การตรวจสอบและแก้ไขข้อผิดพลาด

ประเภทของความผิดพลาด

- ข้อผิดพลาดใน VB2005 แบ่งได้ 3 ประเภทคือ
  1. **Syntax Error** ความผิดพลาดเนื่องจากการพิมพ์คำสั่งผิด
  2. **Runtime Error** ความผิดพลาดที่เกิดขณะโปรแกรมทำงาน
  3. **Logic Error** ความผิดพลาดที่เกิดจากการออกแบบโปรแกรม
- ใน VB2005 เรียกข้อผิดพลาดทุกประเภทว่า **Exception** ดังนั้น จึงอาจได้ยินคำว่า **Exception Handling** ซึ่งหมายถึง การจัดการข้อผิดพลาดต่างๆ นั่นเอง

Syntax Error

- เป็นความผิดพลาดเนื่องจากการพิมพ์คำสั่งผิด เช่น การเรียกใช้งานตัวแปรที่ไม่ได้ประกาศค่า, การพิมพ์คำสั่งตกหล่น เป็นต้น ซึ่งข้อผิดพลาดประเภทนี้สามารถลดลงได้ โดยใช้ความสามารถของ Code Editor ถ้าผู้เขียนโปรแกรมไม่แก้ไขให้ถูกต้อง ในตอน Build ก็จะไม่ผ่าน และไม่สามารถรันได้

Runtime Error

- เป็นความผิดพลาดที่เกิดจากสภาวะแวดล้อมในขณะทำงานส่งผลทำให้การทำงานผิดพลาด เช่น มีการป้อนข้อมูลทำให้ตัวแปรที่เป็นตัวหารเป็นศูนย์, ป้อนข้อมูลผิดประเภท หรือไม่เหลือทรัพยากรที่จะใช้รันโปรแกรมต่อไปแล้วทำให้เครื่องหยุดทำงาน เป็นต้น
- เมื่อเกิด Runtime Error นั้นโปรแกรมจะชี้ให้เห็นจุดที่เป็นต้นเหตุความผิดพลาด อธิบายสาเหตุความผิดพลาดอย่างละเอียด พร้อมแนะนำวิธีแก้ไข โดยแสดงไว้ในหน้าต่าง Exception Assistance

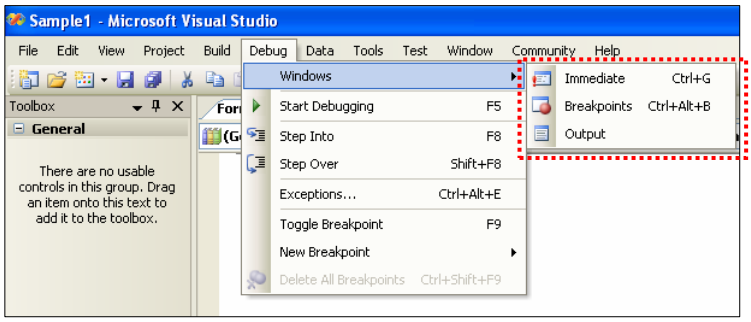
Logic Error

- เป็นความผิดพลาดที่เกิดจากการออกแบบ หรือสร้างโปรแกรมที่ผิดพลาด เพราะแม้ว่าโปรแกรมจะทำงานได้จริง แต่ทำงานไม่ถูกต้องหรือไม่ตรงกับที่ต้องการ ถือเป็นข้อผิดพลาดร้ายแรง และตรวจพบได้ยาก เป็นข้อผิดพลาดที่ผู้เขียนโปรแกรมและผู้ออกแบบโปรแกรมต้องตรวจสอบด้วยตนเอง VB2005 ไม่สามารถตรวจสอบให้ได้

