บทที่ 5

การพัฒนาและทดสอบตัวควบคุม

ความรู้เบื้องต้นที่จำเป็นในการพัฒนาตัวควบคุม

การที่จะพัฒนาตัวควบคุมได้อย่างมีประสิทธิภาพ ผู้พัฒนาตัวควบคุมควรจะมีความรู้เบื้องต้นในหัวข้อดังต่อไปนี้

1. การเขียนโปรแกรมประยุกต์สำหรับวินโดว์โดยใช้ Microsoft Windows Software Development Kit หรือ SDK ผู้พัฒนาควรมีความเข้าใจในหลักการเขียนโปรแกรมสำหรับวินโดว์ รวมทั้งการใช้แฮนเดิลต่าง ๆ ข้อความ การใช้อุปกรณ์ต่าง ๆ (Device Context) การใช้ทรัพยากรต่าง ๆ และการใช้ฟังก์ชันวินโดว์ API

2. มีความชำนาญในการเขียนโปรแกรมภาษา C โดยเฉพาะอย่างยิ่ง ควรมีความเข้าใจอย่างดีถึงการเขียนโปรแกรมเชิงโครงสร้างและการใช้ตัวชี้ที่อยู่ (Pointer) หรืออาจเป็นภาษาอื่นที่สามารถสร้างโปรแกรมสำหรับวินโดว์ได้ก็ใช้ได้เช่นกัน

3. มีความเข้าใจในการเขียนโปรแกรมด้วยวิชวลเบสิค รวมทั้งการใช้งานในด้านคุณสมบัติของตัวควบคุม เหตุการณ์ และวิธีต่าง ๆ

ซอฟต์แวร์ที่จำเป็นในการพัฒนาตัวควบคุม

ซอฟต์แวร์ที่จำเป็นต้องใช้ในการพัฒนาตัวควบคุม ได้แก่

1. ชุดพัฒนาโปรแกรม Microsoft Windows Software Development Kit หรือ SDK สำหรับวินโดว์รุ่น 3.1

2. ภาษาโปรแกรมที่สามารถสร้าง DLL ของวินโดว์ได้ ซึ่งในวิทยานิพนธ์นี้ใช้ Microsoft C/C++ รุ่น 7.0

3. วิชวลเบสิคสำหรับวินโดว์ ซึ่งในวิทยานิพนธ์นี้ใช้รุ่น 3.0 เพื่อใช้ในการทดสอบตัวควบคุมที่พัฒนาขึ้นมา รวมทั้งสารบบ (Directory) ที่ชื่อ CDK ซึ่งมากับวิชวลเบสิค เป็นสารบบที่มีแฟ้มที่จำเป็นต้องใช้งานและเป็นตัวอย่างในการพัฒนาตัวควบคุม ซึ่งได้แก่

- VBAPI.LIB เป็นแฟ้มที่เก็บฟังก์ชัน API ทั้งหมดของวิชวลเบสิค โดยฟังก์ชันเหล่านี้จะมีชื่อขึ้นต้นด้วย "VB" ในการพัฒนาตัวควบคุม จำเป็นต้องมีการเรียกใช้ฟังก์ชันเหล่านี้บางฟังก์ชัน ดังนั้นในขณะทำการสร้างแฟ้มของตัวควบคุม โปรแกรมเชื่อมโยง (Linker) จะทำการเชื่อมโยงฟังก์ชันเหล่านี้เข้ากับแฟ้มของตัวควบคุมด้วย (.VBX)

- VPAPI.H เป็นแฟ้มที่เก็บต้นแบบของฟังก์ชันที่อยู่ในแฟ้ม VBAPI.LIB ทั้งหมด ตลอดจนข้อความ (Messages), โครงสร้าง, ประเภทของข้อมูล และค่าคงที่ต่าง ๆ ที่จำเป็นต้องใช้เพิ่มเติมนอกเหนือไปจากที่มีอยู่ในชุดพัฒนาโปรแกรม SDK

นอกจากนี้ ในสารบบ CDK ยังมีสารบบย่อยที่เป็นตัวอย่างของการพัฒนาตัวควบคุมในลักษณะต่าง ๆ ซึ่งได้แก่

- CIRC1 เป็นตัวอย่างสำหรับการพัฒนาตัวควบคุมอย่างง่าย ๆ ผู้พัฒนาสามารถใช้ตัวอย่างนี้เป็นเค้าโครงในการพัฒนาตัวควบคุมอื่น ๆ ได้

- CIRC2 เป็นตัวอย่างที่พัฒนาเพิ่มเติมจาก CIRC1 โดยมีการวาดรูปวงกลม, การระบายสีพื้นหลัง และเมื่อมีการคลิกเมาส์ภายในวงกลม ก็จะเกิดเป็นเหตุการณ์เข้าไปในวิชวลเบสิค

- CIRC3 พัฒนาเพิ่มเติมจาก CIRC2 โดยมีการเขียนข้อความภายในวงกลม มีการจัดทำความช่วยเหลือที่เป็น Context-sensitive และแสดงตัวอย่างการสร้างหน้าต่างแบบผุดขึ้น (Pop-up Windows) ในการกำหนดค่าคุณสมบัติ

- CNTR สร้างตัวควบคุมที่เป็นตัวนับ (Counter Control) ซึ่งแสดงตัวเลขในรูปแบบของ Odometer

- PAL สร้างตัวควบคุมที่มีการใช้สี 256 สี

- PIX เป็นตัวควบคุมประเภทรูปภาพ ที่สามารถเกิดเหตุการณ์การคลิก (Click Event) ที่ตัวควบคุมได้

- PUSH เป็นการสร้างตัวควบคุมที่เป็นชั้นย่อยจากชั้น Push Button ของวินโดว์

- XLIST เป็นการสร้างตัวควบคุมที่เป็นชั้นย่อยจากชั้น List ของวินโดว์และแสดงตัวอย่างการใช้คุณสมบัติแบบแถวลำดับ (Property Array)

<u>การวางแผนในการพัฒนาตัวควบคุม</u>

การพัฒนาตัวควบคุม ควรจะต้องมีการออกแบบและวางแผนในประเด็นต่อไปนี้

1. มีการออกแบบและวางแผนวิธีการทำงานและลักษณะการแสดงผลของตัวควบคุม

2. การออกแบบและกำหนดคุณสมบัติของตัวควบคุมที่ต้องการ ซึ่งรวมทั้งคุณสมบัติที่เป็นมาตรฐานและคุณสมบัติที่สร้างขึ้นใหม่

3. การออกแบบและกำหนดเหตุการณ์ของตัวควบคุมที่เหมาะสม ซึ่งประกอบด้วยเหตุการณ์ที่เป็นมาตรฐานและเหตุการณ์ที่สร้างขึ้นมาใหม่

4. การกำหนดการทำงานของตัวควบคุมที่จะตอบสนองต่อวิธีมาตรฐานของวิชวลเบสิค วิธีบางชนิดไม่สามารถนำมาใช้กับตัวควบคุมที่สร้างขึ้นมาได้ และก็ไม่สามารถสร้างวิธีใหม่ขึ้นมาได้ จึงอาจทดแทนได้โดยการสร้างฟังก์ชันการทำงานไว้ในแฟ้มของตัวควบคุมขึ้นมาแทนได้

<u>แฟ้มที่สร้างขึ้นมาเพื่อประกอบกันเป็นแฟ้มของตัวควบคุม (.VBX)</u>

1. แฟ้ม VBX.H ใช้ในการกำหนดตารางและรายละเอียดของคุณสมบัติต่าง ๆ ตาราง และรายละเอียดของเหตุการณ์ต่าง ๆ และต้นแบบ (Model) ของตัวควบคุม

2. แฟ้ม HPCTL.H ใช้ในการกำหนดรูปแบบและประเภทของข้อมูลต่าง ๆ ที่ใช้ในโปรแกรม ค่าคงที่ต่าง ๆ กำหนดดัชนีที่เป็นตัวแทนของคุณสมบัติและเหตุการณ์ต่าง ๆ กำหนดโครงสร้างของผู้เขียนโปรแกรม (Programmer-Defined Structure) ตลอดจนต้นแบบของฟังก์ชันต่าง ๆ สำหรับใช้งานภายในตัวควบคุม

3. แฟ้ม HPCTL.CPP เป็นแฟ้มของตัวควบคุม ประกอบด้วยส่วนที่เป็นกระบวนงานของตัวควบคุม ตลอดจนฟังก์ชันที่ใช้ในการเริ่มต้นทำงานและจบการทำงานของตัวควบคุม

4. แฟ้ม DIALOG.CPP เป็นแฟ้มที่ใช้ในการจัดการการทำงานของตัวควบคุมในส่วนที่เกี่ยวกับกรอบรับข้อความ (Dialog box) ในการเพิ่มและการเปลี่ยนแปลงบัพข้อมูล

5. แฟ้ม HPMGT.CPP เป็นแฟ้มที่ใช้ในการจัดการโครงสร้างข้อมูลให้เป็นฮีปและการวาดบัพข้อมูลบางส่วนด้วย

6. แฟ้ม PAINTLN.CPP เป็นแฟ้มหลักที่ใช้ในการวาดโครงสร้างข้อมูลในแบบแถวลำดับ ซึ่งมีการวาดหลายรูปแบบ เช่นการวาดทีเดียวทั้งโครงสร้างข้อมูล การวาดทีละบัพของโครงสร้างข้อมูลแบบมีเวลาหน่วงระหว่างบัพ เพื่อแสดงการเปลี่ยนแปลงของโครงสร้างข้อมูล (Animation) หรือวาดเฉพาะบัพที่กำหนด

7. แฟ้ม PAINTTR.CPP มีวัตถุประสงค์การทำงานเช่นเดียวกับแฟ้ม PAINTLN.CPP แต่ต่างกันที่แฟ้มนี้มีการวาดโครงสร้างข้อมูลเป็นแบบแผนภาพของต้นไม้

8. แฟ้ม HPCTL.RC ใช้กำหนดทรัพยากร (Resource) ต่าง ๆ ที่ตัวควบคุม ใช้ในการทำงานเช่น กรอบรับข้อความในการเพิ่ม การเปลี่ยนแปลงบัพข้อมูล สัญรูป (Icon) ที่เป็นรูปภาพแทนตัวควบคุมในกล่องเครื่องมือ

9. แฟ้ม HPCTL.DEF ใช้กำหนดชื่อโมดูล (Module) ของโปรแกรม และสภาพแวดล้อมการทำงานของแฟ้มที่จะสร้างออกมาให้ทำงานภายใต้วินโดว์ ตลอดจนข้อมูลที่ใช้ในการกำหนดลักษณะของ Code Segment และ Data Segment รวมทั้งขนาดของ Heap และ Stack ของโปรแกรม

10. MAKEFILE ใช้ในการกำหนดเงื่อนไข วิธีการในการแปลภาษา และการเชื่อมโยงของโปรแกรมแปลภาษา (Compiler) และโปรแกรมเชื่อมโยง (Linker) ตลอดจนการแปลของโปรแกรมการแปลทรัพยากร (Resource Compiler) เพื่อให้ได้ขั้นตอนที่ถูกต้องในการสร้างแฟ้มของตัวควบคุม

11. HPCTL.BMP เป็นรูปภาพ 4 แฟ้มที่ใช้เป็นสัญรูปของตัวควบคุมสำหรับจอภาพ 3 ชนิด ตามที่เคยกล่าวไว้แล้วในบทที่ 4

12. HPFTN.CPP เป็นแฟ้มที่ประกอบด้วยฟังก์ชันที่จะให้เรียกใช้ได้จากภายนอกตัวควบคุม เช่นจากโปรแกรมวิชวลเบสิคสำหรับการกระทำการต่าง ๆ ต่อโครงสร้างข้อมูล ฟังก์ชันที่อยู่ในแฟ้มนี้จะถูกใช้งานแทนบางวิธีที่ใช้ไม่ได้ในตัวควบคุมนี้ โดยจะทำการเชื่อมโยงเข้าไปรวมกับแฟ้มของตัวควบคุม ตัวอย่างของฟังก์ชันภายในแฟ้มนี้ได้แก่ ฟังก์ชันในด้านการเพิ่ม การเปลี่ยนแปลง การอ่านบัพข้อมูล ฟังก์ชันในการรวมโครงสร้างข้อมูล 2 โครงสร้างเข้าด้วยกัน และฟังก์ชันการทำ Heapsort เป็นต้น

เหตุที่สร้างฟังก์ชันขึ้นมาใช้งานแทนบางวิธี เนื่องจากว่าเราไม่สามารถสร้างวิธีขึ้นมาใหม่เพื่อให้ทำงานเฉพาะกับตัวควบคุมของเรา อย่างไรก็ตามเราสามารถนำวิธีมาตรฐานที่มีอยู่มาประยุกต์ใช้โดยกำหนดให้กระบวนงานของตัวควบคุมทำงานตามที่ต้องการได้บ้าง แต่ก็อาจไม่สะดวกและไม่เหมาะสม เช่น วิธี AddItem สามารถมีอาร์กิวเมนต์ (Argument) ของข้อมูลได้เพียงตัวเดียว แต่การเพิ่มบัพข้อมูลใหม่สำหรับตัวควบคุมนี้ ต้องการข้อมูล 2 ประเภทด้วยกัน จึงไม่สามารถนำมาใช้กับตัวควบคุมนี้ได้

ฟังก์ชันในแฟ้มต่าง ๆ สามารถดูรายละเอียดเพิ่มเติมได้ในภาคผนวก ก ข และ ค ที่อยู่ท้ายเล่ม

การพัฒนาตัวควบคุมสำหรับโครงสร้างข้อมูลแบบฮีป

จากที่ได้สรุปถึงหน้าที่ของแฟ้มต่าง ๆ ที่จะนำมาประกอบเข้าด้วยกันในการสร้างแฟ้มของตัวควบคุมนั้น เมื่อมองเป็นภาพรวมของการเขียนโปรแกรมเพื่อสร้างตัวควบคุมแล้ว สามารถสรุปเป็นหัวข้อที่สำคัญได้ดังนี้

1. การกำหนดต้นแบบของตัวควบคุม

HPCTL.H บรรทัดที่ 196 - 214 เป็นการกำหนดต้นแบบของตัวควบคุม โดยแต่ละฟิลด์มีความหมายดังนี้

1. usVersion เป็นตัวเลขที่ใช้กำหนดรุ่นของวิชวลเบสิคที่จะใช้กับตัวควบคุมที่พัฒนาขึ้นมา ในที่นี้ใช้ค่าคงที่ VB_VERSION ที่กำหนดอยู่ในแฟ้ม VBAPI.H

2. fl เป็นตัวบ่งชี้ประเภทของต้นแบบ ซึ่งสามารถเลือกใช้ได้ตามตารางที่ 5.1 แต่ในตัวควบคุมนี้ไม่ใช้ จึงกำหนดให้เป็น 0

ค่าของ fl	ความหมาย
MODEL_fArrows	ให้ส่งข้อความคีย์บอร์ดเข้ามายังตัวควบคุมเมื่อมีการกดปุ่มลูกศรต่าง ๆ ถ้าไม่ได้ใช้ตัวบ่งชี้นี้ การกดปุ่มลูกศรต่าง ๆ จะใช้เป็นการเคลื่อนย้ายไปมาระหว่างตัวควบคุมต่าง ๆ
MODEL_fChildrenOk	เป็นการอนุญาตให้ผู้เขียนโปรแกรมวิชวลเบสิคสามารถนำตัวควบคุมอื่นมาวางอยู่ภายในตัวควบคุมนี้ได้ เช่นใช้ในการสร้างรูปภาพต่าง ๆ
MODEL_fDesInteract	ให้มีการส่งข้อความ WM_RBUTTONDOWN, WM_RBUTTONDBLCLICK และ WM_RBUTTONUP เข้ามายังตัวควบคุมขณะอยู่ในช่วงของการพัฒนาโปรแกรม (Design Time)
MODEL_fFocusOk	ให้มีโฟกัส (Focus) อยู่ที่ตัวควบคุมขณะอยู่ในช่วงของการดำเนินงานได้ (Run Time)
MODEL_fGraphical	เป็นการกำหนดให้ตัวควบคุมนี้เป็นตัวควบคุมทางด้านกราฟิก (Graphic)
MODEL_fInitMsg	กำหนดให้ตัวควบคุมสามารถรับข้อความ VBM_INITIALIZE ได้

ตารางที่ 5.1 ตัวบ่งชี้ของต้นแบบ (Model Flag) (Microsoft Corp., 1993)

MODEL_fInvisAtRun	กำหนดให้ไม่สามารถมองเห็นตัวควบคุมได้ในขณะดำเนินงาน โดยในขณะพัฒนาโปรแกรม วิชวลเบสิคจะจัดการให้แสดงสัญรูปของตัวควบคุมที่อยู่ในกล่องเครื่องมือให้เห็นแทน
MODEL_fLoadMsg	ให้มีการส่งข้อความ VBM_CREATED และ VBM_LOADED ไปยังตัวควบคุมได้
MODEL_fMnemonic	ให้มีการรับรู้เมื่อมีการกดปุ่มที่เป็นตัวแทนของตัวควบคุม และให้โฟกัสไปอยู่ที่ตัวควบคุมนั้น พร้อมทั้งส่งข้อความ VBM_MNEMONIC เข้าไปให้

ตารางที่ 5.1 (ต่อ) ตัวบ่งชี้ของต้นแบบ (Model Flag) (Microsoft Corp., 1993)

3. pctlproc ที่อยู่ในหน่วยความจำของกระบวนงานของตัวควบคุม (Control Procedure) สำหรับตัวควบคุมนี้คือที่อยู่ของฟังก์ชันที่ชื่อ HeapCtlProc

4. fsClassStyle กำหนดรูปแบบชั้นของวินโดว์สำหรับตัวควบคุม โดยค่าเหล่านี้จะขึ้นต้นด้วยตัวอักษร CS ซึ่งตัวควบคุมนี้ใช้ค่า CS_VREDRAW และ CS_HREDRAW

5. flWndStyle ค่าโดยปริยายของรูปแบบวินโดว์สำหรับตัวควบคุม โดยค่าเหล่านี้จะขึ้นต้นด้วยตัวอักษร WS สำหรับตัวควบคุมนี้กำหนดให้เป็น WS_BORDER, WS_THICKFRAME และ WS_HSCROLL

6. cbCtlExtra ขนาดของโครงสร้างของผู้เขียนโปรแกรม ในตัวควบคุมนี้มีขนาดเท่ากับขนาดของโครงสร้างที่ชื่อ HEAP

7. idBmpPalette หมายเลข (ID) เริ่มต้นของรูปภาพที่จะใช้เป็นสัญรูปของตัวควบคุมในกล่องเครื่องมือ จะกำหนดเป็นหมายเลขอะไรก็ได้แต่ต้องตรงกับที่กำหนดอยู่ในแฟ้มทรัพยากรของตัวควบคุม (สำหรับตัวควบคุมนี้คือแฟ้ม HPCTL.RC) ตัวควบคุมนี้ใช้ค่าคงที่ IDBMP_HPCTL ซึ่งกำหนดอยู่ในแฟ้ม HPCTL.H

8. npszDefCtlName เป็นสายอักขระ (String) ที่จะใช้เป็นชื่อโดยปริยายของตัวควบคุม โดยวิชวลเบสิคจะนำค่านี้ไปกำหนดค่าให้กับคุณสมบัติ Name ของตัวควบคุม ในตัวควบคุมนี้กำหนดค่าเป็น "Heap"

