IBM

eserver

iSeries

แนวคิดเรื่อง ILE

เวอร์ชัน 5 รีลีส 3

SC09-3449-03

IBM

@server

iSeries

แนวคิดเรื่อง ILE

เวอร์ชัน 5 รีลีส 3

SC09-3449-03

หมายเหตุ

ก่อนใช้ข้อมูลและผลิตภัณฑ์ที่สนับสนุน โปรดแน่ใจว่าได้อ่านข้อมูลใน ภาคผนวก D, “ประกาศ”, ในหน้า 227.

พิมพ์ครั้งที่แปด (เมษายน 2004)

| การพิมพ์ครั้งนี้ใช้กับเวอร์ชัน 5, รีลีส 3, โมดิฟิเคชัน 0 ของ IBM Operating System/400 (หมายเลขผลิตภัณฑ์ 5722-SS1) และใช้กับรี
| ลีสและโมดิฟิเคชันถัดจากนี้ไปจนกว่า จะมีการระบุเป็นอย่างอื่นในการพิมพ์ครั้งใหม่. เวอร์ชันนี้ไม่สามารถรันบนโมเดล RISC (reduced
| instruction set computer) และโมเดล CISC ได้ทุกรุ่น.

| การจัดพิมพ์ครั้งนี้ใช้แทน SC09-3449-02.

สารบัญ

เกี่ยวกับแนวคิดเรื่อง ILE (SC09-3449)

หนังสือเล่มนี้อธิบายแนวคิดและเทอมที่เกี่ยวกับสถาปัตยกรรม Language Environment® (ILE) ของ OS/400® ไลเซนส์โปรแกรม. หัวข้อเหล่านี้จะครอบคลุมถึงการสร้างโมดูล (Module creation), การรวมโมดูล (binding), การรันและดีบักโปรแกรม และการจัดการข้อผิดพลาด (Error Handling).

แนวคิดที่อธิบายในหนังสือเล่มนี้จะเกี่ยวข้องกับภาษา ILE ทุกภาษา. โดยแต่ละภาษาอาจจะมีวิธีการที่แตกต่างกันบ้าง. ถ้าต้องการทราบว่าแต่ละภาษาใช้แนวคิดที่อธิบายในที่นี้ได้อย่างไร ให้ดูจากคู่มือโปรแกรมเมอร์สำหรับภาษา ILE นั้น ๆ.

หนังสือเล่มนี้ยังได้อธิบายฟังก์ชันต่างๆของ OS/400 ที่เกี่ยวข้องกับภาษา ILE. โดยเฉพาะอย่างยิ่งเนื้อหาทั่วไปที่เกี่ยวกับการรวมโมดูล (binding), การจัดการข้อความ และการดีบัก.

หนังสือเล่มนี้จะไม่กล่าวถึงการโอนย้ายระบบ (Migration) จากภาษา OS/400 ที่มีอยู่เดิมไปยังภาษา ILE. ข้อมูลในส่วนนี้จะอยู่ในหนังสือ ILE high-level language (HLL) programmer's guide ของแต่ละภาษา.

ใครควรอ่านหนังสือเล่มนี้

คุณควรอ่านหนังสือเล่มนี้ ถ้า:

- คุณเป็นผู้พัฒนาแอ็พพลิเคชันหรือซอฟต์แวร์ทูล
- คุณมีประสบการณ์ในการพัฒนาแอ็พพลิเคชันที่ประกอบด้วยหลายภาษาบนเซิร์ฟเวอร์ iSeries.
- คุณไม่คุ้นเคยกับเซิร์ฟเวอร์ iSeries แต่มีประสบการณ์ในการเขียนแอ็พพลิเคชันบนระบบอื่น.
- โปรแกรมของคุณใช้โพรซีเดอร์ร่วมกัน และเมื่อโพรซีเดอร์ถูกอัพเดตหรือขยาย คุณต้องเขียนโปรแกรมใหม่อีกครั้งเพื่อเรียกโพรซีเดอร์เหล่านั้น.

ถ้าคุณเป็นโปรแกรมเมอร์ที่เขียนโปรแกรมสำหรับ OS/400 อยู่แล้ว และใช้ภาษาใดภาษาหนึ่งเป็นหลัก คุณก็ควรอ่านเนื้อหาในสี่บทแรกของหนังสือเล่มนี้ เพื่อทำความเข้าใจแนวคิด และประโยชน์ที่ได้รับจากการใช้ ILE. จากนั้นให้คุณอ่านคู่มือโปรแกรมเมอร์ของ ILE สำหรับภาษาที่คุณใช้เพียงเท่านี้คุณก็สามารถพัฒนาแอ็พพลิเคชันได้.

สิ่งที่ต้องการก่อน และข้อมูลที่เกี่ยวข้อง

ใช้ iSeries Information Center เป็นจุดเริ่มต้นของท่านสำหรับข้อมูลทางเทคนิคของ iSeries.

คุณสามารถเข้าถึงข้อมูลใน Information Center ได้สองวิธีคือ:

- จากเว็บไซต์ต่อไปนี้:

  `http://www.ibm.com/eserver/iseries/infocenter`

- จาก *iSeries Information Center*, SK3T-4091-04 CD-ROM. CD-ROM นี้ส่งมาพร้อมกับฮาร์ดแวร์ iSeriesใหม่ของคุณหรือ การสั่งอัพเกรดซอฟต์แวร์ IBM Operating System/400. คุณยังสามารถสั่ง CD-ROM ได้จาก IBM® Publications Center:

  `http://www.ibm.com/shop/publications/order`

ใน iSeries Information Center ประกอบด้วยข้อมูลใหม่และอัพเดตของ iSeries เช่น การติดตั้งซอฟต์แวร์และฮาร์ดแวร์, ลินุกซ์, เว็บสเฟียร์®, จาวา™, high availability, ฐานข้อมูล, โลจิคัลพาร์ติชัน, คำสั่ง CL, และ application programming interfaces (APIs) ของระบบ. นอกจากนี้, ยังประกอบไปด้วยตัวแนะนำ และตัวค้นหา เพื่อช่วยในการวางแผน, แก้ปัญหาข้อบกพร่อง, และ การตั้งค่าฮาร์ดแวร์และซอฟต์แวร์ของ iSeries ของคุณ.

เมื่อคุณสั่งซื้อฮาร์ดแวร์ชุดใหม่, คุณจะได้รับ *iSeries Setup and Operations CD-ROM*, SK3T-4098-02. CD-ROM นี้ประกอบด้วย IBM @server IBM e(logo)server iSeries Access for Windows และ EZ-Setup wizard. iSeries Access Family จะให้ชุดไคลเอ็นต์ที่มีประสิทธิภาพและความสามารถของเซิร์ฟเวอร์สำหรับการเชื่อมต่อเครื่อง PC เข้ากับ เซิร์ฟเวอร์ iSeries™. ซึ่ง EZ-Setup จะช่วยให้งานการติดตั้งต่างๆของ iSeries เป็นไปโดยอัตโนมัติ.

สำหรับข้อมูลอื่นที่เกี่ยวข้อง, โปรดดู “รายชื่อเอกสารอ้างอิง” ในหน้า 231.

การส่งข้อเสนอแนะ

ข้อเสนอแนะของคุณเป็นส่วนสำคัญที่ช่วยให้ข้อมูลมีความถูกต้องมากขึ้น. ถ้ามีข้อเสนอแนะเกี่ยวกับหนังสือเล่มนี้หรือเอกสาร iSeries เล่มอื่น กรุณากรอกแบบฟอร์มข้อเสนอแนะที่ท้ายหนังสือเล่มนี้.

- ในกรณีต้องการส่งข้อเสนอแนะทางไปรษณีย์ให้ใช้แบบฟอร์มข้อเสนอแนะสำหรับผู้อ่านที่พิมพ์ไว้ในด้านหลัง. ถ้าคุณกำลังจะส่งจดหมายแบบแสดงความคิดเห็นจากผู้อ่าน จากประเทศหรือภูมิภาคหนึ่งที่นอกเหนือจากประเทศสหรัฐอเมริกา, คุณสามารถนำส่งแบบฟอร์มนี้ที่ตัวแทนในประเทศที่มี IBM สำนักงานตัวแทน หรือ IBM ตัวแทนสำหรับส่งจดหมายสั่งจ่ายไปรษณีย์ภัณฑ์เรียบร้อย.
- ในกรณีต้องการส่งข้อเสนอแนะทางโทรสาร ให้ใช้หมายเลขใดหมายเลขหนึ่งดังต่อไปนี้:
  - ประเทศสหรัฐอเมริกา แคนาดา และเปอร์โตริโก: 1-800-937-3430
  - ประเทศ หรือภูมิภาคอื่นๆ: 1-507-253-5192
- ในกรณีต้องการส่งข้อเสนอแนะทางอิเล็กทรอนิกส์ ให้ใช้อีเมล์แอดเดรสดังต่อไปนี้:
  - ข้อเสนอแนะเกี่ยวกับหนังสือ:

    RCHCLERK@us.ibm.com
  - ข้อเสนอแนะเกี่ยวกับ iSeries Information Center:

    RCHINFOC@us.ibm.com

อย่าลืมส่งข้อมูลดังต่อไปนี้มาด้วย:

- ชื่อของหนังสือ หรือหัวข้อของ iSeries Information Center.
- หมายเลขสิ่งพิมพ์ของหนังสือ.

- หมายเลขหน้าหรือหัวข้อที่คุณมีข้อเสนอแนะ.

บทที่ 1. แนะนำ Integrated Language Environment

ในบทนี้จะให้คำจำกัดความของรูปแบบ Integrated Language Environment (ILE) , อธิบายถึงประโยชน์ของ ILE, และอธิบายถึงการปรากฏขึ้นของ ILE จากแบบจำลองโปรแกรมก่อนหน้านี้.

โดยนำเสนอในแง่มุมของโปรแกรมเมอร์ภาษาอาร์พีจีหรือโคบอลและอธิบายในเทอมของคุณลักษณะเซิร์ฟเวอร์ iSeries ที่ใช้กันอยู่.

ILE คืออะไร?

ILE เป็นชุดเครื่องมือและส่วนสนับสนุนระบบที่เชื่อมโยงกันซึ่งออกแบบมาเพื่อปรับปรุง การพัฒนาโปรแกรมบนระบบ iSeries.

ความสามารถของโมเดลใหม่นี้จะสามารถใช้งานได้ก็ต่อเมื่อ โปรแกรมถูกสร้างด้วยคอมไพเลอร์ที่อยู่ในตระกูล ILE . ตระกูลเหล่านั้นรวมถึง ILE RPG, ILE COBOL, ILE C, ILE C++, และ ILE CL.

ข้อได้เปรียบของ ILE

ILE มีข้อได้เปรียบกว่าโปรแกรมโมเดลรุ่นก่อนๆ มากมาย. ได้แก่ การรวมโมดูลเข้าด้วยกัน (Binding), การเขียนโปรแกรมแบบโมดูล (Modularity), การนำคอมโพเนนต์มาใช้ใหม่ (Reusable Components), การมีฟังก์ชันบริการ, การใช้ร่วมกับแอ็พพลิเคชันที่มีอยู่เดิม และการดีบัก.นอกจากนี้ยังมีการควบคุมรีซอร์สที่ดีขึ้น การควบคุมการโต้ตอบระหว่างภาษาที่ดีขึ้น การทำ code optimization ที่ดีขึ้น การมีสภาพแวดล้อมสำหรับภาษาซีที่ดีขึ้น และมีสิ่งที่เป็นพื้นฐานสำหรับอนาคต.

การรวมโมดูลเข้าด้วยกัน (Binding)

ประโยชน์ของการรวมโมดูล คือการช่วยลดโอเวอร์เฮดในการเรียกโปรแกรม. และทำให้การเรียกโปรแกรมรวดเร็วขึ้น. กลไกของการเรียกโปรแกรมแบบเดิมยังคงมีอยู่ แต่การรวมโมดูลจะเป็นวิธีการที่เร็วกว่า. เพื่อแยกความแตกต่างระหว่างการเรียก 2 ประเภท เราจะเรียกวิธีเรียกแบบเดิมว่า Dynamic Program Call หรือ External Program Call และเรียกวิธีแบบ ILE ว่า Static Procedure Call หรือ Bound Procedure Call.

ความสามารถของรวมโมดูลบวกกับผลจากการพัฒนาประสิทธิภาพในการเรียกโปรแกรม ทำให้เหมาะกับการพัฒนาแอ็พพลิเคชันระดับสูง. คอมไพเลอร์ของ ILE ไม่ได้สร้างเฉพาะโปรแกรมที่สามารถรันได้. แต่จะสร้างโมดูลอ็อบเจ็กต์ (*MODULE) ที่สามารถรวมกับโมดูลอื่น ๆ เพื่อเป็นส่วนที่สามารถรันได้หน่วยหนึ่งเรียกว่าโปรแกรมอ็อบเจ็กต์ (*PGM).

เช่นเดียวกับการเรียกโปรแกรมที่เขียนด้วยภาษาอาร์พีจีจากโปรแกรมภาษาโคบอล ILE สามารถรวมโมดูลที่เขียนมาจากภาษาที่ต่างกันได้. ดังนั้น, เป็นไปได้ที่จะสร้าง โปรแกรมที่สามารถรันเดี่ยวได้ที่ประกอบด้วยโมดูลต่างๆซึ่งถูกเขียนแยกเป็นส่วนๆใน ใน RPG, COBOL, C, C++, และ CL.

การเขียนโปรแกรมแบบโมดูล (Modularity)

ประโยชน์จากการเขียนแบบโมดูลในการพัฒนาแอ็พพลิเคชัน มีดังนี้:

- คอมไพล์ได้รวดเร็วขึ้น

  หากโค้ดที่คอมไพล์ยิ่งมีขนาดเล็กเท่าไร คอมไพเลอร์ก็จะทำงานได้เร็วขึ้นเท่านั้น. คุณสมบัตินี้มีความสำคัญมากเวลาแก้ไขโปรแกรม เนื่องจากบ่อยครั้งมีการแก้ไขโปรแกรมเพียง 1 หรือ 2 บรรทัด. เมื่อเราแก้ไขเพียง 2 บรรทัด เราอาจต้องคอมไพล์โปรแกรมใหม่ทั้งหมดถึง 2,000 บรรทัด. เป็นการใช้รีซอร์ส อย่างไม่มีประสิทธิภาพ.

  ถ้าแบ่งโค้ดให้เป็นโมดูล และใช้ประโยชน์จากความสามารถในการรวมโมดูลของ ILE เราอาจจะคอมไพล์ใหม่เพียง 100 หรือ 200 บรรทัด. ถึงแม้ว่าจะต้องมีขั้นตอนของการรวมโมดูลด้วย แต่กระบวนการนี้ก็ยังเร็วกว่าอย่างเห็นได้ชัด.

- ง่ายต่อการดูแล

  เมื่อมีการแก้ไขปรับปรุงโปรแกรมที่มีขนาดใหญ่มาก เป็นเรื่องยากที่จะเข้าใจโค้ดได้อย่างชัดเจน. โดยเฉพาะอย่างยิ่ง โปรแกรมเมอร์ที่เขียนโปรแกรมแต่ละคนมีรูปแบบในการเขียนโปรแกรมที่แตกต่างกัน. ชิ้นโค้ดขนาดเล็กกว่าที่แสดงถึงการฟังก์ชันเพียงฟังก์ชันเดียวจะทำให้เข้าใจถึงการทำงานได้ง่ายกว่า.ดังนั้น การทำงานในแต่ละโมดูลจะค่อนข้างชัดเจนมากขึ้น และจะพบผลกระทบข้างเคียงที่ไม่ต้องการน้อยมากเมื่อมีการแก้ไขโค้ด.

- ง่ายต่อการทดสอบ

  หน่วยในการคอมไพล์ที่เล็กลงจะช่วยให้ทดสอบฟังก์ชันการทำงานแบบแยกส่วนได้ดีขึ้น. และช่วยให้แน่ใจว่าการทดสอบได้ครอบคลุมทุกส่วน นั่นคือ ส่วนของอินพุต และตรรกะ (logic) ต่าง ๆ ได้ถูกทดสอบทั้งหมด.

- การใช้รีซอร์สในการเขียนโปรแกรมที่ดีขึ้น

  การเขียนโปรแกรมแบบโมดูลเหมาะสมกับหน่วยงานที่ใช้คนจำนวนมาก. โดยปกติแล้วเป็นเรื่องยากในการแบ่งงานออกเป็นส่วนย่อยสำหรับการเขียนโปรแกรมขนาดใหญ่. แต่การเขียนโปรแกรมทั้งหมดอาจจะเป็นเรื่องยากเกินไปสำหรับโปรแกรมเมอร์ระดับต้น หรืออาจจะเป็นการสูญเปล่าถ้าต้องใช้ความสามารถของโปรแกรมเมอร์ระดับสูง.

- ง่ายต่อการย้ายโค้ดที่มาจากแพล็ตฟอร์มอื่น

  โปรแกรมที่เขียนบนแพล็ตฟอร์มอื่น เช่น ยูนิกซ์® มักจะมีลักษณะเป็นโมดูล.โมดูลเหล่านี้สามารถย้ายมาสู่ OS/400 และทำงานร่วมกับโปรแกรม ILE ได้.

การนำคอมโพเนนต์มาใช้ใหม่ (Reusable Component)

ILE ยอมให้เลือกแพ็กเกจของรูทีนที่สามารถรวมเข้ากับโปรแกรมได้. รูทีนที่เขียนจากภาษา ILE ภาษาใดก็ตามสามารถใช้โดยผู้ใช้ที่ใช้คอมไพเลอร์ ILE ของ iSeries ทั้งหมด. ดังนั้น โปรแกรมเมอร์สามารถเลือกภาษาใดก็ได้ในการเขียนโปรแกรม ซึ่งทำให้คุณมีทางเลือกในรูทีนมากที่สุดเท่าที่จะเป็นได้.

กลไกนี้เหมือนกับที่ ไอบีเอ็มและผู้ค้ารายอื่นๆ ใช้ในการส่งแพ็กเกจเหล่านี้ไปให้คุณ คือให้คุณสามารถนำไปใช้กับแอ็พพลิเคชันของคุณเอง คุณสามารถพัฒนาโปรแกรมติดตั้งของคุณได้ตามรูปแบบมาตรฐาน และสามารถใช้ภาษาใดในการพัฒนาก็ได้.

ไม่เพียงแต่สามารถใช้รูทีนในแอ็พพลิเคชันของคุณ. คุณยังสามารถพัฒนารูทีนในภาษา ILE ที่คุณเลือก และขายรูทีนที่คุณพัฒนาไปยังผู้ใช้ ILE อื่นได้อีกด้วย.

การมีฟังก์ชันบริการ (Common Run-Time Service)

ILE ได้ให้ฟังก์ชันประเภทบริการ (**Bindable APIs**) มาด้วย ซึ่งคุณสามารถนำไปใช้ในแอ็พพลิเคชันได้. โดยฟังก์ชันบริการที่ให้มา ประกอบด้วย:

ฟังก์ชันเกี่ยวกับวันและเวลา

การจัดการเกี่ยวกับแมสเสจ

รูทีนด้านคณิตศาสตร์

การควบคุมการจัดการหน้าจอได้ดีขึ้น

การจองเนื้อที่แบบไดนามิก

ในอนาคตอาจมีรูทีนใหม่ๆเพิ่มขึ้นได้จากทั้งไอบีเอ็มและบริษัทผู้พัฒนาอื่นๆ.

ไอบีเอ็มได้เตรียมข้อมูลออนไลน์ ซึ่งบอกรายละเอียดเกี่ยวกับ API ที่ให้มากับ ILE. โปรดดู ส่วน *API* ซึ่ง พบได้ในหมวด **Programming** ของ iSeries Information Center.

การใช้ร่วมกับแอ็พพลิเคชันที่มีอยู่แล้ว

โปรแกรมของ ILE สามารถใช้ร่วมกับโปรแกรมของ OPM ได้. โดยโปรแกรมของ ILE สามารถเรียกโปรแกรมของ OPM และโปรแกรมของ ILE อื่นๆ ได้. ในขณะเดียวกันโปรแกรมของ OPM ก็สามารถเรียกโปรแกรมของ ILE และโปรแกรมของ OPM อื่นๆ ได้เช่นกัน. ดังนั้นถ้าหากมีการวางแผนที่ดีก็จะสามารถเปลี่ยนไปเป็น ILE ทั้งหมดอย่างค่อยเป็นค่อยไปได้.

Source Debugger

ซอร์สดีบักเกอร์ยอมให้คุณดีบักโปรแกรมและเซอร์วิสโปรแกรมของ ILE ได้. สำหรับข้อมูลที่เกี่ยวข้องกับ ซอร์สดีบักเกอร์ ดูได้จาก บทที่ 10, “ข้อพิจารณาในการดีบักโปรแกรม”, ในหน้า 143.

การควบคุมรีซอร์สที่ดีขึ้น

ก่อนที่จะมี ILE รีซอร์สที่ใช้ในโปรแกรม(ตัวอย่างเช่น ไฟล์ที่เปิดอยู่) จะถูกจำกัดอยู่ใน:

โปรแกรมที่จองรีซอร์สนั้น ๆ

Job

ในหลายกรณีที่ข้อจำกัดนี้บังคับให้ผู้ออกแบบแอ็พพลิเคชันจำเป็นจะต้องเลือกอย่างใดอย่างหนึ่ง.

