บทที่ 3

ไมโครซอฟต์วิชวลเบสิก สำหรับวินโดวส์

ลักษณะของวิชวลเบสิก

วิชวลเบสิก (VISUAL BASIC) นับเป็นผลิตภัณฑ์ที่นำมาซึ่งการปฏิวัติโปรแกรมใช้งานทีเดียว (และกำลังเป็นที่ถูกจับตามองว่า อาจเป็นมาตรฐานทางด้านการเขียนโปรแกรมแบบวิชวลหรือไม่) ทั้งนี้เนื่องจากเป็นโปรแกรมใช้งานตัวหนึ่ง ที่เปลี่ยนวิธีการคิดและวิธีการโปรแกรมของนักเขียนโปรแกรมเสียใหม่ โดยจะมีชุดออกแบบส่วนเชื่อมประสานกับผู้ใช้และภาษาเบสิก ซึ่งช่วยทำให้การเขียนโปรแกรมเป็นไปได้ง่ายมากขึ้น นอกจากนี้ยังนำมาซึ่งความสนุกเพลิดเพลินอีกด้วย ซึ่งเป็นคุณสมบัติหนึ่งที่หาได้ยากในโปรแกรมใช้งานเพื่อการเขียนโปรแกรมทั่วไป

ในการเขียนโปรแกรมด้วยวิชวลเบสิกนั้น จะเริ่มจากออกแบบหน้าจอบนวินโดวส์ หรือในวิชวลเบสิกเรียกว่า ฟอร์ม ในฟอร์มจะประกอบด้วยวัตถุต่าง ๆ ที่จะทำงานด้วยกัน วัตถุต่าง ๆ นี้ในวิชวลเบสิก เรียกว่า คอนโทรล เช่น ข้อความ , ช่องรับข้อความ , ปุ่มควบคุมต่าง ๆ เป็นต้น เมื่อนำวัตถุเหล่านี้จากหน้าต่างกล่องเครื่องมือ (Toolbox) มาวางประกอบกัน เป็นหน้าตาของจอภาพแล้ว จึงค่อยมาระบุว่าวัตถุแต่ละอย่างจะทำงานอย่างไร โดยเขียนโปรแกรมประกอบเข้าไปกับวัตถุเหล่านี้ วัตถุแต่ละตัวจะมีเหตุการณ์เกิดขึ้นกับมันได้หลายอย่าง ถ้าเราต้องการให้วัตถุมันตอบสนองต่อเหตุการณ์ใด ก็เขียนโปรแกรมภาษาเบสิกระบุการทำงานไว้กับเหตุการณ์นั้น เช่น ถ้าเป็นปุ่มควบคุม แล้วเราต้องการให้ทำงาน เมื่อมีการคลิกหรือดับเบิ้ลคลิกเมาส์ ก็ระบุว่าหากมีการคลิกที่ปุ่มควบคุมนี้ จะต้องทำอย่างไร หรือถ้ามีการดับเบิ้ลคลิกจะต้องทำอย่างไร เหตุการณ์ที่ไม่ได้ระบุไว้ ก็จะไม่มีผลต่อวัตถุนั้น

ในทุกวัตถุจะมีลักษณะหรือคุณสมบัติของตัวเอง เช่น วัตถุช่องรับข้อความจะมีชื่อที่ใช้แทนตัวเองในการเขียนโปรแกรม มีความกว้าง ความสูง และสีของช่องรับข้อความ โดยเราสามารถอ้างถึง หรือเปลี่ยนคุณสมบัติเหล่านี้ได้จากการเขียนโปรแกรม หรือโดยใช้หน้าต่างคุณสมบัติของวัตถุแต่ละอัน

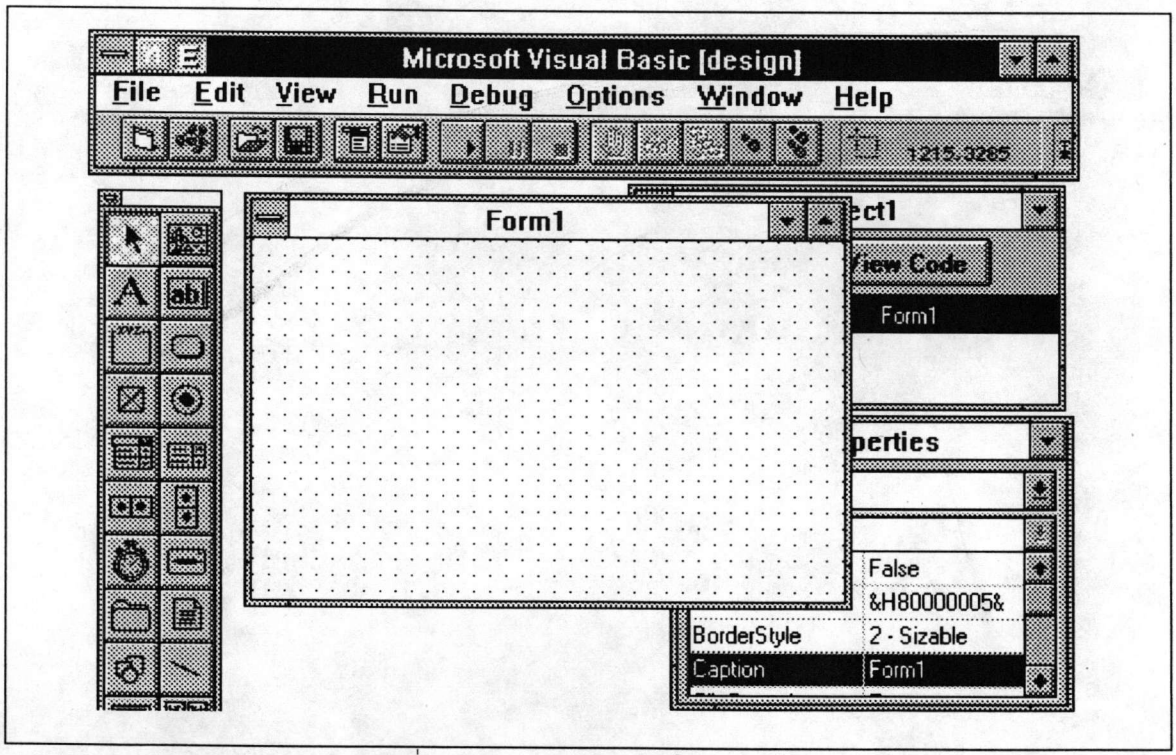
อย่างไรก็ตาม ภาษาเบสิกของซอฟต์แวร์ตัวนี้ จะไม่เหมือนภาษาเบสิกมาตรฐานทั่วไปเสียทีเดียว ทั้งนี้เพราะมีคำสั่งบางคำสั่งหายไป ในขณะที่บางคำสั่งก็ได้เพิ่มเติมเข้ามา

สรุปแล้วผู้ที่เขียนโปรแกรมบนวิชวลเบสิกนี้ ไม่จำเป็นต้องรู้เรื่องวิธีการเขียนโปรแกรมในลักษณะเชิงวัตถุ

(Object-Oriented Language) หรือเรื่องการทำงานภายในของวินโดวส์เลยแม้แต่น้อย ก็สามารถที่จะเขียนโปรแกรมสำหรับวินโดวส์ได้ ซึ่งเมื่อเขียนโปรแกรมบนวิชวลเบสิกเสร็จแล้ว ก็สามารถแปลโปรแกรมให้เป็นแฟ้มข้อมูลที่สามารถทำงานได้เดี่ยว ๆ เป็นแฟ้มนามสกุล .EXE เป็นเหมือนอีกโปรแกรมประยุกต์หนึ่งบนวินโดวส์เลย มีความสามารถในทุกด้านที่โปรแกรมประยุกต์หนึ่ง ๆ บนวินโดวส์ทำได้ ไม่ว่าจะเป็นเรื่องรายการเลือกแบบดึงลง ช่องรับข้อความ การย่อขยายขนาดของวินโดวส์หรืออื่น ๆ

สภาพแวดล้อมในการพัฒนาโปรแกรมด้วยวิชวลเบสิก

เมื่อวิชวลเบสิกเริ่มทำงาน จะเห็นหน้าต่างแสดงลิขสิทธิ์ และชื่อเจ้าของชุดสำเนาของวิชวลเบสิก จากนั้นจะเห็นสภาพแวดล้อมต่าง ๆ ภายในวิชวลเบสิก ดังรูป 3.1



รูปที่ 3.1 สภาพแวดล้อมภายในวิชวลเบสิก

หน้าจอจะประกอบด้วย 7 ส่วนสำคัญ คือ

1. แท่งแสดงชื่อโปรแกรมประยุกต์ (Title Bar)

