บทที่ 5

การพัฒนาและทดสอบคอนโทรล

ฮาร์ดแวร์และซอฟต์แวร์

1. ฮาร์ดแวร์

1.1 เครื่องคอมพิวเตอร์ที่เข้ากันได้กับเครื่องของ IBM ใช้หน่วยประมวลผลกลาง (CPU) 80286 หรือสูงกว่า

1.2 หน่วยความจำแรม (RAM) อย่างน้อย 1 เมกกะไบต์

1.3 ใช้ฮาร์ดดิสค์ความจุอย่างน้อย 80 เมกกะไบต์

1.4 ใช้เมาส์ ของบริษัท ไมโครซอฟต์ หรือเมาส์อื่นที่เข้ากันได้ (Compatible)

1.5 จอภาพแสดงผลระดับ Enhanced Graphics Adapter (EGA) ขึ้นไป

2. ซอฟต์แวร์

2.1 ระบบปฏิบัติการดอส รุ่น 3.1 หรือสูงกว่า

2.2 ไมโครซอฟต์วินโดวส์ (Thai Edition) รุ่น 3.1

2.3 ชุดพัฒนาโปรแกรม Software Development Kit สำหรับวินโดวส์ รุ่น 3.1

2.4 ภาษาโปรแกรมที่สามารถสร้าง DLL ของวินโดวส์ได้ ซึ่งผู้เขียนใช้ Microsoft C/C++ รุ่น 7.0

2.5 ภาษาโปรแกรมวิชวลเบสิกสำหรับวินโดวส์ รุ่น 3.0 ซึ่งเมื่อทำการติดตั้งวิชวลเบสิกแล้วจะมีสารบบ (DIRECTORY) หนึ่งคือ CDK (Control Development Kit) ซึ่งเป็นชุดพัฒนาคอนโทรลที่มีตัวอย่างพื้นฐานในการสร้างคอนโทรลอยู่ในสารบบย่อยต่าง ๆ ต่อไปนี้

CIRC1 เป็นตัวอย่างง่าย ๆ ในการสร้างคอนโทรล ซึ่งเราสามารถใช้เป็นโครงร่างในการสร้างคอนโทรลอื่น ๆ ได้

CIRC2 พัฒนาจาก CIRC1 เพิ่มการวาดวงกลม , วงรี ระบายสีฉากหลัง และจัดการกับเหตุการณ์ที่เกิดขึ้นเมื่อมีการคลิกเมาส์ภายในวงกลม

CIRC3 พัฒนาจาก CIRC2 มีการเขียนอักขระภายในวงกลม จัดทำ Context-sensitive Help แสดงการใช้หน้าต่างแบบผุดขึ้น (POP-UP WINDOWS) ในการกำหนดค่าคุณสมบัติ

CNTR เป็นตัวอย่างของคอนโทรลตัวนับ (COUNTER CONTROL) ซึ่งแสดงการนับในลักษณะของ Odometer

PAL ตัวอย่างของคอนโทรลที่ใช้สี 256 สี

PIX สร้างคอนโทรลที่เป็นรูปภาพ และมีการจัดการกับเหตุการณ์คลิก (CLICK EVENT)

PUSH สร้างคอนโทรลที่เป็นคลาสย่อยของคลาส PUSH BUTTON ของวินโดวส์

XLIST สร้างคอนโทรลที่เป็นคลาสย่อยของคลาส LIST BOX ของวินโดวส์ และแสดงการใช้คุณสมบัติแบบช่องลำดับ (ARRAY PROPERTIES)

การสร้างคอนโทรลขึ้นใหม่นั้นเป็นการสร้าง DLL ให้กับวิชวลเบสิก ดังนั้นจึงจำเป็นต้องมีการใช้ชนิดของตัวแปรของวิชวลเบสิกที่นอกเหนือไปจากชนิดของตัวแปรในวินโดวส์ มีการใช้ค่าคงที่ต่าง ๆ เพิ่มขึ้น และที่สำคัญคือ ต้องมีการใช้ API ของวิชวลเบสิกด้วย วิชวลเบสิกจึงมีคลังชุดคำสั่ง (Library) และเฮดเดอร์ไฟล์มาให้กับผู้ที่ต้องการสร้างคอนโทรลขึ้นใหม่

VBAPI.LIB เป็นคลังชุดคำสั่งที่มีฟังก์ชั่นทั้งหมดที่เป็น API ของวิชวลเบสิก โปรแกรมเชื่อมโยง (Linker) ของภาษาโปรแกรมที่ใช้สร้างคอนโทรล จะต้องเชื่อมโยงคลังชุดคำสั่งนี้เข้ากับแฟ้มคอนโทรล (.VBX) ด้วย

VBAPI.H เป็นเฮดเดอร์ไฟล์ที่กำหนดรูปแบบในการติดต่อเรียกใช้ฟังก์ชั่นทั้งหมดตลอดจนโครงสร้างและค่าคงที่ต่าง ๆ ใน VBAPI.LIB

<u>การพัฒนาคอนโทรล</u>

Microsoft Corp. (1993) ได้อธิบายถึงวิธีการและหลักการพัฒนาคอนโทรลไว้มากพอสมควร ผู้เขียนได้ทำการศึกษาถึงวิธีการและหลักการดังกล่าว พร้อมทั้งทดลองเขียนโปรแกรมสร้างคอนโทรลขึ้นด้วย จากนั้นก็ได้นำความรู้ที่ได้มาสร้างคอนโทรลโครงสร้างข้อมูลทั้งสามแบบ และได้สรุปวิธีการและหลักการพัฒนาคอนโทรลไว้ดังนี้

1. <u>จุดมุ่งหมายของการเขียนโปรแกรมเพื่อสร้างคอนโทรลขึ้นใหม่</u>

- กำหนดว่าจะแสดงและพิมพ์คอนโทรลออกมาในลักษณะใด
- กำหนดคุณสมบัติทั้งหมดของคอนโทรล กำหนดคุณสมบัติที่จะแสดงในหน้าต่างคุณสมบัติ กำหนดว่าจะเกิดอะไรขึ้น เมื่อผู้พัฒนาโปรแกรมด้วยวิชวลเบสิกเปลี่ยนแปลงค่าคุณสมบัติเหล่านั้น
- กำหนดเหตุการณ์ทั้งหมดของคอนโทรล โดยเหตุการณ์ต่าง ๆ จะแสดงอยู่ในหน้าต่างสำหรับเขียนโปรแกรมของวิชวลเบสิก

- กำหนดวิธีการทำงานของวิธีมาตรฐาน (STANDARD METHOD) ที่นำมาใช้กับคอนโทรลที่สร้างขึ้นใหม่ **แต่ไม่สามารถเพิ่มวิธีขึ้นมาใหม่ได้**

และตัวคอนโทรลเองจะมีการติดต่อกับทั้งวินโดวส์ และตัววิชวลเบสิกด้วยข้อความต่าง ๆ เหมือนกับโปรแกรมประยุกต์สำหรับวินโดวส์อื่น ๆ ทั่วไป แฟ้มคอนโทรลของวิชวลเบสิก จะมีสกุลเป็น .VBX เมื่อจะใช้แฟ้มนี้ ก็ทำได้โดยการเพิ่มเข้าไปในหน้าต่างโปรเจ็กต์ (โดยใช้รายการเลือก File ด้วยรายการเลือกย่อย Add File...)

2. แฟ้มข้อมูลที่ใช้แปล (COMPILE) แล้วเชื่อมโยง (LINK) กันเป็นแฟ้ม .VBX ของคอนโทรล

2.1 แฟ้มสกุล .H ซึ่งจะกำหนด

2.1.1 ค่าคงที่และรูปแบบที่ใช้ในการติดต่อกับฟังก์ชั่นต่าง ๆ (PROTOTYPE) ภายในคอนโทรล

2.1.2 ต้นแบบของคอนโทรล (CONTROL MODEL)

2.1.3 ตารางข้อมูลคุณสมบัติ (PROPERTY INFORMATION TABLE)

2.1.4 ตารางข้อมูลเหตุการณ์ (EVENT INFORMATION TABLE)

2.2 แฟ้มสกุล .C กำหนดกระบวนงานของคอนโทรล (CONTROL PROCEDURE) และชุดคำสั่งของฟังก์ชั่นภายในคอนโทรล

ซึ่งทั้งแฟ้มสกุล .H และสกุล .C ได้กล่าวถึงบ้างแล้วในบทที่ 3 หัวข้อคอนโทรลที่สร้างขึ้นใหม่

2.3 แฟ้มสกุล .RC เป็นแฟ้มที่เก็บรูปแบบของทรัพยากร (RESOURCE) ต่าง ๆ ที่ใช้ในการสร้างโปรแกรมประยุกต์ (ในที่นี้คือคอนโทรล) เช่น รายการเลือก , กรอบข้อความ , รูปภาพบิตแมพ , รูปภาพสัญรูป (ICON)

2.4 แฟ้มสกุล .DEF หน้าที่ของแฟ้มนี้ คือ ช่วยโปรแกรมเชื่อมโยงสร้างแฟ้ม .EXE , DLL หรืออื่น ๆ โดยแฟ้มสกุล .DEF นี้จะบอกลักษณะของ Code Segment และ Data Segment ขนาดของ Heap , Stack และอื่น ๆ ของโปรแกรม

2.5 แฟ้มสกุล .MAK แฟ้มนี้จะช่วยโปรแกรม NMAKE ในการเรียกใช้งานโปรแกรมตัวแปล , โปรแกรมเชื่อมโยง , โปรแกรมตัวแปลทรัพยากร (Resource Compiler) ตลอดจนตรวจสอบและดำเนินการขั้นตอนต่าง ๆ เพื่อสร้างโปรแกรมประยุกต์หรือ DLL ขึ้นมา

3. การกำหนดต้นแบบของคอนโทรล (BISEARCH.H บรรทัดที่ 178 - 195)

แต่ละฟิลด์ หมายถึง

1. usVersion เป็นเลขรุ่นของวิชวลเบสิก ที่เราใช้พัฒนาคอนโทรล ในที่นี้คือ ค่า VB_VERSION ตามที่วิชวลเบสิกกำหนดไว้ใน VBAPI.H

2. fl ตัวบ่งชี้ของต้นแบบ (MODEL FLAGS) ให้เลือกใช้ตามตารางที่ 5.1

3. pctlproc ที่อยู่ในหน่วยความจำของกระบวนงานของคอนโทรล

4. fsClassStyle ลักษณะคลาสของหน้าต่าง (WINDOW CLASS STYLE) ของคอนโทรล ค่าเหล่านี้จะนำหน้าด้วยตัว CS ซึ่งถูกกำหนดไว้ใน WINDOWS.H

ตารางที่ 5.1 ตัวบ่งชี้ของต้นแบบ

ตัวบ่งชี้	ความหมาย
MODEL_fArrows	ให้ส่งข้อความคีย์บอร์ดเมื่อมีการกดปุ่มลูกศรต่าง ๆ ถ้าไม่ได้ใช้ตัวบ่งชี้นี้ การใช้ปุ่มลูกศรจะเป็นการเปลี่ยนโฟกัสระหว่างคอนโทรล
MODEL_fChildrenOk	ให้ผู้พัฒนาโปรแกรมประยุกต์ด้วยวิชวลเบสิก สามารถสร้างคอนโทรลอื่นขึ้นภายในคอนโทรลที่ใช้ตัวบ่งชี้นี้ได้ เช่น รูปภาพต่าง ๆ
MODEL_fDesInteract	ส่งข้อความที่เป็นของปุ่มเมาส์ด้านขวา ได้แก่ WM_RBUTTONDOWN , WM_RBUTTONDBLCLK และ WM_RBUTTONUP ไปยังคอนโทรลในช่วงเวลาการออกแบบ
MODEL_fFocusOk	สามารถกำหนดโฟกัสให้กับคอนโทรลในช่วงเวลาดำเนินงานได้
MODEL_fGraphical	กำหนดให้เป็นคอนโทรลทางด้านรูปภาพ
MODEL_fInitMsg	ให้วิชวลเบสิกส่งข้อความ VBM_INITIALIZE ไปยังคอนโทรล
MODEL_fInvisAtRun	กำหนดให้ไม่แสดงตัวคอนโทรลในช่วงเวลาดำเนินงาน
MODEL_fLoadMsg	ให้ส่งข้อความ VBM_CREATED และ VBM_LOADED ไปยังคอนโทรล
MODEL_fMnemonic	ให้ตอบสนองต่อปุ่มสัญลักษณ์ช่วยจำ (Mnemonic Key) ของคอนโทรล โดยกำหนดโฟกัสให้กับคอนโทรล

5. fsWndStyle ลักษณะของหน้าต่างโดยปริยายของคอนโทรล

6. cbCtlExtra ขนาดของโครงสร้างของผู้เขียนโปรแกรม หากไม่มีโครงสร้างนี้ เพราะมีแต่คุณสมบัติมาตรฐานเท่านั้น ก็ให้กำหนดฟิลด์นี้เป็น 0

7. idBmpPalette หมายเลข (IDs) เริ่มต้นของภาพบิตแมพแรกที่ใช้เป็นสัญรูปแทนคอนโทรลในหน้าต่างกล่องเครื่องมือ

8. npszDefCtlName ชื่อโดยปริยายของคอนโทรล ซึ่งจะใช้แทนคอนโทรลในการเขียนโปรแกรม (หรือ คุณสมบัติ Name นั่นเอง)

9. npszClassName ชื่อของคลาสที่จะใช้ในวิชวลเบสิก ที่จะบอกชนิดของคอนโทรลในชุดคำสั่ง If.....TypeOf ของวิชวลเบสิกได้

10. npszParentClassName ชื่อของคลาสที่ให้กำเนิด (WINDOWS PARENT CLASS) คอนโทรล หากไม่มี ให้กำหนดค่าฟิลด์นี้เป็น NULL

11. nppproplist ตัวชี้ขนาด 16 บิตไปยังยังตารางข้อมูลคุณสมบัติ

12. npeventlist ตัวชี้ขนาด 16 บิต ไปยังตารางข้อมูลเหตุการณ์

13. nDefProp ดัชนีของคุณสมบัติ ที่จะใช้เป็นคุณสมบัติโดยปริยายในหน้าต่างคุณสมบัติ

14. nDefEvent ดัชนีของเหตุการณ์ที่จะใช้เป็นเหตุการณ์โดยปริยายในหน้าต่างสำหรับเขียนโปรแกรม

15. nValueProp ดัชนีของคุณสมบัติที่จะเป็นคุณสมบัติโดยปริยายของคอนโทรล เช่น ถ้ากำหนดคุณสมบัติ Name เป็น BiSearch1 แล้วกำหนดให้คุณสมบัติ InsertedKey เป็นคุณสมบัติโดยปริยายของคอนโทรล ดังนั้นเมื่อเขียน

```
BiSearch1 = 12  ก็จะมีความหมายเหมือนกับ
BiSearch1.InsertedKey = 12
```

4. <u>การกำหนดคุณสมบัติให้กับคอนโทรล</u>

ทำได้โดย

4.1 กำหนดแต่ละคุณสมบัติไว้ในตารางคุณสมบัติ โดยใน BISEARCH. H บรรทัดที่ 117 เป็นการให้ที่อยู่ของโครงสร้าง PROPINFO ซึ่งได้กำหนดไว้ใน BISEARCH.H บรรทัดที่ 79 - 88 ก่อนกำหนดตารางข้อมูลคุณสมบัติใน BISEARCH.H บรรทัดที่ 109 - 126