ILE ได้เสนอทางเลือกที่ 3. คือให้ส่วนหนึ่งของ job สามารถเป็นเจ้าของรีซอร์สได้เอง. ซึ่งทำได้โดยใช้ส่วนที่สร้างโดย ILE ที่เรียกว่า **Activation Group**. ภายใต้ ILE รีซอร์สจะสามารถถูกจำกัดวงอยู่ในส่วนหนึ่งส่วนใดในสามส่วนนี้คือ:

โปรแกรม

Activation group

งาน

สถานการณ์ตัวอย่าง—การแบ่งใช้ Open Data Path ร่วมกัน

การแบ่งใช้ Open Data Paths (ODPs) ร่วมกันเป็นตัวอย่างหนึ่งของรีซอร์สที่สามารถควบคุมได้ดีขึ้นด้วย ILE.

ในการพัฒนาประสิทธิภาพของแอ็พพลิเคชันบนเซิร์ฟเวอร์ iSeries โปรแกรมเมอร์ตัดสินใจใช้ shared ODP สำหรับไฟล์ต้นฉบับของลูกค้า. โดยไฟล์นั้นจะถูกใช้ทั้งในแอ็พพลิเคชันสำหรับการป้อนคำสั่งซื้อและแอ็พพลิเคชันสำหรับออกบิล.

เนื่องจากการ Shared ODP ถูกจำกัดวงให้อยู่เฉพาะในงานเท่านั้น จึงเป็นไปได้ที่แอ็พพลิเคชันหนึ่งทำให้เกิดปัญหาต่อแอ็พพลิเคชันอื่นๆ. โดยบังเอิญ การหลีกเลี่ยงปัญหาต่างๆ ต้องมาจากความร่วมมือกันระหว่างผู้พัฒนาแอ็พพลิเคชัน. ดังนั้นถ้าแอ็พพลิเคชันนั้นซื้อมาจากผู้ผลิตหลายราย การหลีกเลี่ยงปัญหาที่กล่าวมาก็อาจจะเป็นไปไม่ได้เลย.

ปัญหาชนิดใดที่จะสามารถเกิดขึ้นได้โดยพิจารณาสถานการณ์ข้างล่างนี้:

1. ไฟล์ต้นฉบับของลูกค้ามีการป้อนหมายเลขลูกค้า (Account Number) และเร็กคอร์ดที่เก็บหมายเลขลูกค้าได้แก่ A1, A2, B1, C1, C2, D1, D2 เป็นต้น.
2. ผู้ปฏิบัติงานกำลังทำการทบทวนเร็กคอร์ดของไฟล์ต้นฉบับ โดยจะต้องมีการปรับปรุงค่าของแต่ละเร็กคอร์ดก่อนที่จะเรียกเร็กคอร์ดถัดไป. สมมติว่า เร็กคอร์ดที่แสดงในปัจจุบันคือ B1.
3. เสียงโทรศัพท์ดังขึ้น. ลูกค้า D1 ต้องการสั่งซื้อของ.
4. โอเปอเรเตอร์กดปุ่มเพื่อไปยังส่วนของการป้อนคำสั่งซื้อ และดำเนินการสั่งซื้อให้กับลูกค้า D1 แล้วกลับไปสู่การแสดงผลของไฟล์ต้นฉบับดังเดิม.
5. โปรแกรมยังคงแสดงเร็กคอร์ด B1 แต่เมื่อโอเปอเรเตอร์เรียกเร็กคอร์ดต่อไป เร็กคอร์ดไหนจะถูกแสดงขึ้นมา.

   ถ้าตอบว่า D2 นั่นคือคำตอบที่ถูกต้อง. เมื่อแอ็พพลิเคชันป้อนคำสั่งซื้ออ่านเร็กคอร์ด D1 ตำแหน่งของไฟล์ในปัจจุบันจะเปลี่ยนไป เพราะ Shared ODP ถูกจำกัดขอบเขตไว้เฉพาะงาน. ดังนั้นการเรียกเร็กคอร์ดต่อไปจะหมายถึงเร็กคอร์ดที่ต่อจาก D1.

ภายใต้ ILE จะสามารถป้องกันปัญหานี้ได้โดยการทำกระบวนการปรับปรุงไฟล์ต้นฉบับใน Activation Group สำหรับการออกบิลโดยเฉพาะ. เช่นเดียวกันกับการป้อนคำสั่งซื้อ ที่จะทำงานอยู่ใน Activation Group อีกกลุ่มหนึ่ง. แอ็พพลิเคชันแต่ละตัวยังคงได้ประโยชน์จาก Shared ODP โดยที่ในแต่ละส่วนจะมี Activation Group ที่มี Shared ODP เฉพาะของตัวเอง. การวางขอบเขตในระดับนี้จะช่วยป้องกันการรบกวนที่บรรยายไว้ในตัวอย่างนี้.

การกำหนดขอบเขตรีซอร์สให้เป็น Activation Group ทำให้โปรแกรมเมอร์มีอิสระที่จะพัฒนาแอ็พพลิเคชันที่ทำงานโดยไม่ขึ้นกับแอ็พพลิเคชันอื่นในงานเดียวกัน. ดังนั้น การพัฒนาโปรแกรมจึงไม่จำเป็นต้องอาศัยความร่วมมือระหว่างผู้พัฒนาแอ็พพลิเคชันมากนัก และยังเป็นการเพิ่มความสามารถในการที่จะเขียนส่วนเพิ่มเติมลงในแอ็พพลิเคชันที่มีอยู่แล้ว.

สถานการณ์ตัวอย่าง—Commitment Control

ความสามารถในการกำหนดขอบเขตของ ODP ให้กับแอ็พพลิเคชันเป็นประโยชน์ในส่วนของ Commitment Control.

สมมุติว่าต้องการใช้งานไฟล์ๆ หนึ่งภายใต้ Commitment Control และยังต้องการใช้ Shared ODP. กรณีไม่มี ILE เมื่อมีโปรแกรมหนึ่งเปิดไฟล์ภายใต้ Commitment Control ทุกโปรแกรมในงานจะต้องทำในลักษณะเดียวกัน. ถึงแม้ว่าความต้องการของ Commitment จะต้องการโปรแกรมเพียงหนึ่งหรือสองโปรแกรมเท่านั้น.

ปัญหาสำคัญสำหรับสถานการณ์นี้คือ เมื่อโปรแกรมใดๆ ในงานมีคำสั่ง commit ข้อมูลใหม่ทั้งหมดจะถูก commit. แม้ว่าจะไม่ใช่ส่วนหนึ่งของแอ็พพลิเคชันที่เกี่ยวข้องก็ตาม.

เราสามารถหลีกเลี่ยงปัญหานี้ได้โดยการรันแต่ละส่วนของแอ็พพลิเคชันที่ต้องการ Commitment Control ใน Activation Group ที่แยกต่างหากไปอีกกลุ่มหนึ่ง.

การควบคุมการโต้ตอบระหว่างภาษาที่ดีขึ้น

ในกรณีที่ยังไม่มี ILE วิธีการที่โปรแกรมทำงานบนเซิร์ฟเวอร์ iSeries จะขึ้นอยู่กับองค์ประกอบดังต่อไปนี้:

มาตรฐานของภาษา เช่น มาตรฐาน ANSI สำหรับภาษาโคบอลและภาษาซี.

ผู้พัฒนาคอมไพเลอร์

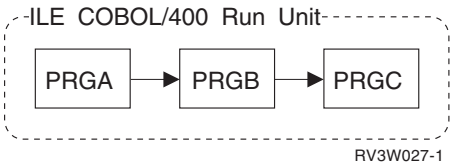
องค์ประกอบเหล่านี้ทำให้เกิดปัญหาได้ในกรณีที่พัฒนาโปรแกรมโดยใช้หลายภาษารวมกัน.

สถานการณ์ตัวอย่าง—การใช้หลายภาษาร่วมกัน

ในตอนที่ยังไม่มี ILE Activation Group การติดต่อระหว่างภาษา OPM นั้นเป็นสิ่งยากเกินกว่าจะคาดเดาได้. การมี ILE Activation Group จะช่วยแก้ไขปัญหานี้ได้.

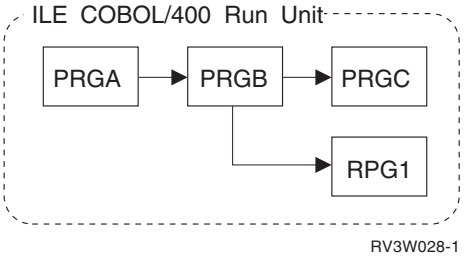
ตัวอย่างเช่น ปัญหาที่เกิดจากการรวมกันระหว่างภาษาโคบอลกับภาษาอื่น. มาตรฐานของภาษาโคบอลมีแนวคิดหนึ่งที่เรียกว่า **Run Unit**. ในหนึ่ง run unit จะประกอบด้วยโปรแกรมหลายโปรแกรมรวมกันอยู่ ดังนั้นภายใต้สภาวะที่แน่นอนนั้น โปรแกรมทั้งหมดใน run unit หนึ่ง ๆ จะมีการทำงานที่ถือเป็นหน่วยเดียวกัน. ซึ่งเป็นคุณลักษณะที่มีประโยชน์มาก.

สมมุติให้โปรแกรม ILE COBOL ทั้งสามโปรแกรม (PRGA, PRGB และ PRGC) รวมกันเป็นแอ็พพลิเคชันขนาดเล็ก โดยที่ PRGA เรียกใช้ PRGB และ PRGB เรียก PRGC อีกทอดหนึ่ง (ดังใน รูปที่ 1 ในหน้า 6). ตามกฎของ ILE COBOL โปรแกรมทั้งสามจะถือว่าอยู่ใน Run Unit เดียวกัน. ผลที่ได้ก็คือ ถ้าโปรแกรมใดโปรแกรมหนึ่งจบการทำงาน โปรแกรมทั้งหมดจะต้องจบการทำงานด้วย และการควบคุมจะรีเทิร์นไปยังโปรแกรมที่เรียก PRGA.



รูปที่ 1. แสดงโปรแกรม ILE COBOL ทั้งสามใน Run Unit เดียวกัน

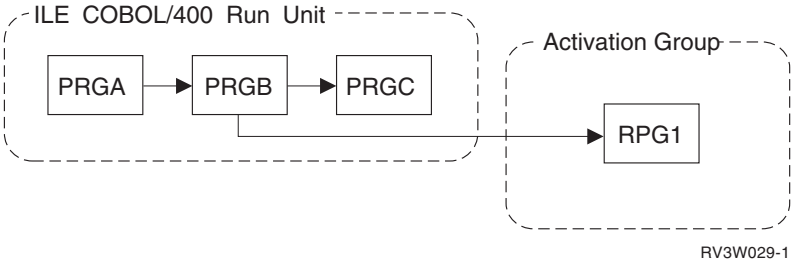
สมมุติว่าเราใส่โปรแกรมภาษาอาร์พีจีโปรแกรมหนึ่ง (RPG1) ลงในแอ็พพลิเคชันและ RPG1 ถูกเรียกโดยโปรแกรมภาษาโคบอล PRGB (รูปที่ 2). โปรแกรมภาษาอาร์พีจีมองว่า ตัวแปร ไฟล์และรีซอร์สอื่นๆของมันยังคงไม่เปลี่ยนแปลงจนกว่าโปรแกรมจะรีเทิร์นพร้อมด้วยตัวแสดงสถานะเร็กคอร์ดสุดท้าย (Last Record Indicator).



รูปที่ 2. แสดงโปรแกรม ILE COBOL สามโปรแกรมและโปรแกรม ILE RPG หนึ่งโปรแกรมใน Run Unit เดียวกัน.

อย่างไรก็ตามโดยความเป็นจริงแล้ว เราไม่สามารถรับประกันได้ว่า โปรแกรม RPG1 ที่เขียนด้วยภาษา อาร์พีจีนั้นจะทำงานได้ตามที่กำหนดไว้ทั้งหมดเมื่อมาเป็นส่วนหนึ่งของ Run Unit ของโคบอล. ถ้า run unit จบการทำงาน RPG1 ก็จะหายไปโดยไม่มีการกำหนดตัวแสดงสถานะเร็กคอร์ดสุดท้าย. ในหลายกรณีสถานการณ์เช่นนี้อาจเป็นสิ่งที่คุณต้องการ. แต่ถ้า RPG1 เป็นยูทิลิตี้โปรแกรมที่อาจจะควบคุมการออกเลขที่ใบกำกับภาษี คุณก็คงไม่สามารถยอมรับสถานการณ์แบบนี้ได้.

เราสามารถป้องกันสถานการณ์แบบนี้ได้โดยการรันโปรแกรมภาษาอาร์พีจีใน activation group ที่แยกออกจาก run unit ของโคบอล (รูปที่ 3). และ Run Unit ของ ILE COBOL ก็เป็น Activation Group ด้วย.



รูปที่ 3. แสดงโปรแกรม ILE RPG ใน Activation Group ที่แยกออกไป.

สำหรับข้อมูลเกี่ยวกับความแตกต่างระหว่าง OPM run unit และ ILE run unit, โปรดดู ILE COBOL for AS/400® Programmer's Guide .

การทำ Code Optimization ที่ดีขึ้น

ตัวแปลภาษา ILE ทำการ optimization ได้หลายชนิดกว่าตัวแปลภาษา OPM. แม้ว่าคอมไพเลอร์จะมีการทำ optimization ในบางส่วน แต่การทำ optimization ส่วนใหญ่บน OS/400 จะทำโดยตัวแปลภาษา.

คอมไพเลอร์แบบ ILE-enabled ไม่ได้สร้างโมดูลขึ้นมาโดยตรง. อันดับแรกจะสร้างรูปแบบที่เรียกว่า Intermediate Form ของโมดูลหลังจากนั้นจะเรียกตัวแปลของ ILE มาเพื่อแปลง Intermediate Code ไปเป็นชุดคำสั่งที่สามารถรันได้.

สภาพแวดล้อมสำหรับภาษาซีที่ดีขึ้น

ภาษาซีเป็นภาษาที่เป็นที่นิยมของนักสร้างเครื่องมือ. ด้วยเหตุนี้ ภาษาซีที่ดีกว่าเดิมจึงหมายถึงเครื่องมือช่วยพัฒนาแอ็พพลิเคชันใหม่ๆ จำนวนมากจะถูกสร้างขึ้นเพื่อใช้กับ OS/400. สำหรับคุณแล้ว นั้นหมายถึงทางเลือกที่มากขึ้นของ:

เครื่องมือสำหรับ CASE

ภาษารุ่นที่ 4 (4GLs)

โปรแกรมภาษาอื่น

เอดิเตอร์

ตัวดีบักเกอร์

รากฐานสำหรับอนาคต

ประโยชน์และฟังก์ชันของ ILE จะเป็นสิ่งสำคัญในอนาคต. คอมไพเลอร์ของ ILE ในอนาคตจะก่อให้เกิดการพัฒนาที่ยิ่งใหญ่. เมื่อเราเปลี่ยนไปใช้ภาษา Object-Oriented Programming และเครื่องมือประเภท Visual Programming ความต้องการใช้งาน ILE ก็จะยิ่งมากขึ้น.

เมื่อวิธีการเขียนโปรแกรมขึ้นอยู่กับการใช้การทำให้เป็นโมดูลในระดับสูงเพิ่มมากขึ้น. แอ็พพลิเคชันก็จะถูกสร้างขึ้นโดยการประกอบส่วนประกอบเล็กๆ ที่สามารถนำกลับมาใช้ได้จำนวนมากเข้าด้วยกัน. ถ้าส่วนประกอบเหล่านี้ไม่สามารถส่งคอนโทรลระหว่างกันได้อย่างรวดเร็วก็จะทำให้แอ็พพลิเคชันนั้นไม่สามารถทำงานได้.

ประวัติความเป็นมาของ ILE

ILE เป็นเพียงก้าวหนึ่งของการวิวัฒนาการของโมเดลการเขียนโปรแกรมบน OS/400. การพัฒนาในแต่ละขั้นทำขึ้นเพื่อสนองความต้องการของโปรแกรมเมอร์.

เมื่อระบบ AS/400 พัฒนาขึ้นในครั้งแรก ได้มีสภาวะแวดล้อมในการโปรแกรมที่เรียกว่า Original Program Model (OPM). และมีการพัฒนา Extended Program Model (EPM) เพิ่มเติมเข้ามาในเวอร์ชัน 1 รีลีส 2.

ลักษณะของ Original Program Model

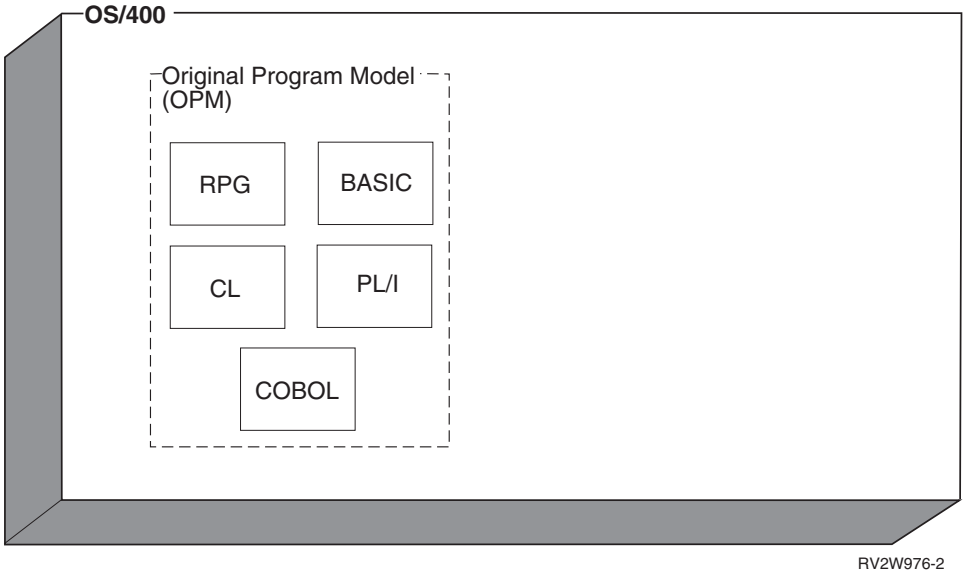
ในการสร้างโปรแกรม นักพัฒนาแอ็พพลิเคชันสำหรับเซิร์ฟเวอร์ iSeries จะป้อนซอร์สโค้ดลงในซอร์สไฟล์แล้วจึงคอมไพล์ไฟล์เหล่านั้น. ถ้าการคอมไพล์นั้นสำเร็จ ก็จะได้โปรแกรมอ็อบเจ็กต์ (Program Object). กลุ่มของฟังก์ชัน, กระบวนการ และกฎต่างๆจะถูกสร้างขึ้นโดย OS/400 เพื่อใช้ในการสร้าง และรันโปรแกรมที่เรียกว่า **Original Program Model (OPM)**.

เมื่อ OPM คอมไพเลอร์สร้างโปรแกรมอ็อบเจ็กต์ จะมีการสร้างโค้ดเพิ่มเติมขึ้น. โค้ดเหล่านี้จะให้ค่าเริ่มต้นแก่ตัวแปรของโปรแกรมและสร้างโค้ดที่จำเป็นสำหรับกระบวนการพิเศษที่ภาษาบางภาษาต้องการ. กระบวนการพิเศษนั้นรวมถึงการอินพุตค่าพารามิเตอร์ที่โปรแกรมคาดว่าจะต้องมีด้วย. เมื่อโปรแกรมเริ่มทำงานโค้ดพิเศษเหล่านั้นก็จะกลายเป็นจุดเริ่มต้น (entry point) ของโปรแกรม.

โปรแกรมจะถูกเรียกทำงานเมื่อ OS/400 ตรวจพบคำสั่งเรียก (Call). ซึ่งการเรียกโปรแกรมอื่นในช่วงรันไทม์ เราเรียกว่า **Dynamic Program Call**. รีซอร์สที่ Dynamic Program Call ต้องการมีความสำคัญมาก. นักพัฒนามักจะออกแบบแอ็พพลิเคชันให้มีโปรแกรมขนาดใหญ่เป็นจำนวนน้อย เพื่อลดจำนวนของ Dynamic Program Call ให้มีจำนวนน้อยที่สุด

รูปที่ 4 แสดงถึงความสัมพันธ์ระหว่าง OPM กับระบบปฏิบัติการ จะเห็นว่ามี อาร์พีจี, โคบอล, เบสิก, CL และ PL/I ทำงานอยู่ในโมเดลนี้.

ขอบเขตที่เส้นประในรูปแสดงถึงขอบเขตของ OPM ซึ่งแสดงให้เห็นว่า OPM เป็นส่วนหนึ่งของ OS/400. นั่นหมายความว่าฟังก์ชันหลายฟังก์ชันที่ถูกสร้างจากผู้พัฒนาคอมไพเลอร์จะถูกเก็บอยู่ในระบบปฏิบัติการ. ทำให้โปรแกรมที่เขียนด้วยภาษาหนึ่งสามารถที่จะเรียกโปรแกรมที่เขียนด้วยภาษาอื่นได้. เช่น แอ็พพลิเคชันที่เขียนด้วยภาษาอาร์พีจีมีโปรแกรมที่เขียนด้วยภาษา CL รวมอยู่ด้วย เพื่อใช้สำหรับการเขียนทับไฟล์, การจัดการเกี่ยวกับสตริง หรือเพื่อส่งแมสเสจ เป็นต้น.