9. npszClassName ชื่อชั้นของตัวควบคุมที่จะใช้ในวิชวลเบสิค ในตัวควบคุมนี้ใช้ชื่อ "HPCTL"

10. npszParentClassName ชื่อชั้นที่ตัวควบคุมเข้าไปเป็นชั้นย่อยอีกทีหนึ่ง ในที่นี้ไม่ใช้

11. npproplist ตัวชี้ขนาด 16 บิตไปยังตารางคุณสมบัติของตัวควบคุม

12. npeventlist ตัวชี้ขนาด 16 บิตไปยังตารางเหตุการณ์ของตัวควบคุม

13. nDefProp ดัชนีของคุณสมบัติที่จะชี้ไปยังคุณสมบัติที่กำหนดนี้ก่อน เมื่อเรียกหน้าต่างของคุณสมบัติขึ้นมา

14. nDefEvent ดัชนีเหตุการณ์ที่จะแสดงเหตุการณ์นี้ขึ้นมาให้เห็นก่อนในหน้าต่างของการเขียนโปรแกรม

15. nValueProp ดัชนีของคุณสมบัติที่จะใช้เป็นคุณสมบัติโดยปริยาย เมื่อมีการกำหนดค่าคุณสมบัติของตัวควบคุมโดยไม่ได้ระบุชื่อของคุณสมบัติ ถ้าไม่ใช้ฟิลด์นี้ให้กำหนดค่าเป็น -1

ในตัวควบคุมนี้กำหนดให้ IPROP_HEAP_MAXMIN ซึ่งเป็นดัชนีของคุณสมบัติที่ชื่อ HeapOrder ดังนั้น เมื่อมีการกำหนด

Heap1 = TRUE จะมีความหมายเช่นเดียวกับ

Heap1.HeapOrder = TRUE

2. <u>การกำหนดคุณสมบัติของตัวควบคุม</u>

การกำหนดคุณสมบัติของตัวควบคุมสามารถนำคุณสมบัติมาตรฐานที่มีอยู่แล้วในวิชวลเบสิคมาเป็นคุณสมบัติของตัวควบคุมหรือสร้างคุณสมบัติใหม่เพิ่มเติมเข้ามาด้วยก็ได้ สำหรับคุณสมบัติมาตรฐานนั้นไม่ต้องมีการเขียนโปรแกรมเพิ่มเติมเลยก็ได้ เพียงแค่กำหนดคุณสมบัตินั้นไว้ในตารางคุณสมบัติพร้อมกับกำหนดดัชนีของคุณสมบัตินั้น แล้วให้ฟังก์ชัน VBDefControlProc ซึ่งเป็นฟังก์ชันการประมวลผลข้อความโดยปริยายของวิชวลเบสิคจัดการงานในส่วนของคุณสมบัติมาตรฐานให้ (HPCTL.C บรรทัดที่ 114) แต่คุณสมบัติที่สร้างขึ้นมาใหม่จะมีขั้นตอนเพิ่มเติมในการเขียนโปรแกรมดังนี้

2.1 ออกแบบและกำหนดการทำงานว่าคุณสมบัติใหม่ที่สร้างขึ้นมา จะมีผลต่อการทำงานหรือการแสดงผลของตัวควบคุมอย่างไร

2.2 จองเนื้อที่สำหรับเก็บค่าของคุณสมบัติใหม่ไว้ในโครงสร้างของผู้เขียนโปรแกรม (HPCTL.H บรรทัดที่ 43 - 57) โดยโครงสร้างของผู้เขียนโปรแกรมนี้จะเป็นข้อมูลโดยเฉพาะของแต่ละรูปเสมือน (Instance) ของตัวควบคุม เราจะเข้าถึงโครงสร้างของผู้เขียนโปรแกรมนี้ได้โดยการเรียกฟังก์ชัน VBDerefControl ซึ่งจะให้ค่าที่อยู่ของโครงสร้างนี้ออกมา เราจึงจะเข้าถึงข้อมูลต่าง ๆ ที่อยู่ในโครงสร้างนี้ได้ หลังจากที่มีการเรียกใช้ฟังก์ชัน API ของวิชวลเบสิค อาจทำให้ที่อยู่

ของโครงสร้างของผู้เขียนโปรแกรมเปลี่ยนแปลงไปได้ จึงต้องใช้ฟังก์ชัน VBDerefControl อีกครั้งหนึ่ง เพื่อให้ได้ที่อยู่ของโครงสร้างที่ถูกต้องเหมือนเดิม

2.3 การกำหนดรายละเอียดของคุณสมบัติใหม่ คุณสมบัติแต่ละข้อนั้นเป็นชุดของโครงสร้าง (Structure) ชนิดหนึ่ง ซึ่งรายละเอียดที่กำหนดอยู่ในโครงสร้างของคุณสมบัติได้แก่ ชื่อของคุณสมบัติ ตัวบ่งชี้ชนิดของข้อมูลและลักษณะการทำงานของคุณสมบัติ ตำแหน่งที่อยู่ในโครงสร้างของผู้เขียนโปรแกรมที่จะใช้เก็บค่าคุณสมบัติ และอื่น ๆ ซึ่งจะกล่าวถึงรายละเอียดของแต่ละฟิลด์ในภายหลัง ในที่นี้จะยกตัวอย่างคุณสมบัติ 2 ตัวที่ได้สร้างไว้มาประกอบการอธิบายคือ

```
PROPINFO Property_MaxMin =
{  "HeapOrder",
   DT_ENUM | PF_fGetData | PF_fSetMsg | PF_fSaveData,
   OFFSETIN(CIRC, maxin),
   0, MAX_ORDER, "Max on Top\0" "Min on Top\0", LIST_MAX  };
```

เป็นคุณสมบัติที่ใช้ในการกำหนดวิธีการจัดลำดับในโครงสร้างข้อมูล คุณสมบัตินี้จะมีชื่อปรากฏในหน้าต่างของคุณสมบัติว่า HeapOrder โดยตัวบ่งชี้กำหนดให้เป็นข้อมูลชนิด DT_ENUM (เป็นข้อมูลที่ใช้เป็นลำดับไปเช่น 0, 1, 2.....) ด้วยค่าของ PF_fGetData, | PF_fSetMsg | PF_fSaveData จะกำหนดให้วิชวลเบสิคส่งข้อความ (ข้อความ VBM_SETPROPERTY) เข้ามา เมื่อมีการกำหนดค่าคุณสมบัติ แต่ไม่ต้องส่งข้อความเข้ามาเมื่อมีการอ่านค่าคุณสมบัติและส่งค่าคุณสมบัติจากโครงสร้างของผู้เขียนโปรแกรมเข้ามาให้โดยตรง รวมทั้งให้วิชวลเบสิคทำงานในส่วนของการอ่านและบันทึกค่าของคุณสมบัติขึ้นและลงสู่จานแม่เหล็กให้เอง โดยค่าของคุณสมบัตินี้จะเก็บอยู่ในโครงสร้างของผู้เขียนโปรแกรม(ชื่อ HEAP)ในตำแหน่งที่ชื่อ MAXIN เมื่อเข้าไปกำหนดค่าของคุณสมบัตินี้ในหน้าต่างของคุณสมบัติ จะมีรายการที่เป็นข้อความขึ้นมาให้เลือกเป็น 2 รายการคือ Max on Top และ Min on Top และมีการตรวจสอบด้วยว่า ค่าของคุณสมบัติจะต้องไม่มากไปกว่าค่าของค่าคงที่ที่ชื่อ LIST_MAX

```
PROPINFO Property_DspTree =
{  "DisplayTree",
   DT_BOOL | PF_fGetData | PF_fSetMsg | PF_fSaveData ,
   OFFSETIN(CIRC, dsptree),
   0, 1, NULL, 0        };
```

เป็นคุณสมบัติที่ใช้ในการกำหนดให้การวาดแผนภาพของโครงสร้างข้อมูลให้เป็นแบบต้นไม้หรือแบบแถวลำดับ ชื่อของคุณสมบัติที่ปรากฏในหน้าต่างของคุณสมบัติคือ

DisplayTree ตัวบ่งชี้กำหนดให้เป็นข้อมูลชนิด DT_BOOL (เป็นข้อมูลประเภทตรรกคือ จริง กับ เท็จ) และวิชวลเบสิคจะทำงานติดต่อกับคุณสมบัตินี้ด้วยวิธีเดียวกับคุณสมบัติ HeapOrder ดังที่ได้กล่าวมาในข้างต้นเนื่องจากมีตัวบ่งชี้อื่น ๆ ที่เหมือนกัน โดยค่าของคุณสมบัตินี้จะเก็บอยู่ในโครงสร้างของผู้เขียนโปรแกรม (HEAP) ในตำแหน่งที่ชื่อ dsptree

2.4 การกำหนดคุณสมบัติในตารางคุณสมบัติ เป็นการนำคุณสมบัติต่าง ๆ ที่ได้กำหนดไว้ของตัวควบคุมมารวมอยู่ในตารางคุณสมบัติ ซึ่งประกอบด้วยคุณสมบัติที่เป็นมาตรฐานและคุณสมบัติที่สร้างขึ้นใหม่ สำหรับคุณสมบัติที่สร้างขึ้นใหม่ให้กำหนดเป็นที่อยู่ของโครงสร้างของคุณสมบัติ จากตัวอย่างของคุณสมบัติในข้อ 2.3 คุณสมบัติที่ชื่อ DisplayTree จะนำไปกำหนดเป็นคุณสมบัติหนึ่งในตารางคุณสมบัติได้เป็น

&Property_DspTree (HPCTL.H บรรทัดที่ 143 - 165)

2.5 การกำหนดค่าคงที่เพื่อเป็นดัชนีของคุณสมบัติ คุณสมบัติที่กำหนดไว้ในตารางคุณสมบัติ จะจัดให้เรียงลำดับอย่างไรก็ได้ เพราะว่าเมื่อแสดงหน้าต่างของคุณสมบัติในวิชวลเบสิคแล้วจะถูกเรียงลำดับตามตัวอักษรของชื่อของคุณสมบัติ แต่การกำหนดค่าคงที่เพื่อเป็นดัชนีของแต่ละคุณสมบัติ จะต้องกำหนดให้ค่าดัชนีของคุณสมบัติเรียงเป็นลำดับที่สอดคล้องกับลำดับที่ของคุณสมบัติที่เรียงอยู่ในตารางคุณสมบัติ โดยเริ่มนับจาก 0 จากคุณสมบัติในข้อ 2.4 จะกำหนดค่าคงที่ของดัชนีได้เป็น

#define IPROP_HEAP_DSPTREE 25 (HPCTL.H บรรทัดที่ 69 - 87)

2.6 โครงสร้างของคุณสมบัติเป็นโครงสร้างที่ชื่อ PROFINFO ประกอบด้วยแต่ละฟิลด์ดังนี้

1. npszName เป็นชื่อของคุณสมบัติที่ใช้ในการเขียนโปรแกรมวิชวลเบสิคและเป็นชื่อที่ปรากฏอยู่ในหน้าต่างคุณสมบัติ

2. fl เป็นตัวบ่งชี้ลักษณะของคุณสมบัติ ซึ่งมีได้หลายตัวประกอบกัน ตัวแรกจะใช้ในการกำหนดชนิดข้อมูลของค่าคุณสมบัติ ซึ่งเลือกได้เพียงตัวเดียวจากตารางที่ 5.2

ค่าของ fl	ความหมาย
DT_BOOL	เป็นข้อมูลทางตรรกศาสตร์ (Boolean) จะมีค่าเป็น 0 เมื่อเป็นเท็จ และมีค่าไม่เท่ากับ 0 เมื่อเป็นจริง
DT_COLOR	เป็นตัวเลขจำนวนเต็มแบบยาว (Long integer) เก็บข้อมูลที่เป็นค่าของสี ซึ่งเป็นค่าของ RGB โดยในหน้าต่างคุณสมบัติจะแสดงค่าเป็นเลขฐาน 16

ตารางที่ 5.2 ตัวบ่งชี้กำหนดชนิดของคุณสมบัติ (Microsoft Corp., 1993)

DT_ENUM	ตัวเลขจำนวนเต็มแบบสั้น (Short integer) มีค่าได้ตั้งแต่ 0 ถึง 255 เมื่อกำหนดให้เป็นข้อมูลชนิดนี้แล้ว จะทำให้สามารถใช้ฟิลด์ npszEnumList และ enumMax ที่จะกล่าวถึงต่อไปได้
DT_HLSTR	ใช้กับข้อมูลที่เป็นสายอักขระ (String) ของวิชวลเบสิคชนิด HLSTR โดยค่าของคุณสมบัติชนิดนี้จะเป็นเพียงส่วนที่จะอธิบายลักษณะของสายอักขระ (String Descriptors) สายอักขระชนิด HLSTR สามาถมี "\0" (Null Character) อยู่ภายในสายอักขระได้ ซึ่งต่างกับข้อมูลชนิด DT_HSZ ที่ต้องใช้ "\0" เป็นตัวปิดท้ายสายอักขระ
DT_HSZ	ใช้กับข้อมูลสายอักขระของวิชวลเบสิคที่เป็นชนิด HSZ
DT_LONG	เป็นจำนวนเต็มขนาด 32 บิต แบบมีเครื่องหมาย
DT_PICTURE	เป็นโครงสร้างแบบรูปภาพ (Picture structure) ที่ใช้แฮนเดิลในการอ้างถึงรูปภาพแต่ละรูป (HPIC)
DT_REAL	จำนวนจริง 4 ไบต์
DT_SHORT	จำนวนเต็ม 16 บิตแบบมีเครื่องหมาย
DT_XPOS	ค่าพิกัดในแนวแกนนอนของตำแหน่งภายในตัวควบคุม
DT_YPOS	ค่าพิกัดในแนวแกนตั้งของตำแหน่งภายในตัวควบคุม

ตารางที่ 5.2 (ต่อ) ตัวบ่งชี้กำหนดชนิดของคุณสมบัติ (Microsoft Corp., 1993)

สำหรับคุณสมบัติที่ใช้ตัวบ่งชี้ที่ DT_HLSTR, DT_HSZ หรือ DT_PICTURE เมื่อจะจบการทำงานของตัวควบคุม ควรจะทำการยกเลิกการจองใช้พื้นที่ในหน่วยความจำของคุณสมบัติที่ใช้ตัวบ่งชี้ดังกล่าวในขณะตอบสนองต่อข้อความ WM_NCDESTROY

ตัวบ่งชี้อื่น ๆ ที่ใช้ในการกำหนดวิธีการทำงานของวิชวลเบสิคในเรื่องเกี่ยวกับคุณสมบัติ จะสามารถนำตัวบ่งชี้หลาย ๆ ตัวมาใช้รวมกันได้ ซึ่งเลือกใช้ได้จากตารางที่ 5.3

ค่าของ fl ตัวอื่น ๆ	ความหมาย
PF_fDefVal	ให้วิชวลเบสิคไม่ต้องบันทึกค่าของคุณสมบัติลงสู่จานแม่เหล็ก ถ้าค่าของคุณสมบัตินั้นเท่ากับค่าของฟิลด์ dataDefault และขณะที่อ่านค่าของคุณสมบัติเข้าสู่หน่วยความจำ ถ้าค่าของคุณสมบัติไม่ได้ถูกบันทึกอยู่ในจานแม่เหล็ก ก็จะใช้ค่าจากฟิลด์ data_Defalut มากำหนดเป็นค่าของคุณสมบัติ
PF_fEditable	อนุญาตให้ผู้พัฒนาโปรแกรมสามารถแก้ไขค่าของคุณสมบัติในช่องรับข้อความได้เลย ถึงแม้ว่าจะมีการใช้รายการเลือก (Listbox) หรือหน้าต่างแบบผุดขึ้น (Pop-up window) ก็ตาม
PF_fGetData	กำหนดให้วิชวลเบสิคอ่านค่าคุณสมบัติจากโครงสร้างของผู้เขียนโปรแกรมเข้ามายังตัวควบคุมได้โดยตรง
PF_fGetHszMsg	กำหนดให้วิชวลเบสิคส่งข้อความ VBM_GETPROPERTYHSZ เข้ามายังตัวควบคุมเมื่อมีการแสดงค่าของคุณสมบัติในหน้าต่างของคุณสมบัติ
PF_fGetMsg	กำหนดให้วิชวลเบสิคส่งข้อความ VBM_GETPROPERTY เมื่อต้องการอ่านค่าของคุณสมบัติ
PF_fLoadMsgOnly	กำหนดให้วิชวลเบสิคส่งข้อความ VBM_LOADPROPERTY เมื่อมีการนำฟอร์มจากแฟ้มเข้าสู่หน่วยความจำ ข้อความนี้จะถูกส่งมาพร้อมกับแฮนเดิลของตำแหน่งที่ตั้งของแฟ้ม ต่อจากนั้น กระบวนงานของตัวควบคุมสามารถใช้แฮนเดิลนี้ในการอ่านค่าของคุณสมบัติเข้าสู่หน่วยความจำได้ โดยใช้ฟังก์ชัน VBReadFormFile
PF_fLoadDataOnly	กำหนดให้วิชวลเบสิคอ่านค่าของคุณสมบัติจากแฟ้มของฟอร์ม แต่ไม่อนุญาตให้บันทึกค่าของคุณสมบัติกลับลงไปในแฟ้มของฟอร์ม
PF_fNoInitDef	ไม่ให้วิชวลเบสิคกำหนดค่าของคุณสมบัติจากฟิลด์ dataDefault ในระหว่างที่มีการอ่านตัวควบคุมเข้าสู่หน่วยความจำ
PF_fNoRuntimeR	เป็นการกำหนดให้คุณสมบัตินี้สามารถเปลี่ยนแปลงค่าได้เท่านั้น (ในระหว่างของการดำเนินงาน) ไม่สามารถอ่านค่าของคุณสมบัติได้ หากมีการอ่านค่าคุณสมบัตินี้ ก็จะทำให้เกิดข้อผิดพลาดในระหว่างการดำเนินงาน (Runtime error) และถ้าใช้พร้อมกับตัวบ่งชี้ PF_fNoRuntimeW ก็หมายความ

ตารางที่ 5.3 ตัวบ่งชี้อื่น ๆ ที่ใช้ในการกำหนดการทำงานของวิชวลเบสิคในเรื่องเกี่ยวกับคุณสมบัติ (Microsoft Corp., 1993)