เป็นแท่งในแนวนอนอยู่บนสุดของหน้าจอ ซึ่งทุกโปรแกรมประยุกต์สำหรับวินโดวส์จะมีแท่งนี้เสมอ แรกเริ่มแท่งนี้จะแสดงคำว่า

Microsoft Visual Basic [design]

หมายความว่า ขณะนี้เป็น ช่วงเวลาการออกแบบ (Design Time) จอภาพและเขียนโปรแกรมประกอบเข้าไปกับวัตถุต่าง ๆ ให้ทำงานตามเหตุการณ์ที่เกิดขึ้น โดยทุกวัตถุจะมีคุณสมบัติเฉพาะที่สามารถเปลี่ยนแปลงค่าได้ เมื่อเราสั่งดำเนินงานโปรแกรมที่ได้ออกแบบไว้ แท่งแสดงชื่อโปรแกรมจะเปลี่ยนเป็น

Microsoft Visual Basic [run]

หมายความว่า ขณะนี้เป็นช่วงเวลาดำเนินงาน (Run Time) และเมื่อต้องมีการแก้ไขจุดบกพร่อง (Debug) โปรแกรมจะหยุดทำงานชั่วคราว แล้วที่แท่งแสดงชื่อโปรแกรมจะเปลี่ยนเป็น

Microsoft Visual Basic [break]

2. แท่งรายการเลือก (Menu Bar)

เราสามารถใช้คำสั่งต่าง ๆ ทำการพัฒนา , ทดสอบ และบันทึกโปรแกรมไว้ รายการเลือก File มีคำสั่งที่ใช้ทำงานกับแฟ้มต่าง ๆ ที่ใช้สร้างโปรแกรมประยุกต์ และที่สำคัญคือใช้เพิ่มคอนโทรลใหม่ได้ (แฟ้มสกุล .VBX). ด้วยคำสั่ง Add File ของรายการเลือกนี้ รายการเลือก Edit ใช้ในการแก้ไขโปรแกรมที่เราเขียน และมีเครื่องมือในการค้นหาและแทนที่ตัวอักษร รายการเลือก View ใช้ในการเข้าถึงส่วนต่าง ๆ ของโปรแกรม รายการเลือก Run ใช้ทดสอบโปรแกรมที่สร้างขึ้น รายการเลือก Debug เป็นเครื่องมือในการแก้ไขข้อผิดพลาด (Bug) ของโปรแกรม รายการเลือก Option ใช้ควบคุมสภาพแวดล้อมของการพัฒนาโปรแกรม รายการเลือก Window ใช้เข้าถึงหน้าต่างต่าง ๆ ภายในวิชวลเบสิก ใช้ออกแบบเมนูสำหรับโปรแกรมของเรา ส่วนรายการเลือก Help ใช้ขอความช่วยเหลือจากวิชวลเบสิก (Cornell, 1993)

ทุกรายการเลือกจะมีอักษรตัวหนึ่งขีดเส้นใต้ไว้ การกด [Alt] แล้วตามด้วยการกดตัวอักษรที่ขีดเส้นใต้จะเป็นการเปิดรายการเลือกนั้น ๆ ขึ้นมาในลักษณะของรายการเลือกแบบดึงลง

3. แท่งเครื่องมือ (Toolbar)

แท่งเครื่องมือจะอยู่ใต้แท่งรายการเลือก เมื่อคลิกเมาส์ที่แต่ละสัญรูปในแท่งเครื่องมือ จะเป็นการเลือกการทำงานบางอย่างแทนการใช้รายการเลือก สัญรูปจากซ้ายไปขวาจะทำงานแตกต่างกันดังนี้

New Form ใช้เพิ่มฟอร์มใหม่ให้กับโปรแกรม

New Module ใช้เพิ่มโมดูล

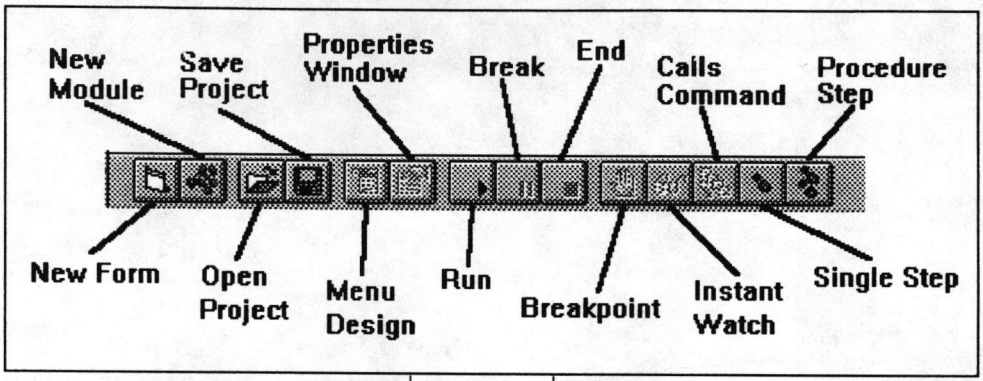
Open Project ใช้สร้างโปรเจ็กต์ใหม่

Save Project ใช้บันทึกแฟ้มโปรเจ็กต์ลงดิสค์

Menu Design ใช้ออกแบบรายการเลือกของโปรแกรม

Activate Properties Window สามารถเปลี่ยนแปลงค่าของคุณสมบัติภายในหน้าต่างคุณสมบัติได้

Run สั่งดำเนินงานโปรแกรม



รูปที่ 3.2 แท่งเครื่องมือ

Break หยุดการทำงานของโปรแกรมชั่วคราว

End หยุดการทำงานของโปรแกรม

Breakpoint กำหนดจุดหยุดชั่วคราวของโปรแกรม เมื่อมีการสั่งดำเนินงานโปรแกรม

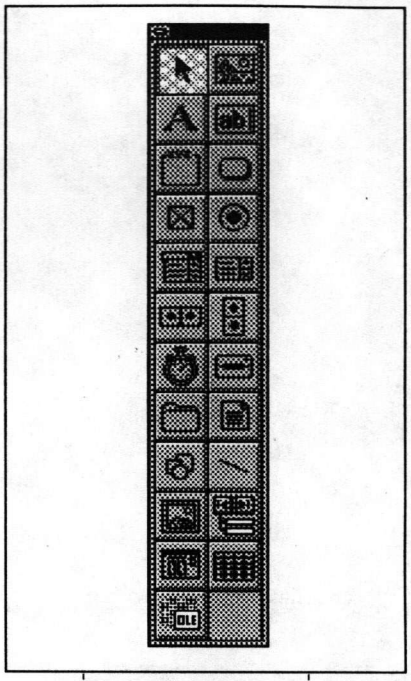
Instant Watch ใช้ดูค่าต่าง ๆ ของตัวแปร โดยการหยุดการทำงานของโปรแกรมไว้ชั่วคราว

Calls Command แสดงรายชื่อของกระบวนงาน (Procedure) ภายในโปรแกรม

Single Step ดำเนินงานโปรแกรมทีละบรรทัด

Procedure Step ดำเนินงานโปรแกรมทีละบรรทัด และให้ถือว่า 1 กระบวนงาน คือ 1 บรรทัด ด้วยเช่นกัน

4. หน้าต่างกล่องเครื่องมือ (Toolbox)



รูปที่ 3.3 หน้าต่างกล่องเครื่องมือ

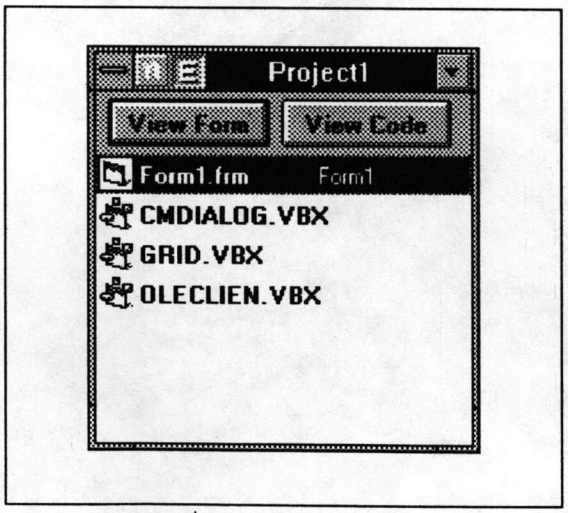
อยู่ทางด้านซ้ายของจอและทางด้านล่างของแท่งเครื่องมือ ซึ่งจะมีวัตถุหรือคอนโทรลพื้นฐานสำหรับการพัฒนาโปรแกรมอยู่ 22 แบบ โดยแต่ละสัญรูปจะแทนคอนโทรลแต่ละแบบ เราสามารถสร้างวัตถุหรือคอนโทรลขึ้นมาใหม่ได้ คอนโทรลนั้นเรียกว่า CUSTOM CONTROL และจะเก็บอยู่ในแฟ้มสกุล .VBX เราใช้หน้าต่างกล่องเครื่องมือนี้ในการสร้างปุ่มคำสั่ง สร้างข้อความ และคอนโทรลต่าง ๆ ของโปรแกรม และเมื่อใช้คำสั่ง Add File ของรายการเลือก File เพิ่มแฟ้มสกุล .VBX แล้ว จะได้สัญรูปเพิ่มขึ้นตามจำนวนของแฟ้ม .VBX ที่ถูกเพิ่มเข้ามา และชื่อของแฟ้ม .VBX นั้นจะไปปรากฏในหน้าต่างโปรเจ็กต์ด้วย

