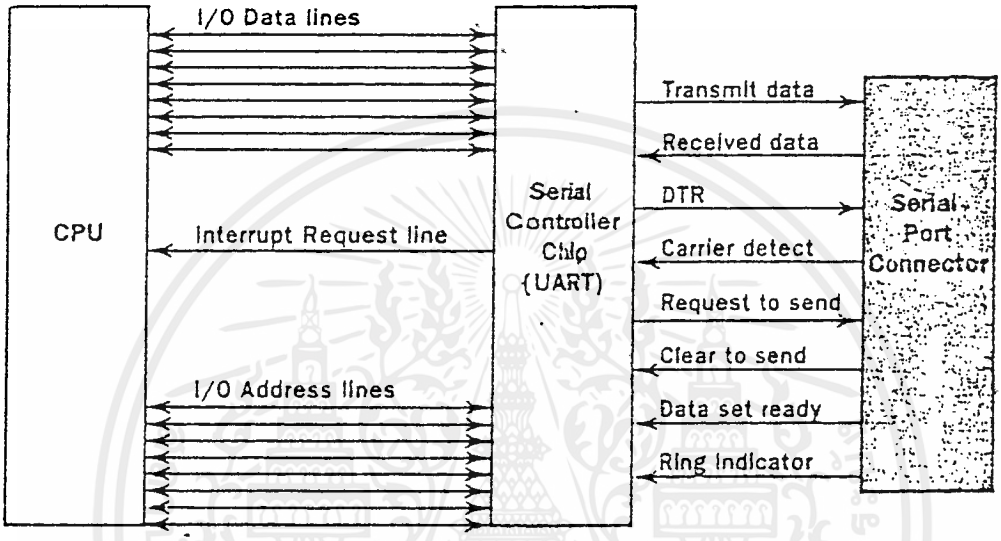
สัญญาณ DTR จะใช้เป็นสิ่งที่บอกโมเด็มให้ทราบว่าเครื่อง PC นั้นกำลังอยู่ในสถานะที่พร้อมที่จะติดต่อสื่อสารกับโมเด็มหรือไม่ และในกรณีเดียวกันสัญญาณ DSR ก็จะใช้เป็นสิ่งที่บอกให้เครื่อง PC ทราบว่าโมเด็มก็พร้อมที่จะติดต่อกับ PC หรือไม่ โดยที่สัญญาณ DSR นั้นจะอยู่ในสถานะ "ON" ก็ต่อเมื่อโมเด็มได้รับสัญญาณ DTR แล้ว

Carrier Detect (ขานำสัญญาณหมายเลข 8) และ Ring Indicator (ขานำสัญญาณหมายเลข 22)

สัญญาณ CD นี้จะใช้เป็นสิ่งที่บอก PC ให้ทราบว่าโมเด็มกำลังเชื่อมต่อกับโมเด็มเครื่องอื่น ๆ และกำลังได้รับสัญญาณพาหะจากโมเด็มปลายทาง ส่วนสัญญาณ RI นั้นจะเป็นการบอกเครื่อง PC ให้ทราบว่ามีสัญญาณกระดิ่งโทรศัพท์เรียกเข้ามาที่โมเด็ม ซึ่งโมเด็มส่วนใหญ่ในปัจจุบันก็มีวงจรที่ใช้ตอบรับโทรศัพท์โดยอัตโนมัติ (Auto - answer) อยู่ภายในตัวเองแล้ว จึงไม่จำเป็นที่จะต้องใช้งานสัญญาณ RI นี้

2.1.4 พอร์ตสื่อสารอนุกรมของเครื่อง PC

การอินเตอร์เฟส RS-232-C เข้ากับเครื่อง PC นั้นไม่ได้เฉพาะข้อกำหนดทางกายภาพของคอนเน็กเตอร์เท่านั้น ด้วยเหตุนี้เองการสื่อสารผ่านพอร์ตอนุกรมสื่อสารของ PC นั้น จำเป็นที่จะต้องมีชิพที่ใช้ในการควบคุมการสื่อสารข้อมูลโดยเฉพาะ ซึ่งชิพนี้อาจจะอยู่บนแผงวงจรหลัก หรือบนแผ่นวงจรควบคุมอุปกรณ์เอนกประสงค์ (Multi I/O Card) ก็ได้ ตัวชิพประเภทนี้จะทำหน้าที่อินเตอร์เฟสข้อมูลแบบอนุกรมเข้ากับระบบบัสข้อมูลของ PC ตามแผนภาพที่แสดงไว้ในรูปที่ 2.3



รูปที่ 2.3 แผนภาพการเชื่อมต่อของชิพควบคุมการสื่อสารเบอร์ 8250 บนเครื่อง PC ทั่วไป

เนื่องจากว่าชิพนี้จะเชื่อมต่ออยู่กับบัสของข้อมูลของ PC และวงจรสื่อสารแบบอนุกรมในแต่ละพอร์ตก็จะถูกกำหนดให้มีแอดเดรส I/O ประจำพอร์ต ดังนั้นจึงทำให้ CPU สามารถอ่านเขียนข้อมูลที่พอร์ตได้โดยตรงเมื่อ CPU ต้องการ

มาตรฐานของเครื่อง PC ทุกรุ่นจะกำหนดให้

- พอร์ต COM1 ใช้แอดเดรสที่ 3F8h
- พอร์ต COM2 ใช้แอดเดรสที่ 2F8h

ส่วนด้านสัญญาณขัดจังหวะนั้นจะกำหนดให้

- COM1 ใช้ IRQ4
- COM2 ใช้ IRQ3

2.1.5 รหัส ASCII

วิธีที่มีการกำหนดรหัสเลขฐานสองขึ้นมาใช้แทนตัวอักษรที่จะใช้งานในคอมพิวเตอร์ โดยตัวอักษรแต่ละตัวจะมีรหัสเลขฐานสองเฉพาะตัวของมัน ซึ่งมาตรฐานของรหัสเหล่านี้ที่เป็นที่นิยมใช้กันทั่วไปจะมีชื่อ

รหัสว่ารหัส ASCII (American Standard Code for Information Interchange) รหัส ASCII นั้นประกอบด้วยตัวอักษรและสัญลักษณ์ที่พิมพ์ได้ 96 ตัว และ 32 ตัวเป็นอักษรที่พิมพ์ไม่ได้ รวมเป็นจำนวนทั้งสิ้น 128 ตัวอักษรซึ่งใช้ขนาดข้อมูลจำนวน 7 บิตพอดี (27) แต่ IBM ได้กำหนดชุดอักษรรหัส ASCII เพิ่มเติมขึ้นมาอีก 128 ตัว โดยใช้ข้อมูลขนาด 8 บิต การเกิดมาตรฐานของรหัส ASCII ขึ้นมานั้นทำให้เครื่องคอมพิวเตอร์และอุปกรณ์จากผู้ผลิตต่างกัน สามารถติดต่อกันได้โดยไม่มีปัญหาที่รหัสจะไม่ตรงกัน

มาตรฐานรหัส ASCII นั้นถูกออกแบบขึ้นมาเพื่อใช้ทดแทนมาตรฐานเดิมที่มีชื่อว่า BAUDOT ซึ่งมีจำนวนเพียง 5 บิตต่อตัวอักษรซึ่งใช้งานกันอยู่กับเครื่องโทรพิมพ์ แต่เนื่องจากการใช้ขนาดข้อมูลเพียง 5 บิตนี้เองทำให้ตัวอักษรที่สามารถใช้ได้จึงมีจำนวนได้เพียง 32 ตัวเท่านั้น

จากที่กล่าวมาแล้วว่ารหัส ASCII นั้นมีตัวอักษรที่ไม่สามารถพิมพ์ได้ 32 ตัวซึ่งก็คือ ASCII หมายเลขตั้งแต่ 0 ถึง 31 เรียกว่า รหัสควบคุมนั่นเอง

2.2 การสื่อสารข้อมูลด้วยระบบโมเด็ม

เมื่อผู้ใช้สั่งให้โมเด็มเริ่มทำการโทรศัพท์ไปยังโมเด็มปลายทาง กระบวนการต่าง ๆ ระหว่างโปรแกรมสื่อสารและโมเด็มก็จะเริ่มขึ้นถ้าหากผู้ใช้เคยใช้งานมาแล้วก็จะคุ้นเคยกับเสียงต่าง ๆ ที่ดังออกมาจากลำโพงของโมเด็ม เช่น เสียง Dial Tone ตามด้วยเสียง DTMF หรือเสียง Pulse ในขณะที่ทำการหมุนเลขหมายเลขโทรศัพท์ปลายทาง เสียง Ringing Tone และเสียงสัญญาณพาหะได้ตอบกันระหว่างโมเด็มต้นทางและปลายทาง หลังจากนั้นโมเด็มก็จะเงียบลงพร้อมกับเข้าสู่การเชื่อมต่อ (Connection) ซึ่งกระบวนการดังกล่าวนี้นับได้ว่าเป็นกระบวนการทำงานปกติที่ ผู้ใช้คุ้นเคยเป็นอย่างดี

ขั้นตอน	ผู้ใช้	ซอฟต์แวร์	โมเด็มต้นทาง	โมเด็มปลายทาง
1	เลือกคำสั่ง "Dial" จากซอฟต์แวร์	เปิดสัญญาณ DRT เพื่อส่งคำสั่งหมุนหมายเลขไปยังโมเด็ม โดยใช้คำสั่ง :ATDT 555-1234	เปิดลำโพง ยกหูโทรศัพท์ รอสัญญาณให้หมุนหมายเลข	
2		รอ result codes จากโมเด็ม	รอการตอบรับจากปลายทาง ทั้งนี้ระยะเวลาการรอขึ้นกับการกำหนดค่ารีจิสเตอร์	
3				เสียงโทรศัพท์ดัง
4				ตอบรับสัญญาณ

5			รับรู้สัญญาณตอบรับและทำการส่งสัญญาณ Originate Carrier	
6			รับทราบวิธีการมอดูเลชั่นและความเร็วของแต่ละฝ่าย	รับทราบวิธีการมอดูเลชั่นและความเร็วของแต่ละฝ่าย
7			โมเด็มตกลงรับรู้โปรโตคอลการควบคุมความผิดพลาดและการบีบอัดข้อมูลของแต่ละฝ่าย	
8			ส่ง result code "Connect" ไปยัง PC ปิดลำโพงและเปิดสัญญาณ CD	
9		รับรู้ result code และสัญญาณ CD รายงานให้ผู้ใช้ทราบว่าการติดต่อได้เกิดขึ้นแล้ว		
10	เริ่มการติดต่อกับโฮสคอมพิวเตอร์	ดำเนินการสื่อสารและคอยดูสัญญาณที่ขาดหายไปจากหน้าจอสัญญาณ CD	ส่งและรับข้อมูล	ส่งและรับข้อมูล
11	การสื่อสารเสร็จสมบูรณ์เลือกคำสั่ง "Disconnect"	ปิดสัญญาณ DRT หรือส่ง +++ ตามด้วยคำสั่ง ATH		
12			วางสายโทรศัพท์	ยกเลิกสัญญาณวางสายโทรศัพท์

รูปที่ 2.4 กระบวนการการทำงานของโมเด็มต้นทางและปลายทาง ตั้งแต่ขั้นตอนที่ 1 ถึง 12

จากรูปที่ 2.4 จะแสดงผังลำดับขั้นตอนการทำงาน โดยเริ่มตั้งแต่ขั้นตอนที่สั่งให้โมเด็มหมุน หมายเลขโทรศัพท์จนกระทั่งถึงการวางหูโทรศัพท์ และจากรูปที่ 2.4 นี้เอง ผู้ใช้สามารถสังเกตได้ว่าสิ่งที่ทำงานมากที่

สุดในกระบวนการสื่อสารก็คือโมเด็ม ส่วนโปรแกรมสื่อสารจะมีหน้าที่เพียงส่งชุดคำสั่ง AT ไปให้กับโมเด็มเท่านั้น การกำหนดเวลาต่าง ๆ ในกระบวนการนี้จะขึ้นอยู่กับข้อมูลที่อยู่ในรีจิสเตอร์ S ด้วย ตัวอย่างเช่น รีจิสเตอร์ S7 จะเก็บค่าของเวลาที่โมเด็มจะรอเสียงตอบจากโมเด็มปลายทางเป็นต้น

ขั้นตอนการเชื่อมต่อระหว่างโมเด็มต้นทางและปลายทางที่ได้กล่าวไปแล้วนั้น โมเด็มทั้งคู่จำเป็นที่จะต้องมีพื้นฐานที่เหมือนกันบางประการ เช่น ความเร็ว รูปแบบของข้อมูล และโปรโตคอลถ่ายโอนไฟล์ในการรับส่งข้อมูล

2.2.1 โปรโตคอลถ่ายโอนไฟล์

โปรโตคอลถ่ายโอนไฟล์จะแบ่งไฟล์ที่จะถูกส่งออกเป็นบล็อกหรือแพคเกต แต่ละแพคเกตประกอบด้วยส่วนหัว ส่วนข้อมูล รหัสตรวจสอบข้อผิดพลาดและเครื่องหมายจบแพคเกต คอมพิวเตอร์เป้าหมายต้องส่งการตอบสนองเพื่อบอกว่าได้รับแพคเกตอย่างถูกต้องหรือไม่ โปรโตคอลที่ใช้กันทั่วไปมี 3 ตัว คือ XMODEM, YMODEM และ ZMODEM ซึ่ง YMODEM และ ZMODEM เป็นโปรโตคอลที่พัฒนามาจาก XMODEM

2.2.1.1 XMODEM

XMODEM เป็นโปรโตคอลถ่ายโอนไฟล์ที่เรียบง่ายมาก แม้ว่ามันจะมีข้อจำกัด แต่ก็เป็นโปรโตคอลที่ใช้กันกว้างขวางที่สุด ข้อมูลที่ถูกส่งโดย XMODEM จะถูกแบ่งออกเป็นบล็อก แต่ละบล็อกประกอบด้วยอักขระ Start-of-Header (01 ฐานสิบหก) หมายเลขบล็อกหนึ่งไบต์ คอมพลีเมนต์ของหนึ่งของหมายเลขบล็อก ข้อมูล 128 ไบต์ และ Checksum หนึ่งไบต์ ดังในตารางที่ 2.1

ออฟเซต	ความหมาย
0	SOH (start-of-header,ASCII 01)
1	หมายเลขบล็อก เริ่มต้นจาก 1 แต่จะกลับเป็น 0 หลังจาก FF
2	คอมพลีเมนต์ของหนึ่งของหมายเลขบล็อก (255-หมายเลขบล็อก)
3-130	ข้อมูล 128 ไบต์
131	Checksum ผลรวมของข้อมูลเท่านั้น

ตาราง 2.1 รูปแบบบล็อกของ XMODEM

ก่อนที่คอมพิวเตอร์ฝ่ายส่งจะสามารถส่งข้อมูลได้ มันต้องรับอักขระ NAK (negative acknowledgment) จากคอมพิวเตอร์ฝ่ายรับ โปรแกรมผู้รับจะส่ง NAK (15 ฐานสิบหก) หลังจากไทม์เอาท์ทุก ๆ 10 วินาทีที่ไม่ได้รับข้อมูล NAK ตัวแรกกระตุ้นให้ผู้ส่งเริ่มทำการส่ง เมื่อโปรแกรมผู้รับเริ่มทำการรับบล็อกมันจะรายงานข้อผิดพลาดเมื่อมีช่องว่าง 1 วินาทีหรือมากกว่า เกิดขึ้นระหว่างตัวอักษรในบล็อก รวมทั้ง Checksum อย่างไรก็ตามมันต้องรอให้สายว่างก่อนที่จะส่ง NAK เพื่อแจ้งข้อผิดพลาดไทม์เอาท์ 1 วินาที ไม่เพียงพอสำหรับ

การสื่อสารทางไกล จึงมักใช้เวลาคอยที่นานกว่านี้ จากนั้นผู้รับจะตรวจสอบหมายเลขบล็อกและรายงานข้อผิดพลาดถ้ามันไม่เรียงลำดับ ถ้าหมายเลขบล็อกเหมือนกับบล็อกล่าสุด แสดงถึงการส่งซ้ำซึ่งไม่ควรถูกพิจารณาเป็นข้อผิดพลาด หลังจากการรับแต่บล็อก ผู้รับจะส่ง ACK (06 ฐานสิบหก) ถ้าบล็อกที่รับมาถูกต้อง หรือ NAK ถ้าไม่ถูกต้อง ในกรณีหลังผู้ส่งจะส่งบล็อกนั้นซ้ำ หลังจากการตอบรับบล็อกนั้นแล้ว บล็อกต่อไปจึงจะถูกส่ง ในตอนจบของการส่ง ผู้ส่งจะส่ง EOT (04 ฐานสิบหก) และรอคอย ACK มันจะส่ง EOT ซ้ำถ้าไม่ได้รับ ACK

เนื่องจาก Checksum หนึ่งไบต์ไม่เพียงพอสำหรับตรวจสอบข้อผิดพลาดทั้งหมด ส่วนขยายของ XMODEM ที่เรียกว่า ตัวเลือก CRC จึงถูกคิดขึ้นโดยใช้ตัวเลขสองไบต์ ตัวเลขนี้เรียกว่า Cyclical redundancy check(CRC-16) ซึ่งสามารถตรวจสอบข้อผิดพลาดได้อย่างน้อย 99.99 % โดยปกติผู้รับต้องบอกผู้ส่งว่าใช้ตัวเลือก CRC โดยการส่งอักขระ C แทน NAK ในการร้องขอให้เริ่มต้นส่ง แต่เนื่องจากไม่ทุกเวอร์ชั่นของ XMODEM ที่มีตัวเลือก CRC ผู้รับจึงควรเปลี่ยนไปส่ง NAK ถ้าไม่มีการตอบสนองหลังจากการส่ง C รูปแบบของบล็อกที่ใช้ตัวเลือก CRC ถูกแสดงไว้ในตารางที่ 2.2

ออฟเซต	ความหมาย
0	SOH (Start-of-Leader,ASCII 01)
1	หมายเลขบล็อก เริ่มต้นจาก 1 แต่จะกลับเป็น 0 หลังจาก FF
2	คอมพลีเมนต์ของหนึ่งของหมายเลขบล็อก (255-หมายเลขบล็อก)
3-130	ข้อมูล 128 ไบต์
132	ไบต์บนของ CRC
133	ไบต์ล่างของ CRC

ตาราง 2.2 รูปแบบบล็อกของ XMODEM ที่ใช้ตัวเลือก CRC

2.2.1.2 YMODEM

YMODEM คือ XMODEM ที่มีการขยายความสามารถบางอย่าง ดังนี้

1. การตรวจสอบข้อผิดพลาดด้วย CRC -16

2. มีบล็อกขนาด 1K เป็นทางเลือก การส่ง STX (02ฐานสิบหก) ที่จุดเริ่มต้นของแต่ละบล็อก แทน SOH (01 ฐานสิบหก) เป็นสัญลักษณ์ว่าบล็อกที่ตามมายาว 1024 ไบต์ แทน 128 ไบต์บล็อกขนาด 1024 และ 128 ไบต์ สามารถถูกส่งผสมกันไปได้ในการส่งครั้งหนึ่ง

3. การยกเลิกด้วย CAN-CAN อักขระ CAN สองตัวติดกันบอกว่าการถ่ายโอนไฟล์ต้องถูกยกเลิก

4. การส่งชื่อไฟล์ บล็อกแรกมีหมายเลขศูนย์ประกอบด้วยชื่อไฟล์ปิดท้ายด้วย ASCII 0 ชื่อไฟล์ควรถูกแปลงเป็นตัวอักษรตัวเล็ก ถ้าคอมพิวเตอร์ฝ่ายส่งไม่สนับสนุนทั้งชื่อไฟล์ตัวเล็กและตัวใหญ่ ชื่อไฟล์สามารถรวมถึงชื่อเส้นทาง แยกโดย Slash (/) ซึ่งเป็นวิธีปกติในยูนิกซ์ ไม่ใช่ Backslash (\) ดังที่ใช้ในดอส อย่างไรก็ตามโดยปกติชื่อไฟล์แบบเต็มจะถูกส่งไป เนื่องจากคอมพิวเตอร์ฝ่ายรับอาจมีโครงสร้างไดเรกทอรี่ต่างกัน

5. การส่งไฟล์แบบแบตช์ (Batch-file transmission) ไฟล์หลายไฟล์สามารถถูกส่งในคราวเดียว ที่จุดสิ้นสุดของแต่ไฟล์ คอมพิวเตอร์ฝ่ายส่งจะส่ง EOT หนึ่งถึงสิบตัว (ASCII 4หรือ Ctrl-D) จนกระทั่งได้รับ ACK จากนั้นจึงสามารถส่งไฟล์อื่น หรือจบการส่ง โดยการส่งแพคเกตศูนย์ที่ชื่อไฟล์เป็น NULL

ส่วนที่เหลือของบล็อกถูกทำให้เป็น NULL ไฟล์ต่อไปถูกส่งโดยมีบล็อกชื่อไฟล์ของตัวเอง ชื่อไฟล์ที่เป็น NULL หมายถึงจบการส่งแบบแบตช์ มีข้อสังเกต คือ โฮสต์คอมพิวเตอร์และเครือข่ายจำนวนมากไม่สามารถจัดการกับบล็อกขนาด 1K ที่ต่อเนื่องกันได้

2.2.1.3 YMODEM-g

โปรโตคอล YMODEM-g คล้ายกับ YMODEM โดยมีความแตกต่างดังนี้

1. คอมพิวเตอร์ฝ่ายรับส่ง G เพื่อร้องขอให้เริ่มต้นทำการส่ง แทนการส่ง C

2. คอมพิวเตอร์ฝ่ายรับไม่ส่ง ACK สำหรับแพคเกตที่ถูกต้อง ถ้ามันตรวจพบข้อผิดพลาด มันจะยกเลิกการถ่ายโอนไฟล์ โดยการส่งอักขระ CAN ติดต่อกัน

3. คอมพิวเตอร์ฝ่ายส่งไม่รอให้บล็อกหนึ่งถูกตอบรับ ก่อนจะส่งบล็อกต่อไป

2.2.1.4 ZMODEM

ZMODEM มีการปรับปรุงเพิ่มขึ้นจาก XMODEM และ YMODEM โดยมีข้อได้เปรียบหลัก คือ สามารถถ่ายโอนข้อมูลแปดบิตบนช่องทางแบบเจ็ดบิตได้ โดยใช้การเข้ารหัสอักขระควบคุมและอักขระพิเศษตัวอื่น มันสามารถถ่ายโอนไฟล์แบบอัตโนมัติหากซอฟแวร์สื่อสารรู้จัก ZMODEM ตัวอย่างเช่น โปรแกรม ProComm Plus รู้จักการเริ่มต้นของการถ่ายโอนไฟล์ด้วย ZMODEM ถ้าร้องขอให้โฮสต์ส่งไฟล์ด้วย ZMODEM ProComm Plus จะเริ่มต้นการดาวน์โหลดโดยอัตโนมัติ เมื่อมันได้รับลำดับที่เริ่มต้นการส่งของ ZMODEM

โดยการเชื่อมต่อโมเด็มในแต่ละครั้ง อย่างน้อยที่สุด ผู้ที่ใช้โมเด็มต้นทางจำเป็นต้องทราบว่าจะให้โมเด็มโทรศัพท์ออกด้วยหมายเลขใด ควรจะตั้งเวลาในการรอเสียงตอบจากโมเด็มปลายทางนานเท่าไร ระบบโทรศัพท์ของตนนั้นใช้ระบบ Tone หรือ Pulse และในขณะเดียวกันผู้ที่ใช้โมเด็มปลายทางก็จะต้อง กำหนดให้โมเด็มรับทราบด้วยว่าจะให้โมเด็มรับสายโทรเข้าหลังจากที่มีเสียง Ringing กี่ครั้ง และจะต้องรอสัญญาณโต้ตอบนานเท่าไรเป็นต้น ซึ่งการดำเนินการทั้งหมดนี้จำสามารถควบคุมได้โดยใช้ชุดคำสั่ง AT

2.2.2 ประเภทของชุดคำสั่ง AT

โดยพื้นฐานแล้ว เราสามารถจะแบ่งชุดคำสั่ง AT ออกเป็น 2 ประเภทใหญ่ ๆ ได้ดังนี้

1. ชุดคำสั่งที่ใช้ในการปฏิบัติงาน อย่างเช่น ATD (คำสั่งให้หมุนหมายเลขโทรศัพท์) หรือ ATH (คำสั่งให้วางสายโทรศัพท์) เป็นต้น

2. ชุดคำสั่งที่ให้กำหนดค่า หรือเปลี่ยนแปลงค่าต่างๆ อย่างเช่น ATS7=90 เป็นการกำหนดค่าให้รีจีสเตอร์ S มีค่า 90 ซึ่งก็คือให้โมเด็มรอการตอบรับจากโมเด็มปลายทางเป็นเวลา 90 วินาที หลังจากนั้นหากยังไม่มีสัญญาณใด ๆ ตอบกลับมาโมเด็มก็จะวางสายทันที

ตามปกติแล้วโปรแกรมสื่อสารมักจะตั้งค่าของรีจีสเตอร์ S ต่างๆ ที่จำเป็นเอาไว้ตั้งแต่ตอนที่ผู้ใช้เรียกโปรแกรมนั้นๆ ขึ้นมา ซึ่งผู้ใช้จะสามารถควบคุมและเปลี่ยนแปลงค่าของรีจีสเตอร์ S เหล่านี้ได้ในภายหลัง

2.2.3 การออนไลน์และออฟไลน์

สภาวะออฟไลน์ จะสามารถเรียกได้อีกอย่างหนึ่งว่าสภาวะคำสั่ง (Command state) หมายถึง สภาวะที่ผู้ใช้สามารถจะส่งคำสั่งต่างๆ ไปยังโมเด็มได้ หรือ จะพูดอีกนัยหนึ่งก็คือ สภาวะที่โมเด็มจะแปลความหมายของข้อมูลที่ได้รับมาจาก PC ให้เป็นคำสั่งเท่านั้น ซึ่งสภาวะนี้จะไม่ได้รับ-ส่งข้อมูลกับโมเด็มปลายทาง แต่จะสื่อสารกันกับ PC เท่านั้น

สภาวะออนไลน์ หมายถึง สภาวะที่โมเด็มได้เชื่อมต่อกับโมเด็มปลายทางเป็นที่เรียบร้อยแล้ว ข้อมูลที่ส่งออกมาจาก PC ก็จะผ่านจากโมเด็มต้นทางไปยังโมเด็มปลายทางเสมอ

ถ้าหากผู้ใช้ต้องการที่จะส่งคำสั่งให้กับโมเด็ม หรือต้องการจะให้โมเด็มกลับมาอยู่ในสภาวะออฟไลน์ เพื่อรับคำสั่งจาก PC ก็สามารถทำได้โดยมีวิธีการอยู่ 2 วิธีคือ วิธีที่หนึ่ง ให้โมเด็มวางสายแล้วกลับมาอยู่ในสภาวะออนไลน์ใหม่ และวิธีที่สองคือ ส่งชุดอักขระ Escape Sequence เข้าไปยังโมเด็ม ในขณะที่ออนไลน์ ซึ่งวิธีหลังนี้จะมีความเหมาะสมกว่า เพราะการเชื่อมต่อระหว่างโมเด็มจะยังคงดำเนินไปอยู่ และหลังจากที่ได้ส่งคำสั่งต่าง ๆ ให้กับโมเด็มเป็นที่เรียบร้อยแล้ว ก็จะสามารถกลับเข้าไปอยู่ในสภาวะออนไลน์ได้เช่นเดิม วิธีการส่งชุดอักขระ Escape sequence สำหรับโมเด็มที่เข้ากันได้กับโมเด็มของ Hayes คือให้รอ 1 วินาที (เรียกว่า Guard Time) แล้วกดปุ่ม + ติดต่อกัน 3 ครั้ง (+++) การที่ต้องรอ 1 วินาทีก่อนที่จะกดปุ่มเครื่องหมาย + เป็นสิ่งที่จำเป็นเนื่องจากว่าโมเด็มจะได้รู้ว่าอักษร + นั้นเป็นส่วนของอักขระ Escape Sequence ไม่ใช่ส่วนของข้อมูลที่จะต้องส่งไปยังโมเด็มปลายทาง

2.2.4 รูปแบบของชุดคำสั่ง AT

การใช้ชุดคำสั่ง AT จำเป็นต้องขึ้นต้นด้วยตัวอักษร AT เสมอ และจะต้องจบลงด้วยการกด Enter หรือ Carriage return ยกเว้นเฉพาะคำสั่ง A/ (ไม่ต้องกดปุ่ม Enter) ซึ่งหมายถึงให้โมเด็มกลับไปทำคำสั่งล่าสุดซ้ำอีกครั้งหนึ่ง การที่ Hayes ได้กำหนดให้คำสั่งต่างๆ ขึ้นต้นด้วยอักษร AT ก็เพราะต้องการให้โมเด็มรับรู้ถึงความสามารถและรูปแบบของอักษรคำสั่ง ที่ถูกส่งออกมาจากพอร์ตสื่อสารอนุกรมของ PC ซึ่งผู้อ่านอย่าสับสนระหว่างความเร็วของพอร์ตสื่อสารอนุกรม (ความเร็วของ DTE) และความเร็วของโมเด็ม (ความเร็วของ DCE) ในกรณีนี้จะหมายถึงความเร็วของ DTE

ตัวอย่างการใช้ชุดคำสั่ง AT ก็ได้แก่ คำสั่ง ATH หมายถึงคำสั่งที่ให้โมเด็มวางสายโทรศัพท์เป็นต้น แต่บางคำสั่งอาจจะต้องการพารามิเตอร์เพิ่มเติม อย่างเช่น คำสั่ง ATDT7113022 หมายถึงให้โมเด็มหมุนหมายเลข 7113022 โดยหมุนระบบ Tone (T) แต่ถ้าต้องการให้โมเด็มหมุนแบบระบบ Pulse ก็ต้องใช้คำสั่ง ATDP7113022 เป็นต้น

2.2.5 การสนองคำสั่งของโมเด็ม

เมื่อผู้ใช้ส่งคำสั่งต่าง ๆ ไปให้กับโมเด็มแล้ว โมเด็มก็จะสนองคำสั่งโดยจะส่งข้อความที่เรียกว่า Result code กลับมายังเครื่อง PC และจะปรากฏขึ้นบนจอภาพขณะที่รันโปรแกรมสื่อสารไว้ Result code เหล่านี้จะเป็นภาษาอังกฤษ อย่างเช่น OK , ERROR , CONNECT 2400 หรืออาจจะเป็นตัวเลขอื่นๆ ซึ่งคำสั่งที่เกี่ยวข้องกับการส่ง Result code ก็จะมีดังนี้

ATV0 เป็นคำสั่งที่กำหนดให้โมเด็ม Result code ที่จำเป็นออกมาเท่านั้น (Non-verbose)

ATV1 เป็นคำสั่งที่กำหนดให้โมเด็มส่ง Result code ออกมาทุกๆ ครั้ง หลังจากที่ได้รับคำสั่งต่างๆ(Verbose)

ATQ1 เป็นคำสั่งที่กำหนดให้โมเด็มไม่ส่ง Result code ออกมา (Quiet mode)

ATQ0 เป็นคำสั่งที่ใช้ Toggle ให้โมเด็มสามารถกลับไปส่ง Result code ได้ดังเดิม

นอกจากนี้ยังมีชุดคำสั่งที่เกี่ยวข้องซึ่งจะได้กล่าวถึงในหัวข้อสรุปชุดคำสั่ง AT ในภาคผนวก ต่อไป

2.2.6 ชุดคำสั่งเพิ่มเติม

ชุดคำสั่งรุ่นแรกๆ ของ Hayes นั้นจะใช้ตัวอักษรตั้งแต่ A จนถึง Z เช่น ATA จนถึง ATZ ซึ่งก็หมายความว่า คำสั่งจะมีให้ใช้ได้ไม่เกิน 26 คำสั่งเท่านั้น แต่โมเด็มในปัจจุบันก็ได้มีความสามารถต่าง ๆ ที่พิเศษเพิ่มเข้ามา ดังนั้น Hayes จึงได้กำหนดชุดคำสั่งเพิ่มเติมขึ้นมา โดยใช้ตัวอักษร & เป็นตัวเข้ามาประกอบ ดังตัวอย่าง AT&F หมายถึง ให้โมเด็มเรียกค่า Default Setting ขึ้นมาใช้งาน ซึ่งโมเด็มของผู้ผลิตรายอื่นๆ ก็อาจจะเพิ่มเติมชุดคำสั่งที่ใช้ตัวอักษรอื่นๆ ประกอบ อย่างเช่น % หรือ * เป็นต้น

2.2.7 การตั้งค่าในรีจีสเตอร์ S

ดังที่กล่าวมาแล้วว่ารีจีสเตอร์ S มีหน้าที่เก็บข้อมูลที่ใช้ควบคุมสถานะการทำงานของโมเด็มเอาไว้ ซึ่งผู้ใช้สามารถตั้งค่า หรือตรวจดูค่าที่เก็บเอาไว้ในรีจีสเตอร์ S ต่างๆ ได้โดยใช้คำสั่งที่ขึ้นต้นด้วย ATS ในกรณีที่ต้องการตั้งค่าให้แก่รีจีสเตอร์ S ผู้ใช้จะต้องใส่ตัวเลขที่ระบุว่าเป็นรีจีสเตอร์ S ตัวที่เท่าใดตามด้วยเครื่องหมาย = และค่าที่ต้องการใส่ลงไป ดังตัวอย่างเช่น ถ้าต้องการให้รีจีสเตอร์ S0 มีค่าเท่ากับ 9 ก็จะต้องใช้คำสั่งดังนี้

ATS0=9

ส่วนในกรณีที่ต้องการตรวจดูค่าของรีจีสเตอร์ S ต่างๆ ก็จะต้องพิมพ์คำสั่งที่ขึ้นต้นด้วย ATS ตามด้วยตัวเลขที่ระบุว่าเป็นรีจีสเตอร์ S ตัวที่เท่าไร และเครื่องหมาย ? หลังจากที่กด Enter หรือ Carriage return แล้ว โมเด็มก็จะส่งค่าที่คุณต้องการทราบในรีจีสเตอร์ S นั้นๆ ออกมาพร้อมกับคำว่า "OK" ดังตัวอย่างเช่นถ้าต้องการตรวจดูค่าของรีจีสเตอร์ S11 จะต้องใช้คำสั่งดังนี้

ATS11?