	ว่าคุณสมบัตินี้สามารถใช้ได้เฉพาะในระหว่างการพัฒนาโปรแกรม (Design time) เท่านั้น
PF_fNoRuntimeW	กำหนดให้คุณสมบัตินี้ถูกอ่านค่าได้เท่านั้น เปลี่ยนแปลงค่าในช่วงระหว่างการดำเนินงานไม่ได้
PF_fNoShow	จะไม่มีการแสดงคุณสมบัตินี้ให้เห็นในหน้าต่างของคุณสมบัติ
PF_fPreHwnd	กำหนดให้คุณสมบัติถูกอ่านเข้ามาก่อนที่โครงสร้างของวิชวลเบสิคจะถูกสร้างขึ้นมา โดยคุณสมบัติที่มีผลต่อรูปแบบของวินโดว์ (Window style) ควรจะกำหนดให้ใช้ตัวบ่งชี้ตัวนี้ด้วย
PF_fPropArray	กำหนดให้คุณสมบัติเป็นแบบแถวลำดับ ตัวบ่งชี้แบบนี้ต้องใช้พร้อมกับตัวบ่งชี้ PF_fNoShow
PF_fSaveData	เมื่อมีการบันทึกแฟ้มของฟอร์ม วิชวลเบสิคจะอ่านค่าของคุณสมบัติและบันทึกลงไปในแฟ้มให้เลย ในทำนองเดียวกัน เมื่อมีการนำตัวควบคุมเข้าสู่หน่วยความจำ วิชวลเบสิคจะอ่านค่าของคุณสมบัติจากแฟ้มที่อยู่ในจานแม่เหล็กโดยตรงและกำหนดค่าให้กับคุณสมบัตินั้น
PF_fSaveMsg	เมื่อมีการสั่งให้บันทึกฟอร์มลงสู่แฟ้ม วิชวลเบสิคจะส่งข้อความ VBM_SAVEPROPERTY มายังตัวควบคุมและส่งข้อความ VBM_LOADPROPERTY มายังตัวควบคุม เมื่อฟอร์มถูกอ่านจากแฟ้มเข้ามาสู่หน่วยความจำ
PF_fSetCheck	กำหนดให้วิชวลเบสิคส่งข้อความ VBM_CHECKPROPERTY เข้ามายังตัวควบคุมก่อนที่จะมีการกำหนดค่าให้กับคุณสมบัติ ซึ่งจะทำให้กระบวนงานของตัวควบคุมสามารถตรวจสอบความถูกต้องของค่าคุณสมบัติก่อนได้
PF_fSetData	ให้วิชวลเบสิคกำหนดค่าของคุณสมบัติเข้าไปในโครงสร้างของผู้เขียนโปรแกรมได้โดยตรง
PF_fSetMsg	กำหนดให้วิชวลเบสิคส่งข้อความ VBM_SETPROPERTY มายังตัวควบคุมเมื่อจะมีการพยายามกำหนดค่าของคุณสมบัติ

ตารางที่ 5.3 (ต่อ) ตัวบ่งชี้อื่น ๆ ที่ใช้ในการกำหนดการทำงานของวิชวลเบสิคในเรื่องเกี่ยวกับคุณสมบัติ (Microsoft Corp., 1993)

PF_fUpdateOnEdit	กำหนดให้ คุณสมบัติถูกบันทึกตลอดเวลาที่มีการใส่ค่าของคุณสมบัติแต่ละตัวอักษรในหน้าต่างของคุณสมบัติ ถ้าไม่ได้ใช้ตัวบ่งชี้นี้ ค่าของคุณสมบัติจะถูกกำหนดลงไปใหม่หลังจากที่ได้กดปุ่ม [Enter] แล้วเท่านั้น ตัวบ่งชี้แบบนี้เหมาะสำหรับคุณสมบัติที่เป็นสายอักขระ (String)

ตารางที่ 5.3 (ต่อ) ตัวบ่งชี้อื่น ๆ ที่ใช้ในการกำหนดการทำงานของวิชวลเบสิคในเรื่องเกี่ยวกับคุณสมบัติ (Microsoft Corp., 1993)

3. offsetData เป็นตำแหน่งที่อยู่ในโครงสร้างของผู้เขียนโปรแกรมเพื่อใช้ในการเก็บค่าของคุณสมบัติ

4. infodata ใช้ฟิลด์นี้เพื่อช่วยประหยัดการใช้เนื้อที่ในโครงสร้างของผู้เขียนโปรแกรม ซึ่งจะสามารถช่วยลดการใช้พื้นที่ให้เหลือเพียง 4 บิตหรือน้อยกว่านั้นได้ โดยใช้ได้กับตัวบ่งชี้ของคุณสมบัติชนิด DT_BOOL, DT_SHORT หรือ DT_ENUMเท่านั้น

5. dataDefault ใช้ฟิลด์นี้ประกอบตัวบ่งชี้ PF_fDefVal เพื่อกำหนดค่าที่จะต้องใช้เป็นส่วนมากของค่าคุณสมบัติ เมื่อมีการบันทึกตัวควบคุมลงสู่จานแม่เหล็ก ค่าของคุณสมบัตินี้จะไม่ถูกบันทึก ถ้ามีค่าของคุณสมบัติเท่ากับฟิลด์นี้

6. npszEnumList ใช้กับตัวบ่งชี้ชนิด DT_ENUM โดยใช้กำหนดสายอักขระของรายการเลือกของคุณสมบัติที่จะแสดงในหน้าต่างคุณสมบัติ และแต่ละรายการเลือกในฟิลด์นี้จะถูกคั่นด้วยเครื่องหมาย "\0" ฟิลด์นี้จะกำหนดให้เป็น NULL ถ้าไม่ใช้รายการเลือกใด ๆ

7. enumMax ฟิลด์นี้ใช้กับตัวบ่งชี้ชนิด DT_ENUM โดยค่าในฟิลด์นี้จะใช้เป็นค่าสูงสุดของค่าคุณสมบัติที่เป็นไปได้ กำหนดให้ฟิลด์นี้เป็น 0 เมื่อไม่ต้องการให้มีการตรวจสอบค่าของคุณสมบัติ

3. การกำหนดเหตุการณ์ของตัวควบคุม

เหตุการณ์ของตัวควบคุมจะประกอบไปด้วยเหตุการณ์ที่เป็นมาตรฐานของวิชวลเบสิคและเหตุการณ์ที่สร้างขึ้นมาใหม่ สำหรับเหตุการณ์ที่เป็นมาตรฐานนั้นเพียงแค่กำหนดเหตุการณ์ที่ต้องการไว้ในตารางเหตุการณ์ของตัวควบคุมพร้อมกับกำหนดดัชนีของเหตุการณ์ในลำดับเดียวกันกับลำดับที่อยู่ในตารางเหตุการณ์ แล้วให้ฟังก์ชัน VBDefControlProc จัดการให้เกิดเหตุการณ์นั้นขึ้นเองตามการทำงานของผู้ใช้ตัวควบคุม (HPCTL.C บรรทัดที่ 114) แต่เหตุการณ์ที่สร้างขึ้นมาใหม่ จะมีขั้นตอนในการพัฒนาดังนี้

3.1 ออกแบบและกำหนดว่า การทำงานในด้านใดบ้างของตัวควบคุมที่จะทำให้เกิดเหตุการณ์อะไรเกิดขึ้นบ้าง ในตัวควบคุมสำหรับโครงสร้างข้อมูลแบบฮีปนี้จะมีเหตุการณ์เกิดขึ้นเนื่องจากมีการเปลี่ยนแปลงภายในโครงสร้างข้อมูลและการอ่านบัพข้อมูลทั้งหมด 6 เหตุการณ์ตามที่ได้กล่าวไว้ในบทที่ 4

3.2 การกำหนดรายละเอียดของแต่ละเหตุการณ์ที่สร้างขึ้นมาใหม่ รายละเอียดของแต่ละเหตุการณ์จะเก็บอยู่ในโครงสร้าง EVENTINFO โดยข้อมูลที่เก็บอยู่ในโครงสร้างนี้ได้แก่ ชื่อของเหตุการณ์ที่จะปรากฏอยู่ในหน้าต่างสำหรับการเขียนโปรแกรมในวิชวลเบสิค (Code window) จำนวนอาร์กิวเมนต์ที่จะส่งเข้าไปให้กับเหตุการณ์ กำหนดชนิดของอาร์กวิเมนต์ที่จะส่งเข้าไป และสายอักขระที่ใช้ในการอธิบายอาร์กิวเมนต์แต่ละตัวที่จะแสดงในหน้าต่างสำหรับการเขียนโปรแกรม โดยรายละเอียดของแต่ละฟิลด์จะกล่างถึงในภายหลัง ในข้างล่างนี้จะยกตัวอย่างเหตุการณ์ที่สร้างขึ้นมาใหม่มาประกอบการอธิบาย 1 เหตุการณ์ คือ

```
WORD Paramtypes_NodeEvent[] = {ET_I2, ET_HLSTR};

EVENTINFO Event_NodeAdd =
{
  "HpNodeAdd",
  2,
  4,
  Paramtypes_NodeEvent,
   "Key As Integer, Rep As String"
  };
```

เหตุการณ์ที่สร้างขึ้นมานี้ชื่อ HpNodeAdd โดยมีอาร์กิวเมนต์ที่จะส่งเข้าไปให้กับเหตุการณ์นี้ 2 ตัว และอาร์กิวเมนต์ทั้ง 2 นี้ เมื่อคำนวนเป็นจำนวนคำ (WORD) แล้วได้เป็น 4 คำ อาร์กิวเมนต์ที่ส่งเข้าไปตัวแรกเป็นข้อมูลชนิดจำนวนเต็ม อาร์กิวเมนต์ตัวที่สองเป็นสายอักขระชนิด HLSTR (จากที่กำหนดไว้ในแถวลำดับชื่อ Paramtypes_NodeEvent[ ]) และจะเห็นคำอธิบายสำหรับอาร์กิวเมนต์ทั้ง 2 ในหน้าต่างสำหรับการเขียนโปรแกรมว่า Key As Integer, Rep As String

3.3 การกำหนดเหตุการณ์ไว้ในตารางเหตุการณ์ เป็นการนำเหตุการณ์ต่าง ๆ ที่ได้กำหนดไว้ของตัวควบคุมที่ประกอบด้วยเหตุการณ์ที่เป็นมาตรฐานและเหตุการณ์ที่สร้างขึ้นมาใหม่มารวมอยู่ในตารางเหตุการณ์ สำหรับเหตุการณ์ที่สร้างขึ้นมาใหม่ ให้กำหนดเป็นที่อยู่ของโครงสร้าง

EVENTINFO ของเหตุการณ์นั้น จากตัวอย่างเหตุการณ์ในข้อ 3.2 จะกำหนดเป็นส่วนหนึ่งในตารางเหตุการณ์ได้เป็น

&Event_NodeAdd (HPCTL.H บรรทัดที่ 183 - 190)

3.4 การกำหนดค่าคงที่เพื่อเป็นดัชนีของเหตุการณ์ เช่นเดียวกับการกำหนดในตารางคุณสมบัติคือ เหตุการณ์ต่าง ๆ ในตารางเหตุการณ์สามารถจัดลำดับอย่างไรก็ได้ เพราะในหน้าต่างสำหรับการเขียนโปรแกรมวิชวลเบสิค จะแสดงเหตุการณ์ต่าง ๆ เรียงลำดับตามตัวอักษรของชื่อเหตุการณ์ แต่การกำหนดค่าคงที่ เพื่อเป็นดัชนีของแต่ละเหตุการณ์ จะต้องกำหนดให้ค่าดัชนีของเหตุการณ์เป็นค่าที่เป็นลำดับที่ของเหตุการณ์ตามที่อยู่ในตารางเหตุการณ์โดยเริ่มนับจาก 0 จากเหตุการณ์ในข้อ 3.3 จะกำหนดค่าคงที่ของดัชนีได้เป็น

#define IEVENT_Heap_NodeAdd 0 (HPCTL.H บรรทัดที่ 93 - 96)

3.5 โครงสร้าง EVENTINFO ของแต่ละเหตุการณ์ประกอบด้วยฟิลด์ต่าง ๆ ดังนี้

1. npszName เป็นชื่อของเหตุการณ์ที่แสดงอยู่ในหน้าต่างสำหรับการเขียนโปรแกรม

2. cParms กำหนดจำนวนอาร์กิวเมนต์ที่จะส่งเข้าไปในเหตุการณ์ซึ่งยังไม่รวมอาร์กิวเมนต์ Index

3. cwParms จำนวนคำ (WORD) ของรายการอาร์กิวเมนต์ (Argument list) เนื่องจากอาร์กิวเมนต์แต่ละตัวจะต้องมีตัวชี้ขนาด 2 ไบต์ไปยังข้อมูลของอาร์กิวเมนต์ ดังนั้นฟิลด์นี้จึงมีค่าเป็น 2 เท่าของฟิลด์ cParms เสมอ

4. npParmTypes เป็นตัวชี้ที่อยู่ของแถวลำดับแบบไบต์ที่บอกถึงชนิดของอาร์กิวเมนต์แต่ละตัวตามลำดับที่อยู่ในกระบวนงานเหตุการณ์ โดยการบ่งชี้ชนิดของอาร์กิวเมนต์จะใช้ตัวบ่งชี้ที่ขึ้นต้น ET ซึ่งชนิดของตัวบ่งชี้จะแสดงอยู่ในตารางที่ 5.4

5. npszParmProf เป็นสายอักขระที่แสดงถึงอาร์กิวเมนต์แต่ละตัวของเหตุการณ์ในหน้าต่างสำหรับการเขียนโปรแกรม สายอักขระนี้จะใช้ "\0" ในการปิดท้ายสายอักขระ และใช้เครื่องหมายจุลภาค "," ในการคั่นสายอักขระของอาร์กิวเมนต์แต่ละตัว แต่จะไม่ใช้วงเล็บภายในสายอักขระ และไม่ต้องรวมสายอักขระของอาร์กิวเมนต์ Index เพราะวิชวลเบสิคจะจัดการในส่วนนี้ให้เอง

6. fl ถ้าใช้จะกำหนดให้เป็นค่า EF_fNoUnload เพื่อบอกให้วิชวลเบสิคไม่ยอมให้จบการใช้ฟอร์มหรือตัวควบคุมอื่น ๆ ที่อยู่บนฟอร์มจนกระทั่งกระบวนงานของเหตุการณ์ที่เกิดขึ้นจะทำงานเสร็จ ฟิลด์นี้ถ้าไม่ใช้จะกำหนดให้เป็น 0

ชนิดของอาร์กิวเมนต์	ความหมาย
ET_I2	เลขจำนวนเต็ม 16 บิต แบบมีเครื่องหมาย
ET_I4	เลขจำนวนเต็ม 32 บิต แบบมีเครื่องหมาย
ET_R4	เลขจำนวนจริง 4 ไบต์
ET_R8	เลขจำนวนจริง 8 ไบต์
ET_CY	เลขจำนวนเงิน ขนาด 8 ไบต์
ET_HLSTR	สายอักขระ
ET_FS	สายอักขระที่กำหนดความยาวคงที่

ตารางที่ 5.4 ตัวบ่งชี้ชนิดอาร์กิวเมนต์ของเหตุการณ์ (Microsoft Corp., 1993)

4. ฟังก์ชันที่ใช้ในการเริ่มต้นการทำงานและจบการทำงานของตัวควบคุม

4.1 ฟังก์ชันภายในแฟ้ม LIBENTRY.OBJ

เนื่องจากตัวควบคุมเป็น DLL ประเภทหนึ่ง จึงต้องการกระบวนการเริ่มต้นการทำงานที่แตกต่างจากโปรแกรมประยุกต์ทั่วไป ตัวควบคุมจะเริ่มต้นการทำงานครั้งแรกสุดโดยการเรียกใช้ฟังก์ชันสำหรับเริ่มต้นการทำงาน (Library Initialization Function) ที่อยู่ในแฟ้ม LIBENTRY.OBJ ซึ่งแฟ้มนี้วิชวลเบสิคเตรียมมาให้แล้วอยู่ในสารบบ CDK โดยถ้าใช้ตัวแปลภาษาซีของไมโครซอฟต์ (Microsoft C Compiler) ตั้งแต่รุ่น 7.0 ขึ้นไปแล้ว ฟังก์ชันนี้จะถูกเชื่อมโยงเข้าไปในแฟ้มของตัวควบคุมให้เองโดยอัตโนมัติ แต่ถ้าใช้ตัวแปลภาษาซีของไมโครซอฟต์รุ่น 6.0 แล้ว เราจะต้องสั่งให้เชื่อมโยงฟังก์ชันที่อยู่ในแฟ้ม LIBENTRY.OBJ เข้าไปในแฟ้มตัวควบคุมเอง

4.2 ฟังก์ชันที่อยู่ในแฟ้มโปรแกรมของตัวควบคุมเอง

ตัวควบคุมยังต้องการฟังก์ชันที่ใช้ในการเริ่มต้นการทำงานอีก 2 ฟังก์ชัน และจบการทำงาน 2 ฟังก์ชัน คือ

4.2.1 ฟังก์ชัน LibMain จะถูกเรียกใช้โดยฟังก์ชันที่อยู่ในแฟ้ม LIBENTRY.OBJ อีกทีหนึ่ง โดยฟังก์ชันนี้จะเริ่มต้นการทำงานเมื่อแฟ้มของตัวควบคุมถูกอ่านเข้ามาในหน่วยความจำครั้งแรก

```
//---------------------------------------------------------------------
// Initialize library. This routine is called when the first client loads the DLL.
//---------------------------------------------------------------------
```

```
int FAR PASCAL LibMain
(
  HANDLE hModule,
  WORD   wDataSeg,
  WORD   cbHeapSize,
  LPSTR  lpszCmdLine
)
{
hmodDLL = hModule;
return 1;
}
```

ฟังก์ชัน LibMain จะกำหนดค่าแฮนเดิลของแฟ้มตัวควบคุมให้กับตัวแปร hmodDLL ซึ่งเป็นตัวแปรประเภท Global โดยค่าในตัวแปร hmodDLL จะถูกใช้โดยฟังก์ชัน VBINITCC อีกทีหนึ่ง

4.2.2 ฟังก์ชัน VBINITCC ใช้ฟังก์ชันนี้ในการขึ้นทะเบียนชั้นของตัวควบคุม และถ้ามีงานที่จะต้องเริ่มต้นทำเป็นครั้งแรกของชั้นของตัวควบคุมแล้ว ก็จะให้มาทำในฟังก์ชันนี้เช่นกัน

```
//---------------------------------------------------------------------
// Register custom control. This routine is called by VB
// when the custom control DLL is loaded for use.
BOOL FAR PASCAL _export VBINITCC
(
  USHORT usVersion,
  BOOL   fRuntime
)
{
::
// Register control(s)
return VBRegisterModel(hmodDLL, &modelHeap);
}
```

ข้อแตกต่างระหว่างฟังก์ชัน LibMain กับฟังก์ชัน VBINITCC ก็คือ ฟังก์ชัน LibMain จะถูกเรียกใช้เพียงครั้งเดียวที่มีการอ่านแฟ้มของตัวควบคุมเข้ามาในหน่วยความจำ แต่ฟังก์ชัน VBINITCC จะถูกเรียกในแต่ละครั้งที่โปรแกรมประยุกต์ต้องการใช้แฟ้มตัวควบคุมเป็นครั้งแรกของแต่ละโปรแกรมประยุกต์ ตัวอย่างเช่น ถ้ามีโปรแกรมประยุกต์ 2 โปรแกรมที่มีการใช้ตัวควบคุมพร้อม ๆ กัน แล้ว ฟังก์ชัน LibMain จะถูกเรียกใช้งานเพียงครั้งเดียว แต่ฟังก์ชัน VBINITCC จะถูกเรียกใช้งาน 2 ครั้ง

ในสารบบ CIRC3 มีตัวอย่างของการสร้างตัวควบคุมที่นอกจากจะใช้ฟังก์ชัน VBINITCC ในการขึ้นทะเบียนต้นแบบของตัวควบคุมแล้ว ยังใช้ในการขึ้นทะเบียนชั้นตัวควบคุมอื่นที่จะใช้ในช่วงระหว่างการพัฒนาโปรแกรม

4.2.3 ฟังก์ชัน VBTERMCC ฟังก์ชันนี้ไม่ต้องมีในแฟ้มของตัวควบคุมก็ได้ เราใช้ฟังก์ชันนี้เพื่อการยกเลิกการจองใช้ทรัพยากรต่าง ๆ ของตัวควบคุม ฟังก์ชันนี้จะถูกเรียกใช้ทุกครั้งที่โปรแกรมประยุกต์จะยกเลิกการใช้งานตัวควบคุม

