ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

---

**ปริญญานิพนธ์** โปรแกรมให้จังหวะดนตรีและคำร้อง

SONG AND MELODY PROGRAM

**ชื่อนักศึกษา** 1. นางสาวกัญญาณี แจ่มสุธี รหัสประจำตัว 37031402

2. นายนักรบ รุ่งแก้ว รหัสประจำตัว 37031409

3. นายพรชัย ยิ่งเจริญธนา รหัสประจำตัว 37031416

**หลักสูตร** ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

**อาจารย์ผู้ควบคุมปริญญานิพนธ์**

1. อาจารย์วรวิทย์ สมหา
2. ดร.สุรสิทธิ์ ราตรี
3. อาจารย์สุชิน อาจหาญ

| คณะกรรมการสอบปริญญานิพนธ์ | ลายมือชื่อ |
|---|---|
| 1. ดร.สุรสิทธิ์ ราตรี | |
| 2. อาจารย์วรวิทย์ สมหา | |
| 3. อาจารย์สุชิน อาจหาญ | |
| 4. อาจารย์โกศล ตราชู | |
| 5. อาจารย์สันติ ตันตระกูล | |

**วันเดือนปีที่สอบ** วันที่ 11 ธันวาคม 2538 **เวลา** 21.00 ถึง 22.00 น.

**สถานที่สอบ** ห้อง ค.303 คณะครุศาสตร์อุตสาหกรรม

ภาควิชารับรองแล้ว

.....

(ศาสตราจารย์ ธีระพล เทพหัสดิน ณ อยุธยา)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ ..... เดือน ..... พ.ศ. .....

# ปริญญานิพนธ์

# โปรแกรมให้จังหวะดนตรีและคำร้อง

# SONG AND MELODY PROGRAM



นางสาวกัญญาณี แจ่มสุธี
นายนักรบ รุ่งแก้ว
นายพรชัย ยิ่งเจริญธนา



ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต
สาขาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2538

**ปริญญานิพนธ์**

**เรื่อง** โปรแกรมให้จังหวะดนตรีและคำร้อง

SONG AND MELODY PROGRAM

**ผู้จัดทำ**

นางสาวกัญญาณี แจ่มสุธี

นายนักรบ รุ่งแก้ว

นายพรชัย ยิ่งเจริญธนา

**อาจารย์ที่ปรึกษา**

ลงนาม .....

(อาจารย์วรวิทย์ สมหา)

ลงนาม .....

(อาจารย์สุชิน อาจหาญ)

ลงนาม .....

(ดร.สุรสิทธิ์ ราตรี)

**หัวหน้าภาควิชา**

ลงนาม .....

(ผศ.ดร.ธีรพล เทพหัสดิน ณ อยุธยา)

# ปริญญานิพนธ์

## โปรแกรมให้จังหวะดนตรีและคำร้อง

## SONG AND MELODY PROGRAM

**วัตถุประสงค์**

1. เพื่อศึกษาระบบการทำงานของวินโดวส์ , การเขียนโปรแกรมภาษา C++ บนวินโดวส์ และการทำงานของการ์ดเสียง
2. เพื่อออกแบบโปรแกรมที่สามารถทำงานบนวินโดวส์ให้ควบคุมการ์ดเสียงได้
3. เพื่อสร้างโปรแกรมฝึกร้องเพลงโดยใช้คอมพิวเตอร์ที่ทำงานบนวินโดวส์ได้
4. เพื่อนำโปรแกรมที่สร้างขึ้นไปประยุกต์ใช้งานในระบบมัลติมิเดีย (multimedia)

**ประโยชน์ที่คาดว่าจะได้รับจากการทำปริญญานิพนธ์**

1. รู้จักระบบการทำงานของวินโดวส์ , เขียนโปรแกรมบนวินโดวส์ และ รู้ถึงหลักการทำงานของการ์ดเสียง
2. สามารถใช้โปรแกรมที่ทำงานบนวินโดวส์ให้ควบคุมการ์ดเสียงได้
3. สามารถใช้คอมพิวเตอร์ที่ทำงานบนวินโดวส์สร้างโปรแกรมฝึกร้องเพลงได้
4. สามารถนำโปรแกรมไปประยุกต์ใช้ในระบบมัลติมิเดีย

# SONG AND MELODY PROGRAM

MISS.KANYANEE JAMSUTHEE

MR. NAKROB RUNGKAEW

MR. PORNCHAI YINGCHAROENTHANA

**ADVISERS**

MR. WORAWIT SOMHA

MR. SUCHIN ARTHAN

DR. SURASIT RATHEE

1995

**ABSTRACT**

This thesis presents a song and melody program , it is using MCI (Multimedia Control Interface ) windows controlled between user and sound card . User can be put the content and selected a song .

# กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ สำเร็จลุล่วงมาได้ เพราะพระคุณของ คุณพ่อ คุณแม่ ที่ให้กำลังใจและให้ความสนันสนุนมาโดยตลอด และขอขอบพระคุณ อาจารย์ วรวิทย์ สมหา , อาจารย์สุชิน อาจหาญ และ ดร. สุรสิทธิ์ ราตรี รวมทั้งอาจารย์ประจำภาควิชาครุศาสตร์วิศวกรรม ทุกๆ ท่านที่ได้กรุณาให้ข้อเสนอแนะต่างๆ แนวทางการแก้ปัญหาในด้านต่างๆ และเพื่อนๆ ที่คอยให้กำลังใจตลอดจนกระทั่งปริญญานิพนธ์ฉบับนี้ สามารถสำเร็จลุล่วงไปได้ด้วยดี จึงขอขอบคุณมา ณ โอกาสนี้ด้วย

# สารบัญ

# สารบัญรูปภาพ

# สารบัญตาราง

# บทที่ 1
# บทนำ

ในปัจจุบันนี้เทคโนโลยีใหม่ๆ ได้เข้ามามีบทบาทมากในชีวิตประจำวัน จากที่เคยใช้เครื่องเล่นวีดีโอในการร้องเพลงประกอบดนตรีที่ทำขึ้นหรือเรียกว่า คาราโอเกะ นั้น และในตอนนี้คอมพิวเตอร์ได้มีการพัฒนาแนวคิดเกี่ยวกับการใช้โปรแกรมคอมพิวเตอร์ สำหรับคอมพิวเตอร์ที่มีการ์ดเสียงอยู่แล้วให้สามารถทำงานคล้ายกับเครื่องเล่นคาราโอเกะในอดีตได้

ดังนั้นในปริญญานิพนธ์ฉบับนี้ ก็ได้นำวิวัฒนาการนี้มาใช้โดยการนำหลักการเกี่ยวกับการให้จังหวะดนตรีของคอมพิวเตอร์มาประยุกต์ใช้เพื่อความสะดวกแก่ผู้ที่มีคอมพิวเตอร์และการ์ดเสียงอยู่แล้ว โดยเนื้อหาในปริญญานิพนธ์นี้จะอธิบายเนื้อหาแต่ละบทอย่างละเอียด พร้อมทั้งเน้นตัวอักษรให้เข้ม ตรงหัวข้อที่สำคัญ เพื่อจะได้มองเห็นได้ง่ายและจดจำไปใช้งานได้ง่ายขึ้น นอกจากนั้นยังเน้นเนื้อหาที่สำคัญไว้ใน หมายเหตุ และ คำเตือน เป็นต้น ซึ่งรายละเอียดอย่างกว้างในปริญญานิพนธ์นี้ประกอบด้วย

**บทที่ 1 บทนำ**

กล่าวถึงบทนำหรือที่มาของการนำเอาคอมพิวเตอร์มาทำเป็นคาราโอเกะ สิ่งสำคัญที่ทำให้คอมพิวเตอร์เกิดเสียงเพลงขึ้นและสามารถนำมาใช้เป็นโปรแกรมให้จังหวะดนตรีได้ สำหรับผู้ที่มีการ์ดเสียงอยู่แล้วก็สามารถนำโปรแกรมต่างๆ ที่ทำงานด้านเสียงมาประมวลผลได้ ดังนั้นจึงได้เล็งเห็นความสำคัญ และนำการ์ดเสียงมาประยุกต์ใช้งาน เพื่อความบันเทิงในระบบมัลติมิเดียได้ ซึ่งการทำงานทางด้านเสียง จะทำให้เกิดการเรียนรู้ได้ง่าย โดยส่วนประกอบที่จำเป็นมีเพียงการ์ดเสียงและลำโพงเท่านั้น

**บทที่ 2 ทฤษฎีและหลักการ**

กล่าวถึงทฤษฎีและหลักการวิเคราะห์โปรแกรม ส่วนที่ทำให้คอมพิวเตอร์เสียงเพลงขึ้นได้ ประกอบด้วยชิป 2 ตัวคือ ชิป DSP (Digital Sound Processor Chip) หมายเลข CT 1741 ที่ใช้ในซาวด์บลาสเตอร์ 16 (Sound Blaster 16 : SB16) ซึ่งทำหน้าที่เป็นตัวกลางการควบคุม

SB16 เพื่อให้เข้ากันได้กับการ์ดซาวด์บลาสเตอร์ ชิป DSP จะเปลี่ยนคำสั่งต่างๆ ที่ส่งไปยัง SB16 และใช้ความสามารถที่มีอยู่บนการ์ดซาวด์บลาสเตอร์ เช่น การแปลสัญญาณต่างๆ ที่เป็นฟังก์ชั่น สำหรับมิดี้ อินพุต และ เอาท์พุต ส่วนชิปยามาฮ่า OPL 3 FM ซินทริไซเซอร์ (FM Synthesizer Chip) จะถูกใช้บนการ์ดซาวด์บลาสเตอร์ โดยเฉพาะรุ่น SB16 ที่นิยมใช้กันอย่างแพร่หลาย ภายในชิปนี้จะมีตัวผลิตสัญญาณ 4 ตัว และมีโหมดการทำงานอยู่ 2 โหมด ในโหมดแรกสามารถสร้างเมโลดี้ได้ 15 ตัวโน้ตและมีเสียงดนตรีที่เป็นเครื่องดีหรือเคาะได้ 5 เสียง ซึ่งชิปทั้งสองตัวนี้เป็นส่วนประกอบสำคัญในการ์ดซาวด์บลาสเตอร์

รูปแบบคลื่นเสียง ซึ่งรายละเอียดจะอ้างถึงรูปแบบมัลติมิเดีย , รูปแบบไฟล์ RIFF ที่เป็นพื้นฐานการสร้างบล็อกของไฟล์ RIFF ,คำนิยามรูปแบบ WAV (Waveform Audio File) , รูปแบบ WAVE Chunk การสร้างรูปแบบคำสั่งของ Format chunk และ ข้อมูล WAV chunk

รูปแบบไฟล์ MIDI (Musical Instrument Interface)ในรายละเอียดจะกล่าวถึงมาตรฐานไฟล์ MIDI รุ่น 1.0 , MIDI Header Chunk , MIDI Track chunk , รูปแบบไฟล์ MIDI , โครงสร้างของไฟล์ MIDI , มาตรฐาน IFF ที่มีโครงสร้างของรูปแบบ SMF (Standard MIDI File) แบบต่างๆ , Header chunk ที่ประกอบด้วย chunk ID 4 ไบท์ ค่าความยาว 4 ไบท์ และ Track chunk

Borland C++ สำหรับวินโดวส์ 3.0 ขึ้นไปมีประสิทธิภาพสูงมากในการใช้งาน คอมไพเลอร์จะช่วยอำนวยความสะดวกให้ผู้ศึกษาภาษา C เป็นอันมาก Borland ได้จัดทำ IDE (Integrated Development Environment) รวมกับคุณสมบัติอื่นๆ ที่ใช้ในการพัฒนาโปรแกรม C++ ได้แก่ เอดิเตอร์ (editor) , คอมไพเลอร์ (compiler) , ลิงเคอร์ (linker) , ดีบักเกอร์ (debugger) และระบบให้ความช่วยเหลือ (help system) ทั้งหมดนี้รวมอยู่ในที่เดียวกัน ทำให้สะดวกในการใช้งานบนจอภาพ ดีกว่าคอมไพเลอร์รุ่นเก่า และ Borland C++ จะช่วยให้เรียนรู้การเขียนโปรแกรมแบบ OOP (Object-Orientde Program) เป็นไปอย่างรวดเร็ว

หลายคนเข้าใจว่า C++ เป็นส่วนขยายเพิ่มเติมของ C แต่จริงๆ แล้ว C++ เป็นมากกว่านั้น C++ เป็นภาษาที่มีความสมบูรณ์ในตัวของมันเอง C++ ได้จากการปรับปรุง C ให้มีประสิทธิภาพในการใช้งานสูงขึ้น ภาษา C ถูกสร้างขึ้นในปี พ.ศ. 1980 ต่อมาในปี พ.ศ. 1990 จึงได้พัฒนา C++ ใช้งานด้านต่างๆ หลายคนเปลี่ยนจากภาษา C หรือภาษาอื่นๆ ไปใช้ ภาษา C++ เนื่องจากคุณลักษณะ Object-Oriented ของ C++ OOP หรือ Object-Oriented Programing

ได้พัฒนาจากโปแกรมรูปแบบเดิมที่ใช้โปรซิเยอร์ (procedure) หรือโปรแกรมย่อยในการทำงาน OOP จะช่วยให้โปรแกรมทำงานมีประสิทธิภาพสูงขึ้น โดยให้ข้อมูลและฟังก์ชั่นอยู่รวมกันและทำงานบนตัวแปรออปเจ็กต์ของคลาสแทนที่ข้อมูลจะรอคอยการทำงาน ส่วนโพลิมอร์ฟีซึม (Polymorphism) จะช่วยให้ทำงานได้หลายงานด้วยคำสั่งเพียงคำสั่งเดียว ซึ่งจะต้องพึ่งพาอาศัยคุณลักษณะของเวอร์ชวล การใช้ฟังก์ชั่นของ C++ มีความคล่องตัวกว่าภาษาอื่นๆ สามารถใช้ชื่อเหมือนกันได้ ช่วงประมวลผลโปรแกรมจะแยกแยะการทำงานของฟังก์ชั่นให้โดยอัตโนมัติ ภายใน C++ จะได้เห็นการทำงานของ พอยน์เตอร์ (pointer) ที่มีพลังงานอำนาจในการชี้แหล่งข้อมูล ตลอดจนการใช้เทมเพลท (template) วางโมเดล (model) หรือแบบจำลองเพื่อใช้เป็นแบบในการสร้างโปรแกรมต่างๆ ไม่ให้ออกนอกลู่นอกทาง ทำให้โปรแกรมสั้นลงมาก แต่ทำงานได้เร็วขึ้น ส่วนคอนเทนเนอร์ (container) ช่วยอำนวยความสะดวก ในการจัดการข้อมูลบนสแตก (stack) โดยอาศัยความช่วยเหลือจากฟังก์ชั่น และคลาสเทมเพลททั้งเทมเพลท และคอนเทนเนอร์ เป็นคุณลักษณะล่าสุดบน Borland C++ ver 3.1 ขึ้นไป และในการเขียนโปรแกรมประยุกต์บนวินโดวส์ด้วย Borland C++ ในบทนี้ซึ่งมีวิธีการออกแบบ 2 วิธี คือ การออกแบบทั่วไป และการเขียนโค้ด ซึ่งจะกล่าวโดยละเอียดพร้อมรูปประกอบตาม ขั้นตอนต่างๆ ที่ผู้ใช้สามารถทำตามได้อย่างถูกต้องโดยดูตามรูป

**บทที่ 3 การออกแบบ**

การออกแบบโปรแกรมให้จังหวะดนตรีและคำร้องนี้กล่าวถึงรูปแบบของแอพพลิเคชั่น ซึ่งจุดประสงค์หลักของ Windows คือให้สามารถติดต่อกับผู้ใช้ได้ง่ายกว่าเดิมซึ่งแอพพลิเคชั่นจะติดต่อกับผู้ใช้โดยผ่านระบบหรือเครื่องมือใหม่ที่ประกอบด้วย

- วินโดวส์
- เมนู
- กรอบข้อความ (dialog box)
- เมสเสจ (message)

ซึ่งในส่วนวินโดวส์นี้จะกล่าวถึงส่วนประกอบต่างๆ ของวินโดวส์ ดังนี้ คอนโทรลเมนู ,คอนโทรลเมนูบ็อกซ์ , ไตเติ้ลบาร์ , เส้นเมนู , ปุ่ม minimize , ปุ่ม maximize , สโครลล์บ็อกซ์ , สโครลล์บาร์ และ กรอบวินโดวส์ โดยจะมีรูปประกอบด้วย

ส่วนเมนูจะกล่าวถึงหน้าที่ของเมนูที่ใช้รับข้อมูลหลักของแอพพลิเคชั่นที่ทำงานบนวินโดวส์

ภายในกรอบข้อความประกอบด้วย คอนโทรลแต่ละชนิดที่จะใช้ติดต่อกับผู้ใช้แบบง่ายๆ

เมสเสจ ซึ่งมีส่วนประกอบหลักของแอพพลิเคชั่น ที่มีชื่อเรียกว่า เมสเสจลูป (Message Loop) มีหน้าที่รับเมสเสจและแจกจ่ายไปยังวินโดวส์ที่เหมาะสม , รูปแสดงการจัดการเข้าทางคีย์บอร์ดของเมสเสจ ,การจัดการข้อมูลสำหรับแอพพลิเคชั่นทั้งสองตัว และการจัดการเมสเสจของวินโดวส์

การทำงานภายในโปรแกรมคาราโอเกะบนวินโดวส์จะแสดงภาพโดยรวมของไฟล์ KARAOKE โดยสมมติฟังก์ชั่น TSoundApp ขึ้นเพื่อให้ง่ายในการเข้าใจ แสดงภาพการสืบทอดคลาส Object Windows จะอธิบายส่วนประกอบที่สำคัญภายในภาพ และอัลกอริทึมที่แสดงออกมาเป็นรูปการตอบสนองฟังก์ชั่นของ TSoundWindow เพื่อสมมติให้สามารถดูการทำงานได้ง่ายขึ้น

**บทที่ 4 การทดลองและผลการทดลอง**

เป็นการทดลองและผลการทดลองของโปรแกรม มีขั้นตอนการใช้งานของโปรแกรมให้จังหวะดนตรีและคำร้องโดยละเอียด พร้อมภาพประกอบทุกขั้นตอนทำให้ ผู้ที่ต้องการศึกษาโปรแกรมให้จังหวะดนตรีและคำร้อง เข้าใจการใช้งานของโปรแกรมได้ง่ายขึ้น และหาจุดบกพร่องของโปรแกรม เพื่อนำไปปรับปรุง แก้ไขโปรแกรมให้ดีขึ้นต่อไป

**บทที่ 5 บทวิจารณ์และสรุป**

ส่วนบทสุดท้ายนี้ คือ บทวิจารณ์และการสรุปผลปริญญานิพนธ์ ซึ่งประกอบด้วยบทสรุปของโปรแกรมให้จังหวะดนตรีและคำร้อง ปัญหาต่างๆ ที่ประสบขณะทำโครงงานโดยปัญหาที่กล่าวถึงนั้นแบ่งออกเป็นหัวข้อ 3 หัวข้อดังนี้

- แหล่งข้อมูล
- เครื่องมือทดสอบ , ทฤษฎี และกระบวนการ
- เครื่องมือที่ใช้ในการจัดทำโครงงาน

ข้อเสนอแนะและแนวทางการพัฒนาโปรแกรมให้จังหวะดนตรีและคำร้องนี้ กล่าวถึงความเป็นไปได้ทางด้านทฤษฎี และปฏิบัติในการนำไปพัฒนาให้มีประสิทธิภาพมากขึ้นนั้น จะมีเนื้อหาเกี่ยวกับการเลือกใช้คอมพิวเตอร์ให้เหมาะสมกับโครงงาน , การจัดการโปรแกรม, การเลือกใช้โปรแกรมให้เหมาะสม , ข้อควรปรับปรุง , สิ่งที่ควรหลีกเลี่ยง , ความรู้เกี่ยวกับการหลีกเลี่ยงการเกิด Dead Lock จากเมสเสจ และ การติดต่อกับคอนโซล ซึ่งประกอบด้วยคีย์บอร์ด และจอภาพ ซึ่งเป็นทรัพยากรที่จะต้องใช้ร่วมกัน จึงจำเป็นต้องแบ่งส่วนการทำงานกันอย่างเหมาะสม โดยไม่ทำให้เกิดปัญหาต่างๆ ตามมา

# บทที่ 2
# ทฤษฎีและหลักการ

## 2.1 ซาวด์บลาสเตอร์ 16 / SCSC-2 / MCD

ลักษณะทั่วไปของ ซาวด์บลาสเตอร์ 16 ( Sound Blaster 16 : SB 16) แฟมิลี่ Advance Signal Processor เป็นสิ่งใหม่ที่สามารถหาได้ง่ายบน SB 16 ดังนั้นจึงต้องนำมาพิจารณา อันดับแรกการประยุกต์ใช้ที่เจริญก้าวหน้า Wave Blaster (WB) เป็นการ์ดที่หาได้ ขึ้นอยู่กับ SB 16

**หมายเหตุ** Creative ใช้ชื่อย่อว่า ASP ซึ่งมีชื่อเหมือนกับ ASP Computer Product Inc. เป็นสาเหตุให้ชิปที่กล่าวถึงนั้นเหมือนกับ Advance Signal Processor หรือเรียกสั้นๆ ว่า CSP (Craetive Signal Processor) ดังนั้นการ์ด SB 16s ที่รวม Advance Signal Processor จะถือเป็นอันเดียวกับ "Advance Signal Processing" และใช้ชื่อย่อว่า CSP

ตระกูล ซาวด์บลาสเตอร์16 ทุกรุ่น จะเรียกเป็น SB 16 เพราะลักษณะรูปแบบจะมีตำแหน่งตรงกัน ตัวอย่าง ซาวด์บลาสเตอร์16 กับ Advance Signal Processing จะมี SB 16 CSP

### 2.1.1 โครงสร้างของซาวด์บลาสเตอร์ 16

ตระกูล SB 16 เป็นสิ่งใหม่ล่าสุดและดีที่สุดจาก Creative Labs. ตระกูลนี้จะประกอบด้วยรุ่นต่างๆ ดังนี้ : SB 16 Basic Edition (Basic), SB 16 SCSI-2 (Small Computer Systems interfece) และ SB 16 Multi CD (MCD) ทั้ง SB 16 SCSI-2 และ SB 16 MCD มีประสิทธิภาพพอๆ กับ CSP ชิปติดตั้ง (install) และมีเครื่องหมายเหมือนกัน ตัวอย่าง SB 16 SCSI-2 CSP ดังตาราง 2.1 แสดงลักษณะของ SB 16

ข้อแตกต่างระหว่าง SB16s (ไม่มี CSP) หมายถึงใน CD-ROM อินเทอร์เฟส SB16 ทั่วไปจะมีมาตรฐาน เฉพาะแต่เพียงผู้เดียว พานาโซนิค CD-ROM อินเทอร์เฟสที่สนับสนุนบนซาวด์บลาสเตอร์ โปร ซีรี่ SB16 SCSI บางตัวออกแบบเหมือนฮาร์ดไดร์ฟ SCSI อินเทอร์เฟสใช้ Adaptec Chipset เดียว ซึ่งจะเหมือนกันกับที่พบบน Adaptec AHA-1510/1520/1522 SCSI ที่ทำใหม่จำนวนมาก SB16 MCD บนบางตัวไม่เหมาะสม รวมทั้งพานาโซนิค CD-ROM อินเทอร์เฟส (เหมือนกับ SB16 ทั่วไป) แต่การอินเทอร์เฟส สำหรับเฉพาะผู้เดียว (proprietary) มิตซูมิ CD-ROM ไดร์ฟ ดีพอ ๆกับโซนี่ CD-ROM ไดร์ฟ

| | Sound Blaster 16 Basic Edition | Sound Blaster 16 MultiCD | Sound Blaster 16 SCSI-2 |
|---|---|---|---|
| รูปแบบ | 16 บิท สเตอริโอ | 16 บิท สเตอริโอ | 16 บิท สเตอริโอ |
| Advanced signal processing upgradability | Yes | Yes | Yes |
| Wave Blaster upgradability | Yes | Yes | Yes |
| CD-ROM ที่เข้ากันได้ | Creative Labs, พานาโซนิค | โซนี่ , มิตซูมิ , Creative Labs, พานาโซนิค | SCSI หรือ SCSI-2 |
| รวม ไมโครโฟน | ไม่รวม | รวม | รวม |
| รวมซอร์ฟแวร์ | Monologue สำหรับวินโดวส์<br>Creative WaveStudio<br>Creative Soundo' LE<br>Creative Talking scheduler<br>Creative Mosaic | Monologue สำหรับวินโดวส์<br>Creative WaveStudio<br>Creative Soundo' LE<br>Creative Talking scheduler<br>Creative Mosaic<br>Creative VoiceAssist<br>PC Animate Plus | Monologue สำหรับวินโดวส์<br>Creative WaveStudio<br>Creative Soundo' LE<br>Creative Talking scheduler<br>Creative Mosaic<br>Creative VoiceAssist<br>PC Animate Plus |

**ตาราง 2.1 ลักษณะที่ต่างกันของ SB 16**

จากผลิตภัณฑ์ที่ พัฒนาจน SB16 จะเข้ากันได้กับทั้ง ซาวด์บลาสเตอร์ และ ซาวด์บลาสเตอร์ โปร และร่วมกับตัวอื่น ๆ MIDI / joystick พอร์ทและ CD-ROM อินเทอร์เฟส จะไม่มีการเปลี่ยนแปลงซอร์ฟแวร์ทุกตัวจะสนับสนุน ซาวด์บลาสเตอร์ MIDI พอร์ทจะทำงานได้เท่ากันกับ SB16 และ ซาวด์บลาสเตอร์ Pro CD-ROM ไดร์ฟ และ โปรแกรมควบคุม

(driver) จะทำการหาบน SB16 Basic SB16 SCSI-2 และ SB16 MCD จะกำหนดลักษณะ CD-ROM อินเทอร์เฟส สำหรับตัวมันเอง

แต่เดิมพอร์ท joystick ของ ซาวด์บลาสเตอร์ จะสนับสนุน SB16 ดังนั้นจึงอาจจะพบกับปัญหา เกี่ยวกับความเร็ว 386 หรือ 486 PCs ถ้าสามารถหาซื้อเฉพาะการ์ด joystick อินเทอร์เฟส ที่เหมาะสมกับความเร็วของคอมพิวเตอร์ได้ จากภายนอกของ PC ปริมาณการควบคุมการเคลื่อนไหว เจ็คอินพุต และ เจ็คเอาท์พุต ด้านหลังของบอร์ดอย่างเดียวกันกับที่สนับสนุน ซาวด์บลาสเตอร์ อื่น ๆ

### 2.1.2 ชิปสำหรับซาวด์บลาสเตอร์ 16

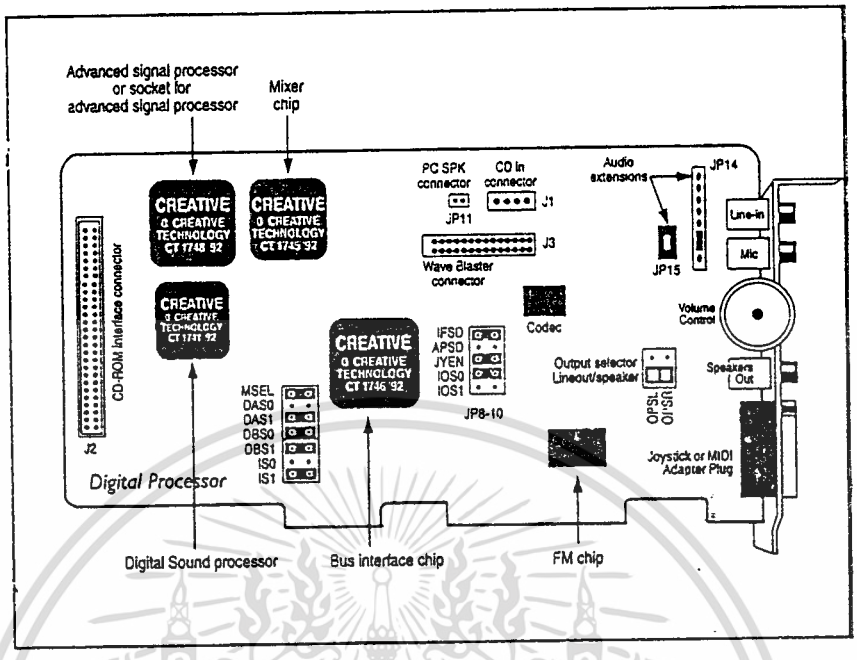
มันเป็นชิปใหม่ที่ใช้บน ซาวด์บลาสเตอร์16 พื้นฐานการเลือกจะไม่มีการเปลี่ยนแปลง และชิปใหม่จะให้ลักษณะพิเศษ และ ประสิทธิภาพที่ดีกว่าชิปที่ใช้บน SB16 รวมทั้ง Digital Sound Processor (DSP), FM ซินติไซเซอร์, มิกซ์เซอร์, การ์ดอินเทอร์เฟส, Advanced Signal Processor และบนรุ่น SCSI-2 , Adapter SCSI ตัวควบคุมชิป

### 2.1.3 ชิป Digital Sound Processor ( DSP )

ซาวด์บลาสเตอร์16 ชิป DSP หมายเลข CT 1741 ในรูป 2.1 เป็นตัวกลางควบคุมของ SB16 เพื่อให้เข้ากันได้กับการ์ดซาวด์บลาสเตอร์ ชิป DSP จะเปลี่ยนคำสั่งต่างๆ ที่เป็นฟังก์ชั่นสำหรับแปลทุก MIDI อินพุตและเอาท์พุต เพื่อควบคุมชิปเสียงทุกตัวบน การ์ด

### 2.1.4 ชิป เอฟ เอ็ม ซินติไซเซอร์ (The FM Syntherizer Chip)

ชิปยามาฮ่า OPL 3 เอฟ เอ็ม ซินติไซเซอร์ จะใช้บน SB16 อันดับแรกจะใช้ใน ซาวด์บลาสเตอร์ โปร 2 ชิปนี้จะมีตัวผลิตสัญญาณ 4 ตัว, 20 เสียงระบบเอฟ เอ็ม สเตอริโอ, การแก้ไขบนตัวที่เก่ากว่าที่มีตัวผลิตสัญญาณ 2 ตัว , 11 เสียงโมโน ยามาฮ่า OPL 2 (3812) ชิป เอฟ เอ็ม จะใช้บน ซาวด์บลาสเตอร์ OPL 3 จะมีโหมดการทำงานอยู่ 2 โหมด คือ โหมด 2 และโหมด 4 ซึ่งสามารถสร้าง เมโลดี้ ได้ 15 ตัวโน้ต และเสียงเครื่องดนตรีที่ตีหรือเคาะได้ 5 เสียง ในตัวกระทำโหมด 2 และเมโลดี้ ได้เพียง 6 ตัวโน้ตและเสียงเครื่องดนตรีที่ตีหรือเคาะ 5 เสียงในตัวกระทำโหมด 4




รูป 2.1 ซาวด์บลาสเตอร์ 16 (CT 1740) การเชื่อมต่อกันและจุดต่อระหว่างตำแหน่ง

### 2.1.5 ชิปมิกซ์เซอร์

ชิปใหม่ที่นำมาใช้บน SB16 การ์ด ถึง CT 1745 มิกซ์เซอร์ ชิปนี้จะยอมให้ปรับปรุงและมิกซ์เสียงจากไมโครโฟน , จุดรับสัญญาณ ( line-in ) , สัญญาณเสียงจาก CD , สัญญาณเสียงระบบดิจิตอล , MIDI (FM หรือ Wave Blaster) และลำโพง PC ทางนี้จะทำให้สามารถได้ยินส่วนผสมของเสียง เช่น เสียงพูดจากไมโครโฟน ขณะเดียวกันกับเพลง FM , เสียงระบบดิจิตอล และสัญญาณเสียงจาก CD ที่เล่นในแบล็กกราวด์ ชิปมิกซ์เซอร์ใหม่นี้สามารถทำการบันทึกเสียงจากต้นฉบับได้หลายทางพร้อมกัน ซาวด์บลาสเตอร์ โปร ในการเปรียบเทียบการบันทึกเสียงจากต้นฉบับเดียวเท่านั้น

ชิปมิกซ์เซอร์จะสามารถควบคุมทำนองเสียงทุ้ม และ เสียงแหลม ซึ่งสามารถตัดเสียงเป็นอย่างที่เราต้องการได้ ถ้าเราใช้ลำโพงเล็กบน SB16 พยายามเพิ่มเสียงทุ้มให้มากขึ้นจะทำให้ต่างจากเกมส์ ถึงแม้ไม่สามารถทดแทนลำโพงที่มีกำลังดีๆ ได้ ถ้ามีเสียงรบกวนจากเกมส์ซึ่งใช้เสียง 8 บิท ให้ลดเสียงแหลมลง

ลักษณะพิเศษของชิป 1745 มิกซ์เซอร์เป็นการเพิ่มการควบคุมระดับลำโพง PCได้ บนซาวด์บลาสเตอร์ โปร มันไม่มีทางปรับปรุงระดับเสียงบี๊ปของลำโพง PC ให้สัมพันธ์กับเอาท์พุตอื่น แต่บน SB16 เราสามารถควบคุมเสียงบี๊ปบน PC ได้เต็มที่ผ่านลำโพงของเราเอง

หมายเหตุ ลำโพง PC จะมีเสียงบี๊ปผ่าน SB16 เท่านั้น ถ้ามีการเชื่อมต่อระหว่าง PC motherboard และ PC-SPK การติดต่อบน SB16

ครั้งก่อนมิกซ์เซอร์ใหม่จะยอมให้ปรับปรุงอินพุตและเอาท์พุตที่ได้มา ถ้ามีปัญหากับระดับเสียงอินพุตจากการ์ดต่ำ ก็สามารถเพิ่มระดับโดยเพิ่มเป็น 8 จังหวะ

มิกซ์เซอร์ที่ให้มา กับการ์ดควบคุมระดับเสียงนั้น สามารถควบคุมเสียงด้วยมือบนซาวด์บลาสเตอร์16 เป็นการเซทเสียงของเอาท์พุตด้วยการควบคุมเสียงได้เอง

### 2.1.6 บัสอินเทอร์เฟส

ชิป CT 1746 บัสอินเทอร์เฟส แสดงในรูป 2.1 ที่ผ่านมาระหว่าง ซาวด์บลาสเตอร์ 16 และ คอมพิวเตอร์ ข้อมูลทั้งหมดทั้งคำสั่ง และ สัญญาณเสียงระบบดิจิตอล (digital audio) อาจจะผ่านชิปบัสอินเทอร์เฟสไปติดต่อกับ PC motherboard ได้ ชิปบัสอินเทอร์เฟสจะกำหนดการเซทรูปร่างภายนอก สำหรับการ์ดที่กล่าวถึงตำแหน่งอินพุต / เอาท์พุตพอร์ท อินเทอร์รัพท์ และการกำหนดช่อง DMA ในที่สุด CT 1746 ที่มีบัฟเฟอร์มาให้ที่มีอัตราการสุ่มสูง สามารถใช้กับคอมพิวเตอร์ที่ช้ากว่า 386 ได้

### 2.1.7 โคเดค (The CODEC)

CT 1701 โคเดคจะทำการเปลี่ยน อนาล็อก เป็น ดิจิตอล เมื่อบันทึกสัญญาณเสียงระบบดิจิตอล และเปลี่ยน ดิจิตอลเป็นอนาล็อก เพื่อเล่นกลับเป็นสัญญาณเสียงระบบดิจิตอล มันสามารถสุ่มสัญญาณเสียงได้ทั้ง 8 บิท หรือ 16 บิท ที่สูงกว่า 44.1 KHz ในระบบสเตอริโอ มันจะเปรียบได้กับคุณภาพเสียง ที่ใช้การได้บน CD และ เทปสัญญาณเสียงระบบดิจิตอล ( digital audio tape : DAT) โคเดคก็จะทำการกรองและตัดสัญญาณรบกวนทิ้งได้

### 2.1.8 การประมวลผลสัญญาณเสียงชั้นสูง (The Advanced Signal Processor)

การประมวลผลสัญญาณเสียงชั้นสูง CT 1748 เป็นการสั่งการ Digital Signal Processor อย่าเข้าใจผิดว่าเป็น Digital Sound Processor การประมวลผลสัญญาณเสียงชั้นสูงจะทำการกำหนดข้อมูลนั้นเข้าส่วนสัญญาณเสียงระบบดิจิตอลของคำสั่งเก็บข้อมูลใน หน่วยความจำเรียกว่า รหัส CPS ( Creative Signal Processor)

ชิปนี้สามารถโปรแกรมเป็นจำนวนของฟังก์ชั่นรวมทั้งเพิ่มกลวิธีพิเศษ(special effect) เป็นสัญญาณเสียงระบบดิจิตอล และกำหนด advance compression และ decompression ของสัญญาณเสียง (audio) ในการดำเนินการประมวลผลสัญญาณเสียงชั้นสูง จะสัมพันธ์กับการจัดการซอร์ฟแวร์เกี่ยวเนื่องกับหัวข้อ " เทคโนโลยีการประมวลผลสัญญาณเสียงชั้นสูง "

### 2.1.9 การเชื่อมต่อของซาวด์บลาสเตอร์ 16

รูป 2.1 , 2.2 และ 2.3 จุดออกจะเปลี่ยนการเชื่อมต่อบนการ์ด SB16 การเชื่อมต่อให้พิจารณาส่วนที่จะใช้กับการ์ด ซาวด์บลาสเตอร์ 16 ทุกรุ่น ยกเว้นการเชื่อมต่อลักษณะของ SB16 SCSI-2 และ SB16 MCD ดู " การเชื่อมต่อสำหรับ ซาวด์บลาสเตอร์ 16 SCSI-2 " และ " การเชื่อมต่อสำหรับซาวด์บลาสเตอร์ 16 MCD " ได้จากหัวข้อต่อไป

การเชื่อมต่อจะใช้สำหรับผ่านเสียงจากการ์ดไปยังลำโพง , สเตอริโอ , หูฟังและสำหรับเสียงจากไมโครโฟน , เครื่องเล่นเทป หรือ สเตอริโอ