เมื่อเราเพิ่มคุณสมบัติในตารางข้อมูลคุณสมบัติ ก็ต้องมีการจองเนื้อที่เพื่อเก็บค่าของคุณสมบัตินั้นด้วย เนื้อที่นั้นเรียกว่า โครงสร้างของผู้เขียนโปรแกรม (BISEARCH.H บรรทัดที่ 59 - 68) และหากเราจัดการกับข้อความที่ส่งมาจากวิชวลเบสิก VBM_SETPROPERTY (ซึ่งข้อความนี้จะถูกส่งมายังกระบวนงานของคอนโทรลเมื่อมีการเปลี่ยนแปลงค่าคุณสมบัติ) เราก็สามารถเปลี่ยนแปลงลักษณะคอนโทรลบนจอภาพได้

ก่อนที่เราจะอ้างถึงโครงสร้างของผู้เขียนโปรแกรมนี้ได้ เราจะต้องได้ตัวชี้ไปยังที่อยู่ในหน่วยความจำของโครงสร้างนี้เสียก่อน ซึ่งในกรณีของคอนโทรล BISEARCH เราต้องได้ตัวชี้ไปยังที่อยู่ของโครงสร้าง BISEARCH โดยการใช้ฟังก์ชั่น API ของวิชวลเบสิก VBDerefControl (ดังเช่นใน BISEARCH.C บรรทัดที่ 28 ใช้มาโครที่ได้กำหนดไว้ใน BISEARCH.H บรรทัดที่ 72 เพื่อให้การใช้ฟังก์ชั่น VBDerefControl สะดวกขึ้น)

หลังจากมีการเรียกใช้ฟังก์ชั่น API ของวิชวลเบสิกแล้ว จะทำให้ที่อยู่ของโครงสร้างของ

ผู้เขียนโปรแกรมเปลี่ยนแปลงไป จึงต้องใช้ฟังก์ชั่น VBDerefControl อีกเพื่อให้ตัวชี้นั้นชี้ไปยังที่อยู่ใหม่ของโครงสร้างด้วย ฟังก์ชั่น VBDerefControl จะใช้กับแฮนเดิลของคอนโทรล (hctl) เพราะค่าแฮนเดิลของคอนโทรลไม่เปลี่ยนแปลง

แต่ละฟิลด์ของโครงสร้าง PROPINFO มีความหมายดังนี้

1. npszName ชื่อของคุณสมบัติที่จะใช้ในหน้าต่างคุณสมบัติ และในการเขียนโปรแกรม

2. fl ตัวบ่งชี้ (Flag) ของคุณสมบัติ ซึ่งมีได้หลายตัว ตัวแรกใช้กำหนดชนิดของคุณสมบัติ (PROPERTY DATA TYPE FLAGS) จาก BISEARCH.H บรรทัดที่ 103 กำหนดให้คุณสมบัตินี้เป็นชนิดอักขระปิดท้ายด้วย " \0 " (Null-terminated String) ตัวบ่งชี้อื่นที่สำคัญที่ใช้กำหนดชนิดของคุณสมบัติเป็นดังตารางที่ 5.2

ตารางที่ 5.2 ตัวบ่งชี้กำหนดชนิดของคุณสมบัติ

ตัวบ่งชี้	ความหมาย
DT_BOOL	กำหนดชนิดเป็นตรรกศาสตร์ (BOOLEAN) ค่าที่ไม่เท่ากับ 0 (จริง) จะถูกเปลี่ยนเป็น -1 เสมอสำหรับคุณสมบัติชนิดนี้
DT_COLOR	เป็นจำนวนเต็ม 16 บิตที่แทนสีแต่ละสีที่ใช้ในวิชวลเบสิก โดยในหน้าต่างคุณสมบัติจะแสดงค่าเป็นเลขฐาน 16
DT_ENUM	เป็นจำนวนเต็มขนาด 8 บิต (0 - 255) แบบไม่คิดเครื่องหมาย
DT_HSZ	เป็นแฮนเดิลของอักขระที่ปิดท้ายด้วย " \0 " ซึ่งคล้ายกับอักขระ (String) ในภาษาซี
DT_HLSTR	เป็นแฮนเดิลของอักขระในภาษาวิชวลเบสิก ซึ่งสามารถมี " \0 " อยู่ในข้อความนี้ได้ (ไม่เหมือนกับ DT_HSZ) และ DT_HLSTR จะใช้เป็นอาร์กิวเมนต์ในกระบวนงานเหตุการณ์ได้
DT_LONG	จำนวนเต็มขนาด 32 บิต แบบคิดเครื่องหมาย
DT_PICTURE	เป็นโครงสร้างแบบรูปภาพ (PICTURE STRUCTURE) ที่ใช้แฮนเดิลในการอ้างถึงรูปภาพแต่ละรูป (HPIC)
DT_REAL	เป็นจำนวนจริงขนาด 4 ไบต์
DT_SHORT	จำนวนเต็ม 16 บิต แบบคิดเครื่องหมาย

ตัวบ่งชี้ที่ถัดจากตัวบ่งชี้ชนิดของคุณสมบัติใช้กำหนดว่า จะให้วิชวลเบสิกติดต่อกับคุณสมบัติในโครงสร้างของผู้เขียนโปรแกรมอย่างไร ได้แก่ PF_fGetMsg, PF_fGetData, PF_fSetMsg, PF_fSetData, PF_fSaveMsg, PF_fSaveData

สมมติกำหนดตัวบ่งชี้ ดังนี้

PF_fGetMsg | PF_SetMsg | PF_fSaveMsg

เป็นการกำหนดให้วิชวลเบสิกส่งข้อความ VBM_GETPROPERTY เมื่อต้องการอ่านค่าคุณสมบัติ , ส่งข้อความ VBM_SETPROPERTY เมื่อมีการกำหนดค่าคุณสมบัติ , ส่งข้อความ VBM_SAVEPROPERTY และ VBM_LOADPROPERTY เมื่อต้องการบันทึกค่าของคุณสมบัติลงดิสก์ หรือนำค่าของคุณสมบัติเข้าสู่หน่วยความจำ ข้อความทั้ง 4 อาจเกิดจากการที่ผู้พัฒนาโปรแกรมด้วยวิชวลเบสิกเขียนชุดคำสั่งที่ต้องการอ่านค่าคุณสมบัติ , เปลี่ยนแปลงค่าคุณสมบัติ , นำฟอร์มเข้าสู่หน่วยความจำ (ด้วยชุดคำสั่ง Load) หรือเปลี่ยนแปลงค่าคุณสมบัติจากในหน้าต่างคุณสมบัติ

ถ้ากำหนดตัวบ่งชี้เป็น

PF_fGetData | PF_SetData | PF_fSaveData

กำหนดให้วิชวลเบสิกทำการโอนถ่ายข้อมูลกับโครงสร้างของผู้เขียนโปรแกรมโดยตรง ไม่ต้องส่งข้อความเมื่อต้องการอ่านค่าคุณสมบัติ (GetData) กำหนดค่าคุณสมบัติ (SetData) และบันทึก (SaveData) หรือนำคุณสมบัติเข้าสู่หน่วยความจำ

ถ้ากำหนดตัวบ่งชี้เป็น

PF_fGetData | PF_SetMsg | PF_fSaveData

กำหนดให้วิชวลเบสิกส่งข้อความ VBM_SETPROPERTY เมื่อมีการกำหนดค่าคุณสมบัติ แต่ให้โอนถ่ายข้อมูลกับโครงสร้างของผู้เขียนโปรแกรมโดยตรงเมื่อต้องการอ่านค่าคุณสมบัติ (GetData) กับบันทึก (SaveData) หรือนำคุณสมบัติเข้าสู่หน่วยความจำ

นอกจากนี้ยังมีตัวบ่งชี้อื่น ๆ อีกดังตารางที่ 5.3

ตารางที่ 5.3 ตัวบ่งชี้อื่น ๆ ของคุณสมบัติ

ตัวบ่งชี้	ความหมาย
PF_fDefVal	ให้วิชวลเบสิกไม่ต้องบันทึกค่าของคุณสมบัติลงสู่ดิสค์ ถ้าค่าของคุณสมบัตินั้นเท่ากับฟิลด์ dataDefault ของโครงสร้าง PROPINFO ดังนั้นเมื่อมีการนำคอนโทรลเข้าสู่หน่วยความจำ ถ้าค่าของคุณสมบัติใดไม่ได้ถูกบันทึกไว้ วิชวลเบสิกก็จะกำหนดค่าคุณสมบัตินั้นตามฟิลด์ dataDefault
PF_fEditable	ให้ผู้พัฒนาโปรแกรมด้วยวิชวลเบสิก สามารถแก้ไขค่าของคุณสมบัติในกล่องกำหนดค่าได้โดยตรง ถึงแม้ว่าจะมีการใช้รายการเลือก หรือ หน้าต่างแบบผุดขึ้น (POP-UP WINDOW) ก็ตาม
PF_fGetHszMsg	ให้วิชวลเบสิกส่งข้อความ VBM_GETPROPERTYHSZ ไปยังคอนโทรล เมื่อต้องการจะแสดงค่าของคุณสมบัติในหน้าต่างคุณสมบัติ
PF_fLoadMsgOnly	ให้วิชวลเบสิกส่งข้อความ VBM_LOADPROPERTY เมื่อมีการนำฟอร์มจากแฟ้มเข้าสู่หน่วยความจำ ซึ่งข้อความนี้จะมีแฮนเดิลของตำแหน่งของแฟ้มนั้นด้วย ซึ่งกระบวนงานของคอนโทรลสามารถใช้แฮนเดิลนี้ ในการนำค่าของคุณสมบัติเข้าสู่หน่วยความจำ โดยใช้ API ของวิชวลเบสิก คือฟังก์ชั่น VBReadFormFile
PF_fLoadDataOnly	ให้วิชวลเบสิกนำค่าของคุณสมบัติจากฟอร์มที่บันทึกไว้เข้าสู่หน่วยความจำ และไม่ต้องบันทึกค่าของคุณสมบัติลงดิสค์อีก ตัวบ่งชี้นี้ช่วยให้คอนโทรลสามารถเข้ากันได้กับวิชวลเบสิกรุ่น 1.0
PF_fNoInitDef	ไม่ให้วิชวลเบสิกกำหนดค่าของคุณสมบัติเท่ากับค่าของฟิลด์ dataDefault ในระหว่างที่มีการนำคอนโทรลเข้าสู่หน่วยความจำ แต่ถ้าตัวบ่งชี้ PF_fDefVal ไม่ได้ถูกกำหนดขึ้น ตัวบ่งชี้ PF_fNoInitDef จะไม่มีผลใด ๆ
PF_fNoRuntimeR	กำหนดให้ผู้พัฒนาโปรแกรมด้วยวิชวลเบสิกไม่สามารถเขียนโปรแกรม เพื่ออ่านค่าคุณสมบัตินี้ได้ในช่วงเวลาดำเนินงาน แต่สามารถเปลี่ยนแปลงค่าได้ หากยังพยายามอ่านค่าของคุณสมบัตินี้ ก็จะเกิดข้อผิดพลาดในช่วงเวลาดำเนินงาน (Run Time Error) ขึ้น และถ้าใช้ตัวบ่งชี้นี้ร่วมกับตัวบ่งชี้ PF_fNoRuntimeW จะทำให้ไม่สามารถกล่าวถึงคุณสมบัตินั้นได้เลยในช่วงเวลาดำเนินงาน (เขียนโปรแกรมโดยใช้คุณสมบัตินั้นไม่ได้เลย) ซึ่งหากยังพยายามกล่าวถึงคุณสมบัตินั้นอยู่ เมื่อสั่งดำเนินงานโปรแกรม ก็ไม่สามารถทำงานได้ จะเกิดข้อผิดพลาดในการแปล (Compilation Error) ขึ้นเสียก่อน
PF_fNoRuntimeW	กำหนดให้คุณสมบัตินั้น ถูกอ่านค่าได้อย่างเดียว แก้ไขเปลี่ยนแปลงค่าไม่ได้ในช่วงเวลาดำเนินงาน

ตารางที่ 5.3 (ต่อ) ตัวบ่งชี้อื่น ๆ ของคุณสมบัติ

PF_fNoShow	จะไม่แสดงคุณสมบัตินั้นในหน้าต่างคุณสมบัติ
PF_fPreHwnd	ให้คุณสมบัติถูกนำเข้าสู่หน่วยความจำก่อนที่โครงสร้างของหน้าต่าง (WINDOW STRUCTURE) ของคอนโทรลจะถูกสร้างขึ้น ซึ่งคุณสมบัติที่จะเป็น PreHwnd ควรจะมีผลต่อลักษณะของหน้าต่าง (WINDOW STYLE) ของคอนโทรล
PF_fPropArray	กำหนดให้เป็นคุณสมบัติแบบช่องลำดับ และควรใช้ตัวบ่งชี้ PF_fNoShow ร่วมด้วย
PF_fSetCheck	ให้วิชวลเบสิกส่งข้อความ VBM_CHECKPROPERTY ก่อนที่จะเปลี่ยนแปลงค่าคุณสมบัติ ซึ่งทำให้กระบวนงานของคอนโทรลสามารถตรวจสอบความถูกต้องของค่าดังกล่าวก่อนได้ หากค่าดังกล่าวไม่ถูกต้อง กระบวนงานของคอนโทรลต้องคืนค่าที่ไม่เท่ากับ 0 ให้กับวิชวลเบสิก ตัววิชวลเบสิกจึงจะไม่เปลี่ยนแปลงค่านั้น
PF_fUpdateOnEdit	ในการใช้กล่องกำหนดค่าคุณสมบัตินั้น ค่าของคุณสมบัติจะไม่เปลี่ยนเป็นค่าใหม่ จนกว่าจะกด [Enter] แล้วเท่านั้น แต่ถ้ามีการใช้ตัวบ่งชี้นี้ ค่าของคุณสมบัติจะเปลี่ยนแปลงทันทีทุกครั้งที่มีการกดคีย์บอร์ด ควรใช้ตัวบ่งชี้นี้กับคุณสมบัติแบบอักขระ

3. offsetData ตำแหน่งที่ใช้เก็บค่าของคุณสมบัติในโครงสร้างของผู้เขียนโปรแกรมจากมาโครในแฟ้ม BISEARCH.H บรรทัดที่ 29 ทำให้การกำหนดตำแหน่งของคุณสมบัติใน BISEARCH.H บรรทัดที่ 104 สะดวกขึ้น

4. infodata จะกำหนดค่าของฟิลด์นี้เมื่อต้องการเก็บค่าของคุณสมบัติในลักษณะของบิตเพื่อประหยัดเนื้อที่ในโครงสร้างของผู้เขียนโปรแกรม ใช้กับ DT_BOOL , DT_SHORT และ DT_ENUM

5. dataDefault ค่าโดยปริยายของคุณสมบัติ

6. npszEnumList กำหนดค่านี้เมื่อเป็นคุณสมบัติชนิดจำนวนเต็ม 8 บิตไม่คิดเครื่องหมายเท่านั้น (DT_ENUM) โดยเป็นอักขระที่ใช้เป็นรายการเลือกของคุณสมบัติในหน้าต่างคุณสมบัติคั่นแต่ละรายการเลือกด้วย " \0 " และกำหนดค่านี้เป็น NULL ในกรณีที่ไม่มีรายการเลือกใด ๆ (BISEARCH.H บรรทัดที่ 86)

7. enmumMax ค่าสูงสุดของรายการเลือกในกรณีที่เป็นคุณสมบัติชนิดจำนวนเต็ม 8 บิตไม่คิดเครื่องหมาย (DT_ENUM)