5. หน้าต่างฟอร์มเริ่มต้น (The Initial Form Window)

จะอยู่ประมาณกลางจอภาพ เราจะใช้หน้าต่างฟอร์มนี้ในการกำหนดหน้าตาของหน้าต่างของโปรแกรม คำว่า "ฟอร์ม" ของวิชวลเบสิกจึงหมายถึง "หน้าต่าง" นั่นเอง

6. หน้าต่างโปรเจ็กต์ (Project Window)

แต่ละโปรแกรมของวิชวลเบสิก สามารถใช้ชุดคำสั่งและฟอร์มที่สร้างขึ้นก่อนแล้วร่วมกันได้ โดยอยู่ในลักษณะที่เรียกว่า โปรเจ็กต์ ในโปรเจ็กต์หนึ่ง ๆ สามารถมีได้หลายฟอร์ม และชุดคำสั่งที่เกี่ยวกับคอนโทรลภายในฟอร์ม จะถูกเก็บแยกเป็นแฟ้มต่างหากจากแฟ้มที่เก็บฟอร์ม (มีสกุลเป็น .FRM) ตัวโปรแกรมก็อาจมีหลายโมดูล ซึ่งแต่ละโมดูลก็อาจจะเก็บแยกเป็นแฟ้มต่างหากได้อีกเช่นกัน (มีสกุลเป็น .BAS)

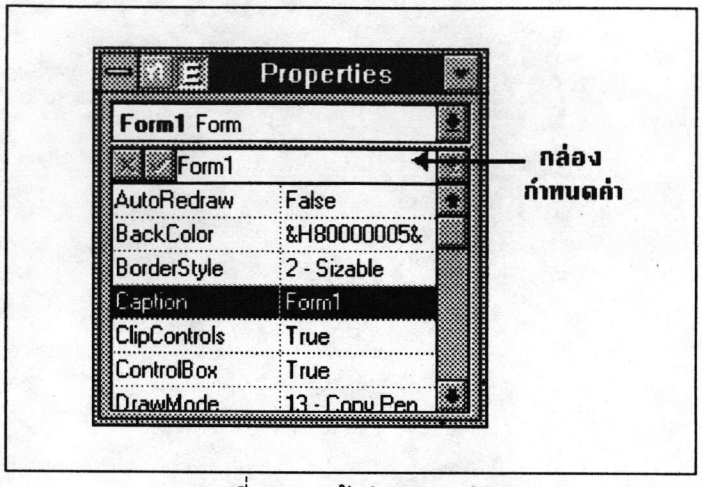


รูปที่ 3.4 หน้าต่างโปรเจ็กต์

ดังนั้นจึงมีหลายแฟ้มข้อมูลที่ใช้ในการสร้างโปรแกรมประยุกต์ จึงต้องมีแฟ้มหนึ่งใช้เก็บรายละเอียดของแฟ้มเหล่านั้น เรียกว่า แฟ้มสร้างโปรเจ็กต์ (Project Make File) (มีสกุลเป็น .MAK)

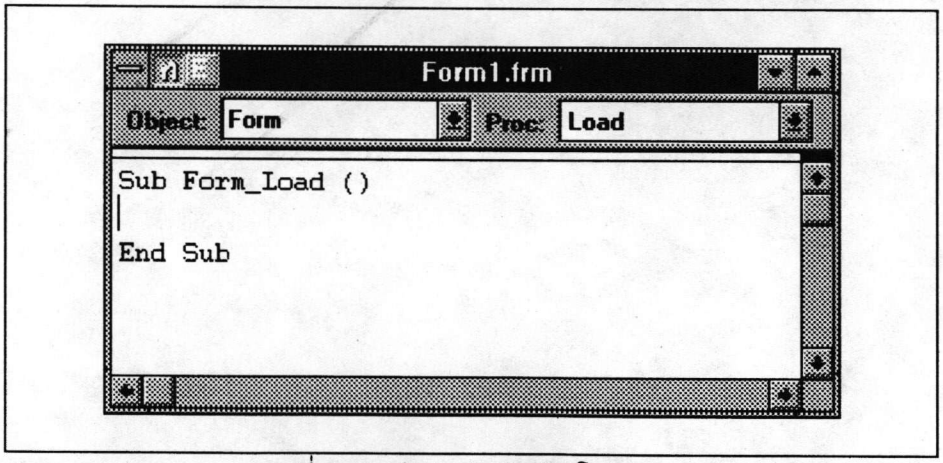
7. หน้าต่างคุณสมบัติ (Properties Window)

ในการเปลี่ยนแปลงค่าของคุณสมบัติ ทำได้ 2 วิธี โดยในตอนแรกใช้ปุ่มลูกศรเลื่อนขึ้นลงหาคุณสมบัติที่ต้องการก่อน เมื่อพบแล้วให้พิมพ์ค่าใหม่ได้เลย เสร็จแล้วกด [Enter] หรือใช้การคลิกเมาส์ที่กล่องกำหนดค่า (Setting Box) ภายในหน้าต่างคุณสมบัติก่อน แล้วจึงค่อยพิมพ์ค่าใหม่เข้าไปก็ได้



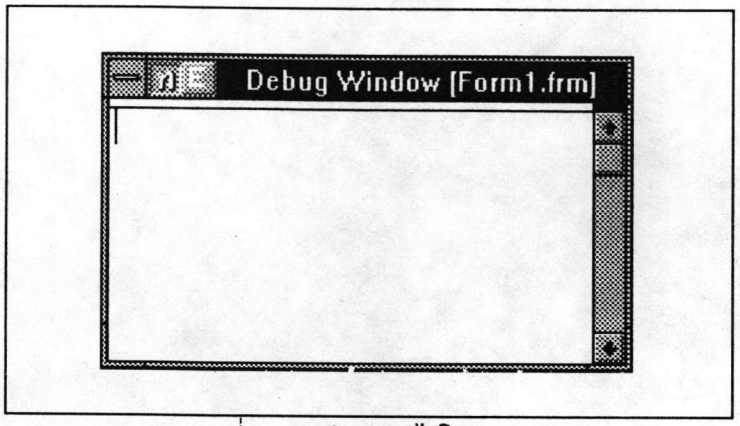
รูปที่ 3.5 หน้าต่างคุณสมบัติ

เมื่อเราดับเบิ้ลคลิกที่หน้าต่างฟอร์มหรือที่คอนโทรลใด ๆ ในฟอร์มในช่วงเวลาการออกแบบ ก็จะปรากฏหน้าต่างสำหรับเขียนโปรแกรม (Code Window) ให้กับเหตุการณ์ของฟอร์ม หรือคอนโทรลนั้น



รูปที่ 3.6 หน้าต่างสำหรับเขียนโปรแกรม

นอกจากนี้ยังมี หน้าต่างแก้ไขโปรแกรม (Debug Window) ที่ใช้ตรวจสอบและแก้ไขข้อผิดพลาดของโปรแกรม ซึ่งปรากฏขึ้นภายหลังเมื่อมีการสั่งดำเนินงานโปรแกรม



รูปที่ 3.7 หน้าต่างแก้ไขโปรแกรม

สามารถศึกษารายละเอียดอื่น ๆ ที่เกี่ยวกับการเขียนโปรแกรมด้วยวิชวลเบสิกจาก Cornell, G. 1993. The Visual Basic 3 for Windows Handbook. Berkeley, U.S.A.: Osborne McGraw-Hill. หรือหนังสือเล่มอื่นที่เกี่ยวข้องกับวิชวลเบสิกสำหรับวินโดวส์

คอนโทรลที่สร้างขึ้นใหม่ (CUSTOM CONTROL)

วิธีการสร้างคอนโทรลขึ้นใหม่นั้น จะคล้าย ๆ กับการสร้างโปรแกรมประยุกต์สำหรับวินโดวส์ คือ ต้องจัดการกับข้อความต่าง ๆ และต้องมีการเรียกใช้ฟังก์ชั่น API แต่ก็มีข้อแตกต่างกัน คือ