จากนั้นหน้าจอก็จะปรากฏข้อความ

095

OK

ผลลัพธ์ที่ออกมาเป็น 095 ก็คือค่าที่เก็บอยู่ในรีจีสเตอร์ S11 นั่นเอง

2.2.8 การส่งหลายคำสั่งในครั้งเดียว

ในกรณีที่ต้องการส่งคำสั่งไปให้โมเด็มมากกว่าหนึ่งคำสั่ง ผู้ใช้อาจจะส่งคำสั่งทั้งหมดไปในครั้งเดียวได้ ซึ่งโมเด็มจะสามารถรับส่งได้ไม่เกิน 40 ตัวอักษร ทั้งนี้จะนับตัวอักษร AT และ Carriage Return ด้วย ผู้ใช้อาจจะพิมพ์คำสั่งทั้งหมดให้ตัวอักษรอยู่ติดกัน หรือเว้นช่องว่างระหว่างแต่ละคำสั่งได้ ดังตัวอย่าง

ATS7=90V1X4DT13055551234 หรือ AT S7=90 V1 X4 DT 1-305-555-1234

ซึ่ง S7=90 หมายถึงให้ใส่ค่า 90 ลงไปในรีจิสเตอร์ S7

V1 หมายถึงให้โมเด็มส่ง Result Code ออกมาได้บางคำ อย่างเช่น OK , ERROR , CONNECT2400 เป็นต้น

DT1-305-555-1234 หมายถึงให้โมเด็มทำการหมุนหมายเลขโทรศัพท์ 1-305-555-1234 โดยหมุนแบบระบบ Tone

2.2.9 การเก็บค่าต่างๆ ที่เซตเอาไว้ใน Profile

จะเห็นได้ว่า ผู้ใช้สามารถตั้งค่ารีจิสเตอร์ S ต่างๆ และใช้คำสั่ง AT เพื่อควบคุมการทำงานของโมเด็มได้ในลักษณะต่างๆ ที่ต้องการ ซึ่ง Hayes ก็ได้ออกแบบหน่วยความจำภายในโมเด็มขึ้นมา เพื่อใช้สำหรับเก็บค่าต่างๆ ที่ตั้งไว้ และมีให้เลือกใช้ได้มากถึง 4 ชุดหน่วยความจำ โดยที่หน่วยความจำชุดแรก เรียกว่า แอกทีฟคอนฟิกูเรชั่น (Active Configuration) ใช้เก็บข้อมูลการตั้งค่าต่างๆ ที่เซตเอาไว้ใช้งานในปัจจุบัน เมื่อผู้ใช้มีการเปลี่ยนแปลงค่าของรีจีสเตอร์ S ต่างๆ และได้ตั้งคำสั่ง AT บางคำสั่ง ข้อมูลการตั้งค่าต่างๆ เหล่านี้ก็จะถูกนำไปเก็บเอาไว้ในแอกทีฟคอนฟิกูเรชั่น และ หลังจากปิดเครื่องไปแล้ว ข้อมูลเหล่านี้ก็จะสูญหายไป นอกจากนี้หน่วยความจำที่เหลือจะมีอยู่ 3 ชุด โดย 2 ชุดจะเป็น Nonvolatile RAM (NVRAM) และอีก 1 ชุดที่เหลือเป็น Read Only Memory (ROM) ซึ่งจะใช้เก็บข้อมูลที่เป็นค่า Default Setting ซึ่งจะอ่านออกมาได้อย่างเดียวไม่สามารถเขียนข้อมูลลงไปได้ ผู้ใช้สามารถจะเรียกเอาค่า Factory Setting ออกมาได้ โดยคำสั่ง AT&F และนำไปเก็บไว้ที่แอกทีฟคอนฟิกูเรชั่น หลังจากที่ได้ติดตั้งและเปิดเครื่องโมเด็มเป็นครั้งแรก โมเด็มก็จะถูกกำหนดให้เรียกเอา Factory Setting ออกมาใช้งานเสมอ ซึ่งจะมีประโยชน์มากในกรณีที่ผู้ใช้อาจจะประสบกับปัญหาการเซตค่าต่างๆ กลับขึ้นมาใช้งานใหม่ได้ และข้อมูลที่เป็นค่าดีฟอลต์เหล่านี้จะมีอยู่ในคู่มือการใช้โมเด็มที่มาพร้อมกับโมเด็ม ส่วนหน่วยความจำที่เป็น NVRAM ทั้งสองชุดจะถูกเรียกว่า User Profiles มีไว้สำหรับให้ผู้ใช้สามารถสำรองเก็บค่าต่างๆ ที่เซตเอาไว้ได้ตามที่ต้องการ ซึ่งหน่วยความจำแบบ NVRAM นี้จะไม่สูญหายเมื่อปิดเครื่อง และทุกครั้งหลังจากที่ได้เปิดเครื่องแล้ว โมเด็มก็จะนำข้อมูลจากโพรไฟล์ที่ 0 มาเก็บไว้ที่แอกทีฟคอนฟิกูเรชั่น โดยอัตโนมัติ แต่ถ้าต้องการให้โมเด็มเรียกโพรไฟล์ที่ 1 ขึ้นมาทุกๆ ครั้งที่เปิดเครื่อง ก็จะต้องใช้คำสั่ง AT&Y1 ดังจะกล่าวในหัวข้อสรุปชุดคำสั่ง AT ในภาคผนวกต่อไป

2.2.9.1 การสร้างและการเรียกใช้ User Profile

การสร้าง User Profile จะมีขั้นตอนดังต่อไปนี้

1.ให้โปรแกรมสื่อสารอยู่ในสภาวะคำสั่ง (หรือเรียกว่า Local Mode)

2.ใช้คำสั่ง AT เพื่อตั้งค่าพารามิเตอร์ต่างๆ และตั้งค่ารีจิสเตอร์ S ที่ต้องการ

3. ใช้คำสั่ง AT&W0 หรือ AT&W1 เพื่อเก็บข้อมูลการเซตต่างๆ เอาไว้ในโพรไฟล์ที่ 0 หรือ 1

หลังจากนั้นผู้ใช้สามารถเรียกใช้ User Profiles ต่างๆ ที่ได้จัดเก็บเอาไว้ ขึ้นมาใช้ได้โดยคำสั่ง ATZ0 หรือ ATZ1 เพื่อรีเซตโมเด็ม และเรียกข้อมูลการเซตต่างๆ ในโพรไฟล์ที่ 0 หรือโพรไฟล์ที่ 1 ขึ้นมาใหม่อีกครั้ง โมเด็มก็จะดึงเอาโพรไฟล์ที่ 0 ขึ้นมาเก็บเอาไว้ที่แอกทีฟคอนฟิกูเรชัน โดยอัตโนมัติ ซึ่งผู้ใช้สามารถกำหนดให้โมเด็มอ่านโพรไฟล์ที่ 1 ขึ้นมาแทน โดยใช้คำสั่ง AT&Y1 ก่อนที่จะปิดเครื่องได้

2.2.9.2 การเรียกดูข้อมูลที่ได้ตั้งไว้ทั้งหมด

หลังจากที่ผู้ใช้ได้ตั้งค่าพารามิเตอร์ต่างๆ เอาไว้เรียบร้อยแล้ว ผู้ใช้จะสามารถเรียกดูข้อมูลที่ได้ตั้งเอาไว้ทั้งหมดได้โดยใช้คำสั่ง AT&V ดังรูปที่ 2.5

AT&V
ACTIVE PROFILE :
B16 B1 B41 B60 E1 L2 M1 N1 P Q0 V1 W0 X4 Y0 &A0 &C1 &D2 &G0 &J0 &K3 &O5 &R0 &S0 &T4 &U0 &X0 &Y0
S00:000 S01.000 S02.043 S03.013 S04.010 S05.008 S06.002 S07.050 S08.002 S09.006 S10.014 S11.095 S12.050 S18.000 S25.005 S26.001 S36.007 S37.000 S38.020 S44.003 S46.002 S48.007 S49.008 S50.016 S97.030
STORED PROFILE 0 :
B16 B1 B41 B60 E1 L2 M1 N1 P Q0 V1 W0 X4 Y0 &A0 &C1 &D2 &G0 &J0 &K3 &O5 &R0 &S0 &T4 &U0 &X0
S00:000 S02.043 S06.002 S07.050 S08.002 S09.006 S10.014 S11.095 S12.050 S18.000 S25.005 S26.001 S36.007 S37.000 S38.020 S44.003 S46.002 S48.007 S49.008 S50.016 S97.030
STORED PROFILE 1 :
B16 B1 B41 B60 E1 L2 M1 N1 P Q0 V1 W0 X4 Y0 &A0 &C1 &D2 &G0 &J0 &K3 &O5 &R0 &S0 &T4 &U0 &X0 &Y0
S00:000 S02.043 S06.002 S07.050 S08.002 S09.006 S10.014 S11.095 S12.050 S18.000 S25.005 S26.001 S36.007 S37.000 S38.020 S44.003 S46.002 S48.007 S49.008 S50.016 S97.030

TELEPHONE NUMBERS :
0
2.
OK

รูป 2.5 Profile ซึ่งแสดงข้อมูลที่เก็บไว้ เรียกใช้โดยคำสั่ง AT&V

2.2.9.3 การเก็บหมายเลขโทรศัพท์ไว้ในโมเด็ม

จากรูปที่ 2.5 ผู้อ่านสามารถเห็นคำว่า Telephone Numbers บริเวณบรรทัดล่างได้ซึ่งหมายความว่าผู้ใช้สามารถเก็บหมายเลขโทรศัพท์เอาไว้ใน NVRAM ได้ 4 หมายเลข รวมทั้งสามารถเก็บตัวอักษรที่จำเป็นในการหมุนโทรศัพท์ได้ดังเช่น T(Tone), P(Pulse), W(Wait) โดยใช้คำสั่ง

AT&Zn=xxxxxx

และสามารถสั่งให้โมเด็มหมุนโทรศัพท์ตามหมายเลขที่เก็บไว้ได้โดยใช้คำสั่ง

ATS=n

ตัวอย่างเช่น

ถ้าต้องการเก็บหมายเลข 3197707 เอาไว้ในที่เก็บตำแหน่งที่ 2 และเมื่อเรียกใช้หมายเลขนี้จะกำหนดให้โมเด็มหมุนแบบ Tone ให้ใช้คำสั่งดังนี้

AT&Z2=T3197707

และเมื่อต้องการให้โมเด็มหมุนหมายเลขโทรศัพท์ที่เก็บเอาไว้นี้ ให้ใช้คำสั่ง

ATDS=2

โดยทั่วไปแล้ว โปรแกรมสื่อสารก็มักจะมีฟังก์ชันการเก็บและหมุนหมายเลขไว้ในตัวเช่นกัน ซึ่งมักจะเรียกว่า โฟนบุ๊ก (Phone Book) หรือ Dialing Directory แต่สำหรับในกรณีการเก็บหมายเลขโทรศัพท์เอาไว้ใน NVRAM เช่นนี้ มักจะใช้เกี่ยวกับกระบวนการที่เรียกว่า Dial Backup Number ซึ่งใช้ในระบบสื่อสารที่ต่อข้างจะอัตโนมัติและต้องการให้โมเด็มติดต่อสื่อสารอยู่ตลอดเวลา กระบวนการ Dial Backup Number นี้จะมีขั้นตอนการทำงานคือ ขณะที่โมเด็มต้นทางติดต่อสื่อสารกับโมเด็มปลายทางแล้วเกิดปัญหาสายขาด หรือสายหลุดไม่สารถจะดำเนินการสื่อสารได้ต่อไปได้ โมเด็มต้นทางก็จะสามารถหมุนเบอร์โทรศัพท์ที่เก็บเอาไว้ใน NVRAM ได้อัตโนมัติ เพื่อทำการโทรศัพท์กลับไปเชื่อมต่อกับปลายทางอีกครั้งหนึ่ง หรือโทรศัพท์ไปยังโมเด็มตัวอื่นๆ ตามที่ผู้ใช้กำหนดเอาไว้

2.2.10 การเชื่อมต่อกับ RS-232-C

วงจร RS-232-C จะประกอบไปด้วย วงจรรับ (Receive Data:RD) และวงจรส่ง (Transmit Data:TD) แยกอิสระไม่ขึ้นต่อกัน ซึ่งแต่ละวงจรก็จะมีลักษณะการเคลื่อนย้ายข้อมูลแบบทางเดียว คือยอมให้ข้อมูลเข้าหรือออกเท่านั้น ดังนั้นจึงทำให้การเชื่อมต่อระหว่าง PC และโมเด็มมีลักษณะแบบ 2 ทางคือ ทั้งรับข้อมูลเข้าและส่งข้อมูลออกไปพร้อมๆกันได้ นอกจากนี้โมเด็มก็ยังมีวงจรต่างๆ ที่ใช้สำหรับเชื่อมต่อกับ PC และเป็นวงจรที่อาจจำเป็นต่อโปรแกรมสื่อสารบางชนิด ซึ่งต่อไปนี้จะได้กล่าวถึงวงจรที่สำคัญและมีผลต่อการทำงานของ PC หรือมีผลต่อโปรแกรมสื่อสารรวมทั้งจะได้กล่าวถึงการควบคุมวงจรดังกล่าวด้วย

2.2.10.1 Data Terminal Ready

สัญญาณ Data Terminal Ready(DTR) คือ สัญญาณไฟฟ้าที่ขา 20 ของคอนเน็กเตอร์แบบ RS-232-C ซึ่งเป็นสัญญาณที่จะถูกส่งออกมาจาก PC ไปยังโมเด็ม เพื่อให้โมเด็มรับทราบว่าในขณะนั้น PC พร้อมที่จะติดต่อกับโมเด็มแล้ว ถ้าสัญญาณ DTR มีค่าเป็นศูนย์ ในระหว่างที่โมเด็มกำลังติดต่อสื่อสารกับโมเด็มปลาย

ทางอยู่ ก็จะทำให้โมเด็มหยุดการสื่อสารและวางหูโทรศัพท์ทันที ดังนั้นถ้าผู้ใช้ต้องการจะหยุดการสื่อสาร ก็อาจจะกำหนดให้ โปรแกรมสื่อสารส่งสัญญาณ DTR ในลักษณะนี้ได้ ซึ่งจะสะดวกและเร็วกว่าวิธีการใช้ Escape Sequence(+++) ตามด้วยคำสั่ง ATH มาก โมเด็มของ Hayes ในยุคต้นๆ นั้น มักจะมีสวิตช์พิเศษที่มีไว้เพื่อใช้ควบคุมการตอบสนองต่อสัญญาณ DTR ของโมเด็ม ซึ่งถ้าสวิตช์อยู่ในตำแหน่ง ON โมเด็มก็จะทำงานโดยขึ้นอยู่กับสัญญาณ DTR แต่ถ้าสวิตช์อยู่ในตำแหน่ง OFF โมเด็มก็จะไม่สนใจสัญญาณ DTR แต่ในปัจจุบันผู้ใช้จะสามารถเลือกการตอบสนองต่อสัญญาณ DTR ของโมเด็มได้โดยตรงโดยใช้คำสั่ง AT&Dn

2.2.10.2 Carrier Detect

สัญญาณ Carrier Detect(CD) คือสัญญาณไฟฟ้าที่ขา 8 ของคอนเน็กเตอร์แบบ RS-232-C ซึ่งเป็นสัญญาณที่ถูกส่งออกมาจากโมเด็มไปยัง PC มีหน้าที่ทำให้ PC รับทราบว่าในขณะนั้นโมเด็มได้เชื่อมต่อกับโมเด็มปลายทางหรือยัง ถ้าสามารถเชื่อมต่อได้เรียบร้อยแล้ว โมเด็มก็จะทำให้สัญญาณ CD มีค่าเป็น 1 พร้อมๆ กับส่งข้อความ Result Code คำว่า"CONNECT" ออกมา ซึ่งโปรแกรมสื่อสารก็จะรับทราบสถานะของโมเด็มว่ากำลังออนไลน์ หรือ ออฟไลน์ ได้จากค่าของสัญญาณ CD และเช่นเดียวกับกับสัญญาณ DTR โมเด็มของ Hayes ในอดีต ก็จะมีสวิตช์พิเศษที่มีหน้าที่ควบคุมการตอบสนองต่อสัญญาณ CD เช่นกัน แต่ในปัจจุบัน ผู้ใช้สามารถเลือกการตอบสนองต่อสัญญาณ CD ของโมเด็มได้โดยใช้คำสั่ง AT&Cn

2.2.11 การควบคุมการไหล

การควบคุมการไหล (Flow Control) ของข้อมูลเปรียบเสมือนได้กับวาล์วปิด-เปิดทางเดินของข้อมูลทั้งทางด้านขาเข้าและขาออก มีหน้าที่คอยควบคุมจังหวะการไหลเข้าและออกของข้อมูลระหว่าง PC และโมเด็ม การที่จำเป็นต้องมีการควบคุมการไหล ก็เพราะว่าโมเด็มจะไม่สามารถรับส่งข้อมูลได้เร็วเท่ากับ PC หรือ PC อาจจะต้องไปทำงานอย่างอื่น จึงต้องสั่งให้โมเด็มหยุดการส่งข้อมูลให้แก่ PC ชั่วคราว การควบคุมการไหลของข้อมูลจะมีอยู่ 2 แบบคือ แบบซอฟต์แวร์ และแบบฮาร์ดแวร์ นอกจากนี้โมเด็มที่มีโปรโตคอลการควบคุมความผิดพลาด(error control) อย่างเช่น MNP และ V.42 จะสามารถส่งสัญญาณการควบคุมการไหลของข้อมูล ไปให้กับโมเด็มปลายทางที่ไม่มีการควบคุมการไหลดังกล่าวได้เช่นกัน

2.2.11.1 การควบคุมการไหลแบบฮาร์ดแวร์ (Hardware Flow Control)

วงจรควบคุมการไหลในโมเด็มนั้น จะมีอยู่ 2 วงจรคือ RTS(Request to Send) และ CTS(Clear to Send) ซึ่งจะเป็นตัวอนุญาตให้โมเด็มหรือ PC ส่งข้อมูลให้แก่กัน ซึ่งข้อมูลที่จะรับหรือส่งนั้นจะถูกหยุดเอาไว้ชั่วคราว จนกว่าสัญญาณดังกล่าวจะอยู่ในสถานะ ON ซึ่งในกรณีที่โมเด็มไม่สามารถที่จะรับข้อมูลได้จาก PC โมเด็มก็จะควบคุมให้สัญญาณ RTS อยู่ในสถานะ OFF และโปรแกรมสื่อสารก็จะรับรู้ความเปลี่ยนแปลงของสัญญาณ RTS แล้วก็จะหยุดการส่งข้อมูลให้กับโมเด็มไว้ชั่วคราว จนกระทั่ง RTS กลับมาอยู่ในสถานะ ON อีกครั้งหนึ่ง และก็เช่นเดียวกัน ถ้าโปรแกรมสื่อสารต้องการที่จะหยุดการรับส่งข้อมูลจากโมเด็มเอาไว้ชั่วคราว ก็จะไปควบคุมให้สัญญาณ CTS อยู่ในสภาวะ OFF และโมเด็มก็จะหยุดการส่งข้อมูลให้แก่ PC ชั่วคราวจนกว่า CTS จะกลับมาอยู่ในสถานะ ON อีกครั้งหนึ่ง ซึ่งการรับส่งข้อมูลก็จะดำเนินไปเช่นเดิม

2.2.11.2 การควบคุมการไหลแบบซอฟต์แวร์ (Software Flow Control)

การควบคุมการไหลแบบซอฟต์แวร์ (Software flow Control) มีลักษณะการทำงานคล้ายกับการควบคุมการไหลแบบฮาร์ดแวร์ แต่แทนที่จะใช้วงจร RTS/CTS การควบคุมการไหลแบบซอฟต์แวร์จะใช้ตัวอักขระที่เรียกว่า XON และ XOFF ซึ่งจะตรงกับอักขระ CTRL-Q และ CTRL-S ที่พิมพ์มาจากคีย์บอร์ดตามลำดับ เทคนิคดังกล่าว ได้พัฒนามาจากการใช้งานโทรพิมพ์ในอดีต เมื่อ PC ต้องการให้โมเด็มหยุดส่งข้อมูลชั่วคราว PC ก็จะส่งอักขระ XOFF ไปยังโมเด็ม และหลังจากที่โมเด็มได้รับอักขระ XOFF แล้ว ก็จะหยุดส่งข้อมูลให้กับ PC ไว้ชั่วคราวจนกว่า PC จะพร้อมที่จะรับข้อมูลใหม่ เครื่อง PC จึงจะส่ง XON ไปยังโมเด็มและข้อมูลก็จะมีการรับส่งกันอย่างเดิม และเช่นเดียวกันโมเด็มก็สามารถส่ง XON/XOFF ไปยัง PC เพื่อที่จะควบคุมจังหวะการหยุดและไหลของข้อมูลจาก PC มายังโมเด็มได้

2.3 ระบบมัลติมีเดีย

การให้คำจำกัดความของคำวา มัลติมีเดีย ในขณะนี้ยังไม่มีที่สิ้นสุด ทั้งนี้เนื่องมาจากการพัฒนาเทคโนโลยีต่าง ๆยังไม่มีที่สิ้นสุด จึงขอนิยามอย่างกว้าง ๆว่า คือ การรวบรวมสื่อต่าง ๆที่มีองค์ประกอบสำคัญใหญ่ ๆ 3 อย่าง คือ ภาพ เสียง การโต้ตอบซึ่งกันและกัน

2.3.1 สื่อต่าง ๆที่เป็นองค์ประกอบของระบบมัลติมีเดีย

2.3.1.1 ภาพ

เป็นสื่อที่มนุษย์เรานิยมใช้มาตั้งแต่อดีตและนิยมใช้มาถึงปัจจุบัน ความจริงแล้วสื่อที่เป็นเสียงนั้น เป็นสื่อแรกที่มนุษย์รู้จักและใช้ แต่เสียงนั้นมีข้อจำกัดอยู่ที่ระยะทางและภาษา การสื่อสารภายในเผ่าเดียวกันก็ไม่เป็นปัญหานัก แต่หากจะต่างเผ่าออกไป การติดต่อสื่อสารระหว่างกันจะเริ่มยุ่งยาก หากสังเกตร่องรอยที่ถูกบันทึกไว้โดยมนุษย์ในยุคแรก ๆจะเป็นภาพวาดมา

ภาพในระบบมัลติมีเดียแบ่งออกเป็นประเภทใหญ่ ๆได้ดังนี้ คือ

1.ตัวอักษร (Text)

2.รูปภาพ (Image)

และสามารถแบ่งออกตามลักษณะของภาพเป็นประเภทใหญ่ ๆได้อีกคือ ภาพนิ่ง และภาพเคลื่อนไหว

ถ้าหากพิจารณาถึงที่มาของภาพแล้ว สามารถจำแนกออกเป็น 2 ลักษณะใหญ่ ๆคือ

1. ภาพจากการจำลองแบบ (Imitated Image) ได้แก่ ภาพที่ถ่ายแบบมาจากของจริง โดยอาศัยเครื่องมือช่วยในการจำลองแบบ เช่น กล้องถ่ายภาพนิ่ง กล้องวีดีโอ เครื่องอ่านภาพ (Scanner) เป็นต้น

2. ภาพจากการสร้าง (Creative Image) เป็นภาพที่สร้างขึ้นจากจินตนาการของมนุษย์ เช่น ภาพวาด ภาพการ์ตูน เป็นต้น

เหตุที่เราไม่แยกตัวอักษรกับภาพออกจากกันโดยเฉพาะถ้ามองในส่วนของการใช้งานคอมพิวเตอร์ในปัจจุบัน ตัวอักษรจัดเป็นภาพจากการสร้าง โดยที่การสร้างแบบตัวอักษร (Font) จะมีลักษณะการสร้างในเชิงรูปภาพ

2.3.1.2 เสียง

เสียงนับได้ว่าเป็นสื่อแรกที่มนุษย์ใช้ และเสียงหรือภาษาพูดก็เป็นสื่อที่มีความสำคัญไม่น้อยไปกว่าภาษาเขียน การผลิตภาษาเชิงภาพจะมีขั้นตอนยุ่งยากกว่าภาษาเสียง ซึ่งอาศัยหูเป็นสื่อกลางเพียงอย่างเดียวก็ใช้งานได้แล้ว อย่างไรก็ดี เสียงก็มีข้อจำกัดที่ลดลงไปมากกว่าในอดีต เนื่องจากการพัฒนาสื่อทางเสียงได้รับการพัฒนาไปเร็วกว่าสื่อทางภาพ และที่สำคัญการพัฒนาสื่อทางเสียง จะไม่ร้ายแรงเท่ากับความผิดพลาดในเชิงภาพ ทั้งนี้ เพราะความผิดพลาดทางเสียงจะมีผลแต่เพียง ทำให้ความชัดเจนของเสียงลดลงหรือเพี้ยนไปจากเดิม เช่น ทุ้มเกินไป แหลมเกินไป เป็นต้น แต่ถ้าเป็นความผิดพลาดในลักษณะของสื่อทางภาพแล้ว อาจถึงขั้นทำให้ใช้ไม่ได้เลยทีเดียว ระบบมัลติมีเดียจึงเป็นการประสานกันระหว่างสื่อทางภาพกับสื่อทางเสียงนั่นเอง

2.3.1.3 การโต้ตอบซึ่งกันและกัน

คำว่าโต้ ตอบกันได้ (Interactive) หมายถึง แสดงผลการทำงานว่าถูกต้องหรือไม่ ใช้ได้หรือไม่ได้ในขณะนั้น ระบบมัลติมีเดียต้องมีการโต้ตอบกันได้จึงจะสมบูรณ์ จากลักษณะนี้เองที่ทำให้ขอบเขตของมัลติมีเดียกว้างขวางออกไป จำแนกได้ดังนี้

1. <u>การตอบโต้กันได้ในลักษณะตัวเลือก</u> มีการจำลองสถานการณ์ (Simulation) จากปัญหาหนึ่งไวหลายรูปแบบ โดยพิจารณาจากเงื่อนไขที่แตกต่างกัน หากสามารถจำลองสถานการณ์ได้มาก และซับซ้อนครอบคลุมได้มากเท่าไร จะทำให้ระบบมัลติมีเดียนั้นสมบูรณ์มากขึ้นเท่านั้น

2. <u>การตอบโต้กันได้ในลักษณะการติดต่อสื่อสาร</u> ใช้สมรรถนะของการคมนาคมที่ทันสมัย เข้าช่วย ผลลัพธ์ที่ได้จะปรับเปลี่ยนตามเหตุการณ์ที่เปลี่ยนไป เช่น เดิมถ้าต้องการทราบข้อมูลเพียงว่า มีโรงแรมอะไรบ้างเท่านั้น แต่ปัจจุบันเมื่อเราเลือกโรงแรมใดแล้ว เราจะทราบถึงขนาดว่าโรงแรมนั้นมีห้องพักกี่ห้องและมีห้องว่างหรือไม่ด้วย เป็นต้น

ขอบเขตของระบบมัลติมีเดีย ได้รับการพัฒนากว้างขวางมากขึ้นทำให้ข้อจำกัดในการใช้งานมีน้อยลง เดิมคิดกันแต่เพียงว่า ระบบมัลติมีเดียจะเหมาะสำหรับงานนำเสนอเท่านั้น แต่ความจริงเราสามารถนำมัลติมีเดียมาประยุกต์ใช้งานได้มากมาย ได้แก่ ประยุกต์ใช้ในการศึกษา ด้านธุรกิจ ด้านการโฆษณา ด้านการให้ข้อมูลตามสถานที่ต่าง ๆ เป็นต้น

<u>2.3.2 ส่วนประกอบของระบบมัลติมีเดีย</u>

สิ่งที่จำเป็นสำหรับระบบมัลติมีเดีย แยกออกเป็น 2 ส่วน คือ

2.3.2.1 ฮาร์ดแวร์มัลติมีเดีย องค์ประกอบที่สำคัญของระบบมัลติมีเดียคือ ภาพ เสียง และการตอบโต้กันได้ ส่วนที่เป็นฮาร์ดแวร์จึงต้องจำแนกออกไปตามองค์ประกอบดังกล่าว โดยมีคอมพิวเตอร์เป็นส่วนสำคัญ

คอมพิวเตอร์ที่ใช้งานในระบบมัลติมีเดียต้องแยกจากกัน ระหว่างการสร้างงานจำเป็นต้องใช้คอมพิวเตอร์ที่มีสมรรถนะสูง ถ้าเป็นระดับเวิร์คสเตชั่นจะดีมาก ซึ่งจะเป็นตัวเชื่อมโยงไปหาฮาร์ดแวร์อื่น ๆ

1. ฮาร์ดแวร์สำหรับงานด้านภาพ อุปกรณ์ที่สำคัญและจำเป็นนอกจากคอมพิวเตอร์ ได้แก่ เครื่องอ่านภาพ (Scanner) ใช้สำหรับงานสำเนาภาพจากต้นฉบับที่เป็นภาพนิ่ง มีหลายแบบ หลายระดับ มีลักษณะเป็นภาพสองมิติ กล้องถ่ายภาพเชิงตัวเลข (Digital Camera) เป็นกล้องถ่ายภาพธรรมดาที่มีการเปลี่ยนส่วนรับภาพที่เป็นฟิล์มเป็นตัวเปลี่ยนสัญญาณภาพมาเป็นสัญญาณเชิงตัวเลขหรือเรียกว่า CCD (Charge Couple Device) ใช้ทำสำเนาภาพนิ่งได้ทั้ง 2 มิติ และ 3 มิติ กล้องถ่ายภาพวิดีทัศน์ (Video Camera) ใช้งานถ่ายภาพเคลื่อนไหว และใช้สัญญาณเชิงตัวเลขและใช้ CCD เช่นเดียวกับกล้องถ่ายภาพเชิงตัวเลข การ์ดแปลงสัญญาณอนาล็อกเป็นสัญญาณเชิงตัวเลข (Digital Card) เนื่องจากเครื่องเล่นวิดีโอและกล้องถ่ายภาพวิดีทัศน์ ส่วนใหญ่ให้สัญญาณแบบอนาล็อก การจะนำสัญญาณจากอุปกรณ์ดังกล่าวไปใช้งานกับคอมพิวเตอร์ จึงจำเป็นต้องแปลงสัญญาณจากอนาล็อกให้เป็นสัญญาณเชิงตัวเลข

2. ฮาร์ดแวร์สำหรับงานด้านเสียง ในส่วนของระบบเสียงนั้น เดิมเป็นแบบอนาล็อก แต่ในปัจจุบันได้มีการประยุกต์และพัฒนาระบบเสียงให้ใช้สัญญาณเชิงตัวเลขได้ ฮาร์ดแวร์ที่ใช้สำหรับงานด้านเสียงจึงแทบจะประยุกต์ใช้กับระบบมัลติมีเดียได้ทันที ทั้งโดยตรงและโดยอ้อม อุปกรณ์มีดังนี้ เครื่องเล่นซีดี (CD-ROM) ถ้าไม่ต้องการบันทึกเสียงลงในคอมพิวเตอร์ก็สามารถจะแยกใช้ระบบเสียงจากภายนอกได้ โดยอาศัยการควบคุมการทำงาน วิธีที่สะดวกที่สุดจะเป็นการใช้เครื่องเล่นซีดี-รอม ที่ในปัจจุบันเล่นได้ทั้งระบบเสียงและระบบอ่านข้อมูลคอมพิวเตอร์ เพียงแต่ต้องบันทึกลงแผ่นซีดีมาก่อนจากนั้นจึงมาเปิดใช้งาน โดยอาศัยคำสั่งจากตัวเครื่องคอมพิวเตอร์เป็นตัวควบคุมอีกทีหนึ่ง เสียงสำเร็จรูป (Clip Sound) เป็นระบบเสียงสำเร็จรูปในลักษณะไฟล์สัญญาณเชิงตัวเลขสามารถใช้งานได้ทันที ส่วนใหญ่จะเป็นเสียงเพลงหรือเอฟเฟ็กต์ต่าง ๆ

2.3.2.2. ซอฟต์แวร์มัลติมีเดีย แยกออกเป็น 2 ตอน คือ ตอนสร้างกับตอนใช้งาน

1. ซอฟต์แวร์สร้างงานระบบมัลติมีเดีย แยกออกเป็นสองส่วนคือ ซอฟต์แวร์สำหรับสร้างภาพและเสียง กับซอฟต์แวร์จัดระบบ

- ซอฟต์แวร์สำหรับสร้างภาพและเสียง มักมีการใช้งานและเป็นที่คุ้นเคยของผู้ใช้คอมพิวเตอร์อยู่แล้ว มีการตกแต่ง ตัดต่อ ดัดแปลง สร้างใหม่ ซึ่งแยกกันทำเป็นชิ้นเป็นตอนได้

- ซอฟต์แวร์จัดระบบมัลติมีเดีย เป็นซอฟต์แวร์ที่ควบคุมงานส่วนต่าง ๆ มาจัดลำดับ เพื่อทำให้มีการตอบโต้กันได้ เช่น บอกให้รู้ว่าถึงตอนนี้จะมีภาพและต่อไปจะมีเสียง หรือถ้ากดตรงนี้จะมีภาพนั้นภาพนี้ออกมา หรือมีเสียงนั้นเสียงนี้ เป็นต้น ซอฟต์แวร์ประเภทนี้ยังไม่เป็นที่คุ้นเคยมากนัก ทำให้ยังมีน้อย ราคาจึงสูงและที่สำคัญยังต้องใช้กับคอมพิวเตอร์ที่มีสมรรถนะสูงด้วย

2. ซอฟต์แวร์ใช้งานระบบมัลติมีเดีย ระบบมัลติมีเดียนั้นเกิดจากการสร้างขึ้นมาโดยผ่านทางระบบคอมพิวเตอร์ ทำให้ไม่เป็นการยากนักหากจะใช้งาน แต่มีข้อจำกัดอยู่ที่ความง่ายในการใช้งาน

2.3.3 เทคโนโลยีที่เกี่ยวข้องกับมัลติมีเดีย

เนื่องจากระบบคอมพิวเตอร์มัลติมีเดีย เป็นการรวบรวมเทคโนโลยีหลายอย่างเข้าด้วยกัน เพื่อให้เกิดความสมบูรณ์ในการทำงาน เทคโนโลยีเหล่านั้น ได้แก่

1. การพัฒนาเทคโนโลยีในการบันทึกข้อมูลการทำงานของมัลติมีเดียประกอบไปด้วยภาพและเสียง

2. การพัฒนาด้านระบบคอมพิวเตอร์เครือข่าย สิ่งที่ระบบคอมพิวเตอร์มัลติมีเดียเข้าไปมีบทบาทร่วมกับระบบคอมพิวเตอร์เครือข่าย เช่น การติดต่อสื่อสารด้วยระบบ Electronics Mail ซึ่งเดิมเป็นการติดต่อที่เป็นลักษณะ Text Base เท่านั้น เป็นการนำสองเทคโนโลยีมาร่วมช่วยกัน ทำให้การติดต่อสื่อสารในระบบเครือข่ายคอมพิวเตอร์ทำได้ทั้งที่เป็นภาพและเสียง

3. การพัฒนาเทคนิคการย่อขนาดข้อมูล การย่อข้อมูลที่มีประสิทธิภาพจะเป็นปัจจัยสำคัญอย่างหนึ่งในการทำงานของระบบคอมพิวเตอร์มัลติมีเดีย

4. การพัฒนาไมโครคอมพิวเตอร์ การทำงานของคอมพิวเตอร์มัลติมีเดียเป็นการทำงานที่เกี่ยวข้องกับข้อมูลในปริมาณมหาศาล กระบวนการย่อและขยายขนาดข้อมูลจะต้องเกิดอย่างรวดเร็วและมากพอที่จะทำให้การติดต่อส่งข้อมูลระหว่างหน่วยความจำและอุปกรณ์ต่าง ๆ ไม่เกิดการหยุดชะงัก เพราะถ้าเกิดเหตุการณ์เช่นนี้จะทำให้การแสดงผลทั้งภาพและเสียงอาจผิดพลาดไปจากของจริงได้

5. การพัฒนาของจอภาพ

6. การพัฒนาอุปกรณ์ป้อนข้อมูล

7. การพัฒนาซอฟต์แวร์ ส่วนหนึ่งที่ทำให้โลกคอมพิวเตอร์มัลติมีเดียเป็นจริงก็คือ การพัฒนาซอฟต์แวร์ให้มีประสิทธิภาพสูงและมีการใช้งานได้ง่ายขึ้น การพิจารณาเลือกซอฟต์แวร์เพื่อมาทำงานด้านมัลติมีเดียอาจพิจารณาได้จาก

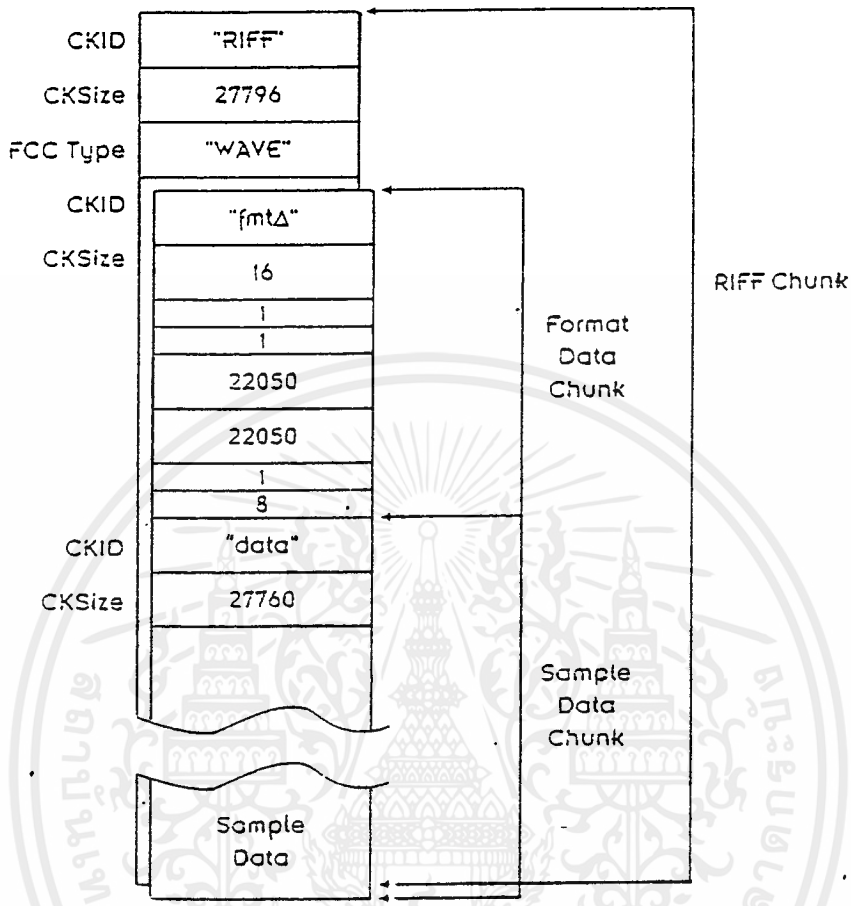
- ความง่ายในการใช้งาน
- ความสามารถในการนำเสนองาน
- ความสามารถในการติดต่อกับผู้ใช้
- ความสามารถในการใช้ฟังก์ชันในการคำนวณ
- ความสามารถในการใช้งานร่วมกับโปรแกรมอื่น ๆ
- มี Library สนับสนุนการทำงาน
- ความสามารถในการทำเอกสารประกอบโปรแกรม
- ความสามารถในการส่งแอพพลิเคชันที่เสร็จแล้วให้ผู้ใช้