4.2 กำหนดค่าคงที่เพื่อใช้เป็นดัชนีของแต่ละคุณสมบัติตามลำดับในตารางคุณสมบัติ เพื่อให้การเขียนโปรแกรมจัดการกับคุณสมบัติทำได้ง่ายขึ้น (BISEARCH.H บรรทัดที่ 43 - 54) และเราไม่จำเป็นต้องเรียงคุณสมบัติทั้งหมดตามลำดับอักษร ตัววิชวลเบสิกจะจัดเรียงให้เองอยู่ภายในหน้าต่างคุณสมบัติ

4.3 อาจเขียนโปรแกรมเพื่อจัดการกับข้อความที่มีผลต่อคุณสมบัติของคอนโทรล

5. การกำหนดเหตุการณ์ให้คอนโทรล

5.1 กำหนดเหตุการณ์ต่าง ๆ ไว้ในตารางข้อมูลเหตุการณ์ (BISEARCH.H บรรทัด ที่ 158 - 170) หากเหตุการณ์ต้องมีอาร์กิวเมนต์ด้วย จะต้องประกาศชนิดของอาร์กิวเมนต์ด้วยตัวบ่งชี้ชนิดของอาร์กิวเมนต์ (ARGUMENT TYPE FLAGS) ก่อนโครงสร้าง EVENTINFO (BISEARCH.H บรรทัดที่ 142) โดยเลือกใช้ตัวบ่งชี้ดังตารางที่ 5.4

ตารางที่ 5.4 ตัวบ่งชี้ชนิดของอาร์กิวเมนต์

ตัวบ่งชี้	ความหมาย
ET_I2	เลขจำนวนเต็ม 16 บิต แบบคิดเครื่องหมาย
ET_I4	เลขจำนวนเต็ม 32 บิต แบบคิดเครื่องหมาย
ET_R4	เลขจำนวนจริงขนาด 4 ไบต์
ET_R8	เลขจำนวนจริงขนาด 8 ไบต์
ET_CY	เลขจำนวนเงินขนาด 8 ไบต์
ET_HLSTR	อักขระ
ET_FS	ตัวแปรอักขระที่มีความยาวคงที่

ใน BISEARCH. H บรรทัดที่ 165 เป็นการให้ที่อยู่ของโครงสร้าง EVENTINFO ซึ่งได้กำหนดไว้ใน BISEARCH.H ในบรรทัดที่ 143 - 147 ก่อนกำหนดตารางข้อมูลเหตุการณ์

แต่ละฟิลด์ของโครงสร้าง EVENTINFO มีความหมายดังนี้

1. npszName ชื่อของเหตุการณ์ที่ใช้ในหน้าต่างสำหรับเขียนโปรแกรม

2. cParms จำนวนของอาร์กิวเมนต์ (ไม่รวมอาร์กิวเมนต์ Index ในกรณีที่เป็นคอนโทรลแบบแถวลำดับ)

3. cwParms จำนวนคำ (WORDS) ของรายชื่ออาร์กิวเมนต์ (Argument List) โดยจะมีตัวชี้ขนาด 2 ไบต์ไปยังแต่ละอาร์กิวเมนต์ ดังนั้นโดยส่วนใหญ่ ฟิลด์นี้จะมีค่าเป็น 2 เท่าของฟิลด์ cParms

4. npParmTypes ตัวชี้ไปยังข้อมูลแถวลำดับแบบไบต์ที่บอกถึงชนิดของแต่ละ

อาร์กิวเมนต์ตามลำดับในกระบวนงานเหตุการณ์

5. npszParmProf ข้อความที่ปิดด้วย " \0 " ที่จะใช้แสดงอาร์กิวเมนต์ของเหตุการณ์ในหน้าต่างสำหรับเขียนโปรแกรม โดยใช้เครื่องหมายจุลภาค (,) คั่นแต่ละอาร์กิวเมนต์ และไม่ต้องมีวงเล็บเหมือนในหน้าต่างสำหรับเขียนโปรแกรม และไม่ต้องเขียนอาร์กิวเมนต์ Index ด้วย เพราะวิชวลเบสิกจะจัดการอาร์กิวเมนต์นี้ให้คอนโทรลเอง

6. fl ฟิลด์นี้ของเหตุการณ์สามารถกำหนดค่าเป็น EF_fNoUnload หรือ 0 ถ้ากำหนดค่าเป็น EF_fNoUnload วิชวลเบสิกจะไม่นำฟอร์ม หรือคอนโทรลภายในฟอร์มออกจากหน่วยความจำในขณะที่เหตุการณ์นี้เกิดขึ้น

5.2 กำหนดค่าคงที่เพื่อใช้เป็นดัชนีของแต่ละเหตุการณ์ตามลำดับในตารางข้อมูลเหตุการณ์ (BISEARCH.H บรรทัดที่ 133 - 140)

5.3 อาจต้องเขียนโปรแกรม เพื่อกำหนดว่า เหตุการณ์ใดจะเกิดขึ้นเมื่อใด โดยใช้ฟังก์ชั่น VBFireEvent จาก BISEARCH.C บรรทัดที่ 214 ฟังก์ชั่น FireNodeIncrease จะให้เหตุการณ์ NodeIncrease เกิดขึ้น โดยใช้ฟังก์ชั่น VBFireEvent พร้อมทั้งผ่านค่าโครงสร้าง NODEINCREASEPARAMS เป็นอาร์กิวเมนต์ไปให้กับเหตุการณ์

ฟังก์ชั่น VBFireEvent จะไม่คืนกลับ (Return) จนกว่ากระบวนงานของวิชวลเบสิก ซึ่งกำหนดขึ้นโดยผู้พัฒนาโปรแกรมด้วยวิชวลเบสิก จะทำงานเสร็จลง

ถ้าเราใช้ Microsoft C Compiler รุ่น 6.0 หรือสูงกว่า สามารถใช้คำ enum ในการกำหนดค่าคงที่เพื่อเป็นดัชนีแทนได้ เช่น ใน BISEARCH.H บรรทัดที่ 133 - 140 สามารถใช้ enum แทนได้เป็น

```
enum event_indexes {
    IEVENT_BISEARCH_CLICK ,
    IEVENT_BISEARCH_DBCLICK ,
    .... ,
    .... ,
    IEVENT_BISEARCH_NODEINCREASE ,
    .... ,
    ....
};
```

ค่าแรกใน enum นั้นจะเท่ากับ 0 ค่าต่อไปเป็น 1, 2, 3, ... ไปเรื่อย ๆ

6. ฟังก์ชั่นที่สำคัญของ DLL สำหรับวิชวลเบสิก

6.1 ฟังก์ชั่นภายในแฟ้ม LIBENTRY.OBJ

เมื่อแฟ้มคอนโทรลที่สร้างขึ้นใหม่เป็น DLL จึงต้องมีการเริ่มต้น (Initialize) ที่แตกต่างไปจากโปรแกรมประยุกต์ทั่ว ๆ ไป ซึ่งโปรแกรมประยุกต์จะติดต่อกับ DLL ได้ด้วยฟังก์ชั่น Library Initialization ที่อยู่ภายในแฟ้ม LIBENTRY.OBJ ถ้าใช้ Microsoft C/C++ Compiler รุ่น 7.0 หรือ

สูงกว่า ฟังก์ชั่นนี้จะถูกเชื่อมโยงเข้ากับ .VBX โดยอัตโนมัติ แต่ถ้าใช้ Microsoft C Compiler รุ่น 6.0 เราจะต้องเชื่อมโยงฟังก์ชั่น Library Initialization เข้ากับ .VBX เอง (แฟ้ม LIBENTRY.OBJ นี้จะมีอยู่ในสารบบย่อยของ CDK)

6.2 ฟังก์ชั่นภายในแฟ้มคอนโทรล (สกุล .C)

แฟ้มคอนโทรลที่สร้างขึ้นใหม่จะต้องมีฟังก์ชั่นที่ใช้ในการเริ่มทำงาน 2 ฟังก์ชั่น และจบการทำงาน 2 ฟังก์ชั่น ได้แก่

6.2.1 ฟังก์ชั่น LibMain (BISEARCH.C บรรทัดที่ 113 - 124)

ถูกเรียกใช้เมื่อ DLL (หรือในที่นี้คือ แฟ้มสกุล .VBX) ถูกนำเข้าหน่วยความจำเป็นครั้งแรก ฟังก์ชั่นนี้จะกำหนดค่า hmodDLL เป็นตัวแปรแบบ Global (ซึ่งฟังก์ชั่น VBINITCC จะใช้ค่านี้ด้วย โดยจะกล่าวถึงต่อไป) ค่าของ hmodDLL เป็นค่าแฮนเดิลของแฟ้มคอนโทรล

6.2.2 ฟังก์ชั่น VBINITCC (BISEARCH.C บรรทัดที่ 130 - 139)

ฟังก์ชั่นนี้ทำหน้าที่ลงทะเบียนให้กับแต่ละคอนโทรลคลาสในแฟ้มคอนโทรลที่สร้างใหม่

ข้อแตกต่างระหว่าง LibMain และ VBINITCC ก็คือ ฟังก์ชั่น LibMain จะถูกเรียกใช้ครั้งเดียวในครั้งแรกที่มีการนำแฟ้มคอนโทรล (สกุล .VBX) เข้าสู่หน่วยความจำ แต่ฟังก์ชั่น VBINITCC จะถูกเรียกใช้ 1 ครั้ง เมื่อมี 1 โปรแกรมประยุกต์ใช้แฟ้มคอนโทรลที่เหมือนกันในเวลาเดียวกัน ในสารบบของ CIRC3 มีตัวอย่างการใช้ VBINITCC ในการลงทะเบียนต้นแบบของคอนโทรล และลงทะเบียนคอนโทรลคลาสที่ใช้ในช่วงเวลาการออกแบบของวิชวลเบสิก

3. ฟังก์ชั่น VBTERMCC (BISEARCH.C บรรทัดที่ 145 - 150)

ฟังก์ชั่นนี้จะมีหรือไม่มีก็ได้ ซึ่งจะใช้ในการยกเลิกการจองทรัพยากร และยกเลิกค่าต่าง ๆ ฟังก์ชั่นนี้คล้ายกับฟังก์ชั่น VBINITCC โดยจะถูกเรียกทุกครั้งที่โปรแกรมประยุกต์ยกเลิกการใช้งานคอนโทรลที่สร้างขึ้นใหม่

ในสารบบตัวอย่าง CIRC3 ใช้ฟังก์ชั่นนี้ในการยกเลิกการจองหน่วยความจำให้กับทรัพยากรของคอนโทรล CIRC3

4. ฟังก์ชั่น WEP (BISEARCH.C บรรทัดที่ 180 - 189)

ฟังก์ชั่นนี้ถูกเรียกใช้เมื่อ DLL (.VBX) ถูกนำออกจากหน่วยความจำ ฟังก์ชั่นนี้จะเพียงแต่คืนค่า 1 เท่านั้น เพื่อแสดงว่า DLL จบการทำงานโดยสมบูรณ์

สังเกตว่า ฟังก์ชั่น WEP จะถูกแปลและเชื่อมโยงเข้ากับคอนโทรล เมื่อเราใช้ Microsoft C Compiler รุ่น 6.0 เท่านั้น ในรุ่น 7.0 หรือสูงกว่า จะถูกเชื่อมโยงเข้ากับแฟ้ม VBX ให้โดยอัตโนมัติ

7. ฟังก์ชั่น VBDefControlProc

ในกระบวนงานของคอนโทรลมักต้องมีฟังก์ชั่นนี้อยู่ด้วยเสมอ เช่น

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

เมื่อกระบวนงานของคอนโทรลรับข้อความจากวินโดวส์หรือวิชวลเบสิกแล้ว หากไม่จัดการข้อความนั้นเอง ก็จะส่งต่อไปยังกระบวนงานควบคุมโดยปริยายของวิชวลเบสิก ซึ่งจะเห็นว่ากระบวนงานของคอนโทรลจะคืนค่าที่ได้จากกระบวนงานควบคุมโดยปริยาย

พารามิเตอร์ตัวแรกเป็นแฮนเดิลของคอนโทรล (hctl) ที่วิชวลเบสิกสร้างขึ้นแล้วส่งมาให้ ส่วนอีก 4 ตัวหลังก็เกิดจากข้อความที่วินโดวส์ส่งมายังวิชวลเบสิก (ดังที่ได้เคยกล่าวถึงไปแล้วในบทที่ 3 หัวข้อคอนโทรลที่สร้างขึ้นใหม่)

8. การสร้างสัญรูปแทนคอนโทรลในหน้าต่างกล่องเครื่องมือ

ในการสร้างสัญรูปเพื่อใช้แทนคอนโทรลในหน้าต่างกล่องเครื่องมือนั้น จะต้องสร้างขึ้นมาเพื่อใช้กับจอภาพแบบ VGA, Monochrome และ EGA (ส่วน CGA นั้น วินโดวส์จะทำงานเหมือนกับจอ Monochrome) ซึ่งจะต้องใช้แฟ้มบิตแมพ 4 แฟ้ม 2 แฟ้มสำหรับจอภาพแบบ VGA เพราะต้องใช้สำหรับแสดงตอนที่ไม่ได้มีการคลิกเมาส์ในสัญรูป 1 แฟ้ม และมีการคลิกเมาส์อีก 1 แฟ้ม ส่วนจอ EGA และ Monochrome นั้นใช้อย่างละ 1 แฟ้ม เพราะเมื่อมีการคลิกเมาส์ที่สัญรูป ก็จะใช้การสลับสีของแต่ละบิตในภาพแทน ส่วนจอภาพขนาดใหญ่จะใช้แฟ้ม 2 แฟ้มเดิมที่ใช้กับจอ VGA

แฟ้มบิตแมพทั้ง 4 แฟ้มเป็นทรัพยากรของคอนโทรล จึงต้องมีหมายเลข (IDs) แทนแต่ละแฟ้ม ดังตารางที่ 5.5

ตารางที่ 5.5 หมายเลข (IDs) ของแต่ละแฟ้มภาพบิตแมพ

แฟ้มบิตแมพสำหรับ	หมายเลข
จอ VGA ตอนที่ไม่ได้ถูกคลิกด้วยเมาส์	idBmpPalette เป็นเลขจำนวนเต็มที่กำหนดไว้ในต้นแบบของคอนโทรล (BISEARCH.H บรรทัดที่ 186)
จอ VGA ตอนที่ถูกคลิกด้วยเมาส์	idBmpPalette + 1
จอ Monochrome	idBmpPalette + 3
จอ EGA	idBmpPalette + 6

เราจะกำหนดเป็นเลขจำนวนเต็มใดก็ได้ให้กับภาพสำหรับจอ VGA ตอนที่ไม่ได้ถูกคลิกด้วยเมาส์ ส่วนภาพอื่นกำหนดโดยการเพิ่มค่าขึ้นอีก 1 , 3 และ 6 ตามลำดับ การกำหนดค่าซ้ำกับคอนโทรลอื่น จะไม่มีผลใด ๆ กับคอนโทรลทั้งสอง

เราอาจใช้โปรแกรมสำหรับวาดภาพ (PAINT) เช่น Microsoft Image Editor , Windows Paintbrush สร้างภาพบิตแมพทั้ง 4 ขึ้นมา โดยมีขนาดเป็น 28 x 28 ทุกภาพ ยกเว้นภาพสำหรับจอ EGA มีขนาด 28 x 22 หรืออาจเรียกแฟ้มบิตแมพจากสารบบตัวอย่างต่าง ๆ ของ CDK มาแก้ไขสร้างเป็นภาพตามที่เราต้องการ แล้วจึงบันทึกเป็นชื่อแฟ้มของเราเองก็ได้ หากใช้ภาพที่มีขนาดใหญ่กว่า เช่น ขนาด 30 x 35 วิชวลเบสิกจะตัดส่วนที่เกินทิ้ง แล้วแสดงเฉพาะส่วนที่อยู่ในขนาด 28 x 28 หรือ 28 x 22 เท่านั้น