- คอนโทรลที่สร้างใหม่นั้นจะเก็บเป็นแฟ้มข้อมูลแบบ DLL ซึ่งจะมีวิธีการนำเข้าหน่วยความจำที่แตกต่างไปจากโปรแกรมประยุกต์ธรรมดา
- คอนโทรลที่สร้างใหม่นี้ สร้างขึ้นสำหรับวิชวลเบสิก ดังนั้นจึงมีรูปแบบที่ใช้ในการติดต่อกันโดยเฉพาะ

1. โครงสร้างพื้นฐานของคอนโทรลคลาส (CONTROL CLASS)

คอนโทรลที่สร้างขึ้นมาใหม่ มีความหมายเช่นเดียวกับคำว่า คอนโทรลคลาส โดยคอนโทรลคลาสถูกนำเข้าสู่หน่วยความจำเป็นส่วนหนึ่งของโปรเจ็กต์ จากนั้นผู้พัฒนาโปรแกรมประยุกต์ด้วยวิชวลเบสิกก็สามารถสร้างตัวตน (Instance) ของคลาสนี้ขึ้นมากี่ตัวตนก็ได้ หากมีหน่วยความจำเพียงพอ

จะเข้าใจลักษณะของคอนโทรลคลาสได้ง่ายขึ้น โดยให้นึกถึงคอนโทรลมาตรฐาน (STANDARD CONTROL) ในวิชวลเบสิก แต่ละสัญรูปในหน้าต่างกล่องเครื่องมือ ยกเว้นเครื่องมือตัวชี้ (Pointer) หมายถึงแต่ละคลาส เช่น ผู้พัฒนาโปรแกรมประยุกต์สามารถสร้างช่องรับข้อความขึ้นมากี่อันก็ได้ แต่ทุกอันล้วนเป็นตัวตนของคลาสช่องรับข้อความ

(ต่อไปขอให้ดูโครงร่างของแฟ้ม BISEARCH.H และ BISEARCH.C ที่ท้ายบทนี้พร้อมกันไปด้วย)

คอนโทรลคลาส ประกอบด้วยส่วนสำคัญ 2 ส่วน คือ ต้นแบบ (CONTROL MODEL) และกระบวนงาน (CONTROL PROCEDURE)

1.1 ต้นแบบของคอนโทรล เป็นโครงสร้างของภาษาซีที่กำหนดลักษณะพื้นฐานของคอนโทรลคลาส (BISEARCH.H บรรทัดที่ 178 - 195) เช่น ชื่อของคอนโทรลโดยปริยาย , ชื่อของคลาสที่ให้กำเนิด (PARENT CLASS) คอนโทรล และมีตัวชี้ (Pointer) ไปยังตารางที่สำคัญมาก 2 ตาราง (BISEARCH.H บรรทัดที่ 190 - 191) คือ

1. ตารางข้อมูลคุณสมบัติ (PROPERTY INFORMATION TABLE) เก็บรายละเอียดของคุณสมบัติทั้งหมดของคอนโทรล (BISEARCH.H บรรทัดที่ 109 - 126)

2. ตารางข้อมูลเหตุการณ์ (EVENT INFORMATION TABLE) เก็บรายละเอียดของเหตุการณ์ และอาร์กิวเมนต์ (Argument) ทั้งหมดที่ใช้ในคอนโทรล (BISEARCH.H บรรทัดที่ 158 - 170)

1.2 กระบวนงานของคอนโทรล (BISEARCH.C บรรทัดที่ 19 - 107 ในที่นี้ คือ TreeCtlProc) ทำหน้าที่เหมือนกับกระบวนงานของวินโดวส์ (WINDOW PROCEDURE) ของโปรแกรมประยุกต์สำหรับวินโดวส์ กล่าวคือ จะรับข้อความ (Message) ต่าง ๆ และจัดการกับข้อความเหล่านั้น (BISEARCH.C บรรทัดที่ 30 - 106) ตลอดจนกำหนดว่า เหตุการณ์แต่ละอย่างจะเกิดขึ้นเมื่อไร และต้องทำหน้าที่วาดตัวคอนโทรลบนจอภาพด้วยถ้าตัวคอนโทรลนั้นไม่ได้เป็นคลาสย่อย (SUBCLASS) ของคอนโทรลคลาสของวินโดวส์ (WINDOWS CONTROL CLASS) (BISEARCH.C บรรทัดที่ 80 - 91)

เนื่องจากตัวคอนโทรลคลาสทำงานในลักษณะของ DLL ดังนั้นจึงไม่มีกระบวนงาน WinMain เพราะตัวคอนโทรลจะถูกเชื่อมเข้ากับโปรแกรมประยุกต์ที่กำลังทำงานอยู่ก่อนแล้ว

2. <u>ลักษณะของคอนโทรลคลาส</u>

แฟ้มของคอนโทรลที่สร้างขึ้นใหม่ (สกุล .VBX) สามารถทำงานได้โดย

1. วิชวลเบสิกสั่งดำเนินงานฟังก์ชั่น VBINITCC ซึ่งจะต้องกำหนดให้เป็นกระบวนงานที่วินโดวส์สามารถเรียกใช้ได้ (EXPORTED) ไว้ในแฟ้มของคอนโทรลที่สร้างใหม่ (BISEARCH.C บรรทัดที่ 130 - 139)

2. ฟังก์ชั่น VBINITCC จะทำการลงทะเบียน (Register) ต้นแบบของคอนโทรลด้วยการใช้ฟังก์ชั่น VBRegisterModel พร้อมทั้งผ่านค่าที่อยู่ของโครงสร้างของต้นแบบไปให้ (BISEARCH.C บรรทัดที่ 137) ซึ่ง VBINITCC สามารถลงทะเบียนต้นแบบของคอนโทรลได้หลายครั้งในกรณีที่มีการนำหลาย ๆ คอนโทรลคลาสเข้าสู่หน่วยความจำ และในโครงสร้างต้นแบบของคอนโทรล จะมีตัวชี้ไปยังที่อยู่ของกระบวนงานของคอนโทรล (BISEARCH.H บรรทัดที่ 182) ซึ่งกระบวนงานนี้ก็ต้องกำหนดให้เป็นกระบวนงานที่วินโดวส์สามารถเรียกใช้ได้ด้วยเช่นกัน (BISEARCH.C บรรทัดที่ 19)

3. จากนั้นผู้พัฒนาโปรแกรมด้วยวิชวลเบสิกจะสร้างตัวตนของแต่ละคลาสขึ้นมาเท่าไรก็ได้

4. เมื่อมีข้อความส่งไปยังแต่ละตัวตนของคลาส กระบวนงานของคอนโทรลจะจัดการกับข้อความเหล่านั้น (BISEARCH.C บรรทัดที่ 30 - 106) โดยที่ตัวข้อความที่ถูกส่งไปยังกระบวนงานของคอนโทรล จะมีแฮนเดิล (HANDLE) ของโครงสร้างของคอนโทรล (CONTROL STRUCTURE) ส่งไปด้วย (BISEARCH.C บรรทัดที่ 21) ซึ่งแฮนเดิลของโครงสร้างนี้เองเป็นตัวกำหนดว่าข้อความนั้นเป็นของตัวตนใด

เนื่องจากแฟ้มคอนโทรลที่สร้างใหม่มีลักษณะเป็น DLL ดังนั้นจึงต้องมีกระบวนการเริ่มต้น (INITIALIZATION) DLL 1 ครั้งในตอนที่คอนโทรลถูกนำเข้ามาในหน่วยความจำครั้งแรก ด้วยฟังก์ชั่น LibMain (BISEARCH.C บรรทัดที่ 113 - 124) ซึ่งด้วยลักษณะของ DLL แล้ว จึงสามารถใช้แฟ้มคอนโทรลสกุล .VBX ร่วมกันได้ระหว่างตัวตนต่าง ๆ ของวิชวลเบสิก

ส่วนฟังก์ชั่น VBINITCC อาจถูกเรียกใช้หลายครั้ง โดยจะถูกเรียก 1 ครั้ง เมื่อ 1 ตัวตนของวิชวลเบสิก มีการใช้แฟ้ม .VBX ของคอนโทรลคลาส

สภาพแวดล้อมในการพัฒนาโปรแกรม ก็มีหลายตัวตน เช่น VB.EXE ก็เป็นตัวตนหนึ่งของวิชวลเบสิก