2.4 ประเภทของไฟล์ชนิดต่าง ๆ

2.4.1 โครงสร้างของ RIFF ไฟล์

ไฟล์ RIFF (ย่อมาจาก Resource Interchange File Format) เป็นไฟล์ที่ได้รับการพัฒนาสำหรับการจัดเก็บข้อมูลทางด้านมัลติมีเดีย (Multimedia) โดยเฉพาะ โดยข้อมูลที่เก็บลงในรูปแบบของไฟล์ RIFF จะจัดเก็บ

เป็นบล็อกๆ โดยแต่ละบล็อกเราเรียกว่า Chunk สำหรับเก็บไฟล์ที่มาระบบปฏิบัติการวินโดว์ ดังรูปที่ 2.6 เป็นตัวอย่างโครงสร้างของ RIFF ไฟล์



รูปที่ 2.6 โครงสร้างของ RIFF ไฟล์

ในระดับสูงสุดของ Chunk โดยตัวมันเอง จะระบุชื่อซึ่งเป็นตัวระบุของ Chunk Tds ถ้าจุดสูงสุดของไฟล์เสียงจะพบว่า 4 ไบท์แรกจะบรรจุตัวอักษร R,I,F และ F อยู่ การอ่านและการเขียน RIFF ไฟล์จะใช้มาตราฐานโครงสร้างข้อมูลซึ่งเรียกว่า MMCKINFO (Multimedia Chunk Information) ซึ่งจะมีโครงสร้างดังนี้

2.4.1.1 ข้อมูลโครงสร้าง MMICKIFO

เป็นข้อมูลโครงสร้างสำหรับจัดเก็บข้อมูลของบล็อกข้อมูล RIFF โดยแต่ละฟิลด์มีรายละเอียดดังนี้

ckid

เป็นข้อมูลสตริงขนาด 4 ไบต์ ซึ่งฟิลด์ชนิดนี้ได้ถูกประกาศให้เป็นข้อมูลชนิด FOURCC ซึ่งข้อมูลโครงสร้างนี้ก็จะประกอบไปด้วยสมาชิกเพียงฟิลด์เดียวคือ

```
Type FOURCC
    FourBytes As String *4
End Type
```

cksize

ข้อมูลชนิดเลขจำนวนเต็ม long บอกขนาดของฟิลด์ข้อมูลของกลุ่ม โดยที่ค่าของ cksize จะเท่ากับขนาดของฟิลด์ข้อมูล และไม่รวม 4 ไบต์สำหรับฟิลด์ cksize

fccType

ข้อมูลชนิดสตริงขนาด 4 ไบต์เช่นเดียวกับฟิลด์ ckid ซึ่งจะจัดเก็บสตริงซึ่งบอกถึงประเภทของรูปแบบ (form) สำหรับบล็อก RIFF และบอกถึงประเภทของ List ของบล็อก LIST

dwFlags

ข้อมูลชนิดตัวเลขจำนวนเต็ม long ซึ่งเราจะใช้กำหนดเงื่อนไขการทำงานของฟังก์ชั่นไฟล์มัลติมีเดีย I/O ซึ่งสามารถกำหนดได้ตามค่าคงที่ดัง ตารางที่ 2.3

ค่าคงที่	ค่าตัวเลข	รายละเอียด
MMIO_READ	&H0	กำหนดให้เปิดไฟล์สำหรับการอ่านเท่านั้น
MMIO_WRITE	&H1	กำหนดให้เปิดไฟล์สำหรับการเขียนเท่านั้น
MMIO_READWRITE	&H2	กำหนดให้เปิดไฟล์สำหรับทั้งการอ่านและการเขียน
MMIO_COMPAT	&H0	กำหนดให้เปิดไฟล์ในโหมดของการคอมแพตทิเบิล
MMIO_EXCLUSIVE	&H10	กำหนดให้เปิดไฟล์ในโหมด Exclusive เพื่อใช้ในงานส่วนตัว
MMIO_DENYWRITE	&H20	ให้เปิดไฟล์โดยไม่ให้มีการเขียนทับจากแอพพลิเคชั่นอื่น
MMIO_DENYRED	&H30	กำหนดให้เปิดไฟล์โดยไม่มีการอ่านจากแอพพลิเคชั่น
MMIO_DENYNONE	&H40	กำหนดให้เปิดไฟล์โดยสามารถอ่านหรือเขียนทับได้จากแอพพลิเคชั่น

ตารางที่ 2.3 ค่าคงที่ของฟังก์ชั่นมัลติมีเดีย

แต่สำหรับส่วนถัดไปของ RIFF ที่เรียกว่า subchunk หรือ format chunk นั้น จะมี ความแตกต่างจากบล็อกอื่นๆ อยู่บ้าง ดังนั้นเราจะอ่านข้อมูลในบล็อกนี้โดยใช้ข้อมูลโครงสร้าง PCMWAVEFORMAT ดังมีรายละเอียดดังนี้

2.4.1.2 โครงสร้างของไฟล์เสียง (WAVE FILE)

การที่จะเข้าใจว่าไฟล์เสียงมีหลักการทำงานอย่างไรนั้น เราจะพิจารณาจากตัวอย่างซึ่งทำการวิเคราะห์ไฟล์ wave

ในระดับต่ำที่สุดของไฟล์เสียงจะบรรจุด้วยกลุ่ม (chunk) 3 กลุ่ม โดยที่กลุ่ม ของ RIFF จะมีข้อมูลระบุอยู่มากที่สุด ซึ่งในความเป็นจริง กลุ่มของไฟล์เสียงทั้งหมดก็คือกลุ่ม RIFF นั่นเอง ซึ่งการตรวจสอบขนาด (คำสั่ง cksize) จะพบได้หลังจากปรากฏ 'RIFF' อย่างทันที ถัดไปคือตรวจสอบค่าอุปกรณ์ (คำสั่ง ckID) จะระบุค่าของไฟล์หารด้วย 8 ไบต์ ที่ต้องการจะบันทึกลงใน RIFF ไฟล์ ในลำดับถัดไป จะเรียกว่ากลุ่มย่อย (subchunk) ซึ่งจะระบุอยู่ภายในกลุ่ม RIFF ส่วนแรกของกลุ่มนี้คือ 'fmt' จะระบุข้อมูลที่สำคัญของโครงสร้างไฟล์เสียงแบบ พีซีเอ็ม (PCMWAVEFORMAT) ซึ่งอยู่ในรูปของข้อมูลแบบ ดิจิตอล (digital) ในกลุ่มย่อยที่ 2 จะบรรจุข้อมูล (data) เป็นส่วนที่ใหญ่ส่วนหนึ่งของไฟล์ ตอนจบของข้อมูล เป็นข้อตกลงว่าเป็นการสิ้นสุดของ RIFF ขนาดของ RIFF เท่ากับผลรวมของจำนวนไบต์ที่มีอยู่ของ'fmt' และ 'data' ตัวอย่างของไฟล์. Wav มีในตารางที่ 2.4 RIFF ไฟล์อาจจะบรรจุไปด้วยชนิดของกลุ่มอื่น เรียกว่า LIST ซึ่งรายการที่ระบุลงไปนี้จะเป็นข้อมูลที่บอกถึง ลิขสิทธิ์ หรือ ข้อมูลส่วนตัวของผู้ผลิต

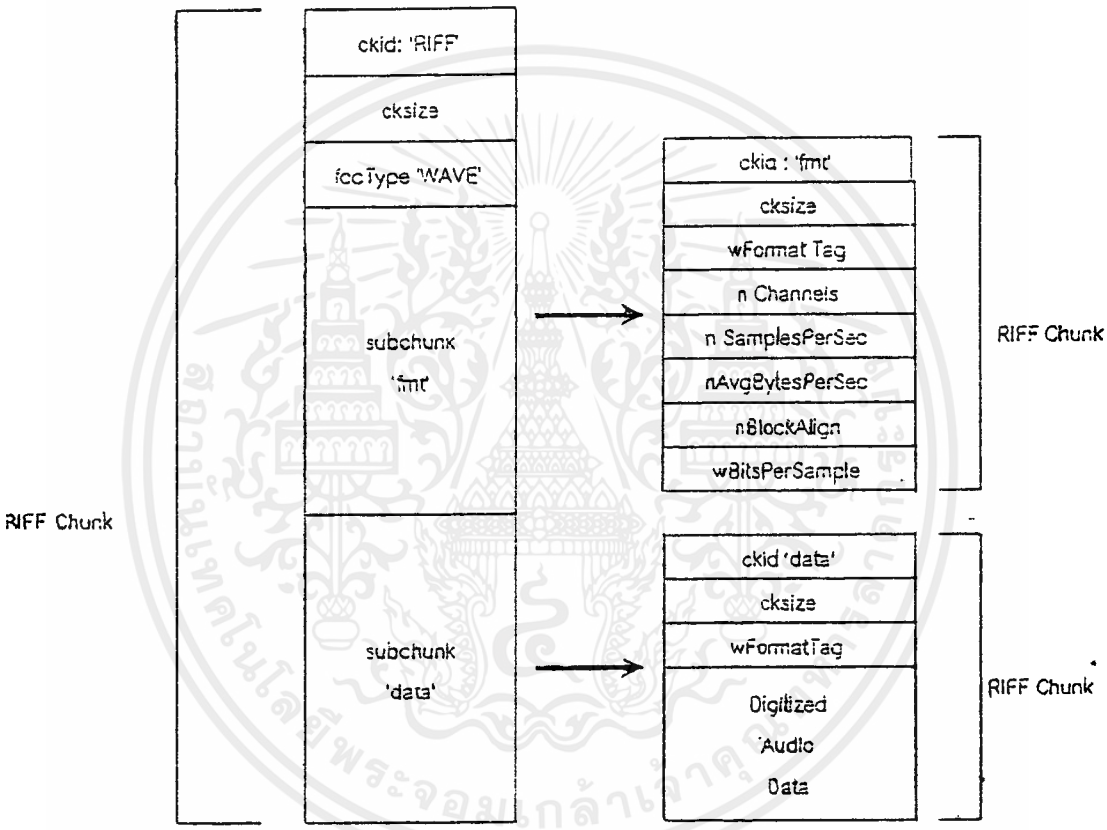
Position Hex	Dec	Size in Bytes	Contents	Comments
0000	0	4	"RIFF"	Each byte contains one charactor,ckId
0004	4	4	27796	Equals the file size minus eight bytes,csize
0008	8	4	"WAVE"	fccType
000c	12	4	"fmt"	Next ckID;notice the blank,must be four characters
0010	16	4	16	The WAV format chunk is 16 bytes,ckSize
0014	20	2	1	1 indicates a PCM WAV format,wFormatTag
0016	22	2	1	Number of channels,nchannels
0018	24	4	22050	Sampling rate, nSamplesPerSEc
001C	28	4	22050	nAvgBytesPerSec
0020	32	2	1	Effectively bytes per sample,nBlockAlign
0022	34	2	8	wBitsPerSample
0024	36	4	"data"	Next ckID;this chunk contains the wavedata itself
0028	40	4	27760	Next CD;size of wave data
002C	44	Depends on Data		The digtezed audio data

ตารางที่ 2.4 โครงสร้างของไฟล์.wav

2.4.1.3 ข้อมูลโครงสร้าง PCMWAVEFORMAT

โดยที่ข้อมูลต่าง ๆ ที่จัดเก็บในบล็อกนี้จะเป็นข้อมูลที่ถูกใช้ในการคำนวณและบอกคุณลักษณะของข้อมูลของสัญญาณเสียง ในรูปแบบ PCM สำหรับข้อมูลแบบโครงสร้าง PCMWAVEFORMAT จะประกอบด้วยสมาชิกเพียง 2 ฟิลด์ เท่านั้นคือ

```
Type PCMWAVEFORMAT
        wf      As WAVEFORMAT
        wBitsPerSample As Integer
End   Type
```



รูปที่ 2.7 ตัวอย่างของ PCMWAVEFORMAT

จากรูปที่ 2.7เป็นข้อมูลโครงสร้าง WAVEFORMAT ซึ่งแต่ละตัวสมาชิกมีรายละเอียดดังนี้

```
Type WAVEFORMAT
        wFormatTag      As      Integer   'Format Type
        nChannels       As      Integer   'Number of chennels(i.e. mono , stereo , etc)
        nSamplesPerSec  As      Long      'Sample rate
        nAvgBytesPerSec As      Long      'For buffer estimation
        nBlockAlign     As      Integer   'Block size of data
End Type
```

ข้อมูลชนิดจำนวนเต็ม ได้มีการกำหนดในรูปแบบเดียวเท่านั้นตามตารางที่ 2.5 คือ

ค่าคงที่	ค่าตัวเลข	รายละเอียด
WAVE_FORMAT_PCM	1	ข้อมูลในรูปแบบ PCM

ตารางที่ 2.5 กำหนดค่าคงที่

nChannels

ข้อมูลชนิดจำนวนเต็ม กำหนดจำนวนแชนแนลของข้อมูลโดยที่ข้อมูลโมโน (mono)จะใช้ 1 แชนแนล ในขณะที่ข้อมูลสเตอริโอ (stereo) จะใช้ 2 แชนแนล

nSamplePerSec

ข้อมูลชนิดเลขจำนวนเต็ม long กำหนดอัตราส่วนการสุ่มข้อมูล ในหน่วยของอัตราการสุ่มข้อมูลใน 1 วินาที

nAvgBytesPerSec

ข้อมูลชนิดเลขจำนวนเต็ม long กำหนดขนาดของข้อมูลเฉลี่ยในหน่วยของ ไบต์ สำหรับการขนถ่ายในช่วงเวลา 1วินาที

nBlockAlign

ข้อมูลชนิดเลขจำนวนเต็ม กำหนดค่าขอบเขตการจัดเรียง (alignment) บล็อกในหน่วยของไบต์ โดยที่ข้อมูลในรูปแบบ PCM จะเรียงข้อมูลตามจำนวนไบต์ที่ใช้ในการสุ่มแต่ละครั้ง เช่น ถ้าหากข้อมูลเป็นแบบสเตอริโอ nBlockAlign ก็จะมีค่าเท่ากับ 4 ไบต์ เป็นต้น

wBitserSample

ข้อมูลชนิดเลขจำนวนเต็ม กำหนดจำนวนบิตที่ใช้ในการสุ่มข้อมูล 1ครั้ง เช่น 8 หมายถึง ใช้ 8 บิตในการสุ่มข้อมูลแต่ละครั้ง เป็นต้น

ส่วนในบล็อก subchunk 'data' หรือที่เรียกว่า data chunk นั้น ก็เป็นบล็อกที่จัดเก็บข้อมูลของสัญญาณเสียงในรูปแบบข้อมูลดิจิตอล ดังนั้นในการเล่นกลับไฟล์เสียง เราจะต้องทำการอ่านข้อมูลทั้งหมดในบล็อกนี้ เพื่อส่งให้กับอุปกรณ์ส่งสัญญาณเสียงต่อไป

2.4.2 Microsoft Audio Vidro Interleave (AVI)

เป็น Format ของไฟล์ที่ใช้ในการ Capture, แก้ไข, PlayBack ลำดับของภาพและเสียง โดยทั่วไปไฟล์ AVI จะเก็บ stream ของข้อมูลหลากหลายรูปแบบ แต่ส่วนใหญ่จะเก็บข้อมูลของภาพและเสียงเป็นหลัก ไฟล์ AVI พื้นฐานจะเก็บเฉพาะลำดับของรูปภาพ จนถึงไฟล์ AVI แบบพิเศษที่สามารถเพิ่มเติม Control Track หรือ MIDI Track ลงไปได้ Control Track จะทำให้สามารถควบคุมอุปกรณ์ภายนอก เช่น เครื่องบันทึก Video Disc ได้ และ MIDI Track ทำให้สามารถเล่นเสียง MIDI เป็น BackGround ได้ แต่ต้องใช้โปรแกรมเฉพาะจึงจะสามารถใช้ความสามารถเหล่านี้ได้ โปรแกรมประยุกต์ทั่วไปไม่สามารถใช้ความสามารถเหล่านี้ได้ จึงแสดงผลเป็นภาพและเสียงได้เท่านั้น

2.4.3 Picture File

Format ของ Image File ในปัจจุบันมีอยู่มากมายตามลักษณะการใช้งาน เราสามารถแบ่งลักษณะของ Image File เป็น 2 ประเภท คือ

1.Bitmap File เป็นไฟล์ที่เก็บรายละเอียดของภาพในลักษณะของ Pixel จะมีการกำหนดไว้อย่างแน่นอนว่า ที่ตำแหน่งใดของภาพจะมีลักษณะอย่างไร เมื่อเราย่อหรือขยายภาพของไฟล์ชนิด Bitmep นี้เราจะสูญเสียความคมชัดของภาพไป แต่ในการเก็บภาพเพื่อใช้ในงานมัลติมีเดีย เรามักใช้ Image File ประเภทนี้ เพราะว่าสามารถเรียกแสดงได้รวดเร็ว ในปัจจุบันได้นำเทคโนโลยีการบีบข้อมูลมาใช้กันอย่างแพร่หลาย และได้นำมาใช้ในการเก็บ Still Image ด้วย เช่น RLE File, JPEG File, GIF File ในแต่ละวิธีจะมีอัลกอริทึมในการบีบข่อมูลที่แตกต่างกันออกไป

2.Vector File เป็นไฟล์ที่สามารถย่อและขยายภาพได้โดยมีการสูญเสียรายละเอียดน้อยมาก แต่เรามักใช้กับภาพที่ไม่มีรายละเอียดมากหรือลักษณะของภาพที่มีการย่อและขยายขนาด เช่น Logo และในงานเขียนแบบต่าง ๆ

2.4.3.1 BMP

เป็น File Format มาตรฐานของ Windows โปรแกรมส่วนใหญ่ของ Windows ที่สามารถทำงานกับ Still Image ได้ จะสามารถทำงานกับไฟล์ใน Format นี้ได้ และ Visual Basic ก็สามารถเรียกไฟล์ใน format นี้ใช้ได้โดยตรง อาทิเช่น Form ,Picture Control 'Image Control ไฟล์ใน Format นี้สามารถแสดงสีได้ถึง 16.7 ล้านสี (True Color 24 Bits) แต่ข้อด้อยของไฟล์ใน Format นี้คือ ไม่มีการใช้เทคโนโลยีการบีบข้อมูลเลยทำให้ไฟล์มีขนาดค่อนข้างใหญ่มากเมื่อเปรียบเทียบกับไฟล์ใน Format อื่น

รูปแบบข้อมูลภาพแบบ BMP

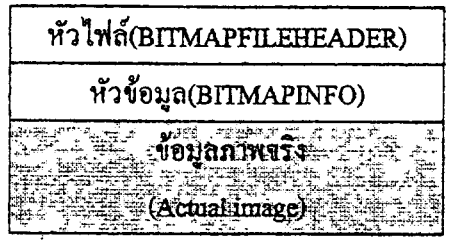
รูปแบบที่มีใช้อยู่โดยทั่วไปมีรูปแบบดังนี้คือ GIF IFF/LBM WPG BMP PIC TGA PCX และ TIFF สำหรับรูปแบบใหญ่ๆ ที่นิยมใช้กันมากคือ PCX GIF BMP และ TIFF แต่ รูปแบบหนึ่งที่ได้รับความนิยมมากที่สุด คือ BMP หรือบิตแมพ สาเหตุก็เพราะว่าข้อมูลแบบบิตแมพนั้น ไม่มีการบีบข้อมูล จึงมีความรวดเร็วและสะดวกในการอ่านข้อมูลอย่างมาก และเป็นรูปแบบข้อมูลกลางที่ใช้ในการแลกเปลี่ยนระหว่างกันของโปรแกรมประยุกต์ ต่างๆ ใช้ในการส่งข้อมูลไปให้อุปกรณ์ต่างๆ เช่น อุปกรณ์การแสดงผล เป็นต้นแต่ในรูปแบบบิตแมพนี้ต้องเปลืองเนื้อที่ฮาร์ดดิสในการจัดเก็บข้อมูลมากกว่ารูปแบบอื่นที่ใช้การเข้ารหัสบีบข้อมูล

รูปแบบข้อมูลแบบ BMP

รูปแบบสำหรับใช้กับโปรแกรมระบายสีในวินโดว์เวอร์ชัน (version) 3.0 เป็นต้นมา โดยรูปแบบข้อมูลแบบนี้สามารถเป็นข้อมูลสีได้ตั้งแต่ 1-24 บิต มีส่วนหัว (header) ในการบอกรายละเอียดต่างๆ ของภาพ โดยกำหนดในลักษณะโครงสร้าง (structure) ในภาษาระดับสูง ข้อมูลภาพในรูปแบบบิตแมพประกอบด้วย 3 ส่วนคือ

- หัวไฟล์(BITMAPFILEHEADER)
- หัวข้อมูล(BITMAPINFO) ซึ่งรวมทั้งข้อมูลต่างๆ และพาเลตของสี(color palette)
- ข้อมูลภาพจริง

จากรูปที่ 2.8 แสดงส่วนประกอบกันของโครงสร้างที่ใหญ่ที่สุดของภาพถ่ายในรูปแบบบิทแมพ ซึ่งข้อมูลพาเลตสีและข้อมูลภาพจะเปลี่ยนโครงสร้างไปตามรูปแบบของจำนวนสีของภาพและวิธีการถอดรหัสที่ใช้ในการบีบข้อมูลภาพ



รูปที่ 2.8 แสดงโครงสร้างของภาพถ่ายในรูปแบบบิทแมพ

2.4.3.1.1 หัวไฟล์(BITMAPFILEHEADER)

หัวของไฟล์ (BITMAPFILEHEADER) จะประกอบด้วยข้อมูลเกี่ยวกับชนิดของ รูปแบบข้อมูลภาพ ขนาดของภาพและโครงร่างดังแสดงในรูปที่ 2.9

BITMAPFILEHEADER ประกอบด้วย
-WORD bfType;
-DWORD bfSize;
-WORD bfReserved1;
-WORD bfReserved2;
-DWORD bfOffBits

รูปที่ 2.9 รูปแสดงหัวไฟล์ BITMAPFILEHEADER

โดยที่ bfType เป็นชนิดของรูปแบบข้อมูลภาพในที่นี้สำหรับบิทแมพคือ “BM”
bfSize เป็นขนาดของไฟล์ทั้งหมด
bfReserbed1,2 ไม่ได้ใช้งานปกติกำหนดให้มีค่าเท่ากับศูนย์
bfOffBits เป็นขนาดของหัวไฟล์ทั้งหมด ใช้เพื่อข้ามไปจุดเริ่มต้นของข้อมูลภาพจริง

2.4.3.1.2 หัวข้อมูล (BITMAPINFO)

หัวข้อมูลจะเป็นตัวกำหนดขนาดต่างๆ และข้อมูลสีดังแสดงในรูปที่ 2.10

BITMAPINFO ประกอบด้วย
-BITMAPINFOHEADER bmiheader;
-RGBQUAD bmiColors[1];

รูปที่ 2.10 รูปแสดงข้อมูล BITMAPINFO

โดยที่ bmiHeader เป็นข้อมูลเกี่ยวกับขนาดต่างๆ จำนวนสีและรูปแบบของสี

bmiColors . เป็นอาร์เรย์ข้อมูลสีอาร์จีบีที่เป็นตัวกำหนดสีในการแสดงผลของภาพ

ที่เป็นรายละเอียดต่างๆ จะอยู่ใน BITMAPINFOHEADER ดังแสดงในรูปที่ 2.11

STRUCTURE BITMAPINFOHEADER ประกอบด้วย	
-DWORD	biSize;
-DWORD	biWidth;
-DWORD	biHeight;
-WORD	biPlanes;
-WORD	biBitCount;
-DWORD	biCompression;
-DWORD	biSizeImage;
-DWORD	biXPelsPerMeter;
-DWORD	biTPelsPerMeter;
-DWORD	biClrUsed;
-DWORD	biClrImportant;

รูปที่ 2.11 รูปแสดงหัวข้อมูล BITMAPINFOHEADER

โดยที่ biSize เป็นขนาดของส่วนหัวข้อมูล (BITMAPINFOHEADER)มีหน่วยเป็นไบต์

biWidth เป็นความกว้างของภาพ (มีหน่วยเป็น จุดภาพ/เส้น)

biHeiht เป็นความสูงของภาพ (มีหน่วยเป็น จุดภาพ/เส้น)

biPlanes เป็นจำนวนหน้าของสี (color plane) สำหรับอุปกรณ์เป้าหมาย (targetdevice) ปกติกำหนดให้เป็น 1 เสมอ

biBitCount จำนวนบิทต่อจุดภาพ(1,4,8,24)

biCompression ชนิดของการบีบอัด(compression)

biSizeImage เป็นขนาดของภาพ(มีหน่วยเป็นไบต์)

biXPelsPerMeter ความละเอียดสำหรับอุปกรณ์เป้าหมายในแนวนอนต่อหนึ่งเมตร

biYPelsPerMeter ความละเอียดสำหรับอุปกรณ์เป้าหมายในแนวตั้งต่อหนึ่งเมตร

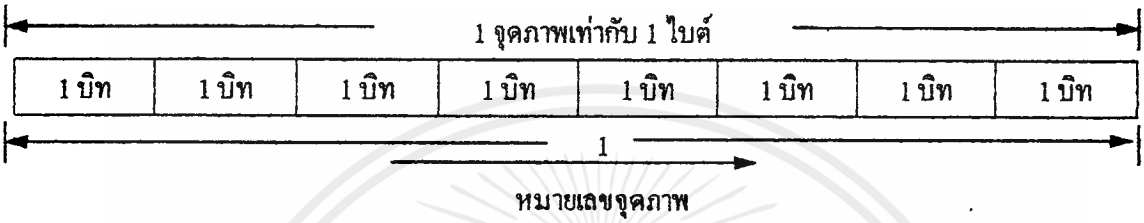
biClrUsed จำนวนดัชนีสี (color index) ในตารางสี (color table) มีค่าได้ 3 กรณี

biClrImportant จำนวนความสำคัญของดัชนีสี (color index) ในการแสดงผล ถ้าเป็นศูนย์ ทุกสีจะมีความสำคัญเท่ากันหมด

ข้อมูลภาพจริง (Actual Image Information)

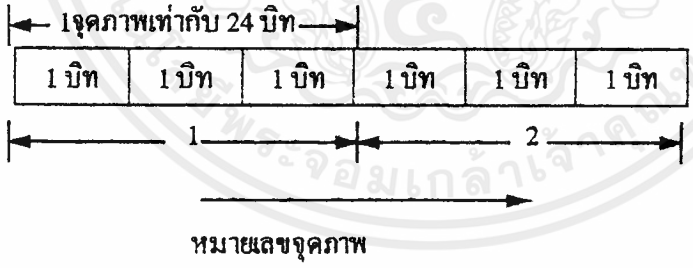
ส่วนของข้อมูลบิทแมพจริงจะมีการเก็บที่แตกต่างกันตามจำนวนบิทข้อมูลภาพ (biBitCount) และลักษณะการบีบอัด biCompression แต่โดยทั่วไปแล้วข้อมูลในรูปแบบบิทแมพปกติที่นิยมในปัจจุบันในการส่งข้ามระหว่างโปรแกรมประยุกต์ต่างๆนั้น จะไม่ใช้การบีบอัดข้อมูล ดังนั้นจึงแบ่งรูปแบบข้อมูลในลักษณะที่นิยมใช้กันในปัจจุบันได้ดังนี้

กรณีที่ 1 ถ้าเป็น 8 บิทต่อภาพ หมายความว่ามีดัชนีสี bmiColors ที่เป็นแอร์เรย์ จำนวน 256 แอร์เรย์ คือมีสี 256 สี โดยข้อมูลแต่ละ 8 บิทจะแทนหนึ่งจุดภาพ ดังรูปที่ 2.12



รูปที่ 2.12 แสดงข้อมูลภาพบิทแมพ 256 สี

กรณีที่ 2 ถ้าเป็น 24 บิทต่อจุดภาพ หมายความว่าจะสามารถแสดงสีได้พร้อมๆ กันเท่ากับ 2^{24} คือ 16.7 ล้านสีคือสามารถผสมสีได้โดยตรง ไม่ต้องมีการใช้ดัชนีสี ดังนั้นในกรณีนี้ bmiColors เท่ากับ NULL ข้อมูลแต่ละ 3 ไบต์ที่เรียงกันจะแสดง relative intensity ของสีฟ้า สีเขียว และสีแดง (BGR) ตามลำดับ โดยข้อมูลแต่ละ 24 บิตจะแทน 1 จุดภาพดังรูปที่ 2.13



รูปที่ 2.13 แสดงข้อมูลภาพบิทแมพ 16.7 ล้านสี

2.4.3.2 GIF

เป็น File Format ของ CompuServe ซึ่งเป็น Format ของไฟล์ที่ใช้ในการสื่อสารข้อมูล (Down Load,Up Load) ไฟล์ใน Format นี้จะสามารถเรียกใช้ได้โดยโปรแกรมตบแต่งภาพ (Image Retouching) โดยทั่วไปสามารถเรียกใช้ได้ใน Visual Basic โดยผ่านทาง Third Party Control ที่ชื่อว่า GIFBOX.BVX ซึ่งมีข้อด้อยกว่า Image Control และ Picture Control ไฟล์ใน Format นี้สามารถแสดงจำนวนสีได้ 256 สี (8 Bits) มีการใช้

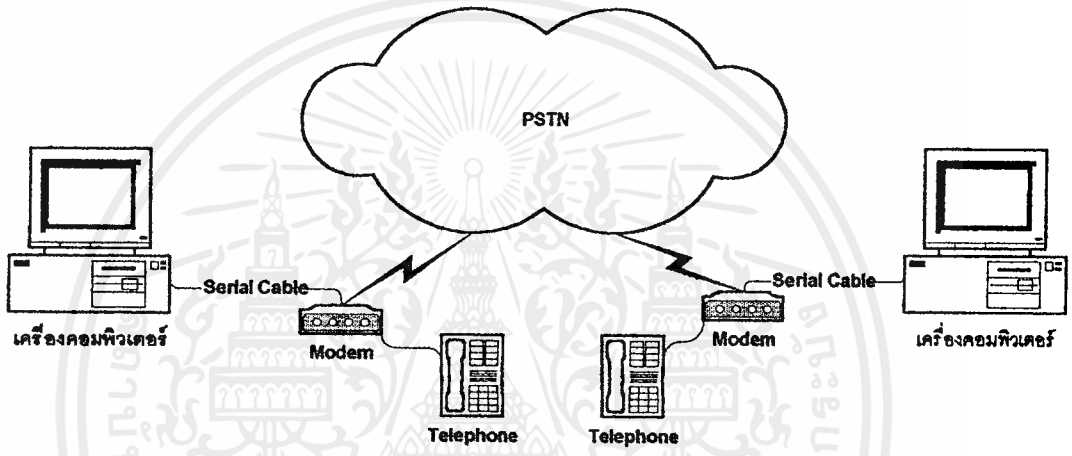
เทคโนโลยีการบีบข้อมูลเข้ามาช่วย เพื่อให้ขนาดของไฟล์ลดลง เพื่อประโยชน์ในการส่งผ่านข้อมูล คุณภาพของภาพหลังจากบีบข้อมูลแล้ว ไม่มีการสูญเสียรายละเอียดของภาพแต่อย่างใด

2.4.3.3 JPEG

เป็น Format ของ Still Image ที่กำลังเป็นที่นิยมมากในการเก็บข้อมูล เพราะว่าสามารถบีบข้อมูลได้มาก มีจำนวนสี 16.7 ล้านสี สามารถกำหนด Factor ของการบีบขนาดภาพได้ และใน Visual Basic สามารถใช้ Third Party Control QTPIC.VBX ในการเรียกภาพขึ้นมาแสดง แต่ข้อเสียของ File Format นี้ คือมีการสูญเสียรายละเอียดของภาพตามอัตราการบีบข้อมูลและการเรียกใช้ใน Visual Basic ยังมีปัญหาอยู่มาก

บทที่ 3
การคำนวณและการสร้าง

โปรแกรมที่สร้างขึ้นในโครงงานนี้ สร้างโดยใช้เดลไฟ 2.0 ซึ่งเป็นเครื่องมือสำหรับโปรแกรมเพื่อสร้างแอพพลิเคชันบนวินโดวส์ โดยภาษาที่ใช้ในการเขียนโปรแกรมเป็นภาษาปาสคาล ซึ่งโปรแกรมทั้งหมดถูกรวมไว้ในโปรแกรมหลักเรียกว่า โปรเจค (project) ในโครงงานนี้มีโปรแกรมหลักชื่อ 'YKMain.dpr' ซึ่งมีซอร์สโค้ดชื่อ 'YKMainCom.pas' ทำหน้าที่หลักในการติดต่อทางภาพ,เสียง และเอกสารข้อความ และเรียกใช้การทำงานของโปรแกรมย่อยต่างๆ ในบทนี้จะอธิบายลำดับการทำงานของโปรแกรมหลักและโปรแกรมย่อยต่างๆ พร้อมทั้งรูปแบบการรับส่งข้อมูลในแต่ละโปรแกรม ในรูปที่ 3.1 แสดงบล็อกไดอะแกรมของระบบการสื่อสารมัลติมีเดียผ่านคู่สายโทรศัพท์



รูปที่ 3.1 บล็อกไดอะแกรมของระบบการสื่อสารมัลติมีเดียผ่านคู่สายโทรศัพท์

โปรแกรม 'YKMainCom.pas' เป็นโปรแกรมหลักที่นอกจากจะติดต่อทั้งรับและส่งทางภาพ,เสียง และเอกสารข้อความแล้วยังเรียกการใช้งานโปรแกรมย่อยต่างๆ ดังนี้

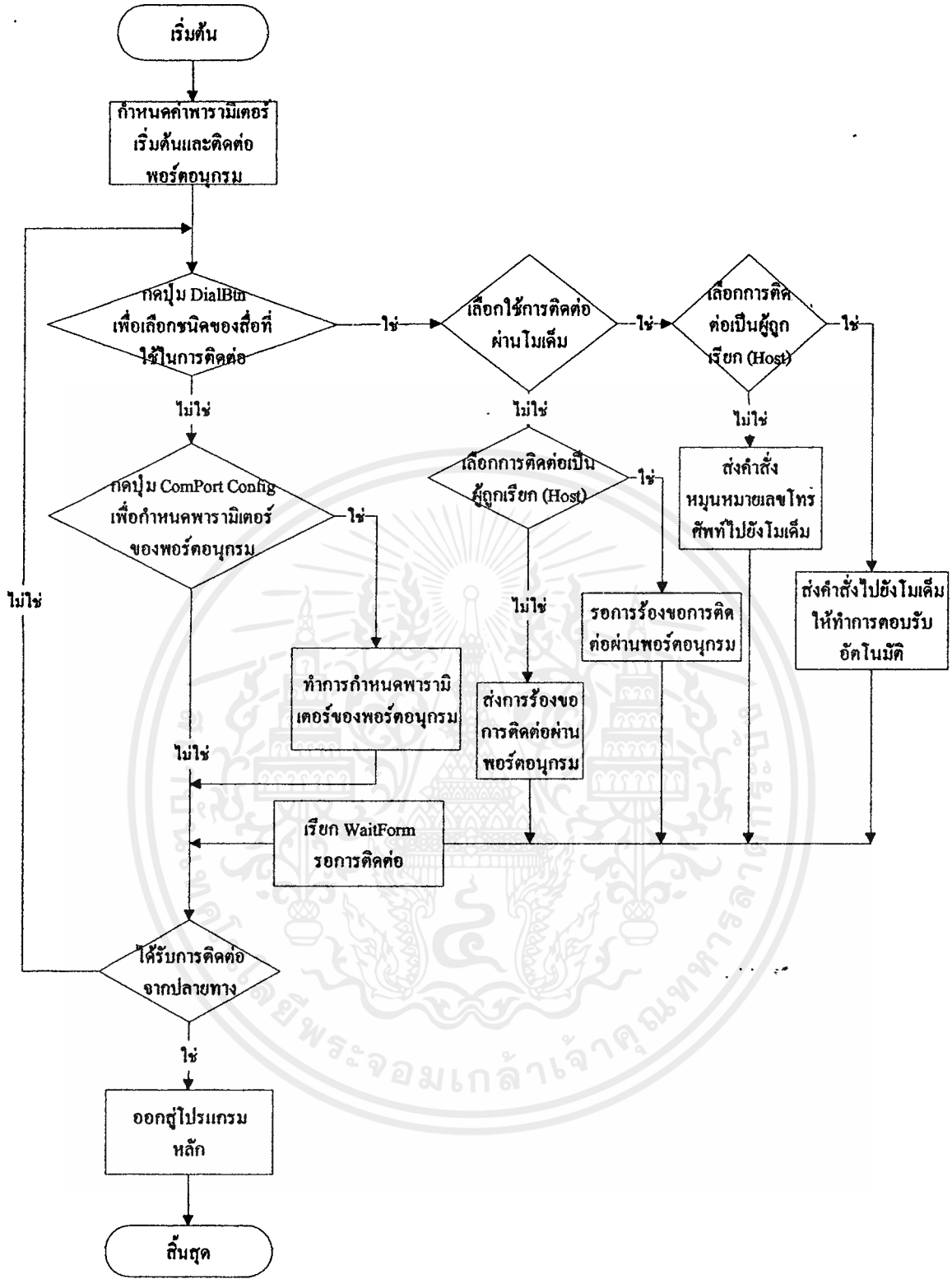
- โปรแกรม 'YKPConfig.pas' ทำหน้าที่กำหนดพารามิเตอร์ในการสื่อสารผ่านพอร์ตอนุกรม ได้แก่ หมายเลขพอร์ต ,อัตราเร็วของข้อมูล, จำนวนบิทข้อมูล,จำนวนแพริตี้บิท และรูปแบบการแฮนเชคกิ้ง
- โปรแกรม 'YKConnect.pas' ทำหน้าที่เลือกรูปแบบการติดว่าผ่านทางสายสัญญาณอนุกรม หรือผ่านอุปกรณ์โมเด็ม ทำการหมุนหมายเลขโทรศัพท์เพื่อเรียกการติดต่อ หรือรอการเรียกจากอุปกรณ์โมเด็ม
- โปรแกรม 'YKChat.pas' ทำหน้าที่ติดต่อระหว่างคอมพิวเตอร์ปลายทาง แบบการพิมพ์ข้อความสนทนาโต้ตอบกัน
- โปรแกรม 'YKSendFile.pas' ทำหน้าที่ส่งและรับไฟล์ข้อมูลต่างๆ ระหว่างคอมพิวเตอร์ปลายทางทั้งสองเครื่อง