การเชื่อมต่อของคอมพิวเตอร์จากภายนอก

- แจ็คจุดรับสัญญาณจะแสดงที่ส่วนบนสุดของการ์ดรูป 2.1 จะเป็นแจ็คสเตอร์โอขนาดเล็ก และจะใช้เกี่ยวไว้ไน จุดส่งสัญญาณ (line-out) จากเทป deck , สเตอริโอ หรือเครื่องเล่น CD
- จุดเชื่อมต่อ ๆไป บนการ์ด คือ จุดเชื่อมต่อไมโครโฟน เป็นแจ็คโมโนขนาดเล็ก
- จุดเชื่อมต่อที่อยู่ด้านใต้ปุ่ม volume เป็น เอาท์พุตเครื่องขยายเสียงจะเป็นแจ็คสเตอริโอขนาดเล็ก จุดต่อด้านนอกลำโพงจะมีมาให้ในเครื่องขยายเสียงที่สามารถเพิ่มขึ้นได้ถึง 4 วัตต์ ของกำลังขยายต่อข้าง ให้แน่ใจว่าปรับระดับความดังลดลงแล้วก่อนที่จะทำการเชื่อมต่อทุกอย่าง ดังนั้นอย่างตั้งปลั๊กโมโนขนาดเล็กกับเอาท์พุตลำโพง จะทำให้วงจรช็อตและเครื่องขยายเสียงอาจเสียได้
- จุดต่อ 15-pin D-sub จะใช้สำหรับอินพุต joystick และ MIDI อินพุต/เอาท์พุต พอร์ท joystrick / MIDI จะใช้ joystickได้ 1 หรือ 2 ตัว ต้องการใช้ 2 ตัวให้ใช้ Y-adapter จาก Creative Labs; Y-adapter ทั่วไป จะใช้งานไม่ได้ จุดต่อ 2 pin บน joystrick / MIDI (pin 12 และ 15) ใช้สำหรับ MIDI Out และ MIDI In ตามลำดับ เราจะต่อ MIDI คีย์บอร์ด และ ซินติไซเซอร์ กับ ซาวด์บลาสเตอร์ ด้วยสายเคเบิ้ล Creative MIDI

คือจุดต่อ 3 จุดบนการ์ดเท่านั้น ที่จะสามารถติดต่อกับภายในของคอมพิวเตอร์ได้ PC-SPK, J1 และ J2 ดังแสดงในรูป 2.1

- จุดต่อ PC-SPK จะยอมให้ต่อกับเอาท์พุตลำโพง motherboard กับ SB 16 การกำหนด pin สำหรับการต่อนั้นมีให้ในตาราง 2.2

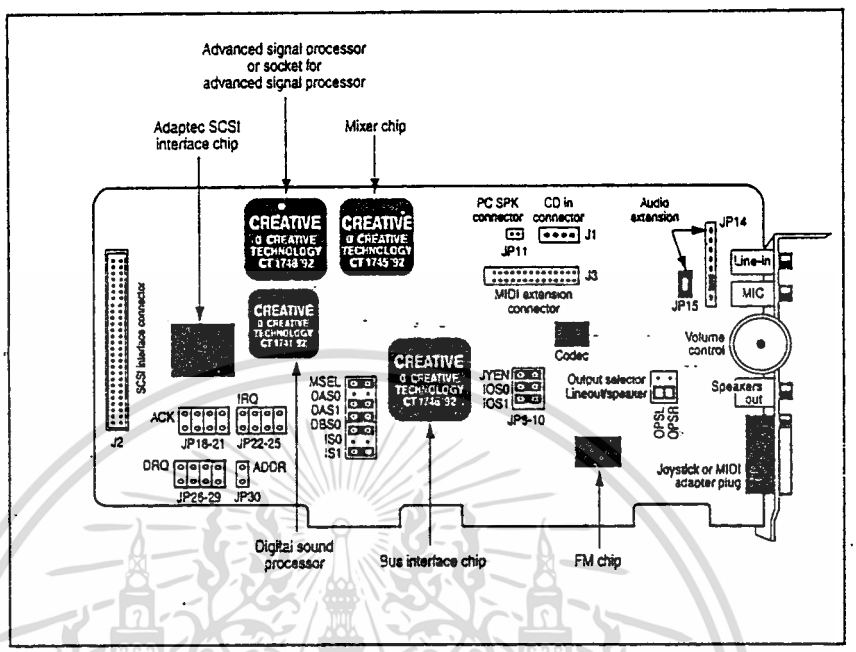
**คำเตือน** ถ้าเราต่อ PC-SPK กับ motherboard ให้เครื่องปรับระดับความดังของเสียงจากคอมพิวเตอร์อินพุตบน PC-SPK สามารถปรับแต่งได้จาก SB 16 มิกซ์เซอร์

จุดต่อ J2 เป็นจุดต่อสายสัญญาณ CD-ROM ยกเว้นบน SB16 MCD สายสัญญาณจะทำงานระหว่าง J2 และ CD-ROM ไดร์ฟ มันจะเป็นจุดต่อ อินเทอร์เฟส สำหรับ Creative Labs CD-ROM ไดร์ฟ และ พานาโซนิค/Matsushita ไดร์ฟ เท่านั้น การติดต่อ Creative Labs

- เพื่อให้รูปแบบถูกต้อง J2 จะไม่สามารถใช้กับ CD-ROM ไดร์ฟ ใด ๆ และมันจะไม่เข้ากับ SCSI หรือ subset แบบ SCSI อินเทอร์เฟส

จุดต่อ J3 ตำแหน่งจะอยู่ด้านใต้ J1 สำหรับ Wave Blaster daughterboard upgrade ยกเว้นบน SB 16 MCD จะอยู่บนเครื่องหมาย J2 เมื่อติดตั้ง Wave Blaster ให้ดูด้านบนของ pin Wave Blaster จะไม่เล่นหากวางแนวเชื่อมต่อผิดพลาด ตาราง 2.4 สรุปการกำหนด pin ของจุดต่อ Wave Blaster

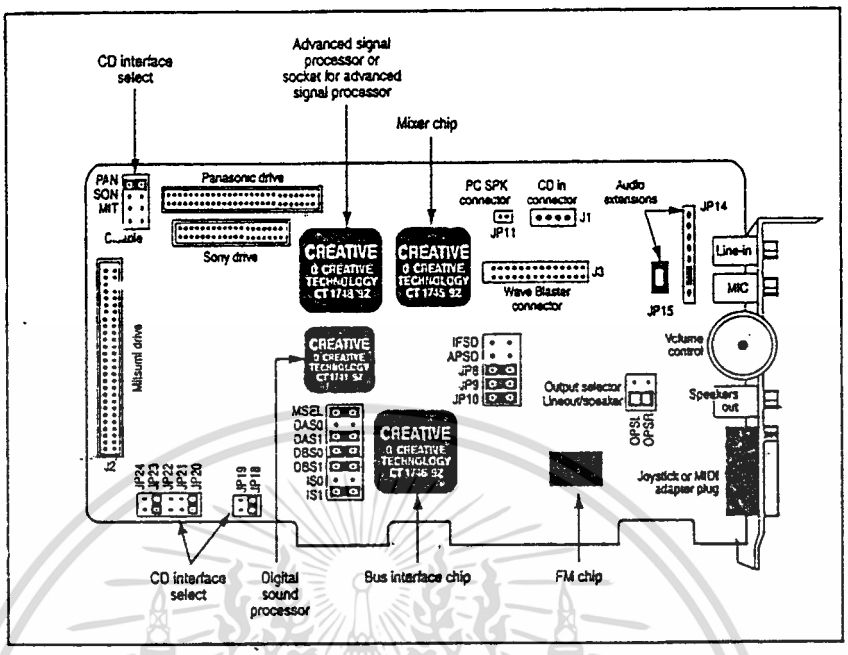
**คำเตือน** การกำหนด pin ของจุดต่อ Wave Blaster เป็นการจัดเสนอสำหรับบอกประโยชน์การใช้เท่านั้น การใช้ pin ผิดพลาดจะทำให้เกิดความเสียหายต่อ ซาวด์บลาสเตอร์16 Wave Blaster หรือทั้งสอง




**รูป 2.2 ตำแหน่งการเชื่อมต่อและจุดต่อของ ซาวด์บลาสเตอร์ 16 SCSI-2 CCT 1770**

| ขั้ว | สัญญาณ | สัญญาณ เข้า/ออก |
|---|---|---|
| 1 | +5 โวลท์ | สัญญาณเข้า |
| 2 | ลำโพง | สัญญาณเข้า |

**ตาราง 2.2 จุดต่อลำโพง PC (PC-RPK)**




**รูป 2.3 ตำแหน่งการเชื่อมต่อและจุดต่อของ ซาวด์บลาสเตอร์ 16 MCD CCT 1750**

### 2.1.10 จุดต่อสำหรับ ซาวด์บลาสเตอร์16 SCSI-2

ในส่วนที่เป็นจุดต่อที่มีลักษณะเป็นรุ่น SCSI ของ ซาวด์บลาสเตอร์16 ดังรูป 2.2 จุดต่ออื่น ๆจะเหมือนกับการ์ด ซาวด์บลาสเตอร์ ทั่วไป

- J2 เป็นสายสัญญาณ CD-ROM บน SB 16 SCSI สายสัญญาณจะทำงานระหว่าง J2 กับ CD-ROM ไดร์ฟ จะยอมให้ ซาวด์บลาสเตอร์ 16 ควบคุม CD-ROM ไดร์ฟ ได้โดยตรง CD-ROM อินเทอร์เฟส นี้ขึ้นอยู่กับมาตรฐาน SCSI-2 และเร็วกว่าการ์ดเสียงทั้งหมด ที่ใช้เพียง SCSI-1 SCSI อินเทอร์เฟส จะยอมให้เราติดต่อกับ SCSI อื่น เช่น ฮาร์ดดิสค์ , เทปแบคอัพ และ สแกนเนอร์

| ขั้ว | สัญญาณ | สัญญาณ เข้า/ออก |
|---|---|---|
| 1 | กราวด์ | สัญญาณเข้า |
| 2 | CD ข้างซ้าย | สัญญาณเข้า |
| 3 | กราวด์ | สัญญาณเข้า |
| 4 | CD ข้างขวา | สัญญาณเข้า |

ตาราง 2.3 จุดเชื่อมต่อ CD IN (J1)

**2.1.11 จุดต่อสำหรับซาวด์บลาสเตอร์ 16 MCD**

ในส่วนนี้เป็นจุดที่มีลักษณะเป็นรูป MCD ของ ซาวด์บลาสเตอร์ 16 ดังรูป 2.3 ส่วนจุดต่ออื่น ๆก็เหมือนกับการ์ดซาวด์บลาสเตอร์ ทั่วไป

- J3 เป็นสายสัญญาณ CD-ROM ต่อกับ CD-ROM ไดร์ฟ จะใช้การเชื่อมต่อกับพานาโซนิค เท่านั้น เหมือนจุดต่อที่ใช้บนซาวด์บลาสเตอร์ 16 ทั่วไปและ SB 16 Basic มันจะสนับสนุน พานาโซนิค CD 521 , CR523 และ CR563 ไดร์ฟ
- J4 เป็นสายสัญญาณ CD-ROM สำหรับ CD-ROM ไดร์ฟ ที่ใช้เชื่อมต่อกับโซนี่ โดยเฉพาะ มันจะสนับสนุน โซนี่ CDU 31A-02 ไดร์ฟ
- J5 เป็นสายสัญญาณ CD-ROM สำหรับ CD-ROM ไดร์ฟ ที่ใช้เชื่อมต่อกับมิทซูมิ เท่านั้น มันจะสนับสนุนมิทซูมิ CRMC LU0055 และ CRMC FX001/FX001 ไดร์ฟ

**2.1.12 การเชื่อมต่อจุดสำหรับซาวด์บลาสเตอร์ 16**

จากรูป 2.1 , 2.2 และ 2.3 การเชื่อมต่อจุดบนการ์ด SB16 โดยพิจารณาในส่วนนี้คือที่การ์ดซาวด์บลาสเตอร์16 ทั่วๆ ไปสำหรับการเชื่อมต่อจุดแบบ SB16 SCSI-2 และ SB16 MCD ให้ดูจาก " การเชื่อมต่อสำหรับซาวด์บลาสเตอร์16 SCSI-2 " และ " การเชื่อมต่อสำหรับซาวด์บลาสเตอร์ 16 MCD " ในหัวข้อต่อไป

การเชื่อมต่อจุดจะใช้กำหนดการ์ด SB 16 ดังนั้นมันจะไม่ขัดแย้งกับการ์ดอื่นในคอมพิวเตอร์การเชื่อมต่อจุดนี้จะเซทในการ์ดที่กำหนดเมื่อติดตั้งการ์ด

| ขั้ว | สัญญาณ | ขั้ว | สัญญาณ |
|---|---|---|---|
| 1 | ดิจิตอล กราวด์ | 2 | ไม่ต่อสัญญาณ |
| 3 | ไม่ต่อสัญญาณ | 4 | มิดี้ (MIDI) เอาท์พุต |
| 5 | ดิจิตอล กราวด์ | 6 | +5 โวล์ท |
| 7 | มิดี้ (MIDI) เอาท์พุต | 8 | ไม่ต่อสัญญาณ |
| 9 | ดิจิตอล กราวด์ | 10 | +5 โวล์ท |
| 11 | ดิจิตอล กราวด์ | 12 | ไม่ต่อสัญญาณ |
| 13 | ไม่ต่อสัญญาณ | 14 | +5 โวล์ท |
| 15 | อนาล็อก กราวด์ | 16 | ไม่ต่อสัญญาณ |
| 17 | อนาล็อก กราวด์ | 18 | +12 โวล์ท |
| 19 | อนาล็อก กราวด์ | 20 | สัญญาณเข้า:ช่องซ้าย |
| 21 | อนาล็อก กราวด์ | 22 | -12 โวล์ท |
| 23 | อนาล็อก กราวด์ | 24 | สัญญาณเข้า:ช่องขวา |
| 25 | อนาล็อก กราวด์ | 26 | รีเซท |

**ตาราง 2.4 จุดต่อ Wave Blaster (J3 บน SB16 และ SB 16 SCSI , J2 บน MCD)**

- การเชื่อมต่อจุด MSEL เลือกตำแหน่ง อินพุต / เอาท์พุต สำหรับ MPU-40 ที่ตรงกับพอร์ท MIDI เราสามารถเลือก 300H หรือ 330H

300H : ไม่มีการเชื่อมต่อจุดเมื่อติดตั้งบน MSEL

330H : มีการเชื่อมต่อจุดบน MSEL (ค่าที่กำหนดไว้)

**คำแนะนำ** เกมส์และลำดับที่ใช้กับ MIDI ทั่วไปหรือ MT-32 จะยกเว้นพอร์ท MPU-401 MIDI ที่ตำแหน่ง 330H ไม่มีค่าที่กำหนดไว้ สำหรับต้นแบบ Roland MPU-401

ถ้ายุ่งยากกับเพลง MIDI ในเกมส์หรือ sequencer ให้ใช้พอร์ท 330H

- การเชื่อมต่อจุด DASO และ DAS1 เลือกช่อง 8 บิท DMA ของการ์ดเสียงช่อง 8 บิท DMA จะใช้เมื่อเล่นและบันทึกเสียง 8 บิทระบบเสียงดิจิตอล ; 16 บิท DMA พิจารณาจากคำแนะนำ

ช่อง 0 : การเชื่อมต่อจุดบน DAS 0 และ DAS 1

ช่อง 1 : การเชื่อมต่อจุดบน DAS 1 เท่านั้น (ค่าที่กำหนดไว้)

ช่อง 3 : การเชื่อมต่อจุดบน DAS 0 เท่านั้น

- การเชื่อมต่อจุด DBS 0 และ DBS1 เลือกช่อง 16 บิท DMA ของการ์ดเสียง ช่อง 16 บิท DMA จะใช้เมื่อเล่นและบันทึกเสียง 16 บิทระบบเสียงดิจิตอล เลือกตามช่อง 5 , 6 และ 7

ช่อง 5 : การเชื่อมต่อจุดบน DBSO และ DBS1 (ค่านี้กำหนดไว้)

ช่อง 6 : การเชื่อมต่อจุดบน DBS1 เท่านั้น

ช่อง 7 : การเชื่อมต่อจุดบน DBS0 เท่านั้น

การเซทค่าที่กำหนดไว้ จะขัดแย้งกับการ์ด SCSI controller บางตัว เช่น Adaptec 15xx series ถ้ามีการติดตั้งการ์ด SCSI ไว้ให้ตรวจสอบก่อนติดตั้ง SB16 พยายามเปลี่ยน DMA ที่เซทบนการ์ด SCSI ถ้าเกิดมีการขัดแย้งขึ้น

คำแนะนำ ถ้าใช้ DMA ช่อง 6 หรือ 7 และมีปัญหาการเล่นเกมส์ ให้ใช้ DMA ช่อง5 แทนเกมส์เก่า ๆ จะไม่เข้าใจการเซท 16 บิท DMA และจะทำงานในค่าที่เซทไว้เท่านั้น

หมายเหตุ ถ้า motherborard บางบอร์ดมีข้อผิดพลาดที่ DMA controller และไม่สามารถบันทึกเสียงหรือเล่นกลับ 16 บิทได้ DMA ดังนั้น SB 16 จึงออกแบบเพื่อให้ใช้ช่อง 8 บิท DMA เพื่อเล่นและบันทึกเสียง 16 บิทเสียงระบบดิจิตอล การรัน TESTSB16.EXE ในไดเรกทอรี SB16 ให้ดูว่า ถ้า motherboard มีปัญหา ถ้ามันทำการรัน SBCONFIG.EXE ในไดเรกทอรี SB16 และทำตามคำสั่งแก้ในบอร์ดสำหรับ 8 บิท DMA เท่านั้น

- การเชื่อมต่อจุด ISO และ IS1 เลือกฮาร์ดแวร์อินเทอร์รัพท์ หมายเลข (IRQ) ของการ์ด การอินเทอร์รัพท์จะใช้สำหรับระบบเสียงดิจิตอล การบันทึกเสียงและเล่นกลับได้ดีเหมือนกับอินพุต MIDI เลือกจาก 2 , 5 , 7 และ 10

IRQ 2 : การเชื่อมต่อจุดบน ISO และ IS1

IRQ 5 : การเชื่อมต่อจุดบน IS1 เท่านั้น (ค่าที่กำหนดไว้)

IRQ 7 : ไม่มีการเชื่อมต่อจุด ติดตั้งบน ISO และ IS1

- การเชื่อมต่อจุด IFSD และ AFSD แสดงถึง SB 16 ไม่ติดตั้งตัวประมวลผลสัญญาณเสียงชั้นสูง ( Advanced Signal Processor) ถ้าซื้อ SB16 กับชิป จะไม่มีการเชื่อมต่อจุดมาให้

แต่ถ้าเราไม่มีชิป ก็จะเคลื่อนย้ายการเชื่อมต่อจุดไม่ได้ หรือ SB 16 จะถือว่ามีตัวประมวลผลสัญญาณเสียงชั้นสูง ( Advanced Signal Processor) บนการ์ด

- การเชื่อมต่อจุด JYEN จะกำหนดให้ว่า joystrick เปิดหรือปิด เฉพาะเวลาที่ต้องการเคลื่อนย้ายการเชื่อมต่อจุด คือ เมื่อเรามีพอร์ท joystrick ในคอมพิวเตอร์การรวมการ์ดอินพุต / เอาท์พุต มีพอร์ท joystrick ก็จะกำหนดมาให้ ถ้าเรามีการ์ดให้ดูเอกสารประกอบด้วย ถ้าใส่ joystrick เข้าไป ก็ให้เอาบางพอร์ทบนการ์ดอินพุต / เอาท์พุต หรือ ซาวด์บลาสเตอร์ 16 ออก แต่ไม่ต้องเอาออกทั้งหมด

- การเชื่อมต่อจุด IOS 0 และ IOS 1 เลือกตำแหน่ง อินพุต/เอาท์พุต หลักบนการ์ด ตำแหน่ง อินพุต / เอาท์พุต คือ ช่องการติดต่อที่คอมพิวเตอร์ จะใช้ส่งและรับข้อมูลจากซาวด์บลาสเตอร์16 เราสามารถเลือกจากตำแหน่ง 220H , 240H , 260H และ 280H

  220H : การเชื่อมต่อจุดบน IOS 0 และ IOS 1 (ค่าที่กำหนดไว้)

  240H : การเชื่อมต่อจุดบน IOS 1 เท่านั้น

  260H : การเชื่อมต่อจุดบน IOS 0 เท่านั้น

  280H : ไม่มีการเชื่อมต่อจุดติดตั้งบน IOS 0 และ IOS 1

- การเชื่อมต่อจุด JP 14 และ JP 15 ( JP 1 และJP 2 ด้านบน MCD ) จะแตกต่างจากการเชื่อมต่อจุดอื่นบนบอร์ด มันจะไม่มีการเปลี่ยนแปลงลักษณะภายนอกบนบอร์ด แสดงว่า JP 14 และ JP 15 เป็นส่วนเพิ่มเติมสัญญาณเสียงจุดต่างบนบอร์ด การทำเส้นสัญญาณสำหรับไมโครโฟนและลำโพงเข้าคอมพิวเตอร์ ลักษณะรูปร่างของ pin จะแสดงในตาราง 2.5 และ 2.6

ในรูป 2.1 สังเกตุการเชื่อมต่อจุดบน pin 6 และ 7 ของ JP 14 และ บน pin 1 และ 2 ของ JP 15 การเคลื่อนย้ายการเชื่อมต่อจุดจะป้องกันเสียงจากภายนอก บนจุดต่างออกลำโพงบนบอร์ด

**คำเตือน** อย่างทดลองกับ Audio Extention การเชื่อมต่อจุดน้อยมาก ให้ทดลองกับสัญญาณเสียงทางอิเล็คทรอนิคส์ เพราะความผิดพลาดในจุดต่างสามารถทำให้การ์ดเสียงเสียหายได้

| ขั้ว | ชนิด |
|---|---|
| 1 | MICGND (ไมโครโฟน อินพุต กราวด์) |
| 2 | MICGND (ไมโครโฟน อินพุต กราวด์) |
| 3 | MIC IN (ไมโครโฟน อินพุต) ย่านอินพุต 0.004 ถึง 0.7 โวล์ท |
| 4 | SPKGND (สายกราวด์ ลำโพง) |
| 5 | SPKR (ลำโพงข้างขวา)<br>แรงดันเอาท์พุตสูงสุด 3 โวลท์ ที่ 4 โอห์ม |
| 6 | SPKL (ลำโพงข้างซ้าย)<br>แรงดันเอาท์พุตสูงสุด 3 โวลท์ ที่ 4 โอห์ม |
| 7 | SPKRL (สัญญาณป้อนกลับ ลำโพงข้างซ้าย) |
| 8 | SPKRR (สัญญาณป้อนกลับ ลำโพงข้างขวา) |

**ตาราง 2.5 การเชื่อมต่อจุด JP 14 ( JP 1 บน SB 16 MCD ) ลักษณะรูปร่าง pin**

| ขั้ว | ชนิด |
|---|---|
| 1 | SPKR (ลำโพงข้างขวา)<br>แรงดันเอาท์พุตสูงสุด 3 โวลท์ ที่ 4 โอห์ม |
| 2 | SPKRR (สัญญาณป้อนกลับ ลำโพงข้างขวา)<br>แรงดันเอาท์พุตสูงสุด 3 โวลท์ ที่ 4 โอห์ม |

**ตาราง 2.6 การเชื่อมต่อจุด JP 15 ( JP 2 บน SB 16 MCD ) ลักษณะรูปร่าง pin**

- การเชื่อมต่อจุด OPSL และ OPSR จะดังกล่าว การเชื่อมต่อจุด JP 14 และ JP 15 เลือกแบบเอาท์พุตสำหรับจุดต่างออกลำโพง เราจะมีตัวเลือกของกำลังขยายเอาท์พุต และเอาท์พุต line-level เมื่อจุดต่าง SB 16 กับกำลังขยายลำโพง หรือระบบสเตอริโอสำหรับเสียงที่ดี

กำลังขยายลำโพงเอาท์พุต : การเชื่อมต่อจุดต่ำกว่า 2 pin (ค่าที่กำหนดไว้)

เอาท์พุต line-level : การเชื่อมต่อจุดสูงกว่า 2 pin

### 2.1.13 การเชื่อมต่อจุดสำหรับ ซาวด์บลาสเตอร์ 16 SCSI -2

ในส่วนนี้เป็นการเชื่อมต่อลักษณะรุ่น SCSI ขาซาวด์บลาสเตอร์ 16 ดังรูป 2.2 การเชื่อมต่อจุดอื่นๆ เหมือนกับการ์ด ซาวด์บลาสเตอร์ 16 ทั่วไป

- JP 30 เลือก SCSI ตำแหน่งพอร์ทอินพุต / เอาท์พุตหลักเลือกจาก 140H และ 340 H

  140 H : ไม่มีการเชื่อมต่อจุดติดตั้งบน JP 30 ( ค่าที่ตั้งไว้ )

  340 H : ไม่มีการเชื่อมต่อจุดติดตั้งบน JP 30

- การเชื่อมต่อจุด JP 22 ผ่าน JP 25 เลือกอินเทอร์รัพท์ (IRQ) สำหรับ SCSI อินเทอร์เฟส เลือกจาก IRQ 9 , IRQ 10 ,IRQ 11 และ IRQ 12

  IRQ 9 : การเชื่อมต่อจุดบน JP 22 เท่านั้น (ด้านซ้ายสุดในกลุ่มการเชื่อมต่อจุด)

  IRQ 10 : การเชื่อมต่อจุดบน JP 23 เท่านั้น

  IRQ 11 : การเชื่อมต่อจุดบน JP 24 เท่านั้น (ค่าที่กำหนดไว้)

  IRQ 12 : การเชื่อมต่อจุดบน JP 25 เท่านั้น (ด้านขวาสุดในกลุ่มการเชื่อมต่อจุด)

- การเชื่อมต่อจุด JP 18 ผ่าน JP 21 และ JP 26 ผ่าน JP 29 เลือกช่อง DMA สำหรับ SCSI อินเทอร์เฟส เมื่อไดร์ฟรูปแบบชิพ Adaptec SCSI -2 อินเทอร์เฟส ใช้โหมด 32 บิท โปรแกรมอินพุต / เอาท์พุตจะไม่ต้องใช้การเชื่อมต่อ โหมด 32 บิท โปรแกรมจะมี 2 advantage โหมด DMA จะเร็วกว่า 20% และไม่มีเสียงคลิ้กโดยอินเทอร์รัพท์จากการ์ดเสียง

### 2.1.14 การเชื่อมต่อจุดสำหรับซาวด์บลาสเตอร์ 16 MCD

ในส่วนนี้เป็นการเชื่อมต่อลักษณะรุ่น MCD ของซาวด์บลาสเตอร์16 ดังรูป 2.3 การเชื่อมต่อจุดอื่นๆ ก็เหมือนกับการ์ด ซาวด์บลาสเตอร์ 16 ทั่วไป

- การเชื่อมต่อจุด CD 0 (JP 18) และ CD 1 (JP 19) เลือกตำแหน่ง อินพุต/เอาท์พุต หลักสำหรับ Mitsumi CD-ROM อินเทอร์เฟส เลือกจาก 310H, 320H, 340H และ 350H

  310H : การเชื่อมต่อจุดบน CD 0 และ CD 1

  320H : การเชื่อมต่อจุดบน CD 1 เท่านั้น

  340H : การเชื่อมต่อจุดบน CD 0 เท่านั้น (ค่าที่กำหนด)

  350H : ไม่มีการเชื่อมต่อจุดติดตั้งบน CD 0 และ CD 1

- JP 20, JP 21 และ JP 22 เลือกอินเทอร์รัพท์ ( IRQ ) เซทสำหรับมิทซูมิ CD-ROM อินเทอร์เฟส เลือกจาก IRQ 3 , IRQ 10 และ IRQ 11

IRQ 3 : การเชื่อมต่อจุดบน JP 22 เท่านั้น

IRQ 10 : การเชื่อมต่อจุดบน JP 21 เท่านั้น

IRQ 11 : การเชื่อมต่อจุดบน JP 20 เท่านั้น (ค่าที่กำหนดไว้)

- JP 23 และ JP 24 เซทช่อง DMA ของ มิทซูมิ CD-ROM อินเทอร์เฟส เลือกจาก DRQ 6 และ DRQ 7

DRQ 6 : การเชื่อมจุดบน JP 24 เท่านั้น

DRQ 7 : การเชื่อมจุดบน JP 23 เท่านั้น (ค่าที่กำหนดไว้)

- JP 25 , JP 26 , JP 27 และ JP 28 เลือก CD-ROM ไดร์ฟที่มีจุดต่อกับ SB16 MCD เราสามารถนำ CD-ROM อินเทอร์เฟส ออกโดยวางการเชื่อมต่อจุดบน JP 28

พานาโซนิค : การเชื่อมต่อจุดบน JP 25 เท่านั้น (ค่าที่กำหนด)

โซนี่ : การเชื่อมต่อจุดบน JP 26 เท่านั้น

มิทซูมิ : การเชื่อมต่อจุดบน JP 27 เท่านั้น

Disable : การเชื่อมต่อจุดบน JP 28 เท่านั้น

**คำแนะนำ** เราสามารถใช้ CD-ROM ไดร์ฟ 2 ตัวบน MCD ได้ สามารถทำให้ พานาโซนิค และ มิทซูมิอินเทอร์เฟส ทำงานในเวลาเดียวกันได้โดย การเชื่อมต่อจุดบน JP 25- และ JP 27 เราสามารถทำให้โซนี่ และ มิทซูมิอินเทอร์เฟส ทำงานในเวลาเดียวกันได้โดยวาง การเชื่อมต่อจุดบน JP 26 และ JP 27

**หมายเหตุ** เราไม่สามารถใช้อินเทอร์เฟส 3 ตัวในเวลาเดียวกันได้ พานาโซนิค และ โซนี่ อินเทอร์เฟส ไม่สามารถใช้ในเวลาเดียวกันได้ ทั้งพานาโซนิค และ โซนี่อินเทอร์เฟส ใช้พอร์ทอินพุต / เอาท์พุต เหมือนกันและจะขัดแย้งกับตัวอื่นๆ

## 2.1.15 มีอะไรในชื่อ ซาวด์บลาสเตอร์

"16" ใน ซาวด์บลาสเตอร์ 16 แสดงถึงลักษณะเฉพาะของบอร์ด นั้นคือความสามารถในการบันทึกเสียงและเล่นกลับไฟล์ 16 บิท โคเดคที่ใช้ใน SB 16 จะมีคุณภาพสูงเหมือนกับที่พบในเครื่อง DAT ( Digitol Audio Tape ) ประเภทต่างๆ ซึ่งการเพิ่มการบันทึกเสียงจะมีการสุ่มสัญญาณเสียงได้ถึง 44.1 KHz ในสเตอริโอ

การสุ่มสัญญาณเสียงที่ 44.1KHz (44, 100 ลูกคลื่นต่อวินาที ) สามารถแก้ไข 16 บิท สเตอริโอ การบันทึกเสียง 1 นาที ต้องดำเนินการและเก็บมากกว่า 10 MB เท่านั้น เราสามารถทำได้มากกว่า 100 MB ใน 10 นาที

ในโปรแกรมเกมส์เก่า, จะไม่นำเอาความได้เปรียบของความสามารถกับคุณภาพเสียงของ SB16 มาใช้ อย่างไรก็ตามมันจะอยู่ในสภาพที่เปลี่ยนแปลงไปเรื่อยๆ เหมือนเกมส์ใหม่ที่นำมาใช้และมากกว่า SB16 ที่จำหน่ายอยู่ ความต้องการที่จะเก็บเสียง 16 บิทนั้นหาไม่มีที่สิ้นสุด แต่โชคดีที่ CD-ROM กำหนดวิธีการเก็บไฟล์ เสียง 16 บิทใหญ่ๆ นี้ไว้ทำให้สิ้นเปลืองน้อยลง ดังนั้นเกมส์พื้นฐาน CD-ROM ใหม่ๆจะออกแบบมาสำหรับเฉพาะวินโดวส์ที่มีคุณภาพ เสียง 16 บิทสูง

วงจรที่มีคุณภาพสูงของ SB 16 บางครั้งการบันทึกเสียง 8 บิททำได้ไม่ดี หลักการที่ประยุกต์ใช้กับเสียง การติดตั้งที่ดีขึ้น การเปิดเผยข้อผิดพลาดในเครื่องต้นแบบ

SB 16 จะแก้ข้อเสียในการติดตั้งการบันทึกเสียง และวิธีการใช้บนโปรแกรมประยุกต์เมื่อเล่นเสียง 8 บิท จะเป็นที่นิยมในเกมส์เก่าๆ เหมือนกับซาวด์บลาสเตอร์ ทั่วไป ที่ SB 16 จะไม่ใช้ในการกรองความถี่สูงเหมือนการ์ด 8 บิท เพราะมันจะใหญ่พอๆ กัน การกรองเป็นอันตรายต่อเสียง 16 บิท เมื่อนำ SB 16 ไปเล่นเสียง 8 บิทจะมีเสียงรบกวน

คำแนะนำ ถ้ามีเสียงรบกวนมากๆ ในบางเกมส์ให้ลดเสียงแหลมบน SB 16 มิกซ์เซอร์ลง

ลองพิจารณาเสียง 16 บิท กับเสียง 8 บิท จะทำอย่างไรจึงจะเพิ่มพื้นฐานเสียง 8 บิท ได้คุณภาพที่สูงกว่า 16 บิท ความสามารถในการบันทึกเสียงและเล่นกลับของ SB 16 จะทำให้คุณภาพเสียง CD ดีขึ้น เทคโนโลยี 16 บิท เกมส์และโปรแกรมประยุกต์ต่างๆ ที่ใช้ในการบันทึกเสียงตอบสนอง (sound effect) ได้ดีจะทำให้เหมือนจริงได้

## 2.1.16 เทคโนโลยีการประมวลผลสัญญาณเสียงชั้นสูง

DSP ( Digital Sound Processor ) จะเหมือนกับสิ่งที่พบก่อน ซาวด์บลาสเตอร์ สิ่งแรกที่จะอธิบาย วิธีการ และ การเปลี่ยนเสียงธรรมดา เป็น SB 16 ตัวประมวลผลสัญญาณเสียงชั้นสูงเป็นฮาร์ดแวร์ใหม่ที่มีประสิทธิภาพเพิ่มขึ้น บน SB 16 SCSI-2 และ SB 16 MCD สามารถหาซื้อได้หลังจาก upgrade SB 16 แล้ว หลังแท่นเสียบที่พบบน SB 16 S ที่มีมาให้จะง่ายในการติดตั้งชิป

## 2.2 รูปแบบไฟล์คลื่นเสียง (Microsoft waveform Audio File Format:WAV)

การพิจารณารูปแบบไฟล์คลื่นเสียง ( WAV ) รายละเอียดจะอ้างถึงรูปแบบมัลติมีเดียของบริษัท ไมโครซอฟฟ์ จำกัด

ไฟล์คลื่นเสียงถูกกำหนดโดย RIFF ( Resource Interchange File Format ) โครงสร้างนี้ถูกพัฒนาขึ้นเพื่อเป็นต้นแบบไฟล์มัลติมีเดีย

### 2.2.1 รูปแบบไฟล์ RIFF ( RIFF File Format )

พื้นฐานการสร้างบล๊อกของไฟล์ RIFF จะเรียก chunk ซึ่งมีรูปแบบดังนี้

*<rID> <rLen> <rData(rLen)>*

เมื่อ

- *<rID>* "RIFF" กำหนดการแบ่งข้อมูลใน chunk ( 4 ไบท์ )
- *<rLen>* คือความยาวของข้อมูลใน chunk ต่อมา ( 4 ไบท์ )
- *<rData>* คือ RIFF Data Chunk ( ความยาวไบท์ rLen )

ในบล๊อกนี้มีรูปแบบที่ใช้สนับสนุน RIFF ต่างกัน แต่จะพิจารณารูปแบบ WAV ก่อนเสมอ

### 2.2.2 คำนิยามรูปแบบ WAV

รูปแบบ WAV ของ RIFF Data Chunk เป็นการแบ่งส่วนภายใน chunk มันจะมี Format Chunk และตามด้วย Data Chunk เสมอ

*<rData> = <wID> <Format Chunk> <Data Chunk>*

เมื่อ

- *<wID>* "WAVE" เหมือนกับข้อมูลเสียงแบบ WAV ( 4 ไบท์ )

### 2.2.3 WAVE Format Chunk

รูปแบบ chunk มีรูปแบบข้อมูลที่แน่นอนของข้อมูลที่บรรจุใน Data Chunk การสร้างรูปแบบคำสั่งของ Format Chunk มีดังนี้

*<Format Chunk> = <ChunkId> <fLen> <wFormatTag> <nChannels>*

*<nSamplesPerSec> <nAvgBytesPerSec>*

*<nBlockAlign> <FormatSpecific>*

เมื่อ

- *<fId>* "fmt" เหมือนกับบล๊อก Format Chunk ( 4 ไบท์ )
- *<fLen>* คือความยาวของข้อมูลใน Format Chunk ต่อมา ( 4 ไบท์ )
- *<wFormatTag>* แสดงลำดับรูปแบบคลื่น (Wave) ของไฟล์ ( 2 ไบท์ ) ดังตัวอย่าง รูปแบบ Pulse Code Modulation (PCM) = 01
- *<nChannels>* แสดงจำนวนของช่องเอาท์พุต ( 2 ไบท์ ) ดังตัวอย่าง 1 = โมโน (mono), 2 = สเตอริโอ (stereo)
- *<nSamplesPerSec>* แสดงอัตราการสุ่ม ( ครั้งต่อวินาที ) ที่แต่ละช่องอาจเล่นอยู่ ( 2 ไบท์ )
- *<nAvgBytesPerSec>* แสดงจำนวนเฉลี่ยของไบท์ต่อวินาที ซึ่งข้อมูลอาจโอนย้ายได้ ( 2 ไบท์ )

*<nAvgBytesPerSec> = nChannels * nSamplesPerSec * ( nBitsPerSample ÷ 8 )*

- *<nBlockAlign>* แสดงแนวบล๊อก (ในไบท์ ) ของข้อมูลใน Data Chunk การเล่นเทปที่อัดไว้ ซอฟท์แวร์ต้องการกระบวนการ *<nBlockAlign>* ไบท์ของข้อมูลตามเวลา ซึ่งค่าของ *<nBlockAlign>* สามารถใช้สำหรับแนวบัฟเฟอร์ ( 2 ไบท์ )