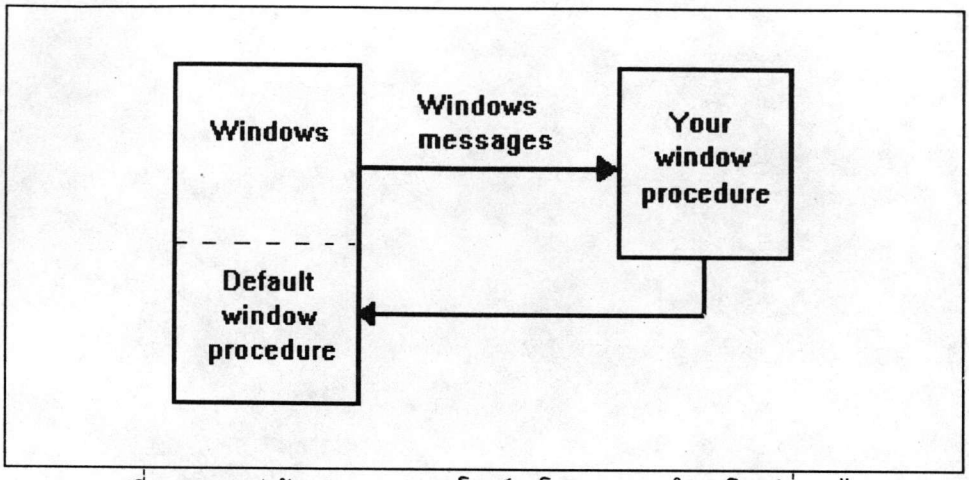
แต่ละโปรแกรมประยุกต์ที่เกิดจากวิชวลเบสิก ก็ถือเป็น 1 ตัวตนของวิชวลเบสิก เพราะฉะนั้นแต่ละโปรแกรมประยุกต์ที่ใช้แฟ้ม .VBX จะเรียกใช้ VBINITCC เพื่อลงทะเบียนให้กับต้นแบบของคอนโทรล

เมื่อโปรแกรมประยุกต์ใดมีการใช้แฟ้มคอนโทรลที่สร้างขึ้นใหม่ ผู้ใช้โปรแกรมประยุกต์นั้นก็ต้องมีสำเนาของแฟ้มคอนโทรล สกุล .VBX นั้นด้วย

3. <u>การส่งข้อความไปยังคอนโทรลของวิชวลเบสิก</u>

ส่วนใหญ่แล้วในการเขียนโปรแกรมประยุกต์สำหรับวินโดวส์ จะต้องเกี่ยวข้องกับการส่ง, รับ และจัดการข้อความต่าง ๆ การสร้างคอนโทรลขึ้นมาใหม่ก็ต้องจัดการกับข้อความเช่นเดียวกัน แต่ก็มีลักษณะบางอย่างที่แตกต่างออกไป

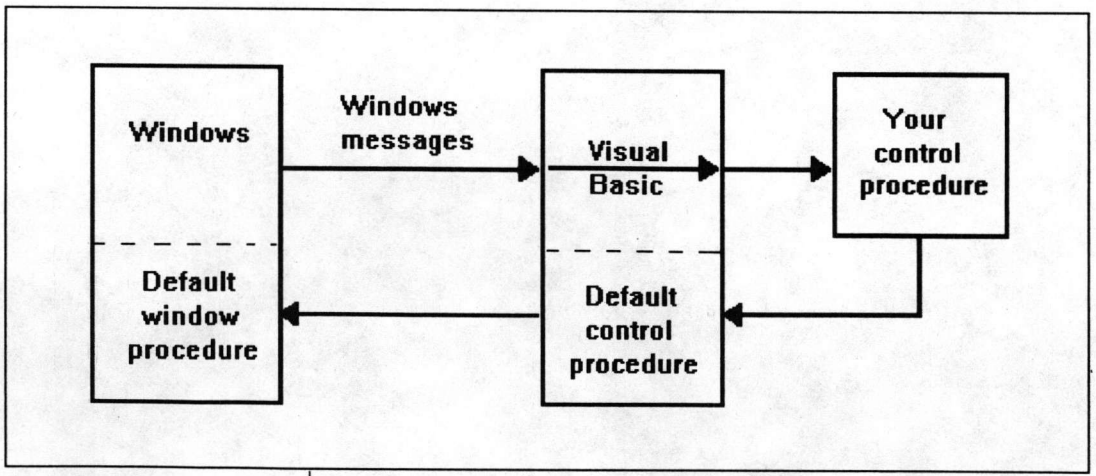
ตัววินโดวส์จะส่งข้อความที่เหมาะสมไปให้กับแต่ละโปรแกรมที่ทำงานอยู่ภายใต้วินโดวส์ ซึ่งกระบวนงานวินโดวส์ของแต่ละโปรแกรม จะจัดการกับข้อความเหล่านี้ หรืออาจส่งข้อความต่อไปยังส่วนประมวลผลข้อความโดยปริยาย (Defalut Message Processor) ดังรูป 3.8



รูปที่ 3.8 การส่งข้อความระหว่างวินโดวส์กับโปรแกรมภายใต้วินโดวส์ทั่ว ๆ ไป

แต่ละข้อความจะเป็นตัวบอกกับกระบวนงานวินโดวส์ว่า เมื่อไรที่หน้าต่างถูกสร้างขึ้นแล้ว เมื่อไรที่ต้องเปลี่ยนแปลงขนาด หรือเคลื่อนย้ายหน้าต่าง และอื่น ๆ หรือ เมื่อมีการกดคีย์บอร์ด หรือเมาส์ ก็จะมีข้อความส่งมายังกระบวนงานวินโดวส์ด้วยเช่นกัน

เมื่อมีการนำคอนโทรลที่สร้างใหม่เข้าสู่หน่วยความจำ วินโดวส์ก็จะส่งข้อความที่เหมือน ๆ กับข้อความที่ส่งไปให้กับโปรแกรมประยุกต์อื่น ๆไปยังคอนโทรลนั้น แต่ตัววิชวลเบสิกจะเป็นผู้รับข้อความนั้นก่อน แล้วจะเลือกว่าจะส่งข้อความนั้นต่อไปยังกระบวนงานของคอนโทรลหรือไม่ ซึ่งกระบวนงานของคอนโทรลเองจะจัดการกับข้อความนั้น หรืออาจส่งข้อความนั้นต่อไปยังส่วนประมวลผลข้อความโดยปริยายของวิชวลเบสิกด้วยการใช้ฟังก์ชั่น VBDefControlProc (BISEARCH.C บรรทัดที่ 105) ดังรูป 3.9



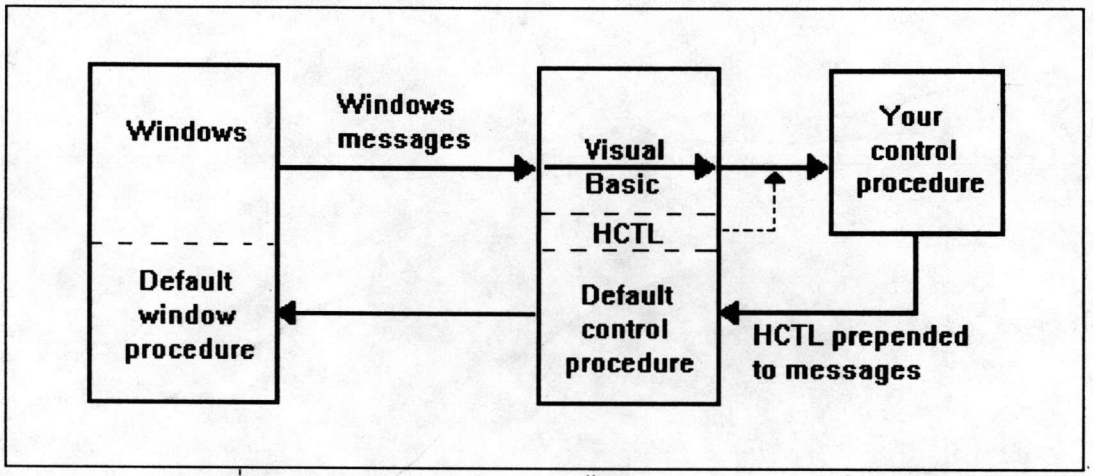
รูปที่ 3.9 การส่งข้อความไปยังกระบวนงานของคอนโทรล

ดังที่กล่าวมาแล้วว่า วิชวลเบสิกจะเป็นผู้เลือกว่าจะส่งข้อความใดไปยังกระบวนงานของคอนโทรล เพราะในบางกรณี ข้อความนั้นมีผลต่อลักษณะของคอนโทรล เช่น ข้อความของคีย์บอร์ดและเมาส์จะไม่ถูกส่งต่อไป

ยังคอนโทรลในช่วงเวลาการออกแบบ เพราะผู้พัฒนาโปรแกรมประยุกต์ด้วยวิชวลเบสิกใช้คีย์บอร์ดและเมาส์ในการเคลื่อนย้าย เปลี่ยนแปลงขนาด และกำหนดคุณสมบัติต่าง ๆ ของคอนโทรล