นอกจากนี้ยังมีโปรแกรมและไฟล์อื่นๆ ที่ใช้ประกอบกับโปรแกรมหลักดังนี้

- โปรแกรม 'YKAbout.pas' บอกรายละเอียดเกี่ยวกับโปรแกรมนี้
- ไฟล์ 'Sound.wav' เป็นไฟล์ที่โปรแกรมหลักจะสร้างขึ้นมาเพื่อใช้เป็นไฟล์เริ่มต้นในการบันทึกเสียง
- ไฟล์ 'YKFile.bmp' เป็นไฟล์ภาพที่รับมาได้ เมื่อรับเสร็จจะนำมาแสดงออกหน้าจอ
- ไฟล์ 'YKFile.txt' เป็นไฟล์ตัวอักษรที่รับมาได้ เมื่อรับเสร็จจะนำมาแสดงออกหน้าจอ
- ไฟล์ 'YKFile.wav' เป็นไฟล์เสียงที่รับมาได้ เมื่อรับเสร็จก็จะนำมาเล่นโดยทันที
- ไฟล์ 'YKFile.avi' เป็นไฟล์ภาพเคลื่อนไหวและเสียงที่รับมาได้

3.1 โปรแกรมกำหนดรูปแบบการติดต่อ

ทำหน้าที่กำหนดพารามิเตอร์เริ่มต้นในการเชื่อมต่อระหว่างคอมพิวเตอร์สองเครื่อง โดยที่ผู้รับจำเป็นจะต้องทำการรอการติดต่อจากผู้เรียกก่อน ผู้เรียกจึงจะสามารถทำการเชื่อมต่อได้ ในรูปที่ 3.2 เป็นรูปแสดงแผนผังการทำงานของโปรแกรม YKConnect.pas



รูปที่ 3.2 รูปแสดงแผนผังการทำงานของโปรแกรม YKConnect.pas

3.2 โปรแกรมหลัก "YKMain.pas"

ทำหน้าที่ติดต่อระหว่างคอมพิวเตอร์ต้นทางและปลายทางด้วยรูปแบบต่างๆ โดยมีส่วนประกอบหลายส่วน ดังนี้ คือ

3.2.1 การสนทนาโต้ตอบ

ทำหน้าที่ส่ง-รับเสียงระหว่างคอมพิวเตอร์สองเครื่อง เมื่อผู้ใช้ด้านใดด้านหนึ่งมีการกดปุ่ม Speak โปรแกรมจะทำให้มีการบันทึกเสียงผู้ใช้ด้านนั้น โดยโปรแกรมจะเรียกใช้ฟังก์ชัน StartRecording ของคอมโพเนนต์ MediaPlayer เมื่อกดปุ่ม Stopแล้วไฟล์เสียงที่บันทึกได้ก็จะถูกส่งไปยังผู้ใช้อีกด้านหนึ่ง เมื่อคอมพิวเตอร์ส่งไฟล์เสียงไปจนครบ และที่คอมพิวเตอร์อีกด้านหนึ่งได้รับไฟล์เสียงอย่างถูกต้องแล้ว โปรแกรมก็จะเล่นเสียงที่บันทึกได้นั้นให้ผู้ใช้ด้านรับฟังได้ทันที

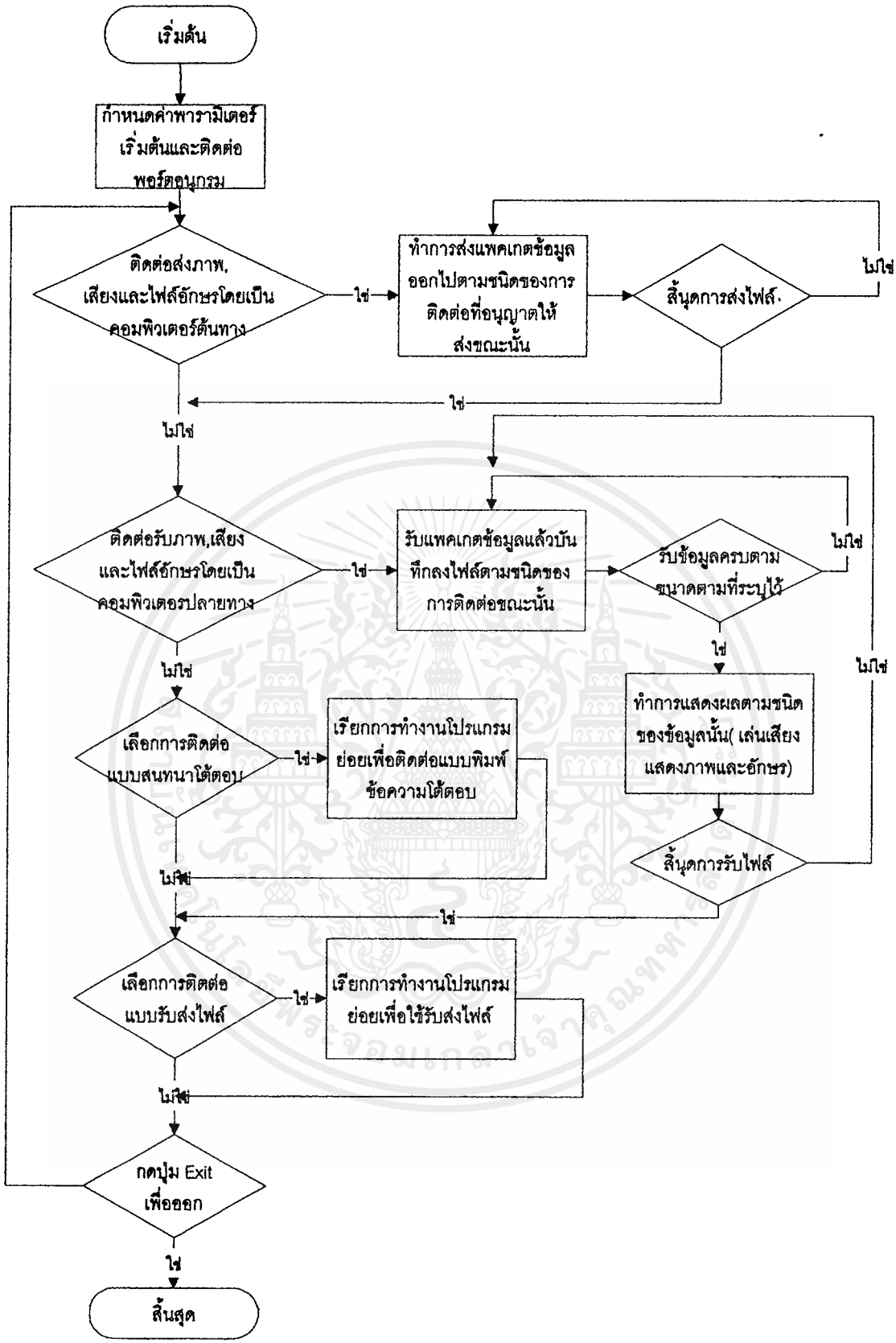
3.2.2 การส่งและรับภาพ

ทำหน้าที่ส่ง-รับภาพระหว่างคอมพิวเตอร์สองเครื่อง เมื่อผู้ใช้ด้านใดด้านหนึ่งมีการเลือกไฟล์ภาพ โดยกดปุ่ม SelectPic เพื่อเปิด OpenDialog และทำการเลือกไฟล์ภาพที่ต้องการ ถ้าเป็นภาพประเภทบิทแมพ (Bitmap) โปรแกรมจะเรียกใช้คอมโพเนนต์ JImage เพื่อแสดงภาพนั้นให้ผู้ใช้เห็น ส่วนถ้าเป็นภาพเคลื่อนไหว(AVI File) จะเรียกใช้คอมโพเนนต์ MediaPlayer เพื่อเล่นภาพเคลื่อนไหวนั้นให้ผู้ใช้เห็นก่อนที่จะส่ง เมื่อผู้ใช้กด SendPic โปรแกรมจะทำการส่งไฟล์ภาพนั้นออกไปยังผู้ใช้อีกด้านหนึ่ง เมื่อคอมพิวเตอร์ส่งไฟล์ภาพไปจนครบ และที่คอมพิวเตอร์อีกด้านหนึ่งได้รับไฟล์ภาพอย่างถูกต้องแล้ว โปรแกรมก็จะแสดงภาพที่รับได้นั้นให้ผู้ใช้ด้านรับเห็นได้ทันที

3.2.3 การส่งและรับข้อความเอกสาร

ทำหน้าที่ส่ง-รับข้อความเอกสารระหว่างคอมพิวเตอร์ เมื่อผู้ใช้ด้านใดด้านหนึ่งมีการพิมพ์ข้อความที่ต้องการลงในส่วน DocMemo เมื่อผู้ใช้กดปุ่ม SendText ข้อความที่ปรากฏใน DocMemo จะถูกบันทึกเป็นไฟล์ Text.txt แล้วโปรแกรมจึงทำการส่งไฟล์ไปยังคอมพิวเตอร์ปลายทาง เมื่อคอมพิวเตอร์ต้นทางส่งไฟล์ข้อความเอกสารไปจนครบ และที่คอมพิวเตอร์อีกด้านหนึ่งได้รับไฟล์อย่างถูกต้องแล้ว โปรแกรมก็จะแสดงข้อความที่รับได้นั้นให้ปรากฏใน DocMemo ให้ผู้ใช้ด้านรับเห็นได้ทันที

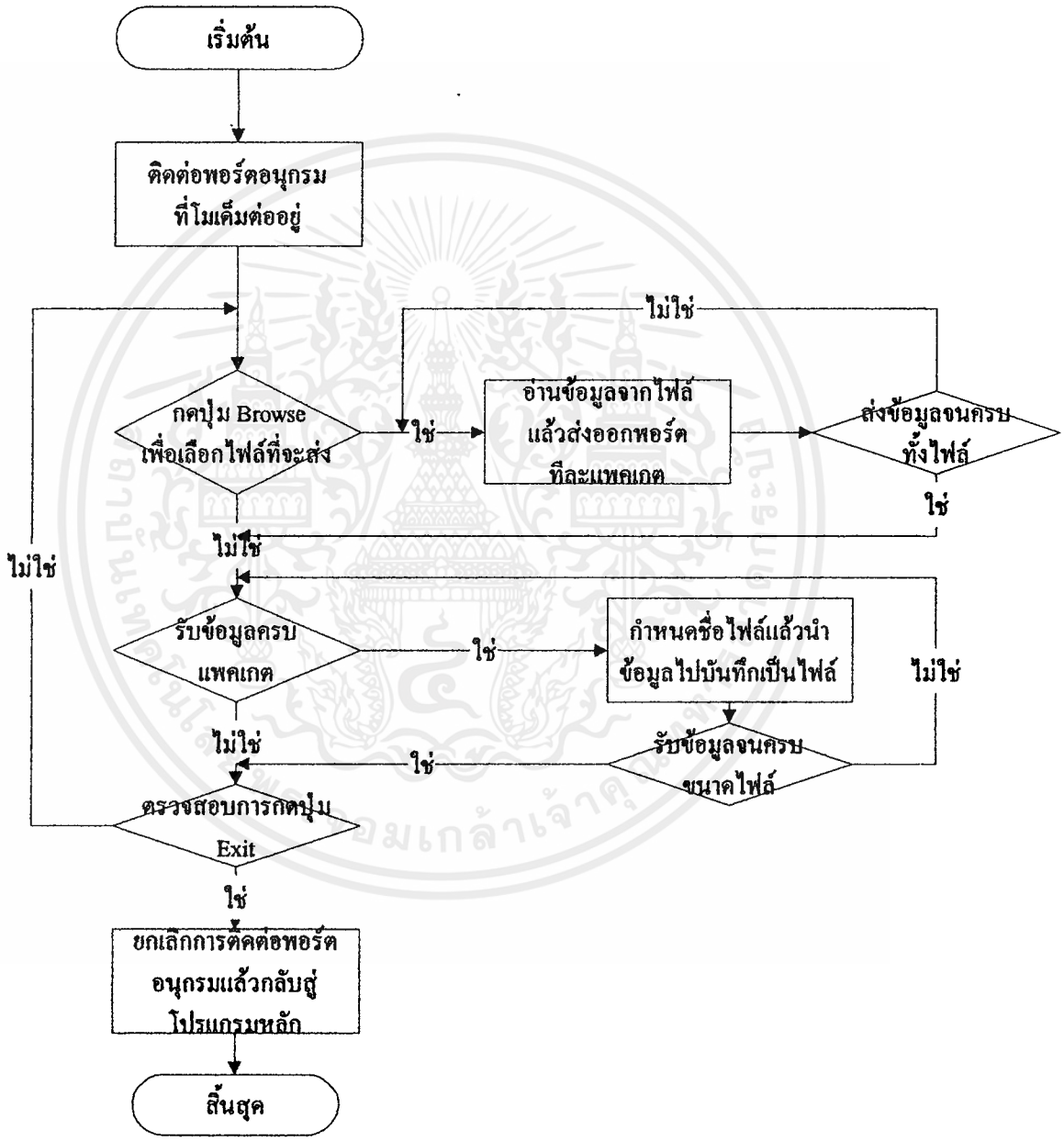
ในรูปที่ 3.3 เป็นรูปแสดงแผนผังการทำงานของโปรแกรม YKMain.pas



รูปที่ 3.3 แผนผังแสดงการทำงานของโปรแกรม YKMain.pas

3.3 โปรแกรมรับและส่งไฟล์ชนิดอื่นๆ

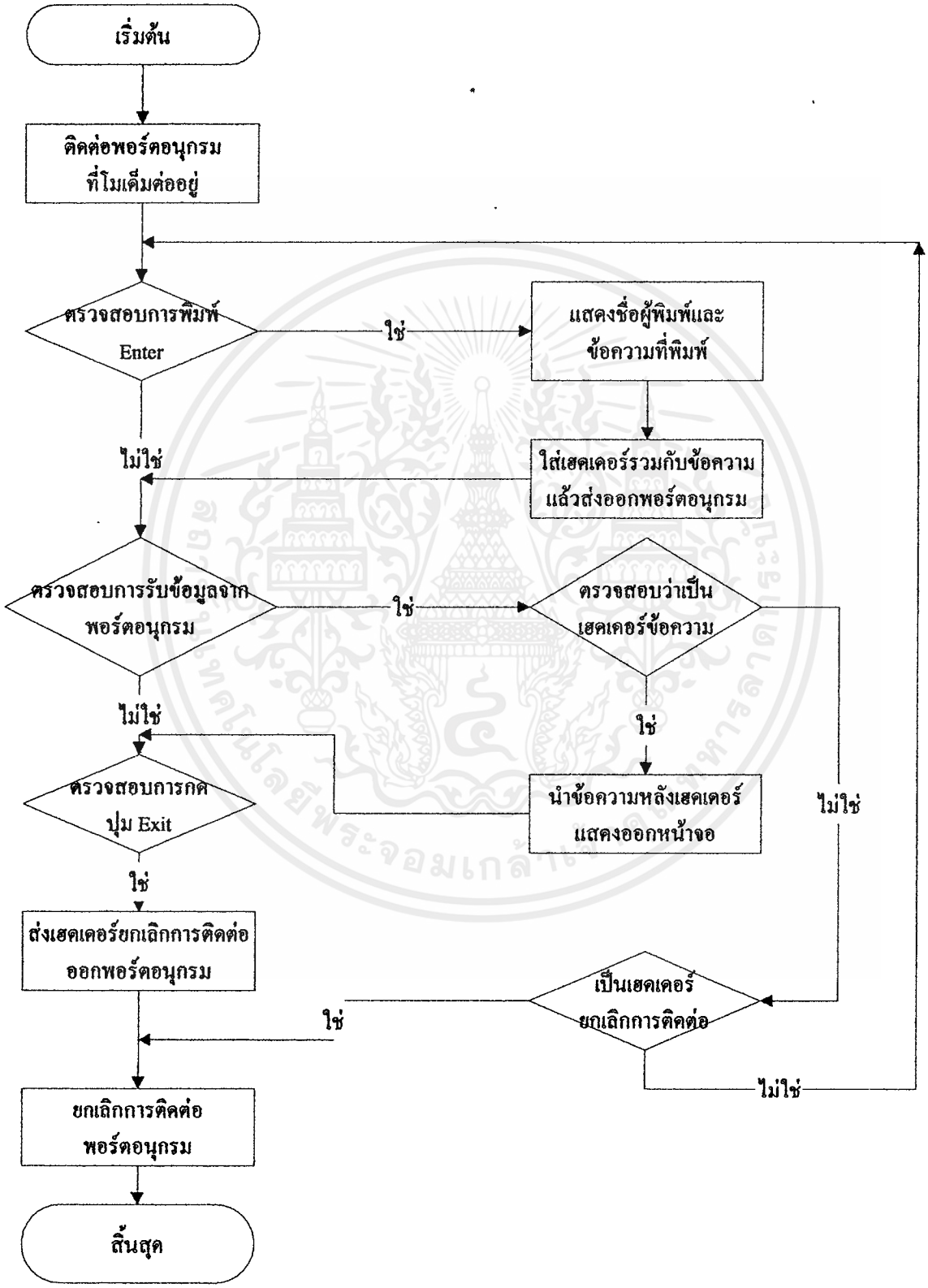
ทำหน้าที่รับ-ส่งไฟล์ชนิดอื่นๆ นอกจากไฟล์ภาพและเสียง เมื่อผู้ใช้ด้านใดด้านหนึ่งต้องการจะส่งและรับไฟล์ข้อมูลชนิดอื่น (SendFile) เมื่อผู้ใช้ด้านใดด้านหนึ่งกดปุ่ม Browse จะปรากฏ OpenDialog ขึ้น เมื่อเลือกไฟล์ที่จะส่งได้แล้ว และกดปุ่ม Send โปรแกรมจะทำการส่งไฟล์ที่ต้องการนั้นไปยัง เครื่องคอมพิวเตอร์ปลายทาง เมื่อคอมพิวเตอร์ด้านรับสามารถรับไฟล์ได้แล้ว จะปรากฏ SaveDialog ขึ้น เมื่อผู้ใช้ใส่ชื่อไฟล์ที่ต้องการบันทึกและทำการบันทึกเป็นอันเสร็จสิ้นการรับไฟล์ ในรูปที่ 3.4 แสดงแผนผังการทำงานของโปรแกรม YKSendFile.pas



รูปที่ 3.4 แผนผังแสดงการทำงานของโปรแกรม YKSendFile.pas

3.4 โปรแกรม "YKChat.pas"

ทำหน้าที่ติดต่อแบบพิมพ์ข้อความสนทนาโต้ตอบ(Chat) กันระหว่างคอมพิวเตอร์สองเครื่อง ในรูปที่ 3.5 แสดงแผนผังการทำงานของโปรแกรม YKChat.pas



รูปที่ 3.5 แผนผังการทำงานของโปรแกรม YKChat.pas

<u>สำหรับในการทดสอบส่งผ่านโมเด็ม</u> มีข้อกำหนดดังนี้

ใช้โมเด็มของ USRobotics ที่มี Bit Rate 28,800 bps และเป็นโมเด็มชนิดต่อภายนอก

คำสั่งต่าง ๆที่ใช้กำหนดค่าเริ่มต้นของโมเด็ม มีดังต่อไปนี้

ATZ	เพื่อรีเซ็ตโมเด็ม
ATB0F1M1X4&W	เพื่อกำหนด Answer Sequence ตามมาตรฐานของ ITU-T V.25 , ไม่ให้มีการสะท้อนกลับของคำสั่งที่ส่งให้แก่โมเด็ม (Local Echo OFF) , ให้ลำโพงเปิดอยู่จนกระทั่งเมื่อ connect แล้วจึงปิด , กำหนดโหมดแสดงการตอบสนอง
AT&A3&B1&G0&H1	กำหนดโปรโตคอลควบคุมความผิดพลาดแบบ LAPM/MNP/NONE และโปรโตคอลบีบอัดข้อมูลแบบ V42 bis/MNP5 , กำหนดอัตราของ serial port คงที่ , ไม่มี guard tone , กำหนดการควบคุมการไหลของข้อมูลในการส่งเป็นแบบ hardware flow control(CTS)
AT&I0&K1&M4&N0	ในการควบคุมการไหลของข้อมูลในการรับเป็นแบบไม่ต้องใช้ software flow control , การบีบอัดข้อมูลเป็นไปโดยอัตโนมัติ , เซ็ต error control เป็นแบบ Normal/ARQ , เซ็ตความเร็วในการเชื่อมต่อแบบแปรผันได้
AT&R2&S0&T5&U0&Y1	จะทำการส่งข้อมูลไปให้คอมพิวเตอร์ได้ก็ต่อเมื่อ RTS อยู่สภาวะ ON , ให้ DSR อยู่สภาวะ ON เสมอ , เซ็ต break handling
ATS0=1	กำหนดให้รับสายอัตโนมัติ(Auto Answer) เมื่อมีเสียงกริ่งโทรศัพท์ดังแล้ว 1 ครั้ง

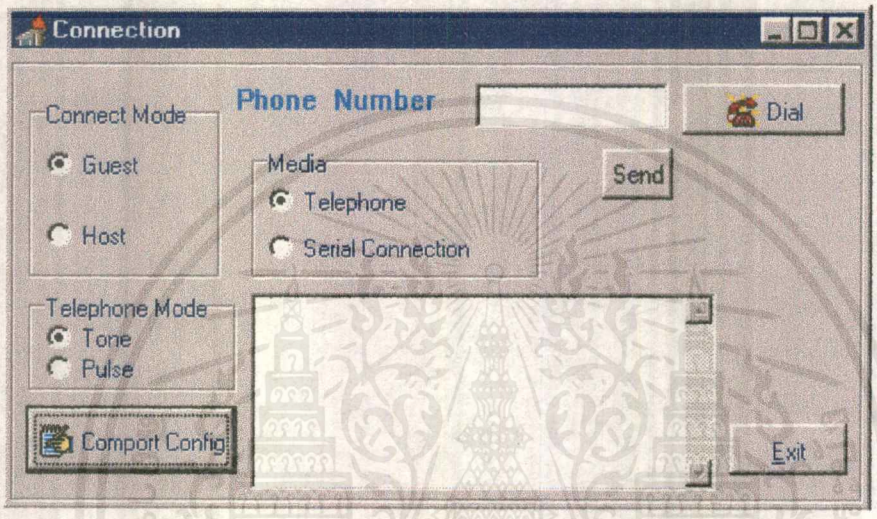
โดยที่ในการทดลอง ต้องกำหนดให้ค่าเริ่มต้น ของโมเด็มทั้งสองด้านเหมือนกัน

บทที่ 4

การทดลองและผลการทดลอง

ในบทนี้เป็นขั้นตอนในการรับส่งข้อมูล โดยมีจุดมุ่งหมาย คือ การส่งและรับไฟล์ข้อมูล ระหว่างเครื่องคอมพิวเตอร์ต้นทางกับเครื่องคอมพิวเตอร์ปลายทาง ซึ่งมีขั้นตอนต่าง ๆดังต่อไปนี้

เมื่อทำการรันโปรแกรม YKMain.exe จะปรากฏฟอร์มของ YKConnect.pas ดังรูปที่ 4.1



รูปที่ 4.1 รูปแสดงฟอร์มของ YKConnect.pas

ในรูปที่ 4.1 นี้ จะมีส่วนที่ให้ผู้ใช้กำหนดได้ 3 ส่วน คือ

1. Connect Mode
   - ถ้าผู้ใช้ต้องการเป็นผู้เรียก ให้เลือก Guest
   - ถ้า ผู้ ใช้ต้องการเป็นผู้รับ ให้เลือก Host
2. Telephone Mode
   - ถ้าต้องการใช้การหมุนหมายเลขโทรศัพท์แบบ Tone ให้เลือก Tone
   - ถ้าต้องการใช้การหมุนหมายเลขโทรศัพท์แบบ Pulse ให้เลือก Pulse
3. Media
   - ถ้าผู้ใช้ต้องการสื่อสารโดยตรงผ่านทางสายนัลโมเด็ม ซึ่งต่อเชื่อมอยู่ระหว่างคอมพิวเตอร์ 2 เครื่องให้ผู้ใช้เลือก Serial Connection

   กรณีที่ผู้ใช้ต้องการเป็นผู้เรียก ให้เลือก Guest แล้วปุ่มกดจะกลายเป็นปุ่ม Connect

   กรณีที่ผู้ใช้ต้องการเป็นผู้รับ ให้เลือก Host แล้วปุ่มกดจะกลายเป็นปุ่ม Wait

ในการเชื่อมต่อผู้รับจำเป็นที่จะต้องทำการรอผู้เรียกก่อน โดยการกดปุ่ม Wait จากนั้น ผู้เรียกจึงจะสามารถกดปุ่ม Connect เพื่อทำการเชื่อมต่อได้

- ถ้าผู้ใช้ต้องการสื่อสารผ่านโมเด็มไปตามคู่สายโทรศัพท์ ให้ผู้ใช้เลือกTelephone

กรณีที่ผู้ใช้ต้องการเป็นผู้เรียก ให้เลือก Guest แล้วปุ่มกดจะกลายเป็นปุ่ม Dialโดยที่ผู้เรียกสามารถกำหนดหมายเลขโทรศัพท์ของผู้รับได้ในช่อง Phone Number การกำหนดหมายเลขโทรศัพท์สามารถทำได้ดังต่อไปนี้

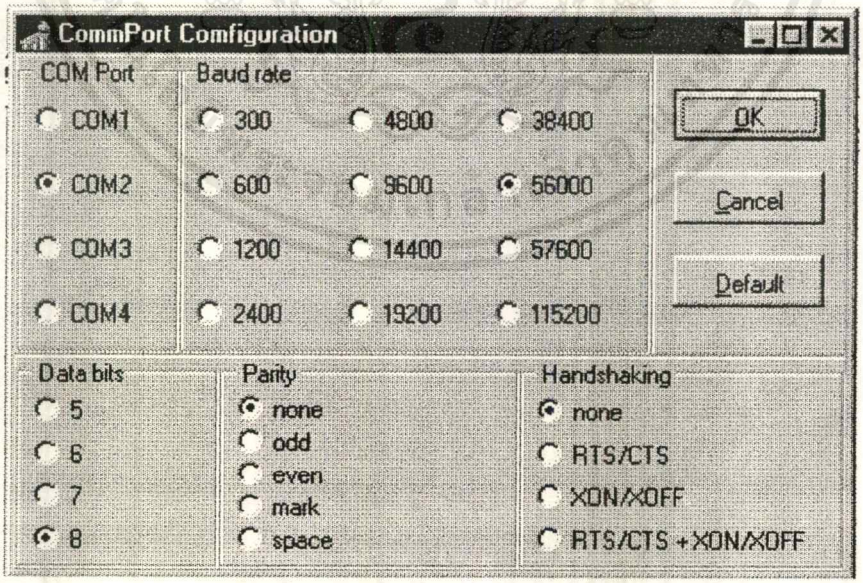
1. ถ้าเป็นหมายเลขภายในให้กำหนดได้โดยตรง เช่น 2594
2. ถ้าเป็นการเรียกออกผ่านOperator ให้กำหนดเป็น 0ww7113022
3. ถ้าเป็นการเรียกออกโดยตรงให้กำหนดเป็น 7113022

กรณีที่ผู้ใช้ต้องการเป็นผู้รับ ให้เลือก Host แล้วปุ่มกดจะกลายเป็นปุ่ม Listen ในการเชื่อมต่อผู้รับจำเป็นที่จะต้องทำการรอผู้เรียกก่อน โดยการกดปุ่ม Listen จากนั้น ผู้เรียกจึงจะสามารถกดปุ่ม Dial เพื่อทำการเชื่อมต่อได้

ส่วนปุ่ม Comport Config มีไว้เพื่อกำหนดเงื่อนไขต่าง ๆในการเชื่อมต่อดังนี้

1. ส่วน Comport เพื่อกำหนดว่าจะรับ-ส่งข้อมูลผ่านทางพอร์ตอนุกรมใด
2. ส่วน Baud Rate เพื่อกำหนดว่าจะใช้อัตราในการส่งข้อมูลเท่าใด
3. ส่วน Data bits เพื่อกำหนดขนาดข้อมูลที่จะส่ง
4. ส่วน Parity เพื่อกำหนดชนิดของการตรวจสอบ Parity
5. ส่วน Handshaking เพื่อกำหนดรูปแบบการ Handshaking

เมื่อผู้ใช้ต้องการกำหนดเงื่อนไขต่าง ๆ ให้ผู้ใช้กดปุ่ม Comport Config โปรแกรมจะทำการแสดงหน้าฟอร์ม YKConfig.pas ออกมาดังแสดงในรูปที่ 4.2



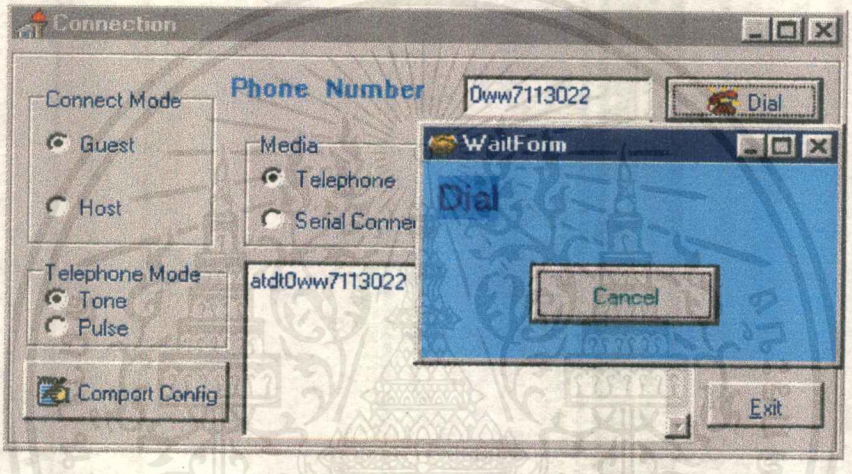
รูปที่ 4.2 รูปแสดงฟอร์มของ YKConfig.pas

ในการกำหนดเงื่อนไขต่าง ๆ ในหน้าฟอร์ม CommPort Configulation นั้น จำเป็นอย่างยิ่งที่ทางผู้เรียกและผู้รับจะต้องกำหนดส่วนต่าง ๆต่อไปนี้ให้ตรงกัน คือ Baud rate , Data bits , Parity , Handshaking สำหรับในส่วน COM Port ไม่จำเป็นต้องกำหนดตรงกันก็ได้ ขึ้นอยู่ว่าผู้ใช้ ใช้ Com Port เบอร์ใดในการติดต่อ

เมื่อกำหนดเงื่อนไขส่วนต่าง ๆ ได้ตามต้องการแล้ว ให้ผู้ใช้กดปุ่ม OK โปรแกรมจะทำการปิดหน้าฟอร์ม YKConfig.pas กลับไปยังหน้าฟอร์ม YKConnect.pas เพื่อให้ผู้ใช้กำหนดส่วนอื่น ๆตามต้องการต่อไป

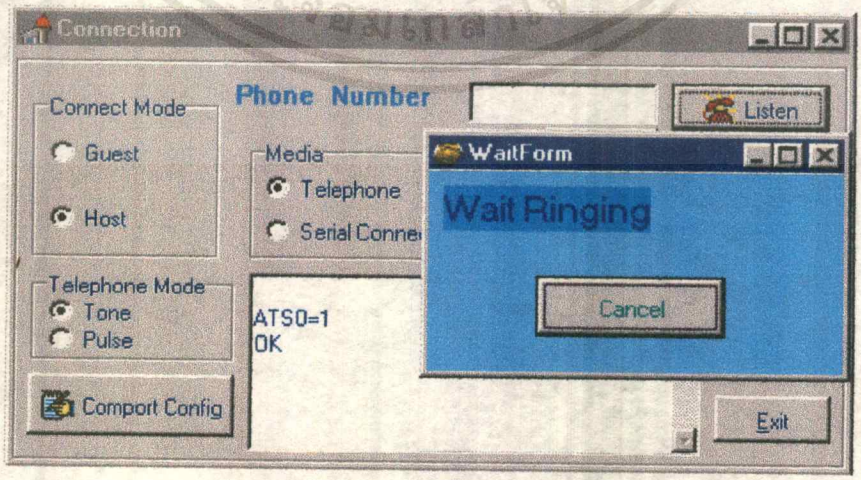
ในการทดลองที่เลือกมาแสดงนี้

<u>ทางด้านผู้เรียก</u> ได้ทำการกำหนดส่วน Media เป็น Telephone ส่วน Telephone Mode เป็น Tone ส่วน Connect Mode เป็น Guest ในรูปที่ 4.3 เป็นรูปแสดงฟอร์มของ YKConnect.pas ที่ด้านผู้เรียก



รูปที่ 4.3 รูปแสดงฟอร์มของ YKConnect.pas ที่ด้านผู้เรียก

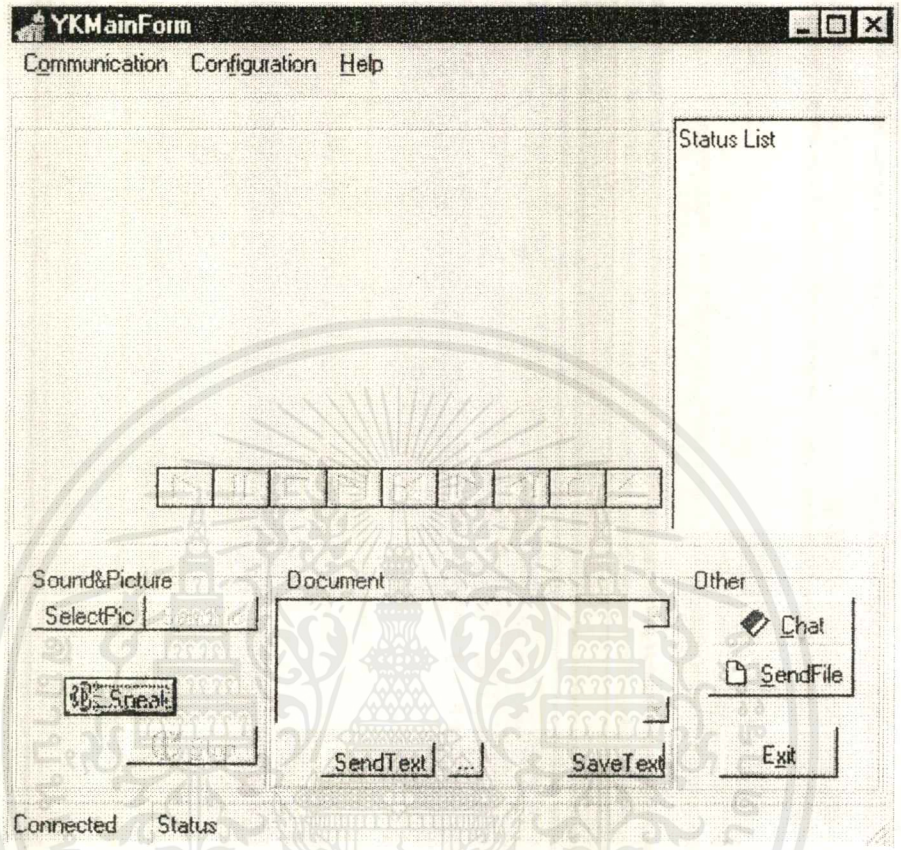
<u>ทางด้านผู้รับ</u> ได้ทำการกำหนดส่วน Media เป็น Telephone ส่วน Telephone Mode เป็น Tone ส่วน Connect Mode เป็น Host ในรูปที่ 4.4 เป็นรูปแสดงฟอร์มของ YKConnect.pas ที่ด้านผู้รับ