*<nBlockAlign> = nChannels * ( nBitsPerSample ÷ 8 )*

- *<FormatSpecific>* ส่วนนี้จะประกอบด้วย ศูนย์ หรือ ไบท์ที่เหลือของตัวแปร (2 ไบท์ )

### 2.2.4 ข้อมูล WAV chunk (WAV Data Chunk )

Data chunk บรรจุข้อมูลเสียง WAV ที่แท้จริง รูปแบบของข้อมูลขึ้นอยู่กับ *<wFormatTag>* ที่เก็บค่าใน Format Chunk

*<Data Chunk> = <dId> <dLen> <dData(dLen)>*



เมื่อ

- *<dId>* "data" เหมือนกับบล๊อก Data Chunk ( 4 ไบท์ )

- *<dLen>* แสดงความยาวของข้อมูลใน Data Chunk ต่อมา ( 4 ไบท์ )
- *<dData>* คือข้อมูลรูปคลื่นที่แท้จริง (ความยาวไบท์ dLen )

## 2.3 รูปแบบไฟล์ Musical Instrument Digital Interface (MID)

การพิจารณารูปแบบไฟล์ MIDI รายละเอียดจะอ้างถึงมาตรฐานไฟล์ MIDI ( Standard MIDI Files : SMF ) รุ่น 1.0 ซึ่งผู้กำหนด คือ สมาคมมิดี้ระหว่างประเทศ (International MIDI Association)

ไฟล์ MIDI เกิดจากการนำ chunk มาประกอบกัน โดยแต่ละ chunk ตัวอักษร 4 แบบ และความยาว 32 บิท ซึ่งเป็นจำนวนของไบท์ใน chunk จะมี chunk อยู่ 2 แบบ ที่ใช้มากคือ Header Chunk และ Track Chunk

ไฟล์ MIDI จะเริ่มต้นด้วย Header Chunk เสมอ และตามด้วย Track Chunk ตัวต่อมา

**"Mthd"** ***<length of header data>*** ***<header data>***

**"MTrk"** ***<length of track data>*** ***<track data>***

**"MTrk"** ***<length of track data>*** ***<track data>***

### 2.3.1 MIDI Header Chunk

Header Chunk จะบอกเกี่ยวกับรายละเอียดทั้งหมดของไฟล์ MIDI การสร้างรูปแบบคำสั่งของ Header Chunk มีดังนี้

***<Header Chunk>*** = ***<chunk type>*** ***<length>*** ***<format>*** ***<ntrks>*** ***<division>***

เมื่อ

- ***<chunk type>*** คือรหัส ASCII 4 ตัวอักษร "MThd"
- ***<length>*** คือจำนวนไบท์ 32 บิท ในที่อยู่ของ chunk
- ***<format>*** จะเป็นไปตามชนิดของไฟล์หรือส่วนประกอบของไฟล์

เมื่อ

0 = single multichannel track

1 = ลำดับแทรคที่ทำพร้อมกัน

2 = รูปแบบแทรคเดียวที่เป็นอิสระ

- ***<ntrks>*** คือจำนวนของ track chunk ในไฟล์

- *<divition>* ลักษณะของเดลต้าไทม์ มี 2 รูปแบบ สำหรับ metrical time และ time-code-base time ดังที่แสดงดังตาราง 2.7

### 2.3.2 MIDI Track Chunk

Track Chunk คือส่วนที่เก็บข้อมูลของเพลง แต่ละ Track Chunk คือ ลำดับการไหลของ MIDI โดยค่าเดลต้าไทม์ที่บรรจุรายละเอียดตั้งแต่ 16 ช่อง MIDI ขึ้นไป แนวความคิดของการเพิ่มจำนวนแทรค , การเพิ่มจำนวนเอาท์พุต MIDI , ลำดับและเพลง อาจเป็นเครื่องมือในการใช้ Track Chunk การสร้างรูปแบบคำสั่งของ track chunk มีดังนี้

***<Track Chunk> = <chunk type> <length> <MTrk event> <MTrk event> ...***

| | | | |
|---|---|---|---|
| bit: | 0 | Ticks per quarter-note | |
| | 15 | 14 <----------> 0 | |
| | | | |
| bit: | 1 | Negative SMPTE format | Ticks per frame |
| | 15 | 14 <----------> 8 | 7<----------->0 |

**ตาราง 2.7 ลักษณะของ เดลต้าไทม์**

เมื่อ

- *<chunk type>* คือรหัส ASCII 4 ตัวอักษร “MTrk”
- *<length>* คือจำนวนไบท์ 32 บิทที่อยู่ใน chunk

***<MTrk event> = <delta time> <event>***

เมื่อ

- *<delta time>* คือจำนวนของเวลาต่อมา เก็บเหมือนความยาวตัวแปร

***<event> = <MIDI event> | <sysex event> | <meta event>***

เมื่อ

- *<MIDI event>* ช่องข่าวสาร MIDI

- *<sysex event>* คือ MIDI System-exclusive ข่าวสาร ดังเช่น

  **F0** ***<length>*** ***<data string>***

หรือ

  **F7** ***<length>*** ***<data string>***

เมื่อ

- *<length>* คือที่เก็บเหมือนความยาวตัวแปร
- *<meta event>* บอกเฉพาะเป็น non-MIDI สำหรับเหตุการณ์ เช่น

  **FF** ***<type>*** ***<length>*** ***<data string>***

เมื่อ

- *<type>* คือไบท์เดี่ยว 0-127
- *<length>* คือความยาวตัวแปร

Meta event กำหนดโดยชนิดดังนี้

| | |
|---|---|
| **FF 00 02 ssss** | ลำดับตัวเลข (Sequence Number) |
| **FF 01** ***<len>*** ***<text>*** | กรณีต้นแบบ (text event) |
| **FF 02** ***<len>*** ***<text>*** | ประกาศลิขสิทธิ์ (Copyright Notice) |
| **FF 03** ***<len>*** ***<text>*** | ลำดับ / ชื่อ แทรค (Sequence/ Track Name) |
| **FF 04** ***<len>*** ***<text>*** | ชื่อเครื่องดนตรี (Instrument Name) |
| **FF 05** ***<len>*** ***<text>*** | ไลริค (Lyric) |
| **FF 06** ***<len>*** ***<text>*** | เครื่องหมาย (Marker) |
| **FF 07** ***<len>*** ***<text>*** | คำแนะนำ (Cue Point) |
| **FF 20 01 cc** | ชื่อคำนำหน้าช่อง MIDI (MIDI Channel Prefix) |
| **FF 2F 00** | แทรคสุดท้าย (End of Track) |
| **FF 51 03 tttttt** | เซทเทมโป้ (Set Tempo) |
| **FF 54 05 hr mn se fr ff** | SMPTE Offset |
| **FF 58 04 nn dd cc bb** | สัญญาณเวลา (Time Signature) |
| **FF 59 02 sf mi** | สัญญาณกุญแจ (Key Signature) |
| **FF 7F** ***<len>*** ***<data>*** | Sequencer-Specific Meta-Event |

บางหมายเลขในไฟล์ MIDI กำหนดเหมือนขอบเขตความยาวตัวแปร หมายเลขนี้จะใช้ตั้งแต่ 1 ถึง 4 ไบท์ ซึ่งบางไบท์ใช้น้อยมาก สัญญาณ 7 บิทจะทำการแสดงค่าทุกค่า ยกเว้นไบท์สุดท้ายจะมีบิทสูงสุดมาเซท (บิท 7 ) ไบท์สุดท้ายจะมีบิท 7 มาเคลียร์

นี่คือบางตัวอย่างของตัวเลขกำหนดขอบเขตความยาวตัวแปร

| **ตัวเลข (hex)** | **ตัวแสดงค่า (hex)** |
|---|---|
| 00000000 | 00 |
| 00000040 | 40 |
| 0000007F | 7F |
| 00000080 | 80 00 |
| 00002000 | C0 00 |
| 00003FFF | FF 7F |
| 00004000 | 80 80 00 |
| 00100000 | C0 80 00 |
| 001FFFFF | FF FF 7F |
| 00200000 | 81 80 80 00 |
| 08000000 | C0 80 80 00 |
| 0FFFFFFF | FF FF FF 7F |

### 2.3.3 รูปแบบไฟล์ MIDI ( MIDI File Format )

ขณะที่ ซาวด์บลาสเตอร์ กำลังทำงานจะพบไฟล์ * .MID เพิ่มขึ้นมา

ซึ่งไฟล์ต่อไปนี้ จะใช้ภายใต้วินโดวส์ และสามารถปฏิบัติงาน โดยโปรแกรม Voyetra Sequencer ปัจจุบันซอฟท์แวร์ที่ใช้กับ การ์ดซาวด์บลาสเตอร์ จะรวมโปรแกรม PLAYMIDI.EXE ( ในการเพิ่ม PLAYCMF.EXE ) ซึ่งสามารถเล่นกลับด้านกับไฟล์นี้ได้

โปรแกรมเหล่านี้สามารถอ่านเป็นมาตรฐานไฟล์ MIDI (SMF : Standard MIDI Files) ได้

### 2.3.4 โครงสร้างของไฟล์ MIDI

MIDI โดยชนิดของคำสั่ง เราสามารถเข้าถึงไฟล์นี้ได้ เพราะเป็นไฟล์มาตรฐาน ซึ่งจะมีหลายๆ โปรแกรมเป็นพื้นฐาน

รูปแบบ SMF (Standard MIDI Files) สามารถพัฒนาได้จากหลายโปรแกรม สามารถแลกเปลี่ยนข้อมูล MIDI บนคอมพิวเตอร์ต่างชนิดกันได้ อย่างไรก็ตามรูปแบบนี้ก็ได้ถูกพัฒนาจากรูปแบบแรก ซึ่งกำหนดโดยสมาคมมิดี้ระหว่างประเทศ (International MIDI Association : IMA) อย่างก็ตามรูปแบบนี้ได้ถูกเปลี่ยนแปลงให้ถูกต้องและเป็นมาตรฐาน

### 2.3.5 อิทธิพลจากมาตรฐาน IFF

โครงสร้างของรูปแบบ SMF พื้นฐานส่วนใหญ่ได้จากแนวคิดของ Electronic Arts introduced และมาตรฐาน IFF

ข้อมูลส่วนใหญ่จะถูกแบ่งจาก chunk ซึ่งภายในจะประกอบไปด้วย ชื่อ 32 บิท และค่าความยาว 32 บิท โดยแบ่งตามข้อมูลส่วนใหญ่ที่แท้จริงใน chunk ซึ่งเราสามารถพิจารณากระบวนการนี้จากรูปแบบ RIFF

ดังนั้นรูปแบบ SMF อาจจะเปลี่ยนแปลงได้ จึงไม่ต้องกังวลถึงปัญหาระหว่างการเข้าถึงข้อมูลแบบเก่าและใหม่

**Header + Track = SMF**

ปัจจุบันมี chunk 2 แบบเท่านั้นที่สามารถสร้าง Header chunk และ Track chunk ได้ โดย Header chunk จะใช้ “MThd” และ Track chunk จะใช้ “Mtrk”

header จะปรากฏที่จุดเริ่มต้นของแต่ละไฟล์ แล้วตามด้วยแทรค

SMF ไม่ได้มีเพียง chunk แต่ภายใน chunk แต่ละ chunk จะเหมือนกับ “track in track“ ที่แทนด้วยรูปแบบ SMF ทั้ง 3 ชนิดที่ต่างกัน

**SMF แบบ 0**

SMF chunk แบบ 0 บรรจุได้สูงสุด 1 แทรค เป็นแบบอย่างที่เป็นพื้นฐานของรูปแบบ SMF แบบง่ายสุด สำหรับตัวอย่างนี้ สามารถบรรจุแทรคเครื่องดนตรี ของซีเควนเซอร์ ( sequencer ) ได้

**SMF แบบ 1**

SMF แบบ 1 สามารถทำได้หลายอย่าง เพราะมันสามารถบรรจุได้หลายๆ แทรคในตัวมันเอง ต่างจากแทรคอื่นคือสามารถบรรจุไฟล์ได้ตามลำดับ บางแทรคจะมีรูปแบบเหมือน track-chunk

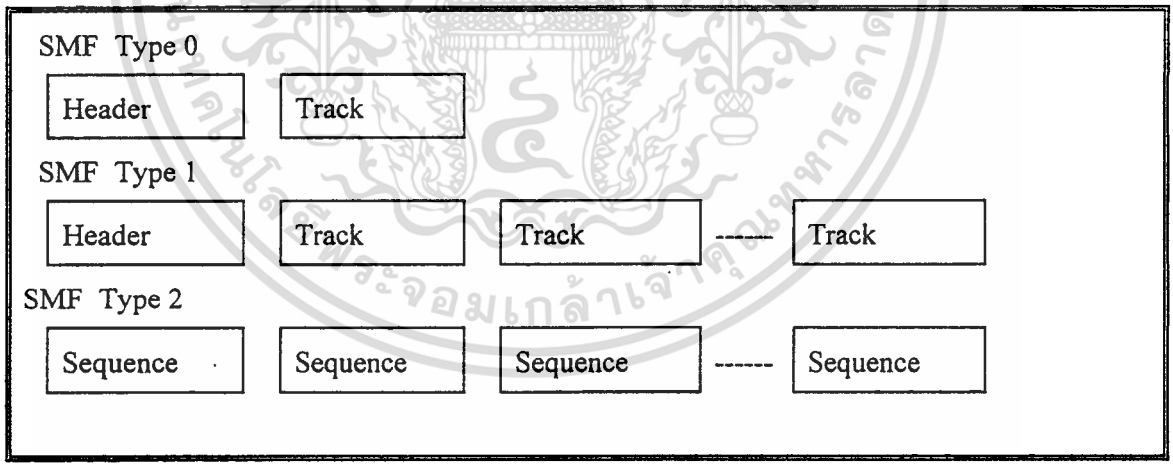
อย่างไรก็ตามแทรคจะบรรจุข้อความที่ใช้อธิบายร่วมด้วย จะมีอยู่ 1 แทรค ที่ใช้ในการโหลด ( load ) วันเดือนปี เพื่อนำไปใช้ในการเปรียบเทียบกับแทรคอื่นๆ

**SMF แบบ 2**

แม้ว่าในแบบที่ 2 นี้จะต่างกันที่ลำดับการบรรจุในแทรคข้อมูลที่บรรจุในแบบนี้ก็ไม่สามารถเล่นในเวลาเดียวกันได้

อย่างไรก็ตามไฟล์ MIDI ในแบบนี้ก็สามารถเก็บได้เพียงลำดับเดียวเท่านั้น ซึ่งเป็นของเครื่องดนตรีชนิดเดียว แม้เราจะสามารถเก็บทุกอย่างลงไปได้

SMF แบบที่ 1 จะเป็นแบบที่ใช้บ่อยที่สุด




**ตาราง 2.8** ข้อแตกต่างของไฟล์ SMF ทั้ง 3 ชนิด

### 2.3.6 Header chunk

Mthd chunk เป็น chunk แรกในไฟล์ SMF เสมอ ซึ่งจะประกอบบด้วย chunk ID 4 ไบท์ และค่าความยาว 4 ไบท์



ความยาว header ที่ใช้บ่อยที่สุด คือ 6 นั่นคือ ความยาวไบท์ตามด้วยข้อมูล 6 ไบท์

**SMF Type info**

ข้อมูล 2 ไบท์แรก จะบรรจุแบบไฟล์ SMF ( มีค่าเท่ากับค่าใดค่าหนึ่ง 0 ,1 หรือ 2 )

**Flip-flopped WORDs**

เมื่อหาค่า WORD จากไฟล์ SMF อย่าลืมลำดับของไบท์ต่ำและสูงจะมีลำดับตรงข้ามกัน กำหนดโดย Intel processors ซึ่งหมายความว่าควรเปลี่ยนค่าไบท์สูงและต่ำ (high and low bytes ) ให้ถูกต้องเสียก่อน

ปัญหาที่เกิดขึ้นในความยาว chunk 32 บิทนั้น อาจนำค่าที่เหมือนกับในไบท์นี้มาใช้ได้ เพราะจะมีไฟล์ Amiga MOD ช่วยแก้ปัญหา

**จำนวนของแทรค**

สองไบท์ต่อไปจะบรรจุจำนวนแทรคที่อยู่ในไฟล์ SMF เหมือนค่า WORD สำหรับไฟล์ SMF แบบ 0 นั้นจะมีค่าเป็น 1 เสมอ

สองไบท์ต่อไปจะมีความหมายแตกต่างกัน ขึ้นอยู่กับบิท 15 จะเซท

ถ้าบิท 15 ไม่เซท บิท 14-0 จะแสดงค่า ซึ่งการแสดงค่าเดลต้าไทม์นั้นจะถูกแบ่งออกเป็น 1/4 ของโน้ตดังตัวอย่าง ไบท์ "00000000 01100000" ชนิด 96 เดลต้าไทม์ ต่อ 1/4 ของโน้ต

อย่างไรก็ตามเมื่อบิท 15 เซทเป็น 1 ก็จะแสดงรหัสเวลาที่ใช้กับไฟล์ MIDI นี้

**ระบบรหัสเวลา**

ระบบรหัสเวลาจะใช้สำหรับ ตัวอย่าง , ระบบวีดีโอเทป บางแถบของวีดีโอเทปจะเป็นรอยเพื่อใช้บอกเวลา ดังนั้นเราจึงสามารถเข้าไปยังแถบที่เราต้องการได้

ประโยชน์ของระบบนี้ คือ สามารถตัดและต่อเทปให้เป็นส่วนของวินาทีได้ ช่วงเวลาแรกเราจะใช้ " เฟรม ต่อ วินาที " ซึ่งเราสามารถใช้กับ ชั่วโมง, นาที, วินาที, เฟรม และเฟรมที่หนึ่งร้อย

เมื่อวีดีโอประกอบเพลงเป็นตัวรวบรวม ทั้งข้อมูลวีดีโอและเครื่องประกอบเพลงก็จะเข้ากันพอดี นี่เป็นเหตุผลหนึ่งที่รูปแบบของเพลงใช้ระบบรหัสเวลาในการค้นหา ระบบนี้จะยอมให้วีดีโอและข้อมูลเสียงเพลงมีรอยต่อในส่วนของวินาที

SMPTE ( ระบบรหัสเวลา MIDI ) คือการเสนอระบบรหัสเวลา ระบบนี้ต้องการจำนวนหน่วยความจำน้อยที่สุดในการเพิ่มข้อมูล MIDI แต่อย่าลืมว่าข้อมูล MIDI มีการโอนย้ายข้อมูลเป็นลำดับ

แม้รหัสเวลาจะเป็นไปตามมาตรฐาน เราก็ไม่สามารถพิจารณาที่ช่วงเวลานั้นได้

เราสารมารถรู้จักระบบรหัสเวลาได้มากกว่านี้โดยอ้างอิงไฟล์ SMF เมื่อบิท 15 เซท, บิท 14 - 8 จะแทนค่าในรูปแบบของค่าลบ 2's คอมพลีเมนท์ ( 2's complement ) จากที่แสดงผลรหัสเวลาเป็น เฟรมต่อวินาที บิท 7 เป็น 0 ตลอดแสดงการกระจายรหัสเวลา

เมื่อพบข้อมูลในไฟล์ header แล้ว ให้นำมาไว้บนแทรคแรก

### 2.3.7 Track chunk

track chunk จะเริ่มต้นด้วย chunk ID และค่าความยาว 32 บิท จากนั้นจะตามด้วยหมายเลขของเหตุการณ์ ( event ) ซึ่งเป็นรูปแบบข้อมูลจริงของ แทรค

<u>เดลต้าไทม์</u> **(Delta time)**

บางกรณีจะกระทำโดยแสดงค่าเวลาที่ใช้ไปแล้ว ก่อนที่กรณีนี้จะทำเสร็จ ค่านี้จะอ้างถึงเดต้าไทม์ ไบท์เดลต้าไทม์ คือ ส่วนประกอบถาวรของเหตุการณ์ ( event )

แนวทางในไบท์นี้ คือ รหัสจะผิดพลาดเพียงเล็กน้อย การรักษาเนื้อที่ ค่าเดลต้าไทม์จะไม่มีการเจาะจงจำนวนความยาวไบท์ที่แน่นอน แทนที่จะใช้บิทที่ 7 ของความยาวไบท์มาแสดงตามด้วยความยาวไบท์อื่นๆ

ความหมายคือ ให้บิท 6 เป็น 0 ตลอดใช้สำหรับค่าความยาว ตามหลักจะยอมให้จำนวนของไบท์เพิ่มขึ้นได้หนึ่งค่า ตามความต้องการของขนาด

อย่างไรก็ตามจำนวนสูงสุดของ 4 ไบท์ที่ใช้มีลำดับเดลต้าไทม์สั้นที่สุด เป็นไบท์ของแบบ "0xxxxxxx" มันจะบรรจุค่าระหว่าง 0 และ 127

ลำดับเดลต้าไทม์ที่ยาวที่สุด คือ "1xxxxxx 1xxxxxx 1xxxxxx 0xxxxxx"

ถ้าต้องการแสดงหมายเลข 255 ในเครื่องหมาย เดลต้าอาจต้องใช้มากกว่า 1 ไบท์ ถ้าใช้ไบท์ที่ 2 ตัวเลขจะเป็นดังนี้ "10000001 01111111" ตัวเลขนี้จะแสดงถึงไบท์สูง /ไบท์ต่ำใช้กับข้อมูลทั่วไป

ค่า 256 ไม่สามารถนำมาใช้ในการคำนวนหาค่าจริงได้ สามารถใช้ค่า 128 แทนได้ เพราะในบิทที่ 7 ไม่ถูกใช้งาน ดังนั้นเราจึงแสดงผลทางโปรแกรมได้

เดลต้าไทม์ซีเควนซ์ ( delta time sequence ) คือจำนวนของกรณี ข้อแตกต่าง 3 แบบ ของกรณีที่เกิดขึ้นในไฟล์ SMF นั่นคือ กรณี MIDI, กรณีระบบข้อความเฉพาะ (System Exclusive) และกรณี Meta

**MIDI event**

ก่อนที่เราจะมาพูดถึงรูปแบบที่ถูกกำหนดโดย IMA กรณี MIDI ที่ได้รวมข้อมูลคำสั่งพื้นฐานของระบบ (System Common)ไว้ด้วย อย่างไรก็ตามก็จะมีข้อมูลคำสั่งควบคุมช่องการทำงาน (Channel Mode)

ระบบข้อความเฉพาะ (System Exclusive) ดูน่าสนใจมาก เริ่มจากความจุของข้อความจะไม่ถูกจำกัดจากมาตรฐาน MIDI ความยาวของข้อความก็สามารถเพิ่มขึ้นได้

ระบบข้อความเฉพาะ (System Exclusive) ในไฟล์ MIDI จะเริ่มต้นที่ไบท์ $F0 ไบท์นี้จะกำหนดตามความยาวของข้อมูล แทนที่ของข้อมูลไบท์แรก ค่านี้มีลักษณะคล้ายค่าเดลต้าไทม์

ไบท์สุดท้ายของระบบเฉพาะ (System Exclusive) คือไบท์ $F7 เริ่มจากความยาวของข้อมูล ซึ่งไบท์นี้ไม่มีความจำเป็นมากนัก อย่างไรก็ตามก็ยังมีเก็บไว้บางส่วน

ภายในไฟล์ MIDI มันจะมีปัญหาในการแบ่งรายละเอียด กรณีระบบเฉพาะจะควบคุมได้ง่ายกว่า ทำให้ข้อมูลรหัสเวลาสามารถเข้าไปยังตำแหน่งที่ต้องการได้

ส่วนที่ 2 ของการแบ่งข้อความนั้น จะเริ่มต้นที่ไบท์ $F7 ข้อความจะไม่สามารถเริ่มที่ $F0 ได้ เนื่องจากมันจะจบด้วย $F7 เสมอ เพื่อแสดงค่าสิ้นสุดของระบบข้อความเฉพาะ

**Meta event**

กรณี Meta เป็นเส่วนพิเศษของไฟล์ SMF มันสามารถบรรจุการเปลี่ยนชนิดของข้อมูลได้

เราจะพิจารณาการใช้กรณีต่างๆ ในรายละเอียดต่อไปนี้

- **กรณี $00 (00)** - ลำดับตัวเลข (Sequence number)

ถ้ากรณี 0 เป็นการรวบรวมไฟล์ SMF มันจะไปอยู่ที่ตำแหน่งเริ่มต้นของแทรคก่อนที่ข้อมูลแรกของ MIDI จะทำการโยกย้าย

กรณีนี้จะมีข้อมูล 2 ไบท์ อธิบายค่า 16 บิทที่บรรจุลำดับตัวเลขของข้อมูลต่อมา

กรณีนี้มีความสำคัญอันดับแรกสำหรับไฟล์ SMF แบบที่ 2 มันสามารถใช้ตรวจสอบลำดับ "ตัวเลข" ที่บรรจุอยู่ในไฟล์นั้นๆ ได้ ถ้าตัดกรณีนี้ทิ้งไปจะพบรายละเอียดในไฟล์ที่ใช้

- **กรณี $01 (01)** - ข้อความทั่วไป (General text)

กรณีนี้จะยอมให้แทรกข้อความที่ต้องการลงในไฟล์ ไบท์ ID $01 ตามด้วยความยาวของข้อความ เหมือนกับบางข้อความที่อธิบายในรูปแบบเดลต้า รหัสนี้จะตามด้วยข้อความในรูปแบบ ASCII

- **กรณี $02 (02)** - ลิขสิทธิ์ข้อความ (Copyright text)

กรณีนี้จะบรรจุข้อความ อย่างไรก็ตามกรณีหมายเลข 1 จะสงวนไว้สำหรับประกาศลิขสิทธิ์ ซึ่งจะประกาศไว้ที่ตำแหน่งแทรคแรกของไฟล์

- **กรณี $03 (03)** - ชื่อแทรค (Track name)

กรณี 3 จะจัดการในการเก็บข้อความในไฟล์ SMF หมายเลข 3 เป็นการจองโดยบอกบนข้อมูลแทรคหรือลำดับข้อมูล

เราสามารถเก็บชื่อทั่วๆ ไปดังเช่น "เบส อินโทร" (bass intro) หรือ "ดรัม โซโล" (drum solo) ในกรณีนี้เราสามารถแสดงข้อมูลเพลงที่เก็บในแทรคนั้นได้ง่าย

ในไฟล์ SMF แบบที่ 2 สามารถใช้กรณีนี้เก็บคำตามลำดับได้

- **กรณี $04 (04)** - ชื่อเครื่องดนตรี (Instrument name)

กรณี 4 จองไว้สำหรับชนิดของเสียงเครื่องดนตรี เราสามาร ถใช้กรณีนี้กับตัวอย่างกรณีที่เปลี่ยนชนิดโปรแกรมในแทรคหรือแสดงเครื่องดนตรีที่เล่นบน ช่องของมัน หลังจากกรณีที่มีชื่อคำนำหน้าช่องนั้น ( กรณี 32 )

- **กรณี $05 (05)** - เพลงไลริค ( Song Lyric )

กรณีนี้จะมีทั้งเพลงและไลริครวมกันอยู่ในไฟล์ SMF หมายความว่ากรณีนี้สามารถแทรกข้อความของเพลงที่พยางค์แรกของเนื้อเพลงได้

- **กรณี $06 (06)** - เครื่องหมาย (Marking)

กรณี 6 เราสามารถทำเครื่องหมายไว้ที่ลำดับได้ ดังนั้นเราจึงสามารถกำหนดช่วงเวลาต่อมาได้ง่าย ค่าความยาวเก็บเหมือนรูปแบบเดลต้า

- **กรณี $07 (07)** - คำแนะนำ (Cue point)

กรณีนี้มีประโยชน์มากเมื่อไฟล์ MIDI ใช้กับอุปกรณ์วีดีโอ คำแนะนำเป็นจุดที่สามารถแนะนำหรือย้อนกลับได้ ลักษณะข้อความในกรณี 7 นั้นจะบรรจุคำแนะนำไว้เป็นพิเศษ

- **กรณี $08 - $0F (08 - 0F)** - ไม่ได้ใช้ (Unused)

กรณี 8 ถึง 15 จะสงวนไว้สำหรับกรณีของข้อความ

- **กรณี $20 (32)** - ชื่อคำนำหน้าช่อง (Channel prefix)

กรณีนี้ใช้แสดงภายหลังกรณี Meta เป็นการแนะลักษณะช่อง MIDI หมายเลขช่องจะเก็บไว้ในไบท์เดี่ยว ตามด้วยหมายเลขกรณี (event ID ) ช่องนี้จะเลือกส่วนที่เหลืออยู่จนกระทั่งมีกรณีที่มีคำนำหน้าช่องใหม่เกิดขึ้นหรือจนกระทั่งพบกับกรณี MIDI ถัดไป

ความหมายของกรณี 32 จะไม่เปลี่ยนข้อมูลของ MIDI ไปไว้ที่ช่องอื่น มันจะใช้บอกจุดมุ่งหมายเท่านั้น เพราะฉนั้นมันสามารถใช้ในการอ้างถึงกรณี 4 ได้ ถ้าหมายเลขช่องของเครื่องดนตรีในกรณี 4 ยังไม่พร้อม

- **กรณี $2F (47)** - แทรคสุดท้าย (End of track)

กรณีนี้จะแสดงแทรคสุดท้าย เมื่อไม่มีข้อมูลเพิ่ม ความยาวของกรณีนี้จะเป็น 0 อย่างไรก็ตามมันจะเป็นเครื่องหมายมาตรฐานของกรณี ค่าความยาวนี้จะเก็บไว้ในรูปแบบเดลต้า หลังกรณี ID ในกรณี 47 มักจะเป็นตำแหน่งของแทรคสุดท้ายเสมอ

- **กรณี $51 (81)** - เซทเทมโป้ ( Set tempo )

กรณี 51 จะบนนจุค่าที่ต่อจากค่าความยาวจำนวน 24 บิท ( 3 ไบท์ ) จำนวนนี้จะแทนค่าเป็นไมโครวินาที (หนึ่งล้านของวินาที ) ภายใน 1/4 ของโน้ต หมายความว่าค่า 1,000,000 แสดง 1/4 ของโน้ตจะเล่นแต่ละวินาที

- **กรณี $54 (84)** - SMPTE offset

กรณี 84 ใช้กับระบบรหัสเวลา SMPTE ซึ่งเราสามารถที่จะพิจารณาได้ง่ายกว่า

กรณีนี้จะบรรจุข้อมูล 5 ไบท์ ซึ่งไบท์นี้ จะมีลักษณะเป็น ชั่วโมง , นาที , วินาที, เฟรม และเฟรมที่หนึ่งร้อย

- **กรณี S58 (88)** - สัญญาณเวลา (Time signature)

ในกรณี 88 คุณจะพบข้อมูล 4 ไบท์ในการเพิ่มค่าความยาวเดลต้า ข้อมูลไบท์แรกจะบรรจุตัวทำเครื่องหมายของฟังก์ชั่นสัญญาณเวลาและไบท์ที่ 2 จะบรรจุคำอธิบายเศษส่วนที่มีค่าส่วนเป็น 2 ดังนั้นค่า 3 ในไบท์นี้จะแสดงตัวเลข " 2 ยกกำลัง 3 " ( มีค่าเท่ากับ 8 )

ข้อมูลไบท์ที่ 3 บอกลักษณะ MIDI clock มันจะแสดง 1 metronome click และไบท์ที่ 4 จะบรรจุหมายเลขโน้ต 32ndth ซึ่งจะตรงกับ1/4 ของโน้ต ( 24 MIDI clock )

กำหนดสัญญาณเวลา 4/4 ด้วย metronome click ที่แต่ละ 1/4 ของโน้ต และ 8 โน้ต 32 ndth ต่อ 24 MIDI clock ดังต่อไปนี้ FF 58 04 04 02 18 08

- **กรณี S7F (127)** - ซีเควนเซอร์ (Sequencer)

มาตรฐาน SMF จะบรรจุกรณี 127 เป็นอันดับแรกจะทำให้สะดวกสำหรับความแตกต่างของซีเควนเซอร์ กรณีนี้จะเหมือนกับระบบข้อความเฉพาะมาก เริ่มตั้งแต่สามารถดัดแปลงฮาร์แวร์หรือซอร์ฟแวร์ที่ต้องการโดยใช้ซีเควนเซอร์

ดังนั้นเราไม่สามารถจัดการกับตัวอย่างของกรณีซีเควนเซอร์ตรงจุดนี้ได้ วิธีการแปลสำหรับไบท์ข้อมูลของกรณีขึ้นอยู่กับการกระทำเฉพาะ

ปัจจุบันเราสามารถตรวจสอบไฟล์ MIDI ได้ละเอียดขึ้น หลังจากมีโปรแกรมช่วย

## 2.4 การเขียนโปรแกรมประยุกต์บนวินโดวส์ด้วย Borland C++

การเขียนโปรแกรมประยุกต์บนวินโดวส์ด้วย Borland C++ นั้นทำได้ง่ายมาก เพราะการออกแบบนั้นจะใช้เครื่องมือจากที่ Borland C++ ให้มา

การเขียนโปรแกรมประยุกต์บนวินโดวส์ด้วย Borland C++ ทำได้ 2 วิธี คือ

1. การออกแบบทั่วไป
2. การเขียนโค้ด (code)

ในวิธีการออกแบบทั่วไปนั้น ไม่จำเป็นต้องเขียนรหัสใดๆ เราสามารถใช้เมาส์และคีย์บอร์ดได้ง่ายขึ้นในการออกแบบกรอบข้อความ, เมนู, bitmaps และต้นแบบที่จะนำไปใช้โดยการประยุกต์ขึ้นเอง ดังตัวอย่างการออกแบบ กรอบข้อความ ที่ตำแหน่งใดๆ ภายในบล๊อก ( ปุ่มกด, ปุ่มเรดิโอ, edit box, ควบคุม multimedia และอื่นๆ )

วิธีการเขียนรหัสที่จะนำมาใช้กับปุ่มกด, เมนูรายการ และตำแหน่งใดๆ ที่จะออกแบบโดยสร้างจากรหัส C++

ก่อนอื่นเราจะต้องรู้ส่วนสำคัญดังต่อไปนี้

- วิธีการออกแบบกรอบข้อความ
- วิธีการสร้างกรอบข้อความ วินโดวส์หลักของโปรแกรมประยุกต์
- วิธีการสร้างรหัสควบคุม กรอบข้อความ
- วิธีการออกแบบเมนูของโปรแกรมประยุกต์
- วิธีการสร้างรหัสเมนูรายการ

### 2.4.1 โปรแกรมประยุกต์ SayHello

ในส่วนนี้เราจะเขียนโปรแกรมประยุกต์บนวินโดวส์อย่างง่าย ชื่อ SayHello โดยจะดูโครงสร้างเป็นขั้นตอน ซึ่งจะทำตามบทเรียนในการออกแบบทั่วไปและรหัสของโปรแกรม Borland C++

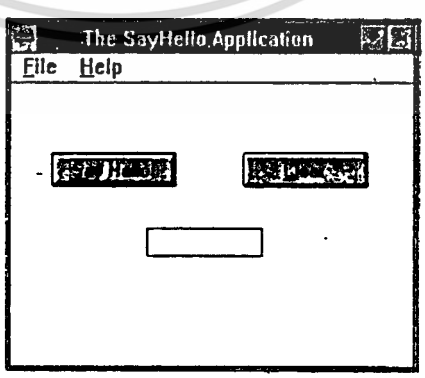
เพื่อให้เข้าใจในโปรแกรมประยุกต์ SayHello มากขึ้นให้ copy โปรแกรมไว้ก่อนที่จะนำมาเขียนในขั้นต่อไป

เมื่อทำการ copy โปรแกรมประยุกต์ SayHello.EXE เรียบร้อยแล้ว

- เลือก Run จากเมนู File ของ Program Manager และพิมพ์

  C:\MMBPROG\ORIGINAL\CH04\SayHello\SayHello.EXE

โปรแกรมวินโดวส์ จะตอบรับโดยแสดง โปรแกรมประยุกต์ SayHello.EXE วินโดวส์หลักของโปรแกรม SayHello จะปรากฏดังรูป 2.4




**รูป 2.4 วินโดวส์หลักของโปรแกรมประยุกต์ SayHello**

จากรูปจะเห็นปุ่ม 2 ปุ่ม (SayHello และ Clear) และ edit box ซึ่ง edit box นั้นจะว่างเปล่า

- คลิ้กปุ่ม SayHello

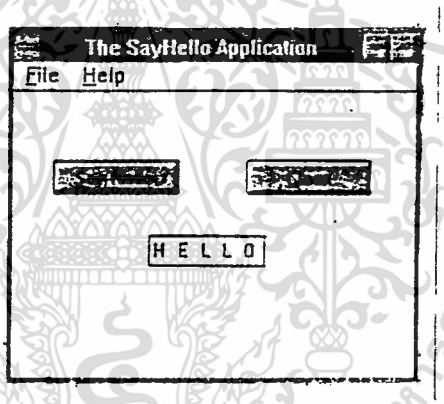
โปรแกรมประยุกต์จะตอบรับโดยแสดงข้อความ “Hello” ภายใน edit box (รูป 2. 5)

- จากนั้น คลิ้กปุ่ม Clear

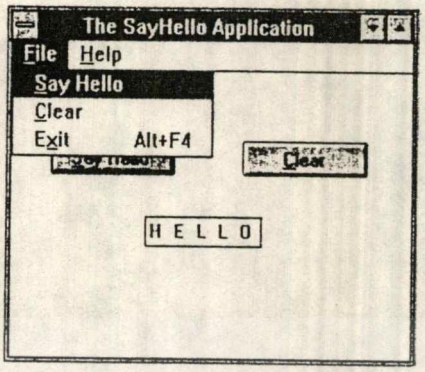
โปรแกรมประยุกต์ SayHello จะตอบรับโดยการเคลียร์ edit box

โปรแกรมประยุกต์ SayHello จะมี menu bar ด้วยกัน 2 ส่วน: File และ Help ในส่วนนี้จะแสดงในรูป 2.6 และ 2.7