เมื่อวิชวลเบสิกจะส่งข้อความต่อไปยังกระบวนงานของคอนโทรล วิชวลเบสิกจะส่งแฮนเดิลของคอนโทรลไปด้วย (BISEARCH.C บรรทัดที่ 21) ซึ่งแฮนเดิลนี้จะใช้ในการเข้าถึงโครงสร้างของแต่ละตัวตนของคอนโทรล และโครงสร้างนี้ก็เป็นลักษณะเฉพาะที่ไม่มีในโครงสร้างวินโดวส์ (WINDOW STRUCTURE)

ตัววิชวลเบสิกเองสามารถสร้างข้อความเพื่อติดต่อกับคอนโทรลได้ อาจเป็นการขอข้อมูลจากคอนโทรล หรืออาจต้องการดำเนินงานบางอย่างกับคอนโทรล เช่น เมื่อมีการกำหนดค่าของคุณสมบัติ วิชวลเบสิกจะส่งข้อความ VBM_SETPROPERTY ไปยังคอนโทรล (BISEARCH.C บรรทัดที่ 54 - 65) ขอให้พิจารณารูปที่ 3.10 จะเข้าใจการส่งข้อความไปยังกระบวนงานของคอนโทรลได้ดีขึ้น



รูปที่ 3.10 การส่งข้อความและแฮนเดิลไปยังกระบวนงานของคอนโทรล

4. <u>โครงสร้างของคอนโทรล</u> (CONTROL STRUCTURE)

เราสามารถอ้างถึงแต่ละตัวตนของคอนโทรลได้ 2 วิธี คือ

- จากแฮนเดิลของวินโดวส์ (WINDOWS HANDLE)
- จากแฮนเดิลของคอนโทรล (CONTROL HANDLE)

เราเข้าถึงโครงสร้างของวินโดวส์ที่มีข้อมูลของวินโดวส์ของคอนโทรลได้ โดยใช้แฮนเดิลของวินโดวส์และใช้แฮนเดิลนี้ติดต่อกับ API ของวินโดวส์ด้วย

และเราเข้าถึงโครงสร้างของคอนโทรลโดยผ่านทางแฮนเดิลของคอนโทรล โดยวิชวลเบสิกจะจองเนื้อที่ให้กับโครงสร้างของคอนโทรลของแต่ละตัวตน เมื่อแต่ละตัวตนถูกสร้างขึ้นมา โครงสร้างนี้จะเก็บสถานะต่าง ๆ โดยเฉพาะอย่างยิ่งคุณสมบัติเฉพาะตัวของแต่ละตัวตนของคอนโทรล (BISEARCH.H บรรทัดที่ 178 - 195)

เราใช้แฮนเดิลของคอนโทรลในการติดต่อกับ API ของวิชวลเบสิก ซึ่งตัว API จะมีวิธีจัดการกับคอนโทรลเองโดยใช้โครงสร้างของคอนโทรล มีอีกวิธีหนึ่งที่จะใช้แฮนเดิลของคอนโทรล ก็คือการอ้างถึง

(Dereference) โดยใช้ฟังก์ชั่น VBDerefControl (BISEARCH.H บรรทัดที่ 72) ซึ่งจะคืนค่าตัวชี้ไปยังโครงสร้างของผู้เขียนโปรแกรม (Programmer-defined Structure) ของแต่ละตัวตนของคอนโทรล (BISEARCH.H บรรทัดที่ 59 - 68)

เราใช้โครงสร้างของผู้เขียนโปรแกรม ในการเก็บค่าของคุณสมบัติต่าง ๆ ที่สร้างขึ้นมาใหม่ ซึ่งในต้นแบบของคอนโทรลจะมีการกำหนดถึงจำนวนไบต์ที่ต้องใช้สำหรับโครงสร้างของผู้เขียนโปรแกรม (BISEARCH.H บรรทัดที่ 185) และจะต้อง**ไม่**ใช้ตัวแปรที่เป็น Static หรือ Global ในโครงสร้างของผู้เขียนโปรแกรมด้วย

5. <u>การติดต่อกันระหว่างวิชวลเบสิกกับคอนโทรล</u>

วิชวลเบสิกจะติดต่อกับทั้งต้นแบบและกระบวนงานของคอนโทรล โดยมีการติดต่อกันที่สำคัญ ได้แก่ การติดต่อกันเมื่อมีการสร้างและวาดตัวคอนโทรล การติดต่อกันเกี่ยวกับคุณสมบัติ การติดต่อกันเกี่ยวกับเหตุการณ์

5.1 การติดต่อกันเมื่อมีการสร้างและวาดตัวคอนโทรล

เมื่อตัวตนของคอนโทรลคลาสถูกสร้างขึ้น วิชวลเบสิกจะสร้างแฮนเดิลของคอนโทรลให้กับตัวตน จากนั้นจึงสร้างโครงสร้างวินโดวส์ ซึ่งจะมีแฮนเดิลของวินโดวส์เกิดขึ้นด้วย (ยกเว้นในกรณีของคอนโทรลที่เป็นรูปภาพ) แล้ววินโดวส์จึงส่งข้อความ WM_NCCREATE และ WM_CREATE มายังกระบวนงานของคอนโทรล (BISEARCH.C บรรทัดที่ 32 - 38)

คุณสมบัติส่วนใหญ่จะถูกนำเข้าสู่หน่วยความจำทันทีที่หน้าต่างถูกสร้างขึ้น โดยอาจจะเป็นในตอนที่มีการนำฟอร์มเข้าสู่หน่วยความจำ หรือมีการวาง (Paste) คอนโทรลมาจากคลิปบอร์ด (Clipboard)

สุดท้ายวิชวลเบสิกจะแสดงรูปร่างของคอนโทรล โดยการเรียกฟังก์ชั่น ShowWindow ซึ่งจะทำให้กระบวนงานของคอนโทรลได้รับข้อความ WM_PAINT (หรือ VBM_PAINT ถ้าเป็นคอนโทรลที่เกี่ยวกับรูปภาพ) (BISEARCH.C บรรทัดที่ 80 - 91) ซึ่งรูปร่างของคอนโทรลจะเป็นอย่างไรนั้นก็จะจัดการกันที่ข้อความ WM_PAINT นี้เหมือนกับโปรแกรมประยุกต์สำหรับวินโดวส์อื่น ๆ แต่ถ้าเป็นคลาสย่อย (SUBCLASS) ของคอนโทรลคลาสของวินโดวส์ การแสดงรูปร่างของคอนโทรล เราไม่ต้องจัดการเอง ตัววินโดวส์จะจัดการให้

ส่วนการแสดงสัญรูปของคอนโทรลในหน้าต่างกล่องเครื่องมือนั้น เราไม่ต้องจัดการในข้อความ WM_PAINT แต่ในต้นแบบของคอนโทรลจะต้องกำหนดหมายเลข (IDs) (BISEARCH.H บรรทัดที่ 20 - 23 , 186) ของรูปภาพบิตแมพที่จะใช้ในหน้าต่างกล่องเครื่องมือ จากนั้นวิชวลเบสิกจะแสดงภาพบิตแมพนั้นเป็นสัญรูปแทนคอนโทรลในหน้าต่างกล่องเครื่องมือให้เอง

5.2 การติดต่อกันเกี่ยวกับคุณสมบัติ

จะต้องกำหนดคุณสมบัติต่าง ๆ ไว้ใน ตารางข้อมูลคุณสมบัติก่อนเสมอ (BISEARCH.H บรรทัดที่ 109 - 126) จากนั้นอาจมีชุดคำสั่งที่จัดการเกี่ยวกับคุณสมบัติอยู่ในกระบวนงานของคอนโทรล (BISEARCH.C บรรทัดที่ 39 - 65) ในการกำหนดคุณสมบัติใหม่ขึ้นมานั้น จะต้องกำหนดด้วยว่าจะให้วิชวลเบสิกติดต่อกับคุณสมบัตินั้นอย่างไรใน 3 รูปแบบต่อไปนี้

- จะอ่านค่าคุณสมบัติด้วยวิธีใด
- จะกำหนดค่าคุณสมบัติด้วยวิธีใด
- จะนำคุณสมบัติเข้าสู่หน่วยความจำ และบันทึกคุณสมบัติลงดิสค์หรือคลิปบอร์ดอย่างไร

รูปแบบของการติดต่อกันนั้นอาจจะเป็นการโอนถ่ายข้อมูลกันโดยตรง , การส่งข้อความหรือใช้ทั้ง 2 วิธี เช่น ในการกำหนดค่าคุณสมบัติ เราสามารถกำหนดได้ว่า จะให้วิชวลเบสิกส่งข้อความ VBM_SETPROPERTY ทุกครั้งที่มีการกำหนดค่าใหม่หรือไม่

ในการจัดการกับข้อความนั้น จะมีความยืดหยุ่นสูง แต่ต้องเขียนชุดคำสั่งมากขึ้น แต่ถ้าเรากำหนดให้วิชวลเบสิกโอนถ่ายข้อมูลกับตารางข้อมูลคุณสมบัติโดยตรง ก็จะต้องกำหนดตำแหน่ง (Offset) ของคุณสมบัตินั้นในโครงสร้างของผู้เขียนโปรแกรมด้วย (BISEARCH.H บรรทัดที่ 83 , 104)