รูปที่ 4.4 รูปแสดงฟอร์มของ YKConnect.pas ที่ด้านผู้รับ

4.1 การทดลองรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์สองเครื่องผ่านโมเด็มไปตามคู่สายโทรศัพท์

เมื่อทางด้านผู้เรียกและผู้รับทำการเชื่อมต่อได้แล้ว โปรแกรมจะทำการปิดหน้าฟอร์ม YKConnect .pas และทำการเปิดหน้าฟอร์ม YKMain.pas ขึ้น แสดงถึงได้มีการเชื่อมต่อสำเร็จแล้ว ดังแสดงในรูปที่ 4.5



รูปที่ 4.5 รูปแสดงฟอร์มของ YKMain.pas

โดยในหน้าฟอร์ม YKMain.pas นี้ ผู้ใช้จะ พบส่วนประกอบอยู่ 3 ส่วน คือ

1.ส่วนของ Sound&Picture

2.ส่วนของ Document

3.ส่วนของ Other

4.1.1 การทดลองรับ-ส่งไฟล์ข้อมูลที่เป็นรูปภาพ

เมื่อผู้ใช้ฝ่ายใดฝ่ายหนึ่งทำการกดปุ่ม SelectPic ในฟอร์ม YKMain.pas แล้ว จะปรากฏ OpenDialog ขึ้นมา ให้ผู้ใช้เลือกรูปภาพที่ต้องการจะส่ง โดยไฟล์ภาพที่ใช้ได้ในการทดลองนี้ คือ .BMP .JPEG และ.AVI

ถ้ารูปภาพที่เลือกเป็นไฟล์ภาพ .BMP และ .JPEG เมื่อเลือกรูปภาพที่ต้องการได้แล้ว รูปภาพนั้นจะปรากฏให้เห็น จากนั้นให้ผู้ใช้กดปุ่ม SendPic เพื่อทำการส่งไฟล์รูปภาพนั้น

ถ้ารูปภาพที่เลือกเป็นไฟล์ภาพ .AVI เมื่อเลือกรูปภาพที่ต้องการได้แล้ว ผู้ใช้สามารถดูรูปภาพนั้นได้ก่อนทำการส่ง โดยการกดปุ่ม Play จากนั้นให้ผู้ใช้กดปุ่ม SendPic เพื่อทำการส่งไฟล์รูปภาพนั้น ผลการทดลองแสดงดังรูปที่ 4.6 และรูปที่ 4.7

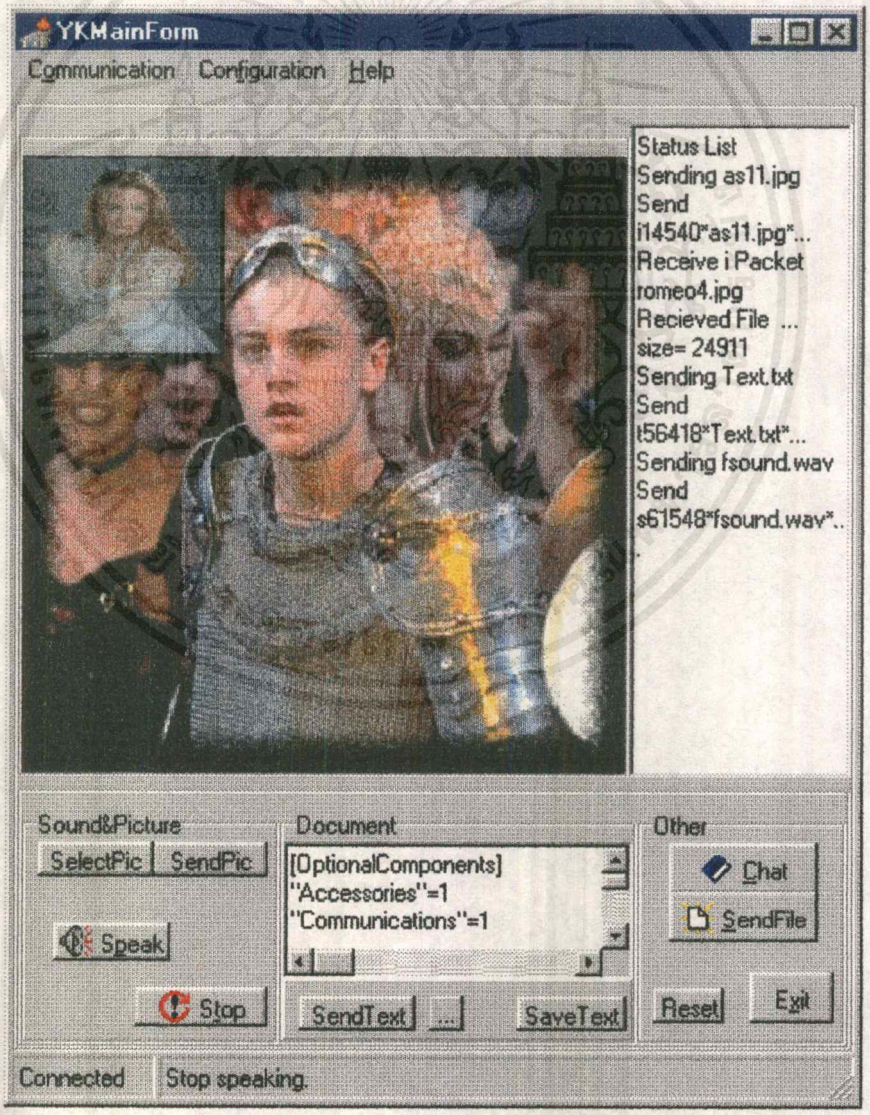
4.1.2 การทดลองรับ-ส่งไฟล์ข้อมูลที่เป็นเสียง

เมื่อผู้ใช้ต้องการส่งเสียงพูด ทำได้โดยกดปุ่ม Speak เมื่อสิ้นสุดการพูด ให้ผู้ใช้กดปุ่ม Stop เพื่อทำการส่งไฟล์เสียงนั้น ผลการทดลองแสดงดังรูปที่ 4.6 และรูปที่ 4.7

4.1.3 การทดลองรับ-ส่งไฟล์ข้อมูลที่เป็นไฟล์อักษร

เมื่อผู้ใช้ต้องการส่งไฟล์ที่เป็นไฟล์อักษร ทำได้โดยพิมพ์ข้อความลงในช่อง Document และกดปุ่ม SendText เพื่อทำการส่งไฟล์อักษรนั้น ผลการทดลองแสดงดังรูปที่ 4.6 และรูปที่ 4.7

ในรูปที่ 4.6 เป็นรูปแสดงผลการทดลองรับส่งไฟล์ต่างๆทั้งไฟล์ภาพ,เสียงและตัวอักษรที่ปรากฏในฟอร์มของ YKMain.pas ทางด้านส่ง



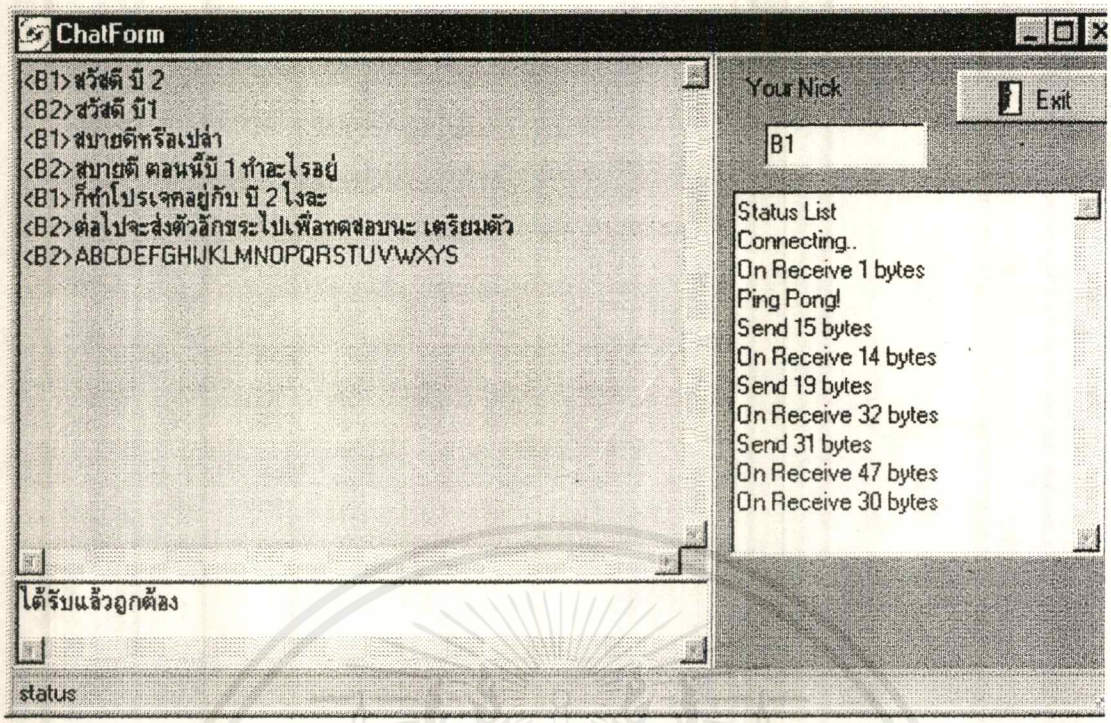
รูปที่ 4.6 รูปแสดงฟอร์ม YKMain.pas ทางด้านส่งเมื่อทำการรับ-ส่งไฟล์ชนิดต่าง ๆเรียบร้อยแล้ว

ในรูปที่ 4.7 เป็นรูปแสดงผลการทดลองรับส่งไฟล์ต่างๆทั้งไฟล์ภาพ,เสียงและตัวอักษรที่ปรากฏในฟอร์มของ YKMain.pas ทางด้านรับ

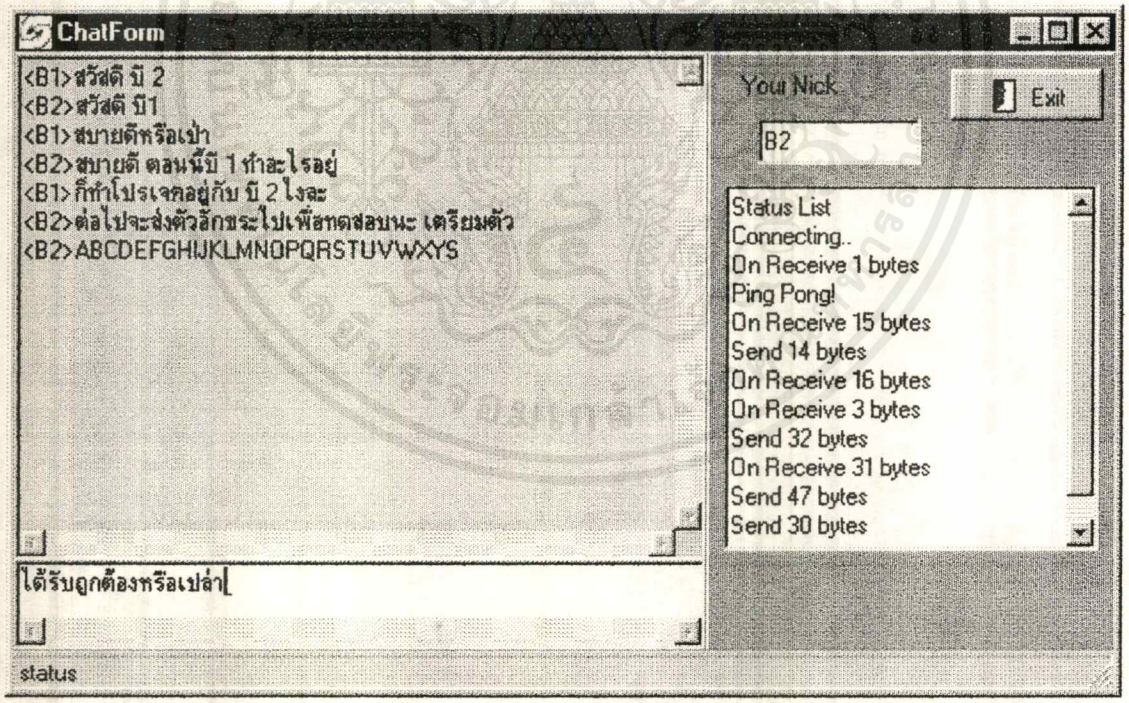
รูปที่ 4.7 รูปแสดงฟอร์ม YKMain.pas ทางด้านรับเมื่อทำการรับ-ส่งไฟล์ชนิดต่าง ๆ เรียบร้อยแล้ว

4.2 การทดลองรับ-ส่งข้อมูลด้วยการพิมพ์ข้อความโต้ตอบ

เมื่อผู้ใช้ทำการกดปุ่ม Chat ในฟอร์ม YKMain.pas จะเป็นการปิดฟอร์ม YKMain.pas แล้วทำให้มีการเปิดฟอร์ม YKChat.pas ขึ้น จากนั้นให้ผู้ใช้กำหนดชื่อผู้ใช้ในช่อง Your Nick และพิมพ์ข้อความต่าง ๆได้ โดยเมื่อมีการกด Enter หลังจากพิมพ์เสร็จ จะถือเป็นการส่งข้อมูลในแต่ละครั้ง จากรูปที่ 4.8 และรูปที่ 4.9เป็นรูปแสดงฟอร์ม YKChat.pas ทั้งสองด้าน



รูปที่ 4.8 รูปแสดงฟอร์ม YKChat.pas เมื่อมีการพิมพ์ข้อความโต้ตอบผู้ส่ง



รูปที่ 4.9 รูปแสดงฟอร์ม YKChat.pas เมื่อมีการพิมพ์ข้อความโต้ตอบผู้รับ

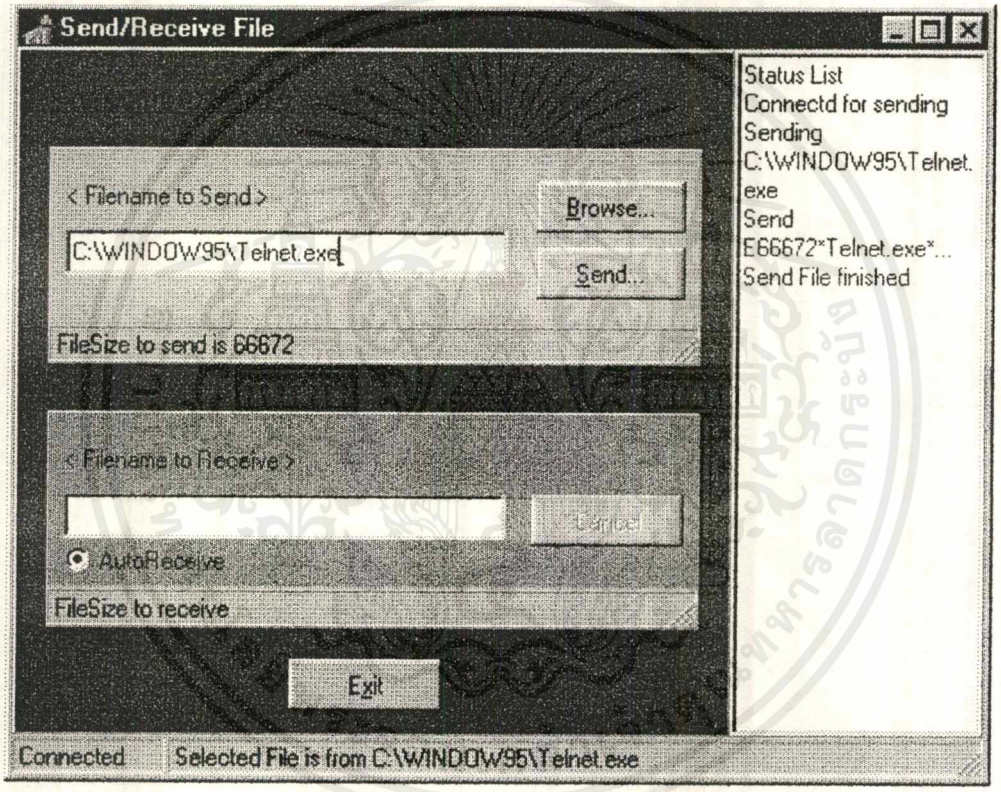
4.3 การทดลองรับ-ส่งไฟล์ข้อมูลชนิดต่าง ๆ เมื่อใช้ฟอร์ม YKSendFile.pas

เมื่อผู้ใช้ทำการกดปุ่ม SendFile ในฟอร์ม YKMain.pas จะเป็นการปิดฟอร์ม YKMain.pas แล้วทำให้มีการเปิดฟอร์ม YKSendFile.pas ขึ้น

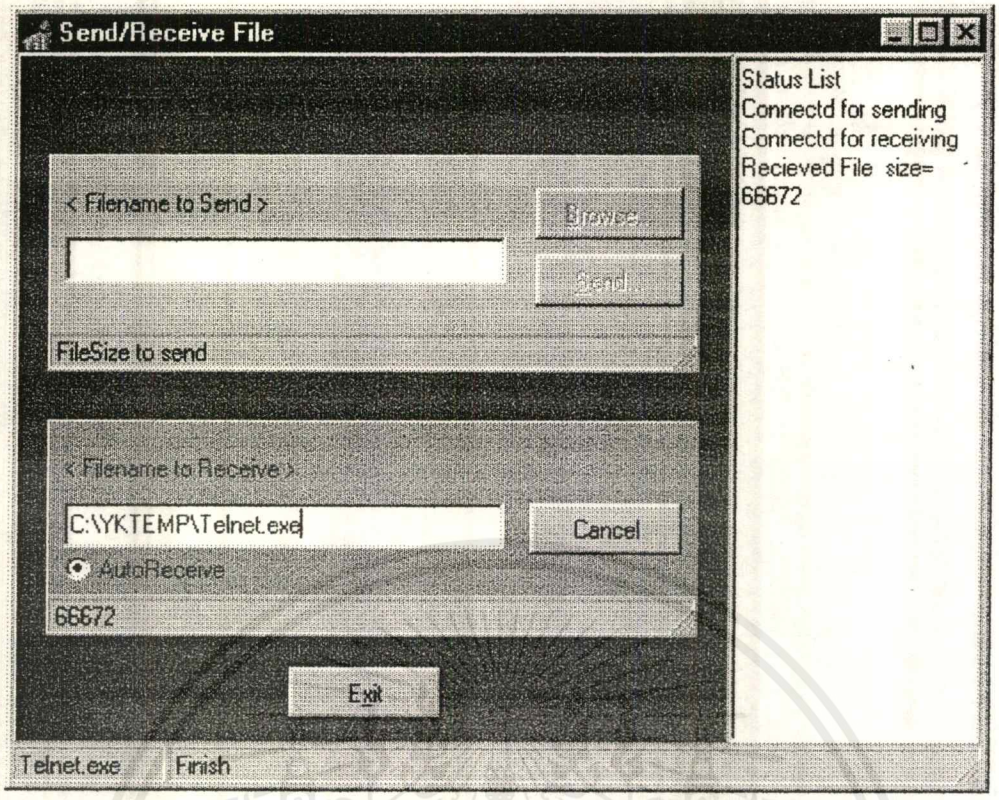
ทางด้านส่ง ถ้าผู้ใช้ต้องการส่งไฟล์ใด ๆ ให้ผู้ใช้กดปุ่ม Browse จะปรากฏ OpenDialog ขึ้นมา ให้ผู้ใช้เลือกชื่อไฟล์ที่ต้องการจะส่ง แล้วกดปุ่ม Send เพื่อส่งไฟล์นั้น

ทางด้านรับ เมื่อการรับไฟล์เสร็จสิ้นแล้ว ถ้าด้านผู้รับไม่ได้กดปุ่ม AutoReceive ไว้ โปรแกรมจะทำการปรากฏ SaveDialog ขึ้นมา เพื่อให้ผู้รับกำหนดชื่อไฟล์ที่ต้องการจัดเก็บเอง แต่ถ้าผู้รับกดปุ่ม AutoReceive โปรแกรมจะจัดเก็บไฟล์ที่ได้รับมาเป็นชื่อไฟล์ที่ด้านส่ง ส่งมานั่นเอง

ในรูปที่ 4.10 เป็นรูปแสดงฟอร์ม YKSendFile.pas ทางด้านส่ง

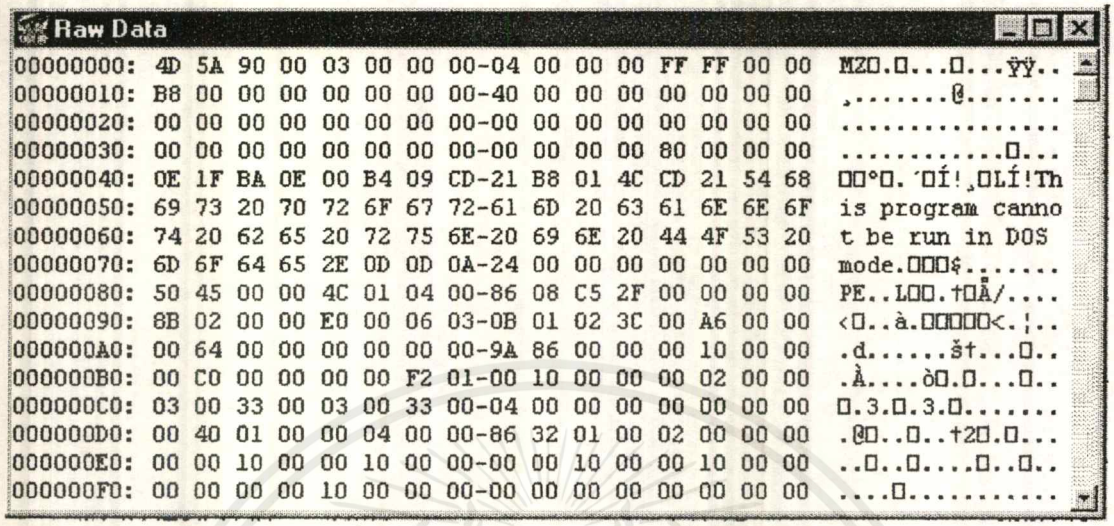


รูปที่ 4.10 รูปแสดงฟอร์ม YKSendFile.pas ทางด้านส่ง

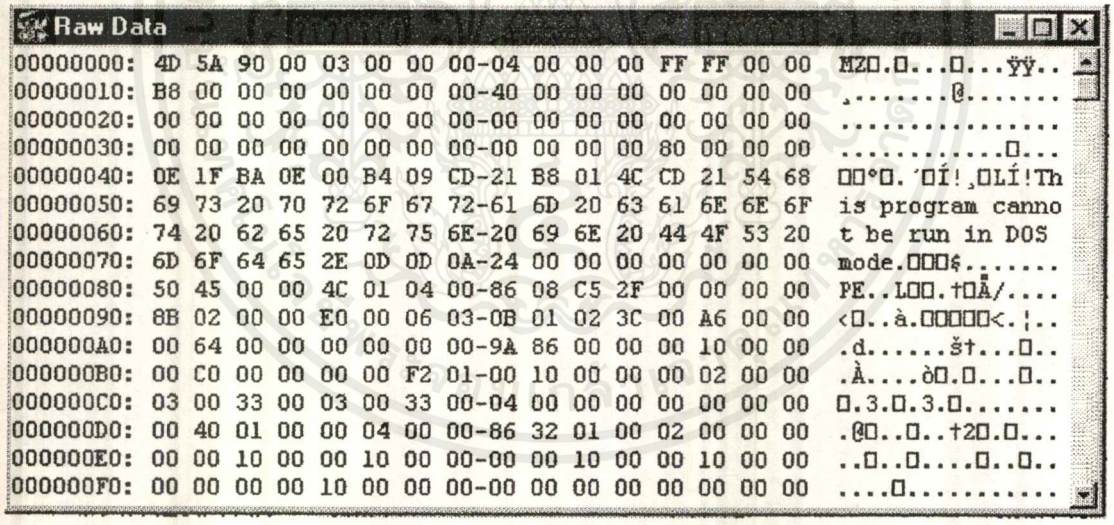


รูปที่ 4.11 รูปแสดงฟอร์ม YKSendFile.pas ทางด้านรับ

ผลจากการใช้โปรแกรมส่งไฟล์แล้ว ได้ทำการตรวจสอบข้อมูลในไฟล์ที่ด้านรับนั้น เปรียบเทียบกับไฟล์ทางด้านส่ง ปรากฏว่าสามารถรับข้อมูลได้ถูกต้อง แสดงผลได้ดังรูปที่ 4.12 และ รูปที่ 4.13

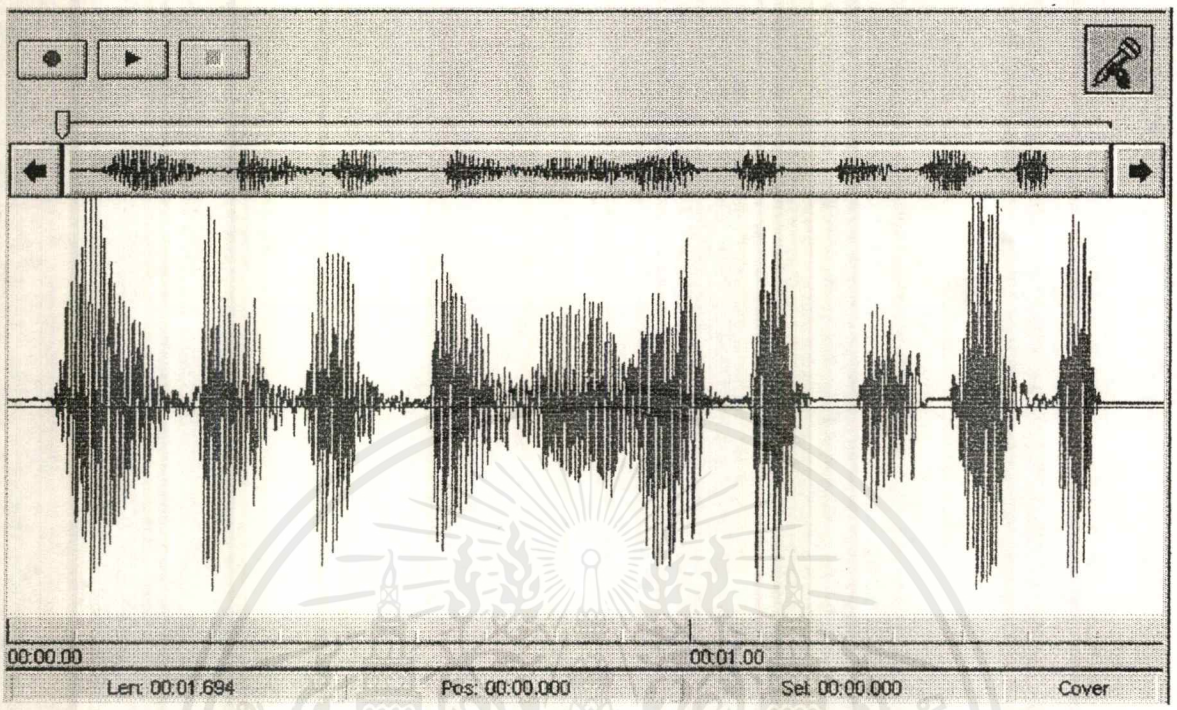


รูปที่ 4.12 ไฟล์ Telnet.exe ทางด้านผู้ส่ง

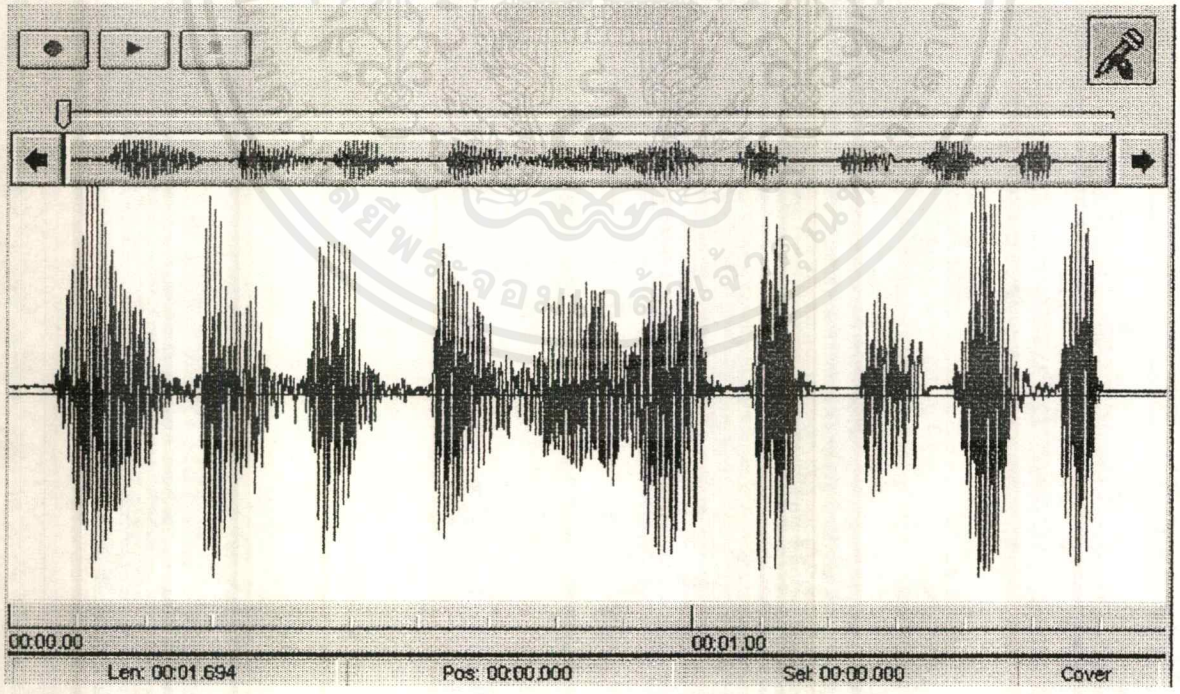


รูปที่ 4.13 ไฟล์ Telnet.exe ที่รับได้ทางด้านผู้รับ

ผลจากการใช้โปรแกรมส่งเสียงพูดแล้ว ได้ทำการลักษณะของไฟล์เสียงที่ด้านรับนั้น เปรียบเทียบกับไฟล์เสียงทางด้านส่ง ปรากฏว่าสามารถรับข้อมูลได้ถูกต้อง แสดงผลได้ดังรูปที่ 4.14 และ รูปที่ 4.15



รูปที่ 4.14 รูปแสดงกราฟเสียงของไฟล์เสียงที่ส่งจากทางด้านส่ง



รูปที่ 4.15 รูปแสดงกราฟเสียงของไฟล์เสียงที่ส่งจากทางด้านรับ

บทที่ 5
บทวิจารณ์และบทสรุป

สรุปผลการทดลอง

จากการทดลองส่งไฟล์จากเครื่องคอมพิวเตอร์ของผู้ใช้ต้นทาง ทั้งชนิดที่เป็นไฟล์ข้อความเอกสาร ไฟล์เสียง ไฟล์ภาพรวมถึงไฟล์ชนิดอื่น ๆ ไปยังเครื่องคอมพิวเตอร์ของผู้ใช้ปลายทาง พบว่า โปรแกรมสามารถส่งข้อมูลและรับข้อมูลได้อย่างถูกต้อง

ปัญหาที่พบจากการทดลอง

1. การติดต่อสื่อสารยังเป็นแบบฮาล์ฟดูเพล็กซ์ (Half Duplex) คือ ในขณะเวลาใดเวลาหนึ่งเครื่องคอมพิวเตอร์เครื่องหนึ่งสามารถเป็นผู้ส่งหรือผู้รับได้เพียงอย่างเดียว
2. ในการส่งไฟล์ที่มีขนาดใหญ่ จะทำได้ด้วยความเร็วไม่มาก จึงทำให้การติดต่อไม่ต่อเนื่องเท่าที่ควร
3. ไฟล์ที่ส่งไม่มีการบีบอัดมาก่อน ทำให้เสียเวลามากในการรับส่ง เมื่อส่งผ่านโมเด็มที่ไม่สนับสนุนการบีบอัดข้อมูล
4. การติดต่อผ่านโมเด็ม ยังไม่ได้ทดสอบกับโมเด็มรุ่นอื่น ๆ

แนวทางในการพัฒนา

1. ลดขนาดของข้อมูลที่ส่ง เพื่อเพิ่มความเร็วในการส่งข้อมูล โดยตัดส่วนของข้อมูลที่ไม่จำเป็น (ข้อมูลขยะ) ออก
2. พัฒนาในส่วนการบีบอัดไฟล์ ก่อนทำการส่ง

หนังสืออ้างอิง

1. Gary Cornell, Troy Strain, *Delphi Nut & Bolts for experienced programmers,* Berkeley,CA: McGraw-Hill, 1995.
2. Scott Jarol, Dan Haygood, Chris D. Coppola, *Delphi 2 Multimedia : adventure set,* Scottsdale, AZ: Coriolis Group Books, 1996.
3. กิตติ ภักดีวัฒนะกุล, Netscape (ALL - IN - ONE), บริษัท เคทีพี คอมพ์ แอนด์ คอนซัลท์ จำกัด, กรุงเทพฯ, 2540
3. ฉัตรชัย สุมามาลย์, การสื่อสารข้อมูลคอมพิวเตอร์และระบบเครือข่าย, บริษัท ด่านสุธาการพิมพ์, กรุงเทพฯ, 2535.
4. ยืน ภู่วรวรรณ, การสื่อสารข้อมูลและคอมพิวเตอร์เน็ตเวอร์ค, บริษัท ซีเอ็ดยูเคชั่น, กรุงเทพฯ, 2533.
5. จิรศักดิ์ เหลืองอุไร,คัมภีร์การใช้งานการสื่อสารอนุกรมบน PC ,บริษัท ซีเอ็ดดูเคชั่น,กรุงเทพฯ,2538.
6. จิรศักดิ์ เหลืองอุไร,นำทางสู่การเชื่อมโยงเครือข่าย, บริษัท ซีเอ็ดดูเคชั่น,กรุงเทพฯ,2538.