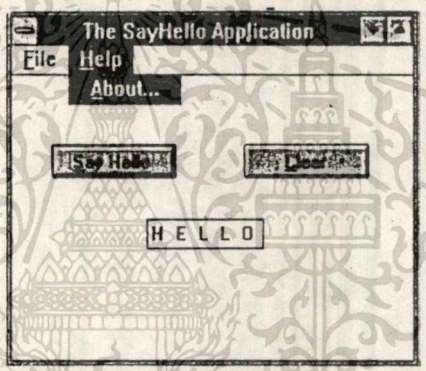
จากรูป 2.6 เมนู File ของโปรแกรมประยุกต์ SayHello จะมี SayHello และ Clear




**รูป 2.5 วินโดวส์หลักของโปรแกรมประยุกต์ SayHello หลังจากคลิ้กปุ่ม SayHello**




รูป 2.6 เมนู File ของโปรแกรม SayHello




รูป 2.7 เมนู Help ของโปรแกรม SayHello

- ทำการทดลองด้วยเมนูรายการ SayHello และ Clear ของเมนู File

จากรูปเมนูรายการนี้จะมีฟังก์ชั่นเหมือนกับปุ่ม SayHello และ Clear

- เลือก about จากเมนู Help

โปรแกรมประยุกต์SayHello จะตอบรับโดยการแสดงกรอบข้อความ About (ดังรูป2.8)


รูป 2.8 กรอบข้อความ About ของโปรแกรมประยุกต์ SayHello

- ปิด กรอบข้อความ About โดยการคลิ้กที่ปุ่ม OK

การจบโปรแกรมประยุกต์ SayHello :

- เลือก Exit จากเมนู File

เมื่อทราบว่าโปรแกรมประยุกต์ SayHello มีลักษณะอย่างไรแล้ว ก็ให้เริ่มเขียนขึ้นมาใหม่ได้ดังนี้

## 2.4.2 การวางแผนสร้างและโครงสร้างไฟล์ของโปรแกรมประยุกต์ SayHello

อันดับแรกในการเขียนโปรแกรมประยุกต์ Borland C++ คือการวางแผนสร้างไฟล์ (*.IDE)

**หมายเหตุ** โปรเจ็กไฟล์ (Project file) ใน Borland C++ เป็นการขยายไฟล์ *.IDE (ตัวอย่าง MyProj.IDE)รายชื่อไฟล์โปรแกรมประยุกต์ของโปรเจ็กไฟล์ทุกไฟล์ที่คอมไพเลอร์ (compiler) ต้องการคอมไพล์ (compile) และติดต่อกับคำสั่งที่ใช้สร้างโปรแกรมประยุกต์ของไฟล์ *.EXE

ในขั้นตอนต่อไปนี้คือการวางแผนสร้างโปรแกรมประยุกต์ SayHello

- เริ่มจาก Borland C++ ดับเบิ้ล-คลิ้กที่ไอคอน (Icon) Borland C++ ในกลุ่มโปรแกรม Borland C++ (ดังรูป 2.9)

วินโดวส์จะตอบรับโดยการเริ่มโปรแกรม Borland C++ วินโดวส์หลักของโปรแกรม Borland C++ จะปรากฏดังรูป 2.10

จากนั้นทำการวางแผนสร้างโปรแกรมประยุกต์ SayHello โดยใช้ AppExpert ซึ่งเป็นยูติลิตี้ที่มีประสิทธิภาพสูงของ Borland C++ AppExpert จะสร้างโปรแกรมประยุกต์ของ

โปรเจ็กไฟล์ และ เทมเพลทไฟล์ (template file) ดังนั้นจึงไม่จำเป็นต้องเขียนรหัสที่ด้านบนสุดทุกครั้งที่เริ่มโปรเจ็กใหม่

ขั้นตอนต่อไปนี้เป็นการสร้างโปรเจ็กไฟล์ และ เทมเพลทไฟล์ ของโปรแกรมประยุกต์ SayHello ด้วย AppExpert :

- เลือก AppExpert จากเมนู Project ของ Borland C++

Borland C++ จะตอบรับโดยการสร้าง กรอบข้อความNew Project

- ใช้ไดเรกทอรี (directory) เรียกรายชื่อของกรอบข้อความ New Project ไปเลือกไดเรกทอรี

C:\MMBPROG\PRACTICE\CH04

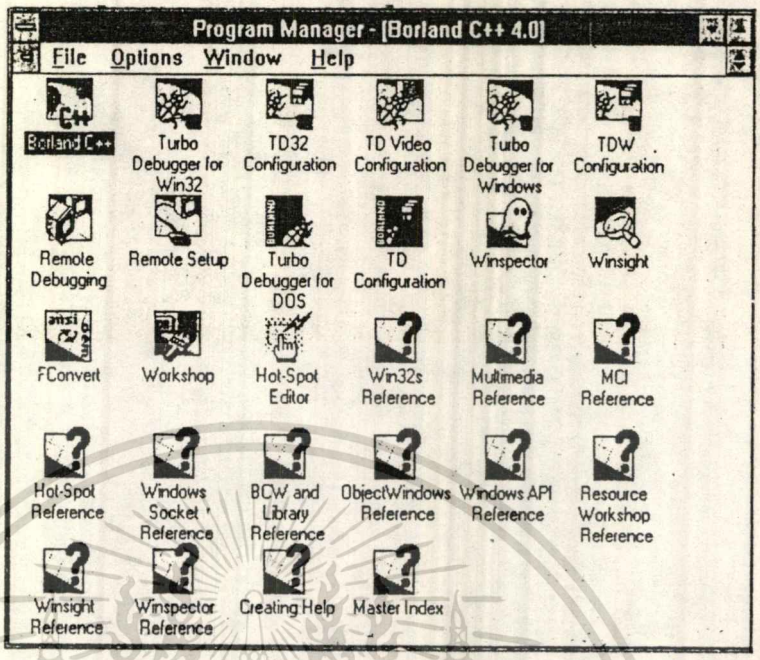
ดูตัวอย่าง กรอบข้อความ New Project จากรูป 2.11

- พิมพ์ SayHello\SayHello.IDE ในบล๊อก File Name

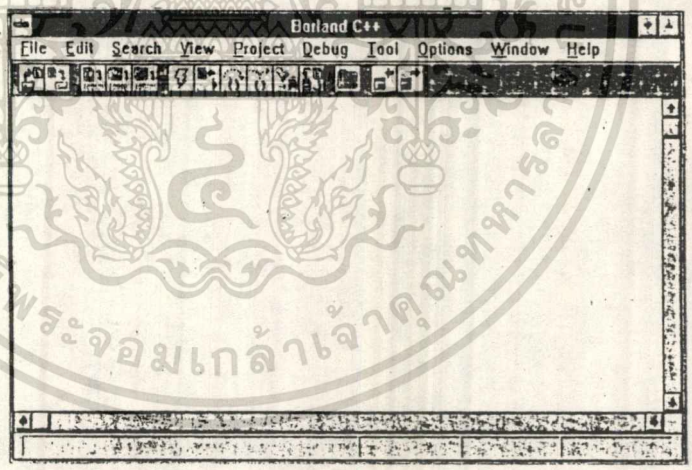
กรอบข้อความ New Project ดูได้จากรูป 2.12 โดยจะมี SayHello\SayHello.IDE เป็นการบอก Borland C++ ว่าจะสร้างไดเรกทอรีย่อยของ SayHello และสร้างโปรเจ็กไฟล์ SayHello.IDE ภายในไดเรกทอรีย่อยนี้

- คลิ้กปุ่ม OK ของ กรอบข้อความ New Project

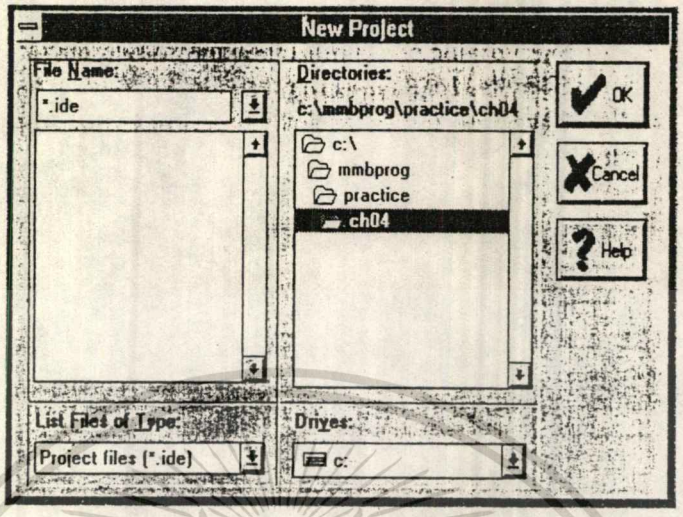
Borland C++ จะตอบรับโดยแสดง AppExpert Application Generation (รูป 2.13) ในขั้นตอนนี้จะใช้กรอบข้อความทำโปรแกรมประยุกต์ที่ต้องการ




**รูป 2.9 ไอคอน Borland C++ ในกลุ่มโปรแกรม Borland C++**




**รูป 2.10 วินโดวส์หลักของ Borland C++**




**รูป 2.11 กรอบข้อความ New Project หลังจากเลือกไดเรกทอรี C:\MMBPROG\PRATICE\CH04**

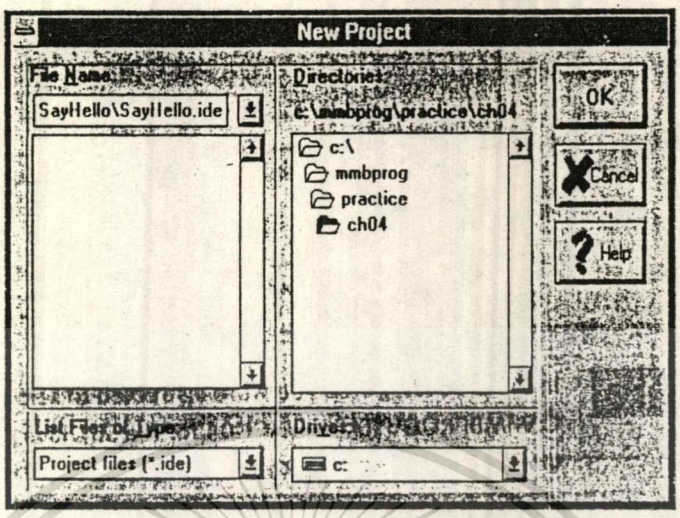
ในรูป 2.13 ส่วนซ้ายสุดของ กรอบข้อความ AppExpert ในส่วน Topics นั้นจะมีอยู่ 3 หัวข้อด้วยกัน :

1. Application
2. Main Window
3. MIDI Child / View

จากหัวข้อทั้ง 3 ให้เลือกที่หัวข้อ Application ซึ่งจะใช้กรอบข้อความ AppExpert ในการเลือกเซทมาที่ Application

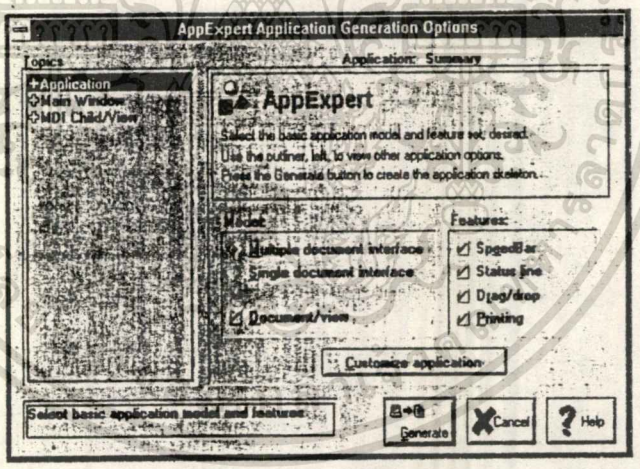
ดังรูป 2.13 หัวข้อ Application ในส่วนที่ 2 (ขวามือ) แสดงรูปแบบ และ โครงร่าง ทำการเซทแบบดังนี้ :

- คลิ้กที่ปุ่ม Single Document อินเทอร์เฟส
- เคลียร์ปุ่ม Document / View




**รูป 2.12 ตั้งชื่อโปรแกรมประยุกต์ SayHello เหมือน**

**C:\MMBPROG\PRACTICE\CH04\SayHello\SayHello.IDE**



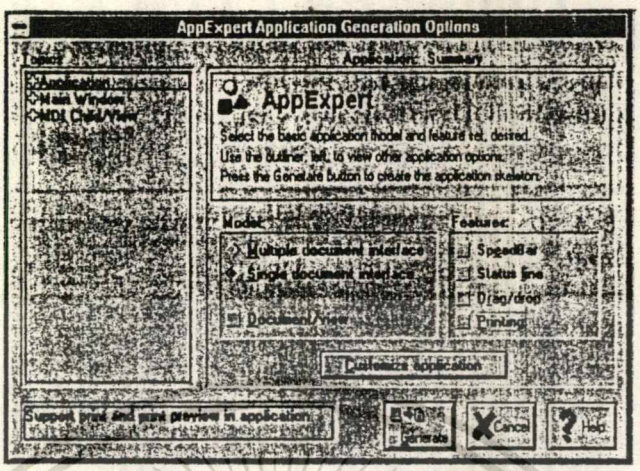

**รูป 2.13 AppExpert Application Generation Option dilog box**

เซทส่วนของ Features ดังนี้

- เคลียร์ทุกปุ่มของส่วน Features

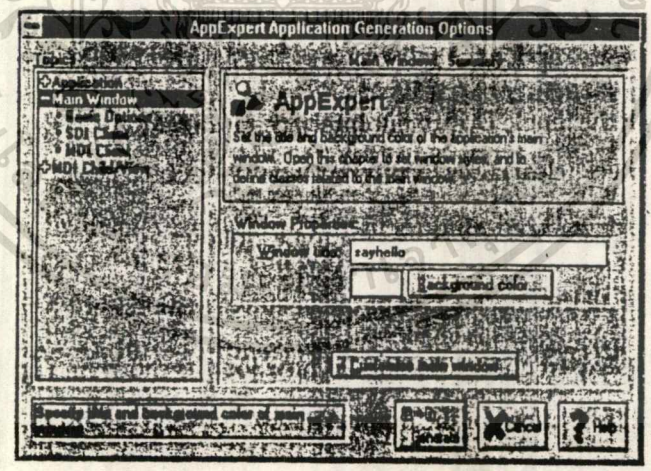
ส่วนกรอบข้อความ .AppExpert ดูได้จากรูป 2.14

ต่อไปเป็นการเซท Main Window ในส่วน Topics




**รูป 2.14 การเซทรูปแบบและโครงสร้างโปรแกรมประยุกต์ SayHello**

Borland C++ จะตอบรับโดยแสดง 3 หัวข้อย่อย : Basic Option , SDI Client และ MIDI Client (ดังรูป 2.15)

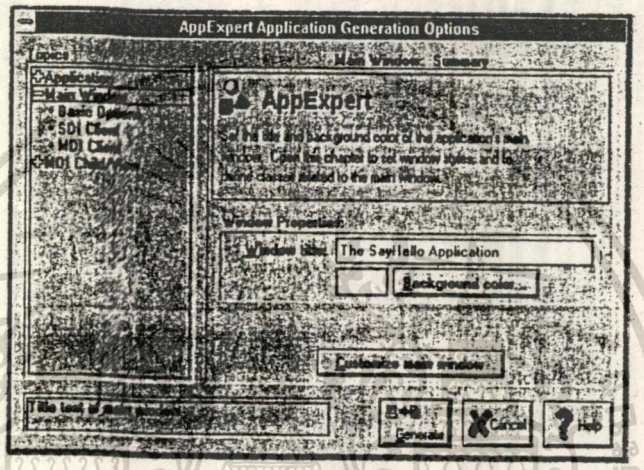



**รูป 2.15 ส่วนหัวข้อย่อยของ Main Window**

ต่อไปนี้ให้ดูที่ edit box (Window title) จะเป็นรูปแบบชื่อวินโดวส์หลักของโปรแกรมประยุกต์ แต่ที่ปรากฏอยู่เป็นชื่อ SayHello เดิม

- เปลี่ยนภายใน Window title edit box เป็น : The SayHello Aplication (รูป 2.16)

- เลือกหัวข้อย่อย SDI Client




**รูป 2.16 เซทชื่อวินโดวส์ของโปรแกรมประยุกต์วินโดวส์หลัก**

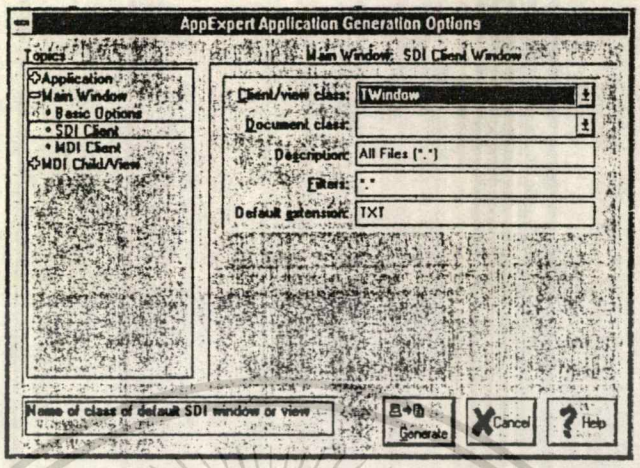
Borland C++ จะตอบรับโดยแสดงแบบทางด้านขวามือของ กรอบข้อความ

- เซท SDI Client ดังรูป 2.17 จากนั้นเซทที่ Client / View Class ที่ลูกศรลงให้เลือก Twindow และใช้คำนั้น

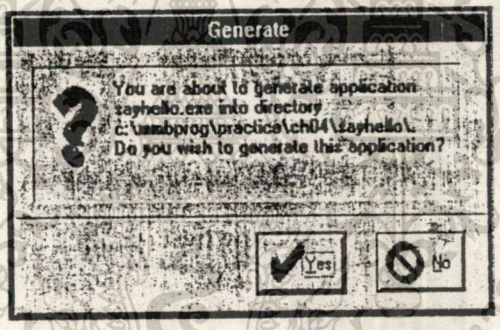
เมื่อทำตามขั้นตอนทั้งหมดแล้วก็จะเสร็จสิ้นการเซทโปรแกรมประยุกต์ SayHello

- คลิ้กปุ่มที่กรอบข้อความ Generate

Borland C++ จะตอบรับโดยการยืนยันว่าต้องการสร้างไฟล์ของโปรแกรมประยุกต์ SayHello (รูป 2.18)



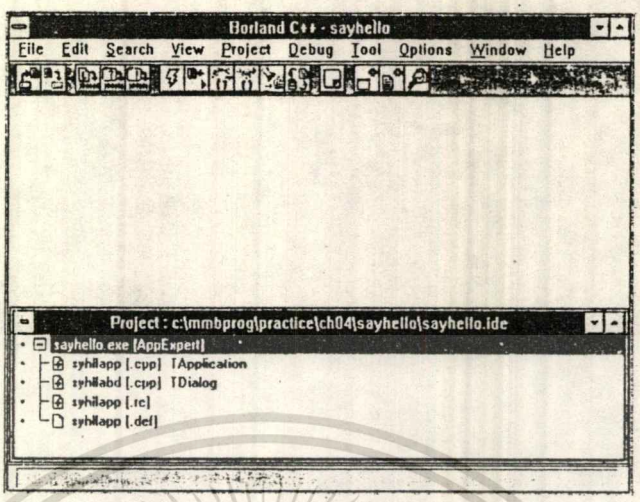

**รูป 2.17 เซท SDI Client สำหรับโปรแกรมประยุกต์ SayHello**




**รูป 2.18 กรอบข้อความ Generate**

- คลิ้กปุ่ม YES

Borland C++ จะตอบรับโดยสร้างโปรเจ็กไฟล์ (SayHello.IDE) และโครงสร้างไฟล์ของโปรแกรมประยุกต์ SayHello ภายในไดเรกทอรี C:\MBPROG\PRACTICE\CH04\Say-Hello โปรเจ็กวินโดวส์ของโปรแกรมประยุกต์ SayHello จะปรากฏดังรูป 2.22 พิจารณาไดเรกทอรี C:\MMBPROG\PRACTICE\CH04\SayHello ว่าตรงกันกับไฟล์ที่สร้าง




รูป 2.19 โปรเจ็กวินโดวส์ของโปรแกรมประยุกต์ SayHello

### 2.4.3 การรันโปรแกรมประยุกต์ SayHello

แม้ว่าจะไม่ได้เขียนโค้ด (code) แต่ละบรรทัดเอง โครงร่างไฟล์ที่ AppExpert สร้างขึ้นก็จะมีโค้ดบางตัวอยู่ก่อนแล้ว

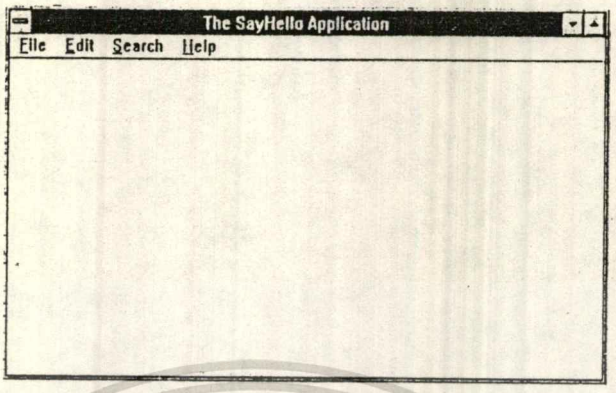
- เลือก Run จากเมนู Debug

Borland C++ จะตอบรับโดย การคอมไพล์ / ลิงค์ ไฟล์ของโปรแกรมประยุกต์ SayHello ไปยังไฟล์ SayHello.EXE จนเสร็จสิ้น วินโดวส์หลักของ SayHello.EXE จะปรากฏดังรูป 2.20

ชื่อของวินโดวส์หลักจะเหมือนกับใน AppExpert คือ “The SayHello Application”

การทดลองด้วยโปรแกรมประยุกต์ SayHello รายละเอียดจะอยู่ในเมนู About ของเมนู Help โปรแกรมประยุกต์จะตอบรับโดยแสดงกรอบข้อความ About (รูป 2.14) ซึ่งหมายถึงโปรแกรมประยุกต์ SayHello รวมทั้งโค้ดแสดงกรอบข้อความ About

โค้ด คือ ส่วนที่ AppExpert ให้มา ถ้าทดลองกับเมนูรายการของโปรแกรมประยุกต์ SayHello ก็จะมีมาให้พร้อม (เหมือนมาจาก AppExpert) ในเมนูทั่วไปกับบางฟังก์ชั่น อย่างไรก็ตามมันจะไม่เป็นโปรแกรมประยุกต์ SayHello ที่สร้างขึ้นเองทั้งหมด




**รูป 2.20 วินโดวส์หลักของ SayHello.EXE**

- ออกจากโปรแกรมประยุกต์ SayHello โดยเลือก Exit จากเมนู File

ในส่วนต่อไปนี้จะใช้เครื่องมือของ Borland C++ ในการกำหนดไฟล์ของโปรแกรมประยุกต์ SayHello ที่ต้องการ

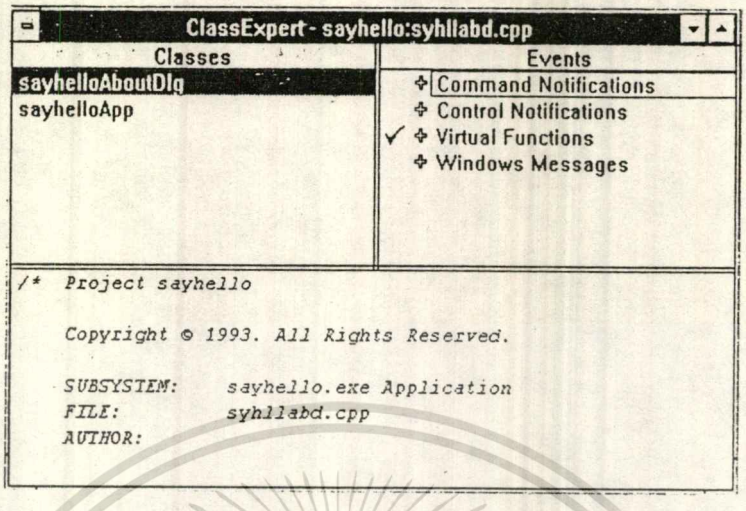
### 2.4.4 การสร้างกรอบข้อความที่จะใช้เช่นเดียวกับวินโดวส์หลัก

จากรูป 2.4 ที่วินโดวส์หลักของโปรแกรมประยุกต์ SayHello บรรจุ 2 ปุ่ม (SayHello และ Clear) และ edit box ในขั้นตอนต่อไปนี้เป็นการสร้างกรอบข้อความ ; หลังจากนั้นจะสามารถออกแบบได้เหมือนวินโดวส์หลักของโปรแกรมประยุกต์ SayHello

ขั้นตอนต่อไปนี้เป็นการสร้าง กรอบข้อความ

- เลือก ClassExpert จากเมนู View ของ Borland C++

Borland C++ จะตอบรับโดยแสดงวินโดวส์ ClassExpert (รูป 2.21)




**รูป 2.21 วินโดวส์ของ ClassExpert**

ClassExpert คือเครื่องมือที่มีประสิทธิภาพ ซึ่งสามารถเขียนประกาศคลาสได้ง่าย เมื่อเขียนโปรแกรมประยุกต์วินโดวส์ด้วย Borland C++ จะต้องประกาศคลาส ซึ่งบางส่วนจะอยู่ใน OWL คลาสจะเป็นตัวประกาศว่าต้องการใช้โค้ด และ ClassExpert ก็จะเขียนโค้ดขึ้นมา

ClassExpert วินโดวส์แบ่งออกเป็น 3 ส่วน

1. Classes
2. Events
3. Code sections (ที่ปุ่มของวินโดวส์)

ต่อไปนี้จะพิจารณาเพียงส่วน Classes เท่านั้น (ส่วนของ Events และ Code sections นั้นจะศึกษาเมื่อต้องเขียนโค้ดของโปรแกรมประยุกต์)

Class แบ่งออกเป็น 2 ส่วน : sayhelloAboutDlg และ sayhelloApp ทั้งสองคลาสนี้เป็นที่ AppExpert ให้มา นั่นคือเมื่อใช้ AppExpert ในการสร้างแผนงานและโครงร่างไฟล์ของโปรแกรมประยุกต์ SayHello AppExpert เขียนโค้ดของทั้งสองคลาสนี้

คลาส sayhelloAboutDlg จะคล้ายกับกรอบข้อความ About ของโปรแกรมประยุกต์ SayHello มันจะบรรจุทุกๆ โค้ดเป็นการตอบสนองสำหรับแสดงกรอบข้อความ About

คลาส sayhelloApp เป็นการแสดงโปรแกรมประยุกต์ คลาสนี้ได้มาจากคลาส OWL Tapplication ซึ่งจะเหมือนกับโปรแกรมประยุกต์ ฟังก์ชั่นของคลาสนี้เริ่มต้นที่วินโดวส์หลักของโปรแกรมประยุกต์

หัวข้อนี้โค้ดของโปรแกรมประยุกต์ SayHello จะมี 2 คลาส sayhelloAboutDlg และ sayhelloApp ซึ่งโค้ดของ sayhelloAboutDlg จะตอบรับสำหรับแสดงกรอบข้อความ About และโค้ดของ sayhelloApp จะตอบรับสำหรับ "main" ใช้กับโปรแกรมประยุกต์ทั่วไป ( สำหรับตัวอย่างเริ่มต้นที่วินโดวส์หลักของโปรแกรมประยุกต์ ) โดยโค้ดทั้งสองคลาสนี้สร้างโดยอัตโนมัติจาก AppExpert

ตอนนี้จุดมุ่งหมาย คือ การสร้างกรอบข้อความที่จะใช้กับวินโดวส์หลักของโปรแกรมประยุกต์ ในขั้นตอนนี้จะใช้ ClassExpert สร้าง กรอบข้อความ และสร้างคลาส ซึ่งจะสัมพันธ์กับกรอบข้อความนี้

- คลิ้กปุ่มขวา จะเป็นรายการ sayhelloAboutDlg หรือรายการ sayhelloApp ก็ได้ ในส่วน Classes ของ ClassExpert วินโดวส์ก็ได้ (คลิ้กปุ่มขวาของเมาส์บน sayhelloAboutDlg หรือบน sayhelloApp )

Borland C++ จะแสดง popup เมนู ดังรูป 2.22 ซึ่งจะแสดงหลังจากคลิ้กปุ่มขวา sayhelloAboutDlg

- เลือก Creat New Class จาก popup เมนู

Borland C++ จะตอบรับโดยแสดงกรอบข้อความ Add New Class (รูป 2.23)

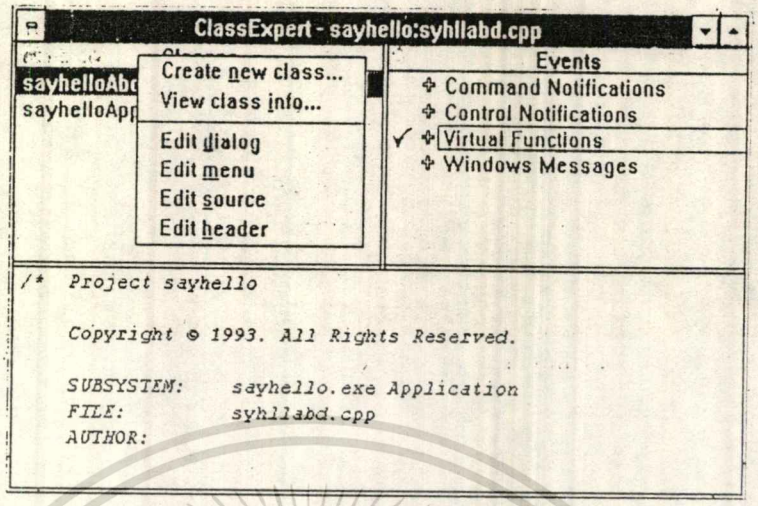
เซทขอบเขตของกรอบข้อความ Add new Class ได้ดังนี้ :

- เซทที่ Base Class ให้เป็น Tdialog (หรือใช้ Base Class ที่ลูกศรลง เป็นรายการที่ทำมาให้แล้ว)
- พิมพ์ TMainDlg ใน Class Name

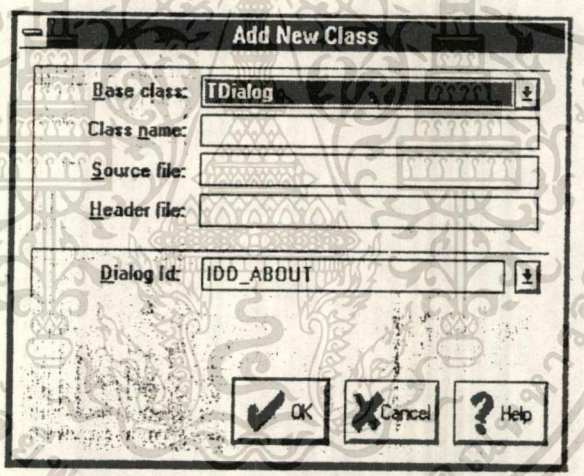
หมายเหตุ ในขั้นตอนนี้เป็นโครงสร้างชื่อคลาสใหม่ TMainDlg เพราะคลาสที่สร้างขึ้นสำหรับกรอบข้อความ จะทำเหมือนวินโดวส์หลักของโปรแกรมประยุกต์ เพราะฉะนั้นจะสามารถตั้งชื่อคลาสเป็นอะไรก็ได้ (เช่น TMyDlg) อย่างไรก็ตามชื่อ TMainDlg จะเหมือนกับที่แสดงชื่อคลาสนี้จะเหมือนกับในกรอบข้อความหลักของโปรแกรมประยุกต์ทั้งส่วน Source File และ Header File ที่ Source File ให้เซทเป็น tmaindlg.cpp และ Header File เซทเป็น tmaindlg.h

- พิมพ์ IDD_MAINDLG ใน Dialog ID

ซึ่งกรอบข้อความ Add New Class จะแสดงดังรูป 2.24



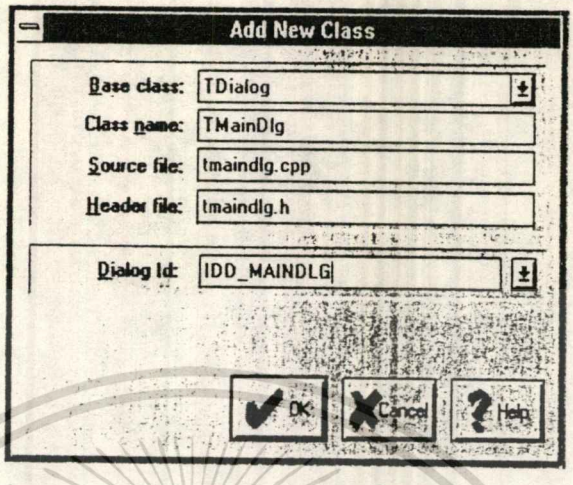

**รูป 2.22 popup เมนู ซึ่งปรากฏหลังจากคลิกปุ่มขวาที่รายการ sayhelloAboutDlg**




**รูป 2.23 กรอบข้อความ Add New Class**

ทำการปิดส่วนที่ทำการเซทไว้ของ กรอบข้อความ Add New Class

- เซท Base Class เป็น Tdialog เป็นการบอก ClassExpert ว่าการสร้างคลาสสมมติจากคลาส Tdilog คลาส OWL เป็นลักษณะการออกแบบด้วยกรอบข้อความ
- เซท Class Name เป็น TMainDlg ดังนั้นคลาสที่ ClassExpert จะสร้างให้นั้นเรียกว่า TMainDlg




**รูป 2.24 Add New Class**

- ชื่อ Source File จะเซทอัตโนมัติ เซทเป็น tmaindlg.cpp และ Header File โดยอัตโนมัติ เซทเป็น tmaindlg.h ClassExpert จะเขียนทุกโค้ดที่จำเป็นสำหรับการสร้าง TMainDlg class ภายในไฟล์ tmaindlg.cpp และ tmaindlg.h
- เซท Dialog ID เป็น IDD_MAINDLG หมายความว่ากรอบข้อความใหม่ที่สร้างขึ้นนี้จะมีค่าเป็น IDD_MAINDLG เสมอ

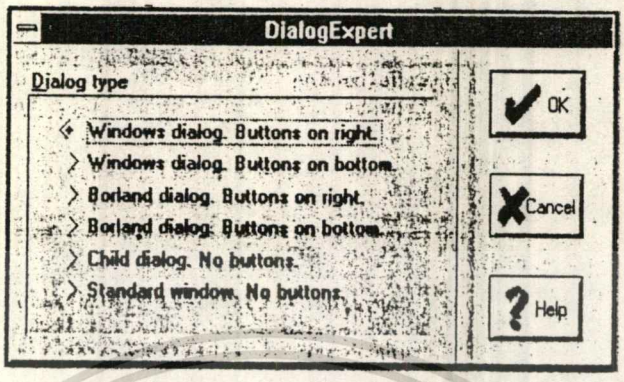
เมื่อพิมพ์บอก ClassExpert ที่สร้างคลาสใหม่ (TMainDlg) และกรอบข้อความใหม่ (IDD_MAINDLG) เรียบร้อยแล้ว

- คลิ้กที่ปุ่ม OK ของกรอบข้อความ Add New Class

Borland C++ จะตอบสนองโดยการรันโปรแกรมเรียก Resource Workshop ซึ่งแสดงกรอบข้อความ DialogExpert (ดังรูป 2.25)

Resource Workshop เป็นโปรแกรมที่มีประสิทธิภาพมาก ที่สามารถออกแบบได้เอง เช่น กรอบข้อความ , เมนู , bitmap , ไอคอน และอื่นๆ

ClassExpert ตอบสนองโปรแกรม Resource Workshop เพราะในขั้นตอนนี้จะแสดงว่าต้องการสร้างจากคลาส OWL TDialog ดังนั้นมันจะตอบสนองโดยการสร้างกรอบข้อความขึ้นมา ID ของกรอบข้อความนี้จะเป็น IDD_MAINDLG เพราะมันเป็น ID ในกรอบข้อความ Add New Class (ดังรูป 2.24)




**รูป 2.25 กรอบข้อความ DialogExpert**

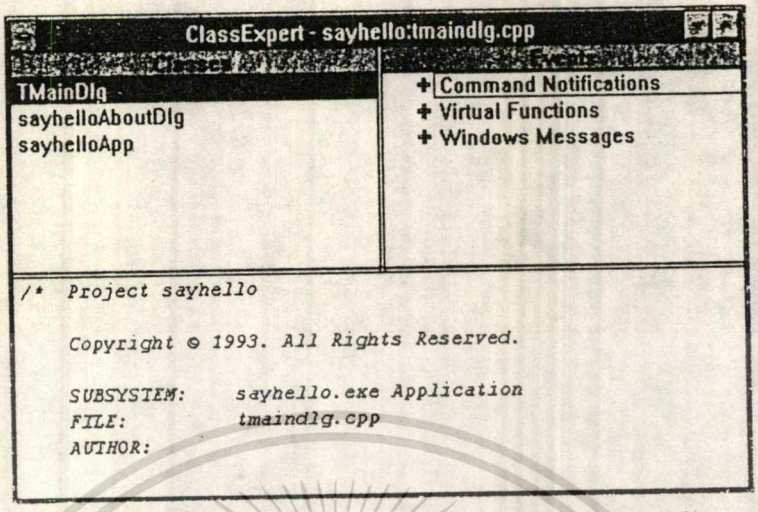
กรอบข้อความ DialogExpert (รูป 2.25) สามารถเลือกรูปแบบกรอบข้อความที่ต้องการสร้างได้ เพราะการสร้างกรอบข้อความ จะสามารถทำได้เหมือน กรอบข้อความหลักของโปรแกรมประยุกต์นั้น จะต้องเลือก Child Dialog No Buttons

- คลิ้กปุ่ม Child Dialog No Button แล้วคลิ้กปุ่ม OK

Resource Workshop จะตอบสนองโดยการสร้างกรอบข้อความ child ซึ่งจะแสดงวินโดวส์ ClassExpert อีกครั้ง (รูป 2.26) ส่วนคลาสจะเป็นคลาสใหม่ที่สร้างขึ้นมาเอง

หัวข้อที่ต้องทำต่อไปนี้

- สร้างโปรเจ็กและเทมเพลทไฟล์ของโปรแกรมประยุกต์ SayHello โดยใช้ AppExpert
- เพิ่มคลาสใหม่ที่ชื่อ TMainDlg เป็นโปรแกรมประยุกต์ SayHello ใช้ ClassExpert โดยนำคลาสนี้มาจาก OWL TMainDlg และสร้างกรอบข้อความ chil (IDD_MAINDLG) ซึ่งจะสัมพันธ์กับคลาสนี้



