บทที่ 3

ไมโครซอฟต์วิชวลเบสิค สำหรับวินโดว์

เนื่องจากตัวควบคุมที่จะพัฒนาขึ้นมานี้ มีจุดมุ่งหมายให้ทำงานภายใต้วิชวลเบสิค โดยเฉพาะ จึงสมควรที่จะมีความรู้เบื้องต้นเกี่ยวกับข้อกำหนดและลักษณะการทำงานของวิชวลเบสิค เพื่อประโยชน์ในการใช้งานและทดสอบตัวควบคุมที่จะสร้างขึ้นมา

ในปัจจุบันนี้ การเขียนโปรแกรมให้ทำงานบนวินโดว์ไม่ใช่เรื่องยุ่งยากอีกต่อไป โดยวิชวลเบสิคนำวิธีการที่เรียกกันว่า เชิงทัศน์(Visual) มารวมกับภาษาเบสิค ซึ่งเป็นภาษาที่ง่ายต่อการเข้าใจ จึงทำให้การเขียนโปรแกรมสำหรับทำงานบนวินโดว์เป็นไปอย่างสะดวกยิ่งขึ้น

หลักการใช้วิชวลเบสิค

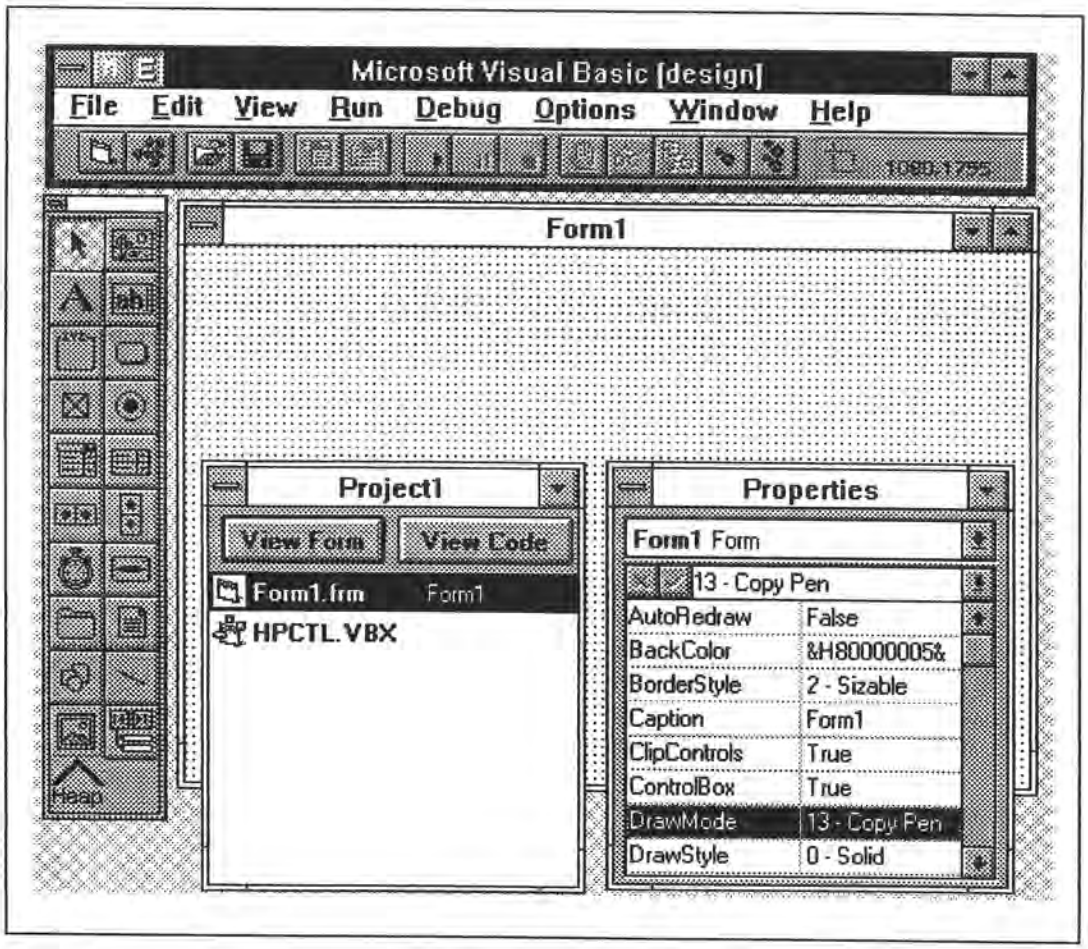
การเขียนโปรแกรมวิชวลเบสิคนั้น ก่อนอื่นจะต้องออกแบบรูปแบบและส่วนการติดต่อกับผู้ใช้ (User Interface) ของโปรแกรมที่จะเขียนก่อนว่า มีลักษณะอย่างไร โดยการใช้วัตถุ (Object) หรือตัวควบคุม (Control) ต่าง ๆ ที่มีอยู่ นำมาวางลงบนฟอร์ม (Form) แล้วก็ปรับขนาดของวัตถุนั้นให้เหมาะสมตามที่ต้องการก่อน (หรืออาจจะปรับขนาดภายหลังก็ได้) และก็ไปกำหนดคุณสมบัติของวัตถุนั้น ๆ ซึ่งมีอยู่หลายข้อ หลังจากกำหนดคุณสมบัติเสร็จแล้ว ก็ทำการเขียนส่วนโปรแกรมที่ควบคุมวัตถุนั้น ๆ ว่า จะให้ทำงานอย่างไร ซึ่งสามารถสรุปเป็นหลักใหญ่ ๆ 3 ขั้นตอนดังนี้ (กิตติ เลิศพิริยสุวัฒน์, 2536)

1. เลือกวัตถุซึ่งจะเป็นส่วนหน้าตาของโปรแกรมที่จะติดต่อกับผู้ใช้
2. กำหนดคุณสมบัติของแต่ละวัตถุ
3. เขียนส่วนโปรแกรมควบคุมการทำงานของวัตถุนั้น ๆ

หลังจากที่เราได้ทราบอย่างคร่าว ๆ ถึงรูปแบบการเขียนโปรแกรมโดยวิชวลเบสิคแล้ว ต่อไปจะเป็นการใช้งานจริงโดยเรียกโปรแกรมวิชวลเบสิคขึ้นมาใช้งาน หน้าจอการทำงานของ

วิชวลเบสิคจะประกอบไปด้วยส่วนหลักๆ 5 ส่วน ดังตัวอย่างรูปที่ 3.1 (กิตติ เลิศพิริยสุวัฒน์, 2536) คือ

1. หน้าต่างคุณสมบัติของวัตถุ (Properties Window)
2. กล่องเครื่องมือของวิชวลเบสิค (Visual Basic Toolbox)
3. หน้าต่างโปรเจค (Project Window)
4. ฟอร์มสำหรับการทำงาน (Form1)
5. แท่งรายการคำสั่ง (Visual Baisc Toolbar)

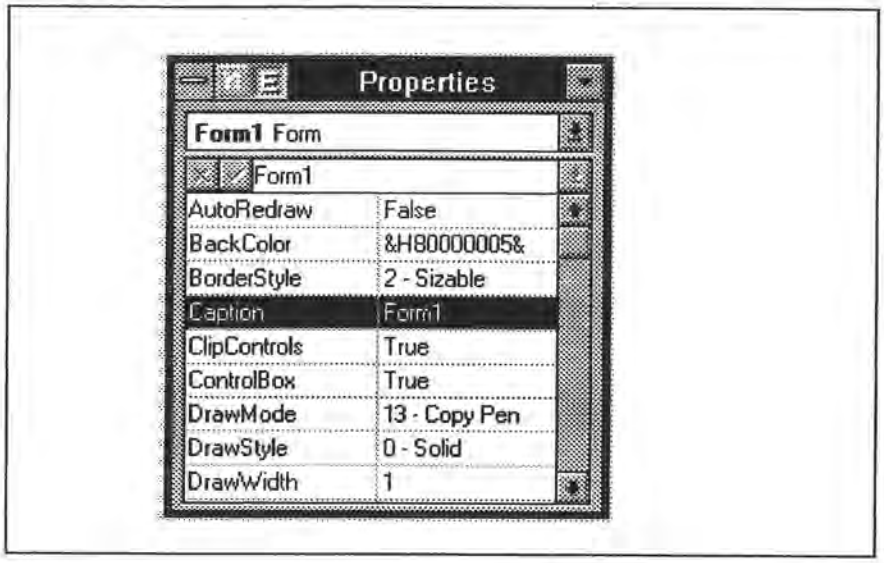


รูปที่ 3.1 แสดงส่วนประกอบ 5 ส่วนหลักของวิชวลเบสิค

1. หน้าต่างคุณสมบัติของวัตถุ (Properties Window)

เป็นส่วนที่ใช้ควบคุมหรือจัดการเกี่ยวกับการกำหนดคุณสมบัติของวัตถุต่าง ๆ รวมทั้งคุณสมบัติของฟอร์มด้วย เมื่อต้องการที่จะกำหนดหรือเปลี่ยนแปลงคุณสมบัติของวัตถุไหนหรือ

ฟอร์มไหน ก็ให้ใช้เมาส์คลิก (click) ที่วัตถุนั้นหรือฟอร์มนั้น หลังจากที่คลิกแล้วที่หน้าต่างคุณสมบัติจะเปลี่ยนค่าไปเป็นค่าของวัตถุที่เราคลิก ซึ่งในตอนแรกจะเป็นค่าที่ถูกกำหนดไว้อย่างทั่วไป ตามตัวอย่างในรูปที่ 3.2



รูปที่ 3.2 แสดงหน้าต่างคุณสมบัติของวัตถุ (Properties Window)

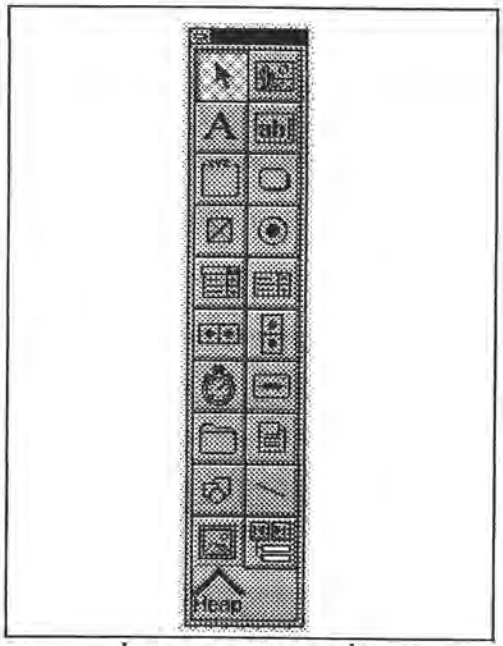
2. กล่องเครื่องมือของวิชวลเบสิค (Visual Basic Toolbox)