มีคุณสมบัติจำนวนหนึ่งที่มีอยู่แล้วในคอนโทรลต่าง ๆ ของวิชวลเบสิก และเราอาจเขียนชุดคำสั่งเพิ่มเติมอีกเล็กน้อย หรือไม่ต้องเขียนเลยในการจัดการกับคุณสมบัติเหล่านั้น โดยกระบวนงานควบคุมโดยปริยาย (Default Control Procedure) ของวิชวลเบสิกจะจัดการให้เอง (BISEARCH.C บรรทัดที่ 49 , 63) คุณสมบัติเหล่านี้เรียกว่า คุณสมบัติมาตรฐาน (STANDARD PROPERTY) และเราสามารถเลือกคุณสมบัติเหล่านี้มาใช้ได้ ด้วยการกำหนดไว้ในตารางข้อมูลคุณสมบัติ (BISEARCH.H บรรทัดที่ 111 - 113)

5.3 การติดต่อกันเกี่ยวกับเหตุการณ์

เหตุการณ์ต่าง ๆ ของคอนโทรลจะต้องกำหนดไว้ในตารางข้อมูลเหตุการณ์ด้วยเสมอ (BISEARCH.H บรรทัดที่ 158 - 170) ซึ่งถ้าเหตุการณ์นั้นไม่ได้เป็นเหตุการณ์มาตรฐาน (STANDARD EVENT) เราก็ต้องเขียนชุดคำสั่ง เพื่อจัดการกับเหตุการณ์นั้นไว้ในกระบวนงานของคอนโทรลด้วย

ให้ใช้ฟังก์ชั่น VBFireEvent ในการกำหนดว่า ได้เกิดเหตุการณ์นั้นขึ้นแล้ว (BISEARCH.C บรรทัดที่ 214) ฟังก์ชั่นนี้จะทำให้มีการทำงานตามกระบวนงานเหตุการณ์ (EVENT PROCEDURE) ของวิชวลเบสิก ในกรณีที่ผู้พัฒนาโปรแกรมประยุกต์ด้วยวิชวลเบสิกได้กำหนดขึ้น

ซึ่งจากการจัดการกับข้อความจากวินโดวส์และวิชวลเบสิกเอง เราก็สามารถกำหนดได้ว่าจะให้เหตุการณ์ใดเกิดขึ้นเมื่อใด

มีเหตุการณ์มาตรฐานจำนวนหนึ่งอยู่แล้ว ที่เราไม่ต้องเขียนชุดคำสั่งใด ๆ เพื่อจัดการกับเหตุการณ์เหล่านั้น โดยกระบวนงานควบคุมโดยปริยายของวิชวลเบสิกจะจัดการให้เอง และเราสามารถเลือกเหตุการณ์เหล่านี้มาใช้กับคอนโทรลของเราได้ด้วยเช่นกัน โดยการกำหนดไว้ในตารางข้อมูลเหตุการณ์ (BISEARCH.H บรรทัดที่