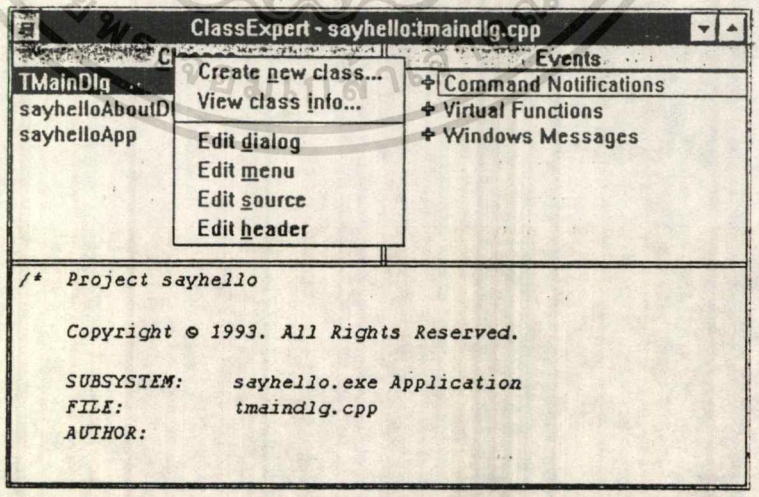

**รูป 2.26 ClassExpert วินโดวส์หลังจากเพิ่มคลาส TMainDlg**

### 2.4.5 ตำแหน่งควบคุมภายในกรอบข้อความ IDD_MAINDLG

IDD_MAINDLG คือส่วนทั้งหมด ซึ่งมันจะไม่สามารถควบคุมได้ เช่นเดียวกับงานที่ทำขึ้น IDD_MAINDLG dialog ดังรูป 2.4 (มี 2 ปุ่ม และ edit box) ดังต่อไปนี้ :

- คลิ้กปุ่มขวาที่รายการ TMainDlg ภายในส่วนของ Classes ของ ClassExpert

Borland C++ จะตอบสนองโดยแสดง popup เมนู (รูป 2.27)

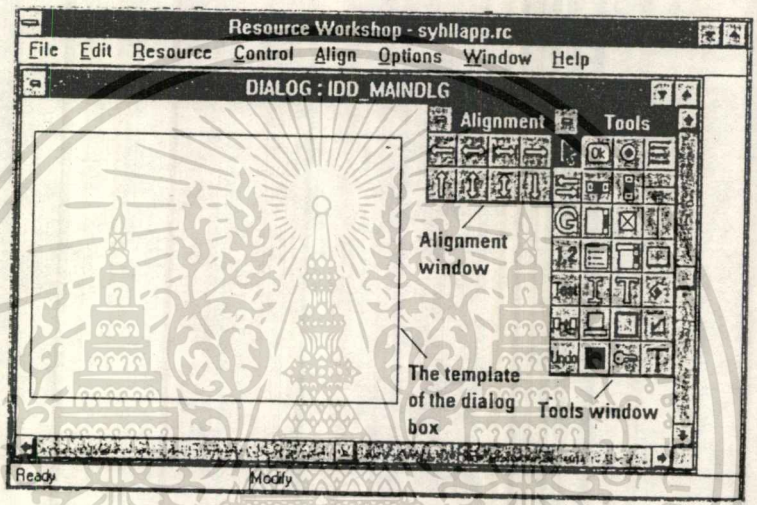





**รูป 2.27 popup เมนู ซึ่ง ClassExpert แสดงหลังจากคลิ้กปุ่มขวาที่ TMainDlg**

- เลือก Edit Dialog จาก popup เมนู (ต้องการกรอบข้อความ edit ของคลาส TMainDlg )

Borland C++ จะตอบสนองโดยการรัน Resource Workshop ซึ่งวินโดวส์หลักจะแสดงดังรูป 2.28 (อาศัยรูปแบบของโปรแกรม Resource Workshop ในวินโดวส์ที่เห็นจะไม่ยากนักดังตัวอย่างเครื่องมือวินโดวส์ที่ใช้ในการควบคุม)




**รูป 2.28 Resource Workshop กับ กรอบข้อความ IDD_MAINDLG พร้อมสำหรับใช้**

จากรูป 2.28 Resource Workshop จะแสดงวินโดวส์ ซึ่งหัวเรื่องเป็น DIALOG IDD_MAINDLG หมายความว่าต่อไปนี้จะใช้ Resource Workshop แสดงกรอบข้อความ IDD_MAINDLG DIALOG : IDD_MAINDLG วินโดวส์จะบรรจุด้วย 3 วินโดวส์

- เทมเพลทที่ออกแบบขึ้นเอง
- เครื่องมือวินโดวส์
- Alignment วินโดวส์

เทมเพลทของกรอบข้อความจะว่างเปล่า ถ้าต้องการเพิ่มการควบคุม ( 2 ปุ่ม และ edit box) ดังรูป 2.4

เครื่องมือวินโดวส์ใช้ควบคุมตำแหน่งภายในเทมเพลทของกรอบข้อความโดยใช้ Alignment ในการวางแนวทางของวินโดวส์

### 2.4.6 วินโดวส์แสดง Tool และ Alignment

ถ้าวินโดวส์ tool และ alignment ไม่มีปรากฏในวินโดวส์ของ Resource Workshop สามารถทำให้มันปรากฏได้ดังนี้

ถ้าวินโดวส์ tool ถูกซ่อนไว้ สามารถทำให้มันปรากฏได้ โดยเลือกให้แสดง tool จากเมนูของ Resource Workshop แทน

ถ้า Alignment วินโดวส์ถูกซ่อนไว้ สามารถทำให้มันปรากฏได้ โดยเลือกให้แสดง Alignment จากเมนูของ Resource Workshop

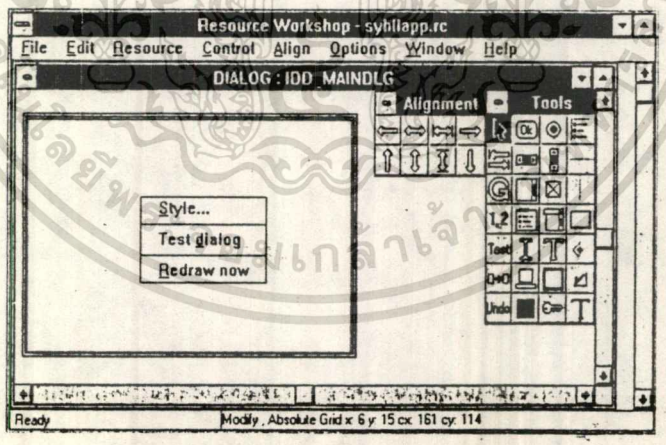
ก่อนอื่นต้องดูตำแหน่งควบคุมภายในเทมเพลทของกรอบข้อความ IDD_MAINDLG อันดับแรกไปที่ Style ของกรอบข้อความนี้ แล้วเซทให้ถูกต้อง มีหน้าที่เหมือนวินโดวส์หลักของโปรแกรมประยุกต์

ต่อไปนี้เป็นการเซท Style ของกรอบข้อความ IDD_MAINDLG

- คลิ้กปุ่มขวาที่เมาส์ ที่ใดก็ได้ภายในเทมเพลทของกรอบข้อความ

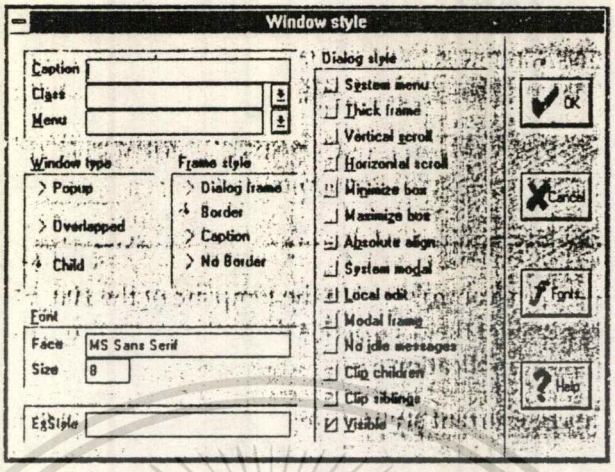
Resource Workshop จะตอบสนองโดยแสดง popup เมนู(รูป 2.29)

- เลือก style จาก popup เมนู




**รูป 2.29 popup เมนู Workshop แสดงหลังจากที่ Resource คลิ้กปุ่มขวาภายในเทมเพลท**

Resource Workshop จะตอบสนองโดยแสดงกรอบข้อความ Window Style (รูป 2.30)




**รูป 2.30 กรอบข้อความ Window style**

การเซทที่แสดงในรูป 2.30 นั้นจะเหมือนกับวินโดวส์หลักของโปรแกรมประยุกต์ ทำการเซท Window style ให้เหมือนดังที่แสดงไว้ได้ดังนี้

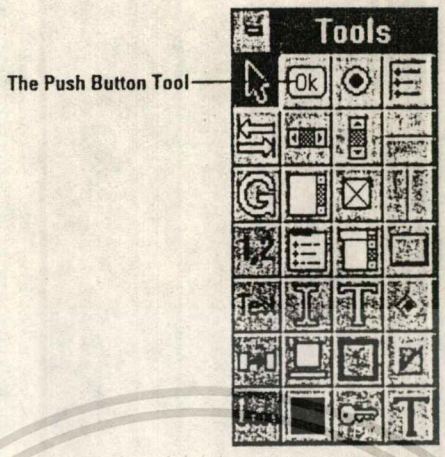
- ที่ Caption จะว่างอยู่
- Window style เซท hild
- Fram style เซท Border
- ใน Dialog style ให้เช็คที่ Visible เท่านั้น
- Close กรอบข้อความ Window Style คลิ้กที่ปุ่ม OK

ขณะนี้จะควบคุมตำแหน่งภายในเทมเพลทของกรอบข้อความ IDD_MAINDLG ทำได้ดังนี้

- ภายใน Tools คลิ้กที่ปุ่มเครื่องมือ (สร้าง OK เหมือนรูป 2.31)

Resource Workshop จะตอบรับโดยเปลี่ยนเมาส์เคอร์เซอร์เป็นปุ่มไอคอน

- คลิ้กเมาส์ภายในเทมเพลทของกรอบข้อความ




**รูป 2.31 ปุ่มเครื่องมือภายใน Tools วินโดวส์**

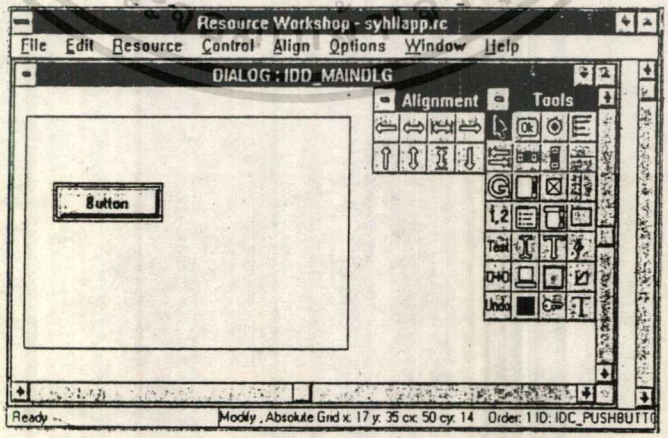
Resource Workshop จะตอบรับโดยตำแหน่งปุ่มควบคุมภายในเทมเพลทอยู่ตรงส่วนที่คลิ้กเมาส์ (รูป 2.32) ซึ่งสามารถย้ายปุ่มนี้ไปที่ใดก็ได้ภายในเทมเพลท

ตอนนี้ที่ปุ่มควบคุมเป็นชื่อ "Button" ให้เปลี่ยนเป็น "SayHello"

- ดับเบิ้ล-คลิ้ก ที่ปุ่มควบคุม

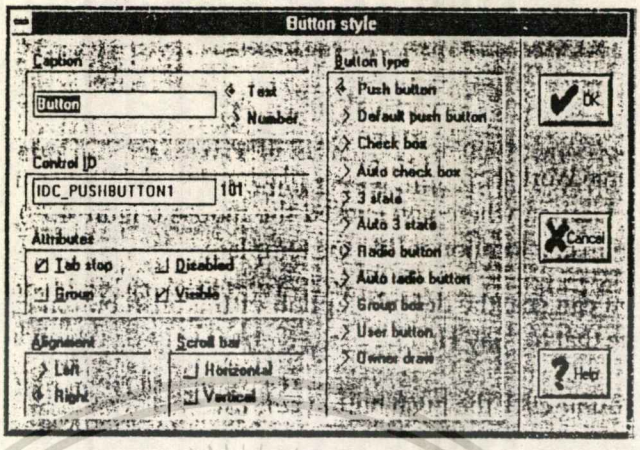
Resource Workshop จะตอบรับโดยแสดงกรอบข้อความ Button Style (รูป 2.33)

- พิมพ์ &Say Hello ในส่วน Caption






**รูป 2.32 กรอบข้อความของเทมเพลทกับปุ่มควบคุมภายใน**




**รูป 2.33 กรอบข้อความ Button Style**

**หมายเหตุ** ตัวอักขระ & ที่อยู่หน้า S ใน &Say Hello จะทำให้ตัวอักษร S มีการขีดเส้นใต้ปรากฏอยู่ใต้ตัว S เรียกว่า "hot key" เมื่อทำโปรแกรมเสร็จแล้วกด Alt-S ที่คีย์บอร์ดก็จะทำเหมือนกับการกดปุ่ม Say Hello

ในส่วนอื่นที่จำเป็นต้องเซทภายในกรอบข้อความ Button Styleคือ Control ID ดังรูป 4.30 ที่ ID ของ Control จะเป็น IDC_PUSH_BUTTON1 ซึ่งจะไม่สัมพันธ์กับปุ่มที่ทำไว้ จึงเปลี่ยน Control ID ของปุ่มดังนี้

- พิมพ์ IDC_BUTTON _HELLO ใน Control ID

กรอบข้อความ Button Style จะเป็นดังรูป 2.34

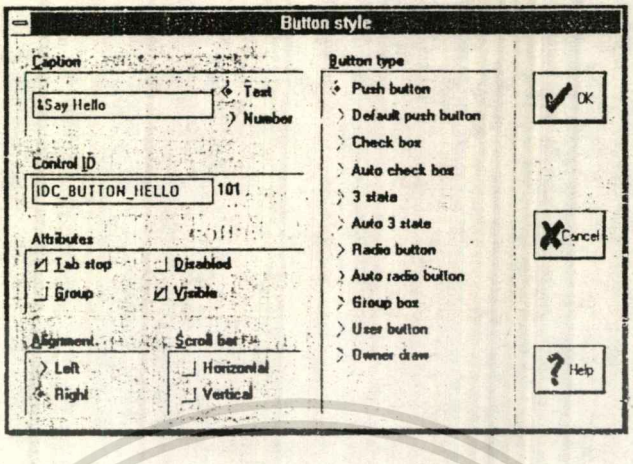
- คลิ้กปุ่ม OK ของกรอบข้อความ Button Style

เทมเพลทของกรอบข้อความ IDD_MAINDLG จะเป็นดังรูป 2.35 ชื่อของปุ่มจะเป็น "Say Hello" และที่ตัวอักษร S จะขีดเส้นใต้อยู่

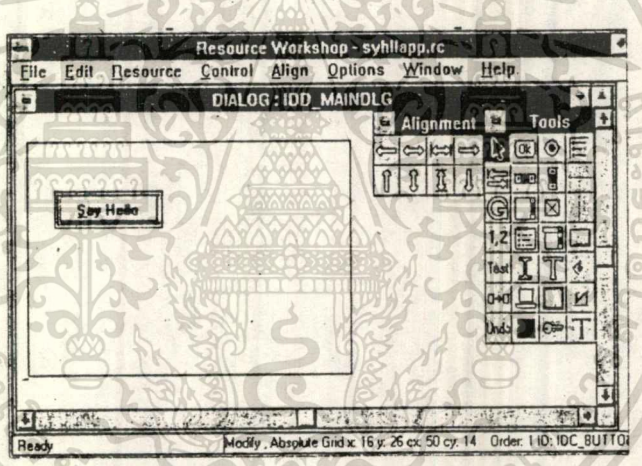
เมื่อได้ปุ่ม Say Hello และเซททที่ Control ID เรียบร้อยแล้ว แต่จะต้องเพิ่มอีก 2 ปุ่ม ควบคุมภายในเทมเพลท : ปุ่มอะไรก็ได้ (ปุ่ม Clear) และ edit box

ขั้นตอนต่อไปนี้เป็นการวางตำแหน่งปุ่ม Clear

- คลิ้กที่เครื่องมือ PUSH BUTTON ภายใน Tools วินโดวส์




**รูป 2.34 การเซท Caption และ Control ID ของปุ่ม Say Hello**




**รูป 2.35 ปุ่ม Say Hello ที่เป็นชื่อใหม่**

- คลิ้กเมาส์ที่เป็นเทมเพลทภายในกรอบข้อความและวางตำแหน่งไว้ทางขวาของปุ่ม Say Hello
- ดับเบิ้ล-คลิ้ก ที่ปุ่มใหม่

Resource Workshop จะตอบรับโดยแสดงกรอบข้อความ Button Style

- เซท Caption ของปุ่มเป็น &Clear
- เซท Control ID ของปุ่มเป็น IDC_BUTTON_CLEAR
- คลิ้ก OK

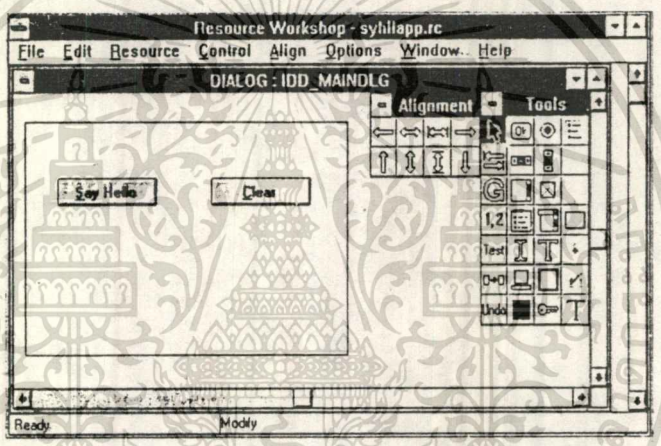
เทมเพลทของกรอบข้อความ IDD_MAINDLG จะเป็นดังรูป 2.36 ชื่อของปุ่มใหม่จะชื่อ “Clear” และที่ตัวอักษร C จะมีขีดเส้นใต้

จากนั้นควบคุมตำแหน่งภายใน กรอบข้อความ IDD_MAINDLG ให้เป็น edit box ทำได้ดังนี้ :

- คลิ้กที่ Edit Box ภายใน Tools วินโดวส์ดังรูป 2.37

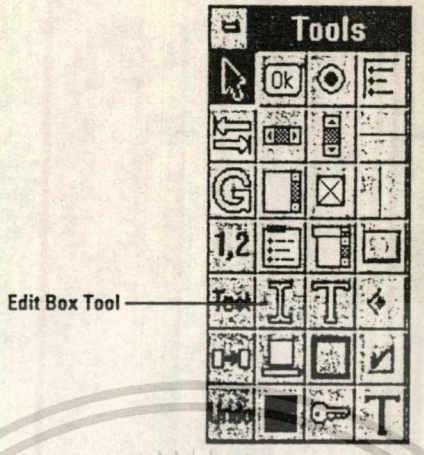
Resource Workshop จะตอบรับโดยเปลี่ยนเมาส์เคอร์เซอร์เป็นไอคอน Edit Box

- คลิ้กเมาส์ภายในเทมเพลทของกรอบข้อความ

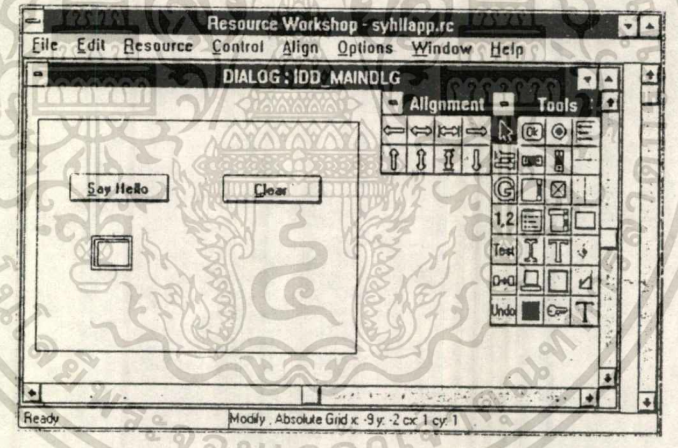



**รูป 2.36 กรอบข้อความ IDD_MAINDLG กับปุ่ม Clear**

Resource Workshop จะตอบรับโดยตำแหน่งภายในเทมเพลทตรงที่คลิ้กเมาส์ (รูป 2.38) สามารถเคลื่อนย้าย edit box ไว้ที่ตำแหน่งใดก็ได้ภายในเทมเพลท




**รูป 2.37 ภาพ Edit Box ภายใน Tools วินโดวส์**




**รูป 2.38 กรอบข้อความเทมเพลท กับ Edit Box ภายใน**

เซท Control ID ของ edit box ได้ดังนี้ :

- ดับเบิ้ล-คลิ้ก ที่ Edit Box control

Resource Workshop จะตอบรับโดยแสดงกรอบข้อความ Edit Text Style

- เซท Control ID เป็น IDC_MY_EDIT_BOX และคลิ้ก OK

ถ้าต้องการ edit box control ที่กว้างพอสำหรับข้อความที่จะปรากฏ (กลับไปดูในรูป 2.1) ทำได้โดยการเลื่อนเมาส์ เพื่อความละเอียดให้ดูที่ตัวเลขด้านล่างด้วย

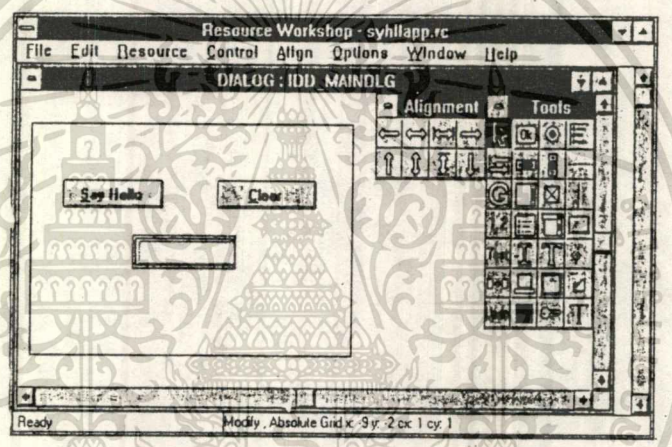
- คลิ้กปุ่มขวา Edit Box control และเลือกขนาดจาก popup เมนูที่แสดงอยู่

Resource Workshop จะตอบรับโดยแสดงกรอบข้อความ Size Control ซึ่งมี 4 ส่วน : X, CX , Y และ CY

- เปลี่ยนค่าของ CX เป็น :47 (CX บอกความกว้างของ control)

ความกว้างของ edit box control เท่ากับรูป 2.4

เมื่อออกแบบกรอบข้อความ IDD_MAINDLG แล้ว จากรูป 2.39 จะแสดงรูปแบบที่สมบูรณ์




**รูป 2.39 รูปแบบที่สมบูรณ์ของ กรอบข้อความ IDD_MAINDLG**

อย่าลืมจัดเก็บงานที่ทำไว้ :

- เลือก Save Project จากเมนู File ของ Resource Workshop

จาก Resource Workshop และกลับไปที่ Borland C++

- เลือก Exit จากเมนู File ของ Resource Workshop

### 2.4.7 การทำกรอบข้อความ IDD_MAINDLG วินโดวส์หลักของโปรแกรมประยุกต์

จากคลาสที่สร้างขึ้นชื่อ TmainDlg (มาจาก base class Tdialog) จะสัมพันธ์กับคลาสนี้มีกรอบข้อความ ชื่อ IDD_MAINDLG และตำแหน่งควบคุม (ทั้ง 2 ปุ่ม และ edit box) ภายในกรอบข้อความ IDD_MAINDLG ในส่วนนี้จะเขียนโค้ด ซึ่งจะทำกรอบข้อความ IDD_MAINDLG วินโดวส์หลักของโปรแกรมประยุกต์

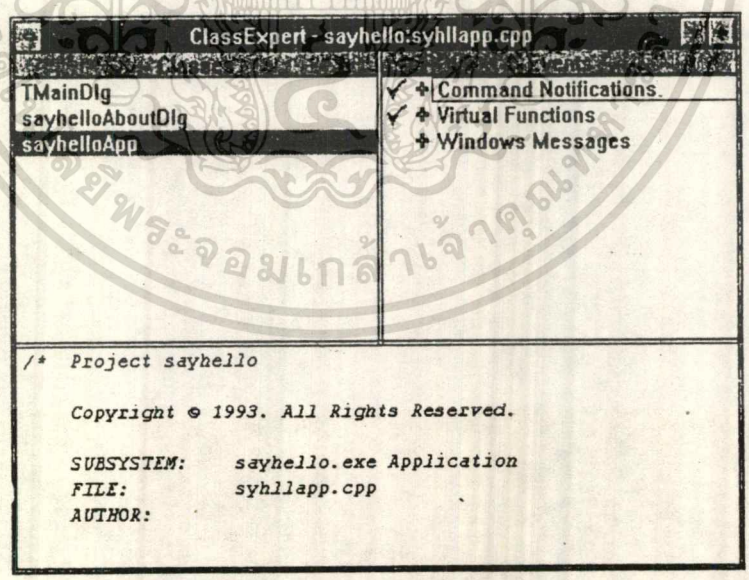
พิจารณาง่ายๆ Classes ที่ ClassExpert สร้างขึ้นชื่อ sayhelloApp (Application class) ฟังก์ชั่นของคลาสนี้ชื่อ InitMainWindow( ) ชื่อนี้หมายความว่า InitMainWindow( ) จะตอบรับสำหรับวินโดวส์หลักตั้งแต่แรกของโปรแกรมประยุกต์

ในขั้นตอนต่อไปนี้เป็นการเขียนโค้ดภายในฟังก์ชั่น InitMainWindow( ) ซึ่งแสดงกรอบข้อความ และสร้าง (IDD_MAINDLG) เหมือนกับวินโดวส์หลักของโปรแกรมประยุกต์

ขั้นตอนการเขียนโค้ดภายในฟังก์ชั่น InitMainWindow( ) ของ sayhelloApp class

- เลือกคลาส sayhelloApp ภายในส่วน Classes ของ ClassExpert (คลิ้กบน sayhelloApp)

ตอนนี้ ClassExpert จะเป็นดังรูป 2.40




**รูป 2.40 ClassExpert วินโดวส์ หลังจากเลือกคลาส sayhelloApp ในส่วน Classes**

ส่วน Event ของ ClassExpert แบ่งเป็น 3 รายการ

1. Command Notifications

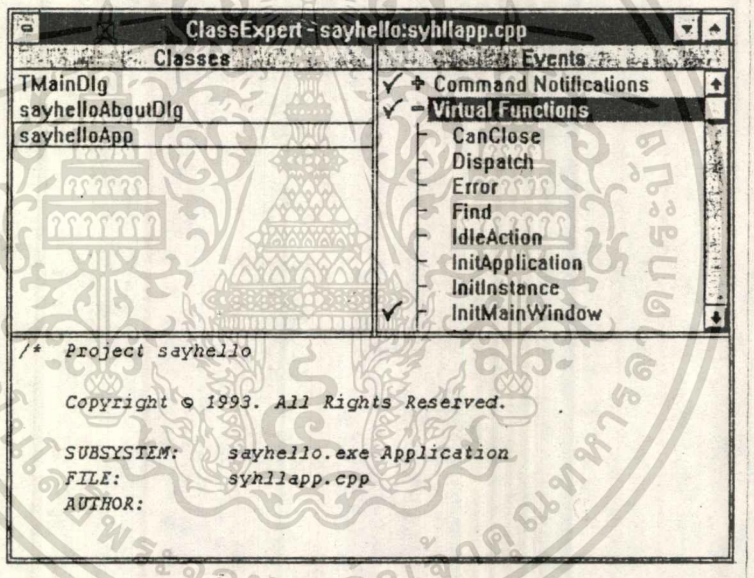
2. Virtual Functiona

3. Windows Messages

InitMainWindow( ) เป็นฟังก์ชั่นทั่วไป แล้วดูที่รายการ Virtual Functions

- คลิ้ก + ด้านหน้ารายการ Virtual Functions ในส่วน Events ของ ClassExpert

Borland C++ จะตอบรับโดยเปิดรายการของฟังก์ชั่นทั่วไปทั้งหมดของคลาส sayhelloApp (รูป 2.41)

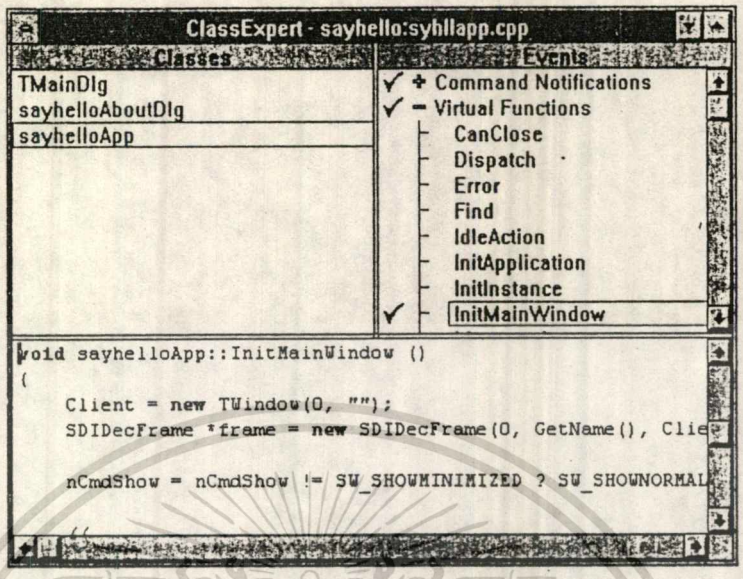



**รูป 2.41 รายการฟังก์ชั่นทั่วไปของคลาส sayhelloApp ในส่วน Events ของ ClassExpert**

สังเกตุที่รายการ InitMainWindow จะมีเช็คเครื่องหมายด้านหน้า หมายความว่าพร้อมจะเขียนโค้ดภายในฟังก์ชั่น InitMainWindow( )

- ดับเบิ้ล-คลิ้ก ที่รายการ InitMainWindow ภายในส่วน Events ของ ClassExpert

Borland C++ จะตอบรับโดยแสดงโค้ดของฟังก์ชั่น InitMainWindow( ) ภายในส่วนโค้ดของ ClassExpert (รูป 2.42) สามารถแก้ไขฟังก์ชั่น InitMainWindow ได้




**รูป 2.42 การแก้ไขฟังก์ชั่น InitMainWindow( )**

แก้ไขฟังก์ชั่น InitMainWindow( ) ดังต่อไปนี้

- ลบทุกโค้ดภายในฟังก์ชั่น InitMainWindow( ) และแทนที่ด้วย “custom” โค้ดดังนี้

```
void sayhelloApp :: InitMainWindow( )
{
//////////////////////////////
//CUSTOM CODE STARTS HERE
//////////////////////////////
//The main window of the application will be
//the dialog box IDD_MAINDLG.
Client = new TMainDlg (0, IDD_MAINDLG);
TFrameWindow  *frame = new TframeWindow (0, GetName( ), Client, TURE);
//Assign icon.
frame -> SetIcon(this, IDI_SDIAPPLACATION);
//Assign menu
frame -> AssignMenu(SDI_MENU);
```

```
//The accelerator table.
frame -> Attr. AccelTable = SDI_MENU;
//Set the Main window.
MainWindow = frame;
//Enable Borland controls.
EnableBWCC( );
//////////////////////////
//CUSTOM CODE ENDS HERE
//////////////////////////
}
```

อย่าลืมจัดเก็บงานที่ทำไว้

- เลือก Save จากเมนู File ของ Borland C++

โค้ดที่พิมพ์ภายในฟังก์ชั่น InitMainWindow( ) เซทกรอบข้อความที่ออกแบบ IDD_MAINDLG เหมือนวินโดวส์หลักโปรแกรมประยุกต์ ซึ่งจะมีรายละเอียดเหมือนในขั้นตอนย่อย สำหรับโปรแกรมประยุกต์ทั้งหมดจะใช้กรอบข้อความเหมือนวินโดวส์หลัก

### 2.4.8 Custom Code และ Borland Code

ในขั้นตอนนี้คือโครงสร้างที่จะเขียน "CUSTOM" โค้ดในฟังก์ชั่น InitMainWindow() โดยการแบ่งระหว่างโค้ดที่ Borland C++ เขียนไว้ และโค้ดที่พิมพ์ตามที่แนะนำไว้ตามขั้นตอน ซึ่งที่พิมพ์ไปนั้นจะล้อมรอบด้วย 2 ส่วนของ boldface

```
//////////////////////////
//CUSTOM CODE STARTS HERE
//////////////////////////
.....
.....โค้ดที่พิมพ์ขึ้น.....
.....
```

```
//////////////////////////
//CUSTOM CODE ENDS HERE
//////////////////////////
```

โค้ดใดที่ไม่ปรากฏระหว่างส่วน CUSTOM CODE STARTS HERE / CUSTOM CODE ENDS HERE คือโค้ดที่เขียนขึ้นโดย Borland C++

การใช้กรอบข้อความเหมือนวินโดวส์หลัก จึงมีประโยชน์มากในโปรแกรมประยุกต์มัลติมิเดีย เพราะมันจะควบคุมในวินโดวส์หลัก เมื่อจะใช้กรอบข้อความเหมือนวินโดวส์หลักของโปรแกรมประยุกต์, ตำแหน่งควบคุม (ควบคุมมัลติมิเดีย, ปุ่มกด, การเลื่อนบาร์ และอื่นๆ) ภายในวินโดวส์หลักก็จะง่ายมาก

### 2.8.9 #include ที่ส่วนบนสุดของไฟล์

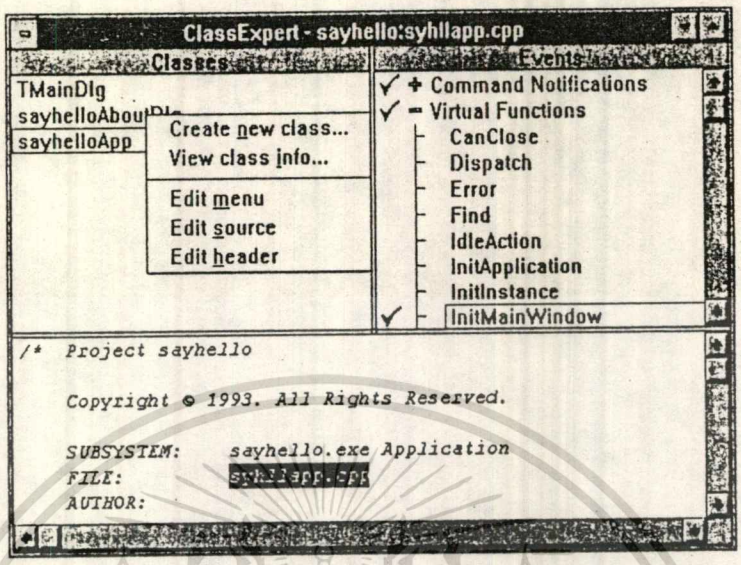
ถ้าคอมไพล์ / ลิงค์โปรแกรมประยุกต์ SayHello แล้วเกิดความผิดพลาดเป็นเพราะในโค้ดที่ใช้อยู่เป็นคลาส TMainDlg แต่คลาส TMainDlg จะไม่เป็นที่รู้จักในไฟล์ syhllapp.cpp (ไฟล์ของ sayhelloApp class) ดังนั้นจึงต้องเพิ่ม #include ไฟล์ tmaindlg.h ที่ส่วนบนสุดของไฟล์ sayhelloApp class ทำได้ดังนี้

- คลิ้กปุ่มขวาที่เมาส์บน sayhelloApp ภายในส่วน Classes ของ ClassExper

Borland C++ จะตอบรับโดยแสดง popup เมนู (รูป 2.43)

- เลือก Edit Header จาก popup เมนู

Borland C++ จะตอบรับโดยเปิด header ไฟล์ของ sayhelloApp class, syhllapp.h




**รูป 2.43 popup เมนูที่ ClassExpert แสดงหลังจากคลิ๊กปุ่มขวาที่เมาส์บนคลาส sayhelloApp**

- กำหนด #include “tmaindlg.h” เพิ่มที่ส่วนเริ่มต้นของไฟล์ syhllapp.h ส่วนเริ่มต้นไฟล์ดูได้ดังนี้

```
#include < owl \ owlpch.h >
#pragma hdrstop
#include “syhllapp.rh” //Definition of all resources.
//////////////////////////////
//CUSTOM CODE STARTS HERE
//////////////////////////////
#include “tmaindlg.h”
//////////////////////////
//CUSTOM CODE ENDS HERE
//////////////////////////
```

- จัดเก็บงานโดยเลือก Save จากเมนู File
- Close ไฟล์ syhllapp.h header โดยดับเบิ้ล-คลิ้ก ที่ไอคอนเมนูของวินโดวส์นั้น

เมื่อเขียนโค้ดเสร็จแล้วทำกรอบข้อความ IDD_MAINDLG วินโดวส์หลักของโปรแกรมประยุกต์ ซึ่งจะเห็นโค้ดในส่วนนี้ :

- เลือก Run จากเมนู Debug ของ Borland C++

Borland C++ จะตอบรับโดยคอมไพล์ / ลิงค์ไฟล์ของโปรแกรมประยุกต์ SayHello และจะได้ผลลัพธ์โปรแกรม SayHello.EXE วินโดวส์หลักของ SayHello.EXE จะปรากฏดังรูป 2.44