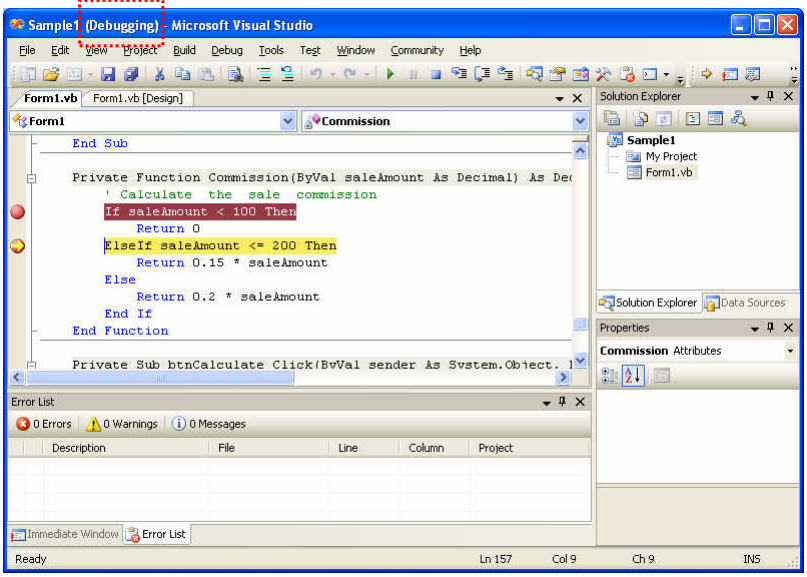
การตรวจสอบและแก้ไขข้อผิดพลาด

- เครื่องมือตรวจสอบข้อผิดพลาด เรียกว่า **Debugger** ซึ่งสามารถตรวจสอบและแก้ไขข้อผิดพลาดได้
- สามารถแบ่งสภาวะการทำงานขณะพัฒนาโปรแกรมได้ 2 สถานะ คือ
  1. **Design Mode** เป็นสภาวะการทำงานขณะที่ออกแบบ และเขียนโปรแกรม
  2. **Debug Mode** เป็นสภาวะที่ใช้ทดสอบการทำงาน หรือเป็นสภาวะที่ใช้หยุดการทำงานของโปรแกรมชั่วคราว เพื่อตรวจสอบว่าโปรแกรมที่เขียนนั้นทำงานได้ถูกต้องหรือไม่ ซึ่งมักเรียกการทำงานสภาวะนี้ว่า “การดีบั๊ก” (Debugging)

รายการเครื่องมือที่ใช้ตรวจสอบความผิดพลาด



สภาวะ Debug Mode



เครื่องมือสำหรับหยุดการทำงานโปรแกรม

เมื่อโปรแกรมทำงานผิดพลาด เราอาจสั่งให้หยุดการทำงาน ณ จุดที่คาดว่าจะเป็นปัญหา แล้วค่อยๆ พิจารณาการทำงานทีละขั้นตอน หรืออาจตรวจสอบตัวแปร, ค่าของข้อมูลต่างๆ ที่เกี่ยวข้อง เพื่อค้นหาจุดที่ผิดพลาดให้พบ

Breakpoint

- **Breakpoint** เป็นการกำหนดจุดที่ต้องการให้โปรแกรมหยุดทำงาน เพื่อทำการตรวจสอบข้อมูลบางอย่าง ซึ่งจุดที่เราหยุดการทำงานจะเรียกว่า "**เบรกพอยต์**" (Break Point)
- สามารถกำหนดเบรกพอยต์โดยเลื่อนเคอร์เซอร์ไปยังบรรทัดที่ต้องการ แล้วกดปุ่ม <F9> หรือใช้เมาส์คลิกที่แถบสีเทาซ้ายมือบรรทัดที่ต้องการ โดยจะปรากฏจุดสีแดงที่หน้าบรรทัดนั้น ส่วนการยกเลิกเบรกพอยต์ทำได้ด้วยวิธีเดียวกัน