160 - 161) อย่างไรก็ตาม เหตุการณ์มาตรฐานเหล่านี้จะเกิดขึ้นก็ต่อเมื่อส่วนประมวลผลข้อความโดยปริยายได้รับข้อความที่เหมาะสมเท่านั้น เช่น ถ้าเราจัดการกับข้อความของเมาส์เองแล้วคืนค่าทันที เหตุการณ์มาตรฐานคลิก (Click Event) ก็จะไม่เกิดขึ้น (เช่น ถ้า BISEARCH.C ไม่มีบรรทัดที่ 98 เหตุการณ์มาตรฐานดับเบิ้ลคลิก (DblClick Event) จะไม่เกิดขึ้น)

```
1   //-----------------------------------------------------------------
2   // BISEARCH.H   TEMPLATE
3   //-----------------------------------------------------------------
4
5   //-----------------------------------------------------------------
6   //  DEFINE ERROR NUMBERS.
7   //-----------------------------------------------------------------
8   #define ERR_None                    0
9   #define ERR_InvPropVal              380          // Error$(380) = "Invalid property value"
10  #define ERR_BadPicFmt               32000
11  #define ERR_WrongType               32001
12      ....
13      ....
14
15  //-----------------------------------------------------------------
16  // Resource Information
17  //-----------------------------------------------------------------
18  // Toolbox bitmap resource IDs numbers.
19  //-----------------------------------------------------------------
20  #define IDBMP_BISEARCH                  8000
21  #define IDBMP_BISEARCHDOWN              8001
22  #define IDBMP_BISEARCHMONO              8003
23  #define IDBMP_BISEARCHEGA               8006
24
25  #ifndef RC_INVOKED
26  //-----------------------------------------------------------------
27  // Macro for referencing member of structure
28  //-----------------------------------------------------------------
29  #define OFFSETIN(struc, field)    ((USHORT)&(((struc *)0)->field))
30
31  //-----------------------------------------------------------------
32  // PROTOTYPE OF ALLS FUNCTIONS IN BISEARCH.C
33  //-----------------------------------------------------------------
34  LONG  FAR  PASCAL  _export TreeCtlProc(HCTL, HWND, USHORT, USHORT, LONG);
35  VOID  NEAR PaintTree(HCTL, HWND, HDC);
36              ....
37              ....
38
39  //-----------------------------------------------------------------
40  // Property list
41  // Define the consecutive indicies for the properties
42  //-----------------------------------------------------------------
43  #define IPROP_BISEARCH_CTLNAME          0       // STANDARD PROPERTY
44  #define IPROP_BISEARCH_INDEX            1       // STANDARD PROPERTY
45  #define IPROP_BISEARCH_BACKCOLOR        2       // STANDARD PROPERTY
46              ....
47              ....
48
49  #define IPROP_BISEARCH_VALUETYPE        22      // CUSTOM PROPERTY
50              ....
51              ....
52  #define IPROP_BISEARCH_LISTOFVALUES     26      // CUSTOM PROPERTY
53              ....
54              ....
55
56  //-----------------------------------------------------------------
57  //  Programmer-defined structure
58  //-----------------------------------------------------------------
59  typedef struct tagBISEARCH
60    {
61    ENUM  ValueType;
62          ....
63          ....
64
65    HSZ   ListOfValues;
66          ....
67          ....
68  }  BISEARCH;
69
70  typedef BISEARCH FAR * LPBISEARCH;
71
72  #define LpbisearchDEREF(hctl)    ((LPBISEARCH)VBDerefControl(hctl))
```

```
73
74  //---------------------------------------------------------------------
75  // Property info
76  //---------------------------------------------------------------------
77  #define     NUM_TYPE        0
78  #define     STR_TYPE        1
79  PROPINFO Property_ValueType =
80     {
81     "ValueType",
82     DT_ENUM | PF_fGetData | PF_fSetMsg | PF_fSaveData | PF_fDefVal,
83     OFFSETIN(BISEARCH, ValueType),
84        0,
85     NUM_TYPE,
86     "0 - Number\0" "1 - String\0",
87     STR_TYPE
88     };
89
90  PROPINFO ........
91     {
92      .....,
93      ........,
94      ........
95      ....., ....,  ....
96  };
97         ....
98         ....
99
100 PROPINFO Property_ListOfValues =
101    {
102    "ListOfKeys",
103    DT_HSZ | PF_fGetData | PF_fSetData | PF_fSetMsg | PF_fSaveData | PF_fDefVal,
104    OFFSETIN(BISEARCH, ListOfValues),
105    0, 0, NULL, 0
106 };
107
108
109 PPROPINFO BiSearch_Properties[] =
110    {
111    PPROPINFO_STD_CTLNAME,
112    PPROPINFO_STD_INDEX,
113    PPROPINFO_STD_BACKCOLOR,
114        ....
115        ....
116
117    &Property_ValueType,
118        ....
119        ....
120
121    &Property_ListOfValues,
122        ....
123        ....
124
125    NULL
126    };
127
128 //---------------------------------------------------------------------
129 // Event list
130 //---------------------------------------------------------------------
131 // Define the consecutive indicies for the events
132 //---------------------------------------------------------------------
133 #define IEVENT_BISEARCH_CLICK                    0      // STANDARD EVENT
134 #define IEVENT_BISEARCH_DBLCLICK                 1      // STANDARD EVENT
135                 ....
136                 ....
137
138 #define IEVENT_BISEARCH_NODEINCREASE             7      // CUSTOM EVENT
139                 ....
140                 ....
141
142 WORD ParamTypes_NodeIncrease[] = {ET_I2, ET_HLSTR};
143 EVENTINFO Event_NodeIncrease =
144    {
```

```
145         "NodeIncrease",
146         2, 4, ParamTypes_NodeIncrease, "NodeID As Integer, Key As String"
147       };
148
149     WORD .....
150     EVENTINFO  .....
151       {
152         .....,
153         ...., ...., ...., ......
154       };
155           ....
156           ....
157
158     PEVENTINFO BiSearch_Events[] =
159       {
160        PEVENTINFO_STD_CLICK,
161        PEVENTINFO_STD_DBLCLICK,
162           ....
163           ....
164
165        &Event_NodeIncrease,
166           ....
167           ....
168
169       NULL
170       };
171
172
173     //---------------------------------------------------------------
174     // MODEL STRUCTURE
175     //---------------------------------------------------------------
176     // Define the control model (using the event and property structures).
177     //---------------------------------------------------------------
178     MODEL modelBiSearch =
179       {
180       VB_VERSION,                          // VB version being used
181       0,                                   // MODEL flags
182       (PCTLPROC)TreeCtlProc,               // Control procedure
183       CS_VREDRAW | CS_HREDRAW,             // Class style
184       WS_BORDER,                           // Default Windows style
185       sizeof(BISEARCH),                    // Size of BISEARCH structure
186       IDBMP_BISEARCH,                      // Palette bitmap ID
187       "BiSearch",                          // Default control name
188       "BISEARCH",                          // Visual Basic class name
189       NULL,                                // Parent class name
190       BiSearch_Properties,                 // Property information table
191       BiSearch_Events,                     // Event information table
192       IPROP_BISEARCH_VALUEINNODE,          // Default property
193       IEVENT_BISEARCH_CLICK,               // Default event
194       -1                                   // Property representing value of ctl
195       };
196
197     #endif    // RC_INVOKED
198
199     //------------------------ EOF.  BISEARCH.H ------------------------
```

```
1   //-----------------------------------------------------------------
2   //  BISEARCH.C   TEMPLATE
3   //-----------------------------------------------------------------
4   // Contains control procedure for BISEARCH control
5   //-----------------------------------------------------------------
6
7   #include <windows.h>
8   #include <vbapi.h>
9   #include "bisearch.h"
10
11  //-----------------------------------------------------------------
12  // Global Variables
13  //-----------------------------------------------------------------
14  HANDLE hmodDLL;
15
16  //-----------------------------------------------------------------
17  // BiSearch control Procedure
18  //-----------------------------------------------------------------
19  LONG FAR PASCAL _export TreeCtlProc
20  (
21     HCTL   hctl,
22     HWND   hwnd,
23     USHORT msg,
24     USHORT wp,
25     LONG   lp
26  )
27  {
28     LPBISEARCH lpbisearch = LpbisearchDEREF(hctl);
29
30     switch (msg)
31     {
32        case WM_NCCREATE:
33           lpbisearch->LevelDist = 20,
34           lpbisearch->ScrollBar = 3;
35              ....
36              ....
37
38           break;
39        case VBM_SAVEPROPERTY:
40           switch (wp) {
41              case IPROP_BISEARCH_DELETEDVALUE:
42                    ....
43                    ....
44
45              case IPROP_BISEARCH_VALUEINNODE:
46                    ....
47                    ....
48              default:
49                 return VBDefControlProc(hctl, hwnd, msg, wp, lp);
50           }
51           break;
52
53        case VBM_SETPROPERTY:
54           switch (wp) {
55              case IPROP_BISEARCH_DELETEDVALUE:
56                    ....
57                    ....
58              case IPROP_BISEARCH_VALUEINNODE:
59                    ....
60                 FireNodeIncrease(hctl, 1);
61                    ....
62              default:
63                 return VBDefControlProc(hctl, hwnd, msg, wp, lp);
64           }
65           break;       // VBM_SETPROPERTY
66
67        case VBM_METHOD:
68           switch (wp) {
69              case METH_ADDITEM:
70                    ....
71                    ....
72
73              case METH_REMOVEITEM:
74                    ....
```

```
75
76                default:
77                    return VBDefControlProc(hctl, hwnd, msg, wp, lp);
78             }     // switch (wp)
79             break;
80          case WM_PAINT:
81             if (wp)
82                PaintTree(hctl, hwnd, (HDC) wp);
83             else
84                {
85                PAINTSTRUCT ps;
86
87                BeginPaint(hwnd, &ps);
88                PaintTree(hctl, hwnd, ps.hdc);
89                EndPaint(hwnd, &ps);
90             }
91             break;
92
93          case WM_LBUTTONDBLCLK:
94          case WM_RBUTTONDOWN:
95                ....
96                ....
97
98             return VBDefControlProc(hctl, hwnd, msg, wp, lp);
99          case WM_NCDESTROY:
100               ....
101               ....
102
103            break;
104         default:
105            return VBDefControlProc(hctl, hwnd, msg, wp, lp);
106      }        // switch (msg)
107   }    // TreeCtlProc()
108
109   //---------------------------------------------------------------------
110   // Initialize library. This routine is called when the first client loads
111   // the DLL.
112   //---------------------------------------------------------------------
113   int FAR PASCAL LibMain( HANDLE hModule,   WORD  wDataSeg,
114                           WORD  cbHeapSize,   LPSTR  lpszCmdLine)
115   {
116      // Avoid warnings on unused (but required) formal parameters
117      wDataSeg    = wDataSeg;
118      cbHeapSize  = cbHeapSize;
119      lpszCmdLine = lpszCmdLine;
120
121      hmodDLL = hModule;
122
123      return 1;
124   }
125
126   //---------------------------------------------------------------------
127   // Register custom control. This routine is called by VB when the custom
128   // control DLL is loaded for use.
129   //---------------------------------------------------------------------
130   BOOL FAR PASCAL _export VBINITCC(USHORT usVersion, BOOL  fRuntime)
131   {
132      // Avoid warnings on unused (but required) formal parameters
133      fRuntime  = fRuntime;
134      usVersion = usVersion;
135
136      // Register control(s)
137      return VBRegisterModel(hmodDLL, &modelTree1);
138
139   }
140
141   //---------------------------------------------------------------------
142   // Unregister custom control.  This routine is called by VB when the custom
143   // control DLL is being unloaded.
144   //---------------------------------------------------------------------
145   VOID FAR PASCAL _export VBTERMCC(VOID)
146   {
147      ....
148      ....
```

```
149     return;
150  }
151
152  //---------------------------------------------------------------------
153  // WEP
154  //---------------------------------------------------------------------
155  // C7 and QCWIN provide default a WEP:
156  //---------------------------------------------------------------------
157  #if (_MSC_VER < 610)
158
159  int FAR PASCAL WEP(int fSystemExit);
160
161  //---------------------------------------------------------------------
162  // For Windows 3.0 it is recommended that the WEP function reside in a
163  // FIXED code segment and be exported as RESIDENTNAME.  This is
164  // accomplished using the alloc_text pragma below and the related EXPORTS
165  // and SEGMENTS directives in the .DEF file.
166  //
167  // Read the comments section documenting the WEP function in the Windows
168  // 3.1 SDK "Programmers Reference, Volume 2: Functions" before placing
169  // any additional code in the WEP routine for a Windows 3.0 DLL.
170  //---------------------------------------------------------------------
171  #pragma alloc_text(WEP_TEXT,WEP)
172
173  //---------------------------------------------------------------------
174  // Performs cleanup tasks when the DLL is unloaded.  WEP() is
175  // called automatically by Windows when the DLL is unloaded (no
176  // remaining tasks still have the DLL loaded).          It is strongly
177  // recommended that a DLL have a WEP() function, even if it does
178  // nothing but returns success (1), as in this example.
179  //---------------------------------------------------------------------
180  int FAR PASCAL WEP()
181  (
182     int fSystemExit
183  )
184  {
185     // Avoid warnings on unused (but required) formal parameters
186     fSystemExit = fSystemExit;
187
188     return 1;
189  }
190  #endif // C6
191
192  VOID NEAR PaintTree(HCTL hctl, HWND hwnd, HDC hdc)
193  {
194           ....
195           ....
196
197  }    // PaintTree( )
198
199           ....
200           ....
201
202  typedef struct tagNODEINCREASEPARAMS
203     {
204        HLSTR  IncreaseStr;
205        int far *NumNode;
206        LPVOID Index;
207     } NODEINCREASEPARAMS;
208
209  VOID NEAR FireNodeIncrease(HCTL hctl, int Idx)
210  {
211     NODEINCREASEPARAMS params;
212           ....
213           ....
214     VBFireEvent(hctl, IEVENT_TREE1_NODEINCREASE, &params);
215           ....
216           ....
217
218  }    // FireNodeIncrease()
219           ....
220           ....
221  //-------------------------- EOF.   BISEARCH.C --------------------------
```