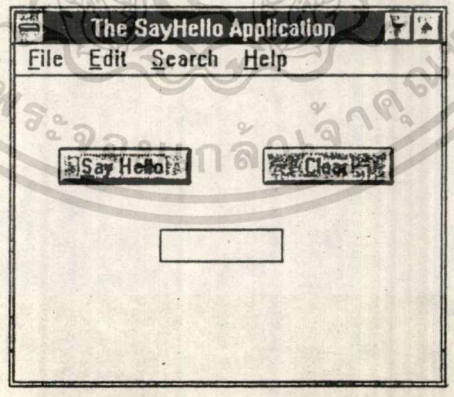
กรอบข้อความ IDD_MAINDLG ที่ออกแบบจะเหมือนวินโดวส์หลักของโปรแกรมประยุกต์

- ทดลองคลิ้กปุ่ม SayHello และ Clear

เมื่อคลิ้กที่ปุ่มแล้วจะไม่มีอะไรเกิดขึ้น เพราะยังไม่ได้รวมโค้ดเข้าด้วยกัน

- คลิ้กภายใน edit box และทดลองพิมพ์อะไรก็ได้

ในโปรแกรมประยุกต์ SayHello นั้นจะไม่สามารถพิมพ์ลงใน edit box ได้ ดังนั้น edit box จึงยังใช้ไม่ได้ ในส่วนต่อไปนี้จะสามารถควบคุม edit box ได้




**รูป 2.44 วินโดวส์หลักของ SayHello.EXE**

- ออกจากโปรแกรมประยุกต์ SayHello โดยเลือก Exit จากเมนู File

### 2.8.10 การควบคุมข้อเสียของ edit box

การควบคุมข้อเสียของ edit box คือ ผู้ใช้จะไม่สามารถพิมพ์อะไรเข้าไปภายในได้เลย ต้องมีการแก้ไข IDD_MAINDLG dialog ด้วย Resource Workshop ซึ่งทำได้ดังนี้

- คลิ้กปุ่มขวาบน Tmaindlg ภายในส่วน Class ของ ClassExpert และส่วน Edit Dialog จาก popup เมนู

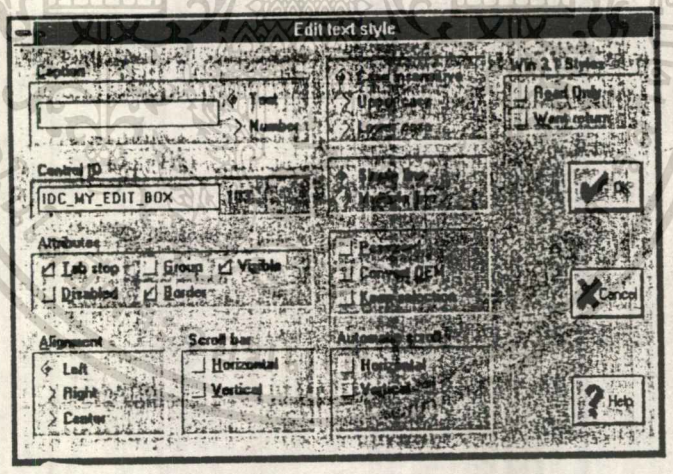
Borland C++ จะตอบรับโดยการรับ Resoure Workshop ที่แก้ไขกรอบข้อความ IDD_MAINDLG

- ดับเบิ้ล-คลิ้ก Edit Box control

Resource Workshop จะตอบรับโดยแสดงกรอบข้อความ Edit Text Style (รูป 2.45)

- ตำแหน่งเครื่องหมายภายใน Disable check box ( ภายในส่วน Attributes ) และคลิ้ก OK

จากนั้น Edit Box control ก็จะใช้งานได้




**รูป 2.45 กรอบข้อความ Edit Text Style**

พิสูจน์ว่า edit box ใช้การได้หรือไม่โดย :

- เลือก Save Project จากเมนู File ของ Resource Workshop
- เลือก Exit จากเมนู File ของ Resource Workshop
- เลือก RUN จากเมนู Debug ของ Borland C++

วินโดวส์หลักของโปรแกรมประยุกต์ SayHello จะปรากฏดังนี้

- ลองพิมพ์อะไรก็ได้ภายใน Edit Box

edit box จะไม่ปรากฏแก้ไขได้

- ออกจากโปรแกรมประยุกต์ SayHello โดยเลือก Exit จากเมนู File

### 2.8.11 การรวมโค้ดกับปุ่มกด Say Hello

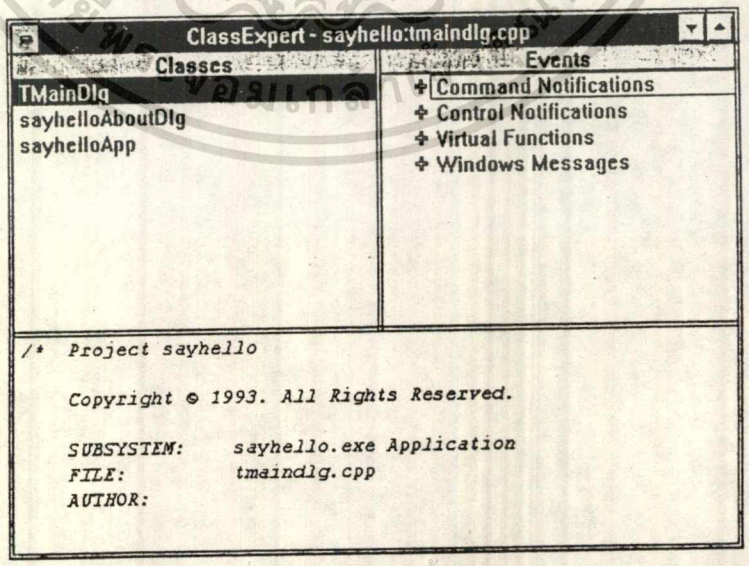
เรียกเมื่อผู้ใช้คลิ้กปุ่ม Say Hello จะต้องรวมตัวแปรที่ edit box code ที่ปุ่ม Say Hello จะใช้แสดงข้อความภายใน edit box

ขั้นตอนต่อไปนี้เป็นการรวมตัวแปรกับ editbox

- คลิ้กที่ TMainDlg ภายในส่วน Classes ของ ClassExpert

ClassExpert วินโดวส์ที่แสดงดังรูป 2.46 ส่วน Events ของ ClassExpert จะแบ่งเป็น 4 รายการดังนี้

1. Command Notification
2. Control Notification
3. Virtual Function
4. Windows Messages






**รูป 2.46 ClassExpert วินโดวส์หลังจากเลือก TMainDlg class ในส่วน Classes**

ทางขวามือนั้นจะต้องรวมตัวแปรกับ IDC_MY_EDIT_BOX control ซึ่งรายการอยู่ใต้ Control Notiication

- คลิก + หน้า Control Notification ในส่วน Event

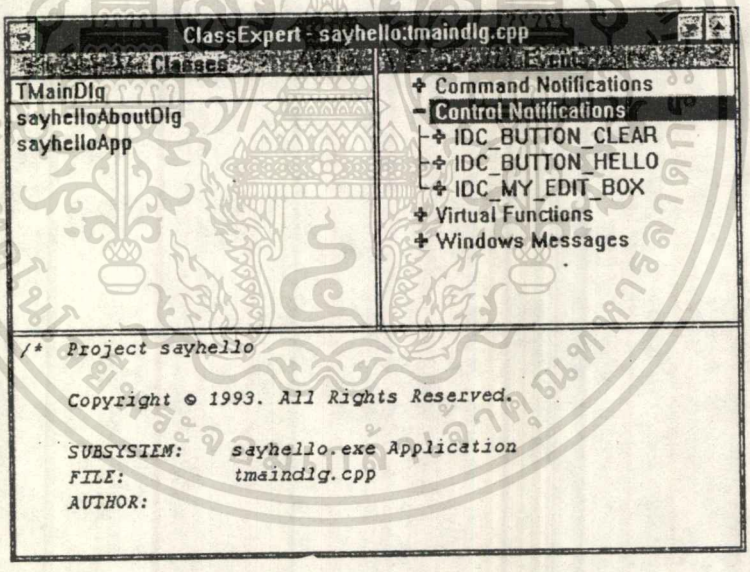
Borland C++ จะตอบรับโดยเปิดรายการของทุก control จากคลาส TMainDlg ( รูป 2.47 )

การรวมตัวแปรกับ IDC_MY_EDIT_BOX control :

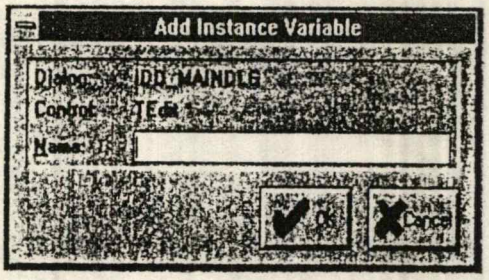
- คลิ้กปุ่มขวาตรงรายการ IDC_MY_EDIT_BOX และเลือก Add Instance Variable จาก Popup เมนู และจะปรากฏกรอบข้อความ

Borland C++ จะตอบรับโดยแสดงกรอบข้อความ Add Instance Variable (รูป 2.48)

- พิมพ์ m_MyEditBox ในช่อง Name และคลิ้ก OK




**รูป 2.47 รายการ control ของคลาสในส่วน Event ของ ClassExpert**




**รูป 2.48 กรอบข้อความ Add Instance Variable**

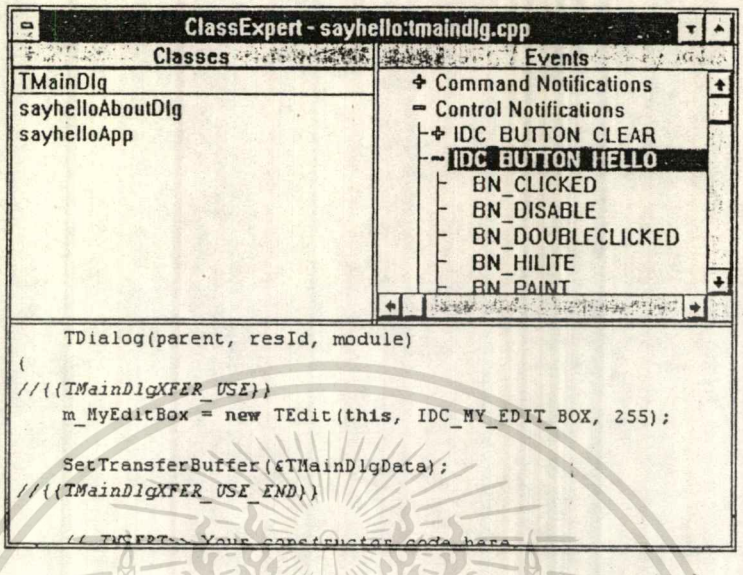
**หมายเหตุ** ในขั้นตอนนี้จะใช้ CassExpert ในการรวมตัวแปร m_MyEditBox กับ IDC_MY_EDIT_BOX control ClassExpert จะประกาศตัวแปร m_MyEditBox เหมือนเป็นข้อมูลของคลาส TMainDlg การนำหน้าด้วย "m_" แสดงว่า m_MyEditBox เป็นเหมือนข้อมูล (ไม่ใช่ตัวแปร "regular") รูปแบบข้อมูลของตัวแปรเป็น TEdit* (พอยน์เตอร์ที่เป็นที่หมายของ TEdit) TEdit เป็นประโยชน์ในคลาส OWL จะรวมฟังก์ชั่นที่จะใช้งานดังที่ใช้ใน edit box ดังนั้นการใช้ฟังก์ชั่นของคลาส Tedit กับ IDC_MY_BOX edit box

ตอนนี้ IDC_MY_EDIT_BOX control จะมีตัวแปรที่สามารถรวมโค้ดกับปุ่ม Say Hello และ Clear ซึ่งจะเริ่มที่ปุ่ม Say Hello

- คลิ๊ก + ด้านหน้า IDC_BUTTON_HELLO ในส่วน Events

จากรูป 2.49 Borland C++ จะตอบรับโดยเปิดรายการของ Events ทั้งหมดที่เกี่ยวข้องกับปุ่ม IDC_BUTTON_HELLO

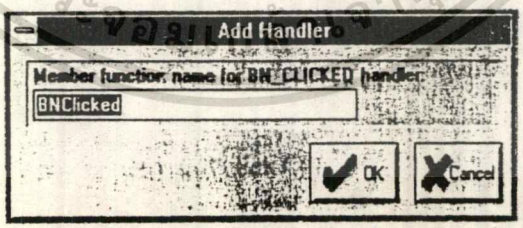
ส่วนด้านขวามือ จะเป็นการรวมโค้ดเท่านั้น ให้ Click events ของปุ่ม IDC_BUTTON_HELLO ซึ่งทำให้การรวมโค้ดกับฟังก์ชั่นนั้นเสร็จสมบูรณ์ โดยคลิ๊กปุ่ม Say Hello




**รูป 2.49 รายการ Events ซึ่งเกี่ยวข้องกับปุ่ม IDC_BUTTON_HELLO**

ดังนั้นถ้าต้องการเพิ่มฟังก์ชั่นสำหรับ BN_CLICKED ให้ทำดังนี้

- คลิ้กปุ่มขวาที่รายการ BN_CLICKED (ด้านใต้รายการ IDC_BUTTON_HELLO) และเลือก Add Handler จากเมนูที่ popup (ดังแสดงในรูป 2.50)
- พิมพ์ OnSayHello ในช่อง Function Name และคลิ้ก OK




**รูป 2.50 กรอบข้อความ Add Handler**

Borland C++ จะตอบรับโดยเพิ่มฟังก์ชั่น OnSayHello( ) ดูที่รายการ BN_CLICKED ของปุ่ม IDC_BUTTON_HELLO จะมีการเช็คเครื่องหมาย ซึ่งหมายความว่า BN_CLICKED events ของปุ่ม IDC_BUTTON_HELLO มีฟังก์ชั่นรวมอยู่ด้วย

การเขียนโค้ดในฟังก์ชั่น OnSayHello( )

- ดับเบิ้ล-คลิ้ก ที่ BN_CLICKED ของปุ่ม IDC_BUTTON_HELLO
- เขียนโค้ดภายในฟังก์ชั่น OnSayHello( ) ดังนี้

```
void TMainDlg :: OnSayHello( )
{
/ /INSERT >> Your code here
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ /CUSTOM CODE STARTS HERE
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
m_MyEditBox -> SetText (" H E L L O ");
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ /CUSTOM CODE ENDS HERE
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / /
}
```

โค้ดที่พิมพ์ภายในฟังก์ชั่น OnSayHello( ) จะมีการกำหนด 1 ส่วน m_MyEditBox -> SetText (" H E L L O "); โดยจะใช้กับฟังก์ชั่น SetText( ) ของคลาส TEdit กับ m_MyEditBox edit box (ซึ่ง IDC_MY_EDIT_BOX control) ด้วยสตริง " H E L L O " ( ข้อความของสตริงจะคั่นด้วยช่องว่าง เพื่อความสวยงาม )

- เลือก Save จากเมนู File

เมื่อทำโปรแกรม SayHello เสร็จ สามารถดูโค้ดได้โดย

- เลือก Run จากเมนู Debug ของ Borland C++

Borland C++ จะทำการคอมไพล์ / ลิงค์ ไฟล์ของโปรแกรมประยุกต์ SayHello และกระทำผลลัพธ์เป็นไฟล์ SayHello.EXE

- คลิ้กปุ่ ม Say Hello

จากนั้นโปรแกรมประยุกต์ SayHello จะแสดงข้อความ "HELLO" ภายใน edit box

- ออกจากโปรแกรมประยุกต์ SayHello โดยเลือก Exit จากเมนู File

### 2.4.12 การรวมโค้ดกับปุ่ม Clear

การรวมโค้ดกับปุ่ม Clear ทำดังนี้

- ที่คลาส TMain เลือกในส่วน Classes ของ ClassExpert
- คลิ้ก + หน้า IDC_BUTTON_CLEAR ในส่วน Events ของ ClassExpert

Borland C++ จะรายงาน events ที่เกี่ยวข้องกับ IDC_BUTTON_CLEAR

- คลิ้กปุ่มขวาที่รายการ BN_CLICKED และเลือก Add Handler จากเมนูที่ popup

Borland C++ จะแสดงกรอบข้อความ Add Handler

- พิมพ์ OnClear ภายในช่อง Function Name และคลิ้ก OK

Borland C++ จะตอบรับโดยเพิ่มฟังก์ชั่น OnClear( ) ดูที่รายการ BN_CLICKED จะมีเครื่องหมายเช็คอยู่แสดงว่าในฟังก์ชั่นทำการรวมกันแล้ว

แก้ไขฟังก์ชั่น OnClear ดังนี้

```
void TMainDlg :: OnClear( )
{
/ /INSERT >> Your code here
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ /CUSTOM CODE STARTS HERE
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / /
m_MyEditBox -> SetText ("  ");
/ / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ /CUSTOM CODE ENDS HERE
/ / / / / / / / / / / / / / / / / / / / / / / / / / / /
}
```

การจัดเก็บงาน

- เลือก Save จากเมนู File
- เลือก Run จากเมนู Debug ของ Borland C++

Borland C++ จะทำการคอมไพล์ / ลิงค์ ไฟล์จากโปรแกรมประยุกต์ SayHello และกระทำผลลัพธ์ไฟล์ SayHello.EXE

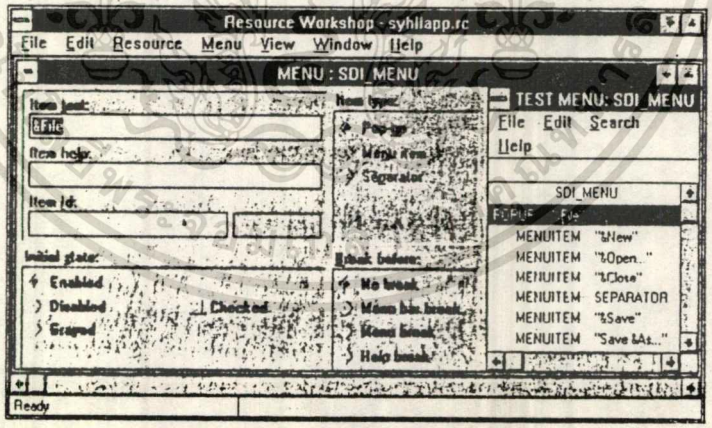
- ทำการทดลองโดยคลิ้กปุ่ม Say Hello จะมีข้อความ " H E L L O " แสดงภายใน edit box และคลิ้กปุ่ม Clear โปรแกรมก็จะทำการเคลียร์ข้อความใน Edit Box
- ออกจากโปรแกรมประยุกต์ Say Hello โดยเลือก Exit จากเมนู File

**2.4.13 เครื่องมือ Menu Bar ของโปรแกรมประยุกต์ SayHello**

เครื่องมือและโค้ดจะเขียนเพื่อกรอบข้อความ หลักของโปรแกรมประยุกต์ ตอนนี้เครื่องมือโปรแกรมประยุกต์ของเมนูหลัก ดูตามขั้นตอนดังนี้

- คลิ้กปุ่มขวาที่รายการ TMainDlg ภายในส่วน Classes ของ ClassExpert และเลือก Edit Menu จากเมนูที่ popup

Resource Workshop จะปรากฏ ซึ่งเราสามารถแก้ไขเมนูของโปรแกรมประยุกต์ SayHello วินโดวส์หลัก ของ Resource Workshop ดังรูป 2.51




**รูป 2.51 Resource Workshop กับเมนูของโปรแกรมประยุกต์ SayHello พร้อมสำหรับแก้ไข**

Resource Workshop จะปรากฏวินโดวส์ชื่อ MENU : SDI_MENU หมายความว่า เราจะใช้ Resource Workshop แก้ไขเมนูชื่อ SDI_MENU เมนูนี้จะสร้างขึ้นโดย AppExpert เมื่อสร้างโครงสร้างไฟล์ของโปรแกรมประยุกต์ ซึ่งมันจะไปทำที่เมนู SDI_MENU ดูได้จากรูป 2.6 และ 2.7

MENU : SDI_MENU ประกอบด้วยวินโดวส์อื่นชื่อ TEST MENU : SDI_MENU

ด้านล่างของวินโดวส์ TEST MENU จะแสดงรายการ popup เมนูทั้งหมด และเมนูรายการ SDI_MENU ดังตัวอย่างของ File popup จะมีรายชื่อดังนี้

```
POPUP "&File"
        MENUITEM    "&New"
        MENUITEM    "&Open"
        MENUITEM    "&Close"
        MENUITEM    SEPARATOR
        MENUITEM    "&Save"
        MENUITEM    "Save &As"
        MENUITEM    SEPARATOR
        MENUITEM    "E&xit \ Ail +F4"
```

ทำการเปลี่ยน File เมนู popup ดูจากรูป 2.6 ซึ่งจะดูเมนูที่ไม่ต้องการ

- ทำแถบแสงที่รายการ MENUITEM "&New" และกดคีย์ Delete ที่คีย์บอร์ด

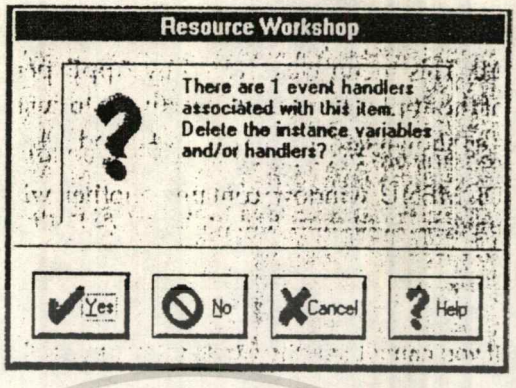
Resource Workshop จะตอบรับโดยแสดงกรอบข้อความคำเตือน (รูป 2.52) บอกยืนยันการลบเมนูรายการ

- คลิ้กปุ่ม Yes เพื่อยืนยันการลบเมนูรายการ
- ลบทุกเมนูรายการของ File popup ยกเว้นรายการ Exit

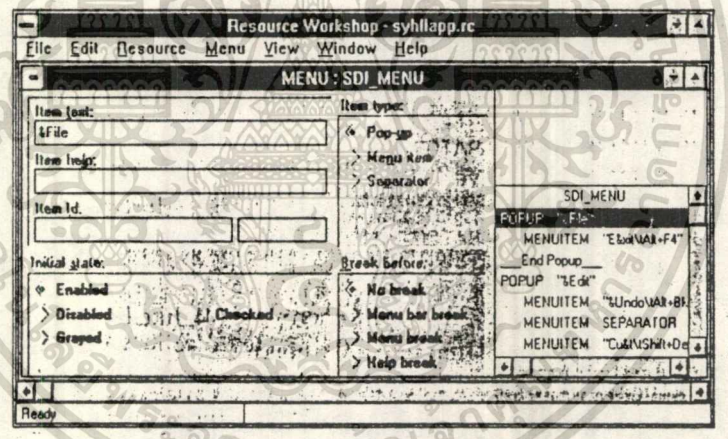
วินโดวส์ TEST MENU ดูจากรูป 2.52

จากนั้นทำการเพิ่มเมนูรายการ Say Hello ใน File popup ดังนี้

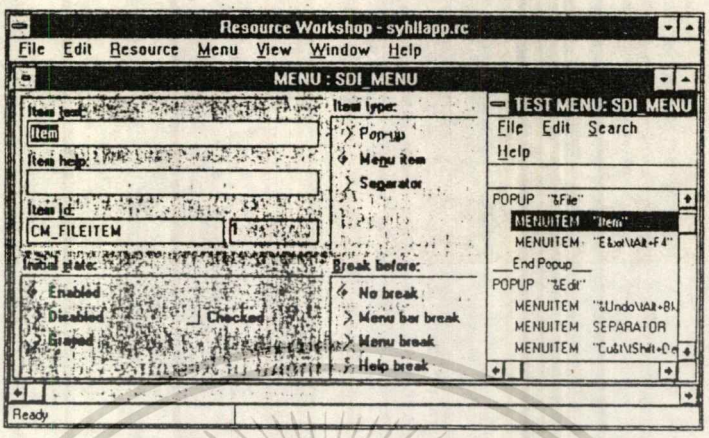
- ทำแถบแสงที่รายการ POPUP "&File" (เพราะต้องแทรกรายการด้านล่างรายการนี้) และกดคีย์ Insert (ดังรูป 2.54)




**รูป 2.52 คำเตือนที่ Resource Workshop แสดงหลังจากลบเมนูรายการ MENUITEM "&New"**




**รูป 2.53 วินโดวส์ TEST MENU หลังจากลบทุกเมนูรายการของ File popup ออก ยกเว้น Exit**




**รูป 2.54 การเพิ่มเมนูรายการใหม่ที่ File popup**

จากรูปหัวข้อของเมนูรายการใหม่เป็น "Item" แก้ไขที่หัวข้อนี้ ดังนี้

- พิมพ์ &Say Hello ภายในช่อง Item Text ที่ช่อง Item Id จะเปลี่ยนโดยอัตโนมัติกับ ID CM_FILESAY_HELLO

เพิ่มเมนูรายการ Clear

- กดคีย์ Insert อีกครั้ง
- พิมพ์ &Clear ในช่อง Item Text

ทำการลบทั้งสองเมนู popup Edit และ Search ดังนี้

- ทำแถบแสงที่รายการ POPUP "Edit" และ "&Search" จากนั้นกดคีย์ Delete

Resource Workshop จะตอบรับโดยลบ Edit popup เมนู และ Search popup เมนู

- เลือก Save Project จากเมนู File ของ Resource Workshop
- เลือก Exit จากเมนู File ของ Resource Workshop เพื่อกลับไปที่ Borland C++
- เลือก Run จากเมนู Debug ของ Borland C++

Borland C++ จะทำการคอมไพล์ / ลิงค์ ไฟล์ของโปรแกรมประยุกต์ SayHello และกระทำผลลัพธ์เป็นไฟล์ SayHello.EXE

- ทำการทดลองกับเมนูของโปรแกรมประยุกต์ SayHello โดยดูจากรูป 2.6 และ 2.7

**2.4.14 การรวมโค้ดกับเมนูรายการ SayHello และ Clear**

ขั้นตอนต่อไปนี้เป็นการรวมโค้ดกับเมนูรายการ SayHello :

- เลือกคลาส TMainDlg ภายในส่วน Classes ของ ClassExpert (โดยคลิ้กบน TMainDlg)
- คลิ้ก + หน้ารายการ Command Notification ในส่วน Events

Borland C++ จะตอบรับโดยการเปิดรายการ IDs ของทุกเมนูรายการ

- คลิ้ก + หน้ารายการ CM_FILESAY_HELLO

Borland C++ จะรายงาน events ที่เกี่ยวข้องกับเมนูรายการ IDC_FILESAY_HELLO events นั่นคือ Command และ Command Enable

Command events เกิดขึ้นเมื่อผู้ใช้เลือกเมนูรายการ Say Hello มันจะทำการรวมโค้ด

- คลิ้กปุ่มขวาที่รายการ Command และเรียก Add Handler จากเมนู popup
- พิมพ์ OnFileSayHello ในช่อง Fucntion Name และคลิ้ก OK

Borland C++ จะทำการเพิ่มฟังก์ชั่น OnFileSayHello( ) บันทึกรายการ Command ของเมนูรายการ IDC_FILESAY_HELLO จะมีเครื่องหมายเช็คอยู่แสดงว่า events ฟังก์ชั่นทำการรวมโค้ดแล้ว

ตอนนี้เราสามารถเขียนโค้ดภายในฟังก์ชั่น OnFileSayHello( ) :

- ดับเบิ้ล-คลิ้ก ที่รายการ Command ของเมนูรายการ IDC_FILESAY_HELLO
- เขียนโค้ดภายในฟังก์ชั่น OnFileSayHello( ) ได้ดังนี้ :

```
void TMainDlg :: OnFileSayHello( )
{
/ /INSERT >> Your code here
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ /CUSTOM CODE STARTS HERE
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
m_MyEditBox -> SetText (" H E L L O ");
```

```
//////////////////////////
//CUSTOM CODE ENDS HERE
//////////////////////////
}
```

โค้ดนี้จะแสดงโค้ดที่เขียนไว้เอง ภายในฟังก์ชั่น OnSayHello( ) มันจะแสดงสตริง " H E L L O" ภายใน กรอบข้อความ

ขั้นตอนสุดท้ายที่ต้องทำคือการรวมโค้ดกับเมนูรายการ Clear :

- คลิ้ก + หน้ารายการ CM_FILECLEAR ภายในส่วน evnts ของ ClassExpert

Borland C++ จะรายงาน events ที่เกี่ยวข้องกับเมนูรายการ IDC_FILECLEAR (Command และ Command Enable)

- คลิ้กปุ่มขวาที่รายการ Command และเรียก Add Handler ภายในส่วน Events ของ ClassExpert

Borland C++ จะตอบรับ โดยแสดงกรอบข้อความ Add Handler

- พิมพ์ OnFileClear ในช่อง Fucntion Name
- คลิ้กปุ่ม OK

Borland C++ จะทำการเพิ่มฟังก์ชั่น OnFileClear( ) บันทึกรายการ Command ของเมนูรายการ IDC_FILECLEAR จะมีเครื่องหมายเช็คอยู่แสดงว่า events ฟังก์ชั่นทำการรวมโค้ดแล้ว

ตอนนี้เราสามารถเขียนโค้ดภายในฟังก์ชั่น OnFileClear( ) :

- ดับเบิ้ล-คลิ้ก ที่รายการ Command ของเมนูรายการ IDC_FILECLEAR
- เขียนโค้ดภายในฟังก์ชั่น OnFileClear( ) ได้ดังนี้

```
void TMainDlg :: OnFileClear( )
{
//INSERT >> Your code here
//////////////////////////
//CUSTOM CODE STARTS HERE
//////////////////////////
m_MyEditBox -> SetText ("  ");
```

```
//////////////////////////
//CUSTOM CODE ENDS HERE
//////////////////////////
}
```

- ทำการจัดเก็บงาน โดยเลือก Save จากเมนู File

ดูผลลัพธ์ของโปรแกรมประยุกต์ SayHello โดย :

- เลือก Run จากเมนู Debug ของ Borland C++

Borland C++ จะทำการคอมไพล์ / ลิงค์ไฟล์จากโปรแกรมประยุกต์ SayHello และกระทำผลลัพธ์ไฟล์ SayHello.EXE

- ทำการทดลองเมนูรายการ Say Hello และ Clear ของเมนู File

การทดลองเมนูรายการ Say Hello และ Clear ของเมนู File จะเหมือนฟังก์ชั่นปุ่ม Say Hello และ Clear

- ออกจากโปรแกรมประยุกต์ Say Hello โดยเลือก Exit จากเมนู File

**2.4.15 สรุป**

ในหัวข้อนี้เป็นการเริ่มต้นเขียนโปรแกรมประยุกต์ Borland C++ วินโดวส์ ทำให้รู้จักการสร้างกรอบข้อความ , ตำแหน่งควบคุมภายในกรอบข้อความ , การทำกรอบข้อความวินโดวส์หลักของโปรแกรมประยุกต์ , การรวมโค้ดกับการควบคุมกรอบข้อความ , เครื่องมือเมนูทั่วไปของโปรแกรมประยุกต์ และการรวมโค้ดกับเมนูรายการ

# บทที่ 3

# การออกแบบ

## 3.1 รูปแบบของแอพพลิเคชั่น

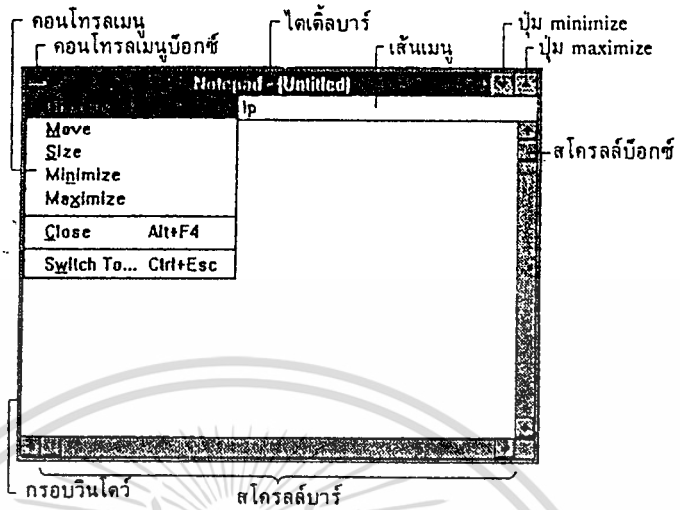
จุดประสงค์หลักของวินโดวส์ คือสร้างระบบติดต่อกับผู้ใช้แบบใหม่ที่จะใช้ได้ง่ายกว่าเดิม แอพพลิเคชั่นจะติดต่อกับผู้ใช้ โดยผ่านระบบใหม่เรียกได้ว่าเป็นเครื่องมือแบบใหม่อันประกอบไปด้วย

- วินโดวส์
- เมนู
- กรอบข้อความ (dialog box)
- เมสเสจ (message)

### 3.1.1 วินโดวส์

วินโดวส์เป็นสิ่งที่ติดต่อกับผู้ใช้อย่างพื้นฐานที่ทุกแอพพลิเคชั่นจะต้องมีใช้เสมอ วินโดวส์จะประกอบไปด้วยไตเติ้ลบาร์ , เมนูบาร์ , สโครล์บาร์ , กรอบข้อความ ฯลฯ เมื่อเราต้องสร้างวินโดวส์ก็เพียงแต่เลือกสรรชิ้นส่วนต่างๆ เพื่อมาประกอบเข้าด้วยกันเป็นวินโดวส์แบบที่ต้องการ แล้วเรียกคำสั่ง Create Windows ( ) หลังจากนั้น Windows ก็จะวาดวินโดวส์ตามที่แอพพลิเคชั่นต้องการ

การสร้างวินโดวส์นั้นแอพพลิเคชั่นจะเป็นผู้สร้าง แต่การบริหารงานวินโดวส์จะเป็นหน้าที่ของระบบวินโดวส์ ไม่ว่าจะเป็นการวาดวินโดวส์บนจอภาพด้วยรูปแบบต่างๆ (วินโดวส์หรือว่าไอคอน) ตำแหน่งของวินโดวส์, การเปลี่ยนขนาดของวินโดวส์ ฯลฯ จะมีก็แต่ส่วนหนึ่งของวินโดวส์ที่เรียกว่า พื้นที่ทำงาน (Client Area) ของวินโดวส์ ซึ่งส่วนนี้เองที่แอพพลิเคชั่นที่เราเขียนจะสามารถควบคุมรูปแบบการใช้งานต่างๆ อย่างเต็มที่ และเนื่องจากแอพพลิเคชั่นอันหนึ่งสามารถมีวินโดวส์ได้หลายวินโดวส์พื้นที่ทำงานของแต่ละวินโดวส์ก็แตกต่างกันออกไป จะมีการกำหนดให้เขียนส่วนตอบสนองของแต่ละวินโดวส์ (หรือพื้นที่ทำงาน) แต่ละอันโดยเฉพาะในรูปแบบของฟังก์ชั่นประจำวินโดวส์ (WinProc) ซึ่งการเขียนวินโดวส์ฟังก์ชั่นก็จะมีรูปแบบพิเศษตายตัวซึ่งจะได้อธิบายในเรื่องต่อๆ ไป




**รูป 3.1 ส่วนประกอบต่างๆ ของวินโดวส์**

### 3.1.2 เมนู

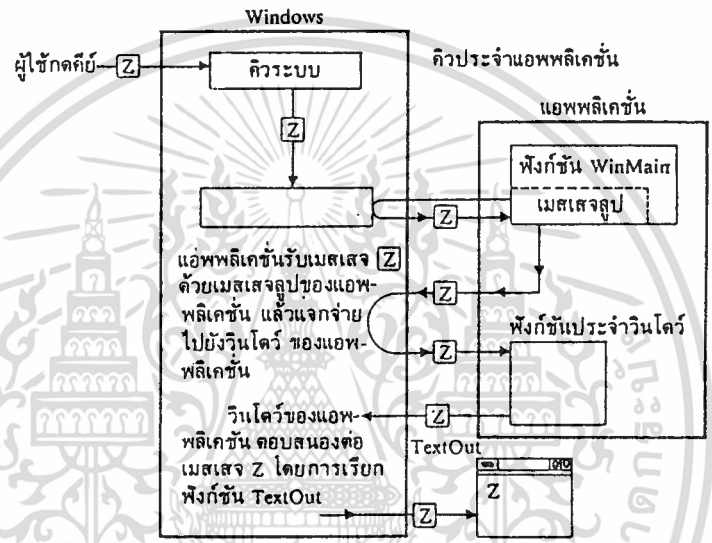
เมนูเป็นสิ่งที่ใช้รับข้อมูลหลักของแอพพลิเคชั่นที่ทำงานบนวินโดวส์ เมนูประกอบไปด้วยรายการคำสั่งเรียงกันไป ซึ่งจะให้ผู้ใช้ได้เลือกดูและใช้งานได้ง่ายๆ สำหรับหน้าที่การจัดการ การแสดงเมนู และการเลือกรายการในเมนูเป็นหน้าที่ของ Windows แต่หลังจากผู้ใช้เลือกเรียบร้อยแล้ว Windows ก็จะส่งคำสั่งที่ผู้ใช้เลือกส่งมาทางเมสเสจคิว เพื่อให้แอพพลิเคชั่นได้ตอบสนองต่อแอพพลิเคชั่นนั้นๆ

### 3.1.3 กรอบข้อความ (Dialog Boxes)

กรอบข้อความเป็นวินโดวส์ชนิดหนึ่งที่ใช้งานชั่วคราว เพื่อให้แอพพลิเคชั่นได้ติดต่อกับผู้ใช้ เช่น รับคำสั่งในรายละเอียดให้มากขึ้น ภายในกรอบข้อความจะประกอบไปด้วยคอนโทรล (Control - คอนโทรลก็เป็นวินโดวส์ชนิดหนึ่ง) คอนโทรลแต่ละชนิดก็จะใช้สำหรับติดต่อกับผู้ใช้แบบง่ายๆ หนึ่งแบบ เช่น Edit จะเป็นวินโดวส์ที่ให้ผู้ใช้ใส่ข้อความ ปุ่มกด (push button) จะเป็นวินโดวส์สำหรับการคลิ้กเมาส์ สมมติว่าเรามีกรอบข้อความอันหนึ่งเป็นกรอบข้อความสำหรับเปิดไฟล์ ก็จะประกอบไปด้วยคอนโทรลอันหนึ่งสำหรับใส่ชื่อไฟล์ คอนโทรลอันหนึ่งสำหรับเลือกไดเรกทอรี คอนโทรลอันหนึ่งสำหรับบอกให้เริ่มการทำงาน (OK) และคอนโทรลอีกอันหนึ่งสำหรับให้ยกเลิกการทำงาน (cancel) เป็นต้น