ในสารบบ CIRC3 แสดงตัวอย่างการใช้งานฟังก์ชัน VBTERMCC ในการยกเลิกการจองพื้นที่หน่วยความจำที่ใช้ในการเก็บชั้นของตัวควบคุมอื่นซึ่งถูกขึ้นทะเบียนไว้โดยฟังก์ชัน VBINITCC

4.2.4 ฟังก์ชัน WEP เป็นฟังก์ชันที่ถูกเรียกใช้เมื่อแฟ้มของตัวควบคุม ถูกนำออกจากหน่วยความจำ ฟังก์ชันนี้เพียงแต่คืนค่า 1 กลับออกมา เพื่อแสดงว่าแฟ้มของตัวควบคุมจบการใช้งานโดยบริบูรณ์

เป็นที่สังเกตว่า ฟังก์ชัน WEP นี้ จะถูกแปลและเชื่อมโยงเข้าไปในแฟ้มของตัวควบคุมเมื่อเราใช้ Microsoft C Compiler รุ่น 6.0 เท่านั้น แต่ถ้าเราใช้รุ่น 7.0 ขึ้นไป ตัวเชื่อมโยงจะเพิ่มฟังก์ชันนี้เข้าไปในแฟ้มของตัวควบคุมให้เอง โดยไม่ต้องไปกำหนดฟังก์ชันนี้ไว้ในแฟ้มโปรแกรมของตัวควบคุมเลย

5. การตอบสนองต่อข้อความที่สำคัญของกระบวนงานของตัวควบคุม

การทำงานของกระบวนงานของตัวควบคุม จะคล้ายกับโปรแกรมประยุกต์ทั่วไป คือเป็นการตอบสนองต่อข้อความต่าง ๆ ที่ส่งเข้ามายังตัวควบคุม กระบวนงาน HeapCtlProc จะทำงานตอบสนองต่อข้อความที่สำคัญซึ่งได้แก่

5.1 WM_NCCREATE เราใช้ข้อความนี้ในการกำหนดค่าโดยปริยายให้คุณสมบัติต่าง ๆ ที่ต้องการได้ ข้อความนี้จะถูกส่งเข้ามาหลังจากที่โครงสร้างของวินโดว์ได้ถูกสร้างขึ้นมา

การกำหนดค่าโดยปริยายให้กับคุณสมบัติของตัวควบคุมหลังจากที่ได้รับข้อความนี้ จะไม่ทำให้ค่าคุณสมบัติที่เคยกำหนดไว้ก่อนหน้านี้เปลี่ยนแปลงไป เพราะว่าข้อความนี้ถูกส่งเข้ามาก่อนแล้ว ค่าคุณสมบัติของตัวควบคุมที่บันทึกอยู่ในจานแม่เหล็กจึงถูกอ่านเข้า

```
case WM_NCCREATE:
    {
    List_type FAR *hp = &LpheapDEREF(hctl)->hpdta ;
    hp->count = 0 ;
    LpheapDEREF(hctl)->dsptree = TRUE ;
    // *** lpheap may now be invalid due to call to VB API ***
    break;
    }
```

ส่วนของโปรแกรมนี้เป็นการกำหนดค่าโดยปริยายให้กับคุณสมบัติ DisplayTree ของตัวควบคุมให้เป็น TRUE เพื่อให้การวาดแผนภาพของโครงสร้างข้อมูลเป็นแบบต้นไม้ เนื่องจากรูปแบบการวาดแผนภาพของโครงสร้างข้อมูลจะวาดตามค่าที่กำหนดอยู่ในคุณสมบัตินี้

5.2 WM_LBUTTONDBLCLK ข้อความนี้จะถูกส่งเข้ามาเมื่อมีการกดปุ่มข้างซ้ายของเมาส์ติดกัน 2 ครั้ง โดยตัวควบคุมจะตอบสนองต่อข้อความนี้โดยการเรียกใช้ฟังก์ชัน InputNode เพื่อแสดงกรอบรับข้อความขึ้นมารับข้อมูลทั้ง 2 ส่วนของบัพข้อมูลใหม่

```
case WM_LBUTTONDBLCLK:
    {
    hctldlg = hctl ;
    DialogBox (hmodDLL, "INPUT", hwnd, (FARPROC) InputNode) ;
    return 0 ;
    }
```

5.3 WM_RBUTTONDBLCLK ข้อความนี้จะถูกส่งเข้ามาเมื่อมีการกดปุ่มข้างขวาของเมาส์ติดกัน 2 ครั้งบนบัพข้อมูลที่แสดงอยู่ภายในกรอบของตัวควบคุม จะมีการเรียกใช้ฟังก์ชัน InNodeTre หรือฟังก์ชัน InNodeLin ขึ้นอยู่กับรูปแบบการแสดงผลในขณะนั้น เพื่อตรวจสอบว่าตำแหน่งที่กดเมาส์ในพื้นที่ของตัวควบคุมนั้นมีบัพข้อมูลแสดงอยู่จริงหรือไม่ ถ้ามีบัพข้อมูลแสดงอยู่จริงแล้ว ตัวควบคุมจะเรียกใช้ฟังก์ชัน ChgNode ในการแสดงกรอบรับข้อความที่มีบัพข้อมูลนั้นอยู่เพื่อรอการแก้ไขเปลี่ยนแปลง

```
case WM_RBUTTONDBLCLK:
  {
  BOOL tree, innode ;
  VBGetControlProperty(hctl, IPROP_HEAP_DspTree, &tree);
  if (tree)
    innode = InNodeTre(hctl, hwnd, (SHORT)lp, (SHORT)HIWORD(lp), inode) ;
  else
    innode = InNodeLin(hctl, hwnd, (SHORT)lp, (SHORT)HIWORD(lp), inode) ;
  if (innode)
    {
    hctldlg = hctl ;
    DialogBoxParam(hmodDLL, "CHANGE", hwnd, (FARPROC)ChgNode, *inode) ;
    return 0 ;
    }
    break ;
    }
```

5.4 WM_PAINT ข้อความนี้จะถูกส่งเข้ามาเมื่อมีความต้องการให้ตัวควบคุมแสดงผลของโครงสร้างข้อมูลที่เหมาะสมออกมา หรือกล่าวอีกนัยหนึ่งก็คือ การตอบสนองต่อข้อความนี้ก็เพื่อการแสดงผลของตัวควบคุมนั่นเอง ก่อนที่จะแสดงผลของบัพข้อมูลออกมา จะมีการตรวจสอบก่อนว่า ค่าคุณสมบัติ Visible ของตัวควบคุมกำหนดให้แสดงตัวควบคุมออกมาให้เห็นหรือไม่ ถ้าแสดงตัวควบคุมให้เห็นได้ ก็จะทำการวาดแผนภาพของโครงสร้างข้อมูลออกมา และค่าคุณสมบัติ DisplayTree จะเป็นตัวกำหนดรูปแบบการแสดงผลว่าจะแสดงเป็นแบบต้นไม้หรือแบบแถวลำดับ ก็จะมีการเรียกใช้ฟังก์ชันในการวาดแผนภาพที่สอดคล้องกัน

```
BOOL vis, tree ;
    VBGetControlProperty(hctl, IPROP_HEAP_VISIBLE, &vis);
    VBGetControlProperty(hctl, IPROP_HEAP_DspTree, &tree);
    if (vis) {
    PAINTSTRUCT ps;
  BeginPaint(hwnd, &ps);
  if (tree)
```

```
        PaintTree(hctl, hwnd, ps.hdc);
      else
        PaintLine(hctl, hwnd, ps.hdc);
      EndPaint(hwnd, &ps);
      }
      break;
```

นอกจากการแสดงผลเนื่องจากการตอบสนองต่อข้อความนี้แล้ว ยังมีการแสดงผลเนื่องจากการทำงานในด้านอื่น ๆ เช่นการแสดงผลเนื่องจากการจัดการให้โครงสร้างข้อมูลเป็นฮีปซึ่งจะมีการแสดงการเปลี่ยนแปลงภายในโครงสร้างข้อมูลตามการทำงานทีละขั้นตอน (Animation)

5.5 VBM_SETPROPERTY ข้อความนี้จะถูกส่งเข้ามา เมื่อมีการกำหนดค่าคุณสมบัติของตัวควบคุมเกิดขึ้น ตัวควบคุมจะตรวจดูว่าเป็นการกำหนดคุณสมบัติในข้อใด ก็จะมีการทำงานตามคุณสมบัติในข้อนั้นที่ได้กำหนดไว้ ในส่วนของโปรแกรมข้างล่างนี้ จะเป็นการกำหนดคุณสมบัติ BuildHeap การทำงานในส่วนนี้ก็จะเป็นการนำค่าของคุณสมบัติที่กำหนดเข้ามานั้น ไปเก็บไว้ในโครงสร้างของผู้เขียนโปรแกรม และทำการจัดให้โครงสร้างข้อมูลให้เป็นฮีป

```
    case IPROP_HEAP_BuildHeap:
      {
      BOOL hpified = LpheapDEREF(hctl)->bldhp ;
      if (hpified != (BOOL)lp) {
        LpheapDEREF(hctl)->bldhp = (BOOL)lp;
        if ((BOOL)lp) {
        lptmp = LpheapDEREF(hctl) ;
        hp = &lptmp->hpdta ;
        BuildHeap(hwnd, hctl, FALSE, 'B', 0) ;
       }
     }
      return 0;
      }
```

5.6 VBM_METHOD ข้อความนี้จะเกิดขึ้น เมื่อมีการสั่งให้ทำงานด้วยวิธีของตัวควบคุม ซึ่งภายในตัวควบคุมจะต้องมีการจัดการให้ทำงานตอบสนองวิธีอย่างถูกต้อง โดยตัวควบคุมต้องตรวจดูก่อนว่า เป็นวิธีที่ตัวควบคุมจะทำงานให้ได้หรือไม่ สำหรับวิธีที่ตัวควบคุมทำงานให้ไม่

ได้ ก็จะส่งไปให้ฟังก์ชัน VBDefControlProc ทำงานต่อ โดยตัวควบคุมนี้จะทำงานให้ได้เฉพาะวิธี Clear เท่านั้น ในการพิจารณาวิธีและการทำงานให้กับวิธีนั้น จะถูกจัดการโดยฟังก์ชัน Methods ของตัวควบคุมเอง

```
case VBM_METHOD:
  {
  // Respond to the Clear method for the Control.
  LONG lRetVal = Methods(hctl, hwnd, wp, lp);
  if (lRetVal != USE_DEFAULT_PROC)
    return lRetVal;
  break;
  }
```

5.7 WM_NCDESTROY ข้อความนี้จะถูกส่งเข้ามาเมื่อจะจบการใช้งานตัวควบคุม ตัวควบคุมควรจะตอบสนองต่อข้อความนี้ โดยการยกเลิกการจองใช้ทรัพยากรต่าง ๆ โดยในตัวควบคุมนี้มีการจองหน่วยความจำเพื่อเก็บสายอักขระชนิด HSZ อยู่ในโครงสร้างของผู้เขียนโปรแกรม การทำงานของโปรแกรมที่ดีจึงควรยกเลิกการจองพื้นที่ในหน่วยความจำนั้นด้วยเมื่อจะจบการใช้งาน

```
case WM_NCDESTROY:
  LPHEAP lptmp = LpheapDEREF(hctl) ;
  if (lptmp->nrep)
    VBDestroyHsz(lptmp->nrep) ;
  break ;
```

6. <u>การจัดการกับคุณสมบัติที่สร้างขึ้นใหม่</u>

จากคุณสมบัติที่เราได้กำหนดไว้ในข้อ 2. จะต้องมีการเขียนโปรแกรมเพิ่มเติมเพื่อจัดการให้ตัวควบคุมทำงานตามค่าของคุณสมบัติที่ได้กำหนดไว้ โดยเริ่มจากการตอบสนองต่อข้อความ VBM_SETPROPERTY ซึ่งจะทำให้เราทราบได้ว่ามีการกำหนดค่าของคุณสมบัติเกิดขึ้นแล้วเราก็จะนำค่าของคุณสมบัติที่กำหนดเข้ามา ไปเก็บไว้ในโครงสร้างของผู้เขียนโปรแกรม แต่การทำงานที่กล่าวมานี้อาจไม่ต้องเขียนโปรแกรมเพิ่มเติมเลยก็ได้ ถ้าคุณสมบัตินั้นได้กำหนดไว้ให้วิชวลเบสิคนำค่าของคุณสมบัติเข้าไปเก็บไว้ในโครงสร้างของผู้เขียนโปรแกรมโดยตรง งานใน

ส่วนที่เหลือก็จะเป็นการเขียนโปรแกรมเพื่อให้ตัวควบคุมทำงานตามค่าของคุณสมบัติที่กำหนดเข้ามา ในลำดับต่อไปจะกล่าวถึงการเขียนโปรแกรมเพื่อจัดการคุณสมบัติ DisplayTree ที่ได้กล่าวถึงในข้อ 2. ซึ่งเป็นคุณสมบัติที่ใช้ในการกำหนดรูปแบบการแสดงผลโครงสร้างข้อมูลว่าจะแสดงแบบต้นไม้หรือแถวลำดับ

```
case VBM_SETPROPERTY:
  ::
  case IPROP_HEAP_DspTree:
  {
  BOOL tree = LpheapDEREF(hctl)->dsptree ;
  if (tree != (BOOL)lp) {
    LpheapDEREF(hctl)->dsptree = (BOOL)lp;
    InvalidateRect(hwnd, NULL, TRUE);
    LPRECT lprect = new RECT ;
    GetClientRect(hwnd, lprect);
    SendMessage(hwnd, WM_SIZE, 0, MAKELONG((lprect->right - lprect->left), 0));
    delete lprect ;
    }
return 0;
}
```

เป็นการนำค่าของคุณสมบัติที่กำหนดเข้ามา ไปเก็บไว้ในฟิลด์ dsptree ที่อยู่ในโครงสร้างของผู้เขียนโปรแกรม จากนั้นจะเป็นการเตรียมพื้นที่สำหรับการวาดแผนภาพของโครงสร้างข้อมูลโดยการเรียกฟังก์ชัน InvalidateRect ซึ่งจะทำให้เกิดข้อความ WM_PAINT เข้ามายังตัวควบคุมในการส่งข้อความครั้งต่อไป จึงทำให้มีการวาดแผนภาพของโครงสร้างข้อมูลใหม่อีกครั้งหนึ่ง แล้วจะเป็นการกำหนดอัตราส่วนของ Scroll bar ขึ้นใหม่ให้เหมาะสมกับรูปแบบของการแสดงผล

```
case WM_PAINT:
  BOOL vis, tree ;
  VBGetControlProperty(hctl, IPROP_HEAP_VISIBLE, &vis);
  VBGetControlProperty(hctl, IPROP_HEAP_DspTree, &tree);
  if (vis) {
```

```
        PAINTSTRUCT ps;
      BeginPaint(hwnd, &ps);
      if (tree)
        PaintTree(hctl, hwnd, ps.hdc);
      else
        PaintLine(hctl, hwnd, ps.hdc);
      EndPaint(hwnd, &ps);
        }
    break;
```

การวาดแผนภาพโครงสร้างข้อมูลในช่วงของการตอบสนองต่อข้อความ WM_PAINT มีส่วนหลักของการทำงานอยู่ที่การพิจารณาว่า จะใช้รูปแบบใดในการแสดงโครงสร้างข้อมูลออกมา โดยพิจารณาจากคุณสมบัติ DisplayTree ของตัวควบคุม โดยถ้าใช้รูปแบบต้นไม้ (มีค่าเป็นจริง) จะวาดโดยฟังก์ชัน PaintTree แต่ถ้าเป็นรูปแบบแถวลำดับ (มีค่าเป็นเท็จ) จะวาดโดยใช้ฟังก์ชัน PaintLine

7. <u>การจัดการกับเหตุการณ์ที่สร้างขึ้นใหม่</u>

จากเหตุการณ์ที่เราได้กำหนดไว้ในข้อ 3. จะต้องมีการเขียนโปรแกรมเพิ่มเติม เพื่อทำให้เหตุการณ์ที่กำหนดไว้เกิดขึ้นเมื่อพบว่ามีการทำงานของตัวควบคุมตามที่ได้กำหนดไว้เกิดขึ้น และถ้าเหตุการณ์นั้นเป็นเหตุการณ์ที่มีอาร์กิวเมนต์ด้วยแล้ว ก็จะต้องเตรียมข้อมูลเพื่อให้เป็นอาร์กิวเมนต์ ของเหตุการณ์นั้นด้วย ในข้างล่างนี้จะกล่างถึงส่วนของโปรแกรมเพื่อจัดการเหตุการณ์ HpNodeAdd ซึ่งเหตุการณ์นี้จะเกิดขึ้นก็ต่อเมื่อมีการเพิ่มบัพข้อมูลใหม่เข้าไปในโครงสร้างข้อมูล และเหตุการณ์นี้ต้องการอาร์กิวเมนต์ 2 ตัว โดยอาร์กิวเมนต์ตัวแรกจะเป็นส่วนที่เป็นคีย์ของบัพข้อมูล และอาร์กิวเมนต์ตัวที่สองจะเป็นสายอักขระที่เป็นตัวแทนของบัพข้อมูล ส่วนของโปรแกรมที่แสดงในข้างล่างนี้ เป็นการเพิ่มบัพข้อมูลใหม่เข้าไปโดยใช้การกำหนดค่าของคุณสมบัติเข้าไปให้ก็จะมีการเรียกฟังก์ชัน FireNodeEvt พร้อมกับส่งข้อมูลที่จะเป็นอาร์กิวเมนต์เข้าไปเพื่อจัดการทำงานให้เหตุการณ์ HpNodeAdd เกิดขึ้น

```
case IPROP_HEAP_NodeVal:
  {
  lptmp = LpheapDEREF(hctl) ;
```

```
hp = &lptmp->hpdta ;
::
hp->entry[hp->count] = (short)lp ;
(hp->count)++ ;
::
LPSTR ntmp = new __far char[STRLEN] ;
lptmp = LpheapDEREF(hctl) ;
lstrcpyn(ntmp, VBDerefHsz(lptmp->nrep), STRLEN) ;
FireNodeEvt(hctl, 0, (int) lp, ntmp, 'A');
delete  [] (char __far *) ntmp ;
return 0 ;
}
```

ฟังก์ชัน FireNodeEvt จะทำการจัดเตรียมโครงสร้างสำหรับอาร์กิวเมนต์ของเหตุการณ์ โดยฟิลด์ที่อยู่ในโครงสร้างนี้ จะต้องเป็นตัวชี้ที่อยู่ของข้อมูลที่จะเป็นอาร์กิวเมนต์ขนาด 32 บิต (Far Pointer) ยกเว้นถ้าเป็นอาร์กิวเมนต์แบบสายอักขระแล้ว จะไม่ใช้ตัวชี้ที่อยู่ แต่จะเป็นแฮนเดิลสำหรับสายอักขระชนิด HLSTR แต่จะสังเกตได้ว่า ลำดับของอาร์กิวเมนต์ที่อยู่ในโครงสร้างนี้จะกลับกันกับที่แสดงอยู่ในวิชวลเบสิค หลังจากที่เตรียมโครงสร้างของอาร์กิวเมนต์เสร็จแล้ว ก็จะเป็นการเรียกใช้ฟังก์ชัน VBFireEvent ซึ่งจะทำให้เหตุการณ์ HpNodeAdd เกิดขึ้น