เป็นส่วนที่ใช้เลือกวัตถุหรือตัวควบคุมต่าง ๆ มาวางลงบนฟอร์ม ตัวอย่างดังรูปที่ 3.3 วัตถุที่เป็นมาตรฐานที่ให้มากับวิชวลเบสิค เช่น ข้อความ (Label) ช่องรับข้อความ (Text box) ปุ่มควบคุม (Command button) และช่องรูปภาพ (Picture box) เป็นต้น นอกจากนี้ยังสามารถเพิ่มวัตถุหรือตัวควบคุมที่สร้างขึ้นเองเข้ามายังกล่องเครื่องมือ (Toolbox) นี้ได้โดยใช้คำสั่ง Add File ในรายการเลือก File

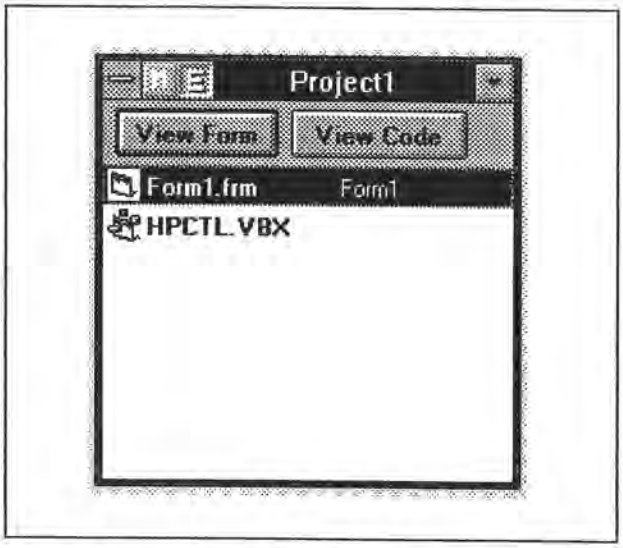
3. หน้าต่างโปรเจค (Project Window)

เป็นส่วนที่บอกว่ามีแฟ้มใดหรือฟอร์มใดประกอบอยู่ในโปรแกรมหรือโปรเจคที่เราสร้างขึ้นบ้าง ซึ่งในหนึ่งโปรแกรมหรือหนึ่งโปรเจคจะประกอบด้วยแฟ้มหลายชนิด เช่น แฟ้มประเภท .FRM ซึ่งมีทั้งฟอร์ม และชุดคำสั่งที่เกี่ยวข้องกับตัวควบคุมต่าง ๆ ตัวโปรแกรมอาจเก็บแยกเป็นหลายโมดูล แต่ละโมดูลอาจเก็บแยกเป็นแฟ้มต่างหากได้ (อาจเป็นแฟ้มประเภท .BAS)

และแฟ้มประเภท .VBX เป็นแฟ้มของวัตถุหรือตัวควบคุมที่เพิ่มเติมเข้ามานอกเหนือจากชุดมาตรฐาน (ตัวอย่างดังรูปที่ 3.4)



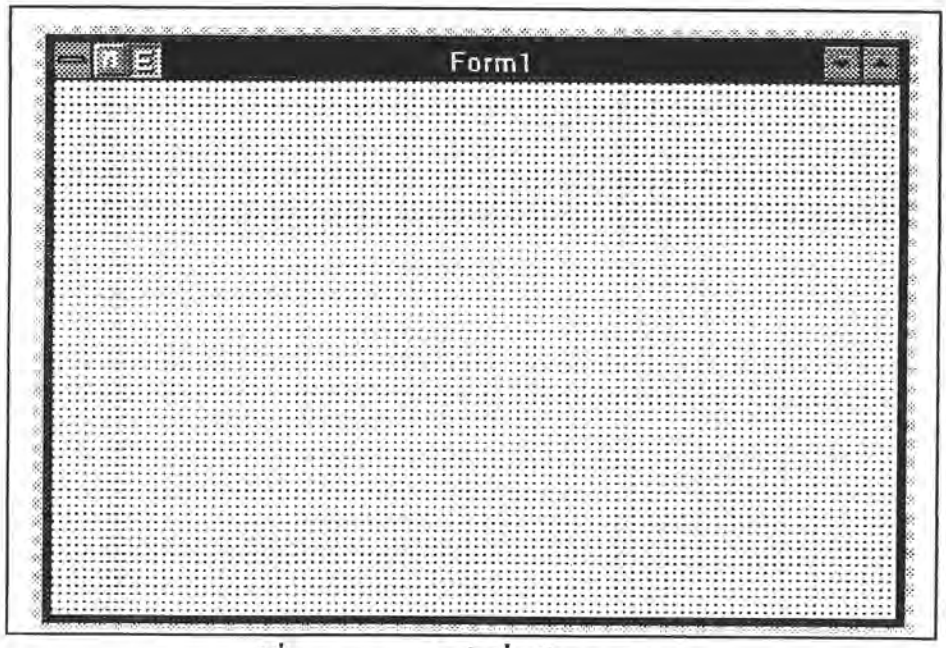
รูปที่ 3.3 แสดงกล่องเครื่องมือ (Toolbox)



รูปที่ 3.4 แสดงหน้าต่างโปรเจค (Project Window)

4. ฟอร์มสำหรับการทำงาน (Form1)

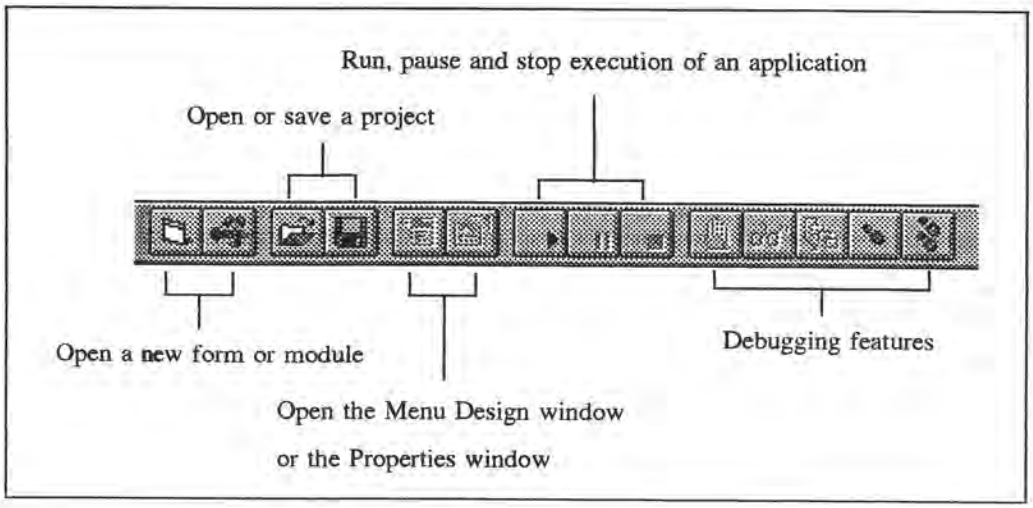
เป็นฟอร์มทั่วไปหรือฟอร์มเริ่มต้นในการทำงาน และในโปรแกรมของเราสามารถที่จะประกอบไปด้วยหลาย ๆ ฟอร์ม ก็ได้ (ตัวอย่างดังรูป 3.5)



รูปที่ 3.5 แสดงฟอร์มเริ่มต้นการทำงาน

5. แท่งรายการคำสั่ง (Visual Baisc Toolbar)

เป็นส่วนที่อำนวยความสะดวกให้กับผู้ใช้เป็นอย่างมาก เพราะตัว Toolbar นี้ ประกอบไปด้วยคำสั่งหลัก ๆ ที่ใช้อยู่เป็นประจำ ในรูปที่ 3.6 เป็นตัวอย่างของ Toolbar ซึ่ง ประกอบไปด้วย ตัว Debugging ส่วนของการเปิดแฟ้ม บันทึกแฟ้ม และสัญรูป(Icon) ในการ สร้างรายการเลือก (Pull down Menu)



รูปที่ 3.6 แสดงแท่งรายการคำสั่ง (Toolbar)

โครงสร้างของภาษา

สำหรับภาษาที่ใช้อ้างอิงนั้นวิชวลเบสิคจะใช้ภาษาเบสิคที่มีลักษณะเดียวกันกับ Quick Basic(TM) เป็นตัวอ้างอิง ซึ่งแตกต่างจากภาษาเบสิคโดยทั่วไป ตรงที่เป็นภาษาที่มีโครงสร้าง เช่นเดียวกับภาษาสูงพวกภาษาซี หรือ ภาษาปาสคาล รายละเอียดของโครงสร้างของภาษาจะกล่าวถึงโดยสังเขปเฉพาะส่วนที่เกี่ยวข้องกับการใช้งานตัวควบคุมที่พัฒนาขึ้นมา

1. Data Types

คือชนิดของข้อมูลที่มีให้ใช้ในภาษานี้ได้แก่

ชนิด	คำอธิบาย	เครื่องหมาย	ช่วงที่ใช้งานได้
Integer	จำนวนเต็ม 2 ไบต์	%	-32,768 ถึง +32,768
Long	จำนวนเต็ม 4 ไบต์	&	-2,147,483,648 ถึง +2,147,483,648
Single	จำนวนจริง 4 ไบต์	!	-3.37E+38 ถึง +3.37E+38
Double	จำนวนจริง 8 ไบต์	#	-1.67D+308 ถึง +1.67D+308
String	ตัวอักษร 1 ไบต์	$	0 ถึง 65,535
Currency	ตัวเลข 8 ไบต์	@	-9.22E+14 ถึง +9.22E+14

ตารางที่ 3.1 แสดงชนิดของข้อมูลที่มีใช้ในวิชวลเบสิค

2. การกำหนดค่าคงที่

ค่าคงที่ (Constants) เป็นการกำหนดว่าตัวคงที่ในโปรแกรมที่เราใช้มีค่าเป็นเท่าไร โดยที่จะมีค่าคงที่ตลอดการทำงานของโปรแกรม ซึ่งการใช้ค่าคงที่จะทำให้โปรแกรมสามารถอ่านได้ง่ายขึ้น และเมื่อต้องการแก้ไขค่าของค่าคงที่ ก็สามารถทำได้ง่าย เพราะแก้ไขที่ค่าคงที่เพียงที่เดียว มีรูปแบบดังนี้

[Global] Const *constantname* = *expression* (คำที่อยู่ใน [...] จะใส่หรือไม่ก็ได้)

เช่น Global Const TRUE = -1

Global Const FALSE = 0

เพื่อความสะดวกในโปรแกรมวิชวลเบสิค ได้มีค่าคงที่ที่ใช้โดยทั่วไปอยู่แล้ว ซึ่งสามารถดูได้ในแฟ้ม CONSTANT.TXT

3. การกำหนดตัวแปร

ในการกำหนดตัวแปรขึ้นใช้งานในวิชวลเบสิค จะใช้คำว่า Dim ในการกำหนดชนิดของตัวแปร โดยมีรูปแบบดังนี้

Dim *varname* As *data type* หรือ

Dim *varname(type-declaration character)* เช่น

Dim Index As Integer

Dim Salary As Double

Dim Index%

Dim Salary#

(หมายเหตุ วิชวลเบสิคไม่สนใจตัวอักษรตัวพิมพ์ใหญ่หรือตัวพิมพ์เล็ก)