รูปที่ 4. แสดงความสัมพันธ์ของ OPM กับ OS/400

คุณสมบัติสำคัญของ OPM

รายชื่อดังต่อไปนี้ระบุถึงลักษณะหลักของ OPM:

- เหมาะสำหรับโปรแกรมที่เขียนด้วยภาษาอาร์พีจีและโคบอลรุ่นเก่า.

  OPM เหมาะสำหรับโปรแกรมภาษาอาร์พีจีและโคบอลรุ่นเก่าที่มีขนาดใหญ่และมีฟังก์ชันการทำงานมากมาย.

- การรวมโมดูลแบบไดนามิก (Dynamic Bindings)

  เมื่อโปรแกรม A ต้องการเรียกโปรแกรม B ก็เพียงแต่เรียกธรรมดาเท่านั้น. การเรียกโปรแกรมแบบไดนามิกนั้นง่ายและมีความสามารถสูง. ในขณะรันระบบปฏิบัติการจะกำหนดตำแหน่งของโปรแกรม B และตรวจสอบว่าผู้ใช้มีสิทธิในการใช้โปรแกรมนี้หรือไม่.

  โปรแกรม OPM มี จุดเริ่มต้นเพียงจุดเดียว ในขณะที่แต่ละโพรซีเดอร์ของโปรแกรม ILE จะสามารถเป็นจุดเริ่มต้นได้.

- จำกัดการแบ่งใช้ข้อมูล (Limited data sharing)

  ในโปรแกรม OPM นั้น โพรซีเดอร์ที่อยู่ภายในจะต้องมีการแบ่งใช้ตัวแปรให้กับโปรแกรมทั้งโปรแกรม ในขณะที่โปรแกรม ILE จะมีตัวแปรสำหรับทำงานเฉพาะในแต่ละโพรซีเดอร์เท่านั้น.

ลักษณะของ Extended Program Model

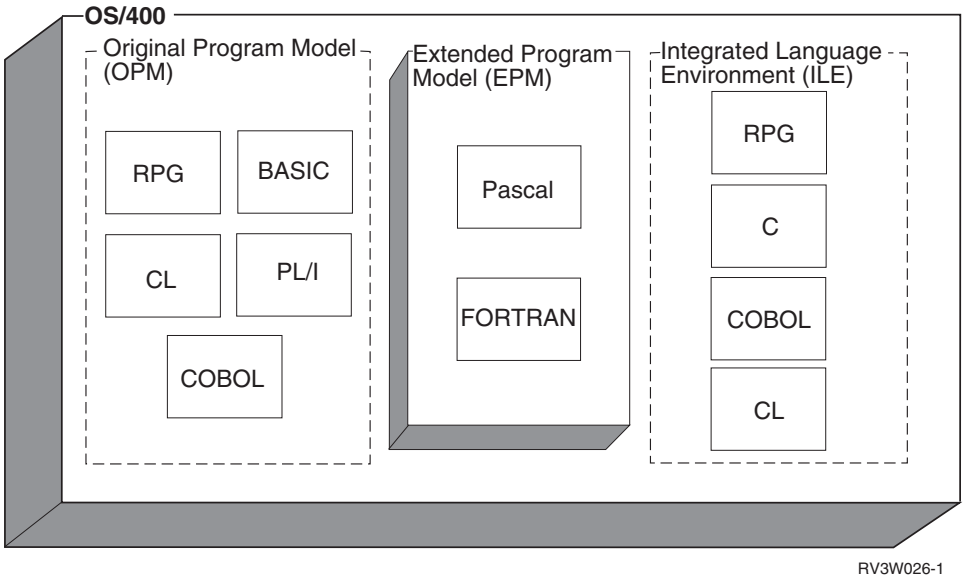
OPM ยังคงตอบสนองความต้องการที่จำเป็น. แต่อย่างไรก็ตาม OPM ก็ไม่ได้ให้การสนับสนุนโดยตรงต่อ โพรซีเดอร์ที่มาจากบางภาษา เช่น ในภาษาซี. **โพรซีเดอร์**คือกลุ่มคำสั่งแบบ Self-Contained High-Level Language (HLL)ที่ทำงานเฉพาะอย่างใดอย่างหนึ่งแล้วรีเทิร์นกลับไปยังผู้เรียก. แต่ละภาษามีการกำหนดชื่อที่แตกต่างกัน. เช่น ในภาษาซี โพรซีเดอร์จะถูกเรียกว่า ฟังก์ชัน.

เพื่อยอมให้ภาษาที่กำหนดให้การเรียกโพรซีเดอร์ระหว่าง Compilation Unit หรือภาษาที่กำหนดโพรซีเดอร์ที่มีตัวแปรภายในสามารถรันบน iSeries ได้ จึงมีการพัฒนา OPM. โดยเรียกส่วนที่พัฒนาขึ้นใหม่นี้ว่า **Extended Program Model (EPM)**. ก่อนการใช้ ILE EPM ถูกใช้เป็นทางแก้ปัญหาแบบชั่วคราวสำหรับภาษาเชิงโพรซีเดอร์ เช่น Pascal และ C.

ซึ่งเซิร์ฟเวอร์ iSeries ไม่สนับสนุนการใช้คอมไพเลอร์ EPM อีกต่อไป.

ลักษณะของ Integrated Language Environment

ดังรูป รูปที่ 5 ในหน้า 10 แสดง ILE ที่ถูกรวมเข้ากับ OS/400 เช่นเดียวกับ OPM. และยังมีรูปแบบของการสนับสนุนภาษาประเภท procedure-based เหมือนกับ EPM แต่มีความสมบูรณ์มากกว่า. ภาษา ILE ยังถูกออกแบบมาเพื่อใช้กับภาษารุ่นเก่าเช่น โคบอลและอาร์พีจีและสำหรับการพัฒนาภาษาในอนาคต.



รูปที่ 5. แสดงความสัมพันธ์ระหว่าง OPM, EPM และ ILE กับ OS/400

คุณสมบัติสำคัญของภาษาเชิงโพรซีเดอร์

ภาษาประเภท Procedure-based มีลักษณะพื้นฐานดังนี้:

- มีตัวแปรแบบโลคอล (locally scoped variable)

  ตัวแปรแบบโลคอลจะเป็นที่รู้จักเฉพาะภายในโพรซีเดอร์เท่านั้น. ความเท่าเทียมกันของตัวแปรแบบโลคอล คือการกำหนดตัวแปร 2 ตัวที่มีชื่อซ้ำกันแต่อ้างถึงข้อมูล 2 ส่วนที่อยู่แยกกันได้. เช่น ตัวแปร COUNT อาจมีความยาว 4 ดิจิตในรูทีนย่อย CALCYR และมีความยาว 6 ดิจิต ในรูทีนย่อย CALDAY.

  ตัวแปรแบบโลคอลมีประโยชน์มากเมื่อเขียนรูทีนย่อยที่ต้องถูกก็อปปี้ไปยังโปรแกรมหลายๆ โปรแกรมที่แตกต่างกัน. ถ้าไม่มีตัวแปรแบบโลคอลโปรแกรมเมอร์จะต้องใช้เทคนิคอื่น เช่น การตั้งชื่อตัวแปรตามชื่อรูทีนย่อยนั้นๆ.

- ตัวแปรแบบอัตโนมัติ (automatic variable)

  เมื่อใดก็ตามที่เข้าสู่โพรซีเดอร์ ตัวแปรแบบอัตโนมัติจะถูกสร้างขึ้น. ขึ้น และจะถูกลบไปเมื่อออกจาก โพรซีเดอร์นั้น.

- ตัวแปรภายนอก (external variable)

  การมีข้อมูลภายนอกเป็นวิธีการหนึ่งในการแบ่งใช้ข้อมูลระหว่างโปรแกรม. ถ้าโปรแกรม A กำหนดให้ข้อมูลหนึ่งเป็นข้อมูลภายนอก โปรแกรม A สามารถ **export** export ข้อมูลนั้นให้กับโปรแกรมอื่นที่ต้องการใช้ข้อมูลนั้นด้วย. โปรแกรม D สามารถ **import** ข้อมูลนั้นได้โดยไม่ต้องมีโปรแกรม B และ C เข้ามาเกี่ยวข้อง. สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ imports และ exports สามารถดูได้ใน “โมดูลอ็อบเจ็กต์ (Module Object)” ในหน้า 14.

- มีจุดเริ่มต้น (entry points) หลายจุด

โปรแกรมโคบอลและอาร์พีจีมีจุดเริ่มต้นเพียงจุดเดียว.เดียว ในโปรแกรมโคบอลจุดเริ่มต้นจะอยู่ที่จุดเริ่มต้นของ PROCEDURE DIVISION. และในโปรแกรมอาร์พีจีจุดเริ่มต้นจะอยู่ที่เอาต์พุต First-Page(1P). ที่กล่าวมานี้เป็นโมเดลที่ OPM สนับสนุน.

แต่สำหรับภาษาเชิงโพรซีเดอร์จะมีลักษณะตรงกันข้ามคืออาจจะมีจุดเริ่มต้นได้หลายจุด. ตัวอย่างเช่น ในโปรแกรมภาษาซี อาจประกอบด้วยรูทีนย่อยที่ถูกใช้โดยโปรแกรมอื่น. โพรซีเดอร์เหล่านี้สามารถถูก export ข้อมูลที่เกี่ยวข้องไปยังโปรแกรมอื่นที่ต้องการใช้ข้อมูลนั้น.

ใน ILE โปรแกรมที่มีลักษณะเช่นนี้เรียกว่า **เซอร์วิสโปรแกรม**. โปรแกรมเหล่านี้สามารถรวมโมดูลที่มาจากภาษา ILE ใดๆ ก็ได้. เซอร์วิสโปรแกรมมีแนวคิดที่คล้ายกับ dynamic link libraries (DLLs) ใน Windows® หรือ OS/2®. เซอร์วิสโปรแกรมมีรายละเอียดที่พูดถึงมากขึ้นใน “เซอร์วิสโปรแกรม (Service Program)” ในหน้า 19.

- ความถี่ของการเรียก

  ภาษาเชิงโพรซีเดอร์จะมีการเรียก (Call) เป็นจำนวนมาก. แม้ว่า EPM จะมีฟังก์ชัน เพื่อลดโอเวอร์เฮดของการเรียก แต่การเรียกระหว่างยูนิต ยังคงมีค่าที่สูงมาก. ใน ILE จึงได้มีการปรับปรุงการเรียกชนิดนี้ไปอย่างเห็นได้ชัด.

บทที่ 2. แนวคิด ILE ขั้นพื้นฐาน

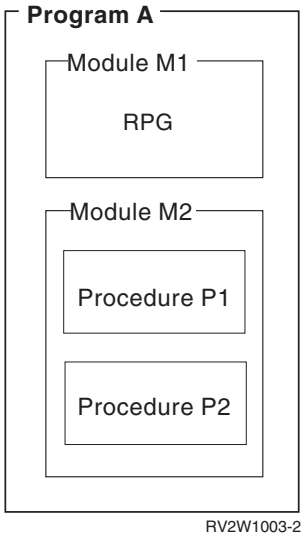
ตารางที่ 1แสดงการเปรียบเทียบระหว่าง originalprogram model (OPM) กับ Integrated Language Environment (ILE). ในบทนี้จะอธิบายถึงความเหมือนและความแตกต่างของโมเดลทั้งสองอย่างคร่าวๆ ตามตาราง.

ตารางที่ 1. ความเหมือนและความแตกต่างระหว่าง OPM และ ILE

OPM	ILE
โปรแกรม	โปรแกรมประเภทเซอร์วิส
การคอมไพล์ทำให้เกิดไฟล์ที่รันได้	การคอมไพล์ทำให้เกิดโมดูลอ็อบเจ็กต์ที่รันไม่ได้
คอมไพล์, รัน	คอมไพล์, รวมโมดูล, รัน
Run unit จำลองตัวเองสำหรับแต่ละภาษา	Activation Groups
การเรียกโปรแกรมเป็นแบบไดนามิก	การเรียกโปรแกรมเป็นแบบไดนามิก
	Static procedure call
เน้นการเขียนโปรแกรมแบบใช้ภาษาเดียว	เน้นการเขียนโปรแกรมหลายภาษา
การจัดการข้อผิดพลาดแบบผูกติดกับภาษา	การจัดการข้อผิดพลาดแบบทั่วไป
	การจัดการข้อผิดพลาดแบบผูกติดกับภาษา
ดีบักเกอร์สำหรับ OPM	ดีบักเกอร์ในระดับ source

โครงสร้างของโปรแกรม ILE

โปรแกรม ILE จะมีอย่างน้อย 1 โมดูล. และในหนึ่งโมดูลจะประกอบด้วยอย่างน้อย 1 โพรซีเดอร์เช่นกัน (รูปที่ 6 ในหน้า 14).



รูปที่ 6. แสดงโครงสร้างของโปรแกรม ILE

โพรซีเดอร์ (Procedure)

*โพรซีเดอร์*คือ กลุ่มของคำสั่งภาษาระดับสูง ที่ทำงานเฉพาะอย่างใดอย่างหนึ่งแล้วส่งคืนไปยังผู้เรียก (Caller).ตัวอย่างเช่น ฟังก์ชันของ ILE C คือโพรซีเดอร์ของ ILE.

โมดูลอ็อบเจ็กต์ (Module Object)

โมดูลอ็อบเจ็กต์ คือ*อ็อบเจ็กต์ที่ไม่สามารถรันได้* เป็นผลจากการคอมไพล์ของ ILE. สัญลักษณ์ที่ใช้แทนโมดูลอ็อบเจ็กต์คือ *MODULE. โมดูลอ็อบเจ็กต์เป็นส่วนประกอบพื้นฐานในการสร้างอ็อบเจ็กต์ที่สามารถรันได้ (Runnable Object). นี่คือความแตกต่างที่เห็นได้ชัดระหว่าง ILE และ OPM. ผลลัพธ์ที่ได้จากตัวคอมไพล์ของ OPM เป็นโปรแกรมที่สามารถ*รันได้เอง* (Runnable Program).

ในโมดูลอ็อบเจ็กต์จะประกอบด้วยโพรซีเดอร์และค่ากำหนดของตัวข้อมูล (Data Item). อ็อบเจ็กต์ของ ILE สามารถที่จะเข้าถึงโพรซีเดอร์หรือตัวข้อมูลของโมดูลอื่นได้โดยตรง. สำหรับรายละเอียดของการเขียนโปรแกรมให้โพรซีเดอร์และตัวข้อมูลยอมให้อ็อบเจ็กต์ของ ILE อื่นเข้าถึงได้ สามารถดูได้จากหนังสือ ILE HLL programmer's guides.

ILE RPG, ILE COBOL, ILE C, และ ILE C++ ทั้งหมดมีแนวคิดทั่วไปดังต่อไปนี้:

- Exports

  Export คือชื่อของโพรซีเดอร์หรือตัวข้อมูลที่กำหนดไว้ในโมดูลอ็อบเจ็กต์ เพื่อให้อ็อบเจ็กต์ของ ILE อื่นมาใช้โพรซีเดอร์หรือหน่วยข้อมูลเหล่านั้นได้. Export จะถูกกำหนดโดยชื่อและชนิดของโพรซีเดอร์ หรือข้อมูล.

  เราสามารถเรียก Export ได้อีกชื่อหนึ่งว่า **Definition**.

- Imports

  Import คือชื่อของโพรซีเดอร์หรือหน่วยข้อมูลที่ถูกอ้างถึงแต่ไม่ถูกสร้างไว้ในโมดูลอ็อบเจ็กต์นั้น. import ถูกกำหนดโดยชื่อและชนิดของโพรซีเดอร์หรือหน่วยข้อมูล.

  เราสามารถเรียก import ได้อีกชื่อหนึ่งว่า **Reference**.

โมดูลอ็อบเจ็กต์เป็นส่วนประกอบพื้นฐานของอ็อบเจ็กต์ของ ILE ที่รันได้. ดังนั้นเมื่อโมดูลอ็อบเจ็กต์ถูกสร้างขึ้น อาจมีส่วนอื่นที่อาจจะถูกสร้างขึ้น ดังนี้:

- ข้อมูลสำหรับการดีบัก (debug data)

  ข้อมูลสำหรับการดีบัก คือข้อมูลที่สำคัญต่อการดีบัก และรันอ็อบเจ็กต์ของ ILE. ข้อมูลส่วนนี้เป็นตัวเลือก (optional) จะมีหรือไม่ก็ได้.

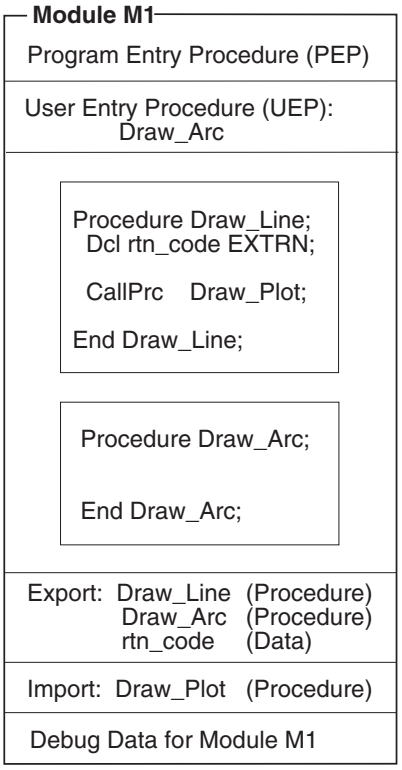
- Program entry procedure (PEP)

  Program Entry Procedure คือโค้ดที่คอมไพเลอร์สร้างขึ้นให้เป็นจุดเริ่มต้นของโปรแกรม ILE บน Dynamic Program Call. จะคล้ายกับโค้ดที่เป็นจุดเริ่มต้น (entry point) ในโปรแกรม OPM.

- User entry procedure (UEP)

  User Entry Procedure, คือตำแหน่งเป้าหมายของ Dynamic Program Call ซึ่งถูกเขียนขึ้นโดยโปรแกรมเมอร์. เป็นโพรซีเดอร์ที่รับช่วงต่อจาก PEP. ตัวอย่างเช่น ฟังก์ชัน main() ของโปรแกรมภาษา C เป็น UEP ของโปรแกรมใน ILE.

รูปที่ 7 ในหน้า 16 แสดงถึงแนวคิดของโมดูลอ็อบเจ็กต์. ในตัวอย่างนี้โมดูลอ็อบเจ็กต์ M1 ได้ export โพรซีเดอร์ คือ Draw_Line และ Draw_Arc และ export หน่วยข้อมูล rtn_code. นอกจากนี้โมดูล อ็อบเจ็กต์ M1 ยัง import โพรซีเดอร์ Draw_Plot. จะเห็นว่าโมดูลอ็อบเจ็กต์นี้ มีทั้ง PEP, UEP (โพรซีเดอร์ Draw_Arc) และ ข้อมูลสำหรับการดีบัก.



รูปที่ 7. แสดงแนวคิดเกี่ยวกับโมดูล

คุณลักษณะของอ็อบเจ็กต์ *MODULE:

- อ็อบเจ็กต์ *MODULE เป็นผลลัพธ์ที่ได้จากคอมไพเลอร์ ILE.
- เป็นส่วนประกอบพื้นฐานของอ็อบเจ็กต์ ILE ที่รันได้.
- เป็นอ็อบเจ็กต์ที่รันด้วยตัวเองไม่ได้.
- อาจมี PEP กำหนดอยู่.
- ถ้ามีการกำหนด PEP, ก็จะมีการกำหนด UEP ด้วย.
- สามารถ export ชื่อของโพรซีเดอร์และหน่วยข้อมูล.
- สามารถ import ชื่อของโพรซีเดอร์และหน่วยข้อมูล.
- สามารถมีข้อมูลสำหรับการดีบักได้.

โปรแกรม ILE

โปรแกรม ILE มีคุณสมบัติบางอย่างที่เหมือนกับโปรแกรม OPM ดังนี้:

- โปรแกรมรับการควบคุมผ่าน Dynamic Program Call.
- โปรแกรมมีจุดเริ่มต้น (Entry Point) เพียงจุดเดียว.
- สัญลักษณ์แทนโปรแกรมในระบบคือ *PGM.

โปรแกรม ILE มีคุณสมบัติดังต่อไปนี้ที่ไม่มีในโปรแกรม OPM:

- โปรแกรม ILE สร้างจากโมดูลอ็อบเจ็กต์หนึ่งอ็อบเจ็กต์หรือมากกว่า.

- อาจประกอบด้วยโมดูลมากกว่าหนึ่งโมดูลที่มี PEP.
- คุณเป็นผู้ควบคุมเหนือโมดูล 's PEP ซึ่งถูกใช้เป็น PEP สำหรับอ็อบเจ็กต์โปรแกรม ILE.

  เมื่อมีการระบุคำสั่ง Create Program (CRTPGM), พารามิเตอร์ ENTMOD อนุญาตให้คุณเลือกโมดูลซึ่งประกอบด้วย PEP เป็นโปรแกรม 's entry point.