ฟังก์ชัน VBFireEvent นี้ จะทำให้กระบวนงานของเหตุการณ์นั้นที่ได้สร้างเตรียมไว้อยู่ในวิชวลเบสิคมีการทำงานพร้อมกับส่งอาร์กิวเมนต์ของเหตุการณ์เข้าไปให้โดยผ่านทางโครงสร้างของอาร์กิวเมนต์ ฟังก์ชัน VBFireEvent จะรอจนกระทั่งกระบวนงานของเหตุการณ์ในวิชวลเบสิคทำงานเสร็จแล้ว จึงจะจบการทำงานของฟังก์ชันกลับมายังตัวควบคุม อนึ่ง ค่าของอาร์กิวเมนต์ที่ถูกส่งเข้าไป อาจจะถูกเปลี่ยนแปลงได้โดยกระบวนงานของเหตุการณ์ในวิชวลเบสิค

```
VOID NEAR FireNodeEvt
(   HCTL hctl,
    int n1,
    int k1,
    LPSTR Text1,
    char opr
)
```

```
{
    NodeOprPARMS params;
    int     xkey, xi;
    USHORT err;
    xkey = k1 ;
    params.nkey = &xkey;
    params.nString = VBCreateHlstr(Text1, lstrlen(Text1));
    err = VBFireEvent(hctl, IEVENT_Heap_NodeAdd, &params);
    VBDestroyHlstr(params.nString);
}
```

8. การจัดการกับสายอักขระ

สายอักขระที่ใช้ในตัวควบคุม มีลักษณะพิเศษที่ต้องพิจารณาคือ

- สายอักขระจะไม่เหมือนกับตัวเลข สายอักขระมีขนาดความยาวที่ไม่คงที่ เพื่อให้การทำงานมีประสิทธิภาพ สายอักขระจำเป็นต้องมีการจองพื้นที่หน่วยความจำแบบพลวัต (Dynamic) และถูกจัดการในหน่วยความจำโดยวิชวลเบสิคเอง และที่อยู่ของสายอักขระในหน่วยความจำนั้น สามารถเคลื่อนย้ายเปลี่ยนแปลงไปได้

- ในการพัฒนาตัวควบคุม จะต้องมีการใช้สายอักขระของภาษาซี ซึ่งเป็นสายอักขระแบบที่มี "\0" ปิดท้าย และมีการใช้สายอักขระของวิชวลเบสิค ซึ่งมีตัวอธิบายลักษณะสายอักขระ (String descriptor) ในการบอกความยาวของสายอักขระ โดยสายอักขระของวิชวลเบสิคสามารถใช้เป็นอาร์กิวเมนต์ส่งเข้าไปในกระบวนงานของเหตุการณ์ได้

สายอักขระของภาษาซีสามารถใช้ในการทำงานบางอย่างได้ แต่ถ้าสายอักขระที่ใช้ต้องการความยาวค่อนข้างมากแล้ว ควรใช้สายอักขระของวิชวลเบสิคจะมีประสิทธิภาพที่ดีกว่า เพราะว่าไม่จำเป็นต้องกำหนดความยาวของสายอักขระที่จะใช้ซึ่งจะทำให้เกิดการจองเนื้อที่ไว้ล่วงหน้า (Microsoft Corp., 1993)

ในการเขียนโปรแกรมสำหรับวินโดว์ ข้อมูลชนิดต่าง ๆ สามารถเปลี่ยนแปลงที่อยู่ในหน่วยความจำได้ แต่ข้อมูลเหล่านั้นจะมีแฮนเดิลในการอ้างอิงถึงได้ แฮนเดิลจะแตกต่างกับตัวชี้ที่อยู่ โดยถ้าที่อยู่ของข้อมูลเปลี่ยนแปลงไป แฮนเดิลก็ยังคงใช้ในการอ้างอิงข้อมูลได้ แต่ตัวชี้ที่อยู่จะใช้ไม่ได้

สายอักขระที่ใช้ได้กับวิชวลเบสิค จะแบ่งแยกตามลักษณะของแฮนเดิลได้ 2 ชนิด ดังนี้

1. HSZ เป็นแฮนเดิลของสายอักขระที่มี "\0" ปิดท้าย สายอักขระชนิดนี้จะคล้ายกับสายอักขระของภาษาซี แต่ว่าจะต้องอ้างอิงแฮนเดิล HSZ ก่อน เพื่อให้ได้ที่อยู่ของข้อมูลสายอักขระ

2. HLSTR เป็นแฮนเดิลของสายอักขระในภาษาเบสิค สามารถใช้ได้โดยตรงกับคำสั่งในวิชวลเบสิค ความยาวของสายอักขระชนิดนี้จะสามารถเปลี่ยนแปลงได้ตามต้องการ ที่อยู่ของข้อมูลก็สามารถเปลี่ยนแปลงได้ด้วย และสามารถมี "\0" อยู่ภายในสายอักขระได้ ตัวอธิบายลักษณะของสายอักขระ จะใช้ในการบอกความยาวของสายอักขระ นอกจากนั้นสายอักขระชนิดนี้ยังเป็นสายอักขระชนิดเดียวที่จะใช้เป็นอาร์กิวเมนต์ที่จะส่งเข้าไปในกระบวนงานของเหตุการณ์ได้

ฟังก์ชันที่จะใช้ในการจัดการสายอักขระชนิด HSZ จะเป็นฟังก์ชัน API ของวิชวลเบสิค ซึ่งได้แก่

1. VBCreateHsz ฟังก์ชันนี้ จะทำการจองเนื้อที่ให้กับสายอักขระในเซกเมนต์ที่กำหนดและคืนค่าแฮนเดิลของสายอักขระนั้นออกมาให้

2. VBDerefHsz ฟังก์ชันนี้จะคืนค่าที่อยู่ของสายอักขระชนิด HSZ ออกมาให้

3. VBDestroyHsz ยกเลิกการจองเนื้อที่ของสายอักขระ และยกเลิกการใช้แฮนเดิลของสายอักขระนั้น

4. VBLockHsz ใช้ในการป้องกันไม่ให้ที่อยู่ของสายอักขระเคลื่อนย้ายไปที่อื่น และคืนค่าที่อยู่ของสายอักขระชนิดที่มี "\0" ปิดท้ายออกมาให้

5. VBUnlockHsz ยกเลิกการป้องกันการเคลื่อนย้ายของสายอักขระจากการทำงานของฟังก์ชัน VBLockHsz ทำให้สายอักขระสามารถเคลื่อนย้ายเปลี่ยนแปลงที่อยู่ได้เหมือนเดิม

ขนาดของสายอักขระชนิด HSZ จะถูกกำหนดโดยเครื่องหมาย "\0" ที่พบเป็นตัวแรกในสายอักขระและไม่มีวิธีที่จะเปลี่ยนแปลงขนาดความยาวนี้ได้ ดังนั้น เมื่อมีการกำหนดค่าใหม่ให้กับคุณสมบัติชนิดที่เป็น HSZ แล้ว จะต้องยกเลิกการใช้สายอักขระชนิด HSZ ตัวเดิมก่อน แล้วจึงสร้างสายอักขระ HSZ ตัวใหม่ขึ้นมา และกำหนดค่าแฮนเดิล HSZ ตัวใหม่ให้เป็นค่าของคุณสมบัติได้

ฟังก์ชันที่จะใช้ในการจัดการสายอักขระชนิด HLSTR จะเป็นฟังก์ชัน API ของวิชวลเบสิค ซึ่งได้แก่

1. VBCreateHlstr ใช้ในการจองเนื้อที่ให้กับสายอักขระของภาษาเบสิค ตามที่อยู่ของสายอักขระและความยาวที่ได้กำหนดเข้าไปให้ และจะคืนค่าแฮนเดิลของสายอักขระ HLSTR ออกมาให้

2. VBDerefHlstr ฟังก์ชันจะคืนค่าที่อยู่ของข้อมูลสายอักขระชนิด HLSTR ออกมาให้ ซึ่งฟังก์ชันนี้ควรใช้งานคู่กับฟังก์ชัน VBGetHlstrLen เพื่อจะได้ทราบความยาวของสายอักขระ

3. VBDestroyHlstr ยกเลิกการจองใช้เนื้อที่ของสายอักขระและยกเลิกการใช้แฮนเดิล HLSTR ของสายอักขระนั้น

4. VBGetHlstrLen คืนค่าความยาวของสายอักขระชนิด HLSTR ออกมาให้

8.1 การสร้างคุณสมบัติใหม่ ชนิดที่เป็นสายอักขระ HSZ

ในส่วนนี้จะเป็นการอธิบายจากคุณสมบัติที่ชื่อ Node_rep ของตัวควบคุม คุณสมบัตินี้จะใช้ในการรับค่าของส่วนที่เป็นตัวแทนของบัพข้อมูลใหม่เข้ามา โดยเริ่มจากการกำหนดฟิลด์ เพื่อใช้เก็บแฮนเดิล HSZ ของสายอักขระไว้ในโครงสร้างของผู้เขียนโปรแกรม

```
typedef struct tagHEAP
  {
  BOOL anim ;
  ::
  HSZ nrep ;
  ::
  } HEAP;
```

ฟิลด์ nrep ในโครงสร้างของผู้เขียนโปรแกรมนี้ ใช้เก็บแฮนเดิลของสายอักขระ HSZ เท่านั้น โดยข้อมูลของสายอักขระจริง ๆ จะเก็บอยู่ใน HEAP SEGMENT ซึ่งจัดการโดยวิชวลเบสิค

คุณสมบัติ Node_rep จะมีการกำหนดเป็นโครงสร้าง PROPINFO ซึ่งมีการกำหนดรายละเอียดลักษณะการใช้งานไว้ดังนี้

```
PROPINFO Property_NodeRep =
  {
  "Node_rep",
  DT_HSZ | PF_fGetData | PF_fSetData | PF_fSetMsg | PF_fNoShow,
  OFFSETIN(HEAP, nrep),
  0, 0, NULL, 0
  };
```

ที่อยู่ของโครงสร้างนี้จะกำหนดไว้ในตารางคุณสมบัติ ดังนี้

```
&Property_NodeRep,
```

และมีการกำหนดดัชนีของคุณสมบัติได้เป็น

```
#define IPROP_HEAP_NodeRep          12
```

เนื่องจากเราใช้ตัวบ่งชี้ PF_fGetData และ PF_fSetData ทำให้วิชวลเบสิคนำค่าแฮนเดิล HSZ ของสายอักขระเข้ามาเก็บไว้ในโครงสร้างของผู้เขียนโปรแกรมโดยตรง โดยที่เราไม่ต้องเขียนโปรแกรมเพื่อจัดการงานในส่วนที่เกี่ยวกับการยกเลิกการใช้แฮนเดิลของสายอักขระ HSZ ตัวเก่า และสร้างแฮนเดิลตัวใหม่เข้าไปเก็บในโครงสร้างของผู้เขียนโปรแกรม แต่ด้วยตัวบ่งชี้ PF_fSetMsg ทำให้กระบวนงานของตัวควบคุมได้รับข้อความ VBM_SETPROPERTY จากวิชวลเบสิคเมื่อมีการกำหนดค่าของคุณสมบัตินี้เข้ามา เราจึงนำค่าสายอักขระที่กำหนดมาไปใช้ประโยชน์ได้ ในที่นี้ก็จะเป็นการคัดลอกสายอักขระของแฮนเดิล HSZ ไปเก็บไว้เป็นบัพข้อมูลใหม่ในโครงสร้างข้อมูลฮีป

```
case VBM_SETPROPERTY:
 ::
 case IPROP_HEAP_NodeRep:
  {
  lptmp = LpheapDEREF(hctl) ;
  hp = &lptmp->hpdta ;
 ::
  LPSTR ntmp = new __far char[STRLEN] ;
  lstrcpyn(ntmp, VBDerefHsz(lptmp->nrep), STRLEN) ;
  hp->ent1[hp->count] = ntmp ;
  return 0 ;
  }
```

และเมื่อจบการใช้งานตัวควบคุม ก็ควรที่จะยกเลิกการจองใช้เนื้อที่ของสายอักขระ HSZ ในขณะที่ได้รับข้อความ WM_NCDESTROY ตามที่ได้กล่าวไว้ในข้อ 5.7

8.2 การส่งอาร์กิวเมนต์ชนิดที่เป็นสายอักขระเข้าไปให้กระบวนงานของเหตุการณ์

จากเหตุการณ์ที่สร้างขึ้นใหม่ที่ชื่อ HpNodeAdd ในข้อ 7. ซึ่งมีการส่งอาร์กิวเมนต์ที่เป็นสายอักขระเข้าไปให้กระบวนงานของเหตุการณ์ อาร์กิวเมนต์ที่เป็นสายอักขระจะต้องเป็นชนิด HLSTR เท่านั้น โดยเริ่มจากการกำหนดสายอักขระชนิด HLSTR ไว้ในโครงสร้างของ

อาร์กิวเมนต์ เมื่อมีการเพิ่มบัพข้อมูลใหม่เข้ามา จะทำให้เหตุการณ์ HpNodeAdd เกิดขึ้น บัพข้อมูลใหม่จะถูกส่งเข้าไปเป็นอาร์กิวเมนต์ให้กับกระบวนงานของเหตุการณ์ แต่ก่อนหน้านั้น จะต้องเตรียมข้อมูลที่เป็นสายอักขระของภาษาซีให้เป็นข้อมูลสายอักขระชนิด HLSTR ก่อน และเมื่อจบการทำงานของกระบวนงานของเหตุการณ์แล้ว ก็จะยกเลิกการจองเนื้อที่ของแฮนเดิลสายอักขระ HLSTR ที่อยู่ในโครงสร้างของอาร์กิวเมนต์ด้วยฟังก์ชัน VBDestroyHlstr

```
typedef struct tagNodeOprPARMS
   {
   HLSTR    nString;
   int far  *nkey;
   LPVOID   Index;
   } NodeOprPARMS;
::
NodeOprPARMS params;
int      xkey, xi;
USHORT err;
xkey = k1 ;
params.nkey = &xkey;
params.nString = VBCreateHlstr(Text1, lstrlen(Text1));
err = VBFireEvent(hctl, IEVENT_Heap_NodeAdd, &params);
VBDestroyHlstr(params.nString);
```

9. การจัดการกับลักษณะของตัวอักขระ (Font Property)

เมื่อได้สร้างคุณสมบัติที่เป็นสายอักขระแล้ว ก็ควรที่จะมีคุณสมบัติของลักษณะอักขระด้วย เพื่อใช้ในการกำหนดลักษณะต่าง ๆ ของสายอักขระที่จะนำมาแสดงผล

การจัดการลักษณะต่าง ๆ ของตัวอักขระจะทำได้ง่ายมาก โดยการนำคุณสมบัติมาตรฐานของลักษณะตัวอักขระ (Standard Font Properties) มารวมไว้ในตารางคุณสมบัติ และเพิ่มแฮนเดิลของลักษณะอักขระไว้ในโครงสร้างของผู้เขียนโปรแกรม เราใช้แฮนเดิลนี้ในการอ่านข้อมูลเกี่ยวกับลักษณะต่าง ๆ ของตัวอักขระเพื่อใช้ในการแสดงสายอักขระต่าง ๆ

```
typedef struct tagHEAP
```

```
{
 ::
HFONT hfont ;
 ::
} HEAP;
```

คุณสมบัติมาตรฐานของลักษณะตัวอักขระที่เพิ่มอยู่ในตารางคุณสมบัติมีดังนี้

```
PPROPINFO_STD_FONTNAME,
PPROPINFO_STD_FONTBOLD,
PPROPINFO_STD_FONTITALIC,
PPROPINFO_STD_FONTSTRIKE,
PPROPINFO_STD_FONTUNDER,
PPROPINFO_STD_FONTSIZE,
```

ในกระบวนงานของตัวควบคุมจะมีการทำงานเพิ่มเติมในเรื่องต่อไปนี้

- การทำงานเมื่อได้รับข้อความ WM_SETFONT และ WM_GETFONT
- ก่อนการแสดงผลของข้อมูลสายอักขระ ต้องทำการเลือกใช้ลักษณะของตัวอักขระจากคุณสมบัติที่ได้กำหนดไว้

การทำงานเมื่อได้รับข้อความ WM_SETFONT ก็คือการนำแฮนเดิลของลักษณะตัวอักขระใหม่เข้าไปเก็บไว้ในโครงสร้างของผู้เขียนโปรแกรม ส่วนข้อความ WM_GETFONT จะคืนค่าแฮนเดิลของลักษณะตัวอักขระที่เก็บไว้นั้นออกมาให้

```
case WM_SETFONT:
    LpheapDEREF(hctl)->hfont = (HFONT)wp;
    return 0;
case WM_GETFONT:
    return LpheapDEREF(hctl)->hfont;
```

วิชวลเบสิคจะจัดการในรายละเอียดด้านอื่น ๆ ของคุณสมบัติมาตรฐานต่าง ๆ ของลักษณะตัวอักขระให้เอง เช่น การเปลี่ยนแปลงค่าคุณสมบัติของลักษณะตัวอักขระ การแสดงรายการเลือกต่าง ๆ ของลักษณะตัวอักขระในหน้าต่างคุณสมบัติ เป็นต้น

ในฟังก์ชัน PaintTree หรือฟังก์ชันการแสดงผลอื่น ๆ จะต้องเลือกข้อมูลต่าง ๆ ของลักษณะตัวอักขระเข้ามาก่อนที่จะแสดงผลข้อมูลสายอักขระออกมา การเลือกทำได้โดยการใช้แฮนเดิลของลักษณะตัวอักขระที่เก็บอยู่ในโครงสร้างของผู้เขียนโปรแกรม ผ่านเข้าไปให้ฟังก์ชัน

SelectObject และก่อนจะจบการทำงานของฟังก์ชันการแสดงผล ก็ควรจะเลือกลักษณะตัวอักขระให้กลับเป็นแบบเดิมด้วย

```
VOID NEAR PaintTree
(
    HCTL hctl,
    HWND hwnd,
    HDC  hdc
)
{
   HFONT  hfontOld = NULL;
   LPSTR szBuffer = new __far char[STRLEN] ;
   if (LpheapDEREF(hctl)->hfont)
    hfontOld = SelectObject(hdc, LpheapDEREF(hctl)->hfont);
   //
   ::
   ::
   TextOut(hdc, x, y, szBuffer, lstrlen(szBuffer));
   ::
   MoveTo(hdc, x, (y-(nTextHeight/4)));
   LineTo(hdc, xp, (yp+nTextHeight));
   ::

   if (hfontOld)
     SelectObject(hdc, hfontOld);
       ::
}
```

10. การสร้างฟังก์ชันสำหรับการใช้งานในวิชวลเบสิค

ฟังก์ชันที่สร้างขึ้นมาเหล่านี้ เป็นฟังก์ชันสำหรับใช้งานในวิชวลเบสิค ซึ่งผู้พัฒนาโปรแกรมในวิชวลเบสิคสามารถใช้ฟังก์ชันเหล่านี้ทำงานทดแทนบางวิธีที่ใช้ไม่ได้กับตัวควบคุมนี้ การใช้งานฟังก์ชันเหล่านี้ในวิชวลเบสิค จะมีวิธีการใช้งานเหมือนกับการใช้ฟังก์ชันวินโดว์ API ที่ได้กล่าวไว้แล้วในบทที่ 3 ฟังก์ชันเหล่านี้ถ้านำมาจัดกลุ่มตามวัตถุประสงค์การใช้งานแล้ว จะแบ่งเป็นกลุ่มได้ดังต่อไปนี้