4. การกำหนดค่า

ในโปรแกรมส่วนใหญ่ จะมีการกำหนดค่าบ่อยมาก ในวิชวลเบสิคมีรูปแบบดังนี้

[Control].destination = source เช่น

Index = 3

Command1.BackColor = Blue

5. การกำหนดแถวลำดับ (Array)

แถวลำดับคือโครงสร้างของข้อมูลที่อนุญาตให้เก็บข้อมูลชนิดเดียวกันไว้ในตัวแปรเดียวกัน แต่มีตัวเลขกำกับว่า เป็นข้อมูลตัวที่เท่าใด มีรูปแบบดังนี้

Dim *arrayname* ([*lower* To] *upper*[, [*lower* To] *upper*]) As *data type*

เช่น

Dim Index(99) As Integer หมายถึงมี Index(0) - Index(99)

Dim Matrix(1 to 10, 15) หมายถึงมี Matrix(1,0) - Matrix(10, 15)

6. การกำหนดเรคคอร์ด

เรคคอร์ดหรือ User-Defined Types เป็นชนิดของข้อมูลที่ผู้ใช้กำหนดขึ้นมาใหม่จากการเชื่อมโยงกันของชนิดข้อมูลหลักที่มีอยู่ โดยใช้ประโยค Type ซึ่งประโยค Type นี้ มิได้เป็นการกำหนดตำแหน่งการเก็บข้อมูล แต่เป็นการกำหนดรูปแบบของข้อมูล ซึ่งมีวิธีใช้ดังนี้

```
Type usertype
    elementname As data type
    elementname As data type
        :
        :
End Type
```

เช่น

```
Type Employee
Name As String * 50
Address As String * 40
HourlyWage As Single
WeeklyPay As Double
End Type
```

นอกจากนี้ยังมีกลุ่มเครื่องหมายทางคณิตศาสตร์ เครื่องหมายความสัมพันธ์ และเครื่องหมายทางตรรกศาสตร์ รวมทั้งประโยคคำสั่งหรือวิธี (Method) ของตัวควบคุม ประโยคการกำหนดเงื่อนไข และประโยคการกำหนดจำนวนรอบการทำงาน ซึ่งมีการนำเข้ามาใช้ประกอบการเขียนโปรแกรมด้วย

การใช้พังก์ชันวินโดว์ API ในวิชวลเบสิค

การเขียนโปรแกรมประยุกต์บนวินโดว์ด้วยวิชวลเบสิค ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมเพื่ออ่านข้อมูลบางส่วนโดยตรงจากวินโดว์ โดยการใช้วินโดว์ API (Windows Application Programming Interface) ซึ่งเป็นชุดของฟังก์ชันมากกว่า 700 ฟังก์ชัน ฟังก์ชันเหล่านี้ส่วนใหญ่จะเก็บไว้ในแฟ้มข้อมูล KERNEL.EXE, USER.EXE และ GDI.EXE เป็นหลัก และ

สามารถเรียกใช้ได้จากทุกโปรแกรมประยุกต์ ไม่เฉพาะในวิชวลเบสิคเท่านั้น ตัวอย่างเช่น หากต้องการทราบข้อมูลบางส่วนจากแฟ้มข้อมูล .INI ซึ่งโปรแกรมวินโดว์ส่วนใหญ่รวมทั้งตัววินโดว์เองจะมีแฟ้มข้อมูล .INI เพื่อเก็บรายละเอียดของโปรแกรม เช่น แฟ้มข้อมูล WIN.INI และ SYSTEM.INI จะเก็บข้อมูลสำหรับการเริ่มต้นทำงานของวินโดว์ และข้อมูลเกี่ยวกับระบบ เนื่องจากความต้องการอ่านข้อมูลจากแฟ้มข้อมูล .INI มีอยู่อย่างสม่ำเสมอ วินโดว์จึงได้จัดให้มีฟังก์ชัน API เพื่อโปรแกรมประยุกต์สามารถเรียกใช้ในการอ่าน และแก้ไขแฟ้มข้อมูลเหล่านี้ได้โดยสะดวก นอกจากนี้ผู้เขียนโปรแกรมยังสามารถเรียกใช้ฟังก์ชันจากแฟ้มคลังคำสั่งเชื่อมโยงแบบพลวัต (DLL) ที่สร้างขึ้นเองก็ได้

ขั้นตอนการเรียกใช้ฟังก์ชันจากคลังคำสั่งการเชื่อมโยงแบบพลวัต หรือฟังก์ชันจากวินโดว์ API สามารถสรุปการเรียกใช้ได้ด้วยขั้นตอนทั่วไปต่อไปนี้

1. ให้คำนิยามฟังก์ชันด้วยคำสั่ง DECLARE ให้วิชวลเบสิครู้จักฟังก์ชันเหล่านี้และกำหนดด้วยว่าฟังก์ชันที่ใช้เป็นแบบ Function หรือแบบ Sub ซึ่งมีข้อแตกต่างกันที่ แบบ Function สามารถให้ค่าของฟังก์ชันหลังจากการทำงานกลับออกมาได้ ส่วนแบบ Sub นั้นคืนค่าของฟังก์ชันกลับออกมาไม่ได้ เพียงทำงานตามกระบวนการของฟังก์ชันเท่านั้น แต่ทั้ง 2 แบบก็ยังสามารถส่งค่าไปกลับผ่านพารามิเตอร์ของฟังก์ชันได้

2. กำหนดเนื้อที่สำหรับตัวแปรด้วยคำสั่ง Global หรือ Dim เป็นต้น (ฟังก์ชันต้องการเนื้อที่สำหรับคำตอบ)

3. เรียกใช้ฟังก์ชัน

```
1. Declare Function GetWindowsDirectory Lib "KERNEL.EXE" (ByVal lpBuffer As String,
   ByVal nSize As Integer) As Integer
2. Dim WinPath As String * 255
3. Worked = GetWindowsDirectory(WinPath, Len(WinPath))
```

รูปที่ 3.7 แสดงตัวอย่างการเรียกใช้ฟังก์ชันจากวินโดว์ API

รูปที่ 3.7 แสดงตัวอย่างการใช้ฟังก์ชันจากวินโดว์ API โดยนิยามเป็นแบบ Function ตามลำดับขั้นตอน 1, 2 และ 3 โดยฟังก์ชันนี้ใช้ในการค้นหาที่ตั้งของไดเร็กทอรีของวินโดว์ ซึ่งส่วนใหญ่จะอยู่บนไดรว์ C เช่น C:\WINDOWS แต่อาจจะมีบ้างที่แตกต่างไปจากนี้ ฟังก์ชันดังกล่าวอยู่ในแฟ้มข้อมูลชื่อ KERNEL.EXE โดยตัวฟังก์ชันมีชื่อว่า GetWindowsDirectory เมื่อเรียกใช้

ฟังก์ชันจะให้ค่าเส้นทาง (Path) ของไดเร็กทอรีของวินโดว์ในรูปของสายอักขระ (String) ในที่นี้คือตัวแปรชื่อ WinPath

ลำดับขั้นในการเรียกใช้คำสั่งคือ ขั้นแรกใช้คำสั่ง DECLARE เพื่อบอกให้วิชวลเบสิครู้ว่า ฟังก์ชันนั้นอยู่ในแฟ้มชื่ออะไร ตลอดจนค่าต่าง ๆ ที่ต้องส่งไป จากนั้นให้กำหนดเนื้อที่ตัวแปรสำหรับการส่งค่ากลับ ในที่นี้คือ ตัวแปร WinPath ซึ่งเป็นสายอักขระความยาว 255 ตัวอักษร ในขั้นตอนสุดท้ายคือการเรียกใช้งานฟังก์ชันพร้อมกับรับค่าของฟังก์ชันเข้าไว้ในตัวแปร Worked ถ้าฟังก์ชันทำงานสำเร็จ ก็จะให้ค่าตัวแปร Worked เป็นศูนย์ แต่ถ้าไม่สำเร็จก็จะให้ค่าเป็นค่าเลขที่ของข้อผิดพลาด (error code) เพื่อให้ผู้เขียนโปรแกรมตรวจสอบต่อไป

```
1. Declare Sub HpSort Lib "HPCTL.VBX" (ByVal hWnd As Integer)
2. HpSort Heap1.hWnd
```

รูปที่ 3.8 แสดงตัวอย่างการเรียกใช้ฟังก์ชันจากคลังคำสั่งการเชื่อมโยงแบบพลวัต

ในรูปที่ 3.8 เป็นการแสดงตัวอย่างการเรียกใช้ฟังก์ชันจากคลังคำสั่งการเชื่อมโยงแบบพลวัตที่สร้างขึ้นเอง โดยนิยามการใช้ฟังก์ชันเป็นแบบ Sub เป็นการเรียกใช้ฟังก์ชันที่ชื่อว่า HpSort ที่อยู่ในแฟ้ม DLL ชื่อ HPCTL.VBX ให้ทำการจัดเรียงลำดับข้อมูลที่อยู่ในตัวควบคุมที่ชื่อ Heap1 โดยส่งค่าแฮนเดิลของวินโดว์ (Window Handle) สำหรับ Heap1 เข้าไปให้ฟังก์ชัน

<u>การพัฒนาตัวควบคุมประดิษฐ์ (Custom Control)</u>

ตัวควบคุมประดิษฐ์เป็นตัวควบคุมที่เพิ่มเข้ามานอกเหนือไปจากตัวควบคุมมาตรฐานในกล่องเครื่องมือ (Toolbox) ของวิชวลเบสิค ซึ่งจะช่วยให้การพัฒนาระบบงานสำหรับผู้ใช้บนวิชวลเบสิคมีความสะดวกและกว้างขวางมากยิ่งขึ้น ตัวควบคุมสำหรับโครงสร้างข้อมูลแบบฮีปชนิดทวิภาคที่จะพัฒนาขึ้นมานี้ก็นับว่าเป็นตัวควบคุมประดิษฐ์ตัวหนึ่ง โดยเหตุการณ์ คุณสมบัติและวิธีของตัวควบคุมประดิษฐ์ สามารถใช้ได้ในสิ่งแวดล้อมของวิชวลเบสิค ในขณะเดียวกันตัวควบคุมประดิษฐ์ก็สามารถสร้างให้มองเห็นและรับรู้แตกต่างไปจากตัวควบคุมมาตรฐานทั่ว ๆ ไปได้ โดยสามารถกำหนดเหตุการณ์ คุณสมบัติและการทำงานที่ใหม่ ๆ ได้ สำหรับการสร้างตัวควบคุมประดิษฐ์นั้น สิ่งที่ควรจะพิจารณามีดังต่อไปนี้

- พิจารณาคุณสมบัติของตัวควบคุมว่า จะมีคุณสมบัติอะไรบ้าง โดยคุณสมบัติของตัวควบคุมจะมีได้ทั้งคุณสมบัติมาตรฐานและคุณสมบัติที่สร้างขึ้นมาใหม่ และจะดำเนินการอะไรต่อไป เมื่อผู้ใช้ทำการกำหนดคุณสมบัติของตัวควบคุมจากตารางคุณสมบัติที่เราสร้างไว้