9. การจัดการกับข้อความ WM_PAINT

ตัวกระบวนงานของคอนโทรลจะจัดการกับข้อความที่มาจากวินโดวส์และวิชวลเบสิก การทำงานที่มีผลต่อลักษณะของหน้าต่างต่าง ๆ และตัวคอนโทรล เช่น การวาดตัวคอนโทรล, การกดคีย์บอร์ด และเมาส์ จะมีข้อความมาจากวินโดวส์ ส่วนข้อความจากวิชวลเบสิก จะเป็นการทำงานกับคุณสมบัติและเหตุการณ์ของคอนโทรล

เราจะแสดงลักษณะของคอนโทรล โดยการจัดการในข้อความ WM_PAINT เหมือนกับโปรแกรมประยุกต์ทั่วไปสำหรับวินโดวส์ (BISEARCH.C บรรทัดที่ 80 - 91) เมื่อวิชวลเบสิกพิมพ์ฟอร์มออกทางเครื่องพิมพ์ (ด้วยคำสั่ง PrintForm ของวิชวลเบสิก) ในกรณีที่เป็นคอนโทรลที่เกี่ยวกับรูปภาพ (GRAPHICAL CONTROL) จะมีการสร้าง Printer - compatible Display Context ซึ่งเป็นแฮนเดิลในพารามิเตอร์ wp ของ ข้อความ WM_PAINT ดังนั้นเมื่อค่าของ wp ไม่เป็น 0 ก็จะใช้ค่าของ hDC ใน wp แต่ถ้าค่าของ wp เป็น 0 ก็จะใช้ฟังก์ชั่น BeginPaint แทน เพื่อหาค่าของ hDC

ส่วนฟังก์ชั่น PaintTree นั้นจะทำหน้าที่วาดตัวคอนโทรล โดยกำหนดการทำงานของฟังก์ชั่นนี้ไว้ใน BISEARCH.C บรรทัดที่ 192 - 197

10. การจัดการกับวิธี

เมื่อผู้พัฒนาโปรแกรมด้วยวิชวลเบสิกได้เขียนชุดคำสั่งที่เป็นวิธีของคอนโทรล แล้วสั่งดำเนินงานชุดคำสั่งนั้น วิชวลเบสิกจะส่งข้อความอันหนึ่งไปยังกระบวนงานของคอนโทรล คือ ข้อความ VBM_METHOD (BISEARCH.C บรรทัดที่ 67 - 79) โดยที่วิธี Drag, LinkSend, Move, Refresh และ Zorder วิชวลเบสิกจะจัดการให้กับคอนโทรลอยู่แล้ว และเราสามารถเพิ่มเติมการจัดการกับวิธีเหล่านี้ตามที่เราต้องการได้ ด้วยการจัดการกับข้อความ VBM_METHOD แต่ก็มีวิธีบางอย่าง ที่เราต้องจัดการเองทั้งหมด ได้แก่ วิธี AddItem, Clear และ RemoveItem (ดูรายละเอียดเพิ่มเติม ได้จาก Microsoft Corp. 1993.

Microsoft Visual Basic Programming System for Windows , Professional Feature Book 1 Version 3.0. U.S.A.: Microsoft Corp.)

11. กรอบของคอนโทรล

เราสามารถแสดงคอนโทรลโดยไม่มีกรอบล้อมรอบได้ ด้วยการเปลี่ยนแปลงฟิลด์ flWndStyle ในต้นแบบของคอนโทรล (BISEARCH.H บรรทัดที่ 184) เพราะฟิลด์นี้กำหนดลักษณะกรอบของคอนโทรล

```
WS_BORDER,        // Default Windows Style
```

หากไม่ต้องการให้มีกรอบรอบคอนโทรลก็ให้ใช้ค่า 0 แทน WS_BORDER

```
0,                // Default Windows Style
```

หรือไม่ต้องใช้ค่า 0 แทน WS_BORDER แต่ให้เพิ่มคุณสมบัติ BorderStyle เข้าไปในตารางข้อมูลคุณสมบัติด้วยค่าใดค่าหนึ่งต่อไปนี้

PPROPINFO_STD_BORDERSTYLEON หรือ

PPROPINFO_STD_BORDERSTYLEOFF

แต่ละค่าจะแทนคุณสมบัติ BorderStyle เหมือนกัน แต่จะให้ค่าเริ่มต้นของคุณสมบัติต่างกัน เราเลือกมาใช้เพียงค่าใดค่าหนึ่งเท่านั้น ใช้พร้อมกันทั้งสองค่าไม่ได้

12. การกำหนดค่าเริ่มต้นให้กับคุณสมบัติต่าง ๆ

ด้วยการจัดการกับข้อความ WM_NCCREATE และอาจใช้ฟังก์ชั่น VBSetControlProperty ด้วย

```
case WM_NCCREATE :
    lpbisearch->LevelDist = 20,
    lpbisearch->ScrollBar = 3,
    lpbisearch->BranchWidth = 1;
    VBSetControlPropety ( hctl, PPROPINFO_BISEARCH_LISTOFVALUES,
        (LONG)(LPSTR) "");
```

13. ขนาดโดยปริยายของคอนโทรล

เมื่อเราดับเบิ้ลคลิกที่สัญรูปของคอนโทรลใด ๆ ในหน้าต่างกล่องเครื่องมือ ก็จะได้คอนโทรลหนึ่งปรากฏที่กลางฟอร์ม ซึ่งขนาดของคอนโทรลขณะนี้ เรียกว่า ขนาดโดยปริยายของคอนโทรล เราสามารถกำหนดขนาดโดยปริยายของคอนโทรลได้โดยการจัดการกับข้อความจากวิชวลเบสิกอันหนึ่ง คือ ข้อความ VBM_GETDEFSIZE

```
case VBM_GETDEFSIZE :
    return  (MAKELONG( 200, 250 );
```

เป็นการคืนค่าของความกว้างและความสูงในหน่วยของ Pixel ให้กับวิชวลเบสิก โดย LOW WORD ของค่าที่คืนเป็น ความกว้าง HI WORD เป็นความสูง (ดูรายละเอียดเพิ่มเติม ได้จาก Microsoft Corp. 1993. Microsoft Visual Basic Programming System for Windows , Professional Feature Book 1 Version 3.0. U.S.A.: Microsoft Corp.)

14. การตรวจสอบสภาวะ (MODE) ของคอนโทรล

ฟังก์ชั่น API ของวิชวลเบสิก VBGetMode จะบอกแก่กระบวนงานของคอนโทรลได้ว่า ขณะนี้ตัวคอนโทรลอยู่ในสภาวะใด โดยฟังก์ชั่นนี้จะคืนค่าคงที่ MODE_DESIGN , MODE_RUN หรือ MODE_BREAK ให้กับกระบวนงานของคอนโทรล

```
if ( VBGetMode( ) = = MODE_DESIGN )
   MessageBox(NULL, (LPSTR) "We are in design-time." , "Binary Search Tree",
            MB_ICONQUESTION | MB_TASKMODAL | MB_OK);
```

15. การลดเวลาที่ใช้บันทึกคอนโทรลลงดิสค์และนำคอนโทรลเข้าสู่หน่วยความจำ

สำหรับโปรแกรมประยุกต์โดยวิชวลเบสิกแล้ว อาจต้องใช้เวลานานในการนำฟอร์มเข้าสู่หน่วยความจำ เพราะต้องนำแต่ละคอนโทรลเข้าสู่หน่วยความจำ แล้วทำการกำหนดค่าโดยปริยายให้กับแต่ละคุณสมบัติของคอนโทรลด้วย อย่างไรก็ตาม เราสามารถใช้ตัวบ่งชี้ PF_fDefVal และ PF_fNoInitDef ในการลดเวลาการนำฟอร์มเข้าสู่หน่วยความจำได้

เมื่อใช้ตัวบ่งชี้ PF_fDefVal กับคุณสมบัติในโครงสร้าง PROPINFO จะทำให้

1. เมื่อจะบันทึกค่าของคุณสมบัติ วิชวลเบสิกจะตรวจสอบดูว่า มีค่าเท่ากับฟิลด์ dataDefault (ในโครงสร้าง PROPINFO) หรือไม่ ถ้าเท่ากัน ก็จะไม่บันทึกคุณสมบัตินั้นลงดิสค์

2. ดัชนีของคุณสมบัติจะถูกบันทึกกับคอนโทรลด้วย ซึ่งถ้าคุณสมบัติใดไม่มีการบันทึก ดัชนีของคุณสมบัตินั้นก็จะไม่ถูกบันทึกด้วย ทำให้ประหยัดเนื้อที่ดิสค์ได้มาก ถ้ามีหลาย ๆ คุณสมบัติที่ไม่ต้องบันทึกลงดิสค์

3. จากนั้นเมื่อมีการนำคอนโทรลเข้าสู่หน่วยความจำ คุณสมบัติที่ไม่ได้ถูกบันทึก ก็จะถูกกำหนดค่าให้เท่ากับฟิลด์ dataDefault ถ้าไม่ได้มีการใช้ตัวบ่งชี้ PF_fNoInitDef ร่วมด้วย ซึ่งการกำหนดค่าคุณสมบัติด้วยวิธีนี้จะรวดเร็วกว่าการต้องบันทึกและอ่านจากดิสค์

```
PROPINFO Property_TreeLevel =
  {
  "TreeLevel",
  DT_SHORT | PF_fGetData | PF_fSetMsg | PF_fSaveData | PF_fDefVal
        | PF_fNoRuntimeW | PF_fNoShow,
  OFFSETIN(BISEARCH, TreeLevel),
  0, 0, NULL, 0
};
```

จากตัวอย่าง แสดงว่าต้องการให้เป็นคุณสมบัติแบบจำนวนเต็ม มีค่าโดยปริยายเป็น 0

16. การจัดการกับอักขระและลักษณะของอักขระ (FONTS)

16.1 การจัดการกับอักขระ

อักขระที่ใช้ในคอนโทรลนั้น มีลักษณะดังนี้

- อักขระไม่เหมือนกับตัวเลข และมีความยาวไม่คงที่ ดังนั้นเพื่อให้มีประสิทธิภาพที่ดี จึงต้องจองเนื้อที่ในหน่วยความจำแบบพลวัตเพื่อใช้เก็บค่าอักขระ และที่อยู่ของอักขระในหน่วยความจำนั้น สามารถเปลี่ยนแปลงได้

- เมื่อเราสร้างคอนโทรลขึ้นใหม่ ก็ต้องคุ้นเคยกับทั้งอักขระของภาษาซีที่เป็นอักขระปิดท้ายด้วย " \0 " และอักขระของภาษาเบสิคที่ไม่ใช่อักขระปิดท้ายด้วย " \0 " แต่วิชวลเบสิกจัดการโดยใช้ตัวบอกอักขระ (STRING DESCRIPTOR) แทน อักขระของภาษาเบสิกใช้เป็นอาร์กิวเมนต์ของกระบวนงานเหตุการณ์ด้วย

ในบางการทำงาน เราอาจใช้อักขระของภาษาซี แต่ถ้าต้องการใช้อักขระจำนวนมาก ก็ควรจะให้วิชวลเบสิกทำการจองหน่วยความจำให้กับอักขระแบบพลวัตให้

การเขียนโปรแกรมสำหรับวินโดวส์นั้น ถ้าข้อมูลชนิดใดสามารถเปลี่ยนแปลงที่อยู่ในหน่วยความจำได้ ก็จะใช้การอ้างถึงด้วยแฮนเดิล ซึ่งไม่เหมือนตัวชี้ตรงที่แฮนเดิลยังคงใช้อ้างถึงข้อมูลนั้นได้ ถึงแม้ว่าที่อยู่ของมันจะเปลี่ยนไปแล้วก็ตาม

วิชวลเบสิกสามารถจองเนื้อที่ในหน่วยความจำให้กับอักขระ 2 ชนิด ในลักษณะของแฮนเดิล ดังนี้

1 HSZ แฮนเดิลของอักขระปิดท้ายด้วย " \0 " ซึ่งคล้ายกับอักขระในภาษาซี แต่ต้องมีการอ้างถึง (Dereference) แฮนเดิลก่อน เพื่อให้ได้ที่อยู่ของอักขระ

2 HLSTR แฮนเดิลของอักขระภาษาเบสิค ที่ไม่ใช้ " \0 " ปิดท้าย แต่วิชวลเบสิกจะจัดการโดยใช้ตัวบอกอักขระแทน และอักขระนี้ใช้เป็นอาร์กิวเมนต์ของกระบวนงานเหตุการณ์ของวิชวลเบสิกด้วย

16.1.1 ฟังก์ชั่นสำหรับแฮนเดิลของอักขระที่ปิดท้ายด้วย " \0 " (HSZ)

VBCreateHsz จองเนื้อที่สำหรับอักขระในเซ็กเมนต์ที่กำหนด และคืนค่าแฮนเดิลของอักขระนั้น

VBDerefHsz คืนค่าตัวชี้ไปยังที่อยู่ของอักขระ HSZ

VBDestroyHsz ยกเลิกการจองเนื้อที่และแฮนเดิลของอักขระนั้น

ตัว " \0 " แรกจะเป็นตัวกำหนดขนาดของ HSZ และไม่มีฟังก์ชั่นที่ใช้เพิ่มขนาดของ HSZ ดังนั้น เมื่อต้องการให้ค่าใหม่กับคุณสมบัติแบบอักขระ HSZ จะต้องยกเลิกการจองเนื้อที่เดิมก่อน (Destroy) แล้วจองใหม่ (Create) แล้วจึงค่อยให้แฮนเดิลใหม่กับคุณสมบัตินั้น

นอกจากนี้ยังมีฟังก์ชั่น VBLockHsz และ VBUnlockHsz ที่คืนค่าของตัวชี้ไปยังที่อยู่ของอักขระ HSZ พร้อมทั้งป้องกันไม่ให้อักขระนั้นถูกเคลื่อนย้ายที่อยู่ด้วย

```
{
    LPSTR lpstr;
    HSZ hsz;

      lpstr = VBDerefHsz(lpbisearch->PreOrderList);
      hsz = VBCreateHsz((_segment)hctl, (LPSTR) lpstr);
      lpbisearch = LpbisearchDEREF(hctl);
      VBDestroyHsz(lpbisearch->ListOfValues);
      lpbisearch = LpbisearchDEREF(hctl);
      lpbisearch->ListOfValues = hsz;
      return 0;
}
```

จากตัวอย่างเป็นการให้แฮนเดิลของ lpbisearch->ListOfValues มีค่าเท่ากับแฮนเดิลของ lpbisearch->PreOrderList

16.1.2 ฟังก์ชั่นสำหรับแฮนเดิลของอักขระภาษาเบสิค (HLSTR)