รายละเอียดโปรแกรมต่าง ๆ

โปรแกรม YKMainComm.pas

```
unit YKMainComm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, StdCtrls, ExtCtrls, Buttons, ComCtrls, ComDrv32, JpgImg,
MPlayer,MMsystem;

type
  TMainForm = class(TForm)
    MainMenu1: TMainMenu;
    Communication: TMenuItem;
    Chat1: TMenuItem;
    SendFile1: TMenuItem;
    Configuration1: TMenuItem;
    Communitcation1: TMenuItem;
    Help1: TMenuItem;
    Manual1: TMenuItem;
    About1: TMenuItem;
    N1: TMenuItem;
    Disconnect1: TMenuItem;
    StatusBar1: TStatusBar;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    Memo1: TMemo;
    GroupBox3: TGroupBox;
    StopBtn: TBitBtn;
    SpeakBtn: TBitBtn;
    SendPic: TBitBtn;
    SelectPic: TBitBtn;
    GroupBox4: TGroupBox;
    GroupImage: TGroupBox;
    JImage2: TJPEGImage;
    JImage1: TJPEGImage;
    DocMemo: TMemo;
    other: TGroupBox;
    ChatBtn: TBitBtn;
    SendFileBtn: TBitBtn;
    ExitBtn: TBitBtn;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    CommPort1: TCommPortDriver;
    CommPort2: TCommPortDriver;
    PopupMenu1: TPopupMenu;
    Minimize1: TMenuItem;
    Maximize1: TMenuItem;
    Minimize2: TMenuItem;
    TrueSize1: TMenuItem;
    FitToPage1: TMenuItem;
    PopupMenu2: TPopupMenu;
    SaveAs1: TMenuItem;
    Minimize3: TMenuItem;
    Maximize2: TMenuItem;
    MediaPlayer1: TMediaPlayer;
    TrueSize2: TMenuItem;
    FitToPage2: TMenuItem;
    Panel1: TPanel;
    SendTBtn: TBitBtn;
    LoadTBtn: TSpeedButton;
    SaveTBtn: TBitBtn;
    ResetBtn: TBitBtn;
    SpeedButton1: TSpeedButton;
    LisAgn: TBitBtn;
```

```
    Timer1: TTimer;
    procedure ChatBtnClick(Sender: TObject);
    procedure SendFileBtnClick(Sender: TObject);
    procedure SendFile1Click(Sender: TObject);
    procedure Communitcation1Click(Sender: TObject);
    procedure About1Click(Sender: TObject);
    procedure CommPort1ReceiveData(Sender: TObject; DataPtr: Pointer;
      DataSize: Integer);
    procedure FormCreate(Sender: TObject);
    procedure QuitClick(Sender: TObject);
    procedure CommPort2ReceiveData(Sender: TObject; DataPtr: Pointer;
      DataSize: Integer);
    procedure SelectPicClick(Sender: TObject);
    procedure JImage1DblClick(Sender: TObject);
    procedure SendPicClick(Sender: TObject);
    procedure SpeakBtnClick(Sender: TObject);
    procedure StopBtnClick(Sender: TObject);
    procedure ExitBtnClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure JImage1Click(Sender: TObject);
    procedure LoadTBtnClick(Sender: TObject);
    procedure SaveTBtnClick(Sender: TObject);
    procedure Minimize1Click(Sender: TObject);
    procedure Maximize1Click(Sender: TObject);
    procedure Minimize2Click(Sender: TObject);
    procedure TrueSize1Click(Sender: TObject);
    procedure FitToPage1Click(Sender: TObject);
    procedure Minimize3Click(Sender: TObject);
    procedure Maximize2Click(Sender: TObject);
    procedure SaveAs1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure Chat1Click(Sender: TObject);
    procedure SendTBtnClick(Sender: TObject);
    procedure TrueSize2Click(Sender: TObject);
    procedure FitToPage2Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Manual1Click(Sender: TObject);
    procedure ResetBtnClick(Sender: TObject);
    procedure DocMemoChange(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure LisAgnClick(Sender: TObject);

  private
    { Private declarations }
    packet,Lpacket,ToImageName,FromImageSize,ToFImageSize:String;
    FromImageF,ToImageF:File;
    PacketAr:Array[1..1024] of Char;
    Index:Integer;
    ToSoundName,FromSoundSize,ToFSoundSize:String;
    FromSoundF,ToSoundF:File;
    ToTextName,FromTextSize,ToFTextSize:String;
    FromTextF,ToTextF:File;
    ExtpicF,TextFilename,ImageFilename:string;
    LNum:Integer;
    PlayF:File;
    {Variable for Calling Fn in MMsystem }
    InNumDevs,OutNumDevs,InDeviceId,OutDeviceId,INuSize,OutuSize:UINT;
    InlpCaps,OutlpCaps:PWaveOutCaps;
    Inhwo,Outhwo:HWAVEOUT;
    InlpdwVolume,OutlpdwVolume:PDWORD;
    IndwVolume,OutdwVolume:DWORD;
    InhWaveIn,OuthWaveOut:HWAVEOUT;
    InlpuDeviceID,OutlpuDeviceID:PUINT;
    OutlphWaveOut:PHWaveOut;
    OutlpFormat:PWaveFormatEx;
    OutdwCallback,OutdwInstance,Outdwlags:DWORD;

    procedure SendPacket(const Value:Char);
    procedure Sendreq(const Value:Char);
    procedure SendFirstPacket(const H:Char;var SendF:String);
    procedure Continue(const Prev:Char);
    procedure WaitFreeBf;
```

```
  public
    { Public declarations }

SoundSending,ImageSending,TextSending,PacketSending,Waiting,Inserting:Boolean;
    PlayFName:String;
    PlayN1:Boolean;

  end;
const  FpacketSize=1024;
       WorkingDir='c:\YKTemp';
       TimeC=1000;
       LoopC=3;
var

  MainForm: TMainForm;


implementation

uses YKConnect,YKchat,YKPConfig,YKsendFile, YKAbout, YKManual;

{$R *.DFM}

procedure TMainForm.ChatBtnClick(Sender: TObject);
var     Loop:Integer;
begin
        CommPort1.SendString('C');
        CommPort1.Disconnect;
        ChatForm.ChatCommPort1.connect;
        ChatForm.ShowModal;
end;

procedure TMainForm.SendFileBtnClick(Sender: TObject);
var     Loop:Integer;
begin
        CommPort1.SendString('F');
        CommPort1.Disconnect;
        SRFileForm1.SRFileCommPort2.connect;
        SRFileForm1.ShowModal;
end;

procedure TMainForm.Chat1Click(Sender: TObject);
begin
     ChatBtn.Click;
end;

procedure TMainForm.SendFile1Click(Sender: TObject);
begin
      SendFileBtn.Click;
end;

procedure TMainForm.Communitcation1Click(Sender: TObject);
begin
     ConfigForm.ApplyComBtn.enabled:=false;
     ConfigForm.ShowModal;
end;

procedure TMainForm.About1Click(Sender: TObject);
begin
     AboutForm.ShowModal;
end;

procedure TMainForm.CommPort1ReceiveData(Sender: TObject; DataPtr: Pointer;
  DataSize: Integer);
var RxDataPtr:Pchar;
    Id,NCheck:Integer;
    FileStr:String;
begin
   {Receive Header Data  }
   RxDataPtr:=DataPtr;
   PacketSending:=false;CommPort1.FlushBuffers(true,true);
```

```
    Memo1.Lines.Add(RxDataPtr^);
                  case RxDataPtr^ of
                  { Chat Connection Request}
                  'C': begin
                         CommPort1.Disconnect;
                         ChatForm.ChatCommPort1.Connect;
                         ChatForm.ShowModal;
                       end;
                  { File Transfer Request }
                  'F': begin
                         CommPort1.Disconnect;
                         SRFileForm1.SRFileCommPort2.connect;
                         SRFileForm1.ShowModal;
                       end;
                  { Exit Request }
                  'X': begin
                         CommPort1.SendString('X');
                         CommPort1.Disconnect;
                         Close;
                       end;
                  { Image Recieving Request}
                  'I': begin
                       SendReq('i');
                       end;
                  { Image Sending  Request}
                  'i': begin
                       FileStr:=Imagefilename;
                       SendFirstPacket('i',FileStr);
                       end;
                  { Image Data Sending Request}
                  'm': begin
                       SendPacket('m');
                       end;
                  { Sound Recieving Request }
                  'S': begin
                       SendReq('s');
                       end;
                  { Sound Sending Request }
                  's': begin
                       FileStr:='c:\YKTemp\fsound.wav';
                       SendFirstPacket('s',FileStr);
                       end;
                  { Sound Data Sending Request}
                  'q': begin
                       SendPacket('q');
                       end;
                  { Text Recieving Request }
                  'T': begin
                       SendReq('t');
                       end;
                  { Text Sending Request }
                  't': begin
                       FileStr:='c:\YKTemp\Text.txt';
                       SendFirstPacket('t',FileStr);
                       end;
                  { Text Data Sending Request}
                  'u': begin
                       SendPacket('u');
                       end;
                  'z': begin
                         with Statusbar1 do
                         Panels.Items[1].Text := 'Nothing Sending';
                         ResetBtn.Click;
                       end;
                 end;

  end;


  procedure SendErrorPacket;
  begin
  end;
```

```
procedure TMainForm.FormCreate(Sender: TObject);
var     Wv:file;
        sHead:array[0..59] of char;
        Nwrite:Integer;

begin
        if not CreateDir('c:\YKTemp\') then
        MessageDlg('Can not Create Directory C:\YKTemp\ Maybe Existing or Disk
Full,if not have it should Exit ?',
         mtInformation,[mbOk], 0);
        with Statusbar1 do
           Panels.Items[0].Text := 'Connected';
        AssignFile(Wv,'c:\YKTemp\sound.wav');
        rewrite(Wv,1);
        sHead:='RIFF4'+#0+#0+#0+'WAVEfmt '+
        #18+#0+#0+#0+#1+#0+#1+#0+  #64+#31+#0+#0+#64+#31+#0+#0+
        #1+#0+#8+#0+#0+#0+#0+#102+#97+  #99+#116+#4+#0+#0+#0+#0+#0+
        #0+#0+#100+#97+#116+#97+#1+#0+  #0+#0+#108+#0;
        blockwrite(Wv,sHead,sizeof(SHead),Nwrite);
        closeFile(Wv);
        SoundSending:=false;ImageSending:=false;TextSending:=false;
        PacketSending:=false;Waiting:=false;Inserting:=false;
        Lnum:=LoopC;Index:=0;
end;

procedure TMainForm.QuitClick(Sender: TObject);
begin
        close;
end;

procedure TMainForm.CommPort2ReceiveData(Sender: TObject; DataPtr: Pointer;
  DataSize: Integer);
var RxDataPtr,WavePtr:Pchar;
    PosId,NCheck,NWrite:Integer;
    Head:Array[0..100] of Char;
    PacketStr:String;
    Label RenameWaitLabel,PlayWaitLabel,LPlayWaitLabel;

begin
 {Timer start}
 if not Timer1.enabled then Timer1.enabled:=True;
 {Receive Data until 1 Packet }
   RxDataPtr:=DataPtr;WavePtr:=DataPtr;inc(WavePtr);
   if length(Packet)<FpacketSize then
        begin
        while (DataSize>0) and (length(Packet)<FpacketSize) do
                begin
                {Fill Each Character to Packet   }
                Packet:=Packet+RxDataPtr^;
                if length(Packet)>1 then PacketAr[Index]:=RxDataPtr^;
                inc(Index);
                inc(RxDataPtr);
                dec(DataSize);
                end;
        while DataSize>0 do
                begin
                LPacket:=LPacket+RxDataPtr^;
                inc(RxDataPtr);
                dec(DataSize);
                end;
        {Receive One Packet}
        if length(Packet)=FpacketSize then
                begin
                CommPort2.Flushbuffers(true,true);Index:=0;
                PosId:=2;Memo1.Lines.Add('Receive '+Packet[1]+' Packet');
                case Packet[1] of
                'i':begin
                      {Kill Timer}
                      Timer1.enabled:=false;Timer1.interval:=TimeC;
                      ImageSending:=True;
                      While Packet[PosId]<>'=' do
```

```
                        begin
                        ToFImageSize:=ToFImageSize+Packet[PosId];
                        inc(PosId);
                      end;
                      inc(PosId);
                      While Packet[PosId]<>'*' do
                        begin
                        ToImageName:=ToImageName+Packet[PosId];
                        inc(PosId);
                        end;

AssignFile(ToImageF,'c:\YKTemp\YKFile'+ExtractFileExt(ToImageName));
                      {$I-}Rewrite(ToImageF,1){$I+};
                      If IOResult<>0 then begin Memo1.Lines.Add('Error Assign
ImageFile');;ResetBtn.Click;end;
                      While (PosId<FpacketSize) and
(FileSize(ToImageF)<StrToInt(ToFImageSize)) do
                        begin
                        inc(PosId);
                        BlockWrite(ToImageF,Packet[PosId],1,Ncheck);
                        end;
                     Memo1.Lines.Add(ToImageName);
                     if PosId<FpacketSize then
                      begin
                      CloseFile(ToImageF);
DeleteFile('c:\YKTemp\YKDPlay'+ExtractFileExt(ToImageName));
                      if not
RenameFile('c:\YKTemp\YKFile'+ExtractFileExt(ToImageName),
                      'c:\YKTemp\YKDPlay'+ExtractFileExt(ToImageName)) then
                      with Statusbar1 do
                         Panels.Items[1].Text := 'Error renaming file!';
                      Packet:='';ToFImageSize:='';ImageSending:=False;
                      CommPort2.SendString('n');Continue('m');
                      ExtpicF:=ExtractFileExt(ToImageName);ToImageName:='';
                      if ExtpicF='.jpg' then
JImage2.Filename:='c:\YKTemp\YKDPlay.jpg';
                      if ExtpicF='.bmp' then
JImage2.Picture.LoadFromFile('c:\YKTemp\YKDPlay.bmp');
                      if ExtpicF='.avi' then
                        begin MediaPlayer1.Filename:='c:\YKTemp\YKDPlay.avi';
                          MediaPlayer1.open;MediaPlayer1.visible:=true;
                          MediaPlayer1.Play;
                        end;
                      end
                     else
                     begin
                     Packet:='';Continue('m');
                     end;
                    end;
                 's':begin
                      {Kill Timer}
                      Timer1.enabled:=false;Timer1.interval:=TimeC;
                      SoundSending:=True;
                      if not SoundSending then SoundSending:=true;
                      While Packet[PosId]<>'*' do
                        begin
                        ToFSoundSize:=ToFSoundSize+Packet[PosId];
                        inc(PosId);
                        end;
                      inc(PosId);
                      While Packet[PosId]<>'*' do
                        begin
                        ToSoundName:=ToSoundName+Packet[PosId];
                        inc(PosId);
                        end;
                      AssignFile(ToSoundF,'c:\YKTemp\YKFile.wav');
                      {$I-}Rewrite(ToSoundF,1){$I+};
                      If IOResult<> 0 then begin Memo1.Lines.Add('Error Assign
SoundFile');ResetBtn.Click;end;
                      While (PosId<FpacketSize) and
(FileSize(ToSoundF)<StrToInt(ToFSoundSize)) do
                        begin
```

```
                    inc(PosId);
                    BlockWrite(ToSoundF,Packet[PosId],1,Ncheck);
                    end;
                  if PosId<FpacketSize then
                  begin
                  CloseFile(ToSoundF);DeleteFile('c:\YKTemp\YKDPlay.wav');
                  if not RenameFile('c:\YKTemp\YKFile.wav',
                  'c:\YKTemp\YKDPlay.wav') then
                  with Statusbar1 do
                      Panels.Items[1].Text := 'Error renaming file!';
                  Packet:='';ToFSoundSize:='';SoundSending:=False;
                  Continue('q');
                  if not sndPlaySound('c:\YKTemp\YKDPlay.wav', SND_ASYNC) then
                      StatusBar1.Panels.Items[1].Text:= 'ERROR With Playing';
                  end
                  else
                  begin
                  Packet:='';Continue('q');
                  end;
                   end;
              't':begin
                    {Kill Timer}
                    Timer1.enabled:=false;Timer1.interval:=TimeC;
                    TextSending:=True;
                    While Packet[PosId]<>'*' do
                      begin
                      ToFTextSize:=ToFTextSize+Packet[PosId];
                      inc(PosId);
                      end;
                    inc(PosId);
                    While Packet[PosId]<>'*' do
                      begin
                      ToTextName:=ToTextName+Packet[PosId];
                      inc(PosId);
                      end;
                    AssignFile(ToTextF,'c:\YKTemp\YKFile.txt');
                    {$I-}Rewrite(ToTextF,1){$I+};
                    If IOResult<>0 then begin Memo1.Lines.Add('Error Assign
TextFile');ResetBtn.Click;end;
                     While (PosId<FpacketSize) and
(FileSize(ToTextF)<StrToInt(ToFTextSize)) do
                      begin
                      inc(PosId);
                      BlockWrite(ToTextF,Packet[PosId],1,Ncheck);
                      end;
                  if PosId<FpacketSize then
                  begin
                  CloseFile(ToTextF);DeleteFile('c:\YKTemp\YKDPlay.txt');
                  if not RenameFile('c:\YKTemp\YKFile.txt',
                  'c:\YKTemp\YKDPlay.txt') then
                   with Statusbar1 do
                      Panels.Items[1].Text := 'Error renaming file!';
                  Packet:='';ToFTextSize:='';TextSending:=False;
                  Continue('u');
                  DocMemo.Lines.LoadFromFile('c:\YKTemp\YKDPlay.txt')
                  end
                  else
                  begin
                  Packet:='';Continue('u');
                  end;
                  end;
              'm':begin
                  {Kill Timer}
                   Timer1.enabled:=false;Timer1.interval:=TimeC;
                   BlockWrite(ToImageF,PacketAr,FpacketSize-1,NCheck);
                   Packet:='';Continue('m');
                  end;
              'q':begin
                  {Kill Timer}
                  Timer1.enabled:=false;Timer1.interval:=TimeC;
                  if Lnum=LoopC then
                     begin
```

```
                        PlayN1:=not PlayN1;
                        If PlayN1 then  PlayFName:='c:\YKTemp\Play.wav' else
                            PlayFName:='c:\YKTemp\CPlay.wav';

                        AssignFile(PlayF,PlayFName);
                        Rewrite(PlayF,1);
                        Head:='RIFF'+#$30+#$0C+#0+#0+'WAVEfmt '+
                        #18+#0+#0+#0+#1+#0+#1+#0+  #64+#31+#0+#0+#64+#31+#0+#0+
                        #1+#0+#8+#0+#0+#0+#0+#102+#97+  #99+#116+#4+#0+#0+#0+#0+#0+
                        #0+#0+#100+#97+#116+#97+#$FD+#$0B+  #0+#0
                        +#150+#90+#150+#100+#150+#100+#150+#100+#150+#100;
                        blockwrite(PlayF,Head,sizeof(Head),Nwrite);
                        BlockWrite(ToSoundF,PacketAr,FpacketSize-1,NCheck);
                        BlockWrite(PlayF,PacketAr,FpacketSize-11,NCheck);
                        end
                    else
                        begin

                        BlockWrite(ToSoundF,PacketAr,FpacketSize-1,NCheck);
                        BlockWrite(PlayF,PacketAr,FpacketSize-1,NCheck);
                        inc(PosId);
                        end;
                         Lnum:=Lnum-1;
                    if Lnum=0 then
                        begin
                        {$I-}CloseFile(PlayF);{$I+}
                        PlayWaitLabel:If PlayN1 then
                           begin  if not sndPlaySound('c:\YKTemp\Play.wav',
SND_ASYNC) then
                                       Goto PlayWaitLabel
                           end else  if not sndPlaySound('c:\YKTemp\CPlay.wav',
SND_ASYNC) then
                                       Goto PlayWaitLabel ;

                         Lnum:=LoopC;
                        end;
                        Packet:='';Continue('q');
                    end;
                'u':begin
                    {Kill Timer}
                    Timer1.enabled:=false;Timer1.interval:=TimeC;
                    BlockWrite(ToTextF,PacketAr,FpacketSize-1,NCheck);
                    Packet:='';Continue('u');
                    end;
                'n':begin
                    {Kill Timer}
                    Timer1.enabled:=false;Timer1.interval:=TimeC;
                    While (StrToInt(ToFImageSize)>FileSize(ToImageF)) do
                        begin
                        BlockWrite(ToImageF,Packet[PosId],1,NCheck);
                        inc(PosId);
                        end;
                     Memo1.Lines.Add('Recieved File '+' ... '+' size= '+
                     IntToStr(FileSize(ToImageF)));
                     CloseFile(ToImageF);ImageSending:=False;
                     Continue('m');

Packet:='';ToFImageSize:='';DeleteFile('c:\YKTemp\YKDPlay'+ExtractFileExt(ToImage
Name));
                     if not
RenameFile('c:\YKTemp\YKFile'+ExtractFileExt(ToImageName),
                      'c:\YKTemp\YKDPlay'+ExtractFileExt(ToImageName)) then
                      with Statusbar1 do
                         Panels.Items[1].Text := 'Error renaming file!';
                     ExtpicF:=ExtractFileExt(ToImageName);ToImageName:='';
                    if ExtpicF='.jpg' then
JImage2.Filename:='c:\YKTemp\YKDPlay.jpg';
                    if ExtpicF='.bmp' then
JImage2.Picture.LoadFromFile('c:\YKTemp\YKDPlay.bmp');
                    if ExtpicF='.avi' then
                    begin MediaPlayer1.Filename:='c:\YKTemp\YKDPlay.avi';
                         MediaPlayer1.open;MediaPlayer1.visible:=true;
```

```
                        MediaPlayer1.Play;
                    end;
                    end;
                'r':begin
                    {Kill Timer}
                    Timer1.enabled:=false;Timer1.interval:=TimeC;
                    While (StrToInt(ToFSoundSize)>FileSize(ToSoundF)) do
                        begin
                        BlockWrite(ToSoundF,Packet[PosId],1,NCheck);
                        inc(PosId);
                        end;
                     Memo1.Lines.Add('Recieved File '+' ... '+' size= '+
                     IntToStr(FileSize(ToSoundF)));
                     CloseFile(ToSoundF);DeleteFile('c:\YKTemp\YKDPlay.wav');
                     Packet:='';ToFSoundSize:='';SoundSending:=False;
                     Continue('q');
                     if Lnum<>LoopC then CloseFile(PlayF);Lnum:=LoopC;
                     if not RenameFile('c:\YKTemp\YKFile.wav',
                     'c:\YKTemp\YKDPlay.wav') then
                     with Statusbar1 do
                         Panels.Items[1].Text := 'Error renaming file!';
                     end;
                'v':begin
                    {Kill Timer}
                    Timer1.enabled:=false;Timer1.interval:=TimeC;
                    While (StrToInt(ToFTextSize)>FileSize(ToTextF)) do
                        begin
                        BlockWrite(ToTextF,Packet[PosId],1,NCheck);
                        inc(PosId);
                        end;
                     Memo1.Lines.Add('Recieved File '+' ... '+' size= '+
                     IntToStr(FileSize(ToTextF)));
                     CloseFile(ToTextF);DeleteFile('c:\YKTemp\YKDPlay.txt');

Packet:='';ToFTextSize:='';TextSending:=False;SendTBtn.enabled:=true;
                     Continue('u');
                     if not RenameFile('c:\YKTemp\YKFile.txt',
                     'c:\YKTemp\YKDPlay.txt') then
                      with Statusbar1 do
                         Panels.Items[1].Text := 'Error renaming file!';
                     DocMemo.Lines.LoadFromFile('c:\YKTemp\YKDPlay.txt');
                     end;
                'X' : begin
                      {Kill Timer}
                      Timer1.enabled:=false;Timer1.interval:=TimeC;
                      CommPort2.DisConnect;
                      close;
                      end
                else
                Timer1.enabled:=true;
                end;

          end;
    end;

end;

procedure TMainForm.SelectPicClick(Sender: TObject);
begin
        with statusbar1 do
             Panels.Items[1].Text := ' ';
        With OpenDialog1 do
             begin Filter := 'Bitmap Image File (*.bmp)|*.bmp|JPeg Image File
(*.jpg)' +
     '|*.jpg|AVI File (*.avi)|*.avi';
             end;
        if OpenDialog1.Execute then
                begin
                 ImageFilename:=OpenDialog1.Filename;
                 ExtpicF:=ExtractFileExt(OpenDialog1.Filename);
                 if ExtpicF='.jpg' then begin
                                    JImage1.Filename:=ImageFilename;
```

```
      MediaPlayer1.StartRecording;


end;

procedure TMainForm.StopBtnClick(Sender: TObject);
var FileStr:String;
    ReturnStr:Array[0..255] of Char;
begin
      if StopBtn.enabled then
      begin
            SpeakBtn.enabled:=True;
            StopBtn.enabled:=False;
            Waiting:=true;
            with MediaPlayer1 do
              begin
              filename:='c:\YKTemp\fsound.wav';
              Stop;
              save;
              close;
              end;
              {mciSendString('stop wave',ReturnStr,256,0);
              mciSendString('save wave c:\YKTemp\fsound.wav',ReturnStr,256,0);
              mciSendString('close wave',ReturnStr,256,0);  }
      end;

           begin
           Waiting:=false;
            if ((not ImageSending) and (not SoundSending) and (not TextSending))
            then  CommPort1.Sendstring('S')
            else
              begin
              FileStr:='c:\YKTemp\fsound.wav';
              Inserting:=True;
              SendFirstPacket('s',FileStr);
              SoundSending:=True;{ Sending Sound status }
              end;
            with StatusBar1 do
            Panels.Items[1].Text:= 'Stop speaking.';
            SoundSending:=True;{ Sending Sound status }

           end;
      end;


procedure TMainForm.ExitBtnClick(Sender: TObject);
begin
     CommPort1.SendString('X');
     close;
end;

procedure TMainForm.FormShow(Sender: TObject);
begin
  ConnectForm.Showmodal;
end;

procedure TMainForm.JImage1Click(Sender: TObject);
begin
     if JImage1.Align=alClient then
    with JImage1 do
    begin Align:=alNone   ;
          Height:=JImage2.Height div 3;
          Top:=JImage2.Top;
          Left:=JImage2.Left;
          Width:=JImage2.Width div 3;
    end
    else JImage1.Align:=alClient;
end;

procedure TMainForm.LoadTBtnClick(Sender: TObject);
begin
    With OpenDialog1 do
```

```
                                        MessageDlg('If Error in Received this
Picture ,it will Close sudden !!',
                                        mtInformation,[mbOk], 0);
                                        end;
                  if ExtpicF='.bmp' then
JImage1.Picture.LoadFromFile(ImageFilename);
                  if ExtpicF='.avi' then
                     begin MediaPlayer1.Filename:=ImageFilename;
                           MediaPlayer1.Visible:=true;MediaPlayer1.Open;
                     end;
               end;
        with statusbar1 do
        Panels.Items[1].Text := 'Selected Pic. is from '+
OpenDialog1.FileName+ExtpicF;
end;

procedure TMainForm.JImage1DblClick(Sender: TObject);
begin
     MainForm.SelectPic.click;
end;

procedure TMainForm.SendPicClick(Sender: TObject);
var FileStr:String;
begin
      if SendPic.enabled then
      begin
      SendPic.enabled:=false;
      Waiting:=true;
      end;
      begin
      Waiting:=false;
      if ((ImageSending=false) and(SoundSending=false) and(TextSending=false))
then
           CommPort1.Sendstring('I')
      else
       begin
       {Sending Between the other Sending }
       FileStr:=Imagefilename;
       Inserting:=True;
       SendFirstPacket('i',FileStr);

       end;
      with StatusBar1 do
        Panels.Items[1].Text:= 'Sending Picture';
        ImageSending:=true;{ Sending Image status }

    end;
end;

procedure TMainForm.SpeakBtnClick(Sender: TObject);
var ReturnStr:Array[0..255] of Char;
     Ret:String;
begin
     { **Use these commands but format in 11025 Hz**
     mciSendString('Open New type WaveAudio alias wave',ReturnStr,256,0);
     Memo1.Lines.Add(ReturnStr);
     mciSendString('set wave samplespersec 11025',ReturnStr,256,0);
     Memo1.Lines.Add(ReturnStr);
     mciSendString('set wave bitpersample 8',ReturnStr,256,0);
     mciSendString('set wave channels 1',ReturnStr,256,0);
     mciSendString('record wave',ReturnStr,256,0); }
      SpeakBtn.Enabled:=False;
      StopBtn.Enabled:=True;
      With MediaPlayer1 do
      begin
             filename:='c:\YKTemp\sound.wav';
             open;
             position:=0;
      end;

      with StatusBar1 do
        Panels.Items[1].Text:= 'Lets speak.';
```

```
        begin Filter := 'Text File (*.txt)|*.txt|All File (*.*)' +
    '|*.*';
        end;
    SendTBtn.enabled:=true;
    if OpenDialog1.Execute then
        DocMemo.Lines.LoadFromFile(OpenDialog1.filename);
end;

procedure TMainForm.SaveTBtnClick(Sender: TObject);
begin
     With SaveDialog1 do
         begin Filter := 'Text File (*.txt)|*.txt|All File (*.*)' +
    '|*.*';
         end;
    if SaveDialog1.execute then
        DocMemo.Lines.SaveToFile(SaveDialog1.filename);
        TextFilename:=SaveDialog1.filename;
end;

procedure TMainForm.Minimize1Click(Sender: TObject);
begin
    with JImage1 do
    begin Align:=alNone    ;
          Height:=GroupImage.Height div 3;
          Top:=GroupImage.Top;
          Left:=GroupImage.Left;
          Width:=GroupImage.Width div 3;
    end
end;

procedure TMainForm.Maximize1Click(Sender: TObject);
begin
 JImage1.Align:=alClient;
end;

procedure TMainForm.Minimize2Click(Sender: TObject);
begin
    with JImage1 do
    begin Align:=alNone    ;
          Height:=GroupImage.Height div 3;
          Top:=(GroupImage.Top+(2*Height));
          Left:=GroupImage.Left;
          Width:=GroupImage.Width div 3;
    end
end;

procedure TMainForm.TrueSize1Click(Sender: TObject);
begin
     JImage1.Stretch:=False;
end;

procedure TMainForm.FitToPage1Click(Sender: TObject);
begin
     JImage1.Stretch:=True;
end;

procedure TMainForm.Minimize3Click(Sender: TObject);
begin
 with JImage2 do
    begin Align:=alNone    ;
          Height:=GroupImage.Height div 3;
          Top:=GroupImage.Top;
          Width:=GroupImage.Width div 3;
          Left:=(GroupImage.Left+GroupImage.Width)-Width;
    end
end;

procedure TMainForm.Maximize2Click(Sender: TObject);
begin
     JImage2.Align:=alClient;
end;
```

```
procedure TMainForm.SaveAs1Click(Sender: TObject);
begin
    SaveDialog1.Filter := 'Bitmap File (*.bmp)|*.bmp';
    if SaveDialog1.execute then JImage2.Picture.SavetoFile(SaveDialog1.Filename);
end;

procedure TMainForm.SpeedButton2Click(Sender: TObject);
var ExtpicF:String;
begin


end;

procedure TMainForm.SendTBtnClick(Sender: TObject);
var FileStr:String;
begin
      if SendTBtn.enabled then
      begin
           Waiting:=true;
           DocMemo.Lines.SaveToFile('c:\YKTemp\TEXT.TXT');
      end ;
      SendTBtn.enabled:=false;
     if ((ImageSending=false) and(SoundSending=false) and(TextSending=false))
     then CommPort1.Sendstring('T')
     else begin
          FileStr:='c:\YKTemp\Text.txt';
          Inserting:=True;
          SendFirstPacket('t',FileStr);
          end;
     TextSending:=True;{ Sending Text status }
     Waiting:=false;
end;

procedure TMainForm.SendPacket(const Value: Char);
var MN_read,FN_read,NData:Integer;
    TxDataPtr,DataBuffPtr:PChar;
    DataBuff:Array[0..FpacketSize-1] of Char;
    EndChar:Char;
    DataString:String;
begin
     {Loop Wait free out buffer}
     case Value of
     {Change Header }
     'm':begin
         EndChar:='n';
         CommPort1.FlushBuffers(true,true);
         DataBuff:='';
         {$I-}Blockread(FromImageF,DataBuff,FpacketSize-1,MN_read){$I+};
         If IOResult<>0 then Memo1.Lines.Add('Error Read FromImageF');
         end;
     'q':begin
         EndChar:='r';
         CommPort1.FlushBuffers(true,true);
         DataBuff:='';
         {$I-}Blockread(FromSoundF,DataBuff,FpacketSize-1,MN_read){$I+};
         If IOResult<>0 then Memo1.Lines.Add('Error Read FromImageF');
         end;
     'u':begin
         EndChar:='v';
         CommPort1.FlushBuffers(true,true);
         DataBuff:='';
         {$I-}Blockread(FromTextF,DataBuff,FpacketSize-1,MN_read){$I+};
         If IOResult<>0 then Memo1.Lines.Add('Error Read FromImageF');
         end;
     end;
     PacketSending:=true;
     NData:=MN_read;
     DataBuffPtr:=DataBuff;
           if MN_read=FpacketSize-1 then
             begin
                 Memo1.Lines.Add(Value+'...'+IntToStr(NData));
                 CommPort1.SendString(Value);
```

```
                CommPort1.SendData(DataBuffPtr,NData);
            end
        else
            Begin
            Memo1.Lines.Add(EndChar+'...'+IntToStr(NData));
            CommPort1.SendString(EndChar);
            CommPort1.SendData(DataBuffPtr,NData);
            FN_read:=FpacketSize-MN_read-1;
            CommPort1.SendData(DataBuffPtr,FN_read);
                case EndChar of
                    'n':begin
                        CloseFile(FromImageF);
                        ImageSending:=false;
                        SendPic.enabled:=true;
                        end;
                    'r':begin
                        CloseFile(FromSoundF);
                        SoundSending:=false;
                        StopBtn.enabled:=true;
                        end;
                    'v':begin
                        CloseFile(FromTextF);
                        TextSending:=false;
                        SendTBtn.enabled:=true;
                        end;
                end;
            end;

end;
procedure TMainForm.Sendreq(const Value:Char);
begin
     CommPort1.Disconnect;CommPort2.Connect;
     SendPic.enabled:=false;
     StopBtn.enabled:=false;
     SendTBtn.enabled:=false;
     CommPort2.SendString(Value);
end;
procedure TMainForm.SendFirstPacket(const H:Char;var SendF:String);
var  HFData,DatHBuff,SFileStr:String;
     DatInH,HN_read,NData:Integer;
     DataHBuffAr:Array[0..FpacketSize-1]of char;
     DatHBuffPtr:Pchar;
     INFile:File;
begin
     SFileStr:=SendF;
     case H of
          'i': begin
               {$I-}AssignFile(FromImageF,SFileStr);
               Reset(FromImageF,1); {$I+}
               FromImageSize:=IntToStr(FileSize(FromImageF));
               CommPort1.FlushBuffers(True,False);
               HFData:=H+FromImageSize+'*'+ExtractFileName(SFileStr)+'*';
               CommPort1.SendString(HFData);
               DatInH:=FPacketSize-Length(HFData);
               Blockread(FromImageF,DataHBuffAr,DatInH,HN_read);
               end;
           's':begin
               {$I-}AssignFile(FromSoundF,'c:\YKTemp\fsound.wav');
               Reset(FromSoundF,1){$I+}; if IOResult<> 0 then
Memo1.Lines.Add('Error FromSoundF');
               FromSoundSize:=IntToStr(FileSize(FromSoundF));
               CommPort1.FlushBuffers(True,False);
               HFData:=H+FromSoundSize+'*'+ExtractFileName(SFileStr)+'*';
               CommPort1.SendString(HFData);
               DatInH:=FPacketSize-Length(HFData);
               Blockread(FromSoundF,DataHBuffAr,DatInH,HN_read);
               end;
           't':begin
               {$I-}AssignFile(FromTextF,'c:\YKTemp\Text.txt');
               Reset(FromTextF,1);{$I+}
               FromTextSize:=IntToStr(FileSize(FromTextF));
```

```
                    HFData:=H+FromTextSize+'*'+ExtractFileName(SendF)+'*';
                    CommPort1.FlushBuffers(True,False);
                    CommPort1.SendString(HFData);
                    DatInH:=FPacketSize-Length(HFData);
                    Blockread(FromTextF,DataHBuffAr,DatInH,HN_read);
                    end;
                end;
                    Memo1.Lines.Add('Sending '+ExtractFileName(SendF));
                    Memo1.Lines.Add('Send '+HFData+'...');
      DatHBuffPtr:=DataHBuffAr;
      NData:=HN_read;
      CommPort1.SendData(DatHBuffPtr,NData);
      CommPort1.SendData(DatHBuffPtr,DatInH-HN_read);
      if DatInH-HN_read<>0 then
      case H of
       'i':begin
           CloseFile(FromImageF);
           SendPic.enabled:=true;
           ImageSending:=false;
           end;
       's':begin
           CloseFile(FromSoundF);
           StopBtn.enabled:=true;
           SoundSending:=false;
           end;
       't':begin
           CloseFile(FromTextF);
           SendTBtn.enabled:=true;
           TextSending:=false;
           end;
      end;
      PacketSending:=true;
end;
procedure TMainForm.Continue(const Prev:Char);
var Cs:Integer;
    Ack:String;
begin
     Cs:=0;
    {Be Sending Image}
     if ImageSending then
        begin
        {Be Sending Sound}
        if SoundSending then
            begin
            if  TextSending then Cs:=7
            else  Cs:=6
            end
        {not be Sending Sound}
        else
            begin
            if  TextSending then Cs:=5
            else  Cs:=4
            end
        end
     else {not be Sending Image}
        begin
        {Be Sending Sound}
        if SoundSending then
            begin
            if  TextSending then Cs:=3
            else  Cs:=2
            end
        {not be Sending Sound}
        else
            begin
            if  TextSending then Cs:=1
            else  Cs:=0
            end
        end;
   Memo1.Lines.Add(IntToStr(CS));
   case Cs of
     7:  case Prev of
```

```
                'm':Ack:='q';
                'q':Ack:='u';
                'u':Ack:='m';
            end;
        6:  case Prev of
                'u':Ack:='q';
                'm':Ack:='q';
                'q':Ack:='m';
            end;
        5:  case Prev of
                'q':Ack:='m';
                'm':Ack:='u';
                'u':Ack:='m';
            end;
        4:  Ack:='m';
        3:  case Prev of
                'm':Ack:='q';
                'u':Ack:='q';
                'q':Ack:='u';
            end;
        2:Ack:='q';
        1:Ack:='u';
        0:begin
        CommPort2.SendString('z');
        SendPic.enabled:=True;
        StopBtn.enabled:=True;
        SendTBtn.enabled:=True;
        Timer1.enabled:=false;Timer1.interval:=TimeC;
            CommPort2.Disconnect;
            CommPort1.Connect;
          end;
        end;
        Memo1.Lines.Add(Ack);
        CommPort2.SendString(Ack);
        end;
procedure TMainForm.TrueSize2Click(Sender: TObject);
begin
   JImage2.Stretch:=False;
end;

procedure TMainForm.FitToPage2Click(Sender: TObject);
begin
 JImage2.Stretch:=True;
end;

procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
        DeleteFile('c:\YKTemp\YKFile.wav');DeleteFile('c:\YKTemp\YKFile.bmp');
        DeleteFile('c:\YKTemp\fsound.wav');DeleteFile('c:\YKTemp\YKFile.avi');
        DeleteFile('c:\YKTemp\sound.wav');DeleteFile('c:\YKTemp\YKFile.jpg');
        DeleteFile('c:\YKTemp\YKFile.txt');DeleteFile('c:\YKTemp\Text.txt');
        DeleteFile('c:\YKTemp\YKDPlay.wav');DeleteFile('c:\YKTemp\YKDPlay.bmp');
        DeleteFile('c:\YKTemp\YKDPlay.jpg');DeleteFile('c:\YKTemp\YKDPlay.avi');
        DeleteFile('c:\YKTemp\YKDPlay.txt');DeleteFile('c:\YKTemp\CPlay.wav');
        DeleteFile('c:\YKTemp\Play.wav');
        RemoveDir('c:\YKTemp\');

end;

procedure TMainForm.Manual1Click(Sender: TObject);
begin
     ManualForm.ShowModal
end;

procedure TMainForm.ResetBtnClick(Sender: TObject);
begin
     if CommPort2.Connected then
     begin
          CommPort2.FlushBuffers(True,True);
          if SoundSending then   {$I-} Closefile(ToSoundF);
          if ImageSending then    Closefile(ToImageF);
          if TextSending then    Closefile(ToTextF); {$I+}
```

```
      end
      else
      begin
           CommPort1.FlushBuffers(True,True);
           if SoundSending then    {$I-} Closefile(FromSoundF);
           if ImageSending then     Closefile(FromImageF);
           if TextSending then    Closefile(FromTextF); {$I+}
      end;
      packet:='';
      SoundSending:=false;StopBtn.enabled :=True;
      ImageSending:=false; SendPic.enabled:=True;
      TextSending:=false;SendTBtn.enabled :=True;

end;

procedure TMainForm.WaitFreeBf;
begin
  Waiting:=true;
end;
procedure TMainForm.DocMemoChange(Sender: TObject);
begin
      SendTBtn.enabled:=True;
end;

procedure TMainForm.Timer1Timer(Sender: TObject);
begin
      Timer1.enabled:=false;Timer1.interval:=TimeC;Packet:='';
      CommPort1.FlushBuffers(True,True);continue('m');
end;

procedure TMainForm.LisAgnClick(Sender: TObject);
begin
      if not sndPlaySound('c:\YKTemp\YKDPlay.wav', SND_ASYNC) then
         StatusBar1.Panels.Items[1].Text:= 'ERROR With Playing';
end;

end.
```