- เหตุการณ์ของตัวควบคุม มีได้ทั้งเหตุการณ์มาตรฐานและเหตุการณ์ที่สร้างขึ้นมาใหม่ โดยเหตุการณ์ที่เรากำหนดสามารถนำเข้าไปใช้ในหน้าต่างของการเขียนโปรแกรม (Code window) ได้ และสามารถทำงานร่วมกับข้อความ (Messages) ของวินโดว์และวิชวลเบสิคเพื่อพิจารณาว่าภายใต้เงื่อนไขใดจึงจะทำให้เหตุการณ์ที่เรากำหนดเกิดขึ้นได้ในวิชวลเบสิค

- พฤติกรรมของวิธี ตัวควบคุมประดิษฐ์สนับสนุนวิธีมาตรฐานของ วิชวลเบสิคในจำนวนที่จำกัด (Microsoft Corp., 1993) กล่าวคือ เราสามารถเขียนโปรแกรมเพื่อกำหนดการทำงานของวิธีมาตรฐานให้ทำงานตามที่เราต้องการได้ แต่เราไม่สามารถเปลี่ยนแปลงการส่งอาร์กิวเมนต์ (Argument) ของวิธีมาตรฐานได้ รวมทั้งไม่สามารถสร้างวิธีขึ้นมาใหม่ได้

- ตัวควบคุม ที่สร้างจะถูกแสดงและพิมพ์ออกมาอย่างไร

เมื่อสร้างตัวควบคุมประดิษฐ์ เราจะได้แฟ้มที่มีส่วนขยายเป็น .VBX แฟ้มนี้ ผู้ใช้วิชวลเบสิค สามารถนำไปใช้ในโปรเจค (Project) ในการพัฒนาโปรแกรมได้ โดยหลักการพื้นฐานในการแสดงผลและการติดต่อใช้งานของตัวควบคุมประดิษฐ์จะเหมือนกับการใช้ตัวควบคุมมาตรฐานทั่ว ๆ ไป แฟ้มตัวควบคุมประดิษฐ์นี้ก็คือคลังคำสั่งเชื่อมโยงแบบพลวัตที่ถูกออกแบบมาให้สามารถติดต่อกับวิชวลเบสิคได้ โดยตัวควบคุมประดิษฐ์นี้จะต้องถูกสร้างให้ทำงานตอบสนองต่อข้อความของวินโดว์และวิชวลเบสิคได้อย่างหมาะสม นอกจากนี้ผู้สร้างควรจะเข้าใจวิธีการเรียกใช้ฟังก์ชัน API ทั้งของวินโดว์และวิชวลเบสิคด้วย (Microsoft Corp., 1993)

วิธีการพัฒนาตัวควบคุมประดิษฐ์ที่จะกล่าวถึงในหัวข้อถัดไป จะมีการยกตัวอย่างประกอบจากโปรแกรมที่พัฒนาขึ้นมาจริงด้วยภาษาซีของ Microsoft C/C++ Version 7.0 โดยจะแสดงเฉพาะโครงร่างหลักของโปรแกรม ซึ่งสามารถนำโครงร่างนี้ไปพัฒนาเป็นตัวควบคุมแบบอื่น ๆ ได้

วิธีการสร้างตัวควบคุมประดิษฐ์ จะคล้ายกับการเขียนโปรแกรมสำหรับวินโดว์ทั่วไป คือต้องมีการรับ ส่ง และตอบสนองต่อข้อความ และมีการเรียกใช้ฟังก์ชัน API ต่าง ๆ แต่สิ่งที่แตกต่างจากการเขียนโปรแกรมบนวินโดว์ คือ

- ตัวควบคุมประดิษฐ์ เป็นแฟ้มวินโดว์ DLL ซึ่งต้องถูกอ่านเข้ามา (load) เข้ามาในลักษณะพิเศษ

- ตัวควบคุมประดิษฐ์ ต้องอาศัยวิชวลเบสิคให้ช่วยทำงานบางอย่างให้และต้องมีการติดต่อกับวิชวลเบสิคด้วยวิธีการเฉพาะ

1. องค์ประกอบพื้นฐานของชั้นของตัวควบคุม (Control Class)

การสร้างตัวควบคุมประดิษฐ์ก็คือการสร้างชั้นของตัวควบคุมนั่นเอง เมื่อชั้นของตัวควบคุมถูกอ่านเข้ามาเข้ามาในโปรเจค ผู้เขียนโปรแกรมก็สามารถสร้างรูปเสมือน (Instance) ของชั้นได้หลายชุดเพื่อการใช้งานในโปรแกรมเดียวกัน เช่น ผู้พัฒนาสามารถที่จะสร้าง ช่องรับข้อความ (Text boxes) ที่แตกต่างกันได้เป็นจำนวนมาก โดยที่ช่องรับข้อความทั้งหมดนั้น ก็คือรูปเสมือนที่มาจากชั้นของช่องรับข้อความ (Text box class) เดียวกัน ชั้นของตัวควบคุมประกอบด้วย 2 ส่วนหลักคือ ต้นแบบของตัวควบคุม (Control Model) และกระบวนงานของตัวควบคุม (Control Procedure) (Microsoft Corp., 1993)

ต้นแบบของตัวควบคุมเป็นโครงสร้างภาษาซี ซึ่งกำหนดออยู่ในโปรแกรมเป็นดังนี้

```
MODEL modelHeap =
{
  VB_VERSION,                          // VB version being used
  0                                    // MODEL flags
  (PCTLPROC)HeapCtlProc,               // Control procedure
  CS_VREDRAW | CS_HREDRAW,             // Class style
  WS_BORDER |                          // Default Windows style
  WS_THICKFRAME | WS_HSCROLL ,         // Default Windows style
  sizeof(Heap),                        // Size of Heap structure
  IDBMP_HPCTL,                         // Palette bitmap ID
  "Heap",                              // Default control name
  "HPCTL",                             // Visual Basic class name
  NULL,                                // Parent class name
  Heap_Properties,                     // Property information table
  Heap_Events,                         // Event information table
  IPROP_Heap_HEAPINPUT,                // Default property
  IEVENT_Heap_NodeAdd,                 // Default event
  IPROP_Heap_MAXMIN                    // Property representing value of ctl
};
```

ต้นแบบของตัวควบคุมใช้เป็นส่วนที่บอกถึงลักษณะหลักของชั้นของตัวควบคุมที่เกี่ยวข้องกับวิชวลเบสิค ตัวอย่างเช่น ชื่อโดยปริยาย(default) ของตัวควบคุม และมีตัวชี้ (Pointer) ไปยังตารางที่สำคัญ 2 ตาราง คือ

- ตารางข้อมูลคุณสมบัติ ซึ่งจะแสดงค่าของคุณสมบัติข้อต่างๆ ของตัวควบคุม
- ตารางข้อมูลเหตุการณ์ ซึ่งจะแสดงเหตุการณ์ที่จะเกิดขึ้นได้ของตัวควบคุมพร้อมกับอาร์กิวเมนต์ (Argument) ที่สอดคล้องกับเหตุการณ์นั้น

กระบวนงานของตัวควบคุม เปรียบเทียบได้กับกระบวนงานของการเขียนโปรแกรมประยุกต์สำหรับวินโดว์ ซึ่งมีการทำงานหลักอยู่ที่การรับข้อความ และตอบสนองต่อข้อความนั้น สิ่งที่สำคัญเช่นกันก็คือจะต้องพิจารณาว่าเหตุการณ์ใดจะส่งเข้ามาเป็นเหตุการณ์ของตัวควบคุมนั้นในวิชวลเบสิค นอกจากนั้นก็จะเป็นการจัดการเกี่ยวกับการวาดรูปของตัวควบคุมออกมาทางจอภาพ กระบวนงานของตัวควบคุมจะมีลักษณะของโปรแกรม ดังนี้

```
LONG FAR PASCAL _export HeapCtlProc
(
    HCTL   hctl,
    HWND   hwnd,
    USHORT msg,
    USHORT wp,
    LONG   lp
)
{
switch (msg)
{
    case WM_NCCREATE:
        ::
    case VBM_METHOD:
        ::
    case WM_LBUTTONDBLCLK:
        ::
    ::
```

```
        case WM_PAINT:
        ::
                PAINTSTRUCT ps;
                BeginPaint(hwnd, &ps);
                PaintTree(hctl, hwnd, ps.hdc);
                EndPaint(hwnd, &ps);
        break;
    ::
    ::
    case VBM_SETPROPERTY:
            ::
    case VBM_HELP:
            ::
    case WM_NCDESTROY:
            ::
}
return VBDefControlProc(hctl, hwnd, msg, wp, lp);
}
```

เนื่องจากชั้นของตัวควบคุมเป็นส่วนหนึ่งของ DLL ดังนั้น จึงไม่มีฟังก์ชัน WinMain เหมือนกับโปรแกรมประยุกต์สำหรับวินโดว์ทั่วไป แต่ตัวควบคุมประดิษฐ์จะถูกเชื่อมโยงเข้ากับระบบงานที่กำลังทำงานอยู่

2. <u>ขั้นตอนการทำงานของชั้นของตัวควบคุม</u>

เมื่อมีการนำชั้นของตัวควบคุมเข้ามาใช้ในระบบ จะเกิดลำดับขั้นการทำงานดังนี้

1. วิชวลเบสิคจะสั่งดำเนินงานฟังก์ชัน VBINITCC ซึ่งจะต้องกำหนดให้เป็นฟังก์ชันที่ถูกเรียกใช้จากภายนอกได้ (Exported) ไว้ในแฟ้มตัวควบคุมประดิษฐ์ที่สร้างขึ้นมา

2. ฟังก์ชัน VBINITCC จะทำการขึ้นทะเบียน(Register)ต้นแบบของตัวควบคุม โดยการเรียกใช้ฟังก์ชันชื่อ VBRegisterModel พร้อมกับผ่านค่าที่อยู่ของโครงสร้างต้นแบบนี้เข้าไป

ซึ่งโครงสร้างนี้จะต้องมีการกำหนดไว้ก่อนแล้ว การกำหนดฟังก์ชัน VBINITCC และการเรียกใช้ฟังก์ชัน VBRegisterModel จะเป็นดังนี้

```
BOOL FAR PASCAL _export VBINITCC
(
  USHORT usVersion,
  BOOL   fRuntime
)
{
  fRuntime  = fRuntime;
  usVersion = usVersion;
  // Register control(s)
  return VBRegisterModel(hmodDLL, &modelHeap);
}
```

ภายในโครงสร้างต้นแบบนี้จะมีตัวชี้ไปยังที่อยู่ของกระบวนงานของตัวควบคุม ซึ่งก็ต้องกำหนดให้เป็นฟังก์ชันที่ถูกเรียกใช้จากภายนอกได้เช่นกัน

```
(PCTLPROC)HeapCtlProc,                    // Control procedure
```

ฟังก์ชัน VBINITCC สามารถทำการขึ้นทะเบียนต้นแบบของตัวควบคุมได้หลายครั้งในกรณีที่ต้องการนำชั้นของตัวควบคุมเข้ามาในหน่วยความจำหลายชุด

3. จากนั้น ผู้พัฒนาระบบงานจะสามารถสร้างรูปเสมือน (Instance) ของตัวควบคุมประดิษฐ์ได้เป็นจำนวนมากตามที่ต้องการ