VBCreateHlstr จองเนื้อที่สำหรับอักขระภาษาเบสิค แล้วคืนค่าแฮนเดิลของอักขระนั้น

VBDerefHlstr คืนค่าตัวชี้ไปยังที่อยู่ของอักขระ HLSTR

VBDestroyHlstr ยกเลิกการจองเนื้อที่และแฮนเดิลของอักขระนั้น

VBGetHlstrLen คืนค่าความยาวของอักขระ HLSTR

ตัวอย่างต่อไปนี้เป็นการสร้างอักขระภาษาเบสิค เพื่อใช้เป็นอาร์กิวเมนต์ของเหตุการณ์ NODEINCREASE

```
typedef struct tagNODEINCREASEPARAMS
  {
    HLSTR  increaseStr;
    int far *NumNode;
    LPVOID index;
  } NODEINCREASEPARAMS;

VOID NEAR FireNodeIncrease(HCTL hctl, int idx)
{
  NODEINCREASEPARAMS params;
  LPBISEARCH lpbisearch = LpbisearchDEREF(hctl);
  LPSTR lpstr;

  lpstr = VBDerefHsz(lpbisearch->ValueInNode);
  params.increaseStr = VBCreateHlstr(lpstr, lstrlen(lpstr));
  params.NumNode = &idx;
  VBFireEvent(hctl, IEVENT_BISEARCH_NODEINCREASE,
              &params);
  VBDestroyHlstr(params.increaseStr);

}   // FireNodeIncrease()
```

16.1.3 การสร้างคุณสมบัติแบบอักขระ

ต้องเพิ่มคุณสมบัติในโครงสร้างของผู้เขียนโปรแกรม

```
typedef struct tagBISEARCH
  {
    ....
    ....
    HSZ ValueInNode;
    ....
    ....
} BISEARCH;
```

ตัวแปร ValueInNode ใช้เก็บแฮนเดิล แต่ตัวอักขระจริง ๆ นั้น วิชวลเบสิกจะจัดให้อยู่ใน HEAP SEGMENT เอง

จากนั้น กำหนดโครงสร้าง PROPINFO ของคุณสมบัติ

```
PROPINFO Property_ValueInNode =
  {
  "InsertedKey",
  DT_HSZ | PF_fGetData | PF_fSetData | PF_fSetMsg | PF_fSaveMsg |
  PF_fDefVal,
  OFFSETIN(BISEARCH, ValueInNode),
  0, 0, NULL, 0
};
```

ใส่ที่อยู่ของโครงสร้าง PROPINFO ในตารางข้อมูลคุณสมบัติ

```
&Property_ValueInNode,
```

กำหนดดัชนีของคุณสมบัติ

```
#define IPROP_BISEARCH_VALUEINNODE      24
```

เราใช้ตัวบ่งชี้ทั้ง PF_fSetData และ PF_fGetData ในโครงสร้าง PROPINFO ดังนั้นเมื่อผู้พัฒนาโปรแกรมด้วยวิชวลเบสิก ต้องการเปลี่ยนแปลงหรืออ่านค่าคุณสมบัติ วิชวลเบสิกก็จะโอนถ่ายค่าข้อมูลโดยตรงกับโครงสร้างของผู้เขียนโปรแกรมเอง ซึ่งช่วยให้เราไม่ต้องเขียนโปรแกรม เพื่อจัดการตรงส่วนนี้ แต่เนื่องจากได้ใช้ตัวบ่งชี้ PF_fSetMsg ด้วย ดังนั้นเมื่อวิชวลเบสิกเปลี่ยนแปลงค่าในโครงสร้างของผู้เขียนโปรแกรมแล้ว ก็ต้องส่งข้อความ VBM_SETPROPERTY มายังกระบวนงานของคอนโทรลด้วย เราจึงสามารถวาดตัวคอนโทรลใหม่ได้จากข้อความ VBM_SETPROPERTY นี้เอง

```
case IPROP_BISEARCH_VALUEINNODE:
    ....
    ....
    InvalidateRect(hwnd, NULL, TRUE);
    ....
    return 0;
```

16.2 การจัดการกับลักษณะของอักขระ

เมื่อได้สร้างคุณสมบัติแบบอักขระแล้ว ก็ควรต้องมีคุณสมบัติของลักษณะอักขระด้วย เพื่อใช้กำหนดรูปร่าง , ขนาด และลักษณะอื่น ๆ ของอักขระที่ใช้ในคอนโทรล

เราใช้คุณสมบัติมาตรฐานเพื่อจัดการกับลักษณะต่าง ๆ ของอักขระ และต้องเพิ่มแฮนเดิลของลักษณะอักขระในโครงสร้างของผู้เขียนโปรแกรมด้วย เพื่อใช้เข้าถึงโครงสร้างของลักษณะอักขระ

```
typedef struct tagBISEARCH
  {
      ....
      ....
      HFONT hfont;
      ....
      ....
} BISEARCH;
```

แล้วเพิ่มคุณสมบัติมาตรฐานลักษณะอักขระเข้าไปในตารางข้อมูลคุณสมบัติ

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

ส่วนกระบวนงานของคอนโทรล จะต้องจัดการกับ

- ข้อความ WM_SETFONT และ WM_GETFONT โดยใช้ชุดคำสั่งไม่มากนัก
- ก่อนแสดงอักขระในคอนโทรล ให้เลือกลักษณะของอักขระก่อน เพื่อใช้กับเนื้อหาการแสดงผล (DISPLAY CONTEXT)

ข้อความ WM_SETFONT จะกำหนดแฮนเดิลของลักษณะอักขระใหม่ ส่วนข้อความ WM_GETFONT จะคืนค่าแฮนเดิลให้กับวินโดวส์หรือวิชวลเบสิก

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

case WM_GETFONT:
    lpbisearch = LpbisearchDEREF(hctl);
    return lpbisearch->hfont;
```

จากที่กล่าวมา ตัวคอนโทรลก็จะสามารถจัดการกับการกำหนดและอ่านค่าของทุกคุณสมบัติที่เป็นลักษณะอักขระได้ โดยที่เราไม่ต้องการจัดการในรายละเอียดอื่น ๆ ของแต่ละลักษณะอักขระเลย เพราะวิชวลเบสิกจะจัดการให้เอง

เมื่อคอนโทรลมีการใช้ลักษณะอักขระ ดังนั้นคอนโทรลอื่นก็อาจใช้แฮนเดิลของลักษณะอักขระที่เหมือนกันก็ได้ จึงไม่ควรลบแฮนเดิลที่ถูกส่งมากับข้อความ WM_SETFONT และในขณะเดียวกันก็ไม่ควรสร้างแฮนเดิลขึ้นเอง เพื่อคืนค่าให้กับวินโดวส์หรือวิชวลเบสิกในข้อความ WM_GETFONT ด้วย

กระบวนงานของคอนโทรลวาดตัวคอนโทรลด้วยฟังก์ชั่น PaintTree ( ) โดยเรียกฟังก์ชั่น DisplayValue ( ) เพื่อแสดงค่าของแต่ละบัพของคอนโทรล ซึ่งต้องทำการเลือกลักษณะของอักขระก่อน เพื่อ

ใช้กับเนื้อหาการแสดงผล แล้วค่อยแสดงอักขระที่ตัวบัพ จากตัวอย่าง แฮนเดิลของลักษณะอักขระเดิมจะถูกเรียกกลับคืนมาที่ท้ายฟังก์ชั่น และให้สังเกตว่า แฮนเดิลของลักษณะอักขระ อาจมีค่าเป็น NULL ได้ จึงต้องใช้ลักษณะอักขระของระบบวินโดวส์แทน

```
VOID NEAR DisplayValue(HWND hwnd, HCTL hctl, HDC hdc,
                       int PofX, int PofY, LPSTR lpstr, BOOL wantBrush)
{
  LPBISEARCH lpbisearch = LpbisearchDEREF(hctl);
  HFONT hfontOld;
      ....
      ....
  hfontOld = SelectObject(hdc, lpbisearch->hfont ? lpbisearch->hfont :
          GetStockObject(SYSTEM_FONT));
      ....
      ....
  SelectObject(hdc, hfontOld);              // Restore old font
      ....
} // DisplayValue ()
```

17. <u>คุณสมบัติแบบรูปภาพ</u> (PICTURE PROPERTIES)

ถึงแม้ว่าเราอาจจะสร้างคุณสมบัติแบบรูปภาพได้ด้วยการเขียนโปรแกรมเองก็ตาม แต่ถ้าหากเราใช้ตัวบ่งชี้ DT_PICTURE ก็จะทำได้ง่ายขึ้น เพราะวิชวลเบสิกจะจัดการในเรื่องต่อไปนี้ให้กับคอนโทรลเอง

- แสดงกรอบข้อความมาตรฐาน (STANDARD DIALOG BOX) ให้กับผู้พัฒนาโปรแกรมด้วยวิชวลเบสิกได้เลือกแฟ้มรูปภาพแบบบิตแมพ , แบบสัญรูป และแบบเมต้าไฟล์

- แสดงข้อความ (none) , (bitmap) , (icon) หรือ (metafile) ในกล่องกำหนดค่าของหน้าต่างคุณสมบัติ ถ้าข้อความถูกลบ (ด้วยปุ่ม [Delete]) ภาพก็จะถูกลบออกจากหน่วยความจำ และจะแสดงข้อความ (none) แทน

- จัดการบันทึกและนำภาพเข้าสู่หน่วยความจำให้ ถ้าใช้ตัวบ่งชี้ PF_fSaveData

- คอยตรวจสอบว่า มีคุณสมบัติหรือคอนโทรลใดบ้างใช้รูปภาพเดียวกัน หากไม่มีคุณสมบัติหรือคอนโทรลใดใช้รูปภาพนั้นแล้ว ก็จะลบภาพนั้นออกจากหน่วยความจำ

- จัดการการคัดลอก (COPY) และการวาง (PASTE) รูปภาพ

ส่วนกระบวนงานของคอนโทรลต้องจัดการเองว่าจะแสดงรูปภาพนั้นเมื่อไร และสามารถกำหนดได้ว่า จะให้คุณสมบัติเป็นภาพแบบใดได้บ้าง (บิตแมพ , สัญรูป , เมต้าไฟล์)

คุณสมบัติแบบรูปภาพใช้ข้อมูล 2 แบบ คือ แฮนเดิล HPIC และโครงสร้าง PIC วิชวลเบสิกใช้แฮนเดิล HPIC ในการอ้างถึงรูปภาพ และใช้ API ของวิชวลเบสิก VBGetPic ในการโอนถ่ายรูปภาพมาไว้ในโครงสร้าง PIC วิชวลเบสิกกำหนดโครงสร้าง PIC ไว้ใน VBAPI.H ดังนี้

```
typedef struct tagPIC
{
      BYTE picType;
      union {
        struct {
          HBITMAP  hbitmap;          // Bitmap
          HPALETTE hpal;             // Accompanying palette
        } bmp;
        struct {
          HANDLE   hmeta;            // Metafile
          int      xExt, yExt;       // Extent
         } wmf;
        struct {
          HICON hicon;               // Icon
        } icon;
      } picData;
      BYTE   picReserved[4];
} PIC;
```

แต่ละค่าหมายถึง

picType กำหนดรูปแบบของภาพ โดยสามารถเป็นค่าใดค่าหนึ่งต่อไปนี้

PICTYPE_NONE ไม่มีภาพใดอยู่ในโครงสร้าง

PICTYPE_BITMAP ภาพบิตแมพ

PICTYPE_METAFILE ภาพจากเมต้าไฟล์

PICTYPE_ICON ภาพสัญรูป

picData.bmp.hbitmap แฮนเดิลของภาพบิตแมพ

picData.wmf.hmeta แฮนเดิลของเมต้าไฟล์

picData.icon.hicon แฮนเดิลของสัญรูป

picData.bmp.hpal แฮนเดิลของสี

picData.wmf.xExt ความกว้างของภาพ ในหน่วยของ Twip

picData.wmf.yExt ความสูงของภาพ ในหน่วยของ Twip

การสร้างคุณสมบัติแบบรูปภาพ ทำได้โดย กำหนดแฮนเดิล HPIC ไว้ในโครงสร้างของผู้เขียนโปรแกรม

```
typedef struct tagBISEARCH
  {
      ....
      HPIC LoadNodeShape;
      ....
} BISEARCH;
```

แล้วกำหนดโครงสร้าง PROPINFO

```
PROPINFO Property_LoadNodeShape =
  {
  "LoadNodeShape",
  DT_PICTURE | PF_fGetData | PF_fSetData | PF_fSaveData | PF_fSetCheck |
          PF_fDefVal,
  OFFSETIN(BISEARCH, LoadNodeShape),
  0, 0, NULL, 0
  };
```

ตัวบ่งชี้ PF_fGetData , PF_fSetData , PF_fSaveData กำหนดให้วิชวลเบสิกโอนถ่ายข้อมูลกับโครงสร้างของผู้เขียนโปรแกรมโดยตรง ส่วนตัวบ่งชี้ PF_fSetCheck ทำให้กระบวนงานของคอนโทรลสามารถตรวจสอบความถูกต้องของรูปภาพก่อนได้ โดยจัดการกับข้อความจากวิชวลเบสิกอันหนึ่ง คือ VBM_CHECKPROPERTY

```
....
#define ERR_BadPicFmt      32000
....
case VBM_CHECKPROPERTY:
   switch (wp) {
      case IPROP_BISEARCH_LOADNODESHAPE:
      {
            PIC pic;
            VBGetPic((HPIC)lp, &pic);
            switch (pic.picType) {
                  case PICTYPE_ICON:
                  case PICTYPE_NONE:
                        lpbisearch->LoadNodeShape = (HPIC) lp;
                        InvalidateRect(hwnd, NULL, TRUE);
                        return ERR_None;
            } // switch (pic.picType)
            return VBSetErrorMessage(ERR_BadPicFmt, "Only icon file supported.");
         } // case IPROP_BISEARCH_LOADNODESHAPE:
      default:
        return VBDefControlProc(hctl, hwnd, msg, wp, lp);
   } // switch (wp)
break;        .// VBM_CHECKPROPERTY
```

จากตัวอย่าง เป็นการกำหนดให้เป็นคุณสมบัติแบบภาพสัญรูปเท่านั้น และเช่นเดียวกับคุณสมบัติอื่น ๆ ที่ต้องกำหนดที่อยู่ของคุณสมบัติในตารางข้อมูลคุณสมบัติ และดัชนีของคุณสมบัติด้วย

```
&Property_LoadNodeShape,
#define IPROP_BISEARCH_LOADNODESHAPE     31
```

18. <u>คุณสมบัติแบบช่องลำดับ</u> (ARRAY PROPERTIES)

ชนิดของคุณสมบัติที่ได้เคยกล่าวมาทั้งหมด สามารถกำหนดให้เป็นแบบช่องลำดับได้ ซึ่งจะมีลักษณะที่

- เมื่อจะอ่านค่าหรือกำหนดค่าคุณสมบัติ จะต้องใช้ตัวชี้ไปยังโครงสร้าง DATASTRUCT ที่มีทั้งข้อมูลและดัชนีของแต่ละช่องที่ใช้เก็บข้อมูล ซึ่งในการอ่านหรือกำหนดค่านั้นจะทำได้ทีละช่องข้อมูลเท่านั้น
- จำนวนช่องข้อมูลที่กำหนดขึ้นในโครงสร้างของผู้เขียนโปรแกรม จะต้องมากพอที่จะเก็บค่าของคุณสมบัติที่เรากำหนดขึ้น
- เมื่อมีการอ่านหรือเปลี่ยนแปลงค่าคุณสมบัตินี้โดยผู้พัฒนาโปรแกรมประยุกต์ด้วยวิชวลเบสิก กระบวนงานของคอนโทรลจะต้องตรวจสอบดัชนีของช่องลำดับด้วยว่า มีค่าอยู่ในช่วงที่กำหนดไว้หรือไม่

- ไม่ควรให้มีการกำหนดค่าของคุณสมบัติชนิดนี้ในช่วงเวลาการออกแบบ ดังนั้นจึงต้องใช้ตัวบ่งชี้ PF_fNoShow ร่วมด้วย ซึ่งก็ทำให้ไม่ต้องมีการบันทึกหรือนำคุณสมบัตินี้เข้าสู่หน่วยความจำแต่อย่างใด (ไม่ต้องใช้ตัวบ่งชี้ PF_fSaveData)