  ส่วน PEP ของโมดูลที่ไม่ได้ถูกเลือกให้เป็นจุดเริ่มต้นของโปรแกรมจะไม่ถูกสนใจ. แต่โพรซีเดอร์และหน่วยข้อมูลอื่นๆ ในโมดูลนั้นยังสามารถใช้ได้. เฉพาะ PEP เท่านั้นที่ถูกละเว้น.

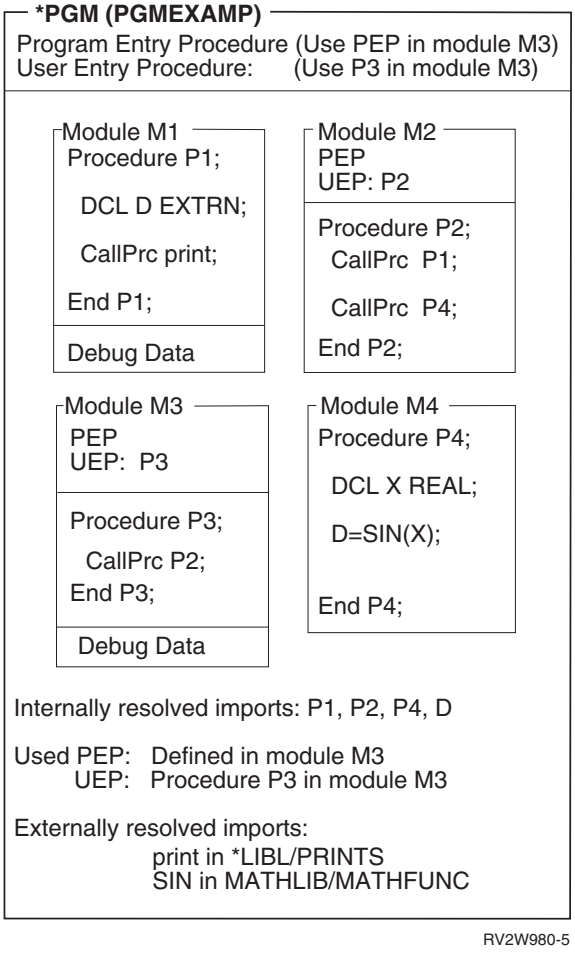
เมื่อ dynamic program call ถูกกำหนดลงในโปรแกรม ILE, โมดูล 's PEP ที่ถูกเลือกที่เวลา program-creation จะถูกควบคุม. จากนั้น PEP จะเรียก UEP ที่เกี่ยวข้องกันอีกต่อหนึ่ง.

เมื่อโปรแกรม ILE ถูกสร้างขึ้น มีเพียงโพรซีเดอร์ของโมดูลที่มีข้อมูลสำหรับการดีบัก (Debug Data) เท่านั้นที่จะถูกดีบักโดยโปรแกรมดีบักเกอร์ของ ILE. ข้อมูลสำหรับการดีบักจะไม่ส่งผลกระทบต่อประสิทธิภาพในเวลารันโปรแกรม ILE.

รูปที่ 8 ในหน้า 18 แสดงแนวคิดของ ILE โปรแกรมอ็อบเจ็กต์. เมื่อโปรแกรม PGMEXAMP ถูกเรียก PEP ที่ถูกกำหนดไว้ในโมดูลอ็อบเจ็กต์ M3 จะถูกใช้. ในขณะที่โมดูล M2 ก็มี PEP เช่นกัน แต่ PEP ของ M2 จะไม่มีการเรียก.

ในโปรแกรมตัวอย่างนี้ มีเพียง 2 โมดูลเท่านั้นที่มีข้อมูลสำหรับการดีบัก คือ M1 และ M3. ส่วนโพรซีเดอร์ในโมดูล M2 และ M4 จะไม่สามารถดีบัก โดยการใช้ ILE ดีบักเกอร์ ได้.

โพรซีเดอร์ที่ถูก import ในโปรแกรมนี้ คือ `print` และ `SIN` จะถูก resolve ไปยังโพรซีเดอร์ที่ถูก export จากเซอร์วิสโปรแกรม ชื่อ PRINTS และ MATHFUNC ตามลำดับ.



รูปที่ 8. แสดงแนวคิดเกี่ยวกับโปรแกรม ILE

คุณสมบัติของ ILE อ็อบเจ็กต์ *PGM:

- สามารถนำโมดูลตั้งแต่ 1 โมดูลขึ้นไปจากภาษา ILE ใดๆ มาก็อปปี้เพื่อทำเป็นอ็อบเจ็กต์ *PGM.
- ผู้ที่สร้างโปรแกรมจะเป็นผู้กำหนดว่า PEP ของ 's PEP โมดูลใด จะกลายมาเป็น PEP สำหรับโปรแกรม.
- สำหรับ dynamic program call, PEP ของ 's โมดูลที่ถูกเลือกเป็น PEP สำหรับโปรแกรมจะเป็นตัวควบคุมการรัน.
- UEP ที่สัมพันธ์กับ PEP ที่เลือกเป็น entry point ของ user's สำหรับโปรแกรม.
- ไม่สามารถ export ชื่อของโพรซีเดอร์และตัวข้อมูลจากโปรแกรมได้.
- ชื่อของโพรซีเดอร์และตัวข้อมูลถูก import จากโมดูลและเซอร์วิสโปรแกรมได้แต่ไม่ใช่จากโปรแกรมอ็อบเจ็กต์. สำหรับรายละเอียดของเซอร์วิสโปรแกรม ดูได้ในหัว "เซอร์วิสโปรแกรม (Service Program)" ในหน้า 19.
- โมดูลสามารถมีข้อมูลสำหรับดีบักได้.
- โปรแกรมเป็นอ็อบเจ็กต์ที่สามารถรันได้ (Runnable Object)

เซอร์วิสโปรแกรม (Service Program)

เซอร์วิสโปรแกรม คือกลุ่มของโพรซีเดอร์ที่รันได้และข้อมูลที่เซอร์วิสโปรแกรมอื่นหรือโปรแกรม ILE อื่น สามารถเรียกได้โดยตรง . ในหลายกรณี เซอร์วิสโปรแกรมจะเหมือนกับซับรูทีนไลบรารี หรือไลบรารีของโพรซีเดอร์.

เซอร์วิสโปรแกรม มีการให้บริการทั่ว ๆ ไปซึ่งอ็อบเจ็กต์ ILE อื่นอาจต้องการ ดังนั้นจึงมีชื่อว่า เซอร์วิสโปรแกรม. ตัวอย่างของชุดเซอร์วิสโปรแกรมที่ OS/400 เตรียมไว้ได้แก่ โพรซีเดอร์แบบรันไทม์สำหรับภาษาต่างๆ. ซึ่งโพรซีเดอร์เหล่านี้จะมีหลายประเภท เช่น โพรซีเดอร์ที่เกี่ยวกับคณิตศาสตร์ และโพรซีเดอร์ที่เกี่ยวกับอินพุต/เอาท์พุตทั่ว ๆ ไป.

พับลิกอินเตอร์เฟสของเซอร์วิสโปรแกรมหนึ่ง ๆ จะประกอบด้วยชื่อของโพรซีเดอร์และชื่อตัวข้อมูล ที่ถูก export ไว้และอ็อบเจ็กต์ ILE อื่นสามารถเข้าถึงได้. มีเพียงส่วนที่ถูก export จากโมดูลอ็อบเจ็กต์ที่ประกอบกันเป็นเซอร์วิสโปรแกรมเท่านั้นที่สามารถ export จากเซอร์วิสโปรแกรมได้.

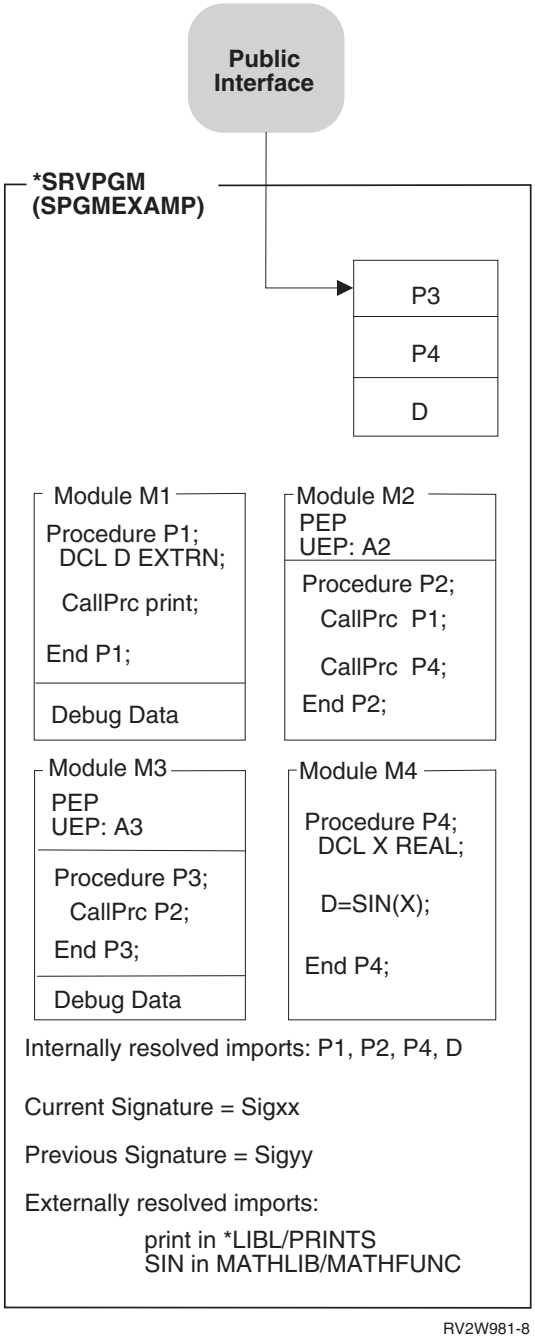
โปรแกรมเมอร์สามารถกำหนดได้ว่า จะให้โพรซีเดอร์หรือข้อมูลใด เป็นที่รู้จักของอ็อบเจ็กต์ ILE อื่น . ดังนั้น เซอร์วิสโปรแกรมสามารถที่จะซ่อน หรือตั้งค่าแบบ private ให้กับโพรซีเดอร์หรือตัวข้อมูลเพื่อไม่ให้อ็อบเจ็กต์ ILE อื่นใช้งานได้.

มีทางเป็นไปได้ที่จะอัพเดตเซอร์วิสโปรแกรม โดยไม่ต้องสร้างโปรแกรมหรือเซอร์วิสโปรแกรม (ที่ใช้เซอร์วิสโปรแกรมที่เราอัพเดต) ใหม่อีกครั้ง . โดยการควบคุมให้การเปลี่ยนแปลงของเซอร์วิสโปรแกรม เข้ากันได้ (compatible) กับระบบเดิม.

วิธีการที่ ILE ให้คุณควบคุมการเปลี่ยนแปลงที่เข้ากันได้ คือการใช้ **Binder Language**. โดย binder language จะให้คุณกำหนดรายชื่อของโพรซีเดอร์และตัวข้อมูลที่จะถูก export. **Signature** จะถูกสร้างขึ้นจากชื่อของโพรซีเดอร์และตัวข้อมูล และจากคำสั่งที่กำหนดไว้ใน binder language. เพื่อที่จะสร้างการเปลี่ยนแปลงที่เข้ากันได้ให้กับเซอร์วิสโปรแกรม ชื่อใหม่ของโพรซีเดอร์และตัวข้อมูลจะถูกรวมเข้าไปที่ส่วนท้ายรายการที่ถูก export. สำหรับข้อมูลเพิ่มเติม เกี่ยวกับลายเซ็น, ภาษาการเชื่อมโยง, และการป้องกันการลงทุนของลูกค้า ' ของคุณ ในเซอร์วิสโปรแกรม, ดู “ภาษา Binder” ในหน้า 90.

รูปที่ 9 ในหน้า 20 แสดงถึงลักษณะของเซอร์วิสโปรแกรม. เป็นที่สังเกตว่า โมดูลที่รวมกันเป็นเซอร์วิสโปรแกรม เป็นโมดูลชุดเดียวกันกับที่รวมกันเป็นโปรแกรมอ็อบเจ็กต์ ILE ชื่อ PGMEXAMP ในรูปที่ 8 ในหน้า 18. SPGMEXAMP จะบรรจุชื่อของโพรซีเดอร์ P3 และ P4 หลังจากเซอร์วิสโปรแกรมการเปลี่ยนแปลง Signature ปัจจุบันคือ Sigyy. ไม่เพียงแต่จะบรรจุชื่อโพรซีเดอร์ P3 และ P4 เท่านั้น แต่ยังมีชื่อตัวข้อมูล D อยู่ด้วย โปรแกรม ILE หรือเซอร์วิสโปรแกรมอื่น ที่ใช้โพรซีเดอร์ P3 หรือ P4 จะไม่ต้องถูกสร้างใหม่.

แม้ว่าโมดูลในเซอร์วิสโปรแกรมอาจมี PEP หลายจุด แต่ PEP เหล่านั้นจะถูกละเลย. เนื่องจากเซอร์วิสโปรแกรมไม่มี PEP ของตัวเอง. ดังนั้นเซอร์วิสโปรแกรมจะไม่สามารถถูกเรียกแบบไดนามิกได้.



รูปที่ 9. แสดงแนวคิดเกี่ยวกับเซอร์วิสโปรแกรม ILE

คุณสมบัติของอ็อบเจ็กต์ *SRVPG:

- โมดูลจากภาษา ILE ใดๆ สามารถถูกก็อปปี้ให้เป็นอ็อบเจ็กต์ *SRVPGM ได้.
- ไม่มี PEP ที่เป็นของเซอร์วิสโปรแกรม. เพราะว่าเซอร์วิสโปรแกรมไม่มี PEP การเรียกเซอร์วิสโปรแกรมแบบไดนามิกจึงทำไม่ได้. PEP ของโมดูล 's จะถูกข้าม ไป.
- โปรแกรม ILE หรือเซอร์วิสโปรแกรมอื่นสามารถใช้ส่วนที่เป็น export ของเซอร์วิสโปรแกรมหนึ่งได้โดยใช้พับลิกอินเตอร์เฟสเป็นตัวกำหนด.
- Signature ถูกสร้างจากชื่อของโพรซีเดอร์และตัวข้อมูลที่ถูก export จากเซอร์วิสโปรแกรม.

- เซอร์วิสโปรแกรมสามารถถูกแทนที่ได้โดยไม่ส่งผลกระทบต่อโปรแกรม ILE หรือเซอร์วิสโปรแกรมที่ใช้งานมัน ตราบใดที่ signature ก่อนหน้านี้ยังถูกสนับสนุนอยู่.
- โมดูลสามารถมีข้อมูลสำหรับดีบักได้.
- เซอร์วิสโปรแกรม คือที่รวมของตัวข้อมูลและโพรซีเดอร์ที่รันได้.
- Weak data สามารถถูก export ไปยัง Activation Group เท่านั้น. และไม่สามารถเป็นส่วนหนึ่งของพับลิกอินเตอร์เฟส ซึ่ง export จากเซอร์วิสโปรแกรมได้. สำหรับข้อมูลเกี่ยวกับ weak data ดูที่หัวข้อ export ในเรื่องแนวคิดเกี่ยวกับ Import และ Export ใน “แนวคิดในการอิมพอร์ตและเอ็กซ์พอร์ต” ในหน้า 88.

Binding Directory

Binding Directory ประกอบด้วยชื่อของโมดูลและเซอร์วิสโปรแกรม ที่คุณอาจจะต้องการในการสร้างโปรแกรม ILE หรือเซอร์วิสโปรแกรม. ชื่อของโมดูลและเซอร์วิสโปรแกรมที่ปรากฏอยู่ใน binding directory จะถูกใช้ก็ต่อเมื่อโมดูลหรือเซอร์วิสโปรแกรมเหล่านั้นมีตัว export ที่เป็นที่ต้องการของ การ import ที่ resolve ไม่ได้. binding directory เป็นอ็อบเจ็กต์ของระบบแบบหนึ่ง มีสัญลักษณ์ในระบบคือ *BNDDIR.

Binding directory เป็นส่วนที่เป็นอ็อพชัน. เหตุผลในการใช้ Binding Directory ก็คือความสะดวกและขนาดของโปรแกรม.

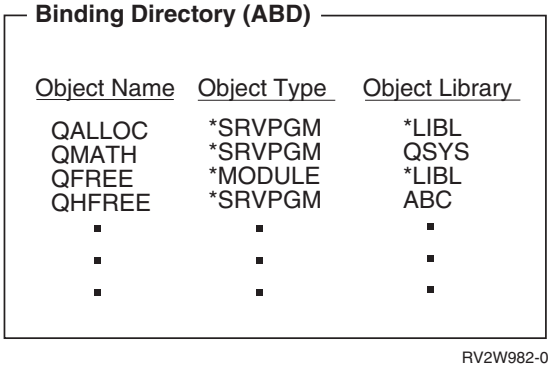
- Binding directory ให้ความสะดวกแก่คุณในการรวบรวมโมดูลหรือ เซอร์วิสโปรแกรม ที่คุณอาจต้องใช้เพื่อจะสร้างโปรแกรม ILE หรือ เซอร์วิสโปรแกรม. ตัวอย่าง เช่น ใน Binding Directory หนึ่งๆ อาจประกอบด้วยโมดูลและ เซอร์วิสโปรแกรม ทั้งหมดที่เป็นฟังก์ชันทางคณิตศาสตร์. ถ้าคุณต้องการใช้งานฟังก์ชันเหล่านั้นเพียงบางส่วน คุณก็เพียงระบุชื่อ Binding Directory โดยไม่ต้องระบุโมดูลทุกโมดูลหรือโปรแกรมทุกโปรแกรมที่คุณต้องการ.

  หมายเหตุ: หาก Binding Directory ยิ่งมีโมดูลหรือเซอร์วิสโปรแกรม มาก ก็จะทำให้เวลาในการรวมโปรแกรมยิ่งนานขึ้น. ดังนั้นใน Binding Directory ควรมีแต่โมดูลหรือเซอร์วิสโปรแกรมที่จำเป็นเท่านั้น.
- Binding directory สามารถลดขนาดของโปรแกรมลงได้ เพราะว่าคุณไม่ต้องใส่โมดูลหรือเซอร์วิสโปรแกรมที่คุณไม่ได้ใช้งาน.

มีข้อจำกัดน้อยมากในการป้อนข้อมูลเข้าไปใน Binding Directory. คุณสามารถใส่ชื่อของโมดูลหรือเซอร์วิสโปรแกรมลงใน Binding Directory ได้ แม้ว่าจะไม่มีอ็อบเจ็กต์นั้นก็ตาม.

สำหรับรายการคำสั่ง CL ที่ใช้กับ binding directory อยู่ในภาคผนวก ค. คำสั่ง CL ที่ใช้กับอ็อบเจ็กต์ของ ILE ในหน้า ภาคผนวก C, “คำสั่ง CL ที่ใช้กับอ็อบเจ็กต์ ILE”, ในหน้า 223.

รูปที่ 10 ในหน้า 22 แสดงถึง แนวคิดของ binding directory.



รูปที่ 10. แสดงแนวคิดเกี่ยวกับ Binding directory

คุณลักษณะของ *BNDDIR object:

- เป็นวิธีที่สะดวกในการจัดกลุ่มชื่อของเซอร์วิสโปรแกรมและโมดูล ที่อาจจำเป็นต่อการสร้างโปรแกรม ILE หรือเซอร์วิสโปรแกรม.
- เนื่องจากข้อมูลที่ป้อนเข้าไปใน Binding Directory เป็นเพียงชื่อเท่านั้น ดังนั้นจึงไม่จำเป็นต้องมีอ็อบเจ็กต์อยู่ในระบบ.
- ชื่อของไลบรารีที่จะใช้งานได้ต้องมีชื่อเป็น *LIBL หรือเป็นไลบรารีเฉพาะ.
- อ็อบเจ็กต์ในรายการนี้เป็นอ็อพชัน. อ็อบเจ็กต์เหล่านั้นจะถูกใช้ก็ต่อเมื่อไม่สามารถ resolve การ import ได้และอ็อบเจ็กต์นั้นมี export ที่ตรงกับ import ที่ไม่ถูก resolve.