โปรแกรม YKSendFile.pas

```
unit YKsendfile;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls, ExtCtrls, ComDrv32,YKMainComm;

type
  TSRFileForm1 = class(TForm)
    SRFileStatusBar1: TStatusBar;
    SRFilePanel3: TPanel;
    Panel1: TPanel;
    SRFilePanel1: TPanel;
    SRFileLabel2: TLabel;
    SRFileSendBtn: TButton;
    SRFileEdit1: TEdit;
    SRFileStatusBar2: TStatusBar;
    SRFileSelectBtn: TButton;
    SRFilePanel2: TPanel;
    SRFileLabel3: TLabel;
    CancelRecBtn: TButton;
    SRFileEdit2: TEdit;
    SRFileStatusBar3: TStatusBar;
    SRFileLabel1: TLabel;
    SRFileOpenDialog1: TOpenDialog;
    SRFileSaveDialog1: TSaveDialog;
    SRFileCommPort1: TCommPortDriver;
    SRFileCommPort2: TCommPortDriver;
    SRFileMemo1: TMemo;
    SRFileExitBtn: TButton;
    Timer1: TTimer;
    AutoRBtn: TRadioButton;
    procedure SRFileExitBtnClick(Sender: TObject);
    procedure SRFileSelectBtnClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure SRFileCommPort1ReceiveData(Sender: TObject; DataPtr: Pointer;
      DataSize: Integer);
    procedure SRFileCommPort2ReceiveData(Sender: TObject; DataPtr: Pointer;
      DataSize: Integer);
    procedure SRFileSendBtnClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure CancelRecBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure AutoRBtnDblClick(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }

  public

    { Public declarations }
  end;
  const FpacketSize=2048;
        Stime=15000;
        Rtime=8000;
  var
  SRFileForm1: TSRFileForm1;
  packet,Lpacket,ToFName,TFileName,FromFileSize,ToFileSize:String;
  PacketAr:Array[1..2048] of Char;
  Index:Integer;
  FromFile,ToFile:File;
  MN_read:Integer;
  Sending:Boolean;
  DataBuff:Array[0..FpacketSize-2]of Char;

implementation
```

```
{$R *.DFM}

procedure TSRFileForm1.SRFileExitBtnClick(Sender: TObject);
var garb:Pchar;
begin
     if  SRFileCommPort2.Connected then
     if Sending then
          With SRFileCommPort2 do
          begin SendString('Y');
                SendData(garb,FPacketSize-1);
                Disconnect;
          end
     else
      With SRFileCommPort2 do
      begin SendString('Y');
            Disconnect;
      end;

     if SRFileCommPort1.Connected then
     With SRFileCommPort1 do
      begin SendString('Y');
            Disconnect;
      end;
     MainForm.CommPort1.Connect;
     SRFileForm1.close;
end;

procedure TSRFileForm1.SRFileSelectBtnClick(Sender: TObject);
begin
       if SRFileOpenDialog1.Execute then
       begin
       SRFileEdit1.Text:=SRFileOpenDialog1.Filename;
       SRFileCommPort2.Sendstring('E');
       end;
       with SRFileStatusBar1 do
         Panels.Items[1].Text := 'Selected File is from '+
SRFileOpenDialog1.FileName;
end;

procedure TSRFileForm1.FormShow(Sender: TObject);
begin
       if SRFileCommPort2.Connected then
       begin
       SRFileMemo1.Lines.Add('Connectd for sending');
       with SRFileStatusBar1 do
       Panels.Items[0].Text := 'Connected';
       end;
end;

procedure TSRFileForm1.SRFileCommPort1ReceiveData(Sender: TObject;
  DataPtr: Pointer; DataSize: Integer);
var RxDataPtr:Pchar;
    PosId,NCheck:Integer;
begin
   {Receive Data until 1 Packet }
   RxDataPtr:=DataPtr;
   if length(Packet)<FpacketSize then
        begin
        while (DataSize>0) and (length(Packet)<FpacketSize) do
                begin
                Packet:=Packet+RxDataPtr^;
                if  Length(Packet)>1 then
                  begin  PacketAr[Index]:=RxDataPtr^;
                         inc(Index);
                  end;
                inc(RxDataPtr);
                dec(DataSize);
                end;

        if length(Packet)=FpacketSize then
                begin
                PosId:=2;Index:=1;
```

```
                case Packet[1] of
                'E': begin
                     While Packet[PosId]<>'*' do
                        begin
                        ToFileSize:=ToFileSize+Packet[PosId];
                        inc(PosId);
                        end;
                        inc(PosId);
                     While Packet[PosId]<>'*' do
                        begin
                        TFileName:=TFileName+Packet[PosId];
                        inc(PosId);
                        end;
                     with SRFileStatusBar1 do
                       Panels.Items[0].Text :=TFileName;
                     if not AutoRBtn.Checked then
                         begin if SRFileSaveDialog1.Execute then
                          SRFileEdit2.text:=SRFileSaveDialog1.filename
                         end
                     else   SRFileEdit2.text:= 'C:\YKtemp\'+TFileName;
                     CancelRecBtn.enabled:=True;
                     AssignFile(ToFile,SRFileEdit2.text);
                     Rewrite(ToFile,1);
                     While (PosId<FpacketSize) and
(FileSize(ToFile)<StrToInt(ToFileSize)) do
                        begin
                        inc(PosId);
                        BlockWrite(ToFile,Packet[PosId],1,Ncheck);
                        end;
                     if PosId<FpacketSize then
                      begin
                      CloseFile(ToFile);
                      Packet:='';ToFileSize:='';TFileName:='';

Sending:=False;Timer1.Enabled:=False;Timer1.interval:=Rtime;
                      SRFileCommPort1.SendString('eN');
                      end
                      else
                      begin
                      Packet:='';
                      SRFileCommPort1.SendString('B');
                      Timer1.interval:=Rtime;
                      Timer1.Enabled:=True;
                      end;
                     end;
                'B': begin
                     //Kill Timeout
                     Timer1.Enabled:=False;
                     BlockWrite(ToFile,PacketAr,FPacketSize-1,NCheck);
                     Packet:='';
                     SRFileCommPort1.SendString('B');
                     with SRFileStatusBar3 do
                       Panels.Items[0].Text := IntToStr(FileSize(ToFile));
                     with SRFileStatusBar1 do
                       Panels.Items[1].Text :='Left '+
                       IntToStr(StrToInt(ToFileSize)-FileSize(ToFile));
                     Timer1.Enabled:=True;
                     end;
                'e': begin
                     //Kill Timeout
                     Timer1.Enabled:=False;
                     BlockWrite(ToFile,PacketAr,StrToInt(ToFileSize)-
FileSize(ToFile),NCheck);
                     with SRFileStatusBar1 do
                       Panels.Items[1].Text :='Finish';
                     SRFileEdit2.Text:='';
                     SRFileMemo1.Lines.Add('Recieved File '+SRFileEdit2.text+'
size= '+
                     IntToStr(FileSize(ToFile)));
                     with SRFileStatusBar3 do
                       Panels.Items[0].Text := IntToStr(FileSize(ToFile)) ;
                     CloseFile(ToFile);CancelRecBtn.enabled:=False;
```

```
  Sending:=False;SRFileSelectBtn.Enabled:=True;SRFileSendBtn.Enabled:=True;
            end;
       'Y' :begin
            if Sending then Closefile(FromFile);
            SRFileCommPort2.DisConnect;
            MainForm.CommPort1.Connect;
            SRFileForm1.close;
            Timer1.enabled:=false;
            end;
       end;


  end;

  procedure TSRFileForm1.SRFileSendBtnClick(Sender: TObject);
  var  HFData,DatHBuff:String;
       DatInH,HN_read,NData:Integer;
       DataHBuffAr:Array[0..FpacketSize-2]of char;
       DatHBuffPtr:Pchar;
  begin
       with SRFileStatusBar1 do
         Panels.Items[1].Text := 'Sending'; Sending:=True;
       AssignFile(FromFile,SRFileEdit1.Text);
       Reset(FromFile,1);
       FromFileSize:=IntToStr(FileSize(FromFile));
       SRFileCommPort2.FlushBuffers(True,True);
       HFData:='E'+FromFileSize+'*'+ExtractFileName(SRFileEdit1.Text)+'*';
       SRFileMemo1.Lines.Add('Sending '+SRFileOpenDialog1.filename);
       SRFileMemo1.Lines.Add('Send '+HFData+'...');
       with SRFileStatusBar2 do
         Panels.Items[0].Text := 'FileSize to send is '+FromFileSize;
       SRFileCommPort2.SendString(HFData);
       DatInH:=FPacketSize-Length(HFData);
       Blockread(FromFile,DataHBuffAr,DatInH,HN_read);
       DatHBuffPtr:=DataHBuffAr;
       NData:=HN_read;
       SRFileCommPort2.SendData(DatHBuffPtr,NData);
       SRFileCommPort2.SendData(DatHBuffPtr,DatInH-HN_read);
       SRFileMemo1.Lines.Add(IntToStr(DatInH-HN_read));
       Timer1.interval:=15000;
       Timer1.enabled:=true;

  end;

  procedure TSRFileForm1.FormClose(Sender: TObject;
  var Action: TCloseAction);
  begin
      SRFileExitBtn.Click;
  end;

  procedure TSRFileForm1.CancelRecBtnClick(Sender: TObject);
  begin
        SRFileCommPort1.SendString('e');
        with SRFileStatusBar1 do
         Panels.Items[1].Text := 'Cancel Receiving';
         CancelRecBtn.enabled:=false;
  end;

  procedure TSRFileForm1.FormCreate(Sender: TObject);
  begin
     with SRFileStatusBar1 do
      Panels.Items[0].Text := 'Connect';
      Sending:=False;Index:=1;
      if SRFileCommPort2.Connected then SRFileMemo1.Lines.Add('Connected');

  end;

  procedure TSRFileForm1.AutoRBtnDblClick(Sender: TObject);
  begin
     if AutoRBtn.Checked then AutoRBtn.Checked:=false;
  end;
```

```
procedure TSRFileForm1.Timer1Timer(Sender: TObject);
var NData:integer;
    DataBuffPtr:PChar;
begin
     packet:='';
     if SRFileCommPort1.Connected then
     begin
     SRFileCommPort1.SendString('R');
     SRFileMemo1.Lines.Add('Send Retransmit Req');
     end
     else
     begin
     DataBuffPtr:=DataBuff;
     if MN_read=FpacketSize-1 then
            begin
                SRFileCommPort2.SendString('B');
                While NData >=2000 do
                begin SRFileCommPort2.SendData(DataBuffPtr,2000);
                      NData:=NData-2000;
                      inc(DataBuffPtr,2000);
                end;
                SRFileCommPort2.SendData(DataBuffPtr,NData);
            end;
     end;
end;

end.
```

โปรแกรม YKChat.pas

```
unit YKchat;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, StdCtrls, Buttons, ComCtrls, ExtCtrls, ComDrv32;

type
  TChatForm = class(TForm)
    ChatPanel1: TPanel;
    ChatPanel2: TPanel;
    ChatStatusBar1: TStatusBar;
    ChatExitBtn: TBitBtn;
    ChatMemo1: TMemo;
    ChatMemo2: TMemo;
    ChatLabel1: TLabel;
    ChatPopupMenu1: TPopupMenu;
    ChatCut1: TMenuItem;
    ChatCopy1: TMenuItem;
    ChatPaste1: TMenuItem;
    ChatDelete1: TMenuItem;
    ChatSelectAll1: TMenuItem;
    ChatEdit1: TEdit;
    ChatN2: TMenuItem;
    ChatCommPort1: TCommPortDriver;
    ChatMemo3: TMemo;
    procedure ChatExitBtnClick(Sender: TObject);
    procedure ChatMemo2KeyPress(Sender: TObject; var Key: Char);
    procedure FormShow(Sender: TObject);
    procedure ChatCommPort1ReceiveData(Sender: TObject; DataPtr: Pointer;
      DataSize: Integer);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  ChatForm: TChatForm;
implementation
 {$R *.DFM}

uses YKMainComm;

procedure TChatForm.ChatExitBtnClick(Sender: TObject);
var Stop:string;
begin
     Stop:=#8;
     ChatCommPort1.SendString(Stop);
     ChatCommPort1.DisConnect;
     MainForm.CommPort1.Connect;
     ChatForm.close;
end;
procedure TChatForm.ChatMemo2KeyPress(Sender: TObject; var Key: Char);
var chats,head:string;
begin
     {with ChatStatusbar1 do
          Panels.Items[0].Text := ' ';}
     case Key of
      #13: if chatMemo2.Lines.Count>0 then
           begin
             chats := ChatMemo2.Lines[ChatMemo2.Lines.Count-1];
             head:=#6;
             chats := head+ChatEdit1.Text+'>'+chats;
             ChatCommPort1.SendData( pchar(chats), length(chats) );
             ChatMemo3.Lines.Add('Send '+IntToStr(length(Chats))+' bytes');
```

```
ChatMemo1.Lines.Add('<'+ChatEdit1.Text+'>'+ChatMemo2.Lines[ChatMemo2.Lines.Count-
1]);
            end;
      end;
      with ChatStatusbar1 do
           Panels.Items[0].Text := 'Finish Sending';
end;
procedure TChatForm.FormShow(Sender: TObject);
var start:string;
begin
  start:=#4;
 ChatCommPort1.Sendstring(start);
  with ChatStatusbar1 do
    Panels.Items[0].Text := 'Connected';
 if ChatCommPort1.Connected then ChatMemo3.Lines.Add('Connecting.. ')
    else ChatMemo3.Lines.Add('Port may be already in use Wait until free');
end;
procedure TChatForm.ChatCommPort1ReceiveData(Sender: TObject;
  DataPtr: Pointer; DataSize: Integer);
var  RxDataPtr,ChatextPtr:Pchar;
     Chatext,TextL:String;
begin
     RxDataPtr:=DataPtr;
     ChatMemo3.Lines.Add('On Receive '+IntTostr(DataSize)+' bytes');
     if RxDataPtr^=#4 then
        ChatMemo3.Lines.Add('Ping Pong!');
     if RxDataPtr^=#8 then
        begin
        ChatCommPort1.DisConnect;
        MainForm.CommPort1.Connect;
        ChatForm.close;
        end;
     if RxDataPtr^=#6 then
     begin
     // Show On ChatMemo
     inc(RxDataPtr);
     dec(DataSize);
     while DataSize>0 do
           begin
           Chatext:=Chatext+RxDataPtr^;
           inc(RxDataPtr);
           dec(DataSize);
           end;

     ChatMemo1.Lines.Add('<'+Chatext)
     end
     else
     //Receive left RxData
     begin
     inc(RxDataPtr);
     dec(DataSize);
     while DataSize>0 do
           begin
           Chatext:=Chatext+RxDataPtr^;
           inc(RxDataptr);
           dec(DataSize);
           end;

     Chatext:=ChatMemo1.Lines[ChatMemo1.Lines.count-1]+Chatext;
     ChatMemo1.Lines[ChatMemo1.Lines.Count-1]:=Chatext;
     end;
     Chatext:='';

end;

procedure TChatForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
        ChatExitBtn.Click;
end;

end.
```

โปรแกรม Connect.pas

```
unit YKConnect;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComDrv32, Buttons, ExtCtrls, DBCtrls;
type
  TConnectForm = class(TForm)
    GroupBox1: TGroupBox;
    BitBtn3: TBitBtn;
    DCommPort: TCommPortDriver;
    DialBtn: TBitBtn;
    Label1: TLabel;
    DialEdit: TEdit;
    ExitBtn: TBitBtn;
    ConnectGrp1: TRadioGroup;
    ConnectGrp2: TRadioGroup;
    Memo1: TMemo;
    Send: TButton;
    MediaGrp: TRadioGroup;
    RCommPort: TCommPortDriver;
    procedure ExitBtnClick(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure DialBtnClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure ComConBtnClick(Sender: TObject);
    procedure ConnectGrp1Click(Sender: TObject);
    procedure DCommPortReceiveData(Sender: TObject; DataPtr: Pointer;
      DataSize: Integer);
    procedure DialEditKeyPress(Sender: TObject; var Key: Char);
    procedure SendClick(Sender: TObject);
    procedure ConnectGrp2Click(Sender: TObject);
    procedure MediaGrpClick(Sender: TObject);
  private
    { Private declarations }
    procedure Modem_Initial;
  public
    { Public declarations }
  end;
var
  ConnectForm: TConnectForm;
  DiL:String='atdt';
  ResultStr:String='';
implementation
uses YKPConfig, YKMainComm, YKWait;
{$R *.DFM}

procedure TConnectForm.ExitBtnClick(Sender: TObject);
begin
     if DCommPort.connected then DCommPort.Disconnect;
     MainForm.CommPort1.Connect;
     close;
end;
procedure TConnectForm.BitBtn3Click(Sender: TObject);
begin
        ConfigForm.ShowModal;
end;
procedure TConnectForm.DialBtnClick(Sender: TObject);
begin
     DCommPort.connect;
     if ConnectGrp2.ItemIndex=0 then
     begin
      if MediaGrp.ItemIndex=0 then
      begin
      Modem_Initial;
      DCommPort.SendString(DiL+DialEdit.text+#13);
      Memo1.Lines.Add(DiL+DialEdit.text+#13);
      WaitForm.Label1.caption:='Dial';
      end
```

```
        else
        begin
        WaitForm.Label1.caption:='Try Connecting';
        DCommPort.SendString('..C');
        end;
       end
       else
        begin
         if MediaGrp.ItemIndex=0 then
         begin
         WaitForm.Label1.caption:='Wait Ringing';
         ResultStr:='';DCommPort.SendString('ATS0=1'+#13);
         end
         else
         begin
         WaitForm.Label1.caption:='Wait Serial Connecting';
         end;
        end;
        WaitForm.ShowModal;
end;
procedure TConnectForm.FormCreate(Sender: TObject);
begin
     DiL:='atdt'
end;
procedure TConnectForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
     ExitBtn.Click;
end;

procedure TConnectForm.ComConBtnClick(Sender: TObject);
begin
     if ConnectGrp2.ItemIndex=0 then
     begin
      WaitForm.Label1.caption:='Connecting';
      DCommPort.SendString(#10#13+'C');
     end
     else WaitForm.Label1.caption:='Wait ';
     WaitForm.Showmodal;
end;
procedure TConnectForm.ConnectGrp1Click(Sender: TObject);
begin
     {Select Tone Or Pulse Dial method  }
     if ConnectGrp1.ItemIndex =1 then DiL:='atpt'
     else DiL:='atdt';
end;
procedure TConnectForm.DCommPortReceiveData(Sender: TObject;
  DataPtr: Pointer; DataSize: Integer);
var RxDataPtr:Pchar;
    PosId,NCheck,DSize:Integer;
    TestRx:String;
begin
   {Receive Header Data }
    RxDataPtr:='';
    ResultStr:='';
    RxDataPtr:=DataPtr;
    while DataSize>0 do
           begin
           ResultStr:=ResultStr+RxDataPtr^;
           inc(RxDataPtr);
           dec(DataSize);
           end;
    Memo1.Lines.Add(IntToStr(length(ResultStr))+ResultStr);
    if length( ResultStr)>=1 then
    begin
            case  ResultStr[3] of
                'B':begin
                    ResultStr:='';
                    WaitForm.Close;
                    exit;
                    end;
                'N':begin
```

```
                    ResultStr:='';
                    WaitForm.Close;
                    exit;
                    end;
                'O':begin
                    ResultStr:='';
                    exit;
                    end;
                'R': begin
                    ResultStr:='';
                    exit;
                    end;
                'C': begin
                     WaitForm.close;
                     if ConnectGrp2.ItemIndex=1 then
                        if MediaGrp.ItemIndex=1 then
                           DCommPort.SendString('..C');
                     DCommPort.Disconnect;
                     MainForm.CommPort1.Connect;
                     Close;
                     end;
                end;
    end;
                ResultStr:='';
end;


procedure TConnectForm.DialEditKeyPress(Sender: TObject; var Key: Char);
begin
     case Key of
     #13:  begin
             DCommPort.SendString(DialEdit.Text);
           end;
     end;
end;
procedure TConnectForm.SendClick(Sender: TObject);
begin
            DCommPort.SendString(DialEdit.Text+#13);
end;
procedure TConnectForm.ConnectGrp2Click(Sender: TObject);
begin
     If ConnectGrp2.ItemIndex=1 then
     If MediaGrp.ItemIndex=1 then DialBtn.caption:='Wait'
     else DialBtn.caption:='Listen'
     else    If MediaGrp.ItemIndex=1 then DialBtn.caption:='Connect'
     else DialBtn.caption:='Dial'
end;
procedure TConnectForm.MediaGrpClick(Sender: TObject);
begin
   ConnectForm.ConnectGrp2Click(nil);
end;
procedure TConnectForm.Modem_Initial;
begin
     DCommPort.Disconnect;
     RCommPort.Connect;
     RCommPort.SendString('ATB0F1M1X4&W'+#13);
     RCommPort.SendString('AT&A3&B1&G0&H1&I0&K1&M4&N0&R2&S0&T5&U0&Y1'+#13);
     RCommPort.Disconnect;
     DCommPort.Connect;
end;
end.
```

โปรแกรม YKPConfig.pas

```
unit YKPConfig;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, Buttons,ComDrv32;
type
  TConfigForm = class(TForm)
    Panel1: TPanel;
    ApplyComBtn: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    Panel2: TPanel;
    ComPortRG: TRadioGroup;
    BaudRateRG: TRadioGroup;
    Panel3: TPanel;
    DataBitsRG: TRadioGroup;
    ParityRG: TRadioGroup;
    HandshakingRG: TRadioGroup;
    procedure ApplyComBtnClick(Sender: TObject);
    procedure BaudRateRGClick(Sender: TObject);
    procedure ComPortRGClick(Sender: TObject);
    procedure DataBitsRGClick(Sender: TObject);
    procedure ParityRGClick(Sender: TObject);
    procedure HandshakingRGClick(Sender: TObject);
    procedure ApplyComm(var Com:TCommPortDriver);
    procedure BitBtn2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    ComPortSpeedValue:TComPortBaudRate;
    ComPortDataBitsValue:TComPortDataBits;
    ComPortParityValue:TComPortParity;
    ComPortHwHandshakingValue:TComPortHwHandshaking;
    ComPortSwHandshakingValue:TComPortSwHandshaking;
  end;
var
  ConfigForm: TConfigForm;
implementation
uses YKMainComm, YKchat, YKsendfile, YKConnect;
{$R *.DFM}

procedure TConfigForm.ApplyComBtnClick(Sender: TObject);
var config:textfile;
    InitStr:string;
begin
//  ApplyCommSettings;
  AssignFile(config,'YKcomm.cfg');
  Rewrite(config);
  InitStr:=IntToStr(ComPortRG.ItemIndex)+IntToStr(BaudRateRG.ItemIndex)+
  IntToStr(DataBitsRG.ItemIndex)+IntToStr(ParityRG.ItemIndex)+
  IntToStr(HandshakingRG.ItemIndex);
  Write(config,initStr);
  closefile(config);
  ApplyComm(ConnectForm.DCommPort);
  ApplyComm(ChatForm.ChatCommPort1);
  ApplyComm(MainForm.CommPort1);
  ApplyComm(MainForm.CommPort2);
  ApplyComm(SRFileForm1.SRFileCommPort1);
  ApplyComm(SRFileForm1.SRFileCommPort2);
  ConfigForm.close;
end;
procedure TConfigForm.BaudRateRGClick(Sender: TObject);
begin
    ComPortSpeedValue := TComPortBaudRate(BaudRateRG.ItemIndex+1);
end;
```

```
procedure TConfigForm.ComPortRGClick(Sender: TObject);
begin
        //ApplyCommSetting
end;
procedure TConfigForm.DataBitsRGClick(Sender: TObject);
begin
        ComPortDataBitsValue := TComportDataBits(DataBitsRG.ItemIndex);
end;
procedure TConfigForm.ParityRGClick(Sender: TObject);
begin
        ComPortParityValue := TComportParity(ParityRG.ItemIndex);
end;
procedure TConfigForm.HandshakingRGClick(Sender: TObject);
begin
        case HandshakingRG.ItemIndex of
    0: // none
      begin
        ComPortHwHandshakingValue := hhNone;
        ComPortSwHandshakingValue := shNone;
      end;
    1: // RTS/CTS
      begin
        ComPortHwHandshakingValue := hhRTSCTS;
        ComPortSwHandshakingValue := shNone;
      end;
    2: // XON/XOFF
      begin
        ComPortHwHandshakingValue := hhNone;
        ComPortSwHandshakingValue := shXONXOFF;
      end;
    3: // RTS/CTS + XON/XOFF
      begin
        ComPortHwHandshakingValue := hhRTSCTS;
        ComPortSwHandshakingValue := shXONXOFF;
      end;
    end;
end;
procedure TConfigForm.BitBtn2Click(Sender: TObject);
begin
     ComPortRG.ItemIndex:=1;BaudRateRG.ItemIndex:=9;
     DataBitsRG.ItemIndex:=3;ParityRG.ItemIndex:=0;
     HandshakingRG.ItemIndex:=0;
end;
procedure TConfigForm.ApplyComm(var Com:TCommPortDriver);
var dcb: TDCB;
begin
     //Apply Commport Component Setting
     Com.ComPort := TComPortNumber(ComPortRG.ItemIndex);
     Com.ComPortSpeed:= TComPortBaudRate(BaudRateRG.ItemIndex+1);
     Com.ComPortDataBits := TComPortDataBits(DataBitsRG.ItemIndex);
     Com.ComPortParity := TComPortParity(ParityRG.ItemIndex);
     case HandshakingRG.ItemIndex of
    0: // none
      begin
        Com.ComPortHwHandshaking := hhNone;
        Com.ComPortSwHandshaking := shNone;
      end;
    1: // RTS/CTS
      begin
        Com.ComPortHwHandshaking := hhRTSCTS;
        Com.ComPortSwHandshaking := shNone;
      end;
    2: // XON/XOFF
      begin
        Com.ComPortHwHandshaking := hhNone;
        Com.ComPortSwHandshaking := shXONXOFF;
      end;
    3: // RTS/CTS + XON/XOFF
      begin
        Com.ComPortHwHandshaking := hhRTSCTS;
        Com.ComPortSwHandshaking := shXONXOFF;
      end;
```

```
    end;
      // Do nothing if not connected
    if  Com.Connected then
      begin
          Com.Connect;
          Com.Disconnect;
      end;
  end;
  procedure TConfigForm.FormCreate(Sender: TObject);
  begin
       ApplyComBtn.Click;
  end;
  procedure TConfigForm.BitBtn3Click(Sender: TObject);
  begin
       Close;
  end;
  end.
```