กลุ่มฟังก์ชันเพื่อการอ่านบัพข้อมูล ได้แก่

- HpGetTopVal คืนค่าข้อมูลที่เป็นคีย์ของบัพรากออกมาให้
- HpGetTopRep ใช้อ่านข้อมูลส่วนที่เป็นตัวแทนของบัพราก
- HpGetNode ใช้อ่านบัพข้อมูลโดยการกำหนดลำดับที่ของบัพที่ต้องการ
- HpGetLChild ใช้อ่านบัพลูกทางซ้ายของบัพข้อมูลที่กำหนด
- HpGetRChild ใช้อ่านบัพลูกทางขวาของบัพข้อมูลที่กำหนด
- HpGetParent ใช้อ่านบัพแม่ของบัพที่กำหนด
- HpGetCount คืนค่าจำนวนบัพทั้งหมดในโครงสร้างข้อมูลออกมาให้

กลุ่มฟังก์ชันเพื่อการเพิ่ม-การเปลี่ยนแปลงบัพข้อมูล ได้แก่

- HpAddNode ใช้ในการเพิ่มบัพข้อมูลใหม่เข้าไปในโครงสร้างข้อมูล
- HpChgNode ใช้ในการแก้ไขข้อมูลทั้งสองส่วนของบัพโดยการกำหนดลำดับที่ของบัพข้อมูลที่ต้องการจะแก้ไข
- HpDelNode ใช้ในการลบบัพข้อมูลออกจากโครงสร้างข้อมูลโดยการกำหนดลำดับที่ของบัพที่ต้องการลบ
- HpPopNode ใช้ในการอ่านบัพรากออกจากโครงสร้างข้อมูล ซึ่งจะเหมือนกับการอ่านและลบบัพที่ 0 นั่นเอง

กลุ่มฟังก์ชันที่มีการใช้งานในระดับโครงสร้างข้อมูล ได้แก่

- HpMerge ใช้รวมข้อมูล 2 โครงสร้างข้อมูลเข้าด้วยกัน
- HpSort ใช้จัดเรียงลำดับข้อมูลในโครงสร้างข้อมูล

การสร้างฟังก์ชันเหล่านี้ จะต้องมีการกำหนดรูปแบบและชนิดของฟังก์ชันเป็นการเฉพาะ เพื่อให้โปรแกรมวิชวลเบสิคสามารถเรียกใช้ฟังก์ชันเหล่านี้ได้ ในที่นี้จะใช้ฟังก์ชัน HpMerge มาเป็นตัวอย่างในการอธิบาย

```
extern "C"
void FAR PASCAL __export HpMerge ( HWND hwnd1, HWND hwnd2 )
```

คำสำคัญ (Keyword) FAR จะทำให้ฟังก์ชันมีตัวชี้ที่อยู่เป็นแบบ 32 บิต ซึ่งจะเป็นการกำหนดให้ฟังก์ชันนี้อยู่ตำแหน่งใดในหน่วยความจำก็ได้ ไม่จำเป็นต้องอยู่ในเซกเมนต์เดียวกับโปรแกรมผู้เรียกใช้ก็ได้ และจะถูกเรียกใช้จากส่วนใดในหน่วยความจำก็ได้เช่นกัน และคำสำคัญ __export จะทำให้ฟังก์ชันนี้ถูกเรียกใช้จากภายนอกของแฟ้มกระทำการ (Executable file) ของตัวมันเองได้

เนื่องจากฟังก์ชันเหล่านี้อยู่ในแฟ้มของตัวควบคุม ซึ่งเป็นแฟ้ม DLL ชนิดหนึ่ง ดังนั้นเมื่อมีการใช้ฟังก์ชันเหล่านี้ก็จะมีการเชื่อมโยงโปรแกรมแบบพลวัตด้วย

การใช้งานในบางฟังก์ชันจะต้องมีการรับส่งข้อมูลกับวิชวลเบสิคด้วยบัพข้อมูล ซึ่งบัพข้อมูลนี้จะประกอบด้วยข้อมูล 2 ส่วน จึงต้องมีการนิยามโครงสร้างชื่อ Node_type ไว้ในแฟ้มของตัวควบคุม โครงสร้างนี้จะประกอบด้วยข้อมูลทั้ง 2 ส่วน เราใช้โครงสร้างนี้ในการรับส่งบัพข้อมูลไปมาระหว่างฟังก์ชันที่สร้างขึ้นมานี้กับวิชวลเบสิค

```
typedef struct Node_tag {
  HLSTR itmrep ;
  int itmkey ;
  } Node_type;
```

การใช้งานในวิชวลเบสิคก็ต้องมีการกำหนดชนิดของข้อมูลที่เป็นโครงสร้างของบัพขึ้นมาด้วยเช่นกัน ซึ่งใช้ประโยค Type ในการกำหนด โดยข้อมูลส่วนแรกจะเป็นสายอักขระของวิชวลเบสิค ข้อมูลส่วนที่สองจะเป็นเลขจำนวนเต็ม เช่น

```
Type Node
   ItemRep As String
   ItemKey As Integer
End Type
```

ในการทำงานของฟังก์ชันที่มีการรับส่งอาร์กิวเมนต์กับวิชวลเบสิคด้วยบัพข้อมูลนั้น ถ้าฟังก์ชันเหล่านั้น ไม่สามารถทำงานได้สำเร็จตามที่ต้องการ เช่น การแก้ไขหรือการลบบัพข้อมูล ถ้าฟังก์ชันนั้นไม่พบบัพข้อมูลที่กำหนด ฟังก์ชันก็จะคืนโครงสร้างของบัพออกมาพร้อมกับทำให้ข้อมูลส่วนที่เป็นเลขจำนวนเต็มนั้นเป็น -1 เพื่อให้ผู้ใช้งานฟังก์ชันเหล่านี้ในวิชวลเบสิคทราบได้ว่าฟังก์ชันนั้นทำงานไม่สำเร็จ

11. การสร้างสัญรูปแทนตัวควบคุมในหน้าต่างกล่องเครื่องมือ

ในการสร้างสัญรูปเพื่อใช้แทนตัวควบคุมในหน้าต่างกล่องเครื่องมือนั้น จะต้องสร้างขึ้นมาเพื่อใช้กับจอภาพแบบ VGA, Monochrome และ EGA (ส่วน CGA นั้น วินโดว์จะทำงานเหมือนกับจอ Monochrome) ซึ่งจะต้องใช้แฟ้มรูปภาพแบบบิตแมพ 4 แฟ้ม โดยเป็นแฟ้มสำหรับจอภาพแบบ VGA 2 แฟ้ม ใช้สำหรับแสดงขณะที่ยังไม่ได้มีการใช้เมาส์คลิกเลือกที่สัญรูปนั้น 1 แฟ้ม และขณะที่ใช้เมาส์คลิกเลือกที่สัญรูปนั้นอีก 1 แฟ้ม และเป็นแฟ้มสำหรับจอ EGA และ Monochrome อีกอย่างละ 1 แฟ้ม สำหรับจอภาพทั้ง 2 ชนิดหลังนี้ เมื่อมีการใช้เมาส์คลิกเลือกที่สัญรูป วิชวลเบสิคจะใช้การสลับสีของแต่ละจุดในภาพแทน ส่วนจอภาพขนาดใหญ่จะใช้แฟ้มทั้ง 2 แฟ้มที่ใช้สำหรับจอ VGA

แฟ้มรูปภาพทั้ง 4 แฟ้มเป็นทรัพยากรที่ตัวควบคุมต้องใช้ จึงต้องมีการกำหนดหมายเลข (ID) แทนแต่ละแฟ้ม การกำหนดจะกระทำอยู่ในแฟ้ม HPCTL.RC ในตารางที่ 5.5 แสดงการกำหนดหมายเลขของแฟ้มรูปภาพแต่ละแฟ้ม

ชนิดของจอภาพ	หมายเลข
จอ VGA ขณะที่สัญรูปไม่ได้ถูกเลือกด้วยเมาส์	idBmpPalette เป็นเลขจำนวนเต็มซึ่งจะต้องนำไปกำหนดไว้ในต้นแบบของตัวควบคุมด้วย (HPCTL.H บรรทัดที่ 205)
จอ VGA ขณะที่สัญรูปถูกเลือกด้วยเมาส์	idBmpPalette + 1
จอ Monochrome	idBmpPalette + 3
จอ EGA	idBmpPalette + 6

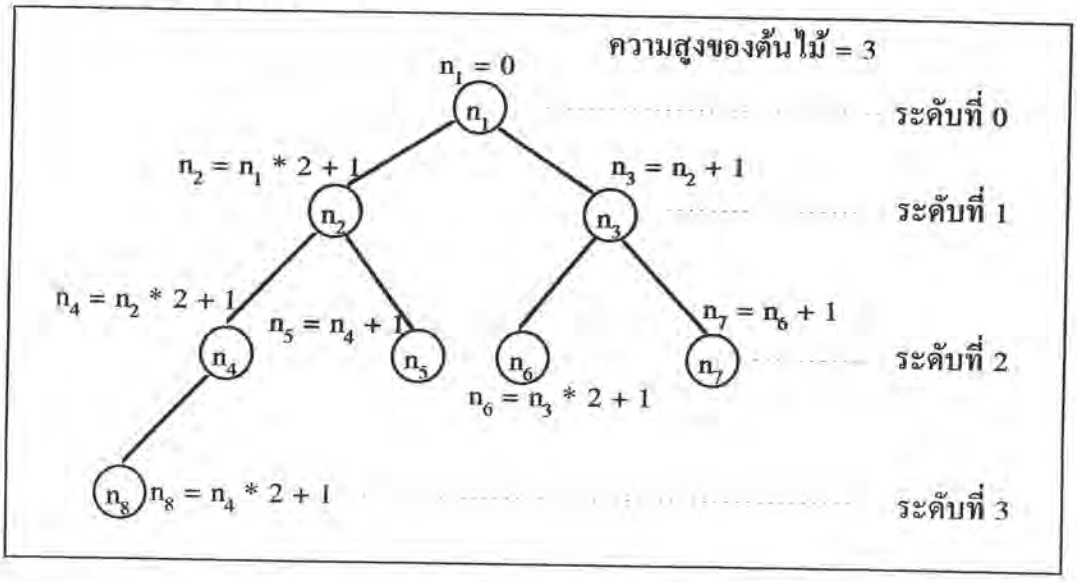
ตารางที่ 5.5 หมายเลข (ID) ของแฟ้มรูปภาพแต่ละแฟ้ม

เราจะกำหนด idBmpPalette ให้เป็นเลขจำนวนเต็มใดก็ได้สำหรับจอภาพ VGA ขณะที่สัญรูปยังไม่ได้ถูกเลือกด้วยเมาส์ ส่วนภาพอื่นกำหนดโดยการเพิ่มค่าขึ้นอีก 1, 3 และ 6 ตามลำดับ การกำหนดหมายเลขไปซ้ำกับตัวควบคุมอื่น ก็จะไม่มีผลต่อกันเลย

รูปภาพทั้ง 4 แฟ้มนี้ ถูกสร้างขึ้นโดยใช้โปรแกรม Microsoft Image Editor โดยคัดลอกมาจากแฟ้มรูปภาพที่อยู่ในสารบบ CIRC2 นำมาใช้เป็นรูปภาพต้นแบบในการแก้ไขปรับปรุงให้เป็นสัญรูปของตัวควบคุมนี้

ขั้นตอนวิธี (Algorithms) ที่สำคัญในการจัดการโครงสร้างข้อมูล

โครงสร้างข้อมูลของตัวควบคุมนี้จัดเก็บอยู่ในหน่วยความจำในรูปแบบของแถวลำดับ การอ้างถึงแต่ละบัพของโครงสร้างข้อมูล จะอ้างโดยใช้ตำแหน่งที่อยู่ในแถวลำดับเป็นหมายเลขของบัพนั้น เช่น บัพรากหรือบัพที่ 0 ก็คือช่องที่ 0 ในแถวลำดับ ในรูปที่ 5.1 เป็นการแสดงตำแหน่งของบัพที่เก็บอยู่ในแถวลำดับ โดยขั้นตอนวิธีต่าง ๆที่จะกล่าวถึง จะมีการอ้างตำแหน่งของบัพต่าง ๆ ตามที่แสดงอยู่ในรูปที่ 5.1



รูปที่ 5.1 แสดงตำแหน่งของบัพที่เก็บอยู่ในแถวลำดับ

1. การจัดต้นไม้ย่อยที่กำหนดให้เป็นฮีป

ในขั้นตอนวิธีนี้เป็นการจัดต้นไม้ที่กำหนดเข้ามาให้เป็นฮีป ต้นไม้ที่กำหนดเข้ามาอาจเป็นต้นไม้ย่อยหรือต้นไม้ใหญ่ของโครงสร้างข้อมูลก็ได้ ดังนั้นบัพรากของต้นไม้ที่กำหนดจึงเป็นบัพเริ่มต้นในการพิจารณา นอกจากนั้นในขั้นตอนวิธีนี้ยังมีการกำหนดตำแหน่งที่เป็นบัพสุดท้ายในต้นไม้ ดังนั้นการพิจารณาบัพข้อมูล จะเริ่มพิจารณาจากบัพรากของต้นไม้ที่กำหนด และจะไม่เกินไปจากตำแหน่งของบัพสุดท้ายที่กำหนด การทำงานของขั้นตอนวิธีนี้ทำงานภายใต้ข้อสมมติฐานที่ว่า ต้นไม้ย่อยทั้งหมดที่อยู่ภายใต้ต้นไม้ที่กำหนดนี้ เป็นต้นไม้ที่เป็นฮีปอยู่แล้ว มีเพียงต้นไม้ที่กำหนดนี้ยังอาจไม่เป็นฮีป การอธิบายขั้นตอนวิธีนี้ จะกล่าวถึงเฉพาะกรณีการจัดให้ข้อมูลที่มากกว่าอยู่ข้างบน โดยการทำงานของขั้นตอนวิธี จะมีขั้นตอนดังนี้

1) ให้ strat = ตำแหน่งที่อยู่ในแถวลำดับ ซึ่งเป็นบัพรากของต้นไม้ที่กำหนด

ให้ maxheap = ตำแหน่งที่เป็นบัพสุดท้ายในต้นไม้

2) คัดลอกค่าของบัพ[strat] ไปเก็บไว้ในบัพชั่วคราว

3) คำนวนหาตำแหน่งบัพลูกของบัพ[strat] โดยการกำหนด

son = 2 * start + 1

4) ถ้า son <= maxheap แสดงว่าตำแหน่งของบัพลูกเป็นบัพที่อยู่ภายในต้นไม้ที่กำหนด ก็จะหาบัพที่มีค่ามากกว่าระหว่างบัพลูกทางซ้ายและขวา มาเป็นบัพ[son] แต่ถ้า son มากกว่า maxheap ให้ข้ามไปทำงานในข้อ 7)

5) ถ้าบัพชั่วคราว >= บัพ[son] ให้ไปทำงานในข้อ 7) แต่ถ้าบัพชั่วคราวมีค่าน้อยกว่าบัพ[son] ให้ทำงานในข้อ 6)

6) ในข้อนี้เป็นการนำค่าของบัพลูกเข้าไปไว้ในบัพแม่ ซึ่งก็คือนำค่าของบัพ[son] ไปไว้ในบัพ[start] แล้วลงไปพิจารณาต้นไม้ย่อยในระดับถัดลงไป โดยให้บัพลูกเป็นบัพเริ่มต้น พร้อมกับคำนวนหาตำแหน่งของบัพลูกของต้นไม้ในระดับที่อยู่ถัดลงไปนี้ โดยสรุปให้เป็นประโยคของการกำหนดค่าได้ดังนี้

บัพ[start] = บัพ[son] และให้คำนวน

start = son

son = 2 * start + 1

แล้วไปทำงานในข้อ 4)

7) ให้นำค่าของบัพชั่วคราวไปไว้ในบัพ [start]

8) จบการทำงาน

2. <u>การเคลื่อนย้ายบัพที่กำหนดขึ้นข้างบนเพื่อจัดโครงสร้างข้อมูลให้เป็นฮีป</u>

เป็นวิธีหนึ่งของการจัดโครงสร้างข้อมูลให้มีคุณสมบัติเป็นฮีป เนื่องจากบัพที่กำหนดนั้นมีการเปลี่ยนแปลงของข้อมูลที่ทำให้ต้นไม้ย่อยที่อยู่ข้างบนไม่มีคุณสมบัติเป็นฮีป เช่น เป็นบัพใหม่ที่ถูกเพิ่มเข้ามา หรืออาจเป็นบัพเดิมที่ถูกเปลี่ยนแปลงข้อมูล ขั้นตอนวิธีนี้จึงพยายามเคลื่อนบัพที่กำหนดขึ้นไประดับบนเพื่อหาตำแหน่งที่เหมาะสม ซึ่งจะทำให้ต้นไม้ย่อยที่อยู่ข้างบนมีคุณสมบัติเป็นฮีปตามเดิม โดยขั้นตอนวิธีนี้ทำงานภายใต้ข้อสมมติฐานที่ว่า ต้นไม้ย่อยทั้งหมดมีคุณสมบัติที่เป็นฮีปอยู่แล้ว ยกเว้นต้นไม้ที่อยู่ระดับบนที่อยู่ในสายเดียวกันของบัพที่กำหนดซึ่งไม่เป็นฮีป โดยมีขั้นตอนวิธีการทำงานดังนี้

1) ให้ noden เป็นตำแหน่งที่อยู่ในแถวลำดับ ซึ่งเป็นบัพที่กำหนดเข้ามา

2) ตำแหน่งบัพแม่ของบัพ noden จะหาได้จาก

parent = (noden + 1) / 2 - 1 ซึ่งผลลัพธ์จากการหารจะพิจารณาแต่ค่าที่เป็นจำนวนเต็ม ไม่สนใจค่าของทศนิยม

3) คัดลอกค่าของบัพ[noden] ไปเก็บไว้ในบัพชั่วคราว

4) ถ้าบัพ[parent] > บัพชั่วคราว และ parent >= 0 ให้ทำงานในข้อ 5) มิฉะนั้นให้ไปทำข้อ 8)

5) คัดลอกค่า บัพ[noden] = บัพ[parent]

noden = parent เป็นการเตรียมย้ายขึ้นไปพิจารณาบัพที่อยู่ระดับบน

6) ถ้า parent <= 0 ให้ไปทำข้อ 8) ซึ่งหมายความว่าการพิจารณาเปรียบเทียบได้ทำมาถึงบัพรากแล้ว ซึ่งจะไม่มีบัพที่อยู่ในระดับที่ถัดขึ้นไปอีกแล้ว

แต่ถ้า parent > 0 ให้ทำข้อ 7)

7) คำนวนหาบัพแม่ขึ้นไปอีกระดับหนึ่ง

parent = (noden + 1) / 2 -1 แล้วไปทำข้อ 4)

8) คัดลอกค่าจากบัพชั่วคราวไปเก็บไว้ในบัพ[noden]

9) จบการทำงาน

3. <u>การจัดโครงสร้างข้อมูลให้เป็นฮีป</u>

เป็นการนำโครงสร้างข้อมูลที่อาจยังไม่เป็นฮีปมาจัดให้มีคุณสมบัติเป็นฮีป โดยมีหลักการทำงานอยู่ที่การจัดต้นไม้ย่อยในระดับต่าง ๆ ให้เป็นฮีป ก็จะได้โครงสร้างข้อมูลที่เป็นฮีปไปด้วย การจัดต้นไม้ย่อยให้เป็นฮีปจะเริ่มจากต้นไม้ย่อยในระดับล่างสุด จัดไล่ขึ้นไปจนถึงระดับบนสุด โดยมีขั้นตอนการทำงานดังนี้