### 3.1.4 เมสเสจ

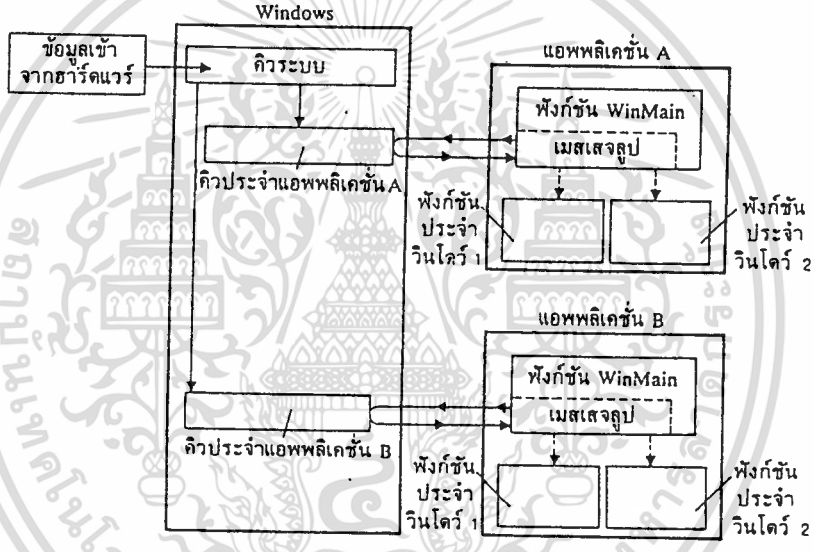
ด้วยเหตุที่ว่าแอพพลิเคชั่นจะรับข้อมูลเข้าโดยผ่านทางเมสเสจคิว ส่วนประกอบหลักของแอพพลิเคชั่นจึงต้องมีส่วนหนึ่งที่เรียกว่า เมสเสจลูป (Message Loop) เพื่อที่จะรับเมสเสจและแจกจ่ายไปยังวินโดวส์ที่เหมาะสม




**รูป 3.2 การจัดการเข้าทางคีย์บอร์ด**

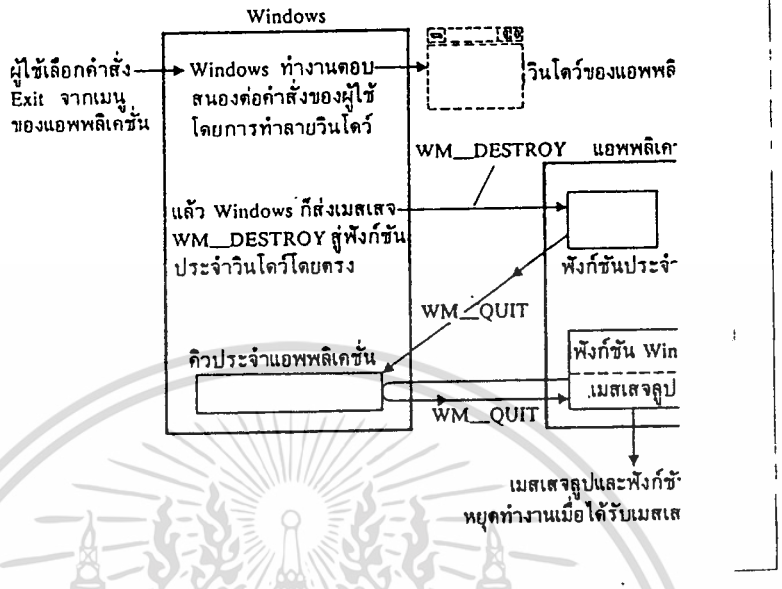
จากรูปที่ 3.2 เมื่อเรากดคีย์บอร์ดหนึ่งคีย์ Windows จะรับทราบและสร้างเมสเสจสำหรับคีย์บอร์ดที่เหมาะสมวางลงบน System que (เป็นเมสเสจของระบบ Windows ทั้งระบบ) แล้งส่งต่อไปยังคิวประจำแอพพลิเคชั่น (Application que) ของแอพพลิเคชั่นที่เหมาะสม จากนั้นเมสเสจลูปจะอ่านเมสเสจแล้วตีความ ซึ่งจะได้เป็นเมสเสจของตัวหนังสือในรูปของรหัส ANSI (ผ่านมากับเมสเสจที่ชื่อ WM_CHAR) แล่วแจกจ่ายในลักษณะเมสเสจของคีย์บอร์ด ไปยัง WinProc ที่เหมาะสม หากฟังก์ชั่นประจำวินโดวส์นั้นตอบสนองเมสเสจด้วยการแสดงตัวหนังสือออกทางจอภาพ ก็จะใช้ฟังก์ชั่น TextOut ในการแสดงตัวอักษรบนพื้นที่ทำงานของวินโดวส์นั้น

วินโดวส์ไม่เพียงแค่สามารถจัดการเมสเสจให้แก่แอพพลิเคชั่นเดียวเท่านั้น แต่ยังสามารถจะรวบรวมและแจกแจงเมสเสจให้แก่แอพพลิเคชั่นหลายๆ แอพพลิเคชั่นพร้อมกันได้ด้วย ดังรูป 3.3 จะเห็นว่า Windows รวบรวมข้อมูลเข้าทั้งหมด แล้วจัดแจงส่งไปให้แต่ละแอพพลิเคชั่น ในแต่ละแอพพลิเคชั่นจะมีเมสเสจลูปคอยรับและแจกจ่ายไปให้ฟังก์ชั่นประจำวินโดวส์ของตน




**รูป 3.3 การจัดการข้อมูลสำหรับแอพพลิเคชั่นสองตัว**

การส่งเมสเสจ นอกจากจะส่งทางคิวประจำแอพพลิเคชั่น Windows ยังสามารถสร้างโดยตรงได้ด้วย เมสเสจบางชนิด เช่น เมื่อ Windows ตอบรับการทำลายวินโดวส์ (หลังจากเราเลือกคำสั่ง Close ในsystem menu) ก็จะส่งเมสเสจชื่อ WM_DETROY ไปยังฟังก์ชั่นประจำวินโดวส์โดยตรง (ไม่ผ่านเมสเสจคิวเลย) หลังจากนั้นฟังก์ชั่นประจำวินโดวส์ก็จะตอบสนองโดยส่งสัญญาณให้แก่ WinMain ว่าวินโดวส์ได้ถูกทำลายแล้ว แอพพลิเคชั่นควรจะเลิกทำงานก็จะใช้ฟังก์ชั่น Post-QuitMessage ส่งเมสเสจชื่อ WM-QUIT ไปยังคิวประจำแอพพลิเคชั่นและจากรูปที่ 3.4 เมื่อเมสเสจลูปได้รับเมสเสจ WM-QUIT ก็จะเลิกทำงาน




**รูป 3.4 การจัดการเมสเสจของวินโดวส์**

## 3.2 การทำงานภายในโปรแกรมคาราโอเกะบน Windows

เมื่อเราทำการคอมไพล์ / ลิงค์ ไฟล์จาก Borland C++ ver 4.0 โปรเจคไฟล์ที่ชื่อ KARAOKE.IDE จะได้ผลลัพธ์เป็น KARAOKE.EXE ซึ่งมีส่วนประกอบภายใน ดังนี้

KARAOKE.EXE

- mcisound.cpp
- mcisound.rc
- kara1.ico
- . .\ . . \ . . \ lib \ default.def

หลังจากนั้นจะได้ไฟล์ KARAOKE .EXE ที่สามารถรัน File Manager ได้ โดยดับเบิ้ล-คลิ้ก ส่วนที่มีโปรแกรม KARAOKE .EXE อยู่บน MS Windows 3.1 หรือสูงกว่าได้ แต่จะต้องมี Driver [MCI] MIDI Sequencer และ [MCI] Sound ที่มีชื่อใน SYSTEM.INI ของ MS Windows ในหัวข้อ

[MCI]

WaveAudio = mciwave.drv

Sequencer = mciseq.drv

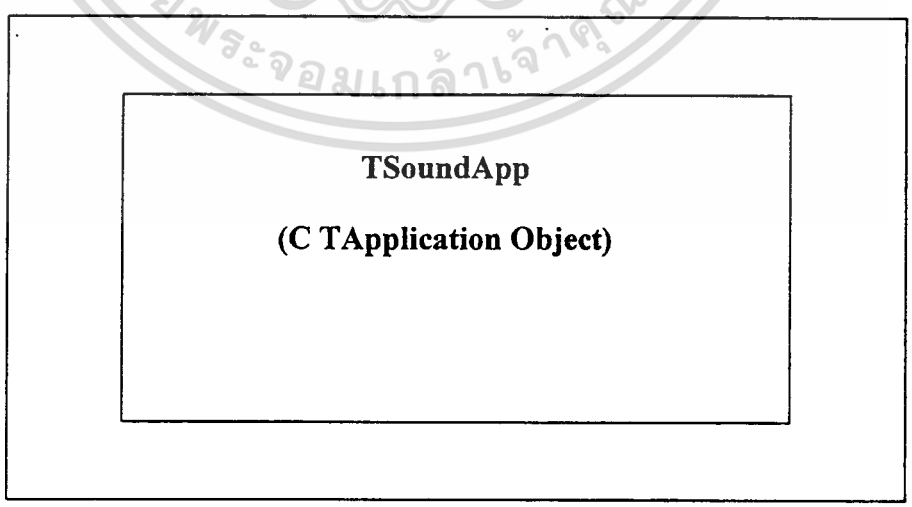
ถ้ามองภาพรวมๆ ภายใน โปรแกรม KARAOKE.EXE (ดังรูป 3.5)

**KARAOKE.EXE**

**รูป 3.5 ภาพโดยรวมของไฟล์ KARAOKE.EXE**

ภายในโปรแกรมจะมี Application object ซึ่งใช้ชื่อว่า TSoundApp ที่สืบทอดมาจาก OWL TApplication class ของ Borland C++ ver 4.0

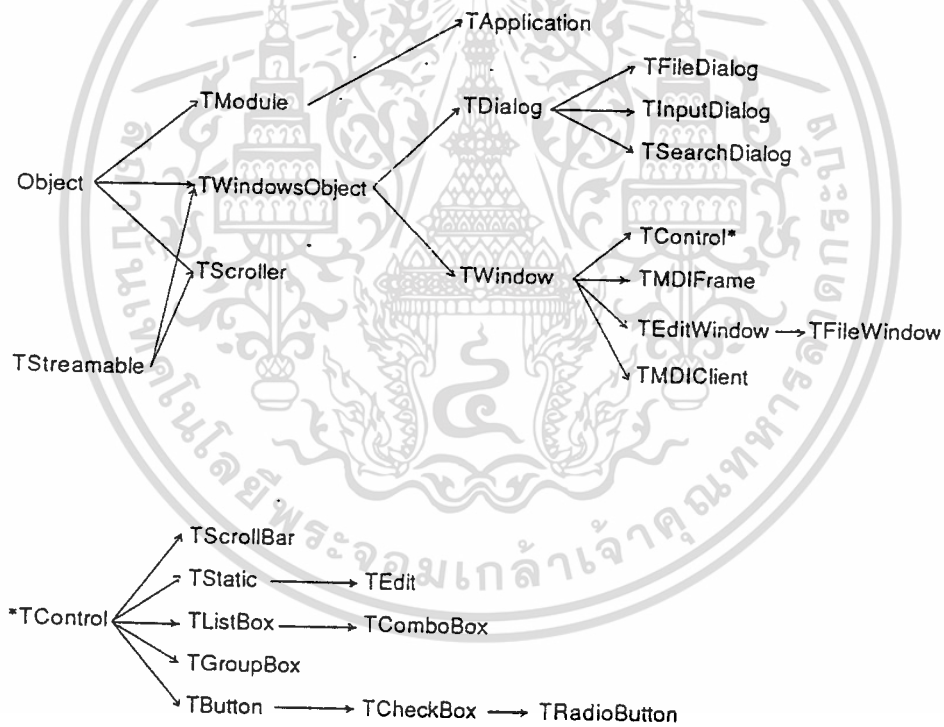
**KARAOKE.EXE**




**รูป 3.6 TSoundApp**

### 3.2.1 การสืบทอดคลาสของ OWL

ในรูป 3.7 ได้แสดงแผนผังการสืบทอดแต่ละคลาส (class) ของ OWL จะเห็นว่าคลาสทั้งหมดของ OWL สืบทอดจากคลาส Object ซึ่ง Borland C++ 4.0 นี้ได้จัดทำไว้ในไลบรารีเสมือนเก็บรวบรวมไว้ในห้องสมุด สามารถนำมาใช้งานได้ในทันที คลาส OWL ทั้งหมดได้สืบทอดจากคลาส TStreamable (ให้การสนับสนุนสตรีม I/O) ทำให้สามารถอ่านและเขียนออปเจ็กของ OWL ทั้งหมดด้วยตัวดำเนินการ << และ >> ในคำสั่งสตรีม (stream)




**รูป 3.7 แสดงการสืบทอดคลาส ObjectWindows**

แต่ละคลาสในรูป 3.7 มีลักษณะการทำงานดังนี้

| | |
|---|---|
| TModule | ให้การสนับสนุนแอพพลิเคชั่น เพื่อบริการโปรแกรมของ OWL ทั้งหมดที่ใช้คลาส TApplication ซึ่งสืบทอดจากคลาส TModul |
| TWindowsObject | จะมีคลาสสืบทอดของ OWL ที่สำคัญๆ จำนวนมาก แต่จะไม่ใช้คลาส TWindowsObject โดยตรง จะใช้คลาสสืบทอดแทน เช่น ใช้คลาส TDLalog สร้างกรอบข้อความ (dialog boxes) แสดงรายการต่างๆ และ TWindow ช่วยเปิดวินโดวส์ได้หลายๆ วินโดวส์ส่วนคลาสอื่นๆ ที่สืบทอดจากคลาส TDialog และ TWindow จะให้การสนับสนุนกรอบข้อความมาตรฐาน และ MDI (Multiple Document Interfaces) ตลอดจนการควบคุม |
| TScroller | ใช้เลื่อนสารบัญของวินโดวส์ให้เลื่อนขึ้น เลื่อนลง เลื่อนไปทางขวา และเลื่อนไปทางซ้าย โดยใช้บาร์คอนโทรล (bar control) |

**TWindow** นี้จะช่วย สนับสนุน ไดอะลอกมาตรฐาน (standard dialog) MDI (Multiple document) และการควบคุม

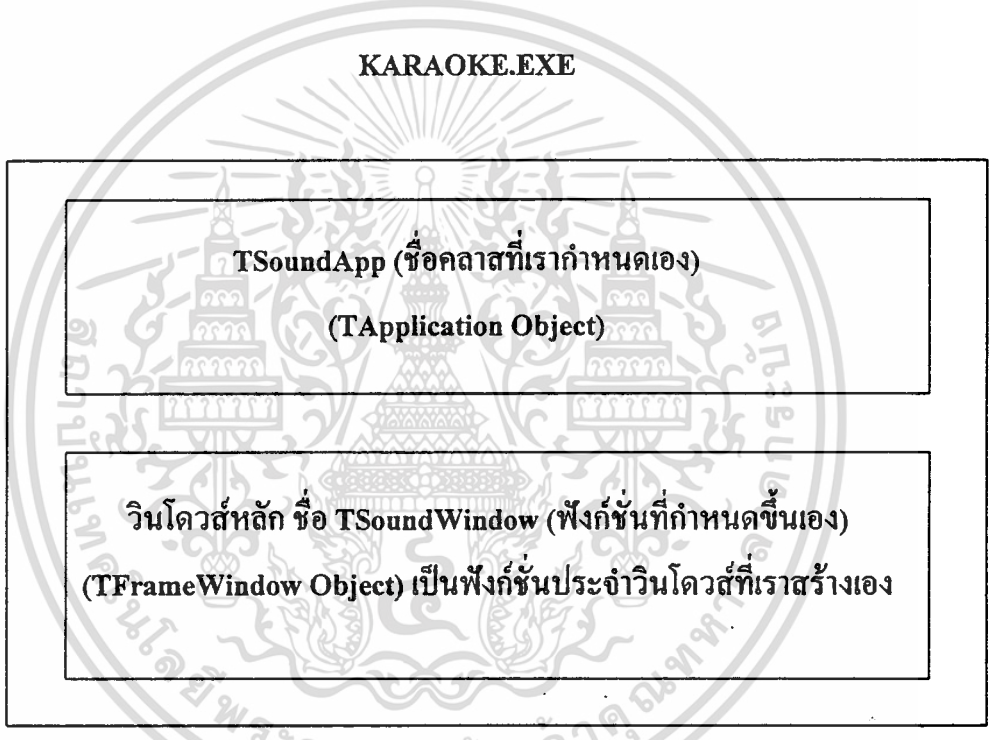
**TControl** ได้ขยายการสืบทอดคลาสด้วยกลุ่มของคลาสชนิดพิเศษ เพื่อใช้เชื่อมโยงกับงานต่างๆ ที่เกี่ยวข้องกับการควบคุมวินโดวส์ เช่น การเลื่อนบาร์ (scroll bars) บ็อกซ์ (boxs) ปุ่ม (button) ปุ่มวิทยุ (radio button) และบ็อกซ์สำหรับการใช้ตรวจสอบ ตลอดจนรายการเท็กซ์เป็นต้น

### 3.2.2 รู้จักโปรแกรมที่เขียนขึ้นจาก Borland C++

โปรแกรมที่เขียนขึ้นจาก Borland C++ ver 4.0 สามารถทำการรันได้เพียงออปเจคเดียว แต่เราสามารถเพิ่มโปรแกรมย่อยตัวอื่นๆ ในวินโดวส์หลักได้ เช่น โปรแกรมใน รูป 3.8 นี้ใช้โปรแกรมย่อยที่มาจาก OWL ชื่อ TFrameWindows ซึ่งเป็นคลาสย่อยของคลาส TWindow โดยเราจะกำหนดคลาสของเราเอง ชื่อว่า TSoundWindow โดยมีวิธีการประกาศ ดังนี้

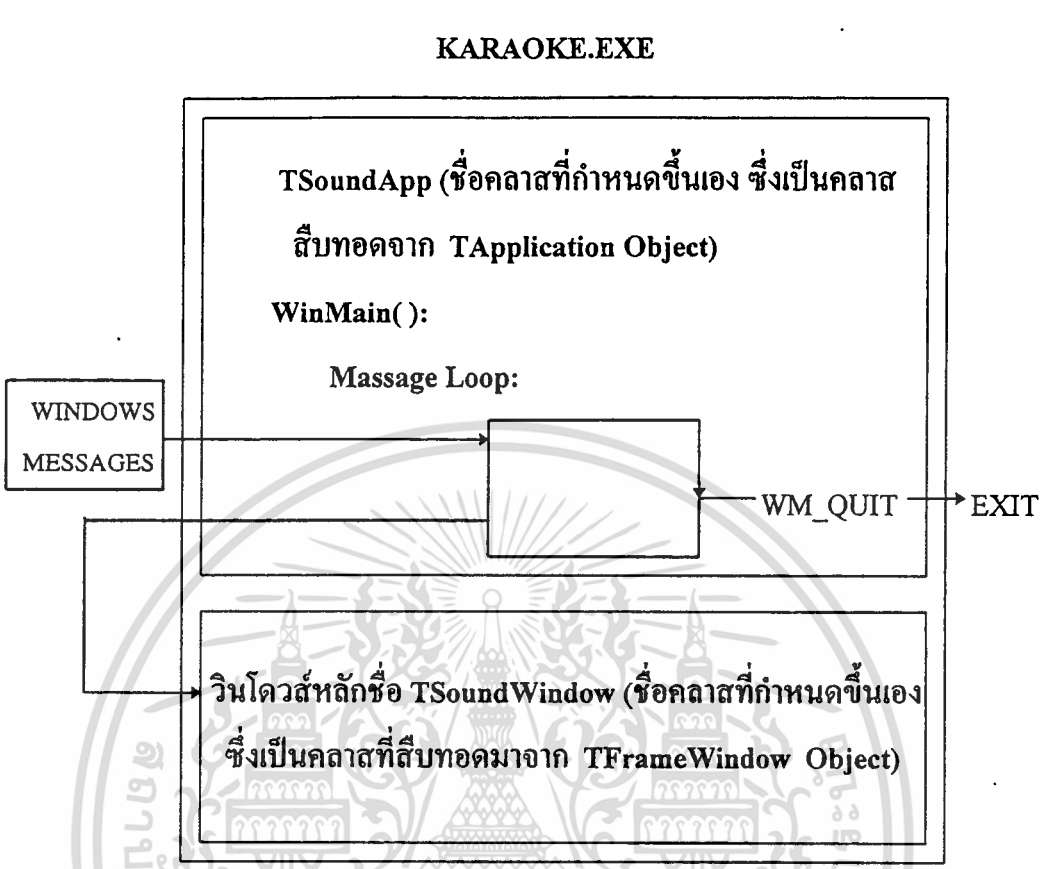
```
class TSoundWindow : Public TFrameWindow
{
//ประกาศฟังก์ชั่นต่างๆ และตัวแปรภายในคลาส TSoundWindow ที่เราใช้
}
```

ดังนั้นวินโดวส์หลักของเราจะชื่อ TSoundWindow



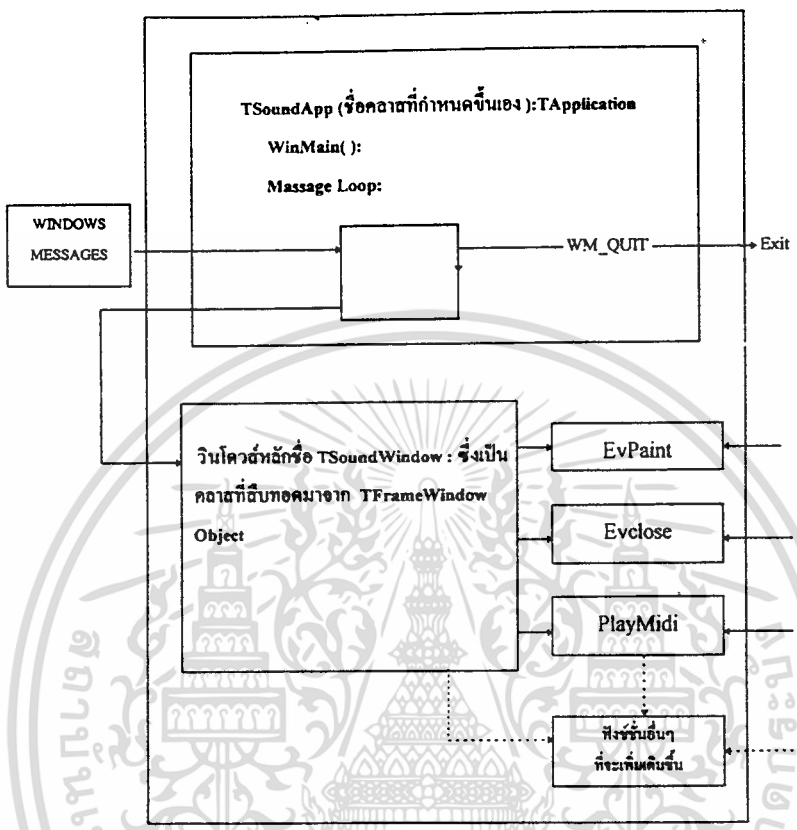

**รูป3.8 โปรแกรม TSoudWindow**

Obtion ที่ชิป TSondApp จะเป็นตัวแทนของโปรแกรมที่คอยรับเมสเสจต่างๆ ที่มีประมาณ 200 เมสเสช เช่น WM_QUIT เป็นเมสเสจที่บอกการหยุดการทำงานของโปรแกรมแอพพลิเคชั่น WM ย่อมาจาก Windows Massage โดยเมสเสจต่างๆเหล่านี้ จะส่งเข้าไปหา TSoundApp หรือ WinMain( ) โดยภายในฟังก์ชั่นนี้จะทำการรับจาก MS Windows หลังจากนั้นจะทำการส่งเมสเสจที่ตรวจสอบแล้วให้กับ TSoundWindow ดังรูป 3.9




**รูป 3.9 การส่งเมสเสจให้กับ TSoundWindow**

ในวินโดวส์หลักชื่อ TSoundWindow จะทำการรับเมสเสส เช่น EV_MESSAGE, EV_WM_PAINT ,EV_COMMAND (SM_PLAY) , ภายในโปรแกรม KARAOKE.EXE โดยพวกเมสเสจที่อยู่ภายใน TSoundWindow ต้องเป็นฟังก์สมาชิก ( Member Functions ) ของ TFrame Window Objet และจะตอบสนองตามฟังก์ใน TSound Window ดังนี้




**รูปที่ 3.10 การตอบสนองฟังก์ชั่นของ TSoundWindow**

ตัวอย่างการประกาศ ฟังชั่นอื่นๆที่จะเพิ่มเติมขึ้นใน คลาส TSound Window เช่น

```
Void    TSoundWindow::CmFileExit( )
{
        CloseWindow( );
}
```

โดยจะเขียนใน MCISOUND.CPP

# บทที่ 4

# การทดลองและผลการทดลอง

## 4.1 การทดลองโปรแกรม

โปรแกรมให้จังหวะดนตรีและคำร้อง เป็นโปรแกรมที่พัฒนาเพื่อให้คอมพิวเตอร์สามารถให้ความบันเทิงแก่ผู้ใช้คอมพิวเตอร์ทั่วไปได้

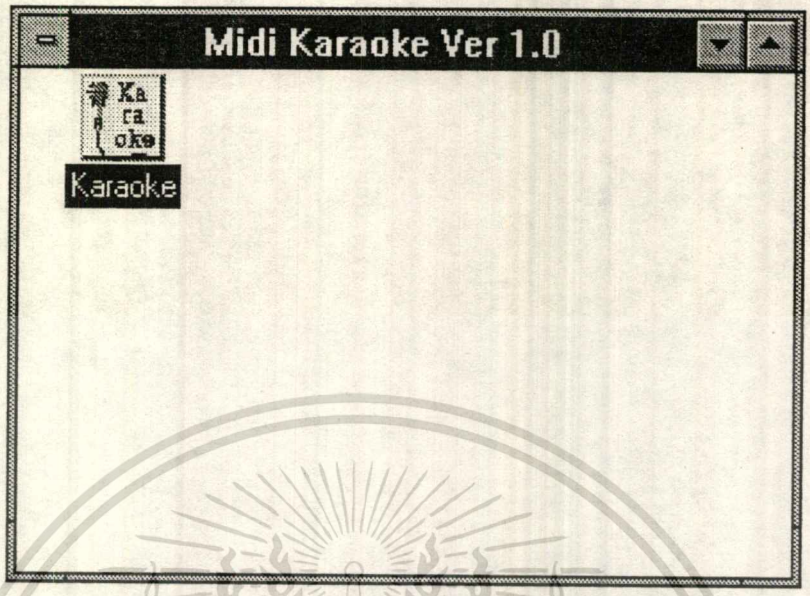
### 4.1.1 การทดลองการทำงานของโปรแกรมมีขั้นตอนดังต่อไปนี้

- ติดตั้งโปรแกรมและอุปกรณ์ต่างๆ ให้เรียบร้อย
- เปิดกลุ่มไอคอน Midi Karaoke Ver 1.00 ดังรูปที่ 4.1

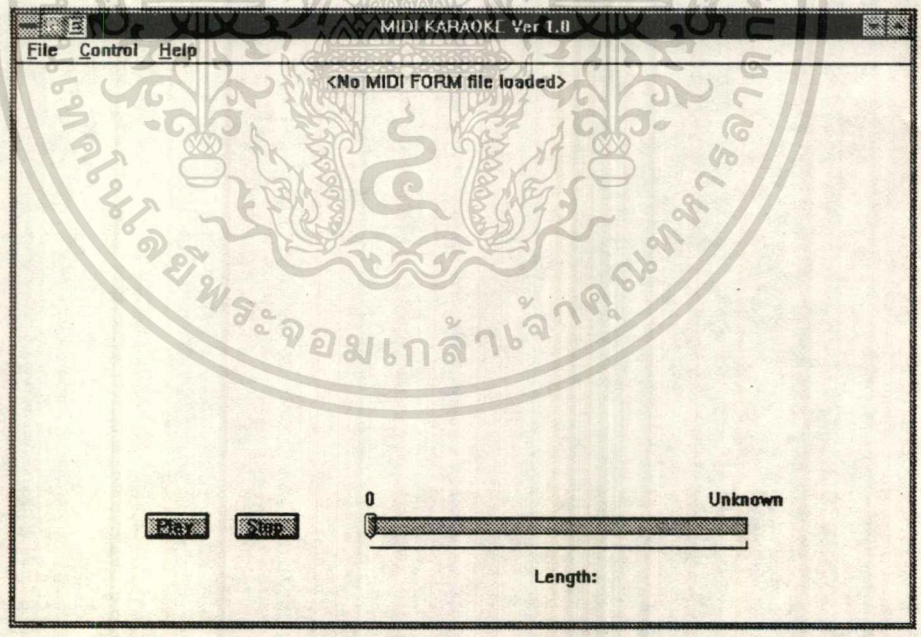

**รูปที่ 4.1 กลุ่มไอคอน MIDI Karaoke ของโปรแกรม Karaoke**

- เรียกโปรแกรม Karaoke โดยคลิ้กที่ไอคอน Karaoke 2 ครั้ง ดังรูปที่ 4.2

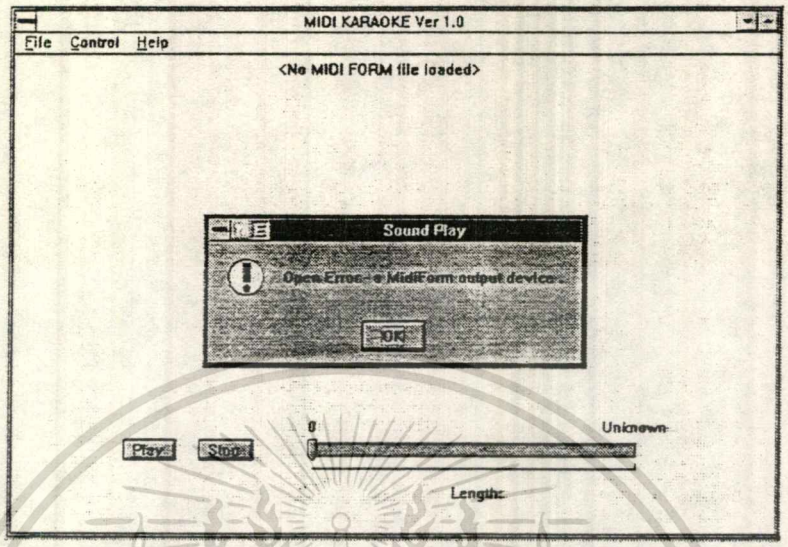
โปรแกรมจะแสดงวินโดวส์หลักของโปรแกรม Karaoke ดังรูปที่ 4.3




รูปที่ 4.2 แสดงไอคอนของโปรแกรม Karaoke




รูปที่ 4.3 วินโดวส์หลักของโปรแกรม Karaoke




**รูปที่ 4.4 กรอบข้อความแสดงความผิดพลาด**

- คลิ้กปุ่ม Play

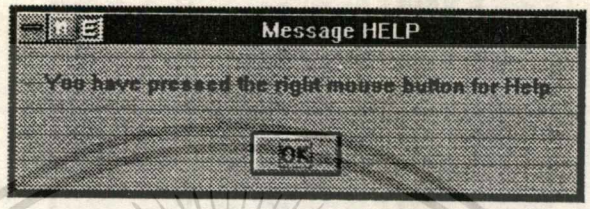
โปรแกรมจะตอบรับโดยแสดงกรอบข้อความดังรูปที่ 4.4 พร้อมทั้งเสียง

- คลิ้กปุ่มด้านขวาของเม้าส์

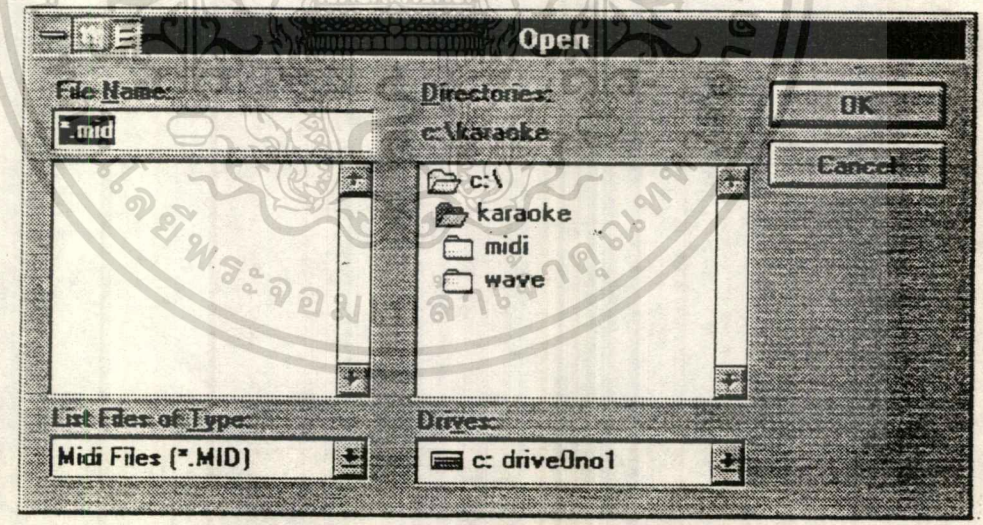
โปรแกรมจะตอบรับโดยแสดงกรอบข้อความเกี่ยวกับการวิธีใช้โปรแกรมดังรูปที่ 4.5

- เลือกคำสั่ง Open จากเมนู File

โปรแกรมตอบรับโดยแสดง กรอบข้อความดังรูปที่ 4.6 ในกรอบข้อความนี้เราสามารถเลือกเปิดไฟล์ที่มีนามสกุล MID หรือ WAV ก็ได้ โดยเลือกในช่อง List File Type เมื่อเลือกเพลงได้แล้วให้ตอบตกลง โดยคลิ้กปุ่ม OK



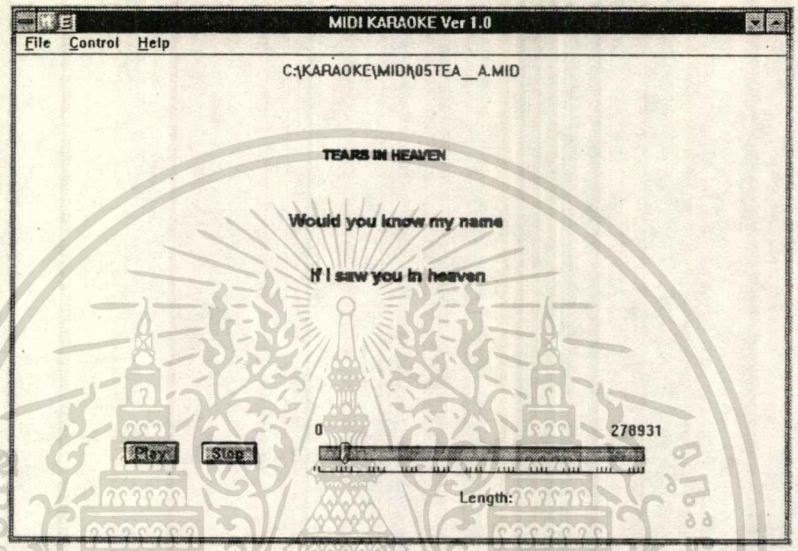

รูปที่ 4.5 กรอบข้อความวิธีการใช้โปรแกรม




รูปที่ 4.6 กรอบข้อความเปิดไฟล์

- คลิ้กปุ่ม Play

โปรแกรมจะเล่นเพลงที่ได้เปิดเอาไว้ก่อนหน้านี้ ดังรูปที่ 4.7 และเปลี่ยนคำสั่งในเมนู Control จาก Play เป็น Pause แทนที่



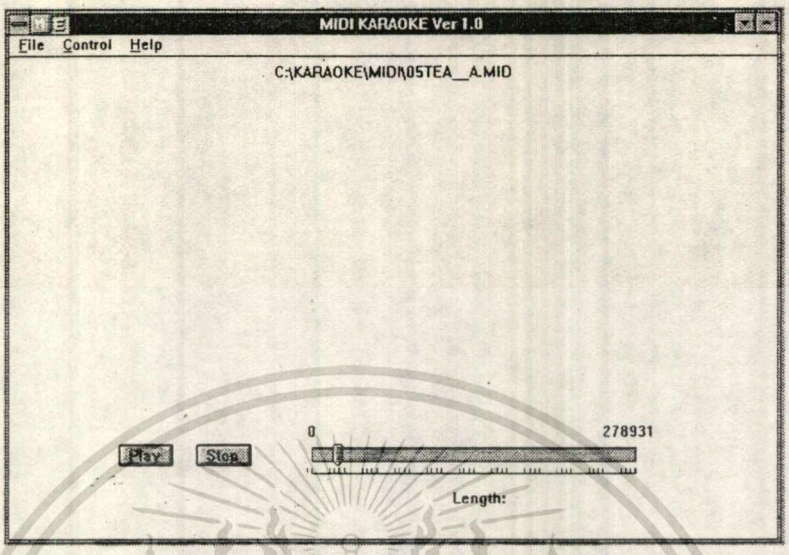

**รูปที่ 4.7 วินโดวส์หลักของโปรแกรมขณะเล่นเพลง**

- คลิ้กปุ่ม Play อีกครั้ง

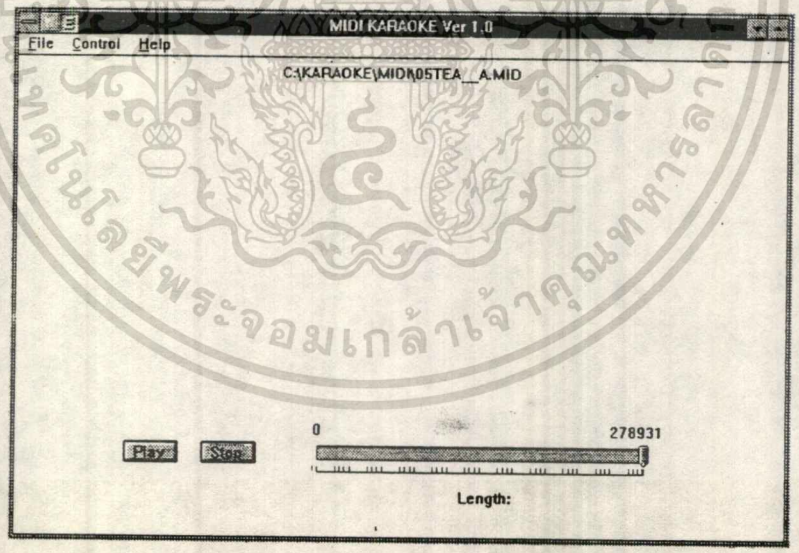
คลิ้กปุ่ม Play ครั้งที่ 2 จะเป็นการหยุดเล่นชั่วคราว ดังรูปที่ 4.8โปรแกรมจะตอบรับโดยหยุดการเล่นเพลง หยุดการเลื่อนบาร์และเปลี่ยนคำสั่งในเมนู Control จาก Pause เป็น Play แทนที่