4. เมื่อมีข้อความส่งมายังรูปเสมือนตัวใดตัวหนึ่งของชั้นแล้ว กระบวนงานของตัวควบคุมจะถูกเรียกใช้เพื่อจัดการข้อความที่ถูกส่งเข้ามาพร้อมกับแฮนเดิลของโครงสร้างของตัวควบคุม ซึ่งแฮนเดิลนี้จะบ่งบอกได้ว่าข้อความที่ส่งเข้ามานี้เป็นของรูปเสมือนชุดใด

```
LONG FAR PASCAL _export HeapCtlProc
(     HCTL   hctl,
      HWND   hwnd,
      USHORT msg,
      USHORT wp,
      LONG   lp          )
```

เนื่องจากตัวควบคุมประดิษฐ์เป็นแฟ้ม DLL จึงมีส่วนเริ่มต้นการทำงาน (Initialization)ที่ทำงานจริง ๆ เพียงครั้งเดียวที่มีการนำตัวควบคุมเข้ามาในหน่วยความจำ ด้วยคุณสมบัติของ DLL แฟ้มของตัวควบคุมจึงถูกใช้ร่วมกันได้ระหว่างรูปเสมือนต่าง ๆ ของวิชวลเบสิค

ฟังก์ชัน VBINITCC อาจถูกเรียกใช้ได้หลายครั้ง โดยรูปเสมือนแต่ละตัวของวิชวลเบสิคจะเรียกใช้ฟังก์ชันนี้เพียงครั้งเดียว

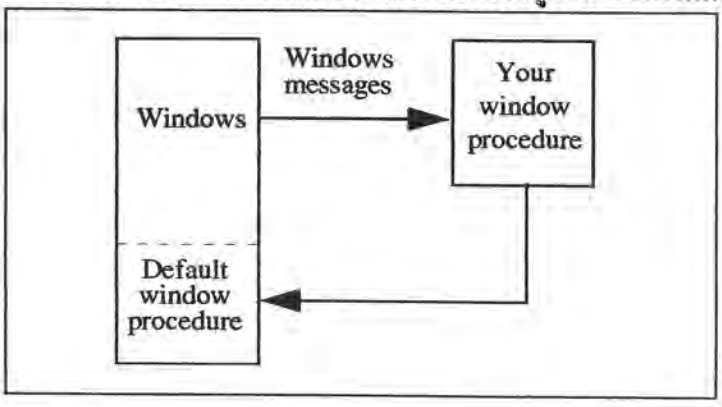
ในสภาพแวดล้อมขณะกำลังพัฒนาโปรแกรม แฟ้ม VB.EXE นั้นเป็นรูปเสมือนตัวหนึ่งของวิชวลเบสิค ซึ่งมีได้หลายตัวในขณะกำลังพัฒนาโปรแกรม

โปรแกรมประยุกต์ที่เขียนโดยวิชวลเบสิคแต่ละระบบก็นับเป็นรูปเสมือนตัวหนึ่งของวิชวลเบสิคเช่นกัน ดังนั้นเมื่อโปรแกรมประยุกต์มีการใช้ตัวควบคุมประดิษฐ์ก็จะมีการเรียกใช้ฟังก์ชัน VBINITCC เพื่อขึ้นทะเบียนต้นแบบของตัวควบคุมด้วยเช่นกัน

เมื่อมีการใช้ตัวควบคุมประดิษฐ์ในการพัฒนาโปรแกรม ผู้ใช้งานโปรแกรมประยุกต์นั้นจะต้องได้รับสำเนาของแฟ้มตัวควบคุมประดิษฐ์มาพร้อมกับโปรแกรมประยุกต์นั้นด้วย

3. การส่งข้อความไปยังตัวควบคุมของวิชวลเบสิค

การสร้างตัวควบคุมประดิษฐ์จะมีส่วนคล้ายกับการเขียนโปรแกรมสำหรับวินโดว์ ตรงที่งานส่วนมากจะเป็นการเขียนโปรแกรมเกี่ยวกับการรับ, การส่ง และการประมวลผลข้อความ แต่ก็ยังมีรายละเอียดของการพัฒนาโปรแกรมที่แตกต่างกันอยู่บ้าง ซึ่งจะได้กล่าวถึงในลำดับต่อไป



รูปที่ 3.9 แสดงการจัดการข้อความในโปรแกรมประยุกต์ของวินโดว์
(Microsoft Corp., 1993)

ในโปรแกรมสำหรับวินโดว์ทั่วไป กระบวนงานของวินโดว์ (Window Procedure) แต่ละโปรแกรมหรือแต่ละชั้นของวินโดว์จะได้รับข้อความต่าง ๆ เข้ามา กระบวนงานของวินโดว์

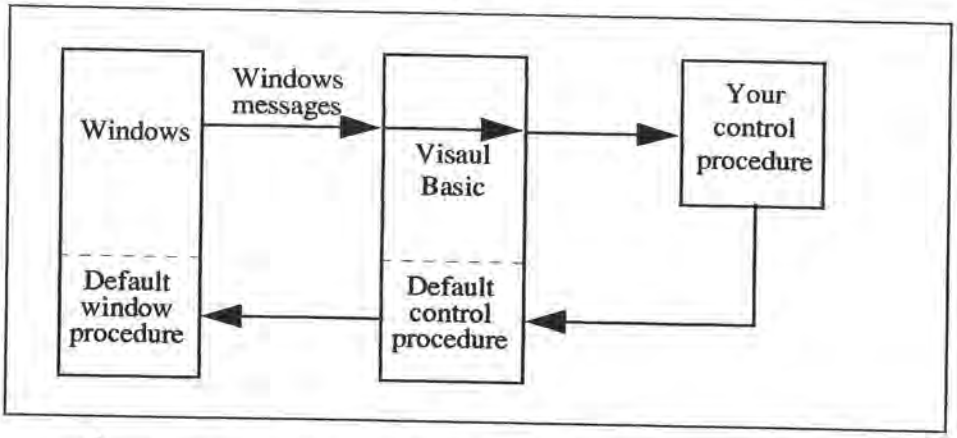
อาจสามารถจัดการข้อความเหล่านั้นได้เอง หรือส่งข้อความเหล่านี้ต่อไปยังส่วนประมวลผลข้อความโดยปริยายของวินโดว์ (Windows default message processor) สรุปได้ดังรูปที่ 3.9

ข้อความของวินโดว์จะทำให้กระบวนงานของวินโดว์ ทราบได้ว่ามีเหตุการณ์อะไรเกิดขึ้นบ้าง เช่น วินโดว์ได้ถูกสร้างขึ้นมา การเปลี่ยนขนาดของวินโดว์ การเคลื่อนย้ายที่ และการทำงานของเมาส์ เป็นต้น

เมื่อตัวควบคุมประดิษฐ์ถูกนำเข้ามาในหน่วยความจำ วินโดว์จะส่งข้อความไปยังตัวควบคุม เช่นเดียวกับที่ส่งไปยังโปรแกรมประยุกต์อื่นๆ แต่วิชวลเบสิคจะเป็นส่วนที่รับข้อความนั้นก่อน จากนั้นอาจจะส่งต่อไปยังกระบวนงานของตัวควบคุมหรือไม่ก็ได้ กระบวนงานของตัวควบคุมจะประมวลผลข้อความที่ได้รับหรืออาจส่งต่อไปให้ไปทำงานยังส่วนประมวลผลข้อความโดยปริยายของวิชวลเบสิค (Visual Basic default message processor) ซึ่งเป็นฟังก์ชันชื่อ VBDefControlProc ก็ได้ การเรียกใช้ฟังก์ชันการประมวลผลข้อความโดยปริยายของวิชวลเบสิคจะเป็นดังนี้

```
return VBDefControlProc(hctl, hwnd, msg, wp, lp);
```

การจัดการข้อความในวิชวลเบสิค สรุปได้ดังรูปที่ 3.10



รูปที่ 3.10 แสดงการจัดการข้อความในวิชวลเบสิค (Microsoft Corp., 1993)

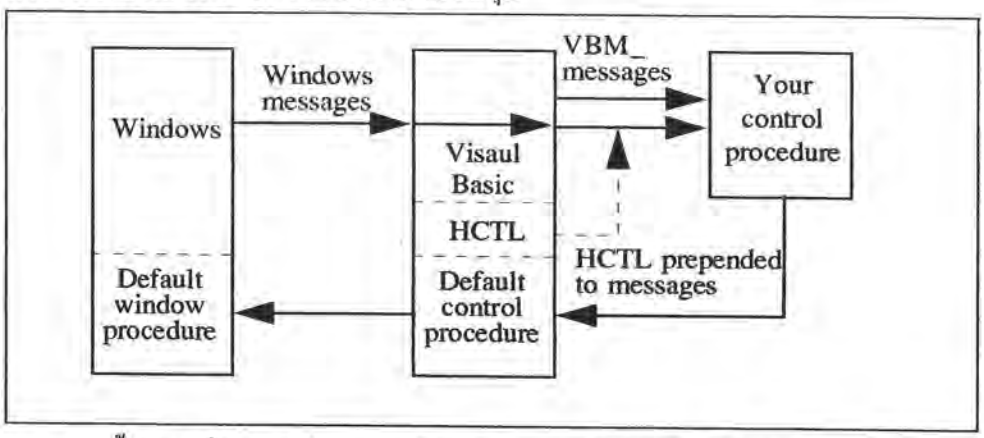
วิชวลเบสิคจะพิจารณาว่า จำเป็นต้องส่งข้อความต่อไปยังตัวควบคุมหรือไม่ ในบางกรณีวิชวลเบสิคสามารถจัดการข้อความนั้นแทนตัวควบคุมได้เลย เช่น ขณะที่อยู่ในช่วงของการพัฒนาโปรแกรม การกดแป้นอักขระต่าง ๆ การใช้เมาส์ การเปลี่ยนขนาด หรือ การเปลี่ยนคุณสมบัติของตัวควบคุม ข้อความจะไม่ถูกส่งต่อไปยังตัวควบคุม วิชวลเบสิคจะจัดการข้อความในกรณีนั้นเอง

เมื่อวิชวลเบสิคส่งข้อความต่อไปยังตัวควบคุม วิชวลเบสิคจะส่งแฮนเดิลของตัวควบคุมไปให้ด้วย โดยส่งเข้าไปเป็นอาร์กิวเมนต์ของฟังก์ชันกระบวนงานของตัวควบคุม ดังนี้

```
LONG FAR PASCAL _export HeapCtlProc
(
    HCTL   hctl,
    HWND   hwnd,
    USHORT msg,
    USHORT wp,
    LONG   lp
)
```

แฮนเดิลที่ส่งเข้ามานี้จะใช้ในการเข้าถึงโครงสร้างของแต่ละรูปเสมือนของตัวควบคุมนั้น โดยโครงสร้างนี้จะเก็บข้อมูลเฉพาะที่เป็นประโยชน์ต่อการทำงานของวิชวลเบสิค ซึ่งจะไม่มีโครงสร้างนี้ในโครงสร้างของวินโดว์ (Window structure)