Binding Directory Processing

ระหว่างการเชื่อมโยง, การประมวลผลได้เกิดขึ้นตามลำดับดังนี้:

1. โมดูลทั้งหมดที่ถูกระบุบนพารามิเตอร์ MODULE ถูกทดสอบ. ตัวเชื่อมโยงจะพิจารณารายการของสัญญลักษณ์ที่ถูกนำเข้ามาและส่งออกไปโดยอ็อบเจ็กต์. หลังจากการทดสอบ, โมดูลถูกจำกัดขอบเขต, ในลำดับรายการ, ภายในโปรแกรมที่ ถูกสร้างขึ้น.
2. เซอร์วิสโปรแกรมทั้งหมดบนพารามิเตอร์ BNDSRVPGM ถูก ทดสอบในตามลำดับรายการ. เซอร์วิสโปรแกรมถูกจำกัดขอบเขตถ้ามีความจำเป็น ในการ resolve การ อิมปอร์ต.
3. binding directory ทั้งหมดบนพารามิเตอร์ BNDDIR ถูกประมวลผล ตามลำดับรายการ. รายชื่ออ็อบเจ็กต์ทั้งหมดใน binding directory เหล่านี้ ถูกทดสอบตามลำดับรายการ, แต่เพียงถูกจำกัดขอบเขตถ้าจำเป็นต้อง resolve การอิมปอร์ต . entry ที่ทำซ้ำใน binding directory จะถูกข้ามโดยทันที.
4. ในแต่ละโมดูลจะมีรายชื่อของ **อ็อบเจ็กต์ของระบบที่ อ้างถึง**. รายชื่อเหล่านี้ถือเป็นรายการปกติของ binding directory. อ็อบเจ็กต์ของระบบ ที่อ้างอิงจากโมดูลขอบเขตถูกประมวลผลตามลำดับ ดังนั้นอ็อบเจ็กต์ของระบบที่อ้างอิงทั้งหมด จากโมดูลแรกจะถูกประมวลผลก่อนอันดับแรก, จากนั้นจึงเป็นอ็อบเจ็กต์จาก โมดูลที่สอง, และต่อๆไป. รายชื่อของอ็อบเจ็กต์ใน binding directory เหล่านี้ ถูกทดสอบตามลำดับรายชื่อ, เท่าที่จำเป็นเท่านั้น, และจำกัดขอบเขตเท่าที่จำเป็น. การประมวลผลนี้จะทำอย่างต่อเนื่องตราบเท่าที่ unresolved import เกิดขึ้น, ถึงแม้ว่ามีการใช้ OPTION(*UNRSLVREF). หรือในอีกทางหนึ่ง, การประมวลผลอ็อบเจ็กต์จะหยุด เมื่อมีการ resolve อิมปอร์ตทั้งหมด.

ในขณะที่อ็อบเจ็กต์ถูกทดสอบ, ข้อความ CPD5D03, “Definition จะส่งสัญญลักษณ์ออกมา หลายครั้ง”, อาจถูกส่งสัญญาณแม้ว่าอ็อบเจ็กต์จะไม่ถูกจำกัดขอบเขตในโปรแกรม ที่กำลังถูกสร้างอยู่ก็ตาม.

หมายเหตุ ตามปกติโมดูลที่มีอิมปอร์ตจะไม่ปรากฏจาก ซอร์สโค้ดของโมดูล. คอมไพเลอร์จะใส่เพิ่มสิ่งเหล่านี้เพื่อปฏิบัติคุณลักษณะของภาษาต่างๆ ซึ่งต้องการการสนับสนุนของรันไทม์จากเซอร์วิสโปรแกรม. ใช้คำสั่ง DSPMOD DETAIL(*IMPORT) เพื่อดูอิมปอร์ตเหล่านี้.

เพื่อที่จะดูรายการอิมปอร์ตและเอ็กซปอร์ตของสัญญลักษณ์ สำหรับโมดูล หรือ เซอร์วิสโปรแกรม, ให้ดูที่ส่วนของ Binder Information Listing ของรายการ CRTPGM หรือ CRTSRVPGM DETAIL (*EXTENDED). มันจะแสดงรายการอ็อบเจ็กต์ที่ถูกทดสอบในระหว่างการ เชื่อมโยง.

โมดูล หรืออ็อบเจ็กต์เซอร์วิสโปรแกรมที่ถูกจำกัดขอบเขตอยู่ภายในโปรแกรม หรือเซอร์วิสโปรแกรมที่กำลังสร้างถูกกำหนดในส่วน Binder Information Listing ของรายการ CRTPGM หรือ CRTSRVPGM DETAIL(*EXTENDED). เมื่อใดที่มีการสร้างอ็อบเจ็กต์, ท่านสามารถใช้คำสั่ง DSPPGM หรือ DSPSRVPGM ระบุ DETAIL(*MODULE) ด้วยได้ เพื่อดูเขตจำกัดของอ็อบเจ็กต์ *MODULE, และ DETAIL(*SRVPGM) เพื่อดูรายการขอบเขตจำกัดของ อ็อบเจ็กต์ *SRVPGM.

คุณสามารถใช้คำสั่ง DSPMOD DETAIL(*REFSYSOBJ) เพื่อดูรายการของอ็อบเจ็กต์ของระบบที่อ้างอิง, ซึ่งก็คือ binding directory. binding directory เหล่านี้โดยทั่วไป จะประกอบด้วยชื่อของเซอร์วิสโปรแกรม APIs ที่ให้มากับระบบปฏิบัติการ หรือระบบสนับสนุนรันไทม์ภาษา. ตามวิธีนี้, โมดูลสามารถถูกจำกัดขอบเขต ในการสนับสนุนรันไทม์ภาษาของมัน และ APIs ของระบบโดยไม่จำเป็นต้องให้โปรแกรมเมอร์ ระบุคำสั่งพิเศษใดๆ.

การทำงานของ Binder

Binder มีลักษณะการทำงานบางอย่างคล้ายกับการทำงานของ linkage editor. โดย **Binder** โดย binder จะทำหน้าที่ import ชื่อของโพรซีเดอร์และตัวข้อมูลจากโมดูลที่กำหนดไว้. Binder จะพยายามหา export ที่ต้องการจากโมดูล เซอร์วิสโปรแกรม และ Binding Directory ที่กำหนด.

ในการสร้างโปรแกรม ILE หรือเซอร์วิสโปรแกรม Binder จะทำงานตามประเภทของการเชื่อมโยงดังนี้:

- การรวมโดยการก็อปปี้ (Bind by copy)

  ในการสร้างโปรแกรม ILE หรือเซอร์วิสโปรแกรม สิ่งต่อไปนี้จะถูกก็อปปี้:

  โมดูลที่กำหนดในโมดูลพารามิเตอร์

  โมดูลที่ถูกเลือกจาก Binding Directory ที่มี export สำหรับ import ที่ยังไม่ถูก resolve

  Physical address ของโพรซีเดอร์และตัวข้อมูลที่ต้องการในโมดูลที่ถูกสำเนา จะถูกสร้างขึ้นเมื่อมีการสร้างโปรแกรมหรือเซอร์วิสโปรแกรม.

  ตัวอย่างเช่นใน รูปที่ 9 ในหน้า 20, โพรซีเดอร์ P3 ในโมดูล M3 เรียกโพรซีเดอร์ P2 ในโมดูล M2. physical address ของโพรซีเดอร์ P2 ในโมดูล M2 ถูกทำให้เป็นที่รู้จักของโพรซีเดอร์ M3 ดังนั้นแอดเดรสเหล่านั้นจะถูกเข้าถึงได้โดยตรง.

- การรวมโดยการอ้างอิง (Bind by reference)

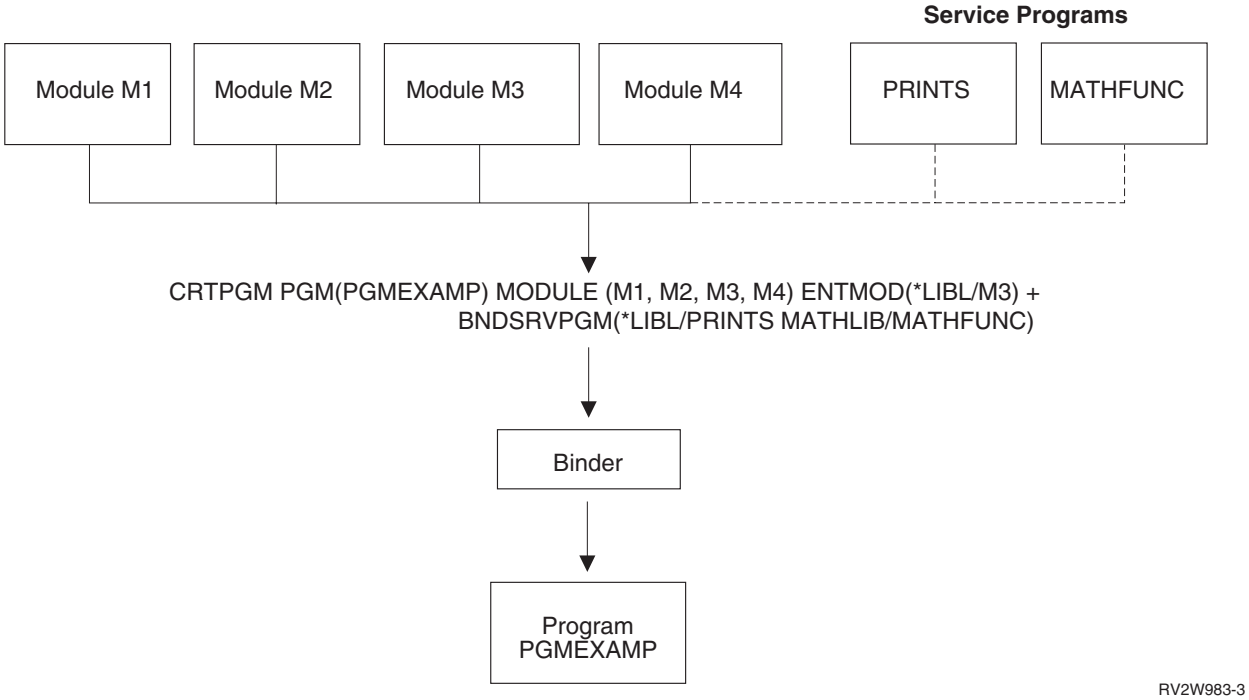
  Symbolic Link ที่ไปยังเซอร์วิสโปรแกรม ที่มี export สำหรับ import ที่ไม่ถูก resolve จะถูกเก็บในโปรแกรมหรือในเซอร์วิสโปรแกรมที่ถูกสร้างขึ้น. . Symbolic Link จะอ้างถึงเซอร์วิสโปรแกรมที่มี export. แล้วลิงก์นั้นจะแปลงเป็น Physical Address เมื่อโปรแกรมอ็อบเจ็กต์ที่มีเซอร์วิสโปรแกรมที่ถูกรวมนั้นถูกเรียกทำงาน.

  รูปที่ 9 ในหน้า 20 แสดงตัวอย่างของ symbolic link ไปที่ SIN ในเซอร์วิสโปรแกรม *MATHLIB/MATHFUNC. Symbolic Link จะถูกแปลงไปเป็น Physical Address เมื่อโปรแกรมอ็อบเจ็กต์ที่มีเซอร์วิสโปรแกรม SPGMEXAMP ที่ถูกรวมได้ถูกเรียกทำงาน.

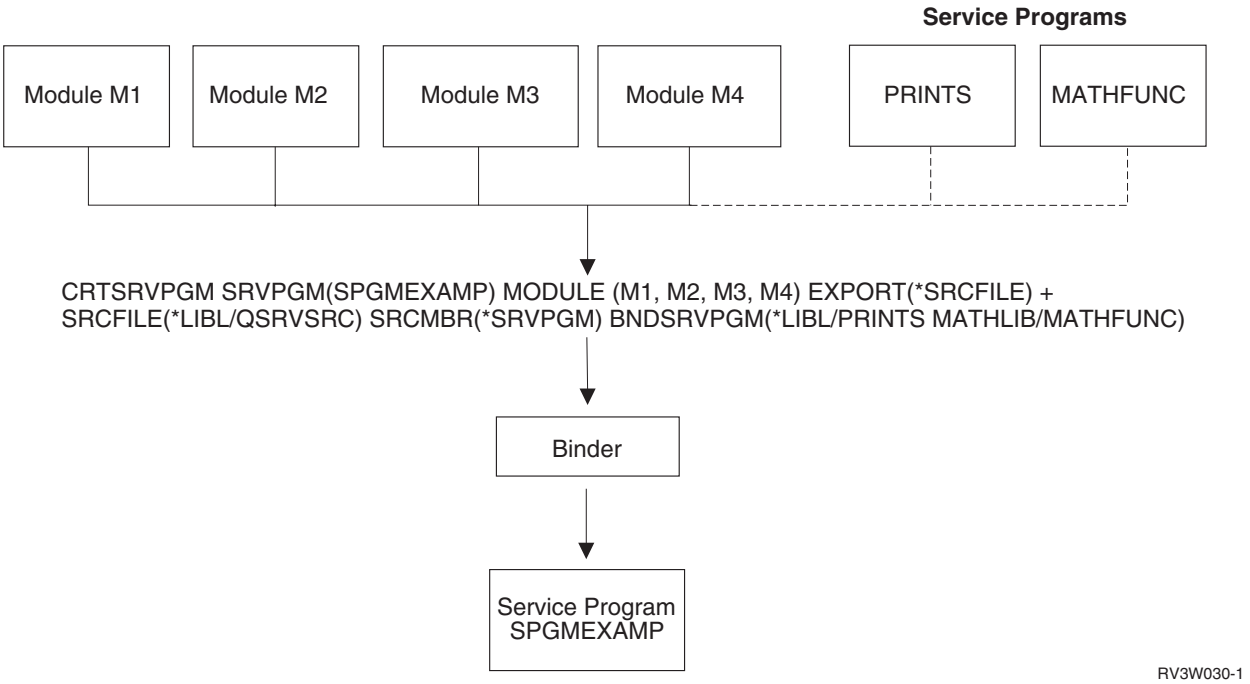
ในขณะรัน การใช้ physical link ไปที่โพรซีเดอร์และตัวข้อมูลจะมีความแตกต่างกันเล็กน้อยในเรื่องของประสิทธิภาพของการเข้าถึง 2 แบบ คือ:

- การเข้าถึงโพรซีเดอร์หรือตัวข้อมูลแบบโลคัล
- การเข้าถึงโพรซีเดอร์หรือตัวข้อมูลในโมดูลหรือเซอร์วิสโปรแกรมอื่นที่รวมเป็นโปรแกรมเดียวกัน.

รูปที่ 11และ รูปที่ 12 ในหน้า 25 แสดงให้เห็นวิธีการสร้างโปรแกรม ILE ชื่อ PGMEXAMP และเซอร์วิสโปรแกรมชื่อ SPGMEXAMP. binder ใช้โมดูล M1, M2, M3, M4 และเซอร์วิสโปรแกรม PRINTS และ MATHFUNC เพื่อสร้างโปรแกรม PGMEXAMP และเซอร์วิสโปรแกรม SPGMEXAMP.



รูปที่ 11. แสดงการสร้างโปรแกรม ILE. เส้นไข่ปลาแสดงเซอร์วิสโปรแกรมถูกรวมโดยการอ้างอิง แทนที่จะเป็นการรวมโดยการก็อปปี.



รูปที่ 12. แสดงการสร้างเซอร์วิสโปรแกรม. เส้นไข่ปลาแสดงเซอร์วิสโปรแกรมถูกรวมโดยการอ้างอิง แทนที่จะเป็นการรวมโดยการก็อปปี.

สำหรับรายละเอียดของการสร้างโปรแกรม ILE และเซอร์วิสโปรแกรม ให้ดู บทที่ 5, “แนวคิดในการสร้างโปรแกรม”, ในหน้า 75.

การเรียกไปยังโปรแกรมและโพรซีเดอร์

ใน ILE คุณสามารถเรียกได้ทั้งโปรแกรมและโพรซีเดอร์. ILE จึงมีข้อกำหนดว่า ผู้เรียกจะต้องระบุด้วยว่าเป้าหมายของคำสั่ง call นั้นเป็นโปรแกรมหรือโพรซีเดอร์. ภาษา ILE ทำตามข้อกำหนดนี้โดยการมีคำสั่ง call แยกกันระหว่างโปรแกรมและโพรซีเดอร์. ดังนั้นเมื่อคุณเขียนโปรแกรม ILE คุณจะต้องทราบว่าคุณกำลังเรียกโปรแกรมหรือโพรซีเดอร์.

ภาษา ILE แต่ละภาษามีไวยากรณ์เฉพาะตัวที่อนุญาตให้คุณแยกแยะระหว่าง Dynamic Program Call และการเรียกโพรซีเดอร์แบบสแตติก. คำสั่ง call มาตรฐานในแต่ละภาษา ILE มีค่าดีฟอลท์เป็นทั้ง Dynamic Program Call และการเรียกโพรซีเดอร์แบบสแตติก. เช่นในภาษาอาร์พีจีและโคบอลมีค่าดีฟอลต์เป็น Dynamic Program Call แต่สำหรับภาษาซีจะกำหนดเป็น การเรียกโพรซิเดอร์แบบสแตติก. เนื่องจากมาตรฐานการเรียกของภาษาต่างๆ ใน OPM และ ILE มีลักษณะเดียวกัน. ดังนั้นการย้ายโอนจากภาษา OPM ไปเป็น ILE จึงเป็นเรื่องง่าย.

Binder สามารถบรรจุชื่อของโพรซีเดอร์ที่มีความยาวถึง 256 ตัวอักขระ. สำหรับข้อกำหนดในเรื่องความยาวของชื่อของโพรซีเดอร์ ดูได้จาก ILE HLL programmer’s guide.

Dynamic Program Calls

Dynamic Program Call ถ่ายโอนการควบคุมไปยังโปรแกรมอ็อบเจ็กต์ ILE หรือโปรแกรมอ็อบเจ็กต์ OPM. Dynamic Program Call มีลักษณะดังนี้:

- โปรแกรม OPM สามารถเรียกโปรแกรม OPM อื่นหรือโปรแกรม ILE ได้ แต่ไม่สามารถเรียกเซอร์วิสโปรแกรมได้.
- โปรแกรม ILE สามารถเรียกโปรแกรม OPM หรือโปรแกรม ILE อื่นได้ แต่ไม่สามารถเรียกเซอร์วิสโปรแกรม ได้.
- เซอร์วิสโปรแกรม สามารถเรียกโปรแกรม OPM หรือโปรแกรม ILE ได้ แต่ไม่สามารถเรียกเซอร์วิสโปรแกรม อื่นได้.

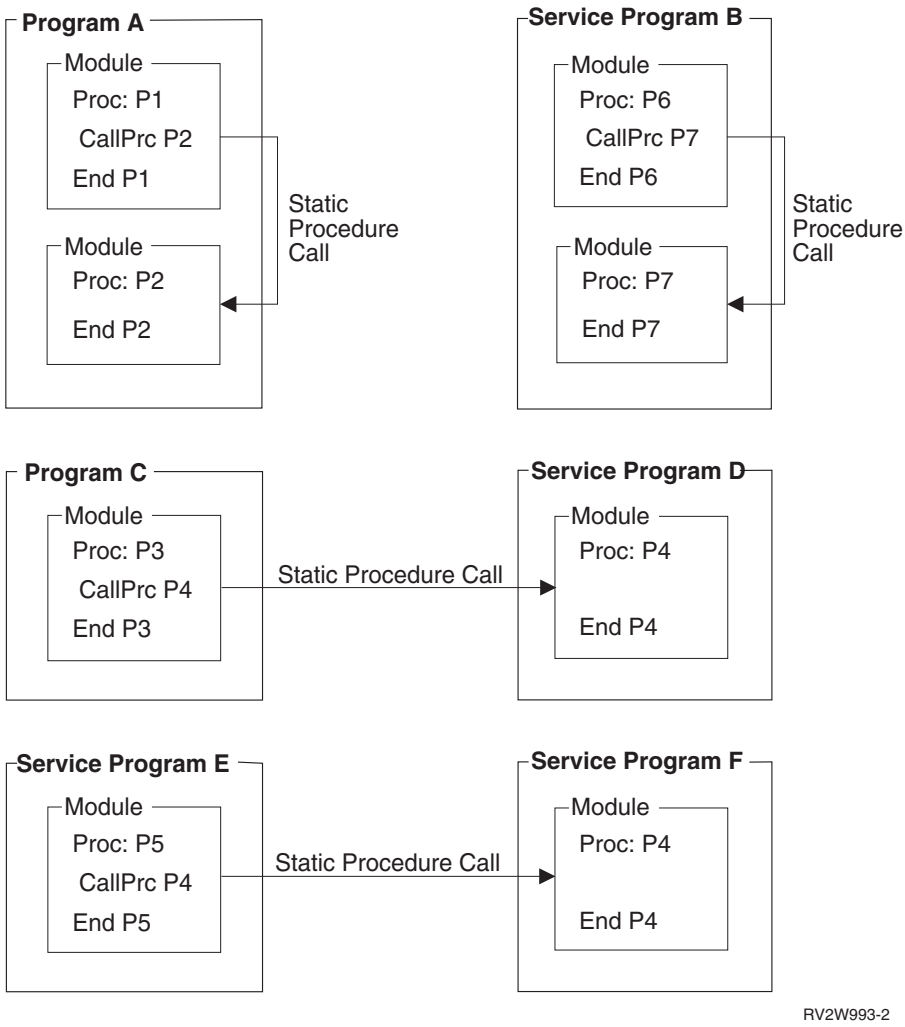
การเรียกโพรซิเดอร์แบบสแตติก

การเรียกโพรซิเดอร์แบบสแตติกถ่ายโอนการควบคุมไปยังโพรซีเดอร์ของ ILE. เรากำหนดให้มีการเรียกโพรซิเดอร์แบบสแตติกได้เฉพาะกับภาษา ILE เท่านั้น. การเรียกโพรซิเดอร์แบบสแตติกสามารถเรียกโพรซีเดอร์ต่างๆ ดังนี้:

- โพรซีเดอร์ในโมดูลเดียวกัน
- โพรซีเดอร์ในโมดูลอื่นในโปรแกรม ILE หรือเซอร์วิสโปรแกรมเดียวกัน.
- โพรซีเดอร์ในเซอร์วิสโปรแกรม ILE อื่น.