ซอร์สโค้ดชื่อ COMDrv32.pas

```
| COM Port Driver for Delphi 2.0
| Written by Marco Cocco
| Please send comments to d3k@mdnet.it
  | URL: http://www.mdlive.com/d3k/
*******************************************************************************
unit ComDrv32;

interface
uses
  Windows, Messages, SysUtils, Classes, Forms;
type
  // COM Port Baud Rates
  TComPortBaudRate = (br110, br300, br600, br1200, br2400, br4800,
                      br9600, br14400, br19200, br38400, br56000,
                      br57600, br115200{v1.02; removed ->, br128000, br256000} );
  // COM Port Numbers
  TComPortNumber = ( pnCOM1, pnCOM2, pnCOM3, pnCOM4 );
  // COM Port Data bits
  TComPortDataBits = ( db5BITS, db6BITS, db7BITS, db8BITS );
  // COM Port Stop bits
  TComPortStopBits = ( sb1BITS, sb1HALFBITS, sb2BITS );
  // COM Port Parity
  TComPortParity = ( ptNONE, ptODD, ptEVEN, ptMARK, ptSPACE );
  // COM Port Hardware Handshaking
  TComPortHwHandshaking = ( hhNONE, hhRTSCTS );
  // COM Port Software Handshaing
  TComPortSwHandshaking = ( shNONE, shXONXOFF );
  TComPortReceiveDataEvent = procedure( Sender: TObject; DataPtr: pointer;
DataSize: integer ) of object;
  TCommPortDriver = class(TComponent)

  protected
    FComPortHandle              : THANDLE; // COM Port Device Handle
    FComPort                    : TComPortNumber; // COM Port to use (1..4)
    FComPortBaudRate            : TComPortBaudRate; // COM Port speed (brXXXX)
    FComPortDataBits            : TComPortDataBits; // Data bits size (5..8)
    FComPortStopBits            : TComPortStopBits; // How many stop bits to use
(1,1.5,2)
    FComPortParity              : TComPortParity; // Type of parity to use
(none,odd,even,mark,space)
    FComPortHwHandshaking       : TComPortHwHandshaking; // Type of hw handshaking
to use
    FComPortSwHandshaking       : TComPortSwHandshaking; // Type of sw handshaking
to use
    FComPortInBufSize           : word; // Size of the input buffer
    FComPortOutBufSize          : word; // Size of the output buffer
    FComPortReceiveData         : TComPortReceiveDataEvent; // Event to raise on
data reception
    FComPortPollingDelay        : word; // ms of delay between COM port pollings
    FEnableDTROnOpen            : boolean; { enable/disable DTR line on connect }
    FOutputTimeout              : word; { output timeout - milliseconds }
    FNotifyWnd                  : HWND; // This is used for the timer
    FTempInBuffer               : pointer;

    procedure SetComHandle( Value: THANDLE );
    procedure SetComPort( Value: TComPortNumber );
    procedure SetComPortBaudRate( Value: TComPortBaudRate );
    procedure SetComPortDataBits( Value: TComPortDataBits );
    procedure SetComPortStopBits( Value: TComPortStopBits );
    procedure SetComPortParity( Value: TComPortParity );
    procedure SetComPortHwHandshaking( Value: TComPortHwHandshaking );
    procedure SetComPortSwHandshaking( Value: TComPortSwHandshaking );
    procedure SetComPortInBufSize( Value: word );
    procedure SetComPortOutBufSize( Value: word );
    procedure SetComPortPollingDelay( Value: word );
    procedure ApplyCOMSettings;
    procedure TimerWndProc( var msg: TMessage );

  public
```

```
    constructor Create( AOwner: TComponent ); override;
    destructor Destroy; override;

    function Connect: boolean;
    procedure Disconnect;
    function Connected: boolean;
    { v1.02: flushes the rx/tx buffers }
    procedure FlushBuffers( inBuf, outBuf: boolean );
    { v1.02: returns the output buffer free space or 65535 if
             not connected }
    function OutFreeSpace: word;

    { Send data }
    { v1.02: changed result time from 'boolean' to 'integer'. See the docs
             for more info }
    function SendData( DataPtr: pointer; DataSize: integer ): integer;
    // Send a pascal string (NULL terminated if $H+ (default))
    function SendString( s: string ): boolean;
    // v1.02: send a C-style strings (NULL terminated)
    function SendZString( s: pchar ): boolean;
    // v1.02: set DTR line high (onOff=TRUE) or low (onOff=FALSE).
    //        You must not use HW handshaking.
    procedure ToggleDTR( onOff: boolean );
    // v1.02: set RTS line high (onOff=TRUE) or low (onOff=FALSE).
    //        You must not use HW handshaking.
    procedure ToggleRTS( onOff: boolean );

    // v1.02: make the Handle to the com port public (for TAPI...)
    property ComHandle: THANDLE read FComPortHandle write SetComHandle;

  published
    // Which COM Port to use
    property ComPort: TComPortNumber read FComPort write SetComPort default
pnCOM2;
    // COM Port speed (bauds)
    property ComPortSpeed: TComPortBaudRate read FComPortBaudRate write
SetComPortBaudRate default br9600;
    // Data bits to used (5..8, for the 8250 the use of 5 data bits with 2 stop
bits is an invalid combination,
    // as is 6, 7, or 8 data bits with 1.5 stop bits)
    property ComPortDataBits: TComPortDataBits read FComPortDataBits write
SetComPortDataBits default db8BITS;
    // Stop bits to use (1, 1.5, 2)
    property ComPortStopBits: TComPortStopBits read FComPortStopBits write
SetComPortStopBits default sb1BITS;
    // Parity Type to use (none,odd,even,mark,space)
    property ComPortParity: TComPortParity read FComPortParity write
SetComPortParity default ptNONE;
    // Hardware Handshaking Type to use:
    //  cdNONE          no handshaking
    //  cdCTSRTS        both cdCTS and cdRTS apply (** this is the more common
method**)
    property ComPortHwHandshaking: TComPortHwHandshaking
             read FComPortHwHandshaking write SetComPortHwHandshaking default
hhNONE;
    // Software Handshaking Type to use:
    //  cdNONE          no handshaking
    //  cdXONXOFF       XON/XOFF handshaking
    property ComPortSwHandshaking: TComPortSwHandshaking
             read FComPortSwHandshaking write SetComPortSwHandshaking default
shNONE;
    // Input Buffer size
    property ComPortInBufSize: word read FComPortInBufSize write
SetComPortInBufSize default 2048;
    // Output Buffer size
    property ComPortOutBufSize: word read FComPortOutBufSize write
SetComPortOutBufSize default 2048;
    // ms of delay between COM port pollings
    property ComPortPollingDelay: word read FComPortPollingDelay write
SetComPortPollingDelay default 50;
    // v1.02: Set to TRUE to enable DTR line on connect and to leave it on until
disconnect.
```

```
    //        Set to FALSE to disable DTR line on connect.
    property EnableDTROnOpen: boolean read FEnableDTROnOpen write
FEnableDTROnOpen default true;
    // v1.02: Output timeout (milliseconds)
    property OutputTimeout: word read FOutputTimeOut write FOutputTimeout default
4000;
    // Event to raise when there is data available (input buffer has data)
    property OnReceiveData: TComPortReceiveDataEvent read FComPortReceiveData
write FComPortReceiveData;
  end;

procedure Register;

implementation

constructor TCommPortDriver.Create( AOwner: TComponent );
begin
  inherited Create( AOwner );
  // Initialize to default values
  FComPortHandle              := 0;       // Not connected
  FComPort                    := pnCOM2;  // COM 2
  FComPortBaudRate            := br9600;  // 9600 bauds
  FComPortDataBits            := db8BITS; // 8 data bits
  FComPortStopBits            := sb1BITS; // 1 stop bit
  FComPortParity              := ptNONE;  // no parity
  FComPortHwHandshaking       := hhNONE;  // no hardware handshaking
  FComPortSwHandshaking       := shNONE;  // no software handshaking
  FComPortInBufSize           := 2048;    // input buffer of 2048 bytes
  FComPortOutBufSize          := 2048;    // output buffer of 2048 bytes
  FComPortReceiveData         := nil;     // no data handler
  FComPortPollingDelay        := 50;      // poll COM port every 50ms
  FOutputTimeout              := 4000;    // output timeout - 4000ms
  FEnableDTROnOpen            := true;    // DTR high on connect
  // Temporary buffer for received data
  GetMem( FTempInBuffer, FComPortInBufSize );
  // Allocate a window handle to catch timer's notification messages
  if not (csDesigning in ComponentState) then
    FNotifyWnd := AllocateHWnd( TimerWndProc );
end;

destructor TCommPortDriver.Destroy;
begin
  // Be sure to release the COM device
  Disconnect;
  // Free the temporary buffer
  FreeMem( FTempInBuffer, FComPortInBufSize );
  // Destroy the timer's window
  DeallocateHWnd( FNotifyWnd );
  inherited Destroy;
end;
// v1.02: The COM port handle made public and writeable.
// This lets you connect to external opened com port.
// Setting ComPortHandle to 0 acts as Disconnect.
procedure TCommPortDriver.SetComHandle( Value: THANDLE );
begin
  // If same COM port then do nothing
  if FComPortHandle = Value then
    exit;
  { If value is $FFFFFFFF then stop controlling the COM port
    without closing in }
  if Value = $FFFFFFFF then
  begin
    if Connected then
      { Stop the timer }
      if Connected then
        KillTimer( FNotifyWnd, 1 );
    { No more connected }
    FComPortHandle := 0;
  end
  else
  begin
    { Disconnect }
```

```
      Disconnect;
      { If Value is = 0 then exit now }
      { (ComPortHandle := 0 acts as Disconnect) }
      if Value = 0  then
        exit;
      { Set COM port handle }
      FComPortHandle := Value;
      { Start the timer (used for polling) }
      SetTimer( FNotifyWnd, 1, FComPortPollingDelay, nil );
    end;
  end;
  procedure TCommPortDriver.SetComPort( Value: TComPortNumber );
  begin
    // Be sure we are not using any COM port
    if Connected then
      exit;
    // Change COM port
    FComPort := Value;
  end;
  procedure TCommPortDriver.SetComPortBaudRate( Value: TComPortBaudRate );
  begin
    // Set new COM speed
    FComPortBaudRate := Value;
    // Apply changes
    if Connected then
      ApplyCOMSettings;
  end;
  procedure TCommPortDriver.SetComPortDataBits( Value: TComPortDataBits );
  begin
    // Set new data bits
    FComPortDataBits := Value;
    // Apply changes
    if Connected then
      ApplyCOMSettings;
  end;
  procedure TCommPortDriver.SetComPortStopBits( Value: TComPortStopBits );
  begin
    // Set new stop bits
    FComPortStopBits := Value;
    // Apply changes
    if Connected then
      ApplyCOMSettings;
  end;
  procedure TCommPortDriver.SetComPortParity( Value: TComPortParity );
  begin
    // Set new parity
    FComPortParity := Value;
    // Apply changes
    if Connected then
      ApplyCOMSettings;
  end;
  procedure TCommPortDriver.SetComPortHwHandshaking(Value: TComPortHwHandshaking );
  begin
    // Set new hardware handshaking
    FComPortHwHandshaking := Value;
    // Apply changes
    if Connected then
      ApplyCOMSettings;
  end;
  procedure TCommPortDriver.SetComPortSwHandshaking(Value: TComPortSwHandshaking );
  begin
    // Set new software handshaking
    FComPortSwHandshaking := Value;
    // Apply changes
    if Connected then
      ApplyCOMSettings;
  end;
  procedure TCommPortDriver.SetComPortInBufSize( Value: word );
  begin
    { Do nothing if connected }
    if Connected then
      exit;
```

```
  // Free the temporary input buffer
  FreeMem( FTempInBuffer, FComPortInBufSize );
  // Set new input buffer size
  FComPortInBufSize := Value;
  // Allocate the temporary input buffer
  GetMem( FTempInBuffer, FComPortInBufSize );
end;
procedure TCommPortDriver.SetComPortOutBufSize( Value: word );
begin
  { Do nothing if connected }
  if not Connected then
    exit;
  // Set new output buffer size
  FComPortOutBufSize := Value;
end;
procedure TCommPortDriver.SetComPortPollingDelay( Value: word );
begin
  // If new delay is not equal to previous value...
  if Value <> FComPortPollingDelay then
  begin
    // Stop the timer
    if Connected then
      KillTimer( FNotifyWnd, 1 );
    // Store new delay value
    FComPortPollingDelay := Value;
    // Restart the timer
    if Connected then
      SetTimer( FNotifyWnd, 1, FComPortPollingDelay, nil );
  end;
end;

const
  Win32BaudRates: array[br110..br115200] of DWORD =
    ( CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400, CBR_4800, CBR_9600,
      CBR_14400, CBR_19200, CBR_38400, CBR_56000, CBR_57600, CBR_115200{v1.02
removed: CRB_128000, CBR_256000} );

const
  dcb_Binary              = $00000001;
  dcb_ParityCheck         = $00000002;
  dcb_OutxCtsFlow         = $00000004;
  dcb_OutxDsrFlow         = $00000008;
  dcb_DtrControlMask      = $00000030;
  dcb_DtrControlDisable   = $00000000;
  dcb_DtrControlEnable    = $00000010;
  dcb_DtrControlHandshake = $00000020;
  dcb_DsrSensivity        = $00000040;
  dcb_TXContinueOnXoff    = $00000080;
  dcb_OutX                = $00000100;
  dcb_InX                 = $00000200;
  dcb_ErrorChar           = $00000400;
  dcb_NullStrip           = $00000800;
  dcb_RtsControlMask      = $00003000;
  dcb_RtsControlDisable   = $00000000;
  dcb_RtsControlEnable    = $00001000;
  dcb_RtsControlHandshake = $00002000;
  dcb_RtsControlToggle    = $00003000;
  dcb_AbortOnError        = $00004000;
  dcb_Reserveds           = $FFFF8000;

// Apply COM settings.
procedure TCommPortDriver.ApplyCOMSettings;
var dcb: TDCB;
begin
  // Do nothing if not connected
  if not Connected then
    exit;
  // Clear all
  fillchar( dcb, sizeof(dcb), 0 );
  // Setup dcb (Device Control Block) fields
  dcb.DCBLength := sizeof(dcb); // dcb structure size
  dcb.BaudRate := Win32BaudRates[ FComPortBaudRate ]; // baud rate to use
```

```
    // Set fBinary: Win32 does not support non binary mode transfers
    // (also disable EOF check)
    dcb.Flags := dcb_Binary;
    if EnableDTROnOpen then
      { Enabled the DTR line when the device is opened and leaves it on }
      dcb.Flags := dcb.Flags or dcb_DtrControlEnable;

    case FComPortHwHandshaking of // Type of hw handshaking to use
      hhNONE:; // No hardware handshaking
      hhRTSCTS: // RTS/CTS (request-to-send/clear-to-send) hardware handshaking
        dcb.Flags := dcb.Flags or dcb_OutxCtsFlow or dcb_RtsControlHandshake;
    end;
    case FComPortSwHandshaking of // Type of sw handshaking to use
      shNONE:; // No software handshaking
      shXONXOFF: // XON/XOFF handshaking
        dcb.Flags := dcb.Flags or dcb_OutX or dcb_InX;
    end;
    dcb.XONLim := FComPortInBufSize div 4; // Specifies the minimum number of bytes
allowed
                                           // in the input buffer before the XON
character is sent
                                           // (or CTS is set)
    dcb.XOFFLim := 1; // Specifies the maximum number of bytes allowed in the input
buffer
                      // before the XOFF character is sent. The maximum number of
bytes
                      // allowed is calculated by subtracting this value from the
size,
                      // in bytes, of the input buffer
    dcb.ByteSize := 5 + ord(FComPortDataBits); // how many data bits to use
    dcb.Parity := ord(FComPortParity); // type of parity to use
    dcb.StopBits := ord(FComPortStopbits); // how many stop bits to use
    dcb.XONChar := #17; // XON ASCII char
    dcb.XOFFChar := #19; // XOFF ASCII char
    SetCommState( FComPortHandle, dcb );
    { Flush buffers }
    FlushBuffers( true, true );
    // Setup buffers size
    SetupComm( FComPortHandle, FComPortInBufSize, FComPortOutBufSize );
end;
function TCommPortDriver.Connect: boolean;
var comName: array[0..4] of char;
    tms: TCOMMTIMEOUTS;
begin
    // Do nothing if already connected
    Result := Connected;
    if Result then
      exit;
    // Open the COM port
    StrPCopy( comName, 'COM' );
    comName[3] := chr( ord('1') + ord(FComPort) );
    comName[4] := #0;
    FComPortHandle := CreateFile(
                                  comName,
                                  GENERIC_READ or GENERIC_WRITE,
                                  0, // Not shared
                                  nil, // No security attributes
                                  OPEN_EXISTING,
                                  FILE_ATTRIBUTE_NORMAL,
                                  0 // No template
                                ) ;
    Result := Connected;
    if not Result then
      exit;
    // Apply settings
    ApplyCOMSettings;
    // Setup timeouts: we disable timeouts because we are polling the com port!
    tms.ReadIntervalTimeout := 1; // Specifies the maximum time, in milliseconds,
                                  // allowed to elapse between the arrival of two
                                  // characters on the communications line
    tms.ReadTotalTimeoutMultiplier := 0; // Specifies the multiplier, in
milliseconds,
```

```
                                        // used to calculate the total time-out
period
                                        // for read operations.
  tms.ReadTotalTimeoutConstant := 1; // Specifies the constant, in milliseconds,
                                     // used to calculate the total time-out
period
                                     // for read operations.
  tms.WriteTotalTimeoutMultiplier := 0; // Specifies the multiplier, in
milliseconds,
                                        // used to calculate the total time-out
period
                                        // for write operations.
  tms.WriteTotalTimeoutConstant := 0; // Specifies the constant, in milliseconds,
                                      // used to calculate the total time-out
period
                                      // for write operations.
  SetCommTimeOuts( FComPortHandle, tms );
  // Start the timer (used for polling)
  SetTimer( FNotifyWnd, 1, FComPortPollingDelay, nil );
end;
procedure TCommPortDriver.Disconnect;
begin
  if Connected then
  begin
    // Stop the timer (used for polling)
    KillTimer( FNotifyWnd, 1 );
    // Release the COM port
    CloseHandle( FComPortHandle );
    // No more connected
    FComPortHandle := 0;
  end;
end;
function TCommPortDriver.Connected: boolean;
begin
  Result := FComPortHandle > 0;
end;
// v1.02: flish rx/rx buffers
procedure TCommPortDriver.FlushBuffers( inBuf, outBuf: boolean );
var dwAction: DWORD;
begin
  if not Connected then
    exit;
  // Flush the incoming data buffer
  dwAction := 0;
  if outBuf then
    dwAction := dwAction or PURGE_TXABORT or PURGE_TXCLEAR;
  if inBuf then
    dwAction := dwAction or PURGE_RXABORT or PURGE_RXCLEAR;
  PurgeComm( FComPortHandle, dwAction );
end;
// v1.02: returns the output buffer free space or 65535 if
//        not connected }
function TCommPortDriver.OutFreeSpace: word;
var stat: TCOMSTAT;
    errs: DWORD;
begin
  if not Connected then
    Result := 65535
  else
  begin
    ClearCommError( FComPortHandle, errs, @stat );
    Result := FComPortOutBufSize - stat.cbOutQue;
  end;
end;
// Send data
{function TCommPortDriver.SendData( DataPtr: pointer; DataSize: integer ):
boolean;
var nsent: DWORD;
begin
  Result := WriteFile( FComPortHandle, DataPtr^, DataSize, nsent, nil );
  Result := Result and (nsent=DataSize);
end;}
```

```
{ Send data (breaks the data in small packets if it doesn't fit in the output
  buffer) }
function TCommPortDriver.SendData( DataPtr: pointer; DataSize: integer ):
integer;
var nToSend, nsent: integer;
    t1: longint;
begin
  { 0 bytes sent }
  Result := 0;
  { Do nothing if not connected }
  if not Connected then
    exit;
  { Current time }
  t1 := GetTickCount;
  { Loop until all data sent or timeout occurred }
  while DataSize > 0 do
  begin
    { Get output buffer free space }
    nToSend := OutFreeSpace;
    { If output buffer has some free space... }
    if nToSend > 0 then
    begin
      { Don't send more bytes than we actually have to send }
      if nToSend > DataSize then
        nToSend := DataSize;
      { Send }
      WriteFile( FComPortHandle, DataPtr^, DataSize, nsent, nil );
      { Update number of bytes sent }
      Result := Result + abs(nsent);
      { Decrease the count of bytes to send }
      DataSize := DataSize - abs(nsent);
      { Get current time }
      t1 := GetTickCount;
      { Continue. This skips the time check below (don't stop
        trasmitting if the FOutputTimeout is set too low) }
      continue;
    end;
    { Buffer is full. If we are waiting too long then
      invert the number of bytes sent and exit }
    if (GetTickCount-t1) > FOutputTimeout then
    begin
      Result := -Result;
      exit;
    end;
  end;
end;
// Send a pascal string (NULL terminated if $H+ (default))
function TCommPortDriver.SendString( s: string ): boolean;
var len: integer;
begin
  len := length( s );
  {$IFOPT H+}
  // New syle pascal string (NULL terminated)
  Result := SendData( pchar(s), len ) = len;
  {$ELSE}
  // Old style pascal string (s[0] = length)
  Result := SendData( pchar(@s[1]), len ) = len;
  {$ENDIF}
end;
// v1.02: send a C-style strings (NULL terminated)
function TCommPortDriver.SendZString( s: pchar ): boolean;
var len: integer;
begin
  len := strlen( s );
  Result := SendData( s, len ) = len;
end;
// v1.02: set DTR line high (onOff=TRUE) or low (onOff=FALSE).
//        You must not use HW handshaking.
procedure TCommPortDriver.ToggleDTR( onOff: boolean );
const funcs: array[boolean] of integer = (CLRDTR,SETDTR);
begin
  if Connected then
```

```
    EscapeCommFunction( FComPortHandle, funcs[onOff] );
end;

// v1.02: set RTS line high (onOff=TRUE) or low (onOff=FALSE).
//        You must not use HW handshaking.
procedure TCommPortDriver.ToggleRTS( onOff: boolean );
const funcs: array[boolean] of integer = (CLRRTS,SETRTS);
begin
  if Connected then
    EscapeCommFunction( FComPortHandle, funcs[onOff] );
end;
// COM port polling proc
procedure TCommPortDriver.TimerWndProc( var msg: TMessage );
var nRead: dword;
begin
  if (msg.Msg = WM_TIMER) and Connected then
  begin
    nRead := 0;
    if ReadFile( FComPortHandle, FTempInBuffer^, FComPortInBufSize, nRead, nil )
then
      if (nRead <> 0) and Assigned(FComPortReceiveData) then
        FComPortReceiveData( Self, FTempInBuffer, nRead );
  end;
end;
procedure Register;
begin
  { Register this component and show it in the 'System' tab
    of the component palette }
  RegisterComponents('System', [TCommPortDriver]);
end;
end.
```

ภาคผนวก ข

คำสั่งโมเด็มที่ปรากฏในโปรแกรม

ชุดคำสั่ง AT ที่สำคัญ

Basic "AT" Command Set

+++ ให้เปลี่ยนจากโหมดการส่งข้อมูลเป็นโหมดคำสั่ง และในขณะที่อยู่ในโหมดคำสั่ง ผู้ใช้สามารถติดต่อโดยตรงกับโมเด็ม โดยใช้กลุ่มคำสั่ง "AT" และถ้าต้องการกลับสู่โหมดการส่งข้อมูลให้ใช้คำสั่ง AT0

ATA เข้าสู่โหมดการตอบรับ

ATDn ให้โมเด็มยกหูโทรศัพท์ขึ้นและหมุนหมายเลขโทรศัพท์ออก ด้วยหมายเลขที่กำหนดไว้ในพารามิเตอร์ต่าง ๆดังนี้

!	วางหูโทรศัพท์ชั่วคราว
,	หยุดรอเป็นเวลา 2 วินาที ขณะหมุนหมายเลขโทรศัพท์
;	หลังจากหมุนหมายเลขแล้วจะส่งไปที่สภาวะคำสั่ง
P	หมุนหมายเลขด้วยระบบ Pulse
R	เรียกโหมดรีเวิร์ส
S = x	เก็บหมายเลขโทรศัพท์ไว้ที่พื้นที่ x
T	หมุนหมายเลขด้วยระบบ Tone
W	รอฟังเสียงสัญญาณ "สายว่าง"

ATEn กำหนดโหมดการแสดงผล จากการใช้คีย์บอร์ด

n = 0 Echo off (ไม่ให้แสดงผลจากการใช้คีย์บอร์ด)

n = 1 Echo on (ให้แสดงผลจากการใช้คีย์บอร์ด)

ATHn ควบคุมสถานะการวางสาย

n = 0 ให้สร้างสถานะการวางสาย

n = 1 ให้สร้างสถานะการยกหูโทรศัพท์ขึ้น

ATLn ควบคุมความดังของเสียง

n = 0 ให้มีเสียงเบาที่สุด

n = 1 ให้มีเสียงดังปานกลาง

n = 2 ให้มีเสียงดังที่สุด

ATMn ควบคุมการเกิดเสียงของโมเด็ม

n = 0 ให้โมเด็มปิดเสียงทั้งหมด

n = 1 ให้โมเด็มส่งเสียงได้ในขณะที่หมุนหมายเลข และปิดเสียงเมื่อเชื่อมต่อกับโมเด็มปลายทางเรียบร้อยแล้ว

n = 2 ให้โมเด็มส่งเสียงออกจากลำโพงตลอดเวลา

n = 3 ให้โมเด็มเปิดเสียงจนกระทั่งได้รับสัญญาณพาหะจากโมเด็มปลายทางจึงจะปิดเสียง

ATQn ควบคุม Result code

n = 0 ให้โมเด็มส่ง Result code ออกมาได้

n = 1 ไม่ให้โมเด็มส่ง Result code ออกมา

n = 2 ให้โมเด็มส่ง Result code ออกมา ถ้าเป็นโมเด็มตัวต้นทางและไม่ให้ส่ง Result code ออกมา ถ้าเป็นโมเด็มตัวปลายทาง

ATSn=y คำสั่งตั้งค่าให้แก่รีจิสเตอร์ S จะช่วยให้สามารถดูและเปลี่ยนแปลงค่าที่อยู่ในรีจิสเตอร์ S ต่าง ๆ ได้ สำหรับการตั้งค่ารีจิสเตอร์มีรูปแบบดังนี้ ATSn=y โดยที่ n เป็นหมายเลขรีจิสเตอร์ S และ y หมายถึงค่าที่ต้องการบรรจุลงไปในรีจิสเตอร์นั้น ๆ สำหรับรีจิสเตอร์ S ที่สำคัญต่าง ๆ จะมีดังต่อไปนี้

S0 : ใช้เก็บค่าจำนวนเสียงกระดิ่ง ก่อนที่โมเด็มจะรับสายโทรศัพท์ สามารถตั้งค่าให้สูงได้ถึง 255 หรือถ้าตั้งเป็น 0 หมายความว่า ไม่ให้โมเด็มรับสายโทรเข้า

S1 : ใช้เก็บค่าจำนวนครั้งเสียงกระดิ่ง เป็นรีจิสเตอร์แบบอ่านได้อย่างเดียว

S2 : ใช้เก็บค่ารหัส ASCII (ในลักษณะเลขฐาน 10) ที่ต้องการใช้เป็นรหัส Escape Sequence โดยปกติจะมีดีฟอลต์เป็น 43 ซึ่งตรงกับตัวอักษร "+"

S3 : ใช้เก็บค่ารหัส ASCII (ในลักษณะเลขฐาน 10) ที่ต้องการใช้เป็นรหัส Carriage Return โดยปกติจะมีดีฟอลต์เป็น 13 ซึ่งจะเหมือนการกดปุ่ม Enter

S4 : ใช้เก็บค่ารหัส ASCII (ในลักษณะเลขฐาน 10) ที่ต้องการใช้เป็นรหัส Line Feed โดยปกติจะมีดีฟอลต์เป็น 10

S5 : ใช้เก็บค่ารหัส ASCII (ในลักษณะเลขฐาน 10) ที่ต้องการใช้เป็นรหัส BackSpace โดยปกติจะมีดีฟอลต์เป็น 8 ซึ่งเหมือนการกดปุ่ม BackSpace

S6 : ใช้เก็บค่าเวลาในการหยุดรอ ก่อนที่จะหมุนหมายเลขโทรศัพท์ออก มีหน่วยเป็นวินาที ซึ่งค่าในรีจิสเตอร์ S6 นี้จะใช้งานก็ต่อเมื่อ โมเด็มถูกสั่งให้ไม่สนใจฟังไดอัลโทน หรือคำสั่ง ATXn หลังจากนั้นเมื่อต้องการให้โมเด็มหมุนออก โมเด็มก็จะไม่สนใจว่ามีเสียงไดอัลโทนหรือไม่ และจะรอเป็นเวลา S6 วินาทีจึงหมุนออก

S7 : ใช้เก็บค่าเวลาที่โมเด็มรอฟังสัญญาณพาหะของโมเด็มปลายทางมีหน่วยเป็นวินาที หลังจากที่โมเด็มหมุนออกหรือรับสายเข้า โมเด็มจะรอฟังสัญญาณพาหะเป็นเวลา S7 วินาที

S8 : ใช้เก็บค่าเวลาในการรอ มีหน่วยเป็นวินาที หลังจากที่โมเด็มได้รับพารามิเตอร์ "," ในคำสั่ง ATD โมเด็มก็จะหยุดรอเป็นเวลา S8 วินาที ก่อนที่จะทำคำสั่งหรือพารามิเตอร์ตัวต่อไป

S9 : ใช้เก็บค่าของช่วงเวลาที่โมเด็มรอรับทราบสัญญาณพาหะของโมเด็มปลายทาง มีหน่วยเป็น 1/10 วินาที โดยที่โมเด็มจะรอสัญญาณพาหะจากปลายทางภายในช่วงเวลาดังกล่าว แล้วจากนั้นโมเด็มก็จะส่งสัญญาณ CD ที่ขา 8 ของขั้วต่อ RS-232-C ไปยัง PC

S10 : มีหน่วยเป็น 1/10 วินาที มีค่าได้ตั้งแต่ 1 จนถึง 255 มักจะใช้อยู่ 2 กรณี คือ กรณีแรกจะใช้เก็บช่วงเวลาที่โมเด็มจะทำการรอสัญญาณพาหะที่ขาดหายไปในขณะกำลังออนไลน์

ซึ่งอาจจะเกิดจากสัญญาณรบกวน และถ้าสัญญาณพาหะยังคงขาดตอนไปในช่วงเวลาดังกล่าว โมเด็มก็จะวางสายทันที ส่วนในกรณีที่สองก็คือ ใช้เก็บค่าของช่วงเวลาในการรอสัญญาณพาหะก่อนที่จะวางสาย เมื่อทำการวางสายแล้วจะให้โมเด็มตรวจสอบให้แน่ใจว่าการสื่อสารถูกตัดขาดอย่างแน่นอนหรือไม่

ATVn กำหนด Verbose Mode คำสั่งนี้ใช้กำหนดชนิดของ Result Code ว่าจะให้ส่งเป็นแบบตัวอักษรหรือแบบตัวเลข โดยมีรูปแบบดังนี้

n = 0 ให้โมเด็มส่ง Result Code เป็นตัวเลข

n = 1 ให้โมเด็มส่ง Result Code เป็นตัวอักษร

สำหรับค่าตอบสนองแสดงไว้ในตาราง ผ.1

ATXn กำหนดชนิดของ Result Code ดูได้จากตาราง ผ.1

Result Code	X0	X1	X2	X3	X4
0/OK	*	*	*	*	*
1/CONNECT	*	*	*	*	*
2/RING	*	*	*	*	*
3/NO CARRIER	*	*	*	*	*
4/ERROR	*	*	*	*	*
5/CONNECT 1200		*	*	*	*
6/NO DIAL TONE			*		*
7/BUSY				*	*
8/NO ANSWER				*	*
9/Reserved					
10/CONNECT 2400		*	*	*	*
11/RINGING					*
13/CONNECT 9600		*	*	*	*
18/CONNECT 4800		*	*	*	*
20/CONNECT 7200		*	*	*	*
21/CONNECT 12000		*	*	*	*
25/CONNECT 14400		*	*	*	*
43/CONNECT 16800		*	*	*	*
85/CONNECT 19200		*	*	*	*
91/CONNECT 21600		*	*	*	*
99/ CONNECT 24000		*	*	*	*
103/ CONNECT 26400		*	*	*	*

107/ CONNECT 28800		*	*	*	*
151/ CONNECT 31200		*	*	*	*
155/ CONNECT 33600	*	*	*	*	*
180/ CONNECT 33333	*	*	*	*	*
184/ CONNECT 37333	*	*	*	*	*
188/ CONNECT 41333	*	*	*	*	*
192/ CONNECT 42666	*	*	*	*	*
196/ CONNECT 44000	*	*	*	*	*
200/ CONNECT 45333	*	*	*	*	*
204/ CONNECT 46666	*	*	*	*	*
208/ CONNECT 48000	*	*	*	*	*
212/ CONNECT 49333	*	*	*	*	*
216/ CONNECT 50666	*	*	*	*	*
220/ CONNECT 52000	*	*	*	*	*
224/ CONNECT 53333	*	*	*	*	*
228/ CONNECT 54666	*	*	*	*	*
232/ CONNECT 56000	*	*	*	*	*
236/ CONNECT 57333	*	*	*	*	*
Adaptive Dialing			*	*	*
Wait for 2nd Dial Tone(W)			*		*
Wait for Answer @				*	*
Fast Dial			*		*

*Requires @ in dial string; replaces NO CARRIER.

Extended "AT&" Command Set

&Cn กำหนดโหมดการตรวจจับคลื่นพาห์

n=0 ให้มีการตรวจตลอดเวลา

n=1 ให้มีการตรวจเฉพาะเมื่อมีโอกาสที่จะมีคลื่นพาห์เข้ามา

&Dn โหมดการตรวจสอบสัญญาณ DTR (Data Terminal Ready)

n=0 ไม่ทำการตรวจสอบ โดยสมมติว่ามี DTR อยู่ตลอดเวลา

n=1 ถ้า DTR เปลี่ยนจากสถานะ on เป็น off จะทำให้ไม่มีการหมุนหมายเลขใด ๆเกิดขึ้น

n=2 ถ้า DTR มีสถานะ off จะสร้างสถานะการวางสาย

n=3 ถ้า DTR มีสถานะ off จะสร้างสถานะการวางสาย และรีเซ็ตโมเด็ม

ภาคผนวก ค

ความหมายของคำสั่งเคลไฟล์โดยทั่วไป

Absolute

เป็นวิธีการกำหนดชนิดของตัวแปรโดยให้มีการจองเนื้อที่หน่วยความจำตามที่ระบุ หรือให้ใช้หน่วยความจำเดียวกันกับตัวแปรอื่นก็ได้

AssignFile(f,st) Proc :

กำหนดให้ตัวแปรไฟล์ f (ตามแบบของไฟล์ที่กำหนด) แทนชื่อไฟล์ใน st

BlockRead(f,v,cnt[,res]) Proc :

อ่านข้อมูลของไฟล์ไม่กำหนดแบบ f ให้แก่ตัวแปรอาร์เรย์ของอักขระ v เป็นจำนวน cnt ไบต์ ให้ผลคือ จำนวนที่อ่านได้จริงแก่ตัวแปรเลขจำนวนเต็ม res

BlockWrite (f,v,cnt[,res]) Proc :

เขียนข้อมูลในตัวแปรของอักขระ v ลงไฟล์ไม่กำหนดแบบ f ให้แก่ตัวแปรอาร์เรย์ของอักขระ v เป็นจำนวน cnt ไบต์ ให้ผลคือ จำนวนที่เขียนได้จริงแก่ตัวแปรเลขจำนวนเต็ม res

Break Proc :

ให้ออกจากลูป for,while หรือ repeat

ChDir(st) Proc :

เปลี่ยนไดเรกทอรี่เป็นตามที่กำหนดในสตริง st หรือรวมทั้ง st เป็นไดรฟ์

Chr(n) :Char Func :

ให้ค่าซึ่งเป็นผลจากการแปลงค่าเลขจำนวนเต็มไบต์ n เป็นข้อมูลแบบอักขระ เช่น Chr(65) ให้ค่าเป็นอักขระ 'A'

CloseFile(f) Proc :

ปิดไฟล์ f ซึ่งเปิดอยู่

Copy(st,p,cnt) : String Func :

ให้ค่าในสตริง st จากตำแหน่ง p ไปเป็นจำนวน cnt อักขระ

Dispose(p) Proc :

ให้ยกเลิกชั้นข้อมูลที่กำหนดขึ้นด้วย New ซึ่งตัวแปร p ชี้อยู่ในขณะนั้น

EOF(f) :Boolean Func :

ให้ค่าความเป็นจริงเมื่อไฟล์พอยเตอร์ของไฟล์ f อยู่ที่ท้ายไฟล์ (สุดไฟล์)

Exit Proc :

ให้ออกจากโปรแกรมย่อยนี้ แต่ถ้าใช้ในโปรแกรมหลักจะยุติการทำงานของโปรแกรม

ExtracFileName(st) : String Func : SysUtils

ให้ค่าเป็นชื่อและส่วนขยายของไฟล์ ในตัวแปรสตริง st ซึ่งมีค่าเป็น path+filename

FileSize(f) : LongInt Func :

ให้ขนาดไฟล์ f (ขนาดตามแบบข้อมูล เช่น ถ้าเป็นไฟล์ของอักขระจะเป็นจำนวนของอักขระ ถ้าเป็นไฟล์ของเรคอร์ดจะเป็นขนาดเรคอร์ด

GetDir(D,st) Proc :

ให้ค่าเป็นไดเรกทอรี่ของไดรฟ์ปัจจุบัน D แก่ตัวแปรสตริง stD มีค่าได้เป็น 0 หมายถึงไดร์ฟปัจจุบัน 1 หมายถึงไดร์ฟ A 2 หมายถึงไดร์ฟ B 3 หมายถึงไดร์ฟ C

Inc(v,[,n]) Proc :

เพิ่มค่าในตัวแปรแบบมีลำดับที่ v ขึ้น n ถ้าไม่กำหนดค่า n จะเพิ่มทีละ 1

IntToStr(n) : String Func :

เปลี่ยนจำนวน n เป็นสตริงของตัวเลข เช่น st := IntToStr(126); หมายถึงให้ค่าแก่ตัวแปรสตริง st = '126'

IOResult : Integer Func :

ให้ Dos Error Code เมื่อมีการดำเนินกรรมวิธีกับไฟล์ (I/O stream) ค่า 0 คือ ไม่มี error

Length(st) : Integer Func :

ให้ค่าความยาวของสตริง st

MessageDlg(Msg,Atype,Abuttons,HelpCtx) : Word Func : Dialogs

แสดง Message Dialog Box ที่กลางหน้าจอ โดยจะมีชนิดของปุ่มให้ผู้ใช้ เลือก เช่น Yes, No, OK เป็นต้น เมื่อผู้ใช้เลือกใช้แล้วก็จะส่งผลที่ผู้ใช้เลือกกลับไปยังโปรแกรม Msg เป็นข้อมูลสตริง string Atype เป็นแบบของ Message Box ที่จะแสดงบนหน้าจอ Abuttons เป็นแบบของปุ่มที่จะแสดงบนหน้าจอ HelpCtx เป็นแบบของ help สำหรับสนับสนุน dialog box นั้น

Pos(w,st) : Byte Func :

ให้ตำแหน่งเริ่มต้นของสตรง w ในสตริง st

ProcessMessages Proc :

ขัดจังหวะการทำงานของโปรแกรม เพื่อให้วินโดว์ตอบสนองคำสั่งของผู้ใช้ได้ เช่น สั่งให้โปรแกรมตอบสนองการกดแป้นพิมพ์ หรือการเลื่อนเมาท์ของผู้ใช้

Reset(f[,n]) Proc :

เปิดไฟล์ทุกชนิด f เพื่ออ่านหรือเขียนคราวละ n เรคอร์ด ซึ่ง n จะใช้เมื่อ f เป็นไฟล์แบบไม่กำหนดแบบข้อมูลเท่านั้น เมื่อเปิดแล้วไฟล์พอยเตอร์จะอยู่ที่ต้นไฟล์

Result Proc :

ใช้ในส่วนโปรแกรมย่อยที่เป็นฟังก์ชันเท่านั้น เพื่อให้ส่งผลของฟังก์ชันไปให้โปรแกรมหลัก

ReWrite(f,n) Proc :

เปิดไฟล์ f เพื่อการเขียนคราวละ n เรคอร์ด ซึ่ง n จะใช้เมื่อ f เป็นไฟล์แบบไม่กำหนดแบบข้อมูลเท่านั้น ถ้าเป็นไฟล์ข้อมูลเดิมจะถูกยกเลิก ถ้าเป็นไฟล์ใหม่จะสร้างไฟล์ขึ้นในดิสก์ เมื่อเปิดแล้วไฟล์พอยเตอร์จะอยู่ที่ต้นไฟล์

SizeOf(v) :Word Func :

ให้ขนาดข้อมูลหรือแบบข้อมูล v จำนวนเป็นไบต์

StrToInt(st) : LongInt Func :

เปลี่ยนสตริงของตัวเลข st เป็นเลขจำนวน n

เช่น n :=StrToInt('23'); หมายถึงให้ค่าแก่ตัวแปรเลขจำนวน n = 23

ภาคผนวก ง

ความหมายของคำสั่งต่าง ๆ ของคอมโพเนนท์ CommPortDriver

CommPortDriver.Connect

เป็นฟังก์ชั่นติดต่อและจองพอร์ตอนุกรม

CommPortDriver.Disconnect

เป็นฟังก์ชั่นยกเลิกการตดต่อและของพอร์ตอนุกรม

CommPortDriver.Connected

เป็นฟังก์ชั่นตรวจสอบการจองพอร์ตอนุกรม

CommPortDriver.FlushBuffers(Inuf;OutBuf : Boolean)

เป็นฟังก์ชั่นทำการล้างค่าในบัฟเฟอร์ที่เก็บข้อมูลอนุกรม

CommPortDriver.SendData(DataPtr : pointer; DataSize : Integer)

ส่งข้อมูลซึ่งชี้ด้วยพอยเตอร์เป็นจำนวนเต็ม จำนวนเท่ากับ DataSize ออกไปยังพอร์ตอนุกรม

CommPortDriver.SendString(s: string)

เป็นฟังก์ชั่นส่งข้อมูลที่เป็นสตริงไปยังพอร์ตอนุกรม

CommPortDriver.SendZString(s : pchar)

เป็นฟังก์ชั่นส่งข้อมูลที่เป็นพอยเตอร์ของอักขระ

Property Comport บอกว่าเป็นพอร์ตอนุกรมตัวใด

Property ComportSpeed อัตราความเร็วในการส่งข้อมูลของพอร์ตอนุกรม

Property ComportDataBits จำนวนบิทข้อมูล

Property ComportStopBits จำนวนสตอพบิท

Property ComportParity จำนวนพาริตี้

Property ComportHwHandshaking กำหนดการควบคุมการไหลของข้อมูลแบบ Hardware

Property ComportSwHandshaking กำหนดการควบคุมการไหลของข้อมูลแบบ Software

Property ComportInBufSize กำหนดบัฟเฟอร์ขาเข้า

Property ComportOutBufSize กำหนดบัฟเฟอร์ขาออก

Property CommPortPollingDelay กำหนดค่าหน่วงเวลา

ภาคผนวก