ข้อความสามารถถูกส่งมาโดยตรงจากวิชวลเบสิคก็ได้ โดยข้อความที่ถูกส่งมาจากวิชวลเบสิคโดยตรงจะขึ้นต้นด้วย VBM เช่น VBM_SETPROPERTY จะเป็นข้อความที่ถูกส่งไปยังตัวควบคุมเมื่อมีการกำหนดคุณสมบัติของตัวควบคุม ในรูปที่ 3.11 จะแสดงขั้นตอนที่สมบูรณ์ของการส่งข้อความไปยังกระบวนงานของตัวควบคุม



รูปที่ 3.11 แสดงขั้นตอนที่สมบูรณ์ของการจัดการข้อความในวิชวลเบสิค (Microsoft Corp., 1993)

4. โครงสร้างของตัวควบคุม (Control Structure)

เราจะเข้าใจโครงสร้างของตัวควบคุมได้ง่ายขึ้นเมื่อไปเปรียบเทียบกับโครงสร้างของวินโดว์ เราสามารถติดต่อกับรูปเสมือนต่าง ๆ ของตัวควบคุม ได้ 2 วิธี คือ

- โดยแฮนเดิลของวินโดว์ (Window handle หรือ HWND)
- โดยแฮนเดิลของตัวควบคุม (Control handle หรือ HCTL)

แฮนเดิลของวินโดว์ ใช้สำหรับการเข้าถึงโครงสร้างของวินโดว์ ซึ่งเก็บข้อมูลเกี่ยวกับวินโดว์ของตัวควบคุม และเรียกใช้ฟังก์ชัน API ผ่านทางแฮนเดิลนี้ในการทำงานติดต่อกับตัวควบคุม

แฮนเดิลของตัวควบคุม ใช้สำหรับการเข้าถึงโครงสร้างของตัวควบคุม โดยวิชวลเบสิคจะจัดแบ่งพื้นที่ไว้สำหรับโครงสร้างของตัวควบคุมของแต่ละรูปเสมือนเมื่อมีการสร้างรูปเสมือนนั้นขึ้นมา โครงสร้างนี้จะเก็บข้อมูลต่าง ๆ เช่น ข้อมูลเกี่ยวกับคุณสมบัติของตัวควบคุม โดยที่ข้อมูลทั้งหมดนี้เกี่ยวข้องโดยตรงกับสถานะของรูปเสมือนแต่ละตัว

เราจะใช้ฟังก์ชัน API ของวิชวลเบสิคผ่านทางแฮนเดิลของตัวควบคุมในการจัดการกับข้อมูลในโครงสร้างของตัวควบคุม เช่นการจัดการเกี่ยวกับข้อมูลคุณสมบัติของตัวควบคุม ซึ่งจะมีวิธีการในโปรแกรมเป็นดังนี้

```
::
VBGetControlProperty(hctl, IPROP_HEAP_Dsptree, &tree);
::
```

นอกจากนั้น เรายังใช้แฮนเดิลของตัวควบคุมในการอ้างถึงอีกครั้งหนึ่ง (Dereference) โดยการเรียกใช้ฟังก์ชัน VBDerefControl ซึ่งฟังก์ชันนี้จะให้ค่าตัวชี้ที่อยู่ไปยังโครงสร้างของผู้เขียนโปรแกรม (Programmer-defined structure) สำหรับรูปเสมือนแต่ละตัวของตัวควบคุมที่กำลังใช้งานอยู่ การใช้ฟังก์ชัน VBDerefControl และการกำหนดโครงสร้างของผู้เขียนโปรแกรมจะมีการเขียนโปรแกรมดังนี้

```
// Heap control : Programmer-defined structure
typedef struct tagHeap
{
    BOOL anim ;
    ENUM maxin ;
    HSZ nrep ;
    short nval ;
    ::
    List_type hpdta ;
} Heap;

typedef Heap FAR * LPHeap
```

```
// Use Control Handle to dereference,
// which retrun the pointer to Programmer-defined structure
::
lptmp = (LPHeap) VBDerefControl(hctl) ;
::
```

โครงสร้างของผู้เขียนโปรแกรมเป็นส่วนที่เตรียมไว้สำหรับการใช้งานของผู้พัฒนาตัวควบคุม เช่น ใช้เก็บข้อมูลเกี่ยวกับคุณสมบัติที่กำหนดขึ้นมาใหม่ของตัวควบคุม ซึ่งขนาดพื้นที่ของโครงสร้างของผู้เขียนโปรแกรมได้มีการกำหนดไว้แน่นอนแล้วในต้นแบบของตัวควบคุม

```
MODEL modelHeap =
{
  VB_VERSION,                          // VB version being used
  ::
  sizeof(Heap),                        // Size of Heap structure
  ::
};
```

ผู้เขียนโปรแกรมจะต้องไม่กำหนดตัวแปรที่จะใช้เก็บข้อมูลโครงสร้างนี้เป็นแบบ Static หรือแบบ Global เพราะว่ากระบวนงานของตัวควบคุมจะถูกใช้งานร่วมกันระหว่างรูปเสมือนหลาย ๆ ตัวในชั้นเดียวกัน

5. <u>การทำงานร่วมกันระหว่างวิชวลเบสิคกับตัวควบคุม</u>

การทำงานของตัวควบคุมที่จะกล่าวถึงนี้ จะเป็นการอธิบายถึงการทำงานของฟังก์ชันกระบวนงานของตัวควบคุมในส่วนต่าง ๆ และส่วนของโปรแกรมที่นำมาแสดงเป็นตัวอย่างประกอบการอธิบายนี้ ก็เป็นบางส่วนของโปรแกรมที่อยู่ในฟังก์ชันนี้

วิชวลเบสิคมักจะมีการติดต่อกับต้นแบบของตัวควบคุมและกระบวนงานของตัวควบคุมอยู่บ่อย ๆ โดยส่วนมากจะติดต่อกันเรื่องของการสร้างและการวาดตัวควบคุม รวมทั้งการสนับสนุนในเรื่องคุณสมบัติและเหตุการณ์ต่าง ๆ ของตัวควบคุม

5.1 การสร้างและวาดตัวควบคุม

เมื่อจะมีการสร้างรูปเสมือนของชั้นตัวควบคุม วิชวลเบสิคจะสร้างแฮนเดิลของตัวควบคุมขึ้นมาก่อน จากนั้นจะเป็นการสร้างโครงสร้างของวินโดว์ ซึ่งจะได้แฮนเดิลของวินโดว์

มาด้วย (ยกเว้นกรณีของตัวควบคุมแบบกราฟิก) วินโดว์จะส่งข้อความ WM_NCCREATE และ WM_CREATE ขณะที่โครงสร้างของวินโดว์ได้ถูกสร้างขึ้นมา ซึ่งแสดงการเขียนโปรแกรมได้ดังนี้

```
case WM_NCCREATE:
{
   ::
   ::
}
```

ขณะที่ฟอร์ม(Form)ถูกนำเข้ามาในหน่วยความจำ ข้อมูลคุณสมบัติของตัวควบคุมจะถูกนำเข้ามาด้วย ถ้าตัวควบคุมนั้นเป็นส่วนหนึ่งของฟอร์ม หรือขณะที่มีการวาง (Paste) ตัวควบคุมมาจากคลิปบอร์ด (Clipboard) ก็เช่นกัน

ในขั้นสุดท้ายวิชวลเบสิคจะแสดงตัวควบคุมออกมาโดยการเรียกฟังก์ชัน ShowWindow ซึ่งจะทำให้กระบวนงานของตัวควบคุมได้รับข้อความ WM_PAINT (หรือ VBM_PAINT ในกรณีที่เป็นตัวควบคุมแบบกราฟิก) โดยแสดงการเขียนโปรแกรมได้ดังนี้

```
case WM_PAINT:
   ::
   PAINTSTRUCT ps;
   BeginPaint(hwnd, &ps);
   PaintTree(hctl, hwnd, ps.hdc);
   EndPaint(hwnd, &ps);
   ::
```

ผู้สร้างตัวควบคุมจะทำการวาดรูปของตัวควบคุมเมื่อได้รับข้อความ WM_PAINT จากวินโดว์ แต่ถ้าตัวควบคุมสร้างจากชั้นย่อย (Subclass) ของชั้นตัวควบคุมของวินโดว์ (Window control class) ก็จะไม่ต้องทำในส่วนของการวาดรูปนี้ เพราะว่าวินโดว์จะทำในส่วนนี้ให้อยู่แล้ว

การแสดงสัญรูป (Icon) ของตัวควบคุมในกล่องเครื่องมือ (Toolbox) จะไม่ใช่งานส่วนของการตอบสนองต่อข้อความของการวาดรูป วิชวลเบสิคจะทำงานในส่วนนี้เอง ผู้สร้างตัวควบคุมเพียงแต่กำหนดหมายเลขของรูปภาพ (IDs of bitmaps) ที่จะใช้ไว้ในต้นแบบของตัวควบคุมเท่านั้น ซึ่งทำได้ดังนี้

```
//-----------------------------------------------------------------
// Toolbox bitmap resource IDs numbers.
```

```
#define IDBMP_HPCTL             8000
#define IDBMP_HPCTLDOWN         8001
#define IDBMP_HPCTLMONO         8003
#define IDBMP_HPCTLEGA          8006

MODEL modelHeap =
{
  ::
  DBMP_HPCTL,                           // Palette bitmap ID
  ::
};
```

5.2 การทำงานในส่วนของคุณสมบัติ

คุณสมบัติในเรื่องต่าง ๆ จะมีการกำหนดไว้ในตารางข้อมูลคุณสมบัติ และอาจจะต้องเขียนโปรแกรมเพิ่มเติมไว้ในส่วนของกระบวนงานของตัวควบคุมบ้าง วิชวลเบสิคจะเกี่ยวข้องในเรื่องข้อมูลคุณสมบัติใน 3 ประการ ดังนี้

- การอ่านค่าคุณสมบัติ
- การกำหนดค่าคุณสมบัติ
- การอ่านเข้ามาในหน่วยความจำ(Load) และ การจัดเก็บ(Save) ค่าคุณสมบัติไว้ในจานแม่เหล็กหรือคลิปบอร์ด

ในแต่ละกรณี เราสามารถเลือกว่า จะให้ วิชวลเบสิคส่งข้อมูลเข้ามาโดยตรง หรือ ส่งเป็นข้อความเข้ามา หรือทำทั้ง 2 อย่างก็ได้ ตัวอย่างเช่น การกำหนดค่าคุณสมบัติเราสามารถเลือกที่จะให้วิชวลเบสิคส่งข้อความ VBM_SETPROPERTY เข้ามาเมื่อมีการกำหนดค่าคุณสมบัติใหม่

การตอบสนองต่อข้อความเป็นวิธีการที่มีความยืดหยุ่นมากกว่า แต่ต้องมีการเขียนโปรแกรมในส่วนนี้เพิ่มขึ้น แต่ถ้าจะให้วิชวลเบสิคโอนย้ายข้อมูลโดยตรง ก็จะต้องมีการกำหนดตำแหน่ง (Offset) ของคุณสมบัติในโครงสร้างของผู้เขียนโปรแกรมเพื่อใช้ในการเก็บค่าคุณสมบัตินั้น