รูปที่ 13 ในหน้า 27 แสดงตัวอย่างของ การเรียกโพรซิเดอร์แบบสแตติก. จากภาพแสดงให้เห็นว่า:

- โพรซีเดอร์ในโปรแกรม ILE สามารถเรียกโพรซีเดอร์ที่ถูก export ในโปรแกรมหรือเซอร์วิสโปรแกรมเดียวกัน. เช่น โพรซีเดอร์ P1 ในโปรแกรม A เรียกโพรซีเดอร์ P2 ในโมดูลอื่น. โพรซีเดอร์ P3 ในโปรแกรม C เรียก โพรซีเดอร์ P4 ในเซอร์วิสโปรแกรม D.
- โพรซีเดอร์ในเซอร์วิสโปรแกรมสามารถเรียกโพรซีเดอร์ที่ถูก export ทั้งในเซอร์วิสโปรแกรมเดียวกันและจากเซอร์วิสโปรแกรมอื่น. เช่น โพรซีเดอร์ P6 ในเซอร์วิสโปรแกรม B เรียกโพรซีเดอร์ P7 ในโมดูลอื่น. โพรซีเดอร์ P5 ในเซอร์วิสโปรแกรม E เรียกโพรซีเดอร์ P4 ในเซอร์วิสโปรแกรม F.



รูปที่ 13. Static Procedure Calls

Activation

หลังจากสร้างโปรแกรม ILE เสร็จแล้ว คุณจำเป็นต้องรันโค้ดของคุณ. คุณ กระบวนการที่ทำให้โปรแกรมหรือ เซอร์วิสโปรแกรมพร้อมที่จะรันได้เรียกว่า **Activation**. คุณไม่ต้องมีคำสั่งในการ activate โปรแกรม. โปรแกรม ระบบจะทำ activation เมื่อโปรแกรมถูกเรียก. เนื่องจากเซอร์วิสโปรแกรมจะไม่ถูกเรียก ดังนั้นมันจะถูก activate เมื่อมีการเรียกไปที่โปรแกรมที่ต้องการการบริการจากเซอร์วิสโปรแกรมนั้น ทั้งโดยตรงและโดยอ้อม.

Activation มีหน้าที่ดังนี้:

- หาตำแหน่งเฉพาะให้กับข้อมูลแบบคงที่ ที่โปรแกรมหรือเซอร์วิสโปรแกรมต้องการ
- เปลี่ยน Symbolic Link ของเซอร์วิสโปรแกรม ที่ export ไปเป็น Physical Address

ไม่ว่าจะมีกี่งานที่กำลังรันโปรแกรมหรือเซอร์วิสโปรแกรมอยู่ก็ตาม, จะมีเพียง คำสั่งของอ็อบเจ็กต์'s ชุดเดียวเท่านั้นที่เก็บไว้ในที่เก็บข้อมูล. อย่างไรก็ตามในแต่ละ activation จะมีที่เก็บข้อ

มูลแบบคงที่ของมันเอง. ดังนั้นแม้ว่าโปรแกรมอ็อบเจ็กต์หนึ่งจะถูกใช้งานพร้อมๆ กันโดยหลายๆ งานแต่ค่าตัวแปรแบบคงที่จะถูกแยกไปสำหรับแต่ละ activation. และโปรแกรมก็สามารถถูก activate ได้มากกว่า 1 activation group แม้ว่าจะอยู่ภายในงานเดียวกัน แต่จะมี activation ของแต่ละ activation group แยกจากกัน.

ถ้าสิ่งที่จะกล่าวต่อไปนี้ข้อใดข้อหนึ่งเป็นจริง:

- Activation จะไม่สามารถค้นหาเซอร์วิสโปรแกรมที่จำเป็นพบได้
- เซอร์วิสโปรแกรมไม่สนับสนุนโพรซีเดอร์หรือข้อมูลที่ signature กำหนดไว้.

จะเกิดข้อผิดพลาดขึ้นและคุณไม่สามารถรันแอ็พพลิเคชันได้.

สำหรับรายละเอียดเพิ่มเติมของ program activation ดูได้ในหัวข้อ “การสร้าง Program Activation” ในหน้า 32.

เมื่อ activation กำหนดพื้นที่สำหรับเก็บตัวแปรแบบคงที่ของโปรแกรม พื้นที่ว่างจะถูกจับจองโดย activation group. ในเวลาที่โปรแกรมหรือเซอร์วิสโปรแกรมถูกสร้างขึ้น คุณสามารถกำหนด activation group ที่จะถูกใช้ในขณะรัน.

สำหรับรายละเอียดของ activation group อยู่ในหัวข้อ “Activation Group” ในหน้า 33.

การจัดการข้อผิดพลาด (Error Handling)

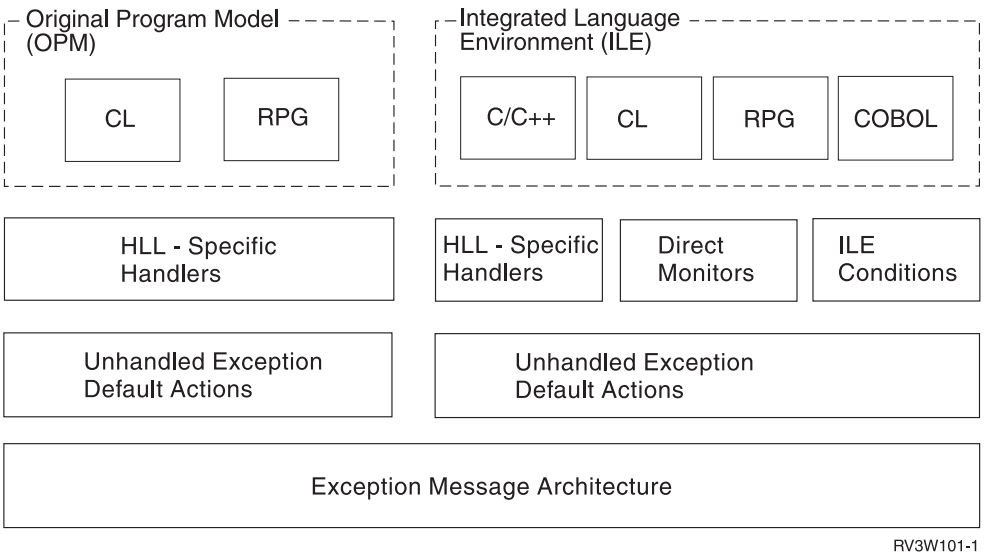
รูปที่ 14 ในหน้า 29 แสดงโครงสร้างทั้งหมดของการจัดการข้อผิดพลาดทั้งในโปรแกรม OPM และ ILE. ซึ่งเราจะใช้ภาพนี้ในการอธิบายกึงความสามารถในการจัดการข้อผิดพลาดตลอดทั้งหนังสือเล่มนี้. หัวข้อนี้จะเป็นการอธิบายอย่างคร่าวๆ ในเรื่องความสามารถของการจัดการข้อผิดพลาดของภาษามาตรฐาน. ฐาน สำหรับรายละเอียดของการจัดการข้อผิดพลาด ดูได้ในหัวข้อ “การจัดการข้อผิดพลาด (Error Handling)” ในหน้า 45.

ในรูปแสดงถึงเลเยอร์พื้นฐานชนิดหนึ่งเรียกว่า exception–message architecture. ระบบจะสร้างข้อความ Exception ขึ้น เมื่อโปรแกรม OPM หรือโปรแกรม ILE มีข้อผิดพลาดเกิดขึ้น. ข้อความ Exception ยังใช้สำหรับแสดงสถานะที่ไม่เกี่ยวข้องกับการผิดพลาดของโปรแกรมด้วย. เช่น เมื่อหาเร็กคอร์ดของฐานข้อมูล ไม่พบก็จะมีการส่งข้อความ Exception ออกมาเป็นต้น.

ภาษาในระดับสูงแต่ละภาษาจะมีการกำหนดความสามารถของการจัดการข้อผิดพลาดเฉพาะภาษานั้นๆ. แม้ว่าความสามารถนี้จะแตกต่างกันตามภาษา แต่โดยทั่วไปแล้ว ผู้ใช้ HLL แต่ละคนสามารถจะกำหนดสถานการณ์ผิดพลาดที่ตนต้องการใช้. ซึ่งจะรวมไปถึงการระบุรูทีนจัดการข้อผิดพลาดไว้ด้วย. ด้วย เมื่อเกิด exception ขึ้น ระบบจะหาตำแหน่งของรูที่จัดการข้อผิดพลาด และผ่านการควบคุมไปยังคำสั่งที่ผู้ใช้เขียนไว้. คุณสามารถปฏิบัติได้หลายวิธีรวมไปถึงการจบการทำงานของโปรแกรมหรือ แก้ไขจากข้อผิดพลาด และดำเนินโปรแกรมต่อไป.

รูปที่ 14 ในหน้า 29 แสดงให้เห็นว่า ILE และโปรแกรม OPM มีโครงสร้างของข้อความ Exception เหมือนกัน. ข้อความ Exception ถูกสร้างขึ้นโดยระบบเมื่อพบข้อผิดพลาดเฉพาะภาษาในโปรแกรม. เลเยอร์ที่อยู่ต่ำสุดในรูปจะทำการรับ–ส่งข้อความ Exception. โดยการใช้คำสั่งหรือ API ของตัวจัด

การแมสเสจ. นอกจากนี้ข้อความ Exception ยังสามารถถูกรับ–ส่งระหว่างโปรแกรม OPM และโปรแกรม ILE ได้.



รูปที่ 14. Error Handling for OPM and ILE

การจัดการข้อผิดพลาดเฉพาะภาษาทำงานเหมือนกันทั้งในโปรแกรม ILE และ OPM แต่ก็มีข้อแตกต่างกัน ดังนี้:

- เมื่อระบบส่งข้อความ Exception ไปที่โปรแกรม ILE ชื่อของโพรซีเดอร์และโมดูลจะถูกใช้ในการตรวจสอบคุณสมบัติของข้อความ Exception. ถ้าคุณส่งข้อความ Exception หนึ่ง คุณสมบัติที่เหมือนกันนี้จะถูกกำหนดไว้. เมื่อข้อความ Exception ปรากฏขึ้นในบันทึกการใช้งาน (job log) ของโปรแกรม ILE ระบบก็จะให้ชื่อของโปรแกรม ชื่อของโมดูลและชื่อของโพรซีเดอร์.
- การ Optimization เพิ่มเติมสำหรับโปรแกรม ILE อาจส่งผลใน HLL statement number หลายๆ ตัวที่มีคำสั่งในการสร้างเหมือนกัน. จากการ optimization ทำให้ข้อความ Exception ในบันทึกการใช้งานอาจมีหลาย HLL statement number.

รายละเอียดเพิ่มเติมเกี่ยวกับความสามารถในการจัดการข้อผิดพลาด อธิบายไว้ในหัวข้อ “การจัดการข้อผิดพลาด (Error Handling)” ในหน้า 45.

Optimizing Translator

บนระบบ OS/400 **optimization** หมายถึงการทำให้อ็อบเจ็กต์มีประสิทธิภาพในการทำงานสูงสุด. ภาษา ILE ทุกภาษาใช้เทคนิคในการ optimization ที่มาจาก ILE optimization translator. โดยทั่วไปยิ่งมีการ optimize มากก็ยิ่งทำให้ใช้เวลาในการสร้างอ็อบเจ็กต์นานขึ้น. แต่ในเวลาทำงาน โปรแกรมหรือเซอร์วิสโปรแกรมที่มีระดับการ optimize มากจะรันได้เร็วกว่าโปรแกรมหรือเซอร์วิสโปรแกรมที่มีการ optimize น้อยกว่า.

แม้ว่าการ optimize จะสามารถกำหนดให้กับโมดูล โปรแกรมอ็อบเจ็กต์และเซอร์วิสโปรแกรม เทคนิคการ optimize จะถูกกำหนดให้กับโมดูลแต่ละโมดูล. ระดับของ optimization มีดังนี้:

10 หรือ *NONE

20 หรือ *BASIC

30 หรือ *FULL

40 (มี optimization เกินระดับ 30 ขึ้นไป)

ในด้านประสิทธิภาพ คุณมักจะต้องการ optimization ในระดับสูงเมื่อคุณใช้โมดูลในการสร้างโปรแกรม. คุณควรทดสอบโค้ดของคุณในระดับของ optimization ที่ต้องการจะใช้จริง. ตรวจสอบว่าทุกอย่างทำงานตามที่คุณกำหนดไว้แล้วจึงค่อยกำหนดให้ผู้ใช้ของคุณใช้โค้ดนั้นได้.

เนื่องจากการ optimize ในระดับ 30 (*FULL) หรือ 40 สามารถส่งผลกระทบต่อการสั่งงานของโปรแกรมของคุณ คุณอาจต้องการทราบถึงข้อจำกัดในการดีบักและการตรวจพบ exception ต่างๆ. ให้ดูใน บทที่ 10, “ข้อพิจารณาในการดีบักโปรแกรม”, ในหน้า 143 สำหรับข้อพิจารณาในการดีบัก. และดูใน ภาคผนวก B, “Exception ในโปรแกรมที่ถูก Optimize”, ในหน้า 221 สำหรับเรื่องของข้อพิจารณาของการกำหนดข้อผิดพลาด.

โปรแกรมดีบักเกอร์ (Debugger)

ILE มีโปรแกรมดีบักเกอร์สำหรับการดีบักในระดับซอร์ส. โปรแกรมดีบักเกอร์สามารถทำงานกับไฟล์ที่กำหนดไว้และยอมให้ตั้งค่าจุดพัก, แสดงค่าตัวแปรและขั้นตอนของคำสั่ง (instruction). ได้โดยไม่ต้องป้อนคำสั่ง (command) ลงในบรรทัดคำสั่ง. ทำให้คุณสามารถใช้บรรทัดคำสั่งได้ในขณะที่ทำงานกับโปรแกรมดีบักเกอร์.

โปรแกรมดีบักเกอร์ระดับซอร์สใช้ APIs ที่ระบบมีให้ เพื่อให้คุณสามารถดีบักโปรแกรมหรือเซอร์วิสโปรแกรมของคุณ. API นี้ทุกคนสามารถใช้งานได้ และยอมให้คุณเขียนโปรแกรมดีบักเกอร์ของคุณเองอีกด้วย.

อย่างไรก็ตาม โปรแกรมดีบักเกอร์สำหรับโปรแกรม OPM ก็ยังคงมีอยู่ในระบบ iSeries แต่สามารถใช้ดีบักเฉพาะโปรแกรม OPM เท่านั้น.

เมื่อคุณดีบักและ optimize โมดูล อาจมีความสับสนเกิดขึ้น. เมื่อคุณใช้โปรแกรมดีบักเกอร์ของ ILE ในการดูหรือเปลี่ยนค่าตัวแปรของโปรแกรมหรือโพรซีเดอร์ที่กำลังรัน สิ่งที่เกิดขึ้นคือ. โปรแกรมดีบักเกอร์ทำการกู้หรืออัพเดตข้อมูลในที่เก็บข้อมูลของตัวแปรนี้. การ optimize ที่ระดับ 20 (*BASIC) 30(*FULL) หรือ 40 ค่าปัจจุบันของข้อมูลในตัวแปรอาจอยู่ในฮาร์ดแวร์เรจิสเตอร์ ซึ่งโปรแกรมดีบักเกอร์ไม่สามารถเข้าถึงได้. (ข้อมูลของตัวแปรที่อยู่ในฮาร์ดแวร์เรจิสเตอร์ จะขึ้นอยู่กับปัจจัยหลายอย่าง. ปัจจัยเหล่านั้นประกอบด้วย วิธีการใช้ตัวแปรนั้น, ขนาด, และตำแหน่งที่คุณหยุดที่จะทดสอบหรือเปลี่ยนข้อมูลในตัวแปร). ดังนั้นค่าที่แสดงออกมาสำหรับตัวแปรอาจไม่ใช่ค่าในปัจจุบัน. ด้วยเหตุนี้คุณควรใช้การ optimize ที่ระดับ 10 (*NONE) ในระหว่างการพัฒนา. แล้วเปลี่ยนค่าเป็นระดับที่ 30 (*FULL) หรือ 40 ในระหว่างการผลิต.

สำหรับข้อมูลเพิ่มเติมของโปรแกรมดีบักเกอร์สำหรับ ILE ดูใน บทที่ 10, “ข้อพิจารณาในการดีบักโปรแกรม”, ในหน้า 143.

บทที่ 3. แนวคิด ILE ขั้นสูง

บทนี้จะเป็นการอธิบายแนวคิดระดับสูงของแบบจำลอง ILE. ก่อนที่จะอ่านบทนี้ ควรคุ้นเคยกับแนวคิดที่อธิบายในบทที่ 2, “แนวคิด ILE ขั้นพื้นฐาน”, ในหน้า 13.

Program Activation

Activation คือกระบวนการที่ใช้สำหรับเตรียมโปรแกรมให้ทำงานได้. ทั้งโปรแกรม ILE และเซอร์วิสโปรแกรมจะต้องถูก activate โดยระบบก่อนถึงจะสามารถรันได้.

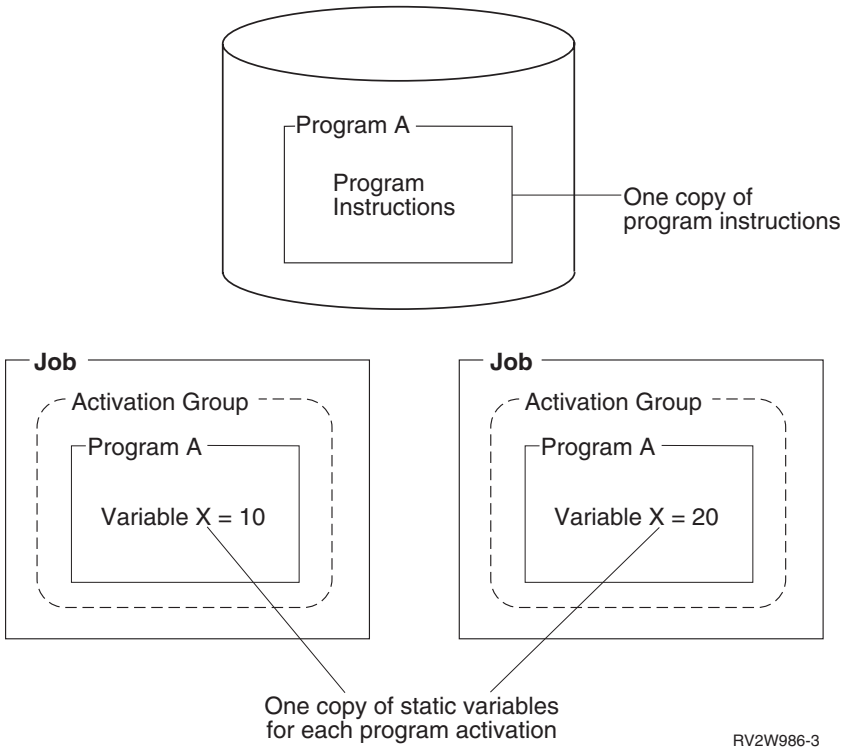
Program activation ประกอบด้วยขั้นตอนหลัก 2 ขั้นตอน คือ:

1. จัดสรรและให้ค่าเริ่มต้นแก่พื้นที่เก็บข้อมูลแบบสแตติกสำหรับโปรแกรม.
2. ทำการรวมโปรแกรมให้เข้ากับเซอร์วิสโปรแกรมจนสำเร็จ.

หัวข้อนี้จะมุ่งเน้นในขั้นตอนที่ 1 ส่วนขั้นตอนที่ 2 จะอธิบายอยู่ในหัวข้อ “Service Program Activation” ในหน้า 40.

รูปที่ 15 ในหน้า 32 แสดงภาพโปรแกรม ILE สองโปรแกรมที่เก็บอยู่ในเนื้อที่ดิสก์แบบถาวร. เช่นเดียวกับอ็อบเจ็กต์ของ OS/400 ทุกๆ อ็อบเจ็กต์ เจ็กต์ คือโปรแกรมอ็อบเจ็กต์นี้อาจถูกแบ่งใช้ให้กับผู้ใช้หลายคนพร้อมๆ กันในรูปแบบงานของ OS/400 ที่แตกต่างกัน. โดยจะมีโค้ดของโปรแกรมอยู่เพียงชุดเดียว. เมื่อโปรแกรม ILE ถูกเรียก ตัวแปรบางตัวที่กำหนดไว้ในโปรแกรมจะถูกจัดสรรพร้อมกับค่าเริ่มต้นสำหรับ program activation ในแต่ละครั้ง.

ดังแสดงใน รูปที่ 15 ในแต่ละ program activation จะสนับสนุนตัวแปรเหล่านี้อย่างน้อย 1 ชุด. ในหนึ่ง program activation สามารถมีตัวแปรที่มีชื่อเหมือนกันหลายชุดได้. ถ้า HLL ของคุณยอมให้กำหนดตัวแปรแบบคงที่ (static variable) ที่จำกัดขอบเขตอยู่ในโพรซีเดอร์แต่ละโพรซีเดอร์เท่านั้น.