```
#define  MAXNODE      1023        // MAX LEVEL IS 9 (0-9)
typedef struct tagBISEARCH
  {
     ....
     SHORT OccupiedArr[MAXNODE + 1];
     ....
} BISEARCH;
```

เป็นการกำหนดคุณสมบัติแบบช่องลำดับของจำนวนเต็มในโครงสร้างของผู้เขียนโปรแกรม ส่วนโครงสร้าง PROPINFO กำหนดดังนี้

```
PROPINFO Property_OccupiedArr =
  {
  "OccupiedArr",
  DT_SHORT | PF_fPropArray | PF_fGetMsg | PF_fSetMsg | PF_fNoShow,
  OFFSETIN(BISEARCH, OccupiedArr),
  0,  0,  NULL,  0
};
```

คุณสมบัติแบบช่องลำดับต้องใช้ตัวบ่งชี้ PF_fPropArray เสมอ และใช้ PF_GetMsg , PF_fSetMsg และ PF_fNoShow ร่วมด้วย ส่วนตัวบ่งชี้ PF_fSaveData นั้นไม่ต้องใช้ และให้เพิ่มดัชนีและที่อยู่ของคุณสมบัติในตารางข้อมูลคุณสมบัติด้วย

```
#define IPROP_BISEARCH_OCCUPIEDARR      31
&Property_OccupiedArr,
```

ตัวกระบวนงานของคอนโทรลต้องจัดการกับข้อความ VBM_GETPROPERTY จากวิชวลเบสิก โดยการตรวจสอบดัชนี และคืนค่าที่ถูกต้องแก่วิชวลเบสิก

```
case VBM_GETPROPERTY:
  switch (wp) {
    case IPROP_BISEARCH_OCCUPIEDARR:
       {
         LONG i;
         LPDATASTRUCT lpDs = (LPDATASTRUCT) lp;
         i = lpDs->index[0].data;              // Get  index
         if (i < 0 || i >= MAXNODE)            // Is index out of range ?
           return ERR_BadIndex;
         lpDs->data = (LONG)lpbisearch->OccupiedArr[i];  // Transfer data
       }
       return 0;          // Return 0 if OK

     default:
       return VBDefControlProc(hctl, hwnd, msg, wp, lp);
  }
break;     // VBM_GETPROPERTY
```

พารามิเตอร์ lp ในที่นี้เป็นตัวชี้ไปยังโครงสร้าง DATASTRUCT โครงสร้างนี้มีฟิลด์ที่สำคัญคือ

index[0]. data ดัชนีของช่องข้อมูลที่ต้องการ

data ข้อมูลในช่องข้อมูล

หลังจากประกาศตัวแปร lpDs แล้ว ก็จะหาค่าดัชนีโดย

```
i = lpDs->index[0].data;      // Get index
```

ดังนั้น หากชุดคำสั่งของวิชวลเบสิกต้องการอ่านค่า OccupiedArr (5) ค่าของ i ก็จะเป็น 5 จึงนำค่าของ i มาตรวจสอบว่า อยู่ในช่วงที่กำหนดไว้หรือไม่ ถ้าไม่อยู่ในช่วงที่กำหนด จะคืนค่าที่ไม่เท่ากับ 0 ให้วิชวลเบสิก

```
if (i < 0 || i >= MAXNODE)          // Is index out of range ?
    return ERR_BadIndex;
```

จากนั้น จึงโอนถ่ายข้อมูลของช่องที่ต้องการให้กับฟิลด์ lpDs->data

```
lpDs->data = (LONG)lpbisearch->OccupiedArr[i];      // Transfer data
```

แล้วคืนค่า 0 ให้กับวิชวลเบสิก แสดงว่า วิชวลเบสิกสามารถอ่านค่าได้โดยสมบูรณ์ โดยค่าที่อ่านได้ ก็จะอยู่ใน lpDs->data เรียบร้อยแล้ว

ส่วนการจัดการกับข้อความ VBM_SETPROPERTY ก็คล้ายกับ VBM_GETPROPERTY ต่างกันที่ทิศทางของการโอนถ่ายข้อมูลกันเท่านั้น

```
case VBM_SETPROPERTY:
    switch (wp) {
      case IPROP_BISEARCH_OCCUPIEDARR:
          {
              LONG i;
              LPDATASTRUCT lpDs = (LPDATASTRUCT) lp;
              i = lpDs->index[0].data;          // Get index
              if (i < 0 || i >= MAXNODE)        // Is index out of range ?
                  return ERR_BadIndex;
              lpbisearch->OccupiedArr[i] = (SHORT)lpDs->data;      // Transfer data
              return 0;              // Return  0 if OK
          }
      default:
          return VBDefControlProc(hctl, hwnd, msg, wp, lp);
      }       // switch (wp)
break;      // VBM_SETPROPERTY
```

19. <u>คุณสมบัติแบบช่องลำดับของอักขระ</u> (HSZ)

ให้กำหนดแฮนเดิลของ HSZ ในโครงสร้างของผู้เขียนโปรแกรม

```
typedef struct tagBISEARCH
  {
      ....
      HSZ  Key[MAXNODE + 1];
      ....
} BISEARCH;

#define IPROP_BISEARCH_VALUEARR          29
&Property_Key,                    // array property
```

กำหนดโครงสร้าง PROPINFO ซึ่งก็คล้ายกับช่องลำดับแบบจำนวนเต็ม ต่างกันที่ตัวบ่งชี้ชนิดของคุณสมบัติเท่านั้น

```
PROPINFO Property_Key =
  {
  "Key",
  DT_HSZ | PF_fPropArray | PF_fGetMsg | PF_fSetMsg | PF_fNoShow ,
  OFFSETIN(BISEARCH, Key),
  0, 0, NULL, 0
};
```

ส่วนการจัดการกับข้อความ VBM_GETPROPERTY เป็นดังนี้

```
      case VBM_GETPROPERTY:
        switch (wp) {
          case IPROP_BISEARCH_VALUEARR:
              {
                LONG i;
                LPDATASTRUCT lpDs = (LPDATASTRUCT) lp;
                LPSTR lpstr;

                i = lpDs->index[0].data;

                if (i < 1 || i > MAXNODE)          // Is index out of range ?
                    return ERR_BadIndex;

                lpbisearch = LpbisearchDEREF(hctl);
                if (lpbisearch->Key[i] == NULL) {
                  lpDs->data = (LONG)VBCreateHsz((_segment)hctl, "\0");
                  return 0;
                }

                lpstr = VBLockHsz(lpbisearch->Key[i]);
                lpDs->data = (LONG)VBCreateHsz((_segment)hctl, lpstr);
                lpbisearch = LpbisearchDEREF(hctl);
                VBUnlockHsz(lpbisearch->Key[i]);

              }
              return 0;      // Return 0 if OK

            default:
              return VBDefControlProc(hctl, hwnd, msg, wp, lp);

            }

      break;     // VBM_GETPROPERTY
```

ตอนแรกก็ต้องตรวจสอบดัชนีเหมือนเดิม แล้วตรวจสอบว่า แฮนเดิล HSZ เป็น NULL หรือไม่ ถ้าเป็น NULL ก็จะคืนค่าอักขระว่าง (Empty String) ให้แก่วิชวลเบสิก ถ้าไม่เป็น NULL ก็จะอ้างถึง (Dereference) อักขระ แล้วสร้างแฮนเดิล HSZ ใหม่เพื่อโอนถ่ายข้อมูลกับ lpDs->data สังเกตว่า เราไม่สามารถโอนถ่ายแฮนเดิลของช่องลำดับได้โดยตรง เพราะวิชวลเบสิกจะยกเลิกแฮนเดิล HSZ หลังจากได้ใช้แฮนเดิลนั้นแล้ว ดังนั้น จึงต้องสร้างแฮนเดิล HSZ ขึ้นมาใหม่ก่อน เพื่อใช้เก็บค่าของอักขระไว้ชั่วคราว

ฟังก์ชั่น VBLockHsz จะทำให้ที่อยู่ของอักขระ HSZ ไม่เปลี่ยนแปลง จนกว่าจะใช้ฟังก์ชั่น VBUnlockHsz

การจัดการกับข้อความ VBM_SETPROPERTY จะมีการยกเลิกแฮนเดิลเดิมของช่องลำดับก่อน แล้วสร้างแฮนเดิลขึ้นใหม่ เพื่อใช้เข้าถึงที่อยู่ของอักขระ HSZ

```
case IPROP_BISEARCH_VALUEARR:
    {
      LONG i;
      LPDATASTRUCT lpDs = (LPDATASTRUCT) lp;
      HSZ hsz;
      LPSTR lpstr;

      i = lpDs->index[0].data;          // Get  index
      if (i < 1 || i > MAXNODE)         // Is index out of range ?
        return ERR_BadIndex;

      lpbisearch = LpbisearchDEREF(hctl);
      if (lpbisearch->Key[i])
        VBDestroyHsz(lpbisearch->Key[i]);