Breakpoint

- การกำหนดเบรกพอยต์มักกำหนดไว้ในจุดที่คาดว่าจะเป็นสาเหตุของปัญหา เมื่อโปรแกรมทำงานมาถึงจุดเบรกพอยต์ก็จะหยุดทำงาน จากนั้นผู้เขียนโปรแกรมก็จะทำการตรวจสอบสภาวะของโปรแกรม เช่น ตรวจสอบค่าตัวแปร, ตรวจสอบพร็อพเพอร์ตี้ของออบเจกต์ เป็นต้น เมื่อตรวจสอบเสร็จแล้ว ก็สามารถสั่งให้รันทีละคำสั่งเพื่อหาจัดที่ผิดพลาดต่อไปได้
- หากต้องการจบการทำงานของเบรกพอยต์ โดยออกจากกการทำงานใน Debug Mode ทำได้โดยคลิกเมนู Debug > Stop Debugging

ประเภทของ Breakpoint

- นอกจากกำหนดเบรกพอยต์ให้บรรทัดคำสั่งที่ต้องการแล้ว ยังสามารถกำหนดเบรกพอยต์ประเภทต่างๆ ให้กับจุดที่ต้องการได้ด้วย ซึ่งมีประเภทของเบรกพอยต์ ดังนี้
  - **Address Breakpoint** คือ เบรกพอยต์ที่หยุดการทำงานก่อนที่จะเข้าถึงหน่วยความจำ
  - **Data Breakpoint** คือ เบรกพอยต์ที่หยุดการทำงานก่อนที่จะมีการเปลี่ยนแปลงค่าในตัวแปร
  - **Function Breakpoint** คือ เบรกพอยต์ที่หยุดการทำงานก่อนที่จะเข้าสู่การทำงานของฟังก์ชัน
  - **File Breakpoint** คือ เบรกพอยต์ที่หยุดการทำงานก่อนที่จะเข้าสู่ไฟล์

หยุดการทำงานอย่างมีเงื่อนไข

- แม้ว่าจะกำหนดเบรกพอยต์ได้โดยตรง แต่บางครั้งเราอาจไม่ต้องการให้โปรแกรมหยุดการทำงานทันที แต่ต้องการตรวจสอบก่อนว่าสมควรจะหยุดหรือไม่ ซึ่งเราสามารถทำได้โดยกำหนดเงื่อนไขให้กับเบรกพอยต์แต่ละตัว

หยุดการทำงานอย่างมีเงื่อนไข

- สำหรับเงื่อนไขที่สามารถกำหนดให้หยุดได้มี 3 ประเภท คือ
  1. **Condition** เป็นเงื่อนไขปกติที่สามารถจะหยุดได้ โดยใช้ตัวแปรที่อยู่ใน code มาสร้างเป็นเงื่อนไข
  2. **Hit Count** เป็นการกำหนดได้ว่าจะให้การทำงานของโปรแกรมผ่านจุดเบรกพอยต์นี้กี่ครั้งจึงจะหยุด (ปกติจะหยุดทันทีที่ผ่าน)
  3. **Filter** เป็นการระบุเงื่อนไขในการหยุดได้จากการระบุ Machine (ชื่อเครื่อง), Process และ Thread ซึ่งเราอาจพบได้ในกรณีที่มีการทดสอบการทำงานจากโปรแกรมที่ทำงานแบบขนาน (Parallel Application) ซึ่งมีการกระจายงานย่อยๆ ผ่านโพรเซสเซอร์หลายๆ ตัว

Trace Point

- Trace Point เป็นความสามารถใหม่ของ VB2005 ซึ่งก็คือ การที่สามารถกำหนดได้ว่าเมื่อการทำงานเดินทางมาถึงเบรกพอยต์ที่กำหนดนั้น เราจะทำอะไรดี
- สามารถกำหนด Trace Point ได้โดยระบุในหน้าต่าง When Hit จะปรากฏไดอะล็อก When Breakpoint Is Hit โดยสามารถกำหนดว่าจะให้ทำอะไรดี โดยมีให้เลือก 3 รูปแบบ คือ
  - **Print a message** เป็นการพิมพ์ข้อความที่เกี่ยวกับการทำงาน เช่น ชื่อฟังก์ชัน
  - **Run a macro** เป็นการระบุถึงมาโครที่สั่งให้ทำ
  - **Continue execution** ให้ทำงานต่อไปได้หลังจากการทำงานพิเศษเสร็จสิ้นแล้ว