รูปที่ 15. แสดงตัวแปรชุดเดียวกันสำหรับในแต่ละ Program Activation

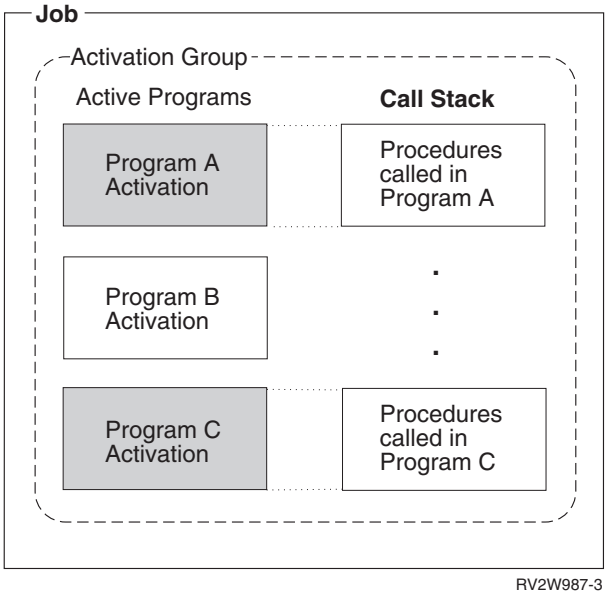
การสร้าง Program Activation

ILE จัดการโพรเซสของ program activation โดยการติดตาม program activation ภายใน activation group. (คำจำกัดความของ activation group ดูได้จากหัวข้อ "Activation Group" ในหน้า "Activation Group" ในหน้า 33). สำหรับโปรแกรมอ็อบเจ็กต์ที่อยู่ใน activation group หนึ่ง ๆ จะมีเพียงหนึ่ง activation เท่านั้น. ดังนั้นโปรแกรมที่มีชื่อเหมือนกันอยู่ในไลบรารี OS/400 ต่างกัน จะถูกพิจารณาว่าเป็นโปรแกรมอ็อบเจ็กต์ที่ต่างกัน.

เมื่อคุณใช้คำสั่ง dynamic program call ในโปรแกรม HLL ของคุณ ILE จะใช้ activation group ที่ถูกกำหนดไว้เมื่อโปรแกรมถูกสร้างขึ้น. แอ็ตทริบิวต์นี้ถูกกำหนดโดยใช้พารามิเตอร์ activation group (ACTGRP) ในคำสั่ง Create Program (CRTPGM) หรือคำสั่ง Create Service Program (CRTSRVPGM). ถ้ามี program activation อยู่ใน activation group ที่ใช้พารามิเตอร์นี้อยู่แล้ว, activation group จะถูกใช้. ถ้าโปรแกรมไม่เคยถูก activate ใน activation group, มันจะถูก activate ก่อนที่จะรัน. ชื่อของ activation group สามารถเปลี่ยนได้โดยใช้พารามิเตอร์ ACTGRP ในคำสั่ง UPDPGM และ UPDSRVPGM

เมื่อโปรแกรมถูก activate มันจะยังคง activate ไปจนกระทั่ง activation group ถูกลบไป. ผลจากกฎข้อนี้เป็นไปได้ว่าจะมีโปรแกรมที่แอคทีฟไม่ได้อยู่ใน call stack ภายใน activation group. รูปที่ 16 ในหน้า 33 แสดงตัวอย่างของโปรแกรมที่แอ็คทีฟ 3 โปรแกรมอยู่ใน activation group หนึ่ง ,แต่มีเพียง 2 ใน 3 โปรแกรมเท่านั้นที่มีโพรซีเดอร์อยู่ใน call stack. ในตัวอย่างนี้, โปรแกรม A เรียกใช้โปรแกรม B ทำให้โปรแกรม B ถูกเรียกทำงาน. โปรแกรมBก็จะส่งผลการทำงานกลับไปยังA. หลัง

จากนั้นโปรแกรมAจะเรียกใช้งานโปรแกรมC. สุดท้าย call stack จะมีแต่โพรซีเดอร์สำหรับโปรแกรม A และ C แต่ไม่มีสำหรับB .สำหรับหัวข้อเรื่องcall stack,กรุณาดูที่ “Call Stack” ในหน้า 117.



รูปที่ 16.

Activation Group

โปรแกรมและเซอร์วิสโปรแกรมของ ILE ทุกโปรแกรม ถูก activate ภายในโครงสร้างย่อยของงานที่เรียกว่า **activation group**. โครงสร้างย่อยนี้ประกอบด้วยรีซอร์สที่จำเป็นต่อการรันโปรแกรม. รีซอร์สเหล่านี้แบ่งออกเป็นหมวดหมู่ได้ดังนี้:

ตัวแปรแบบสแตติกของโปรแกรม

หน่วยเก็บข้อมูลแบบไดนามิก

รีซอร์สที่ใช้จัดการข้อมูลชั่วคราว

ตัวจัดการ exception และโพรซีเดอร์ที่ใช้จบการทำงาน

Activation groups สามารใช้ได้ทั้งหน่วยเก็บข้อมูลแบบ single-level หรือแบบ teraspace เพื่อใช้เป็นหน่วยเก็บข้อมูลสำหรับตัวแปรแบบสแตติกของโปรแกรม. สำหรับข้อมูลเพิ่มเติม ให้ดูได้จากบทที่ 4, “หน่วยเก็บข้อมูลแบบ Teraspace และ Single-level”, ในหน้า 57. เมื่อใช้หน่วยเก็บข้อมูลแบบ single-level, ตัวแปรแบบสแตติกของโปรแกรม และหน่วยความจำไดนามิกจะถูกกำหนดแอดเดรสของพื้นที่หน่วยเก็บข้อมูลแยกจากกันตาม activation group, ซึ่งช่วยป้องกันการเข้าถึงหน่วยเก็บข้อมูลโดยไม่ได้ตั้งใจ. แต่ถ้าใช้หน่วยเก็บข้อมูลแบบteraspace, ตัวแปรแบบสแตติกของโปรแกรม และหน่วยความจำไดนามิกก็จะถูกกำหนดแอดเดรสแยกจากกันภายใน teraspace,ซึ่งช่วยป้องกันการเข้าถึงหน่วยความจำโดยไม่ได้ตั้งใจได้ดีกว่า.

รีซอร์สที่ใช้จัดการข้อมูลชั่วคราว ประกอบด้วย:

การเปิดไฟล์ (open data path หรือ ODP)

Commitment definitions

SQL cursor แบบโลคัล

SQL cursor แบบรีโมต

Hierarchical file system (HFS)

ส่วนจัดการการติดต่อกับผู้ใช้

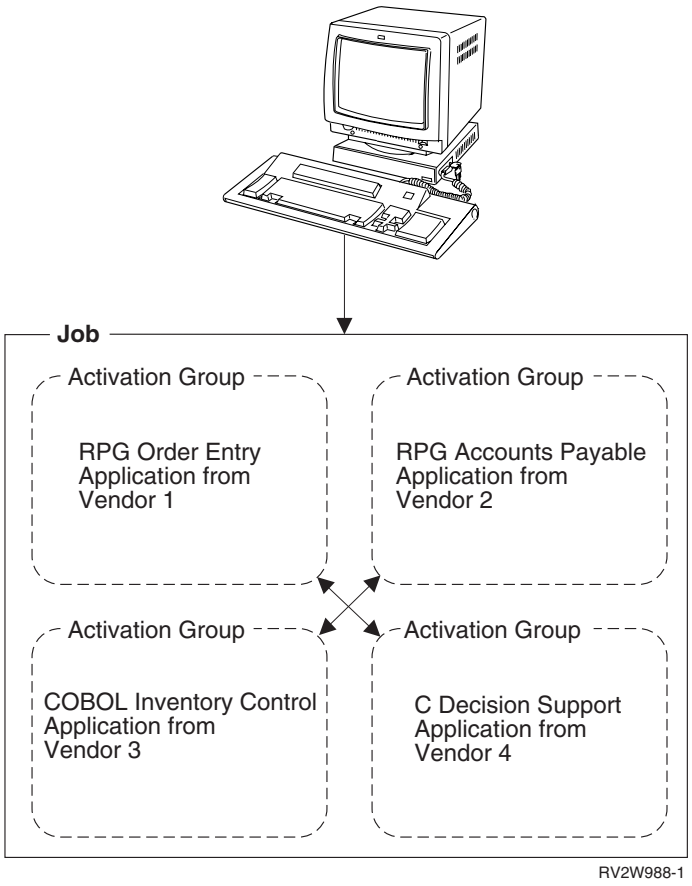
ส่วนจัดการการ Query

การเปิด communications links

การติดต่อ แบบ Common Programming Interface (CPI)

การแยกกันระหว่างรีซอร์สเหล่านี้กับ activation group สนับสนุนแนวคิดพื้นฐาน. ที่ว่า ทุกโปรแกรมที่ activate ใน activation group จะถูกพัฒนาเหมือนเป็นแอ็พพลิเคชันเดียวกัน .

ผู้ผลิตซอฟท์แวร์อาจเลือก activation group ที่ต่างกันเพื่อจะแยกโปรแกรมของเขาจากแอ็พพลิเคชันจากผู้ผลิตรายอื่นที่รันในงานเดียวกัน. ลักษณะดังกล่าวแสดงในรูปที่ รูปที่ 17 ในหน้า 35. ซึ่งเป็นการแสดงโซลูชันสำหรับลูกค้ารายหนึ่งที่เกิดจากการรวมกันของซอฟท์แวร์หลายตัวที่มาจากผู้ผลิตหลายราย. activation group เพิ่มความง่ายในการใช้งานร่วมกันโดยการแยกรีซอร์สของแอ็พพลิเคชันที่มาจากผู้ผลิตแต่ละรายออกเป็นกลุ่มๆ.



รูปที่ 17. แสดง Activation Group เป็นส่วนแบ่งแยกแอ็พพลิเคชันของผู้ผลิตแต่ละรายออกจากกัน

ผลลัพธ์ที่ได้อย่างชัดเจนในการกำหนดรีซอร์สที่กล่าวมาข้างต้นให้แก่ activation group หนึ่ง ๆ. ก็คือ เมื่อ activation group ถูกลบออก รีซอร์สทั้งหมดที่กล่าวมาจะกลับคืนสู่ระบบ. ระบบ รีซอร์สที่ใช้จัดการข้อมูลชั่วคราวที่ยังเปิดอยู่ในขณะที่ลบ activation group ก็จะถูกปิดโดยระบบ. ส่วนที่เก็บข้อมูลสำหรับตัวแปรแบบคงที่และแบบอัตโนมัติกับที่เก็บข้อมูลแบบไดนามิกที่ยังคงค้างอยู่ จะถูกส่งกลับคืนสู่ระบบด้วยเช่นกัน.

การสร้าง Activation Group

คุณสามารถควบคุมการสร้าง activation group ของ ILE ขณะรันได้. โดยการกำหนดพารามิเตอร์ ACTGRP ลงในคำสั่ง CRTPGM หรือคำสั่ง CRTSRVPGM. เนื่องจากไม่มีคำสั่งในการสร้าง activation group โดยตรง.

โปรแกรม ILE จะมี activation group ที่มีแอ็ตทริบิวต์แบบใดแบบหนึ่งดังต่อไปนี้:

- Activation group ที่ถูกกำหนดชื่อโดยผู้ใช้ (User-named activation group)

  ด้วยการกำหนดพารามิเตอร์ ACTGRP(name). แอ็ตทริบิวต์นี้จะยอมให้จัดการกลุ่มของโปรแกรม ILE และเซอร์วิสโปรแกรม ILE เสมือนกับเป็นแอ็พพลิเคชันเดียวกัน. โดย activation group จะถูกสร้างเมื่อมีความต้องการใช้ในครั้งแรก. และถูกใช้โดยทุกโปรแกรมและเซอร์วิสโปรแกรม ที่ระบุชื่อ activation group เดียวกัน.
- Activation group ที่ถูกกำหนดชื่อโดยระบบ (System-named activation group)

ด้วยการกำหนดพารามิเตอร์ ACTGRP (*NEW) ในคำสั่ง CRTPGM. CRTPGM แอ็ตทริบิวต์นี้จะยอมให้สร้าง activation group ใหม่เมื่อโปรแกรมถูกเรียก. ILE จะกำหนดชื่อให้กับ activation group. โดยชื่อนี้จะเป็นชื่อเฉพาะของ activation group นั้นในงานของคุณ. ชื่อที่ถูกกำหนดขึ้นโดยระบบนั้น จะไม่ตรงกับชื่อใดๆ ที่คุณใช้กับ activation group แบบที่ผู้ใช้กำหนดชื่อให้. อย่างไรก็ตาม เซอร์วิสโปรแกรมของ ILE ไม่สนับสนุนแอ็ตทริบิวต์นี้.

- Activation group ของโปรแกรมที่เป็นผู้เรียก

  ด้วยการกำหนดพารามิเตอร์ ACTGRP(*CALLER). แอ็ตทริบิวต์นี้ยอมให้สร้างโปรแกรม ILE หรือ เซอร์วิสโปรแกรม ILE ที่ถูก activate ใน activation group ของโปรแกรมตัวเรียก. ด้วยแอ็ตทริบิวต์นี้ activation group จะไม่ถูกสร้างขึ้นใหม่เมื่อโปรแกรมหรือเซอร์วิสโปรแกรมถูกเรียกทำงาน.

- ใช้ค่าแอ็ตทริบิวต์เป็นตัวกำหนด activation group ที่เหมาะสมกับภาษาของโปรแกรมและรูปแบบของหน่วยจัดเก็บข้อมูล.

  การระบุค่าพารามิเตอร์ ACTGRP(*ENTMOD) ในคำสั่งCRTPGM. โมดูลประเภท program entry procedureที่กำหนดโดยพารามิเตอร์ ENTMOD จะถูกตรวจสอบเมื่อมีการระบุค่า ACTGRP(*ENTMOD). โดยเหตุการณ์จะเป็นดังนี้:

  - ถ้าแอตทริบิวต์ของโมดูลเป็น RPGLE or CBLLE แล้ว QILE จะถูกใช้เป็น activation group.
  - ถ้าแอตทริบิวต์ของโมดูลเป็น CLLE, และ
    - มีการระบุค่า STGMDL(*SNGLVL), QILE จะถูกใช้เป็น activation group.
    - มีการระบุค่า STGMDL(*TERASPACE) , QILETS จะถูกใช้เป็น activation group.
  - แต่ถ้าแอ็ตทริบิวต์ของโมดูลไม่ใช่ RPGLE, CBLLE, หรือ CLLE, แล้ว *NEW จะถูกใช้เป็น activation group.

  ค่า ACTGRP(*ENTMOD) เป็นค่าดีฟอล์ตสำหรับพารามิเตอร์นี้ของคำสั่ง CRTPGM.

ทุก activation group ภายในงานหนึ่ง ๆ จะมีชื่อกำหนดอยู่. ถ้ามีactivation group อยู่ภายในjob, activation group ภายใน job จะถูกใช้โดย ILE ในการ activate โปรแกรมและเซอร์วิสโปรแกรมที่ถูกระบุชื่อนั้น. ชื่อของ activation group จะไม่สามารถซ้ำกันในงานเดียวกัน. อย่างไรก็ตาม, คุณสามารถใช้พารามิเตอร์ ACTGRP ในคำสั่ง UPDPGM และคำสั่ง UPDSRVPGM ในการเปลี่ยนชื่อของ activation group ได้.

Default Activation Groups

เมื่องานของ OS/400 เริ่มขึ้น ระบบจะสร้าง activation group ขึ้น 2 กลุ่ม ซึ่งจะถูกใช้โดยโปรแกรม OPM. Default Activation Group จะใช้หน่วยเก็บข้อมูลแบบ single-level สำหรับตัวแปรสแตติกของโปรแกรม. คุณไม่สามารถลบ default activation group ของ OPM ได้. มันจะถูกลบโดยระบบเมื่องานของคุณสิ้นสุดลง.

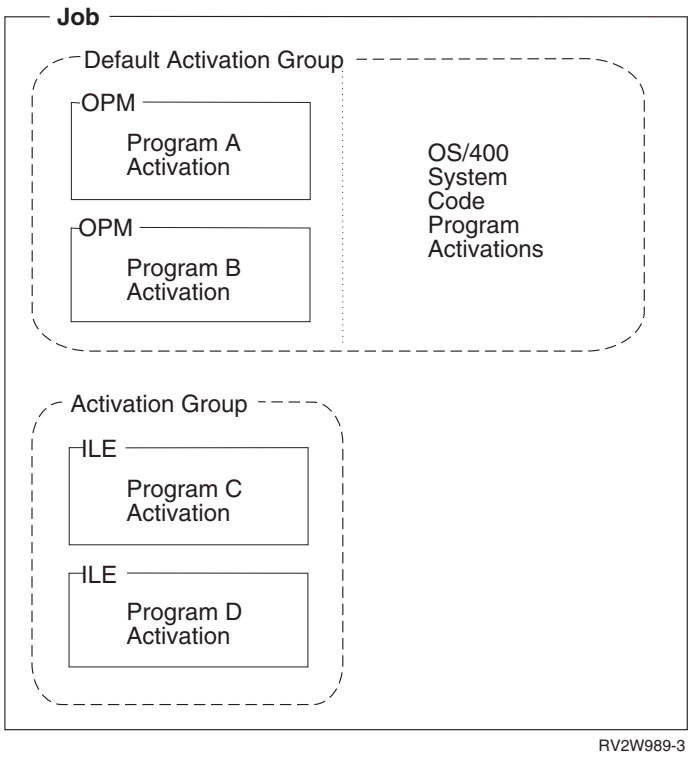
โปรแกรม ILE และเซอร์วิสโปรแกรม ILE สามารถถูก activate ใน default activation group ของ OPM ได้ หากเป็นไปตามเงื่อนไข 2 ข้อ ดังนี้:

- โปรแกรมหรือเซอร์วิสโปรแกรมของ ILE ถูกสร้างด้วย activation group แบบ *CALLER.
- การเรียกโปรแกรมหรือเซอร์วิสโปรแกรมของ ILE มีจุดเริ่มต้นอยู่ใน default activation group ของ OPM.

- โปรแกรม ILE หรือเซอร์วิสโปรแกรมไม่สามารถใช้โมเดลหน่วยเก็บข้อมูลแบบ Teraspaceได้.

เนื่องจาก default activation group ไม่สามารถถูกลบได้ ดังนั้น HLL end verbs ของ ILE ของคุณจะไม่สามารถจบกระบวนการทำงานได้. ไฟล์ที่เปิดอยู่จะไม่ถูกปิดจนกว่างานจะสิ้นสุด. ที่เก็บข้อมูลแบบคงที่และ heap ที่ถูกใช้โดยโปรแกรม ILE ก็จะไม่สามารถถูกส่งกลับไปยังระบบจนกว่า job จะจบลงเช่นกัน.

รูปที่ 18แสดงตัวอย่างงานของ OS/400 ที่มี activation group ของ ILE และ default activation group ของ OPM. โดย default activation group ของ OPM ทั้ง 2 กลุ่มถูกรวมเข้าด้วยกันเนื่องจากการใช้ค่าเฉพาะ *DFTACTGRP ในทั้ง 2 กลุ่ม. ส่วนกรอบสี่เหลี่ยมในแต่ละ activation group หมายถึง program activation.



รูปที่ 18. Default Activation Groups and ILE Activation Group

การลบ ILE Activation Group

Activation group ต้องการรีซอร์สที่ถูกสร้างขึ้นในงาน. เวลาในการทำงานอาจลดลงถ้าแอ็พพลิเคชันมีการ นำ activation group กลับมาใช้อีกครั้ง. ILE มีทางเลือกหลายทางที่ให้คุณออกจาก activation group โดยไม่ต้องจบหรือลบ activation group. ดังนั้นการที่ activation group จะถูกลบนั้นจะขึ้นอยู่กับชนิดของ activation group และวิธีการสิ้นสุดการทำงานของแอ็พพลิเคชัน.

แอ็พพลิเคชันอาจออกจาก activation group และกลับไปยัง call stack entry (ดูหัวข้อ“Call Stack” ในหน้า 117) ที่ทำงานอยู่ใน activation group อื่นด้วยวิธีการต่างๆ ดังนี้:

- HLL end verbs

ตัวอย่างเช่น, คำสั่ง STOP RUN ในภาษาโคบอล หรือ exit( ) ในภาษาซี.