1) คำนวนหาตำแหน่งของบัพรากของต้นไม้ย่อยต้นสุดท้ายในโครงสร้างข้อมูลได้โดย

HpCount = จำนวนบัพทั้งหมดในโครงสร้างข้อมูล

i = ตำแหน่งบัพรากของต้นไม้ย่อย จะได้ว่า

i = (HpCount/2) - 1 จะเป็นต้นไม้ย่อยต้นสุดท้ายในโครงสร้างข้อมูล (เนื่องจาก HpCount และ i เป็นจำนวนเต็ม ดังนั้นผลหารจึงพิจารณาแต่ค่าที่เป็นจำนวนเต็ม ไม่สนใจค่าเป็นทศนิยม)

2) ถ้า i >= 0 คือยังไม่เกินตำแหน่งบัพรากของโครงสร้างข้อมูล ให้ไปทำงานในข้อ 3) แต่ถ้า i น้อยกว่า 0 ให้ทำข้อ 6)

3) ให้ทำขั้นตอนวิธีการจัดต้นไม้ย่อยให้เป็นฮีปในข้อ 1 โดยให้ i เป็นตำแหน่งเริ่มต้นของต้นไม้ย่อยที่จะพิจารณา และมี (HpCount - 1) เป็นตำแหน่งของบัพสุดท้ายในต้นไม้

4) เลือกต้นไม้ย่อยที่อยู่ถัดจากต้นไม้ i โดยกำหนดให้ i = i - 1

5) กลับขึ้นไปทำงานในข้อ 2)

6) จบการทำงาน

4. <u>การเพิ่มบัพข้อมูลใหม่เข้าไปในโครงสร้างข้อมูลฮีป</u>

1) กำหนดให้ MAXLIST เป็นจำนวนบัพที่จะมีได้มากที่สุดในโครงสร้างข้อมูล

2) ตรวจสอบความถูกต้องของข้อมูลที่จะมาเป็นบัพใหม่ โดยข้อมูลส่วนที่เป็นคีย์ต้องมากกว่า 0 และ ข้อมูลส่วนที่เป็นตัวแทนของบัพ ต้องมีข้อความอยู่ ถ้าถูกต้องให้ไปข้อ 3) แต่ถ้าไม่ถูกต้องให้ไปข้อ 8)

3) ถ้าตัวนับจำนวนบัพ >= MAXLIST ให้ไปข้อ 8)

4) เพิ่มตัวนับจำนวนบัพในโครงสร้างข้อมูลขึ้นอีก 1

5) เตรียมพื้นที่สำหรับบัพใหม่ให้อยู่ต่อท้ายจากบัพสุดท้ายในแถวลำดับ

6) นำข้อมูลของบัพใหม่เข้าไปเก็บไว้ในพื้นที่ที่เตรียมไว้ในข้อ 5)

7) ใช้ขั้นตอนวิธีในข้อ 2 ทำการการเคลื่อนย้ายบัพใหม่ขึ้นข้างบนเพื่อจัดโครงสร้างข้อมูลให้เป็นฮีป

8) จบการทำงาน

5. <u>การเปลี่ยนแปลงบัพข้อมูลในโครงสร้างข้อมูลฮีป</u>

1) ตรวจสอบความถูกต้องของข้อมูลที่จะเข้าไปแทนที่ โดยข้อมูลส่วนที่เป็นคีย์ต้องมากกว่า 0 และข้อมูลส่วนที่เป็นตัวแทนของบัพต้องมีข้อความอยู่ ถ้าถูกต้องให้ทำข้อ 2) แต่ถ้าไม่ถูกให้ไปข้อ 5)

2) ตรวจสอบว่า หมายเลขของบัพที่ต้องการแก้ไขนั้นเป็นบัพที่มีอยู่ในโครงสร้างข้อมูลจริง โดยถ้าหมายเลขบัพที่กำหนดมีค่าน้อยกว่าค่าของตัวนับจำนวนบัพ ให้ทำข้อ 3) แต่ถ้าไม่น้อยกว่าให้ไปข้อ 5)

3) นำข้อมูลในข้อ 1) เข้าไปแทนที่บัพข้อมูลที่กำหนด

4) ถ้าบัพที่เปลี่ยนแปลงทำให้ต้นไม้ที่อยู่ในระดับข้างบนไม่เป็นฮีป แล้ว

ให้ทำขั้นตอนวิธีในข้อ 2 ทำการการเคลื่อนย้ายบัพที่เปลี่ยนแปลงขึ้นข้างบน เพื่อจัดโครงสร้างข้อมูลให้เป็นฮีป

แต่ถ้าทำให้ต้นไม้ย่อยที่อยู่ระดับข้างล่างไม่เป็นฮีป ให้ทำขั้นตอนวิธีการจัดต้นไม้ย่อยให้เป็นฮีปในข้อ 1 โดยให้บัพที่เปลี่ยนแปลงเป็นตำแหน่งเริ่มต้นของต้นไม้ย่อยที่จะพิจารณา

5) จบการทำงาน

6. การลบบัพข้อมูลออกจากโครงสร้างข้อมูลฮีป

1) หมายเลขบัพข้อมูลที่ต้องการลบจะต้องมีอยู่ในโครงสร้างข้อมูลจริง โดยหมายเลขบัพที่กำหนดต้องมีค่าน้อยกว่าค่าของตัวนับจำนวนบัพ แล้วให้ทำข้อ 2) มิฉะนั้น ให้ทำข้อ 5)

2) นำบัพข้อมูลบัพสุดท้ายในโครงสร้างข้อมูลเข้าไปแทนที่บัพข้อมูลที่ต้องการจะลบ

3) ลดค่าตัวนับจำนวนบัพลง 1

4) ถ้าการลบบัพทำให้ต้นไม้ที่อยู่ในระดับข้างบนไม่เป็นฮีป แล้ว

ให้ทำขั้นตอนวิธีในข้อ 2 ทำการการเคลื่อนย้ายบัพที่เปลี่ยนแปลงขึ้นข้างบน เพื่อจัดโครงสร้างข้อมูลให้เป็นฮีป

แต่ถ้าทำให้ต้นไม้ย่อยที่อยู่ระดับข้างล่างไม่เป็นฮีป ให้ทำขั้นตอนวิธีการจัดต้นไม้ย่อยให้เป็นฮีปในข้อ 1 โดยให้บัพที่เปลี่ยนแปลงเป็นตำแหน่งเริ่มต้นของต้นไม้ย่อยที่จะพิจารณา

5) จบการทำงาน

7. การรวมโครงสร้างข้อมูล 2 ชุด เข้าด้วยกัน

เป็นการคัดลอกข้อมูลในโครงสร้างข้อมูลชุดที่ 2 เข้ามายังโครงสร้างข้อมูลชุดที่ 1 โดยจำนวนบัพที่จะเพิ่มเข้ามาในโครงสร้างข้อมูลชุดที่ 1 นี้ จะต้องไม่ทำให้จำนวนบัพในโครงสร้างข้อมูลชุดที่ 1 มากกว่า MAXLIST ถ้าเกินกว่า MAXLIST จะตัดส่วนที่เกินออก ไม่นำเข้ามารวม

1) ให้ inc_node เป็นจำนวนบัพที่จะทำการคัดข้อมูลจากโครงสร้างข้อมูลในชุดที่ 2 เข้ามายังโครงสร้างข้อมูลชุดที่ 1

2) ทำการเตรียมพื้นที่ในแถวลำดับของโครงสร้างข้อมูลชุดที่ 1 ต่อจากบัพสุดท้ายออกไปเป็นจำนวน inc_node บัพ

3) คัดลอกบัพข้อมูลจากโครงสร้างข้อมูลชุดที่ 2 เข้ามาต่อท้ายบัพสุดท้ายในโครงสร้างข้อมูลชุดที่ 1 เป็นจำนวน inc_node บัพในพื้นที่ของแถวลำดับที่เตรียมไว้ในข้อ 2)

4) เพิ่มตัวนับจำนวนบัพของโครงสร้างข้อมูลชุดที่ 1 ขึ้นอีกจำนวน inc_node บัพ

5) ใช้ขั้นตอนวิธีในข้อ 3 ทำการจัดโครงสร้างข้อมูลในชุดที่ 1 ให้เป็นฮีป

6) จบการทำงาน

8. การจัดเรียงลำดับข้อมูลที่อยู่ในโครงสร้างข้อมูลด้วยวิธีการของ Heapsort

การจัดเรียงลำดับข้อมูลด้วย Heapsort นี้ต้องนำโครงสร้างข้อมูลที่มีคุณสมบัติเป็นฮีปอยู่ก่อนแล้วมาทำการจัดเรียงจึงจะเรียงลำดับได้ถูกต้อง ในที่นี้จะกล่าวถึงในกรณีที่โครงสร้างข้อมูลฮีปเป็นแบบที่ค่าที่มากกว่าจะอยู่ข้างบน เนื่องจากบัพรากของโครงสร้างข้อมูลฮีปเป็นค่าที่มากที่สุดในโครงสร้างข้อมูล ดังนั้น บัพรากจะเป็นบัพที่ใช้ได้ทันทีในการจัดเรียง จึงคัดลอกบัพรากไปไว้ในตำแหน่งสุดท้ายของโครงสร้างข้อมูลได้เลย แล้วจึงทำการจัดโครงสร้างข้อมูลส่วนที่เหลือให้เป็นฮีปเหมือนเดิม จากนั้นก็จะเริ่มคัดลอกบัพรากไปไว้ในตำแหน่งที่อยู่ก่อนตำแหน่งท้ายสุดของโครงสร้างข้อมูล และจัดโครงสร้างข้อมูลส่วนที่เหลือให้เป็นฮีป ทำเช่นนี้ไปเรื่อย ๆ จนครบทุกบัพ ก็จะได้โครงสร้างข้อมูลฮีปที่มีการจัดเรียงลำดับจากน้อยไปหามากอยู่ภายในโครงสร้างข้อมูล โดยมีขั้นตอนการทำงานดังนี้

1) ให้ i เป็นตำแหน่งของบัพสุดท้ายในโครงสร้างข้อมูล

2) ถ้า i >= 1 ให้ทำในข้อ 3) มิฉะนั้น ให้ทำข้อ 7)

3) คัดลอกบัพ[i] ไปไว้ในบัพชั่วคราว

4) คัดลอกบัพรากไปไว้ที่บัพ[i]

5) ใช้ขั้นตอนวิธีในข้อ 1 ทำการจัดต้นไม้ย่อยที่กำหนดให้เป็นฮีปโดยให้จุดเริ่มต้นของต้นไม้ย่อยที่กำหนดเริ่มจากตำแหน่งที่ 0 ในแถวลำดับ และให้ตำแหน่งสุดท้ายของต้นไม้ย่อยอยู่ที่ (i-1)

6) ลดขนาดของโครงสร้างข้อมูลที่จะพิจารณาลงมา 1 คือ

i = i - 1 แล้วกลับขึ้นไปทำข้อ 2) ซึ่งจะเห็นว่าการทำงานในแต่ละรอบ จำนวนบัพของโครงสร้างข้อมูลที่จะใช้ในการจัดเรียงจะลดลงรอบละ 1 บัพ ซึ่งจะวนทำเช่นนี้ไปจนกระทั่งเหลือบัพข้อมูลที่จะจัดเรียงเพียง 1 บัพ จึงหยุดการทำงาน

หลังจากการจัดเรียงลำดับเสร็จแล้ว โครงสร้างจะมีลำดับของข้อมูลสลับกับคุณสมบัติของฮีปก่อนที่จะจัดเรียง เช่น ถ้าเป็นโครงสร้างข้อมูลฮีปชนิดที่ค่ามากกว่าอยู่ข้างบน หลังจากจัดเรียงเสร็จแล้ว จะได้โครงสร้างข้อมูลที่เรียงจากน้อยไปหามาก หรือค่าน้อยกว่าอยู่ข้างบนนั่นเอง ดังนั้น งานในส่วนที่เหลือจึงเป็นการสลับตำแหน่งระหว่างบัพที่ส่วนหน้ากับบัพที่อยู่ส่วนหลังของโครงสร้างข้อมูล จนกระทั่งมาพบกันที่ตรงกลางของโครงสร้างข้อมูล

7) กำหนดให้ j เป็นตำแหน่งของบัพแรกในโครงสร้างข้อมูล

j = 0 ซึ่งเป็นจุดเริ่มต้นของส่วนหน้าของโครงสร้างข้อมูล

กำหนดให้ k เป็นตำแหน่งของบัพสุดท้ายในโครงสร้างข้อมูล

k = ตัวนับจำนวนบัพ - 1 ซึ่งเป็นจุดเริ่มต้นของส่วนหลังของโครงสร้างข้อมูล

8) ถ้า j < k คือยังไม่มาพบกันที่ตรงกลางของโครงสร้างข้อมูล ให้ไปทำข้อ 9) มิฉะนั้นให้ไปทำข้อ 14)

9) คัดลอกบัพ[j] ไปไว้ในบัพชั่วคราว

10) คัดลอกบัพ[k] ไปไว้ในบัพ[j]

11) คัดลอกบัพชั่วคราวไปไว้ในบัพ[k]

12) ให้ j เพิ่มขึ้นอีก 1 และให้ k ลดลงอีก 1 ซึ่งเขียนเป็นประโยคการคำนวนได้เป็น

j = j + 1

k = k - 1

13) กลับขึ้นไปทำข้อ 8)

14) จบการทำงาน

9. <u>การแสดงโครงสร้างข้อมูลในรูปแบบของโครงสร้างต้นไม้</u>

การแสดงโครงสร้างข้อมูลจะมีการจัดให้รูปแบบต้นไม้ของโครงสร้างข้อมูลอยู่ตรงกลาง เพื่อให้ได้ภาพของต้นไม้ที่มีความสมดุลย์อยู่ภายในกรอบของตัวควบคุม การวาดแต่ละบัพของโครงสร้างข้อมูล จะพยายามวาดให้สามารถมองเห็นได้ทั้งหมดภายในกรอบของตัวควบคุม การแสดงผลแต่ละครั้ง จะนำพื้นที่ของตัวควบคุม และจำนวนบัพทั้งหมดที่มีอยู่ในขณะนั้น มาคำนวนหาจำนวนระดับของโครงสร้างข้อมูล ระยะห่างระหว่างระดับ และระยะห่างระหว่างบัพในแต่ละระดับ ซึ่งค่าที่คำนวนได้เหล่านี้จะใช้ในการกำหนดตำแหน่งของแต่ละบัพที่จะถูกวาดลงบนตัวควบคุม การแสดงโครงสร้างข้อมูล จะวาดจากบนลงล่าง เริ่มจากบัพรากไล่ลงมาทีละระดับ ส่วนเส้นกิ่งแสดงความสัมพันธ์ระหว่างบัพจะลากจากบัพลูกขึ้นไปหาบัพแม่ โดยระยะห่างระหว่างบัพจะต้องคำนวนใหม่ในทุกระดับเนื่องจากแต่ละระดับจะมีจำนวนบัพไม่เท่ากัน โดยขั้นตอนวิธีการแสดงโครงสร้างข้อมูลในรูปแบบของต้นไม้จะมีดังนี้

1) กำหนดให้ HpCount เป็นจำนวนบัพทั้งหมดที่มีอยู่ในโครงสร้างข้อมูล

กำหนดให้ nTextHeight เป็นความสูงของตัวอักขระที่ใช้ในการแสดงผลของบัพข้อมูล

กำหนดให้ PgWidth เป็นความยาวของตัวควบคุม

กำหนดให้ PgHeight เป็นความกว้างของตัวควบคุม

กำหนดให้ h เป็นจำนวนระดับที่มีได้ในโครงสร้างข้อมูลโดยเริ่มนับจาก 0 ซึ่งคำนวนจาก

$$h = \lfloor \log_2 n \rfloor$$

กำหนดให้ levelh เป็นระยะห่างระหว่างบัพ โดยคำนวนจาก

levelh = PgHeight / (h+1) ซึ่งรวมระยะห่างจากขอบบนและล่างข้างละครึ่งหนึ่งของระยะระหว่างระดับ

กำหนดให้ levelh2 เป็นระยะห่างจากขอบบนของการแสดงผล โดยคำนวนจาก

levelh2 = levleh / 2

กำหนดให้ nodew เป็นระยะห่างระหว่างบัพในแต่ละระดับ โดยคำนวนจาก

nodew = PgWidth / 2 สำหรับในระดับที่ 0

2) แสดงบัพ[0]ที่ตำแหน่ง nodew, levleh2 บนตัวควบคุม โดยตำแหน่ง nodew เป็นตำแหน่งในแนวนอนนับระยะจากขอบซ้ายของตัวควบคุมออกมา และเป็นตำแหน่งกึ่งกลางของข้อมูลที่จะนำมาแสดง ส่วน levleh2 เป็นตำแหน่งในแนวตั้งนับจากระยะขอบบนของตัวควบคุมลงมา และเป็นตำแหน่งด้านบนของข้อมูลที่จะแสดง เช่น ข้อมูล "123" ข้อมูลเลข "2" จะปรากฏที่ตำแหน่ง nodew ในแนวนอน และขอบบนของเลข "2" จะอยู่ที่ตำแหน่ง levelh2

3) กำหนดให้ accnod เป็นจำนวนบัพสะสมที่ได้วาดออกไปแล้ว

accnod = 1

กำหนดให้ nodep เป็นจำนวนบัพที่จะมีได้สำหรับระดับที่อยู่ก่อนหน้านี้

nodep = 1

กำหนดให้ y เป็นระยะสะสมที่ได้ใช้ไปในการวาดบัพนับจากขอบบนของตัวควบคุมลงมา

y = lelveh2

กำหนดให้ entinx เป็นตำแหน่งของบัพบนแถวลำดับที่เพิ่งได้วาดออกไป

entinx = 0

กำหนดให้ avlnod เป็นจำนวนบัพที่จะมีได้มากที่สุดในแต่ละระดับ

กำหนดให้ aclnod เป็นจำนวนบัพที่มีอยู่จริงในระดับต่าง ๆ

4) กำหนดให้ i เป็นระดับถัดไปของโครงสร้างข้อมูลที่จะเริ่มวาดออกมา

i = 1

5) ถ้า i <= h คือยังไม่เกินระดับสุดท้ายของโครงสร้างข้อมูล ให้ทำข้อ 6) มิฉะนั้นให้ทำข้อ 22)