      hsz =  VBCreateHsz((_segment)hctl, (LPSTR) (lpDs->data));
      lpbisearch = LpbisearchDEREF(hctl);
      lpbisearch->Key[i] = hsz;
      return 0;         // Return 0 if OK
    }
break;
```

20. ขั้นตอนวิธี (Algorithm) ที่สำคัญ

คอนโทรลทั้งสามใช้การเก็บข้อมูลในหน่วยความจำแบบช่องลำดับ ดังนั้นขั้นตอนวิธีที่ใช้จึงแตกต่างจากการเก็บโดยใช้ตัวชี้

20.1 การลบบัพออกจากต้นไม้ไบนารีค้นหาข้อมูล

ถ้าเราสมมติให้บัพที่ตำแหน่งใด ๆ มีหมายเลขเป็น Idx แล้ว ดังนั้นลูกทางซ้ายของ Idx ก็คือ บัพหมายเลข Idx * 2 ส่วนลูกทางขวา ก็คือหมายเลข Idx * 2 + 1 การลบบัพ สามารถแบ่งขั้นตอนวิธีได้เป็น

20.1.1 การลบบัพใบ เราสามารถตรวจสอบว่าบัพ Idx ใดเป็นบัพใบหรือไม่ โดยผู้เขียนกำหนดขั้นตอนวิธีดังนี้

1) กำหนดบัพ Idx ที่ต้องการลบ

2) ตรวจสอบบัพ Idx * 2 ว่าเป็นบัพว่างหรือไม่ ถ้าไม่เป็นบัพว่างแสดงว่าบัพ Idx นั้นไม่ใช่บัพใบ จึงให้ทำข้อ 4) แต่ถ้าเป็นบัพว่างให้ทำข้อต่อไป

3) ตรวจสอบบัพ Idx * 2 + 1 ว่าเป็นบัพว่างหรือไม่ ถ้าไม่เป็นบัพว่างแสดงว่าบัพ Idx นั้นไม่ใช่บัพใบ จึงให้ทำข้อ 4) แต่ถ้าเป็นบัพว่างแสดงว่า บัพ Idx นั้นเป็นบัพใบ จึงทำการลบบัพ Idx นั้นได้เลย โดยให้บัพ Idx นั้นเป็นบัพว่าง

4) จบขั้นตอนวิธีการลบบัพใบ

20.1.2 การลบบัพที่มีลูกทางซ้าย และ/หรือ มีลูกทางขวา ซึ่งจะต้องใช้ขั้นตอนวิธีของการลบบัพใบตรวจสอบก่อนว่า บัพ Idx นั้นเป็นบัพใบหรือไม่ ถ้าไม่ใช่บัพใบแล้วก็จะใช้ขั้นตอนวิธีที่ผู้เขียนกำหนดขึ้นดังนี้

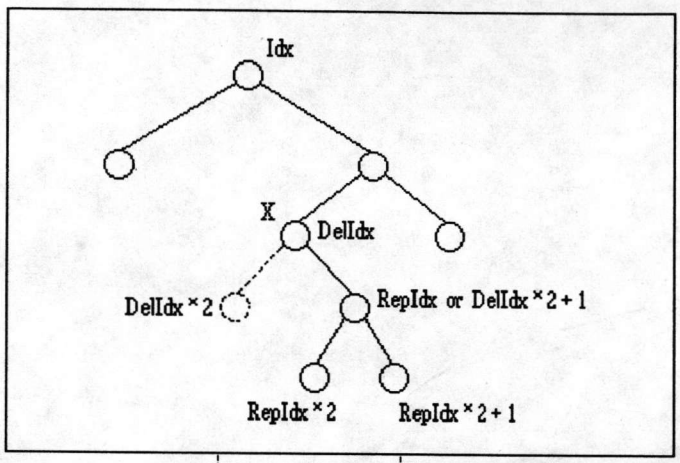
1) กำหนดบัพ Idx ที่ต้องการลบ

2) ค้นหาบัพที่เป็นเพื่อนบ้านถัดไปแบบตามลำดับของ Idx (ผู้เขียนสร้างฟังก์ชั่น FindMinRight( ) ดูรายละเอียดของฟังก์ชั่นนี้ได้จากภาคผนวก ข) ถ้าไม่พบให้ทำข้อ 3) ถ้าพบ ให้ทำข้อ 4)

3) ค้นหาบัพที่เป็นลูกทางซ้ายของ Idx (ผู้เขียนสร้างฟังก์ชั่น FindMaxLeft( ) ดูรายละเอียดของฟังก์ชั่นนี้ได้จากภาคผนวก ข) ซึ่งต้องมีแน่นอน เพราะบัพ Idx นี้ไม่ใช่บัพใบ โดยจะได้หมายเลขบัพ Idx * 2

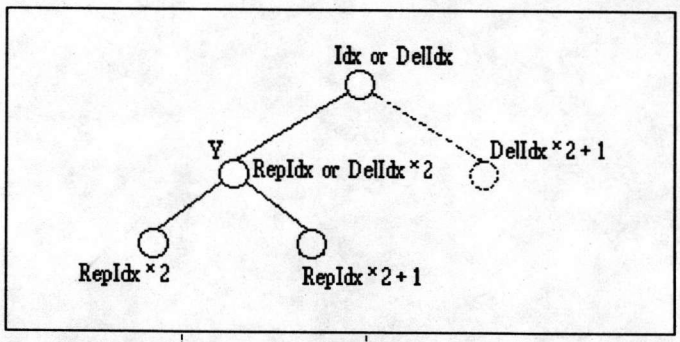
4) ทำการลบบัพ Idx โดยให้ Idx เป็นบัพว่าง จากนั้นถ้ามาจากข้อ 2) ให้ทำข้อ 5) ต่อ ถ้ามาจากข้อ 3) ให้ไปทำข้อ 6)

5) นำบัพที่ค้นหาได้จากข้อ 2) สมมติเป็นบัพ X ไปแทนที่บัพ Idx จากนั้นถ้าบัพ X มีต้นไม้ย่อยทางขวา ก็ให้นำต้นไม้ย่อยนั้นขึ้นมาแทนที่ ณ ตำแหน่งของบัพ X ดังรูปที่ 5.1 (ผู้เขียนสร้างฟังก์ชั่น ReplaceWithRightSon(HCTL hctl, int DelIdx, int RepIdx) ดูรายละเอียดเพิ่มเติมของฟังก์ชั่นนี้ได้ในภาคผนวก ข) ให้สังเกตว่า บัพ X ย่อมต้องไม่มีต้นไม้ย่อยทางซ้ายแน่นอน เนื่องจากเป็นเพื่อนบ้านถัดไปแบบตามลำดับของบัพ Idx



รูปที่ 5.1 การลบบัพที่มีลูกทางขวา

6) นำบัพที่ค้นหาได้จากข้อ 2) สมมติเป็นบัพ Y ไปแทนที่บัพ Idx จากนั้นถ้าบัพ Y มีต้นไม้ย่อยทางซ้าย และ/หรือ ทางขวา ก็ให้นำต้นไม้ย่อยนั้นขึ้นมาแทนที่ ณ ตำแหน่งต้นไม้ย่อยของบัพ Idx เดิม ดังรูปที่ 5.2 (ผู้เขียนสร้างฟังก์ชั่น ReplaceWithLeftSon(HCTL hctl, int DelIdx, int RepIdx) ดูรายละเอียดเพิ่มเติมของฟังก์ชั่นนี้ได้จากภาคผนวก ข)



รูปที่ 5.2 การลบบัพที่มีเฉพาะลูกทางซ้าย

7) จบขั้นตอนวิธีการลบบัพที่มีลูกทางซ้าย และ/หรือ ลูกทางขวา

20.2 การตรวจสอบว่าจะต้องทำต้นไม้ให้สมดุลด้วยวิธีใด

ผู้เขียนได้กำหนดขั้นตอนวิธีใช้ตรวจสอบดังนี้

1) กำหนดหมายเลข Idx ที่จะคำนวณความสูง

2) หาความสูงของต้นไม้ย่อยทางซ้ายของ Idx (ผู้เขียนสร้างฟังก์ชั่น CountHeight(HCTL hctl, int Idx, int HeightNow) ดูรายละเอียดเพิ่มเติมของฟังก์ชั่นนี้ได้จากภาคผนวก ข)

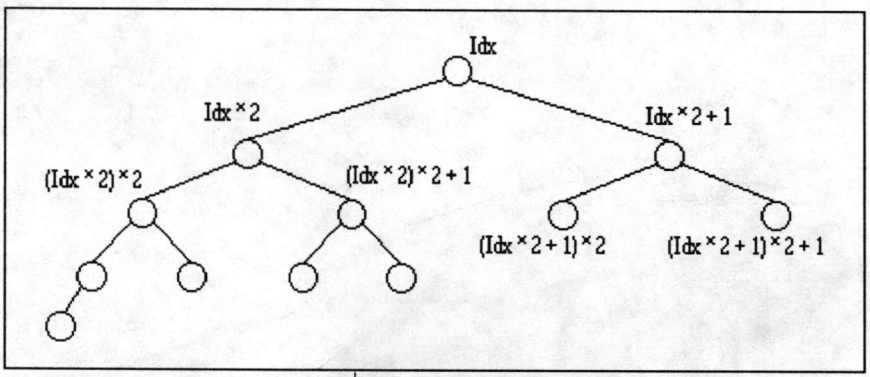
3) หาความสูงของต้นไม้ย่อยทางขวาของ Idx (ด้วยฟังก์ชั่น CountHeight( ) เช่นกัน)

4) คำนวณหาน้ำหนักของบัพ Idx ถ้ามีค่า < -1 (-2) ให้ทำข้อ 5) ต่อไป ถ้าน้ำหนักของบัพ Idx > 1 (2) ให้ทำข้อ 6)

5) เมื่อน้ำหนักของบัพ Idx < -1 แสดงว่าต้นไม้เอียงทางด้านขวา ดังนั้นจึงต้องหาน้ำหนักที่บัพ Idx * 2 + 1 ด้วยขั้นตอนวิธีข้อ 1) ถึงข้อ 3) ถ้าน้ำหนักของบัพ Idx * 2 + 1 = -1 จะต้องทำ SLR ถ้าน้ำหนัก = +1 ก็จะต้องทำ DLR แล้วไปทำข้อ 7)

6) เมื่อน้ำหนักของบัพ Idx > 1 แสดงว่า ต้นไม้เอียงทางด้านซ้าย ดังนั้นจึงต้องหาน้ำหนักที่บัพ Idx * 2 ด้วยขั้นตอนวิธีข้อ 1) ถึงข้อ 3) ถ้าน้ำหนักของบัพ Idx * 2 = +1 จะต้องทำ SRR ถ้าน้ำหนัก = -1 ก็จะต้องทำ DRR

7) จบขั้นตอนวิธีการตรวจสอบฯ



รูปที่ 5.3 การทำ SRR

20.3 การทำ SRR

ผู้เขียนกำหนดขั้นตอนวิธีดังนี้

1) จากขั้นตอนวิธีข้อ 20.2 ทำให้สามารถกำหนดบัพ Idx ที่จะทำ SRR ได้

2) ทำการขยับต้นไม้ย่อยทางขวาของบัพ Idx (ซึ่งมีบัพ Idx * 2 + 1 เป็นบัพราก) ลงมาทางด้านล่าง 1 ระดับ (ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithLeftSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้)

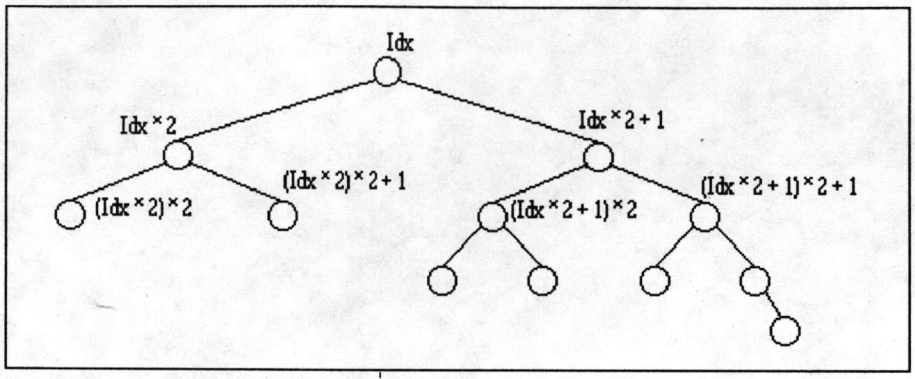
3) นำบัพ Idx ไปแทนที่บัพรากของต้นไม้ย่อยทางขวาของ Idx ซึ่งก็คือบัพ Idx * 2 + 1

4) ทำการเคลื่อนย้ายต้นไม้ที่บัพ (Idx * 2) * 2 + 1 เป็นบัพรากไปเป็นต้นไม้ที่มีตำแหน่ง (Idx * 2 + 1) * 2 เป็นบัพราก ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithLeftSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้

5) ทำการขยับต้นไม้ย่อยทางซ้ายของบัพ Idx ขึ้นไปด้านบน 1 ระดับ ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithLeftSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้ด้วยเช่นกัน

6) จบขั้นตอนวิธีการทำ SRR

20.4 การทำ SLR



รูปที่ 5.4 การทำ SLR

ผู้เขียนกำหนดขั้นตอนวิธีดังนี้

1) จากขั้นตอนวิธีข้อ 20.2 ทำให้สามารถกำหนดบัพ Idx ที่จะทำ SLR ได้

2) ทำการขยับต้นไม้ย่อยทางซ้ายของบัพ Idx (ซึ่งมีบัพ Idx * 2 เป็นบัพราก) ลงมาทางด้านล่าง 1 ระดับ (สามารถใช้ฟังก์ชั่น ReplaceWithRightSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้)

3) นำบัพ Idx ไปแทนที่บัพรากของต้นไม้ย่อยทางซ้ายของ Idx ซึ่งก็คือบัพ Idx * 2

4) ทำการเคลื่อนย้ายต้นไม้ที่บัพ (Idx * 2 + 1) * 2 เป็นบัพรากไปเป็นต้นไม้ที่มีตำแหน่ง (Idx * 2) * 2 + 1 เป็นบัพราก (ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithRightSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้)

5) ทำการขยับต้นไม้ย่อยทางขวาของบัพ Idx ขึ้นไปด้านบน 1 ระดับ (ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithRightSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้ด้วยเช่นกัน)

6) จบขั้นตอนวิธีการทำ SLR

20.5 การทำ DRR

ผู้เขียนกำหนดขั้นตอนวิธีดังนี้

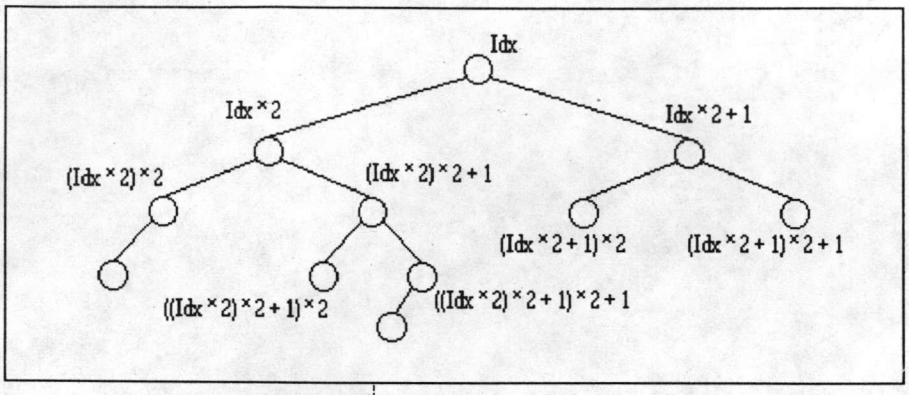
1) จากขั้นตอนวิธีข้อ 20.2 ทำให้สามารถกำหนดบัพ Idx ที่จะทำ DRR ได้

2) ทำการขยับต้นไม้ย่อยทางขวาของบัพ Idx (ซึ่งมีบัพ Idx * 2 + 1 เป็นบัพราก) ลงมาทางด้านล่าง 1 ระดับ (ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithLeftSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้)

3) นำบัพ Idx ไปแทนที่บัพรากของต้นไม้ย่อยทางขวาของ Idx ซึ่งก็คือบัพ Idx * 2 + 1

4) นำบัพ (Idx * 2) * 2 + 1 ไปแทนที่บัพ Idx

5) ทำการเคลื่อนย้ายต้นไม้ที่บัพ ((Idx * 2) * 2 + 1) * 2 + 1 เป็นบัพรากไปเป็นต้นไม้ที่มีตำแหน่ง (Idx * 2 + 1) * 2 เป็นบัพราก (ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithLeftSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้)

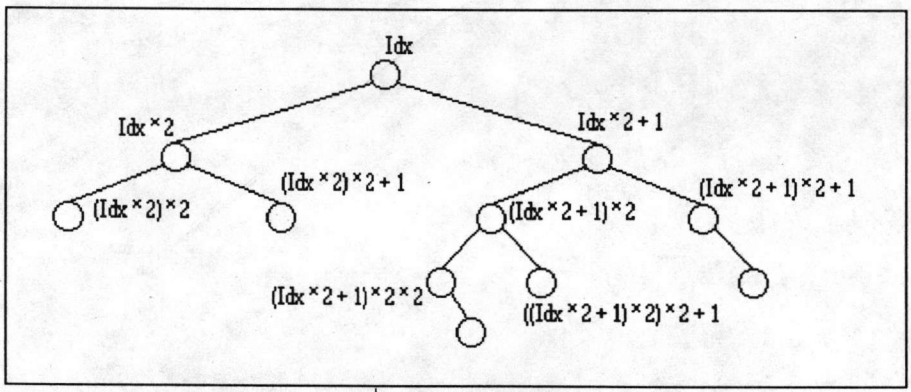


รูปที่ 5.5 การทำ DRR

6) ทำการขยับต้นไม้ย่อยทางซ้ายของบัพ (Idx * 2) * 2 + 1 ขึ้นไปแทนที่บัพ (Idx * 2) * 2 + 1 นี้ (ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithLeftSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้ด้วยเช่นกัน)

7) จบขั้นตอนวิธีการทำ DRR

20.6 การทำ DLR



รูปที่ 5.6 การทำ DLR

ผู้เขียนกำหนดขั้นตอนวิธีดังนี้

1) จากขั้นตอนวิธีข้อ 20.2 ทำให้สามารถกำหนดบัพ Idx ที่จะทำ DLR ได้

2) ทำการขยับต้นไม้ย่อยทางซ้ายของบัพ Idx (ซึ่งมีบัพ Idx * 2 เป็นบัพราก) ลงไปทางด้านล่าง 1 ระดับ (ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithRightSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้)

3) นำบัพ Idx ไปแทนที่บัพรากของต้นไม้ย่อยทางซ้ายของ Idx ซึ่งก็คือบัพ Idx * 2

4) นำบัพ (Idx * 2 + 1) * 2 ไปแทนที่บัพ Idx

5) ทำการเคลื่อนย้ายต้นไม้ที่บัพ ((Idx * 2 + 1) * 2) * 2 เป็นบัพรากไปเป็นต้นไม้ที่มีตำแหน่ง (Idx * 2) * 2 + 1 เป็นบัพราก (ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithRightSon( ) ของ

ขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้)

6) ทำการขยับต้นไม้ย่อยทางขวาของบัพ (Idx * 2 + 1) * 2 ขึ้นไปแทนที่บัพ (Idx * 2 + 1) * 2 นี้ (ซึ่งสามารถใช้ฟังก์ชั่น ReplaceWithRightSon( ) ของขั้นตอนวิธีการลบบัพทำหน้าที่นี้ได้ด้วยเช่นกัน)

7) จบขั้นตอนวิธีการทำ DLR

การทดสอบคอนโทรล

ได้ทำการทดสอบความสามารถ ประสิทธิภาพและการนำแต่ละคอนโทรลไปประยุกต์ใช้ด้วยเครื่องคอมพิวเตอร์ที่เข้ากันได้กับเครื่องของ IBM ใช้หน่วยประมวลผลกลาง (CPU) 80386DX หน่วยความจำแรม 4 เมกกะไบต์ ฮาร์ดดิสค์ความจุ 245 เมกกะไบต์ และจอภาพ VGA

1. ทดสอบความสามารถและประสิทธิภาพของคอนโทรลต้นไม้ไบนารี

ได้กำหนดให้มีการงานร่วมกันของ 4 คอนโทรล ได้แก่ CommandButton , Label , Timer และ BINARY โดยกำหนดคุณสมบัติของแต่ละคอนโทรลเป็นดังนี้

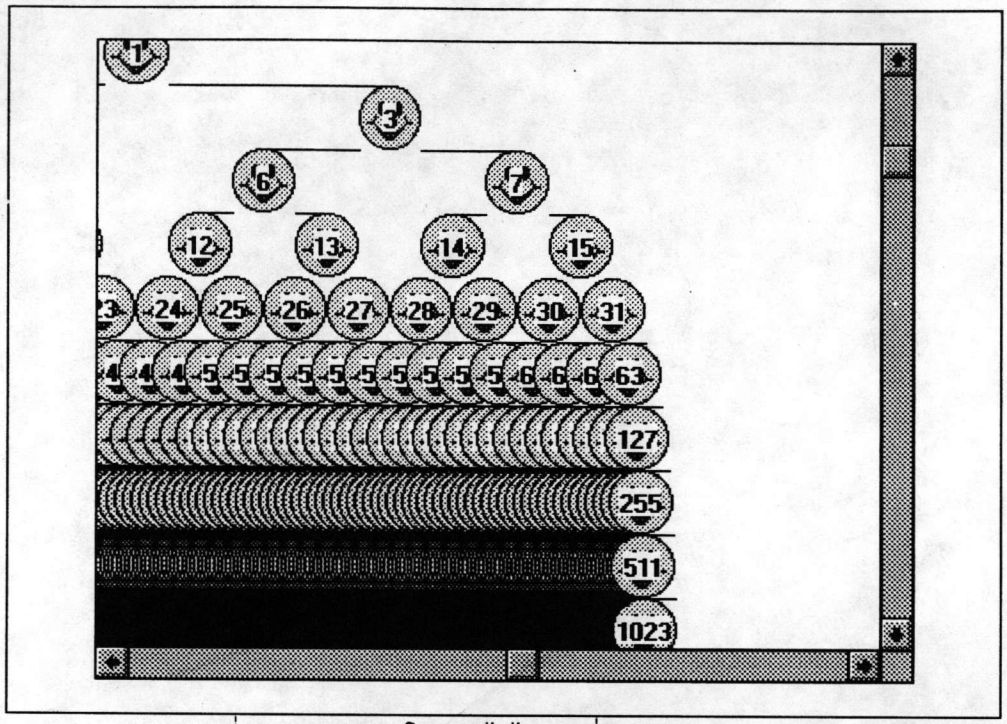
```
Name = Command1
Name = Label1
Name = Timer1
Name = Binary1
Timer1.Interval = 1
Bianry1.NodeShape = 5
Binary1.NodeDistance = - 31
Binary1.LevelDistance = 0
```

แล้วกำหนดกระบวนงานไว้ดังนี้