- คลิ้กปุ่ม Stop

ปุ่มนี้คือปุ่มหยุดการเล่นเพลง เมื่อคลิ้กโปรแกรม และให้บาร์แสดงอยู่ตำแหน่งสุดท้ายของความยาวเพลง ดังรูปที่ 4.9




รูปที่ 4.8 วินโดวส์หลักของโปรแกรมขณะหยุดเล่นเพลงชั่วคราว




รูปที่ 4.9 วินโดวส์หลักของโปรแกรมขณะหยุดเล่นเพลง

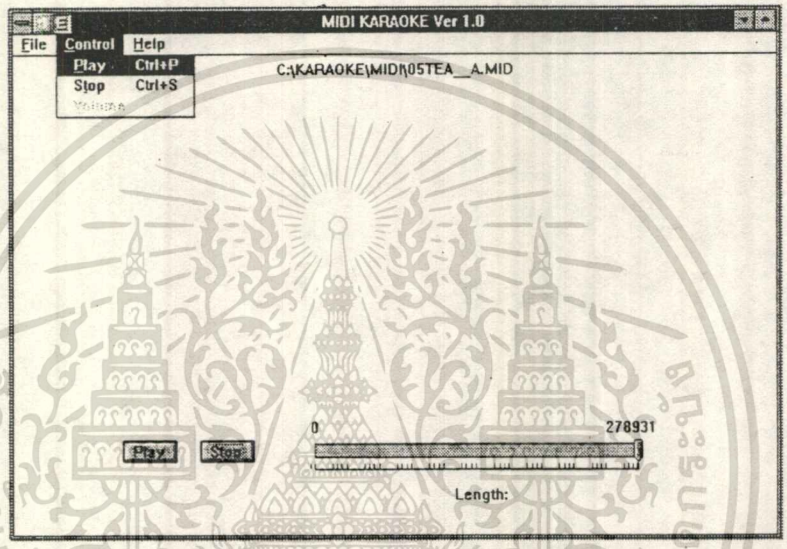
ในส่วนของเมนู Control จะประกอบไปด้วยคำสั่ง Play และ Stop ดังรูป 4.10

- เลือกคำสั่ง Play จากเมนู Control

โปรแกรมจะตอบรับเหมือนตอนคลิ้กปุ่ม Play

- เลือกคำสั่ง Stop จากเมนู Control

โปรแกรมจะตอบรับเหมือนตอนคลิ้กปุ่ม Stop




**รูปที่ 4.10 วินโดวส์หลักของโปรแกรมในส่วนเมนู Control**

- เลือกคำสั่ง Content จากเมนู Help

เมนู Help ประกอบด้วย Content และ About ในส่วน Content เมื่อเรียกโปรแกรมจะแสดงกรอบข้อความคู่มือการใช้งาน

- เลือกคำสั่ง About จากเมนู Help

โปรแกรมจะแสดงกรอบข้อความข้อมูลเบื้องต้นของโปรแกรม

## 4.2 สรุปผลการทำงานของโปรแกรม

การทำงานของโปรแกรมสามารถเล่นไฟล์ที่มีนามสกุล *.WAV และ *.MID ได้ทั้ง 2 แบบ ( แบบ 0 และ แบบ 1 ) โดยจะตรวจสอบขนาด ,ความยาว และรูปแบบของไฟล์เอง ความยาวของเพลงนำมาใช้ในการเซทการเลื่อนบาร์ของโปรแกรม การสั่งงานให้โปรแกรม

เล่นเพลง , หยุดชั่วคราว หรือ หยุดเล่นเพลงจะสามารถสั่งได้จากปุ่ม ( Play และ Stop ) และเมนู Control ก็ได้

# บทที่ 5
# บทวิจารณ์และสรุป

## 5.1 บทสรุป

ในอนาคต แนวโน้ม ในการใช้งานคอมพิวเตอร์ภายในบ้านจะมีเพิ่มมากขึ้น นอกจากจะใช้คอมพิวเตอร์ในการทำงานแล้วก็สามารถสร้างความบันเทิงได้ด้วย โปรแกรมให้จังหวะดนตรีและคำร้องนี้ เป็นโปรแกรมที่สร้างขึ้น โดยเขียนโปรแกรมด้วย Borland C++ ver 4.0 สำหรับวินโดวส์

## 5.2 ปัญหา

ในการจัดทำโครงงาน สร้างโปรแกรมให้จังหวะดนตรีและคำร้อง เป็นโครงงานที่ผู้จัดทำคิดว่าเป็นไปได้ง่าย ถึงแม้จะไม่มีทฤษฎีมากนัก สำหรับการเขียนโปรแกรมเกี่ยวกับดนตรี ผู้จัดทำได้ประสบปัญหามากมาย โดยกล่าวเป็นประเด็น ดังนี้

### 5.2.1 แหล่งข้อมูล

ในการจัดทำโครงการนี้ เรามีข้อมูลทางทฤษฎีน้อย และไม่มีต้นแบบให้ศึกษาเป็นแนวทาง ข้อมูลส่วนใหญ่ได้จากการซื้อหนังสือที่เกี่ยวกับการเขียนโปรแกรมของ Borland C++ 4.0 , ซาวด์บลาสเตอร์ ซึ่งส่วนใหญ่เป็นภาษาต่างประเทศ ต้องเสียเวลากับการแปลข้อมูลเพื่อทำความเข้าใจ โดยอุปกรณ์บางอย่างที่ต้องใช้เป็นข้อมูลที่ต้องการนั้นไม่มีขายในประเทศจึงต้องสั่งซื้อจากต่างประเทศ (เนื่องจากค้นเจอชื่อหนังสือที่ต้องการจากหนังสือที่เป็นภาษาต่างประเทศ)ซึ่งก็มีอุปสรรคมากมายและกว่าที่จะได้อุปกรณ์นี้มาก็ประมาณกลางเดือนพฤศจิกายน จึงทำให้เกิดความล่าช้า ส่วนหนังสือที่เกี่ยวกับการเขียนโปรแกรมด้วย Borland C++บนวินโดวส์นั้นหายากมากเพราะหนังสือที่ขายอยู่ตอนนี้จะมีเนื้อหาในการเขียนโปรแกรมด้วย Borland C++ บนวินโดวส์นั้นจะมีอยู่เพียง 2 หรือ 3 บทท้ายๆ เท่านั้น ซึ่งก็กล่าวถึงได้ไม่

ละเอียดนัก และหนังสือที่มีรายละเอียดเรื่องมัลติมิเดียมากๆ ก็ไม่ค่อยมี จึงทำให้การศึกษาเกี่ยวกับมัลติมิเดียไม่ชัดเจนเท่าที่ควร

### 5.2.2 เครื่องมือทดสอบ , ทฤษฎี และกระบวนการ

ในการพัฒนาโปรแกรมให้จังหวะดนตรีและคำร้อง ซึ่งเป็นโปรแกรมที่เขียนบน MS Windows โดยใช้ Borland C++ ver 4.0 เป็นตัว คอมไพล์ และลิงค์ เพื่อให้ได้ผลลัพธ์ *.EXE ซึ่งในการจัดทำนี้ ผู้จัดทำจะต้องเสียเวลาในการศึกษาหาแนวทาง สำหรับการเขียนโปรแกรมที่ใช้ Borland C++ ver 4.0 เนื่องจากโปรแกรมที่ใช้เป็นภาษา C++ แบบ OWL ของ Borland C++ ver 4.0 ในการใช้งานและพัฒนาบน MS Windows นั้นจึงต้องศึกษาการเขียนโปรแกรมภาษา C++ แบบ OOP เพื่อการเข้าถึงเมสเสจต่างๆ และการส่งผ่านเมสเสจ ฟังก์ชั่น API ของ MS Windows , คลาสต่าง ๆ ของวินโดวส์ และ คลาสต่าง ๆ ของ Borland C++ ver 4.0 หรือ OWL เป็นต้น

ในทางปฏิบัติ การเขียนโปรแกรมจำเป็นต้องศึกษาข้อมูลจากหนังสือ ซึ่งจัดหามาได้ยากและส่วนใหญ่เป็นภาษาต่างประเทศ ในบางครั้งจำต้องทดลองแบบลองผิดลองถูก แล้วนำมาวิเคราะห์หาแนวทางที่เหมาะสมกับโครงงาน อีกทั้งผู้จัดทำมีความรู้เกี่ยวกับภาษา C++ ไม่แตกฉานเท่าที่ควร และในการเขียนโปรแกรม Borland C++ บนวินโดวส์นี้ก็มีวิธีการเขียนแตกต่างจากการเขียนภาษา C++ ธรรมดา จึงเสียเวลามากกว่าที่กำหนดไว้ในกระบวนการวางแผนการทำงานไปมาก

### 5.2.3 เครื่องมือที่ใช้ในการจัดทำโครงงาน

ดังที่ทราบแล้วว่าโปรแกรมให้จังหวะดนตรีและคำร้อง เป็นโปรแกรมที่ต้องใช้การ์ดและทำการรันโปรแกรมบน MS Windows เป็นตัว คอมไพล์ และลิงค์ โดยใช้ Borland C++ ver 4.0 สำหรับวินโดวส์ ดังนั้นจึงต้องใช้คอมพิวเตอร์ที่มีความสามารถในการประมวลสูง เช่น เพนเทียม , RAM ที่ใช้ในการประมวลผล มากกว่า 16 Mbyte ขึ้นไป จึงจะทำให้การคอมไพล์ และ ลิงค์ บนวินโดวส์เร็วขึ้น และต้องมีเนื้อที่ฮาร์ดดิสก์ มากกว่า 200 Mbyte เนื่องจาก โปรแกรม MS Windows เองก็ใช้เนื้อที่มากอยู่แล้ว อีกทั้งโปรแกรม Borland C++ ver 4.0 สำหรับวินโดวส์ ต้องใช้เนื้อที่ในฮาร์ดดิสก์ ประมาณ 80 Mbyte-120 Mbyte ขึ้นอยู่กับการเลือกลงโปรแกรม แต่ในการจัดทำโครงงานนี้ ผู้จัดทำได้ใช้ ตัวประมวลผล

เบอร์ N30486-DX33 , RAM 8 Mbyte ฮาร์ดดิสก์ 420 Mbyte เมื่อทำการรันโปรแกรมจึงใช้เวลานานมาก ดังนั้นจึงต้องทำการ upgrade ตัวประมวลผลเป็น เบอร์ 80486-DX4 , RAM 12 Mbyte , ฮาร์ดดิสก์ 540 Mbyte เพื่อใช้ลองโปรแกรม ซึ่งได้ผลมากขึ้นประมาณ 30%

## 5.3 ข้อเสนอแนะและแนวทางการพัฒนา

โปรแกรมให้จังหวะดนตรีและคำร้องนี้ สามารถนำมาพัฒนาโดยดูความเป็นไปได้ในทฤษฎี และ ปฏิบัติ ดังสรุปเป็นข้อๆ ดังนี้

1. เลือกใช้คอมพิวเตอร์ ที่มีตัวประมวลผลที่มีความเร็วสูง เช่น เพนเทียม , RAM 16 Mbyte สำหรับการคอมไพล์และลิงค์ บนวินโดวส์

2. จัดทำโปรแกรมให้ติดต่อ MS Windows API ในระดับต่ำได้

3. ควรเรียกใช้ฟังก์ชั่น Windows API และใช้ภาษาแอสแซมบลี้ในการติดต่อผ่านไดรฟ์เวอร์ของการ์ดเสียงโดยตรง

4. ไม่ควรเรียกใช้ MCI (The Media Control Interface) แบบระดับสูง

5. ให้เรียกใช้ MCI ระดับต่ำ จะทำให้โปรแกรมที่ต้องการมีความเร็วมากขึ้น

6. จัดทำโปรแกรมให้กระชับขึ้น เนื่องจากโปรแกรมใหญ่ก็จะมีโค้ดมากขึ้นด้วย ต้องใช้ RAM มากขึ้น ทำให้โปรแกรมทำงานได้ช้า

7. ควรปรับปรุงอัลกอลิทึมให้ดีกว่าเดิม

8. ควรหลีกเลี่ยงการเกิด Dead Lock จากเมสเสจ

9. ควรหลีกเลี่ยงการติดต่อกับคอนโซลแบบดอส

**การหลีกเลี่ยงการเกิด Dead Lock จากเมสเสจ**

แอพพลิเคชั่นอาจเกิด Dead Lock ได้ถ้าแอพพลิเคชั่นที่ได้การควบคุมมาจากฟังก์ชั่นอื่น ทาง SendMessage แล้วส่งต่อการควบคุมไปอีก ซึ่งการส่งต่อการควบคุมต่อไปนั้นอาจไม่ต้องใช้ SendMessage ก็ได้ การใช้ฟังก์ชั่นข้างล่างเหล่านี้ก็เปรียบเสมือนการผ่านการควบคุมไปยังแอพพลิเคชั่นอื่น

- DialogBox
- DialogBoxIndirec
- DialogBoxIndirecParam

- DialogBoxParam
- GetMessage
- PeakMessage
- Yield

ฟังก์ชั่นที่เรียกใช้งาน SendMessage เพื่อส่งเมสเสจไปให้อีกฟังก์ชั่นทำงานนั้น จะไม่สามารถทำงานต่อไปได้จนกว่าฟังก์ชั่นที่รับเมสเสจนั้นจะทำเสร็จและให้ค่ากลับมา แต่ถ้าฟังก์ชั่นนั้นเรียกใช้งานฟังก์ชั่นต่างๆ ข้างบนก็จะเกิดกรณี dead lock ขึ้น ดังนั้นก่อนการเรียกใช้งานฟังก์ชั่นเหล่านั้น ควรจะตรวจสอบดูก่อนว่า เมสเสจที่ได้รับนั้นมาจาก SendMessage หรือไม่ โดยใช้ฟังก์ชั่น InSendMessage ถ้าให้ค่ากลับมาจริง ก็หมายความว่าเป็นเมสเสจที่ได้มาจาก SendMessage ให้เรียกใช้งาน ReplayMessage ก่อนที่จะส่งการควบคุมต่อไป

**การติดต่อกับคอนโซล**

คอนโซลอันประกอบไปด้วยคีย์บอร์ดและจอภาพนั้น เป็นทรัพยากรที่ต้องปันส่วนกัน เพราะฉนั้นฟังก์ชั่นมาตรฐานต่างๆ ที่ใช้คอนโซลโดยการยึดครอง จึงไม่ถูกบรรจุไว้ในไลบรารีที่ใช้กับวินโดวส์ฟังก์ชั่นเหล่านั้นได้แก่

- cgets
- cprintf
- cputs
- getch
- getche
- kbhit
- putch
- ungetch

แอพพลิเคชั่นของเราควรจะรับข้อมูลจากคอนโซลผ่านทางเมสเสจ WM_CHAR , WM_KEYDOWN และ WM_KEYUP ที่ถูกส่งไปยังฟังก์ชั่นประจำวินโดวส์และฟังก์ชั่นประจำกรอบข้อความเท่านั้น แต่หากจะให้เทคนิคพิเศษ ก็สามารถใช้ฟังก์ชั่น PeekMessage

เพื่อตรวจดูเมสเสจทุกๆ เมสเสจที่อยู่ในคิวเมสเสจได้ หรือสร้างฟังก์ชั่นใน DLL เพื่อดักเมสเสจด้วยฟังก์ชั่น SetWindowHook ก็ยังได้

# บรรณานุกรม


จิรพัฒน์ จันทร์เจิดศักดิ์ และ วีระ นพนิราพาธ , การเขียนโปรแกรมบน Microsoft Windows , บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน) , พ.ศ. 2521

พอ. เจนวิทย์ เหลืองอร่าม , การเขียนโปรแกรมแบบ OOP ด้วย Borland C++ , สำนักพิมพ์สุขภาพใจ , พ.ศ. 2521

ราบินเดอร์ ศรีกิจจาภรณ์ , การเขียนโปรแกรมแบบโอโอพี ด้วย เทอร์โบ และ บอร์แลนด์ C++ , บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)

พัชญา พิทักไพรวัน , เรียนรู้ C ++ให้เก่งจากภาษา C มาสู่ C++ ใน 2 สัปดาห์ , บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน) , 2538

Brady Publishing , Borland C++ for Windows Programming , Third Edition , 1994

Ian Spencer , Teach Yourself OWL Programming in 21 Days , SAMS Publishing , First Edition, 1995

Martin L. Moor , The Ultimate Sound Blaster Book , Que Corporation , 1993

Microsoft Corporation , Microsoft Windows Multimedia Programmer ' s Reference , Microsoft Corporation , 1991

Microsoft Corporation , Microsoft Windows Multimedia Programmer ' s Work Book , Microsoft Corporation , 1991

Nabajyoti Barkakati , Borland C++ 4 Developer ' s Guide , SAMS Publishing , First Edition , 1994

Ori Gurewich and Nathan Gurewich , Borland C++ Multimedia Programmer ' s Reference , TECH Publication PTE LTD

Paul Perry , Multimedia Developer ' s Guide , SAMS Publishing , First Edition , 1994

Peter M. Ridge , David M. Golden , Ivan Luk and Scott E. Sindorf , Sound Blaster : The Official Book , Second Edition , McGraw-Hill , Inc. , 1994

Steve Rimmer , Advance Multimedia Programming , Windcrest , 1995

# ภาคผนวก ก

## โปรแกรม

# ภาคผนวก

## โปรแกรม

# โปรแกรม

# MCISOUND.CPP

```
//----------------------------------------------------------------------------
// OWL Karaoke          MCISOUND.CPP
#include <owl\owlpch.h>
#include <owl\applicat.h>
#include <owl\opensave.h>
#include <owl\framewin.h>
#include <owl\dc.h>
#include <owl\menu.h>
#include <owl\button.h>
#include <owl\slider.h>
#include <owl\slider.rh>
#include <math.h>
#include <mmsystem.h>
#include "mcisound.h"
#include <owl\point.h>
#define ID_SCROLL   150        // Scroll bar
#define TIMER_ID    264        // Unique timer ID.
#define MCI_PARM2(p) ((long)(void far*)&p)

UINT  DeviceId = 0;            // The global waveform device opened handle
BOOL  FlushNotify = FALSE;
//----------------------------------------------------------------------------
class TSoundBar : public THSlider {
 public:
        TSoundBar(TWindow* parent, int id, int x, int y, int w, int h,
                                TModule* module = 0)
                        : THSlider(parent, id, x, y, w, h, IDB_HSLIDERTHUMB, module)
 {}
```

```
        void   SetInfo(int ratio, long length);
        void   SetName(char* name);

        //
        // Override TScrollBars virtual functions
        //
        void   SBLineUp();
        void   SBLineDown();
        void   SBPageUp();
        void   SBPageDown();
        void   SBThumbPosition(int thumbPos);
        void   SBTop();
        void   SBBottom();

 private:
        int    MidiRatio;
        long   MidiLength;
        char   ElementName[255];
        void   ReposAndPlay(long newPos);
};

void
TSoundBar::ReposAndPlay(long newPos)
{
 MCI_PLAY_PARMS    MciPlayParm;
 MCI_SEEK_PARMS    MciSeekParm;
 MCI_SET_PARMS     MciSetParm;
 MCI_OPEN_PARMS    MciOpenParm;
 MCI_GENERIC_PARMS MciGenParm;
```

```
// Only allow SEEK if playing.
//
if (!DeviceId)
        return;

// Close the currently playing wave.
//
FlushNotify = TRUE;
MciGenParm.dwCallback = 0;
mciSendCommand(DeviceId, MCI_STOP, MCI_WAIT, MCI_PARM2(MciGenParm));
mciSendCommand(DeviceId, MCI_CLOSE, MCI_WAIT, MCI_PARM2(MciGenParm));

// Open the wave again and seek to new position.
//
MciOpenParm.dwCallback = 0;
MciOpenParm.wDeviceID = DeviceId;
#if !defined(__WIN32__)
        MciOpenParm.wReserved0 = 0;
#endif
MciOpenParm.lpstrDeviceType = 0;
MciOpenParm.lpstrElementName = ElementName;
MciOpenParm.lpstrAlias = 0;

if (mciSendCommand(DeviceId, MCI_OPEN, MCI_WAIT| MCI_OPEN_ELEMENT,
MCI_PARM2(MciOpenParm))) {
                MessageBox("Open Error", "Sound Play", MB_OK);
                return;
        }
DeviceId = MciOpenParm.wDeviceID;
```

```
// Our time scale is in SAMPLES.

MciSetParm.dwTimeFormat = MCI_FORMAT_MILLISECONDS;
if (mciSendCommand(DeviceId, MCI_SET, MCI_SET_TIME_FORMAT, MCI_PARM2
(MciSetParm))) {
        MessageBox("Set Time Error", "Sound Play", MB_OK);
        return;
}

// Compute new position, remember the scrollbar range has been scaled based
// on MidiRatio.
//
MciSeekParm.dwCallback = 0;
MciSeekParm.dwTo = (newPos*MidiRatio > MidiLength) ? MidiLength :

                                                    newPos*MidiRatio;
if (mciSendCommand(DeviceId, MCI_SEEK, MCI_TO, MCI_PARM2(MciSeekParm))) {
        MessageBox("Seek Error", "Sound Play", MB_OK);
        return;
}
MciPlayParm.dwCallback = (long)HWindow;
MciPlayParm.dwFrom = 0;
MciPlayParm.dwTo = 0;
if (mciSendCommand(DeviceId, MCI_PLAY, MCI_NOTIFY, MCI_PARM2
(MciPlayParm))) {
        MessageBox("Play Error", "Sound Play", MB_OK);
        return;
}
}
```

```
void
TSoundBar::SetInfo(int ratio, long length)
{
 MidiRatio = ratio;
 MidiLength = length;
}

void
TSoundBar::SetName(char* name)
{
 strcpy(ElementName, name);
}

void
TSoundBar::SBLineUp()
{
 THSlider::SBLineUp();
 ReposAndPlay(GetPosition());
}

void
TSoundBar::SBLineDown()
{
 THSlider::SBLineDown();
 ReposAndPlay(GetPosition());
}
```

```
void
TSoundBar::SBPageUp()
{
 THSlider::SBPageUp();
 ReposAndPlay(GetPosition());
}

void
TSoundBar::SBPageDown()
{
 THSlider::SBPageDown();
 ReposAndPlay(GetPosition());
}

void
TSoundBar::SBThumbPosition(int thumbPos)
{
 THSlider::SBThumbPosition(thumbPos);
 ReposAndPlay(GetPosition());
}

void
TSoundBar::SBTop()
{
 THSlider::SBTop();
 ReposAndPlay(GetPosition());
}
```

```
void
TSoundBar::SBBottom()
{
 THSlider::SBBottom();
 ReposAndPlay(GetPosition());
}


//----------------------------------------------------------------------

class TSoundWindow : public TFrameWindow {
 public:
        TSoundWindow(TWindow* parent, const char far* title);
        ~TSoundWindow();

        void    SetupWindow();

        LRESULT   MciNotify(WPARAM, LPARAM);
        void    EvPaint();
        void    CmFileOpen();
        void    CmFileExit();
        void    CmPlayMidi();
        void    CmStopMidi();
        void                CmVol();
        void                CmContent();
        void    CmHelpAbout();
        void    EvTimer(UINT id);
        void                SetPenSize(int newSize);
```

```
private:
        int                     XPOS,YPOS;
        char    ElementName[255];
        int     Running;
        int     Pause;
        UINT    TimeGoing;
        int     MidiRatio;
        long    MidiLength;
        TSoundBar*  SoundBar;
        MCI_GENERIC_PARMS MciGenParm;
        MCI_OPEN_PARMS   MciOpenParm;
        MCI_PLAY_PARMS   MciPlayParm;
        MCI_STATUS_PARMS  MciStatusParm;
        MCI_SET_PARMS    MciSetParm;

        void    GetDeviceInfo();
        void    StopMidi();
        void    StopMCI();
       // protected:
        void EvRButtonDown(UINT, TPoint&);
 DECLARE_RESPONSE_TABLE(TSoundWindow);
};

DEFINE_RESPONSE_TABLE1(TSoundWindow, TFrameWindow)
 EV_MESSAGE(MM_MCINOTIFY, MciNotify),
 EV_WM_PAINT,
 EV_COMMAND(SM_OPEN, CmFileOpen),
 EV_COMMAND(SM_EXIT, CmFileExit),
 EV_COMMAND(SM_PLAY, CmPlayMidi),
```

```
EV_COMMAND(SM_STOP, CmStopMidi),
EV_COMMAND(SM_ABOUT, CmHelpAbout),
EV_WM_TIMER,
END_RESPONSE_TABLE;


TSoundWindow::TSoundWindow(TWindow* parent, const char far* title)
 : TFrameWindow(parent, title)
{
 AssignMenu(ID_MENU);
 Attr.AccelTable = IDA_SOUNDPLY;
 Attr.X = 50;
 Attr.Y = 50;
 Attr.W = 700;
 Attr.H = 470;

 Running = 0;
 Pause = 0;
 WaveLength = WaveRatio = 0;
 ElementName[0] = 0;

 SoundBar = new TSoundBar = new TSoundBar(this, ID_SCROLL, 270, 340, 300, 40);

 new TButton(this, SM_PLAY, "&Play", 100, 340, 50, 20, TRUE);
 new TButton(this, SM_STOP, "&Stop", 170, 340, 50, 20, TRUE);
 //new  TButton(this, CM_VOL, "&Volume", 50 , 340 , 50 ,20,TRUE);

}
```

```
void TSoundWindow::EvRButtonDown(UINT , TPoint& point)
{
                    sndPlaySound ("c:\\help.wav", SND_SYNC);
                    MessageBox("You have pressed the right mouse button for Help", "
Message HELP", MB_OK);
}


void
TSoundWindow::SetupWindow()
{
 TFrameWindow::SetupWindow();
 SoundBar->SetRange(0, 0);
 SoundBar->SetRuler(0);
}

TSoundWindow::~TSoundWindow()
{
 StopMCI();
}

//
// EvPaint member function responds to WM_PAINT messages
//
void
TSoundWindow::EvPaint()
{
 TPaintDC paintDC(HWindow);
```

```
// File name.
//
if (strlen(ElementName))
        paintDC.TextOut(240, 5, ElementName, strlen(ElementName));
else
        paintDC.TextOut(240, 5, "<No MIDI FORM file loaded>", 26);
        paintDC.TextOut(400, 380, "Length:", 7);
        paintDC.TextOut(270, 321, "0", 1);     // Beginning value.

// Ending number of samples.
//
char buffer[10];
if (MidiLength)
        wsprintf(buffer, "%ld", MidiLength);
else
        strcpy(buffer, "Unknown");
paintDC.TextOut(535, 321, buffer, strlen(buffer));
}

void
TSoundWindow::GetDeviceInfo()
{
 MIDIOUTCAPS midiOutCaps;

 if (!midiOutGetDevCaps(DeviceId, &midiOutCaps, sizeof(midiOutCaps))) {
        MessageBeep(MB_ICONEXCLAMATION);
        MessageBox("GetDevCaps Error", "Sound Play", MB_OK);
        return;
 }
```

```
}

//
// Play the MIDI...
//
void
TSoundWindow::CmPlayMidi()
{
 if (!Running) {
        //
        // MCI APIs to open a device and play a .MID file, using
        // notification to close
        //
        //sndPlaySound ("sounder.wav", SND_SYNC);
        memset(&MciOpenParm, 0, sizeof MciOpenParm);

        MciOpenParm.lpstrElementName = ElementName;

        if (mciSendCommand(0, MCI_OPEN, MCI_WAIT | MCI_OPEN_ELEMENT,
                                                        MCI_PARM2
(MciOpenParm))) {
                //MessageBeep(MB_ICONEXCLAMATION);
                sndPlaySound ("C:\\Karaoke\\read.wav", SND_SYNC);
                sndPlaySound ("C:\\Karaoke\\error.wav", SND_SYNC);
                MessageBox(
                        "Open Error - a MidiForm output device .",
                        "Sound Play", MB_ICONEXCLAMATION);
                return;
        }
```

```
        DeviceId = MciOpenParm.wDeviceID;

        // The time format in this demo is in Samples.
        //
        MciSetParm.dwCallback = 0;
        MciSetParm.dwTimeFormat = MCI_FORMAT_MILLISECONDS;
        if (mciSendCommand(DeviceId, MCI_SET, MCI_SET_TIME_FORMAT,
MCI_PARM2(MciSetParm))) {
                MessageBeep(MB_ICONEXCLAMATION);

                MessageBox("SetTime Error", "Sound Play", MB_OK);
                return;
        }

        MciPlayParm.dwCallback = (long)HWindow;
        MciPlayParm.dwFrom = 0;
        MciPlayParm.dwTo = 0;
        mciSendCommand(DeviceId, MCI_PLAY, MCI_NOTIFY, MCI_PARM2
(MciPlayParm));

        // Modify the menu to toggle PLAY to STOP, and enable PAUSE.
        //
        TMenu menu(HWindow);
        menu.ModifyMenu(SM_PLAY, MF_STRING, SM_PLAY, "P&ause\tCtrl+A");
        //menu.EnableMenuItem(SM_STOP, MF_ENABLED);
         menu.ModifyMenu(SM_STOP, MF_STRING, SM_STOP, "S&top\tCtrl+S");
        // Make sure the Play/Stop toggle menu knows we're Running.
        //
        Running = TRUE;
```

```
        // Start a timer to show our progress through the waveform file.
        //
        TimeGoing = SetTimer(TIMER_ID, 128, 0);

        // Give enough information to the scrollbar to monitor the
        // progress and issue a re-MCI_OPEN.
        //
        SoundBar->SetName(ElementName);

    } else {
        if (!Pause) {
            // Pause the playing.
            //
            MciGenParm.dwCallback = 0;
            mciSendCommand(DeviceId, MCI_PAUSE, MCI_WAIT, MCI_PARM2
(MciGenParm));

            // Toggle Pause menu to Resume.
            //
            TMenu menu(HWindow);
            menu.ModifyMenu(SM_PLAY, MF_STRING, SM_PLAY, "&
Play\tCtrl+P");
            Pause = TRUE;

        } else {
            // Resume the playing.
            //
            MciGenParm.dwCallback = 0;
```

```
                mciSendCommand(DeviceId, MCI_RESUME, MCI_WAIT, MCI_PARM2
(MciGenParm));

                // Toggle Resume menu to Pause.
                //
                TMenu menu(HWindow);
                menu.ModifyMenu(SM_PLAY, MF_STRING, SM_PLAY, "
P&ause\tCtrl+A");
                Pause = FALSE;
        }
 }
}

void
TSoundWindow::CmStopMidi()
{
 StopMidi();
}

void
TSoundWindow::StopMCI()
{
 if (TimeGoing)      // if Timer is Running, then kill it now.
        TimeGoing = !KillTimer(TIMER_ID);

 // Stop playing the waveform file and close the waveform device.
 //
 MciGenParm.dwCallback = 0;
 mciSendCommand(DeviceId, MCI_STOP, MCI_WAIT, MCI_PARM2(MciGenParm));
```

```
 mciSendCommand(DeviceId, MCI_CLOSE, MCI_WAIT,MCI_PARM2(MciGenParm));

 Running = FALSE;
 DeviceId = 0;
}

//
// Reset the menus to Play menu and gray the Pause menu.
//
void
TSoundWindow::StopMidi()
{
 if (DeviceId) {
        StopMCI();
        TMenu menu(HWindow);
        menu.ModifyMenu(SM_PLAY, MF_STRING, SM_PLAY, "&Play\tCtrl+P");
        //aiy

 }
}

void
TSoundWindow::CmFileOpen()
{
 static TOpenSaveDialog::TData data (
        OFN_HIDEREADONLY|OFN_FILEMUSTEXIST,
        "Midi Files (*.MID)|*.mid| WAVE Files (*.WAV)|*.wav|",
        0,
        0,
```

```
        "MID"
    );
    if (TFileOpenDialog(this, data).Execute() == IDOK) {
        if (CanClose()) {
            strcpy(ElementName, data.FileName);  // Remember the wave file to open.

            // Turn the Play menu on.
            TMenu(HWindow).EnableMenuItem(SM_PLAY, MF_ENABLED);

            MidiLength = 0;
            MidiRatio = 0;
            SoundBar->SetPosition(0);
            Invalidate();
        }
    }
}

void
TSoundWindow::CmFileExit()
{
  CloseWindow();
}

//
// Response function MM_MCINOTIFY message when MCI_PLAY is complete.
//
LRESULT
TSoundWindow::MciNotify(WPARAM, LPARAM)
{
```

```
 if (!FlushNotify) {       // Internal STOP/CLOSE, from thumb re-pos?
        StopMidi();

        // Make sure the thumb is at the end. There could be some WM_TIMER
        // messages on the queue when we kill it, thereby flushing WM_TIMER's
        // from the message queue.
        //
        int   loVal, hiVal;
        SoundBar->GetRange(loVal, hiVal);
        SoundBar->SetPosition(hiVal);

 } else
        FlushNotify = FALSE;       // Yes, so ignore the close.
 return 0;
}


void
TSoundWindow::CmHelpAbout()
{
 MessageBox("Midi Karaoke Ver. 1.0 "
        "for files (*.MID) Play/Pause, Stop, "
        "a MidiForm device SoundBlaster, etc.).",
        "About Midi Karaoke", MB_ICONINFORMATION);
}
void
TSoundWindow::CmContent()
{
 MessageBox("Midi Karaoke Ver. 1.0 "
```

```
        """,
        "คู่มือการใช้งาน", MB_OK);
}

void
TSoundWindow::EvTimer(UINT)
{
 if (!FlushNotify) {          // Internal STOP/CLOSE, from thumb re-pos?
        MciStatusParm.dwCallback = 0;   // No, normal close.
        MciStatusParm.dwItem = MCI_STATUS_LENGTH;
        mciSendCommand(DeviceId, MCI_STATUS, MCI_STATUS_ITEM,
MCI_PARM2(MciStatusParm));

        if (MidiLength != MciStatusParm.dwReturn) {
            Invalidate();  // First time it's different update the scrollbar nums
            MidiLength = MciStatusParm.dwReturn;
        }

        // Compute the length and ratio and update SoundBar info.
        //
        MidiRatio = int((MidiLength + 32000/2) / 32000);
        if (!MidiRatio)
            MidiRatio = 1;
        SoundBar->SetInfo(MidiRatio, MidiLength);
        SoundBar->SetRange(0, int(MidiLength / MidiRatio));
        SoundBar->SetRuler(int((MidiLength / MidiRatio) / 10));

        // Update the current position.
        //
```

```
        MciStatusParm.dwCallback = 0;
        MciStatusParm.dwItem = MCI_STATUS_POSITION;
        mciSendCommand(DeviceId, MCI_STATUS, MCI_STATUS_ITEM, MCI_PARM2
(MciStatusParm));

        SoundBar->SetPosition(int(MciStatusParm.dwReturn / MidiRatio));
 }

 FlushNotify = FALSE;        // Yes, ignore this close.
}

//------------------------------------------------------------------------

class TSoundApp : public TApplication {
 public:
        TSoundApp() : TApplication() {}
        void InitMainWindow() {
                MainWindow = new TSoundWindow(0, "MIDI KARAOKE Ver 1.0");
                EnableCtl3d();
        }
};

int
OwlMain(int /*argc*/, char* /*argv*/ [])
{
 return TSoundApp().Run();
}
```

# โปรแกรม

# MCISOUND.RC

```
//-----------------------------------------------------------------------
//        MCISOUND.RC
//-----------------------------------------------------------------------
#ifndef WORKSHOP_INVOKED
#include <windows.h>
#endif
#include "mcisound.h"

ID_MENU MENU
{
 POPUP "&File"
 {
  MENUITEM "&Open Midi", SM_OPEN
  MENUITEM SEPARATOR
  MENUITEM "E&xit", SM_EXIT
 }

 POPUP "&Control"
 {
  MENUITEM "&Play", SM_PLAY, GRAYED
  MENUITEM "&Stop", SM_STOP, GRAYED
  MENUITEM "&Volume", SM_VOL, GRAYED
 }

 POPUP "&Help"
 {
  MENUITEM "&Content", SM_CONTENT
  MENUITEM SEPARATOR
```

```
 MENUITEM "&About", SM_ABOUT
 }

}

IDA_SOUNDPLY ACCELERATORS
{
 "^o", SM_OPEN
 "^r", SM_REWIND
 "^p", SM_PLAY
 "^a", SM_PLAY
 "^s", SM_STOP, ASCII
}

#define NO_BR_VSLIDERTHUMB
#include <owl\slider.rc>

ICON_2 ICON "kara1.ico"
ID_MENU ACCELERATORS
{
 "^O", SM_OPEN
 "c", SM_CONTENT, ASCII, ALT
 "p", SM_PLAY, ASCII, ALT
 "s", SM_STOP, ASCII, ALT
 "a", SM_ABOUT, ASCII, ALT
 "r", SM_REWIND, ASCII, ALT
 "x", SM_EXIT, ASCII, ALT
 "v", SM_VOL, ASCII, ALT
}
```

# โปรแกรม

# MCISOUND.H

```
//-----------------------------------------------------------------------

//        MCISOUND.H

//-----------------------------------------------------------------------


#define ICON_2          2

#define ID_MENU   200     // Sound Play Menus

#define SM_VOL          304

#define SM_CONTENT          1


#define SM_OPEN         201     // Open menu

#define SM_EXIT         202


#define SM_REWIND 301     // Control menu

#define SM_PLAY   302

#define SM_STOP   303


#define SM_ABOUT  401     // Help menu


#define IDA_SOUNDPLY 501
```