6) กำหนดให้ yp เป็นระยะสะสมที่ได้ใช้ไปในการวาดบัพนับจากขอบบนของตัวควบคุมลงมาถึงระดับที่อยู่ก่อนหน้าระดับที่กำลังจะวาด

yp = y

7) กำหนดตำแหน่งในแนวตั้งของระดับที่กำลังวาดบัพข้อมูลอยู่ในปัจจุบัน

y = y + levelh

8) หาจำนวนบัพที่จะมีได้มากที่สุดในระดับปัจจุบัน

avlnod = nodep * 2

9) ถ้า (accnod + avlnod) <= HpCount แล้ว

จำนวนบัพสะสม accnod = accnod + avlnod

จำนวนบัพที่มีอยู่จริงในระดับปัจจุบัน aclnod = avlnod

มิฉะนั้น aclnod = HpCount - accnod

accnod = accnod + aclnod

10) ระยะห่างระหว่างบัพในระดับปัจจุบัน คือ

nodew = PgWidth / avlnod

ระยะที่เว้นห่างจากขอบซ้ายในระดับปัจจุบัน

nodew2 = nodew / 2

ระยะห่างระหว่างบัพของระดับที่อยู่ติดกันข้างบน

nodewp = PgWidth / nodep

ระยะที่เว้นห่างจากขอบซ้ายของระดับที่อยู่ติดกันข้างบน

nodewp2 = nodewp / 2

11) กำหนดให้ x เป็นระยะสะสมในแนวนอนที่ได้วาดบัพออกไปแล้วของระดับปัจจุบัน

x = nodew2

กำหนดให้ xp เป็นระยะสะสมในแนวนอนที่ได้วาดบัพออกไปแล้วของระดับที่อยู่ติดกันข้างบน

xp = nodewp2

12) กำหนดให้ j เป็นตัวนับบัพของระดับปัจจุบันที่จะถูกวาดออกมา

j = 1

13) ถ้า j <= aclnod คือยังไม่เกินจำนวนบัพที่จะมีให้วาดได้ของระดับปัจจุบัน แล้วให้ทำข้อ 14) มิฉะนั้นให้ทำข้อ 19)

14) วาดบัพ[entinx+j] ที่ตำแหน่ง x, y

ลากเส้นกิ่งจากตำแหน่ง x, (y-(nTextHeight/4)) ขึ้นไปยังตำแหน่ง xp, (yp+nTextHeight) ซึ่งเป็นการลากเส้นห่างจากด้านบนของบัพที่วาดเป็นระยะ 1/4 ของความสูงของตัวอักขระไปยังด้านล่างของบัพแม่ที่อยู่ระดับก่อนหน้า

15) ระยะที่ได้ถูกใช้วาดไปแล้วในแนวนอนรวมทั้งระยะที่จะเว้นระหว่างบัพด้วย คือ

x = x + nodew

16) ถ้าบัพที่ j ของระดับปัจจุบันที่ได้วาดออกไปแล้วเป็นบัพเลขคู่ แล้ว

xp = xp + nodewp ซึ่งเป็นการเตรียมตำแหน่งของบัพแม่บัพถัดไปที่อยู่ข้างบนที่จะลากเส้นกิ่งขึ้นไปหาสำหรับบัพที่กำลังจะวาดถัดไป

17) นับบัพถัดไปที่จะวาด j = j + 1

18) ไปทำข้อ 13)

19) เก็บค่าจำนวนบัพของระดับที่อยู่ก่อนหน้า nodep = avlnod

เก็บค่าตำแหน่งของบัพบนแถวลำดับที่ได้วาดออกไปแล้ว

entinx = entinx + aclnod

20) นับระดับถัดไปที่จะวาด i = i + 1

21) ไปทำข้อ 5)

22) จบการทำงาน

10.<u>การแสดงการเปลี่ยนแปลงของแต่ละบัพเมื่อมีการจัดโครงสร้างข้อมูลให้เป็นฮีป</u>

การจัดโครงสร้างข้อมูลให้เป็นฮีป เป็นการเปรียบเทียบค่าระหว่างบัพต่าง ๆ แล้วทำการสลับค่าระหว่างบัพ ซึ่งค่าของบัพจะถูกเปลี่ยนไปเมื่อมีการสลับค่าเกิดขึ้น การแสดงการเปลี่ยนแปลงของบัพข้อมูลเนื่องจากการทำงานในแต่ละขั้นตอนของการจัดโครงสร้างข้อมูลให้เป็นฮีป จึงเป็นการแสดงค่าเดิมของบัพก่อนที่จะถูกแทนด้วยค่าใหม่ และตามด้วยการแสดงค่าใหม่ของบัพออกมาในตอนท้าย โดยมีขั้นตอนการทำงานดังนี้

1) หาตำแหน่งของบัพที่วาดบนตัวควบคุม ซึ่งเป็นบัพที่มีการเปลี่ยนแปลงของข้อมูล โดยมีวิธีการคำนวนหาตำแหน่งคล้ายกับการคำนวนหาตำแหน่งของแต่ละบัพในการแสดงโครงสร้างข้อมูลออกมาที่ได้กล่าวถึงในหัวข้อก่อน

2) แสดงค่าข้อมูลเก่าของบัพโดยใช้สีพื้นหลัง (Background) สลับกับสีของตัวอักขระ (Foreground) ณ ตำแหน่งที่คำนวนได้จากข้อ 1) เป็นระยะนานเท่ากับค่าคุณสมบัติ NodeTime / 2

3) แสดงค่าข้อมูลใหม่ของบัพโดยใช้สีพื้นหลัง (Background) สลับกับสีของตัวอักขระ (Foreground) ณ ตำแหน่งที่คำนวนได้จากข้อ 1) เป็นระยะนานเท่ากับค่าคุณสมบัติ NodeTime / 4

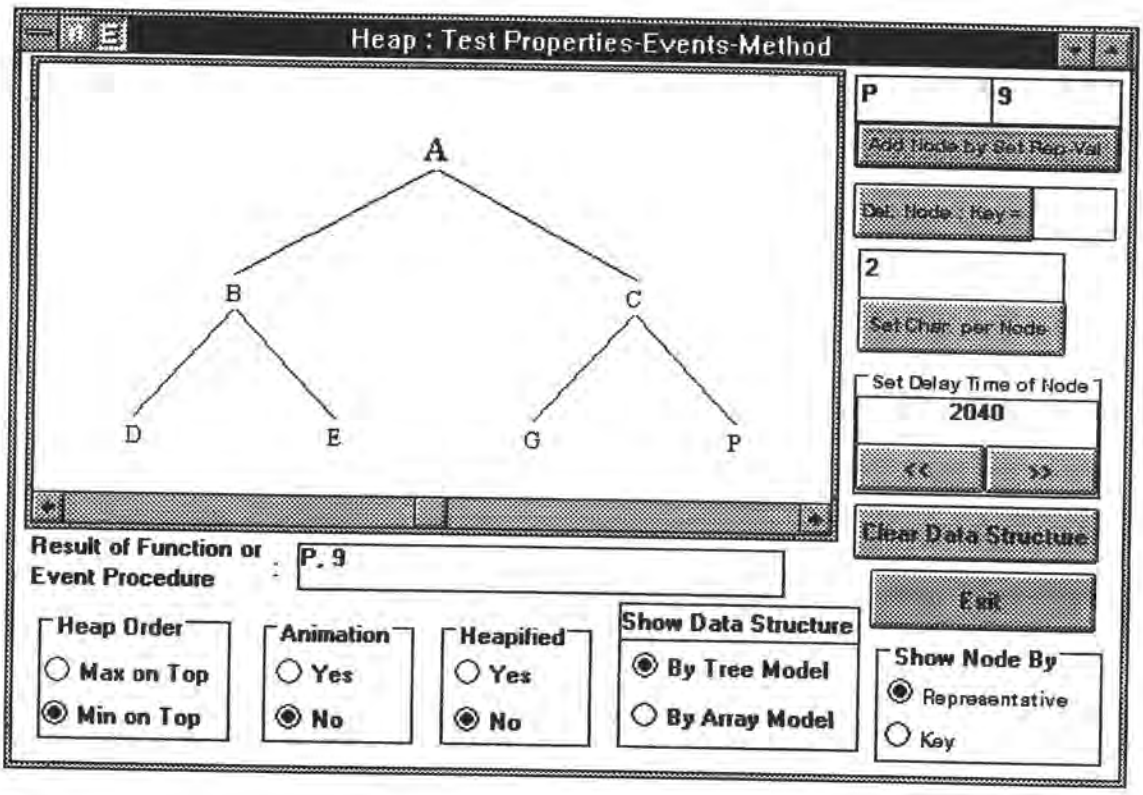
4) จบการทำงาน

การทดสอบตัวควบคุม

การทดสอบตัวควบคุม จะมีการสร้างโปรแกรมวิชวลเบสิคขึ้นมา เพื่อทดสอบการทำงานของตัวควบคุมในด้านการใช้คุณสมบัติที่สร้างขึ้นใหม่ การทำงานของเหตุการณ์ที่สร้างขึ้นใหม่ รวมทั้งการใช้วิธีและฟังก์ชันของตัวควบคุม และในโปรแกรมสุดท้ายจะแสดงตัวอย่างการประยุกต์ใช้งานตัวควบคุมโดยนำมาใช้เป็นแถวคอยบุริมภาพ (Priority Queue)

1. การทดสอบตัวควบคุมในด้านคุณสมบัติ เหตุการณ์ และ วิธี

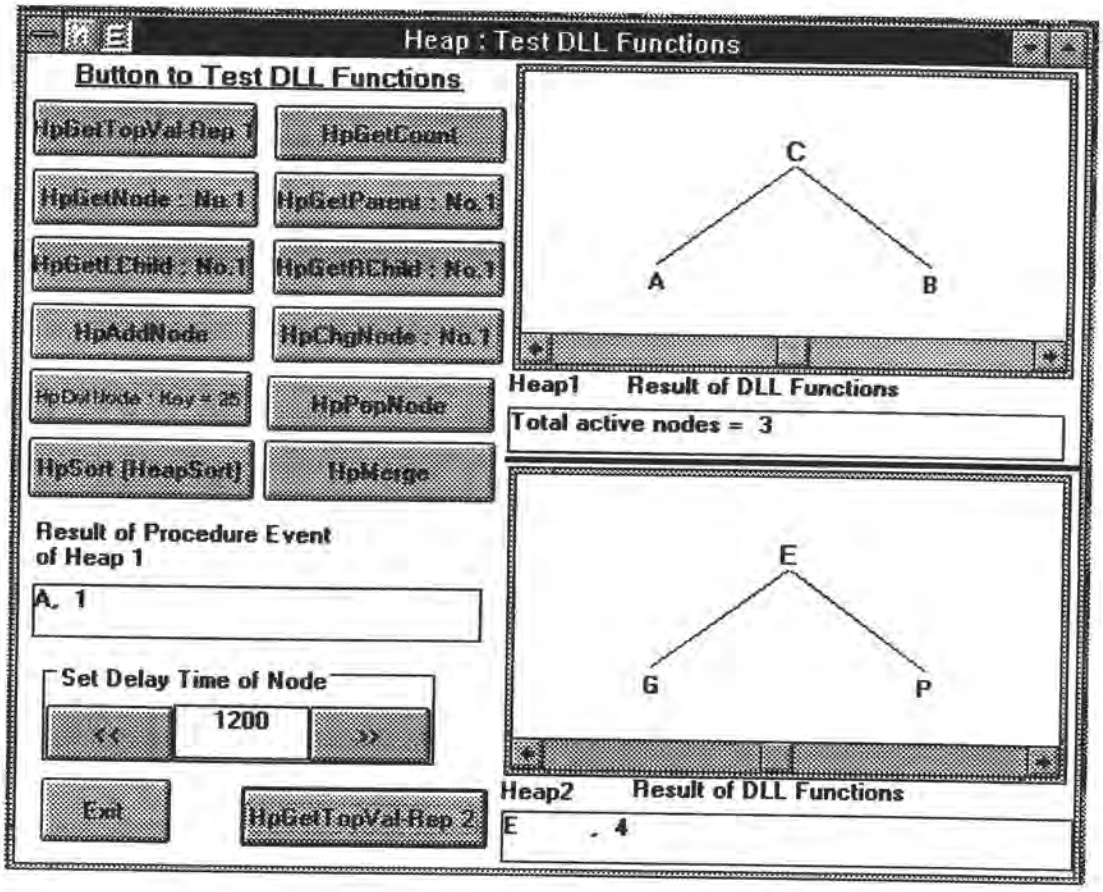
โปรแกรมวิชวลเบสิคที่สร้างขึ้นมาสำหรับการทดสอบในข้อนี้เป็นโปรแกรมชื่อ HPSM1 ใช้ในการทดสอบการทำงานของคุณสมบัติและเหตุการณ์ที่สร้างขึ้นใหม่ของตัวควบคุมนี้ รวมทั้งทดสอบการทำงานของวิธี Clear ลักษณะการใช้งานของโปรแกรมนี้แสดงอยู่ในรูปที่ 5.2 ซึ่งจะเป็นการทดลองเลือกกำหนดค่าคุณสมบัติต่าง ๆของตัวควบคุม และผลการทำงานของฟังก์ชันและกระบวนงานเหตุการณ์ของตัวควบคุมนี้จะแสดงออกมาไว้ในแถบข้อความที่อยู่ข้างใต้ตัวควบคุม



รูปที่ 5..2 โปรแกรมวิชวลเบสิค HPSM1

2. <u>การทดสอบฟังก์ชันการใช้งานของตัวควบคุม</u>

โปรแกรมวิชวลเบสิคที่สร้างขึ้นมาเพื่อทดสอบฟังก์ชันการใช้งาน เป็นโปรแกรมชื่อ HPSM2 ซึ่งแสดงอยู่ในรูปที่ 5.3 โดยมีปุ่มควบคุมต่าง ๆ สำหรับทดสอบใช้งานฟังก์ชันต่าง ๆของตัวควบคุม ซึ่งแต่ละปุ่มจะมีชื่อของฟังก์ชันการทำงานที่จะให้ทดลองใช้งานแสดงอยู่ และผลการทำงานของฟังก์ชันและกระบวนงานเหตุการณ์จะแสดงออกมาอยู่ในแถบข้อความต่าง ๆของโปรแกรม

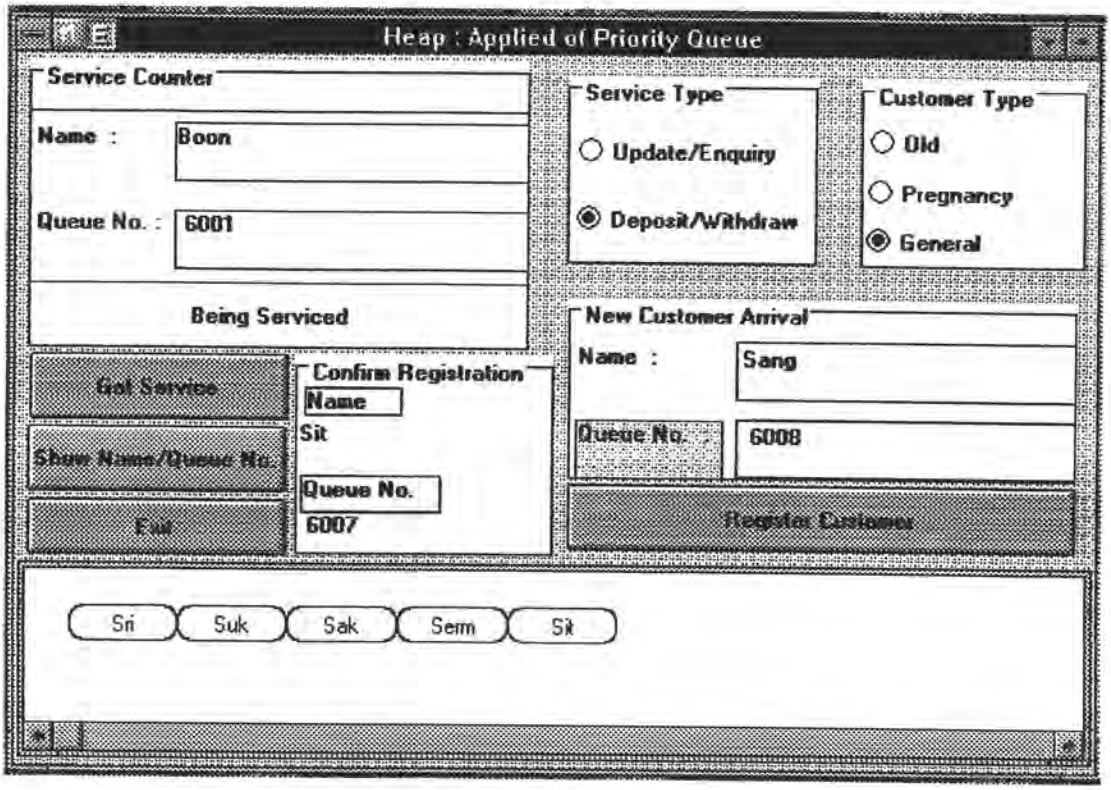


รูปที่ 5.3 โปรแกรมวิชวลเบสิค HPSM2

3. <u>การนำตัวควบคุมมาประยุกต์ใช้งาน</u>

ในโปรแกรมวิชวลเบสิค HPSM3 จะแสดงการนำตัวควบคุมมาประยุกต์ใช้เป็นแถวคอยบุริมภาพ (Priority Queue) แสดงดังในรูป 5.4 โปรแกรมนี้จะจำลองงานบริการลูกค้าของธนาคาร ซึ่งจะมีลูกค้าเข้ามาขอรับการบริการในปริมาณที่ไม่แน่นอน แต่ช่องการให้บริการของธนาคารมีอยู่จำกัด กรณีที่มีลูกค้าเข้ามาเป็นจำนวนมาก ลูกค้าก็จะต้องเข้าไปรอรับการบริการอยู่ในแถวคอย เพื่อรอให้เรียกเข้าไปรับบริการ ซึ่งสามารถกำหนดประเภทของการรับบริการ (Service Type) และประเภทของลูกค้า (Customer Type) เพื่อใช้ในการกำหนดเลขลำดับที่ของลูกค้าที่จะรอรับบริการ โดยประเภทของการรับบริการจะได้รับการพิจารณาก่อนประเภทของลูกค้า กล่าวคือ ลูกค้าที่มาใช้บริการประเภทปรับปรุงรายการ/สอบถามยอดบัญชี จะได้รับบริการก่อนประเภทฝาก/ถอน แต่ถ้าเป็นบริการประเภทเดียวกันแล้ว ลูกค้าผู้สูงอายุจะได้รับการบริการก่อน

เลขลำดับที่ของลูกค้าจะถูกสร้างขึ้นมาโดยโปรแกรม ซึ่งจะถูกสร้างขึ้นมาเมื่อคลิกที่แถบข้อความ Queue No. โดยใช้ประเภทของการรับบริการและประเภทของลูกค้าในการแยกกลุ่มของเลขลำดับที่ออกเป็น 6 กลุ่ม และในแต่ละกลุ่มจะมีเลขลำดับที่ย่อยของแต่ละกลุ่มอีกทีหนึ่ง



รูปที่ 5.4 โปรแกรมวิชวลเบสิค HPSM3

จากการใช้โปรแกรมวิชวลเบสิคทั้งสามโปรแกรมเพื่อทดสอบการทำงานของตัวควบคุม พบว่า ตัวควบคุมที่พัฒนาขึ้นมานี้ สามารถทำงานได้ถูกต้องตามที่ได้ออกแบบไว้ สามารถกำหนดค่าของคุณสมบัติเพื่อใช้ในการกำหนดการทำงานต่าง ๆ ของตัวควบคุมได้ตามต้องการ และเหตุการณ์ของตัวควบคุมพร้อมด้วยอาร์กิวเมนต์ของเหตุการณ์ได้เกิดขึ้นตามการทำงานที่ได้กำหนดไว้จริง ตลอดจนวิธีและฟังก์ชันสำหรับการใช้งานของตัวควบคุมก็ทำงานได้ถูกต้องตามที่ได้ออกแบบไว้เช่นกัน นอกจากนั้น ระบบความช่วยเหลือ (Help) ของตัวควบคุมนี้ก็สามารถนำข้อความคำอธิบายของหัวข้อที่ต้องการออกมาแสดงได้อย่างถูกต้อง