```
Dim i As Integer

Sub Command1_Click ()
    binary1.Visible = Not binary1.Visible
End Sub

Sub Timer1_Timer ()
    i = i + 1
    binary1.Key(i) = i
    label1.Caption = i
    If (i = 1023) Then
      timer1.Interval = 0
    End If
End Sub
```

เมื่อสั่งดำเนินงานแล้ว หากยังคงให้แสดงคอนโทรลต้นไม้ไบนารีบนจอภาพ การเพิ่มบัพให้กับต้นไม้ไบนารีจะช้าลงเรื่อย ๆ เพราะต้องทำการวาดตัวต้นไม้ใหม่ทุกครั้ง ดังนั้นเมื่อจำนวนบัพเพิ่มขึ้น การวาดต้นไม้ก็ทำได้ช้าลง ซึ่งเป็นผลให้การเพิ่มบัพช้าลงด้วย แต่เมื่อใช้เมาส์คลิกที่ปุ่ม Command1 เพื่อไม่ให้แสดงต้นไม้ไบนารีบนจอภาพแล้ว จะทำการเพิ่มบัพได้รวดเร็วสม่ำเสมอ เพราะไม่ต้องทำการวาดต้นไม้แต่อย่างใด เมื่อค่า

ของ Label1.Caption มีค่าเท่ากับ 1023 แล้วจึงค่อยใช้เมาส์คลิกที่ปุ่ม Command1 อีกครั้งหนึ่งก็จะเห็นต้นไม้ไบนารีเป็นดังรูปที่ 5.7



รูปที่ 5.7 แสดงคอนโทรลต้นไม้ไบนารีเมื่อมีข้อมูล 1,023 บัพ

สรุปได้ว่า คอนโทรลต้นไม้ไบนารีสามารถมีบัพได้สูงสุด 1,023 บัพจริง และสามารถทำงานร่วมกับคอนโทรลอื่น ๆ ได้อย่างถูกต้องตามกระบวนงานและคุณสมบัติที่ได้กำหนดขึ้น

2. ทดสอบความสามารถและประสิทธิภาพของคอนโทรลต้นไม้ไบนารีค้นหาข้อมูล

กำหนดคอนโทรลขึ้นมา 4 คอนโทรล ได้แก่ CommandButton , Label , Timer และ BISEARCH แล้วกำหนดคุณสมบัติของแต่ละคอนโทรลเป็นดังนี้

```
Name = Command1
Name = Label1
Name = Timer1
Name = BiSearch1
Timer1.Interval = 1
BiSearch1.NodeShape = 10
BiSearch1.NodeDistance = -31
BiSearch1.LevelDistance = 0
```

กำหนดกระบวนงาน

```
Dim i As Integer

Sub BiSearch1_NodeIncrease (NodeID As Integer, Key As String)
  i = i + 1
  label1.Caption = i
```

```
End Sub

Sub Command1_Click ()
   bisearch1.Visible = Not bisearch1.Visible
End Sub

Sub Timer1_Timer ()
   bisearch1.InsertedKey = Int(Rnd * 10000)
End Sub
```

การเพิ่มบัพของต้นไม้นี้ หากมีการแสดงภาพของคอนโทรลต้นไม้ไบนารีค้นหาข้อมูลด้วย จะทำให้การเพิ่มบัพช้าลงเรื่อย ๆ เหมือนกับคอนโทรลต้นไม้ไบนารี ถ้าไม่แสดงภาพของต้นไม้ การเพิ่มบัพก็ขึ้นอยู่กับว่า ฟังก์ชั่น Rnd ได้สุ่มตัวเลขที่เมื่อคูณด้วย 10,000 แล้วได้ค่าซ้ำกันหรือไม่ ถ้าได้ค่าไม่ซ้ำกันก็จะเพิ่มบัพได้ และค่าที่ไม่ซ้ำกันนั้นจะต้องทำให้ได้ต้นไม้ที่มีความสูงไม่เกิน 9 ระดับด้วย

เมื่อสั่งดำเนินงานโปรแกรมไปแล้วเป็นเวลาประมาณ 3 นาที ปรากฏว่าสามารถเพิ่มบัพได้ประมาณ 500 บัพ

3. ทดสอบความสามารถและประสิทธิภาพของคอนโทรลต้นไม้ที่มีความสูงสมดุล

ใช้คอนโทรล 4 แบบในการทดสอบ ได้แก่ CommandButton , Label , Timer และ AVL จากนั้นกำหนดคุณสมบัติของแต่ละคอนโทรลดังนี้

```
Name = Command1
Name = Label1
Name = Timer1
Name = AVL1
Timer1.Interval = 1
AVL1.NodeShape = 10
AVL1.NodeDistance = - 31
AVL1.LevelDistance = 0
```

กำหนดกระบวนงานเป็น

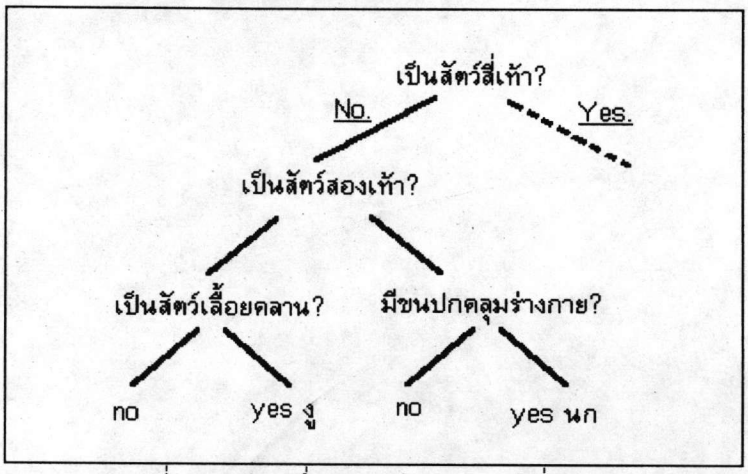
```
Dim i As Integer

Sub Command1_Click ()
    avl1.Visible = Not avl1.Visible
End Sub

Sub Timer1_Timer ()
    i = i + 1
    avl1.InsertedKey = i
    label1.Caption = i
End Sub
```

การเพิ่มบัพของต้นไม้ AVL นี้ ถ้าไม่แสดงลักษณะของต้นไม้บนจอภาพ ก็จะทำการเพิ่มบัพได้ช้าลง เพราะต้องทำการตรวจสอบน้ำหนักของแต่ละบัพแล้วทำการปรับต้นไม้ด้วย แต่ก็ยังสามารถเพิ่มบัพได้รวดเร็วกว่าการแสดงลักษณะของต้นไม้บนจอภาพ

จากการทดสอบปรากฏว่า สามารถเพิ่มบัพได้ถึงค่าที่ 512 เนื่องจากค่าที่ 513 และค่าอื่น ๆ หลังจากนี้จะทำให้ได้ต้นไม้ที่มีความสูงเกิน 9 ระดับ ดังนั้นการทดสอบด้วยวิธีนี้จึงทำให้ได้ต้นไม้ AVL ที่มี 512 บัพ

4. ทดสอบการนำคอนโทรลต้นไม้ไบนารีไปประยุกต์ใช้งาน

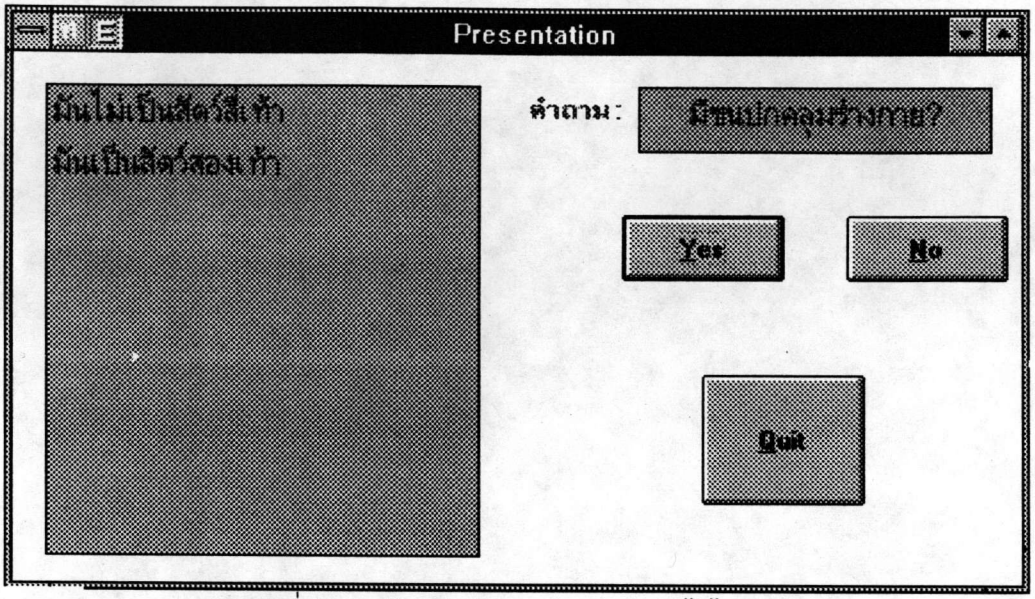
Shiflet (1991) ได้กล่าวถึงการประยุกต์ของต้นไม้ไบนารีไว้ว่า สามารถนำมาเป็นต้นไม้ช่วยตัดสินใจได้ (DECISION TREE) โดยเป็นการถามคำถามที่ต้องการคำตอบว่าใช่หรือไม่ใช่เท่านั้น คำตอบว่า ใช่ จะแทนกิ่งในทิศทางหนึ่ง (ซ้ายหรือขวา) คำตอบว่า ไม่ใช่ ก็จะแทนกิ่งในอีกทิศทางหนึ่ง ส่วนบัพใบนั้นก็จะเป็นคำตอบของการตัดสินใจ ผู้เขียนได้นำมาประยุกต์ใช้ในรูปแบบของระบบผู้เชี่ยวชาญอย่างง่าย ๆ โดยส่วนหนึ่งของฐานข้อมูลของระบบผู้เชี่ยวชาญเป็นดังรูปที่ 5.8



รูปที่ 5.8 ส่วนหนึ่งของฐานข้อมูลระบบผู้เชี่ยวชาญ

จากรูปที่ 5.8 ให้ No. เป็นกิ่งของต้นไม้ย่อยทางซ้าย Yes. เป็นกิ่งต้นไม้ย่อยทางขวา ส่วนค่า no ที่บัพใบหมายถึง ระบบผู้เชี่ยวชาญไม่มีคำตอบให้ ผู้เขียนนำคอนโทรล 7 อันมาวางประกอบกัน เมื่อสั่งดำเนินงานแล้วจะได้หน้าตาของจอภาพดังรูปที่ 5.9 และได้กำหนดคุณสมบัติแต่ละคอนโทรลเป็นดังนี้

```
Name = Label1              ' Label
Name = Label2              ' Label
Name = List1               ' ListBox
Name = yes                 ' CommandButton
Name = no                  ' CommandButton
Name = quit                ' CommandButton
Name = Binary1             ' BINARY
Binary1.Visible = False
```



รูปที่ 5.9 หน้าตาของจอภาพการทดสอบต้นไม้ไบนารี

แล้วกำหนดกระบวนงานต่าง ๆ

```
Dim idx As Integer
' MsgBox return values
Const IDYES = 6                 ' YES button pressed

Sub Form_Load ( )
    binary1.Key(1) = "เป็นสัตว์สี่เท้า?"
    binary1.Key(2) = "เป็นสัตว์สองเท้า?"
    binary1.Key(3) = "มีกระดูกสันหลัง?"
    binary1.Key(4) = "เป็นสัตว์เลื้อยคลาน?"
    binary1.Key(5) = "มีขนปกคลุมร่างกาย?"
    binary1.Key(6) = "มีแปดขา?"
    binary1.Key(7) = "เป็นสัตว์เลี้ยง?"
    binary1.Key(8) = "no"
    binary1.Key(9) = "yes งู"
    binary1.Key(10) = "no"
    binary1.Key(11) = "yes นก"
    binary1.Key(12) = "no"
    binary1.Key(13) = "yes แมลงมุม"
    binary1.Key(14) = "เป็นสัตว์ดุร้าย?"
    binary1.Key(15) = "yes แมว"
    binary1.Key(28) = "yes ฮิปโป"
    binary1.Key(29) = "yes เสือ"
    idx = 1
    label1.Caption = binary1.Key(idx)
```

```
End Sub

Sub yes_Click ( )
    list1.AddItem "มัน" + Mid$(binary1.Key(idx), 1, Len(binary1.Key(idx)) - 1)
    idx = binary1.RightNode(idx)
    Call checking
End Sub

Sub no_Click ( )
    list1.AddItem "มันไม่" + Mid$(binary1.Key(idx), 1, Len(binary1.Key(idx)) - 1)
    idx = binary1.LeftNode(idx)
    Call checking
End Sub

Sub quit_Click ( )
    End
End Sub

Sub checking ( )
    Whichbutton = 0
    If (InStr(binary1.Key(idx), "no") <> 0 Or InStr(binary1.Key(idx), "yes") <> 0) Then
     Beep
     If (InStr(binary1.Key(idx), "no") <> 0) Then
        label1.Caption = "Oh ! I don't know."
     Else
        label1.Caption = "มันคือ " + Mid$(binary1.Key(idx), 4)
     End If
     label2.Caption = "คำตอบ :"
     Whichbutton = MsgBox("ต้องการทดสอบอีกหรือไม่?", 36, "Presentation")
     If (Whichbutton = IDYES) Then
        idx = 1
        label2.Caption = "คำถาม :"
        list1.Clear
     Else
        End
     End If
    End If
    label1.Caption = binary1.Key(idx)
End Sub
```

เมื่อสั่งดำเนินงานโปรแกรมแล้ว ทดลองตอบคำถามโดยใช้เมาส์คลิกที่ปุ่ม Yes หรือ No ปรากฏว่า ระบบผู้เชี่ยวชาญนี้สามารถตอบคำถามได้ถูกต้องตามโครงสร้างข้อมูลต้นไม้ไบนารีที่กำหนดขึ้นด้วยกระบวนงาน Sub Form_Load( )