ตัวอย่างของโปรแกรมในการจัดการคุณสมบัติ และการกำหนดตำแหน่งของคุณสมบัติในโครงสร้างของผู้เขียนโปรแกรม จะเป็นดังนี้

```
//-----------------------------------------------------------------------
PROPINFO Property_Animation =
{
    "Animation",
    DT_BOOL | PF_fGetData | PF_fSetData | PF_fSaveData,
    OFFSETIN(Heap, anim),
    0, NULL, 0
};
PROPINFO Property_MaxMin =
{
::
OFFSETIN(Heap, maxin),
::
};
::
PROPINFO Property_NodeVal =
{
::
OFFSETIN(Heap, nval),
};
//-----------------------------------------------------------------------

case VBM_SETPROPERTY:
    ::
    switch (wp)
    {
      case IPROP_HEAP_DspTree:
          ::
          return 0;
      case IPROP_HEAP_BuildHeap:
```

```
        ::
        VBDerefControl(hctl)->bldhp = (BOOL)lp;
        ::
    case IPROP_HEAP_NodeVal:
        ::
        VBGetControlProperty(hctl, IPROP_HEAP_Dsptree, &tree);
        FireNodeAdd(hctl, hwnd, (SHORT)lp, (SHORT)HIWORD(lp));
        return 0 ;
    ::
}
```

คุณสมบัติมาตรฐานเป็นจำนวนมากมีเตรียมไว้ในวิชวลเบสิคอยู่แล้ว โดยที่เราอาจต้องเขียนโปรแกรมเพิ่มอีกเพียงเล็กน้อยหรืออาจจะไม่ต้องเลย โดยส่วนมากจะให้ส่วนประมวลผลข้อความโดยปริยายของวิชวลเบสิคจัดการข้อความที่เกี่ยวข้อง ผู้พัฒนาตัวควบคุมสามารถเลือกและรวมคุณสมบัติมาตรฐานที่ต้องการเข้าไว้ในตารางข้อมูลคุณสมบัติของตัวควบคุมได้เลย การกำหนดตารางคุณสมบัติจะเป็นดังนี้

```
//-----------------------------------------------------------------------
// Property list
PPROPINFO Heap_Properties[] =
{
    PPROPINFO_STD_CTLNAME,
    PPROPINFO_STD_BACKCOLOR,
    PPROPINFO_STD_LEFT,
    PPROPINFO_STD_TOP,
    PPROPINFO_STD_VISIBLE,
    ::
    &Property_Animation,
    &Property_MaxMin,
    &Property_WaitVal,
    &Property_DspbyKey,
    ::
```

```
    PPROPINFO_STD_HWND,
    ::
    &Property_BuildHeap,
    NULL
};
```

5.3 การทำงานในส่วนของเหตุการณ์

เหตุการณ์ต่าง ๆ ของตัวควบคุมที่จะให้เป็นที่รู้จักได้ในวิชวลเบสิคจะต้องมีการกำหนดไว้ในตารางข้อมูลเหตุการณ์

```
//-----------------------------------------------------------------------
// Event list
PEVENTINFO Heap_Events[] =
{
 &Event_NodeAdd,
 &Event_NodeDel,
 PEVENTINFO_STD_DRAGDROP,
 PEVENTINFO_STD_DRAGOVER,
 NULL
};
//-----------------------------------------------------------------------
```

ถ้าเหตุการณ์ที่ต้องการไม่ใช่เหตุกาณ์มาตรฐาน เราจะต้องมีการเขียนโปรแกรมเพิ่มเติมไว้ในส่วนของกระบวนงานของตัวควบคุม เพื่อที่จะได้ใช้เหตุการณ์นั้น ๆ ได้ในวิชวลเบสิค ซึ่งตัวอย่างของโปรแกรมจะเป็นดังนี้

```
//-----------------------------------------------------------------------
case IPROP_HEAP_NodeVal:
    ::
    VBGetControlProperty(hctl, IPROP_HEAP_Dsptree, &tree);
    FireNodeAdd(hctl, hwnd, (SHORT)lp, (SHORT)HIWORD(lp));
    return 0 ;
//-----------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------
// Fire the NodeAdd event, passing the Node Key and Node String of the new node
// to event procedures.
VOID NEAR FireNodeAdd
(   HCTL  hctl,
    HWND  hwnd,
    SHORT x,
    SHORT y    )
{
NodeAddPARMS params;
USHORT    cbCaption, err;
char      strBuf[20];
::
params.NodeString = VBCreateHlstr(strBuf, cbCaption);
::
err = VBFireEvent(hctl, IEVENT_HEAP_NodeAdd, &params);
VBDestroyHlstr(params.NodeString);
}
```

เราจะใช้ฟังก์ชัน VBFireEvent เพื่อให้วิชวลเบสิคได้รู้จักเหตุการณ์ที่เกิดขึ้น และถ้ามีฟังก์ชันอื่นที่เกี่ยวกับการประมวลผลเหตุการณ์ของวิชวลเบสิคก็จะให้ทำงานต่อเนื่องกับฟังก์ชันนี้ไปด้วย เราจะทำให้เหตุการณ์ต่าง ๆ ที่ต้องการให้เกิดขึ้นในวิชวลเบสิคได้ในขณะประมวลผลข้อความที่มาจากวินโดว์และวิชวลเบสิค

ในวิชวลเบสิคได้เตรียมเหตุการณ์ที่เป็นมาตรฐานไว้ให้แล้วเป็นจำนวนมาก โดยที่เราไม่จำเป็นต้องเขียนโปรแกรมเพิ่มเติมเลย เพียงแต่ให้ส่วนประมวลผลข้อความโดยปริยายของวิชวลเบสิคตอบสนองต่อข้อความที่เกี่ยวข้อง เราเพียงแต่เลือกเหตุการณ์มาตรฐานที่เราต้องการไว้ในตารางข้อมูลเหตุการณ์

5.4 สัญรูปที่เป็นตัวแทนของตัวควบคุมที่ปรากฏในกล่องเครื่องมือ

เราควรจะสร้างสัญรูปที่เป็นตัวแทนของตัวควบคุมประดิษฐ์ เพื่อที่จะแสดงอยู่ในกล่องเครื่องมือของวิชวลเบสิคได้ โดยสัญรูปนี้จะสามารถแสดงอยู่บนจอภาพสำหรับ กราฟิก-อแดปเตอร์ (Graphic Adapter) ได้ 3 ชนิดคือ VGA, Monochrome และ EGA โดยที่สัญรูปนี้

ต้องมีแฟ้มรูปภาพที่เป็นประเภทแผนภาพของจุด (bitmaps) 4 แฟ้ม โดย VGA ต้องการ 2 แฟ้ม สำหรับภาพที่ยกขึ้นมา (กรณีที่ยังไม่ได้ถูกเลือกโดยเมาส์) และสำหรับภาพที่ลึกลง (กรณีที่ถูกเลือกโดยเมาส์) ส่วน EGA และ monochrome ต้องการอย่างละ 1 แฟ้ม เพราะว่า วิชวลเบสิคจะกลับสลับรูปภาพให้เองเพื่อที่จะแสดงภาพที่ยกขึ้นและลึกลง แฟ้มสัญรูปเหล่านี้สามารถสร้างได้โดยใช้โปรแกรมสำหรับการวาดรูปเช่น Microsoft Image Editor หรือ Windows Paintbrush เป็นต้น

เพื่อความสะดวกในการเชื่อมโยงสัญรูปทั้ง 4 แฟ้มเข้าเป็นทรัพยากรของตัวควบคุมจึงควรมีการกำหนดหมายเลขประจำแฟ้มสัญรูปไว้ดังนี้

```
//-----------------------------------------------------------------------
// Toolbox bitmap resource IDs numbers.
#define IDBMP_HPCTL             8000
#define IDBMP_HPCTLDOWN         8001
#define IDBMP_HPCTLMONO         8003
#define IDBMP_HPCTLEGA          8006
```