การสั่งทำงานหลังจาก Breakpoint

- หลังจากหยุดการทำงานตามเบรกพอยต์ที่ตั้งไว้แล้ว สามารถสั่งให้โปรแกรมทำงานในช่วงการดีบักได้ด้วยรูปแบบ ดังนี้
  1. **Step Info** : รันทีละคำสั่ง
  2. **Step Over** : รันคำสั่งในโปรแกรมย่อยเป็นคำสั่งเดียว
  3. **Step Out** : รันคำสั่งที่เหลือในโปรแกรมย่อยในขั้นตอนเดียว
  4. **Run to Cursor** : หยุดรันโปแกรม ณ จุดที่เคอร์เซอร์ค้างไว้

เครื่องมือสำหรับตรวจสอบข้อมูลขณะดีบัก

1. **Locals Window** เป็นหน้าต่างที่ใช้ตรวจสอบค่าตัวแปรทุกๆ ตัวที่อยู่ภายในขอบเขตของโปรแกรมย่อยที่กำลังรันใน Debug Mode เรียกใช้จากคำสั่ง Debug > Windows > Locals
2. **Watch Window** คล้ายกับ Locals Window แต่เพิ่มความสามารถในการตรวจสอบผลกระทำ (Expression) หรือผลการทำงานของโปรแกรมย่อยที่เราต้องการได้
   นอกจากนี้ยังสามารถตรวจสอบค่าคงที่ โดยไม่เปลี่ยนไปมองค่าอื่นๆ ตามที่คำสั่งเปลี่ยนไป (ต่างจาก Locals Window ที่จะแสดงค่าตัวแปรตามโปรแกรมย่อยที่มันเข้าไปทำงาน)

โครงสร้างการเขียนโปรแกรมเพื่อจัดการกับข้อผิดพลาด

- ความผิดพลาด (Exception) ของโปรแกรมสามารถเกิดได้จากสาเหตุหลายๆ ประการ เช่น
  - การหารที่ตัวหารเป็นศูนย์
  - การกรอกข้อมูลผิดประเภท
  - การเปิดไฟล์ที่ไม่มีอยู่ในระบบ
  - การเข้าถึงหน่วยความจำที่สงวนไว้สำหรับระบบ
  - การพยายามเปลี่ยนชื่อไฟล์โดยไม่มีสิทธิ์
  - การละเมิดสิทธ์ด้านความปลอดภัยของระบบ เป็นต้น

โครงสร้างการเขียนโปรแกรมเพื่อจัดการกับข้อผิดพลาด

- การเขียนโปรแกรมเพื่อรองรับข้อผิดพลาด เรียกว่า **Exception Handling** เป็นการพยายามประคับประคองให้โปรแกรมสามารถทำงานได้ตลอดรอดฝั่ง

การใช้คำสั่ง Try…Catch…Finally

- เป็นคำสั่งสำหรับการเขียนโปรแกรมจัดการข้อผิดพลาด มีลักษณะเป็นโครงสร้างการทำงานเป็นบล็อก ดังนี้

```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

คำสั่งที่เสี่ยงต่อความผิดพลาด

คำสั่งจัดการกับความผิดพลาด

คำสั่งหลังจากจัดการกับความผิดพลาดแล้ว

การใช้คำสั่ง Try…Catch…Finally

- จากโครงสร้าง เราต้องนำคำสั่งที่คาดว่าน่าจะผิดพลาด ไว้ระหว่าง Try กับ Catch และนำคำสั่งส่วนที่จะรองรับข้อผิดพลาดมาวางระหว่าง Catch กับ Finally
- ในส่วน Finally อาจมีหรือไม่ก็ได้
- คำสั่งนี้ช่วยให้โปรแกรมสามารถทำงานต่อไปได้ แม้ว่าจะเกิดความผิดพลาดขึ้น ซึ่งเราสามารถใช้โครงสร้างคำสั่ง Try…Catch…Finally ดักความผิดพลาด หรือใช้ครอบคำสั่งที่เราคาดว่าเสี่ยงต่อความผิดพลาด