<u>โครงร่างหลักของโปรแกรมตัวควบคุม</u>

```
1.  //-----------------------------------------------------------------------
2.  // HPCTL.H
3.  //-----------------------------------------------------------------------
4.  #ifndef ONCE
5.  #define ONCE
6.  ::
7.  typedef int Item_type;
8.  typedef LPSTR Item_rep;
9.  typedef struct list_tag {
10.   Item_rep ent1[MAXLIST];
11.   Item_type entry[MAXLIST];
12.   int count;
13. } List_type;
14. typedef struct Node_tag {
```

```
15.   HLSTR itmrep ;
16.   int itmkey ;
17.  } Node_type;
18.  ::
19.  ::
20.  //---------------------------------------------------------------------
21.  // Resource Information
22.  //---------------------------------------------------------------------
23.  // Toolbox bitmap resource IDs numbers.
24.  //---------------------------------------------------------------------
25.  #define IDBMP_HPCTL                8000
26.  #define IDBMP_HPCTLDOWN            8001
27.  #define IDBMP_HPCTLMONO            8003
28.  #define IDBMP_HPCTLEGA             8006
29.  //---------------------------------------------------------------------
30.  // Macro for referencing member of structure
31.  //---------------------------------------------------------------------
32.  #define OFFSETIN(struc,field)     ((USHORT)&(((struc *)0)->field))
33.  //---------------------------------------------------------------------
34.  // Function Prototypes
35.  //---------------------------------------------------------------------
36.  LONG FAR PASCAL _export HeapCtlProc(HCTL, HWND, USHORT, USHORT, LONG);
37.  BOOL FAR PASCAL _export InputNode (HWND, UINT, WPARAM, LPARAM) ;
38.  ::
39.  ::
40.  //---------------------------------------------------------------------
41.  // Heap control data and structure
42.  //---------------------------------------------------------------------
43.  typedef struct tagHeap
44.    {
```

```
45.   BOOL anim ;
46.   ENUM maxin ;
47.   HSZ nrep ;
48.   short nval ;
49.   short dval ;
50.   short wval ;
51.   short cnod ;
52.   BOOL dspkey ;
53.   BOOL dsptree ;
54.   BOOL bldhp ;
55.   HFONT hfont ;
56.   List_type hpdta ;
57.   } Heap;
58. typedef Heap FAR * LPHeap;
59. #define LpcircDEREF(hctl)       ((LPCIRC)VBDerefControl(hctl))
60. //---------------------------------------------------------------------
61. // Property info
62. //---------------------------------------------------------------------
63. #define MAX_ORDER       0
64. #define MIN_ORDER       1
65. #define LIST_MAX          1
66. //---------------------------------------------------------------------
67. // Define the consecutive indicies for the properties
68. //---------------------------------------------------------------------
69. #define IPROP_HEAP_CTLNAME                  0
70. #define IPROP_HEAP_BACKCOLOR                1
71. #define IPROP_HEAP_LEFT                     2
72. #define IPROP_HEAP_TOP                      3
73. #define IPROP_HEAP_WIDTH                    4
74. #define IPROP_HEAP_HEIGHT                   5
```

```
75. #define IPROP_HEAP_VISIBLE                6
76. ::
77. #define IPROP_HEAP_Animation              9
78. #define IPROP_HEAP_MAXMIN                 10
79. #define IPROP_HEAP_HEAPINPUT              11
80. #define IPROP_HEAP_NodeRep                12
81. #define IPROP_HEAP_NodeVal                13
82. #define IPROP_HEAP_DelVal                 14
83. #define IPROP_HEAP_WaitVal                15
84. #define IPROP_HEAP_DspbyKey               16
85. #define IPROP_HEAP_HWND                   17
86. ::
87. #define IPROP_HEAP_BuildHeap              27
88. //-------------------------------------------------------------------------
89. // Event list
90. //-------------------------------------------------------------------------
91. // Define the consecutive indicies for the events
92. //-------------------------------------------------------------------------
93. #define IEVENT_HEAP_NodeAdd               0
94. #define IEVENT_HEAP_NodeDel               1
95. #define IEVENT_HEAP_DRAGDROP              2
96. #define IEVENT_HEAP_DRAGOVER              3
97. #endif  // ONCE
98. //-------------------------------------------------------------------------
99. //-------------------------------------------------------------------------
100. // Property info
101. //-------------------------------------------------------------------------
102. PROPINFO Property_Animation =
103. {
104.   "Animation",
```

```
105.  DT_BOOL | PF_fGetData | PF_fSetData | PF_fSaveData,
106.  OFFSETIN(Heap, anim),
107.  0, NULL, 0
108. };
109. PROPINFO Property_MaxMin =
110. {
111.  "HeapOrder",
112.  DT_ENUM | PF_fGetData | PF_fSetMsg | PF_fSaveData,
113.  OFFSETIN(Heap, maxin),
114.  MAX_ORDER, "Max on Top\0" "Min on Top\0", LIST_MAX
115. };
116. ::
117. ::
118. PROPINFO Property_NodeVal =
119. {
120.  "Nodeval",
121.  DT_SHORT | PF_fGetData | PF_fSetData | PF_fSetMsg | PF_fNoShow,
122.  OFFSETIN(Heap, nval),
123.  0, NULL, 0
124. };
125. ::
126. PROPINFO Property_DspTree =
127. {
128.  "DisplayTree",
129.  DT_BOOL | PF_fGetData | PF_fSetMsg | PF_fSaveData ,
130.  OFFSETIN(Heap, dsptree),
131.  1, NULL, 0
132. };
133. PROPINFO Property_BuildHeap =
134. {
```

```
135.  "BuildHeap",
136.  DT_BOOL | PF_fGetData | PF_fSetMsg | PF_fSaveData ,
137.  OFFSETIN(Heap, bldhp),
138.  0, NULL, 0
139. };
140. //-------------------------------------------------------------------------
141. // Property list
142. //-------------------------------------------------------------------------
143. PPROPINFO Heap_Properties[] =
144.  {
145.  PPROPINFO_STD_CTLNAME,
146.  PPROPINFO_STD_BACKCOLOR,
147.  PPROPINFO_STD_LEFT,
148.  PPROPINFO_STD_TOP,
149.  PPROPINFO_STD_WIDTH,
150.  PPROPINFO_STD_HEIGHT,
151.  PPROPINFO_STD_VISIBLE,
152.  ::
153.  &Property_Animation,
154.  &Property_MaxMin,
155.  &Property_HeapInput,
156.  &Property_NodeRep,
157.  &Property_NodeVal,
158.  &Property_DelVal,
159.  &Property_WaitVal,
160.  &Property_DspbyKey,
161.  PPROPINFO_STD_HWND,
162.  ::
163.  &Property_BuildHeap,
164.  NULL
```

```
165.  };
166. //---------------------------------------------------------------------
167. // Event info
168. //---------------------------------------------------------------------
169. WORD Paramtypes_NodeAdd[] = {ET_R4, ET_R4, ET_SD};
170. EVENTINFO Event_NodeAdd =
171. {
172.  "NodeAdd",
173.  3,
174.  6,
175.  Paramtypes_NodeAdd,
176.  "X As Single,Y As Single,NodeRep As String"
177. };
178. ::
179. ::
180. //---------------------------------------------------------------------
181. // Event list
182. //---------------------------------------------------------------------
183. PEVENTINFO Heap_Events[] =
184.  {
185.  &Event_NodeAdd,
186.  &Event_NodeDel,
187.  PEVENTINFO_STD_DRAGDROP,
188.  PEVENTINFO_STD_DRAGOVER,
189.  NULL
190. };
191. //---------------------------------------------------------------------
192. // Model structure
193. //---------------------------------------------------------------------
194. // Define the control model (using the event and property structures).
```

```
195. //---------------------------------------------------------------------
196. MODEL modelHeap =
197.  {
198.  VB_VERSION,                          // VB version being used
199.  0                                    // MODEL flags
200.  (PCTLPROC)HeapCtlProc,               // Control procedure
201.  CS_VREDRAW | CS_HREDRAW,             // Class style
202.  WS_BORDER |                          // Default Windows style
203.  WS_THICKFRAME | WS_HSCROLL ,         // Default Windows style
204.  sizeof(Heap),                       // Size of Heap structure
205.  IDBMP_HPCTL,                        // Palette bitmap ID
206.  "Heap",                             // Default control name
207.  "HPCTL",                            // Visual Basic class name
208.  NULL,                               // Parent class name
209.  Heap_Properties,                    // Property information table
210.  Heap_Events,                        // Event information table
211.  IPROP_Heap_HEAPINPUT,               // Default property
212.  IEVENT_Heap_NodeAdd,                // Default event
213.  IPROP_Heap_MAXMIN                   // Property representing value of ctl
214.  };
215. //---------------------------------- End of File  HPCTL.H ----------------------------------
```

```
1   //-------------------------------------------------------------------------------
2   // Contains control procedure for HPCTL control  :  HPCTL.CPP
3   //-------------------------------------------------------------------------------
4   #include <windows.h>
5   #include <vbapi.h>
6   #include "hpctl.h"
7   #include "vbx.h"
8
```

```
9   //-------------------------------------------------------------------
10  // Heap Control Procedure
11  //-------------------------------------------------------------------
12  LONG FAR PASCAL _export HeapCtlProc
13  (
14    HCTL   hctl,
15    HWND   hwnd,
16    USHORT msg,
17    USHORT wp,
18    LONG   lp
19  )
20  {
21  switch (msg)
22  {
23    case WM_NCCREATE:
24    {
25          LPHeap lptmp ;
26          lptmp = (LPHeap) VBDerefControl(hctl) ;
27          ::
28          break;
29    }
30    case VBM_METHOD:
31    {
32          ::
33          break;
34    }
35    case WM_LBUTTONDBLCLK:
36    {
37          ::
38          return 0 ;
```

```
39    }
40    case WM_RBUTTONDBLCLK:
41    {
42          ::
43          break ;
44    }
45    ::
46    ::
47    case WM_PAINT:
48    ::
49          PAINTSTRUCT ps;
50          BeginPaint(hwnd, &ps);
51          PaintTree(hctl, hwnd, ps.hdc);
52          EndPaint(hwnd, &ps);
53          break;
54    ::
55    ::
56    case VBM_SETPROPERTY:
57          switch (wp)
58          {
59            LPHeap lptmp ;
60            List_type FAR * hp ;
61            int i ;
62            case IPROP_HEAP_DspTree:
63            {
64               ::
65               return 0;
66            }
67            case IPROP_HEAP_BuildHeap:
68               ::
```

```
69              VBDerefControl(hctl)->bldhp = (BOOL)lp;
70              ::
71            case IPROP_HEAP_NodeVal:
72              ::
73              VBGetControlProperty(hctl, IPROP_HEAP_Dsptree, &tree);
74              FireNodeAdd(hctl, hwnd, (SHORT)lp, (SHORT)HIWORD(lp));
75              return 0 ;
76            ::
77          }
78          break;
79  case VBM_HELP:
80          switch (LOBYTE(wp))
81          {
82            case VBHELP_PROP:
83            ::
84            switch (HIBYTE(wp))
85            {
86              ::
87              case IPROP_HEAP_DspbyKey:
88                DisplayHelpTopic('P', "FlashColor");
89                return 0L;
90            }
91            break;
92            case VBHELP_EVT:
93            switch (HIBYTE(wp))
94            {
95              case IEVENT_HEAP_NodeAdd:
96                DisplayHelpTopic('V', "ClickIn");
97                return 0L;
98              ::
```

```
99                  ::
100               }
101               break;
102               case VBHELP_CTL:
103                  DisplayHelpTopic('H', "HPCTL");
104                  return 0L;
105            }
106            break;
107   ::
108   case WM_NCDESTROY:
109            LPHeap lptmp = LpcircDEREF(hctl) ;
110            if (lptmp->nrep)
111              VBDestroyHsz(lptmp->nrep) ;
112              break ;
113 }
114 return VBDefControlProc(hctl, hwnd, msg, wp, lp);
115 }
116
117 //-------------------------------------------------------------------
118 // TYPEDEF for parameters to the Node Add event.
119 //-------------------------------------------------------------------
120 typedef struct tagNodeAddPARMS
121 {
122 HLSTR   NodeString;
123 int far  *NodeKey;
124 LPVOID     Index;
125 } NodeAddPARMS;
126
127 //-------------------------------------------------------------------
128 // Fire the NodeAdd event, passing the Node Key and Node String of the new node
```

```
129 // to event procedures.
130 //---------------------------------------------------------------------
131 VOID NEAR FireNodeAdd
132 (
133 HCTL  hctl,
134 HWND  hwnd,
135 SHORT x,
136 SHORT y
137 )
138 {
139 NodeAddPARMS params;
140 USHORT cbCaption, err;
141 char        strBuf[20];
142 ::
143 params.NodeString = VBCreateHlstr(strBuf, cbCaption);
144 ::
145 err = VBFireEvent(hctl, IEVENT_HEAP_NodeAdd, &params);
146 VBDestroyHlstr(params.NodeString);
147 }
148
149 //---------------------------------------------------------------------
150 // Initialize library. This routine is called when the first client loads the DLL.
151 //---------------------------------------------------------------------
152 int FAR PASCAL LibMain
153 (
154 HANDLE hModule,
155 WORD   wDataSeg,
156 WORD   cbHeapSize,
157 LPSTR  lpszCmdLine
158 )
```

```
159 {
160 hmodDLL = hModule;
161 return 1;
162 }
163 //---------------------------------------------------------------------
164 // Register custom control. This routine is called by VB
165 // when the custom control DLL is loaded for use.
166 //---------------------------------------------------------------------
167 BOOL FAR PASCAL _export VBINITCC
168 (
169 USHORT usVersion,
170 BOOL   fRuntime
171 )
172 {
173 fRuntime  = fRuntime;
174 usVersion = usVersion;
175 // Register control(s)
176 return VBRegisterModel(hmodDLL, &modelHeap);
177 }
178 //---------------------------------------------------------------------
179 // Unregister custom control.  This routine is called by VB when the custom
180 // control DLL is being unloaded.
181 //---------------------------------------------------------------------
182 VOID FAR PASCAL _export VBTERMCC
183 (
184     VOID
185 )
186 {
187   --cVbxUsers;
188   UnregisterClass(CLASS_FLASHPOPUP, hmodDLL);
```

```
189    return;
190 }
191 //---------------------------------------- End of File  HPCTL.CPP ------------------------------
```