- Unhandled exceptions

  Unhandled exceptions สามารถถูกย้ายโดยระบบไปยัง call stack entry ใน activation group อื่น.
- คำสั่ง return ของภาษาระดับสูง

  ตัวอย่างเช่นคำสั่ง return ในภาษาซี คำสั่ง EXIT PROGRAM ในภาษาโคบอล และ คำสั่ง RETURN ในภาษาอาร์พีจี.
- การกระโดดข้าม (Skip operation)

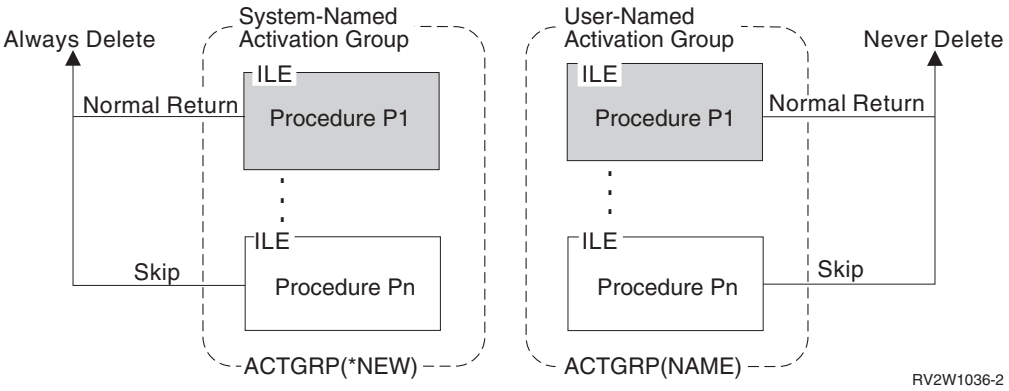
  เช่น การส่งข้อความ exception หรือข้ามไปยัง call stack entry ที่ไม่อยู่ใน activation group.

คุณสามารถลบ activation group จากแอ็พพลิเคชันของคุณโดยการใช้ HLL end verbs. และ unhandled exception สามารถเป็นเหตุให้ activation group ของคุณถูกลบได้เหมือนกัน. ขอบเขตการควบคุม (Control Boundary) ที่ใกล้ที่สุดที่ปรากฏคือ call stack entry ที่เก่าที่สุดใน activation group (บางครั้งเรียก hard control boundary). ในกรณีที่ control boundary ที่ใกล้ที่สุดไม่ใช่ call stack entry ที่เก่าที่สุด (บางครั้งเรียก soft control boundary) การควบคุมจะส่งไปยัง call stack ที่อยู่ก่อน control boundary. อย่างไรก็ตาม activation group จะไม่ถูกลบ.

Control boundary คือ call stack entry ที่แสดงขอบเขตของแอ็พพลิเคชันของคุณ. ILE จะกำหนด control boundary เมื่อมีการเรียกระหว่าง activation group. สำหรับคำจำกัดความของ control boundary ดูรายละเอียดได้จากหัวข้อ “ขอบเขตการควบคุม” ในหน้า 42.

Activation group ที่ถูกกำหนดชื่อโดยผู้ใช้ อาจจะยังอยู่ใน job สำหรับการใช้ครั้งต่อไป. สำหรับ activation group ประเภทนี้, การรีเทิร์นแบบปกติหรือการข้าม (skip) ผ่าน hard control boundary จะไม่เป็นการลบ activation group. แต่การปฏิบัติในลักษณะเดียวกันนี้ใน activation group ที่ถูกกำหนดชื่อโดยระบบจะเป็นการลบ activation group. activation group ที่ถูกกำหนดชื่อโดยระบบมักจะถูกลบ เนื่องจากคุณไม่สามารถใช้มันซ้ำได้อีกโดยบอกชื่อที่ระบบตั้งให้กับ activation group นั้นได้. สำหรับเกี่ยวกับการรีเทิร์น แบบปกติจาก call stack entry ที่เก่าที่สุดของ activation group, ดูได้ในหนังสือ ILE HLL programmer’s guides.

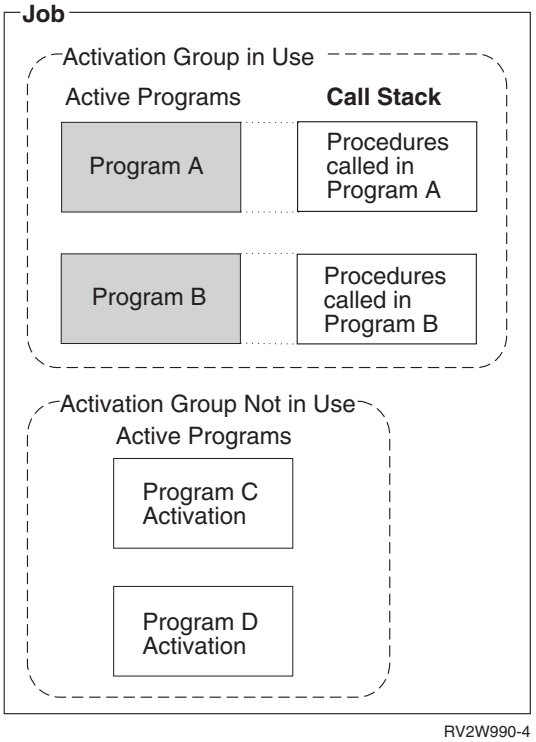
รูปที่ 19 ในหน้า 39 แสดงตัวอย่างของวิธีการออกจาก activation group. ในรูป,โพรซีเดอร์ P1 เป็น call stack entry. สำหรับ activation group ที่ถูกกำหนดชื่อโดยระบบ (ซึ่งถูกสร้างขึ้นด้วย option ACTGRP(*NEW)), การรีเทิร์นแบบปกติจาก P1 เป็นการลบ activation group. และสำหรับ activation group ที่ถูกกำหนดชื่อโดยผู้ใช้ (ซึ่งถูกสร้างขึ้นด้วย option ACTGRP(name)), การรีเทิร์น แบบปกติจาก P1 จะไม่ลบ activation group.



รูปที่ 19. แสดงการออกจาก Activation Group ที่ถูกกำหนดชื่อโดยผู้ใช้และ Activation Group ที่ถูกกำหนดชื่อโดยระบบ

ถ้า activation group ที่ถูกกำหนดชื่อโดยผู้ใช้ถูกค้างไว้ใน job, คุณสามารถลบมันได้โดยใช้คำสั่ง Reclaim Activation Group (RCLACTGRP). คำสั่งนี้ยอมให้คุณลบ activation group ได้. โดยจะใช้ลบได้เฉพาะ activation group ที่ไม่ถูกใช้งานในขณะนั้นเท่านั้น.

รูปที่ 20แสดงงานของ OS/400 ที่มี activation group กลุ่มหนึ่งที่ไม่ถูกใช้งานและมี activation group อีกกลุ่มหนึ่งที่กำลังถูกใช้งาน. activation group จะถูกพิจารณาว่ากำลังถูกใช้งาน เมื่อมี call stack entry สำหรับโพรซีเดอร์ ILE ที่ activate ใน activation group นั้น . คำสั่ง RCLACTGRP ในโปรแกรม A หรือโปรแกรม B ใช้ในการลบ activation group ของโปรแกรม C และโปรแกรม D.



รูปที่ 20. แสดง Activation Group ที่ถูกใช้จะมีรายการอยู่บน Call Stack

เมื่อ activation group ถูกลบโดย ILE, จะเกิดกระบวนการสิ้นสุดการทำงานขึ้น. กระบวนการนี้ประกอบด้วย การเรียกโพรซีเดอร์ exit ที่ถูกเรจิสเตอร์โดยผู้ใช้, data management cleanup, และ language cleanup (เช่นการปิดไฟล์). ดูหัวข้อ “กฎในการจำกัดขอบเขตการบริหารข้อมูล (Data Management Scoping Rules)” ในหน้า 53 สำหรับรายละเอียดเกี่ยวกับกระบวนการจัดการข้อมูลที่เกิดขึ้นเมื่อ activation group ถูกลบ.

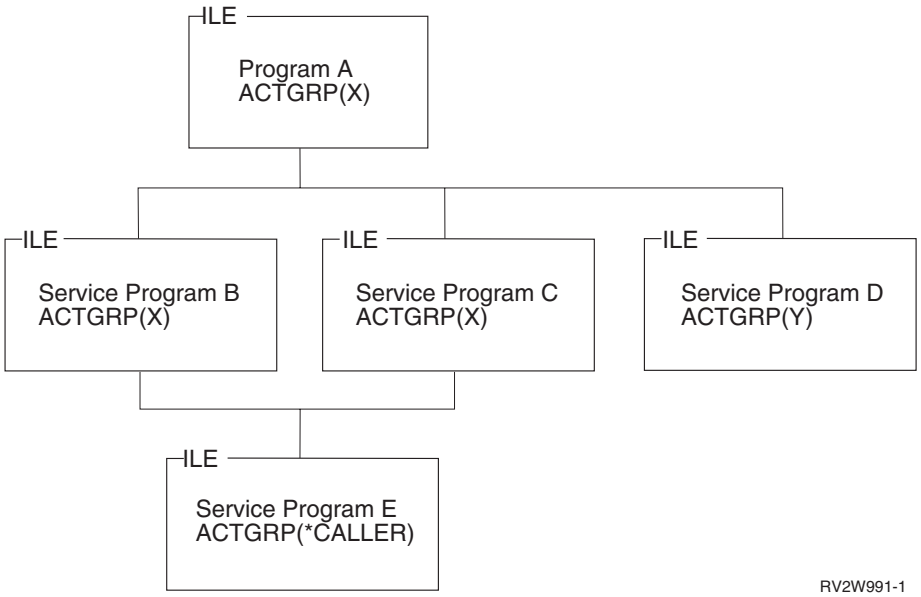
Service Program Activation

หัวข้อนี้จะอธิบายถึงขั้นตอนเฉพาะขั้นตอนที่ระบบใช้ในการ activate เซอร์วิสโปรแกรม. ขั้นตอนทั่วไปสำหรับโปรแกรมและเซอร์วิสโปรแกรม ได้อธิบายไว้แล้วในหัวข้อ “Program Activation” ในหน้า 31. ลักษณะการ activate ที่จะกล่าวต่อไปนี้ เป็นลักษณะเฉพาะสำหรับเซอร์วิสโปรแกรมเท่านั้น:

- การ activate เซอร์วิสโปรแกรม เริ่มต้นทางอ้อมโดยการเป็นส่วนหนึ่งใน dynamic program call ของโปรแกรม ILE.
- การ activate เซอร์วิสโปรแกรม จะมีขั้นตอนการเชื่อมต่อระหว่างโปรแกรม โดยการแม็พ symbolic link เข้ากับ physical link.
- การ activate เซอร์วิสโปรแกรม จะมีกระบวนการตรวจสอบ signature.

เมื่อโปรแกรม ILE มีการ activate เกิดขึ้นครั้งแรกใน activation group จะถูกตรวจสอบว่ามีการเชื่อมต่อกับ เซอร์วิสโปรแกรมอื่นหรือไม่. ถ้าเซอร์วิสโปรแกรมถูกเชื่อมเข้ากับโปรแกรมที่ถูกเรียกทำงาน, เซอร์วิสโปรแกรม เหล่านั้นก็จะถูกเรียกทำงานด้วยโดยถือว่าเป็นส่วนหนึ่งของกระบวนการ dynamic program call เดียวกัน. กระบวนการนี้จะมีการทำซ้ำไปจนกระทั่งเซอร์วิสโปรแกรมที่จำเป็นถูก activate ทั้งหมด.

รูปที่ 21 ในหน้า 41 แสดงการเชื่อมกันระหว่าง ILE โปรแกรม A กับ ILE เซอร์วิสโปรแกรม B, C และ D เซอร์วิสโปรแกรม B และ C ก็เชื่อมเข้ากับ ILE เซอร์วิสโปรแกรม E โดยที่แอ็ตทริบิวต์ของ activation group สำหรับแต่ละโปรแกรมและเซอร์วิสโปรแกรม ถูกแสดงไว้ดังรูป.



รูปที่ 21. Service Program Activation

เมื่อ ILE โปรแกรม A ถูก activate จะเกิดกระบวนการต่างๆ ดังนี้:

- เซอร์วิสโปรแกรมถูกกำหนดตำแหน่งโดยการกำหนดชื่อไลบรารีที่แน่นอน หรือการใช้รายชื่อไลบรารีในขณะนั้น. อ็อพชันนี้คุณสามารถกำหนดได้ในขั้นตอนการสร้างโปรแกรมหรือเซอร์วิสโปรแกรม.
- เช่นเดียวกับโปรแกรม, การ activate เซอร์วิสโปรแกรมจะเกิดขึ้นเพียงครั้งเดียวใน activation group หนึ่ง ๆ. ใน รูปที่ 21เซอร์วิสโปรแกรม E ถูก activate เพียงหนึ่งครั้งแม้ว่าจะถูกใช้โดยเซอร์วิสโปรแกรม ทั้ง B และ C.
- Activation group กลุ่มที่สอง (Y) จะถูกสร้างขึ้นสำหรับเซอร์วิสโปรแกรม D.
- การตรวจสอบ signature จะเกิดขึ้นกับทุกโปรแกรมและเซอร์วิสโปรแกรม.

กระบวนการนี้อาจมองได้ว่า เป็นการสิ้นสุดของกระบวนการเชื่อมต่อที่เริ่มต้นเมื่อโปรแกรมและเซอร์วิสโปรแกรมถูกสร้างขึ้น. คำสั่ง CRTPGM และคำสั่ง CTRSRVPGM จะเก็บชื่อและไลบรารีของเซอร์วิสโปรแกรม อ้างอิงแต่ละตัว. ดัชนีของตารางโพรซีเดอร์และตัวข้อมูลที่ถูก export ก็จะถูกเก็บไว้ในโปรแกรมหรือ เซอร์วิสโปรแกรมของไคลเอนต์ในขณะสร้างโปรแกรมเช่นกัน. กระบวนการ activate เซอร์วิสโปรแกรมสิ้นสุดขั้นตอนของการเชื่อมโปรแกรมโดยการเปลี่ยนสัญลักษณ์อ้างอิง (symbolic reference) เหล่านี้ไปเป็น แอดเดรสที่จะถูกใช้ในขณะรัน.

เมื่อเซอร์วิสโปรแกรมถูก activate ทำให้ การเรียกโพรซิเดอร์แบบสแตติก และตัวข้อมูลที่อ้างอิงถึงโมดูลในเซอร์วิสโปรแกรมอื่นถูกโพรเซส. และจะมีความต้องการกระบวนการนี้ในจำนวนเท่าๆ กัน ถ้าโมดูลถูกรวมโดยการ ก็อปปี้ (bind by copy) เข้าไปในโปรแกรมเดียวกัน. อย่างไรก็ตาม,โมดูลที่ถูกรวมโดยการก็อปปี้จะใช้เวลาในการ activate น้อยกว่าเซอร์วิสโปรแกรม.

การ activate โปรแกรมและเซอร์วิสโปรแกรมต้องการ authority ในโปรแกรมและเซอร์วิสโปรแกรมของ ILE ทุกตัว. ใน รูปที่ 21authority ปัจจุบันของผู้เรียก (caller) โปรแกรม A ถูกใช้ในการตรวจสอบ authority ใน โปรแกรม A และเซอร์วิสโปรแกรมทั้งหมด. และ authority ของโปรแกรม A ก็ถูก

ใช้ในการตรวจสอบ authority ของเซอร์วิสโปรแกรมทั้งหมด. เป็นที่สังเกตว่า authority ของเซอร์วิสโปรแกรม B, C หรือ D ไม่ถูกใช้ในการตรวจสอบ authority ของเซอร์วิสโปรแกรม E.

ขอบเขตการควบคุม

เมื่อมี function check ที่ไม่ถูกจัดการหรือ HLL end verb ถูกใช้งาน. ILE จะย้ายการควบคุมไปยังตัวเรียกของ call stack entry ซึ่งทำหน้าที่เป็นขอบเขตของแอ็พพลิเคชันของคุณ. call stack entry นี้จะถูกเรียกว่า **ขอบเขตการควบคุม**.

จากหัวข้อ ขอบเขตการควบคุม. สำหรับ “ขอบเขตการควบคุมสำหรับ Activation Group ของ ILE” และ “ขอบเขตการควบคุม สำหรับ Default Activation Group ของ OPM” ในหน้า 43 แสดงคำจำกัดของ.

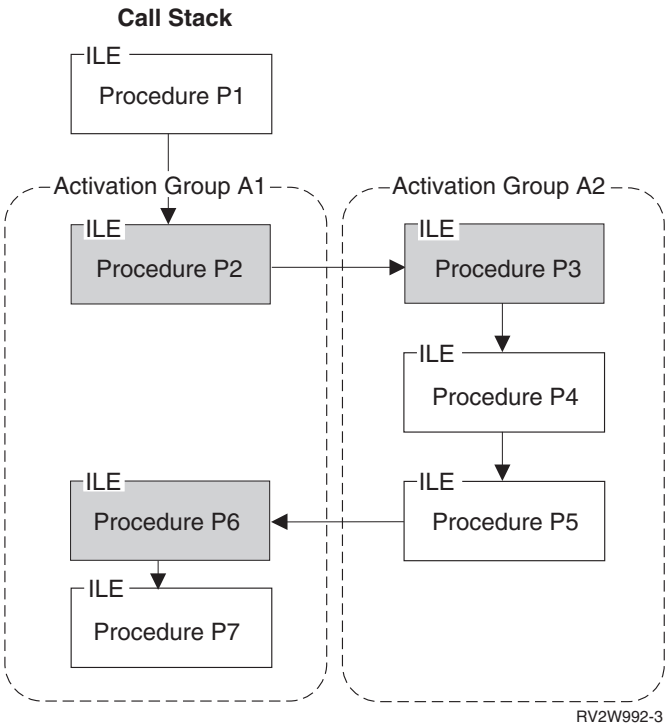
ขอบเขตการควบคุม ไว้ว่าเป็นไปตามลักษณะในข้อใดข้อหนึ่ง ดังต่อไปนี้:

- Call stack entry ของ ILE ใดๆ ที่มี call stack entry ตัวก่อนหน้าเป็น activation group อื่นที่ไม่ใช่ default activation group.
- Call stack entry ของ ILE ใดๆ ที่มี call stack entry ตัวก่อนหน้าเป็นโปรแกรม OPM.

ขอบเขตการควบคุมสำหรับ Activation Group ของ ILE

ตัวอย่างนี้จะแสดงถึงวิธีการที่ ขอบเขตการควบคุม ถูกกำหนดระหว่าง activation group ของ ILE.

รูปที่ 22 ในหน้า 43 แสดงถึง ILE activation group 2 กลุ่มและ ขอบเขตการควบคุม ที่สร้างขึ้นโดยการเรียกหลายครั้ง. โพรซีเดอร์ P2, P3, และ P6 คือโพรซีเดอร์ที่มีแนวโน้มจะเป็น ขอบเขตการควบคุม. ตัวอย่างเช่น เมื่อคุณรันโพรซีเดอร์ P7, โพรซีเดอร์ P6 จะเป็นขอบเขตการควบคุม. เมื่อคุณรันโพรซีเดอร์ P4 หรือ P5 โพรซีเดอร์ P3 จะกลายเป็น ขอบเขตการควบคุม.

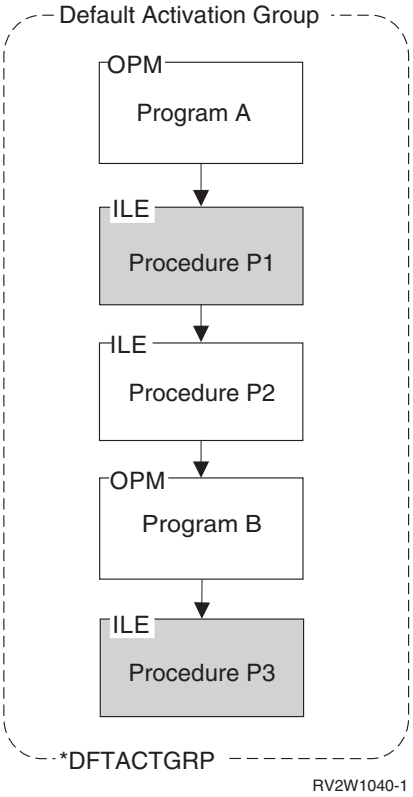


รูปที่ 22. Control Boundaries. โพรซีเดอร์ในพื้นที่แรเงาคือ ขอบเขตการควบคุม.

ขอบเขตการควบคุม สำหรับ Default Activation Group ของ OPM

ตัวอย่างนี้แสดงให้เห็นถึงวิธีการที่ ขอบเขตการควบคุม ถูกกำหนดขึ้นเมื่อโปรแกรม ILE ทำงานใน default activation group ของ OPM.

รูปที่ 23 ในหน้า 44 แสดงถึงโพรซีเดอร์ของ ILE 3 โพรซีเดอร์ คือ P1, P2, และ P3 ทำงานอยู่ใน default activation group ของ OPM. ตัวอย่างนี้ถูกสร้างขึ้นได้โดยการใช้คำสั่ง CRTPGM หรือ CRTSRVPGM ที่มีพารามิเตอร์ ACTGRP(*CALLER). โพรซีเดอร์ P1 และ P3 คือโพรซีเดอร์ที่มีแนวโน้มจะเป็น ขอบเขตการควบคุม เพราะว่า call stack entry ตัวแรก คือโปรแกรม OPM A และ B.



รูปที่ 23. แสดงขอบเขตการควบคุม ใน Default Activation Group.. โพรซีเดอร์ในพื้นที่แรเงาคือ ขอบเขตการควบคุม.

การใช้งานขอบเขตการควบคุม

เมื่อคุณใช้ HLL end verb ของ ILE , ขอบเขตการควบคุม ล่าสุดที่อยู่บน call stack จะถูกใช้เพื่อพิจารณาว่าจะย้ายการควบคุมไปที่ใด. call stack entry ที่อยู่ก่อนหน้า ขอบเขตการควบคุม จะรับการควบคุมหลังจาก ILE จบกระบวนการสิ้นสุดการทำงานทั้งหมด.

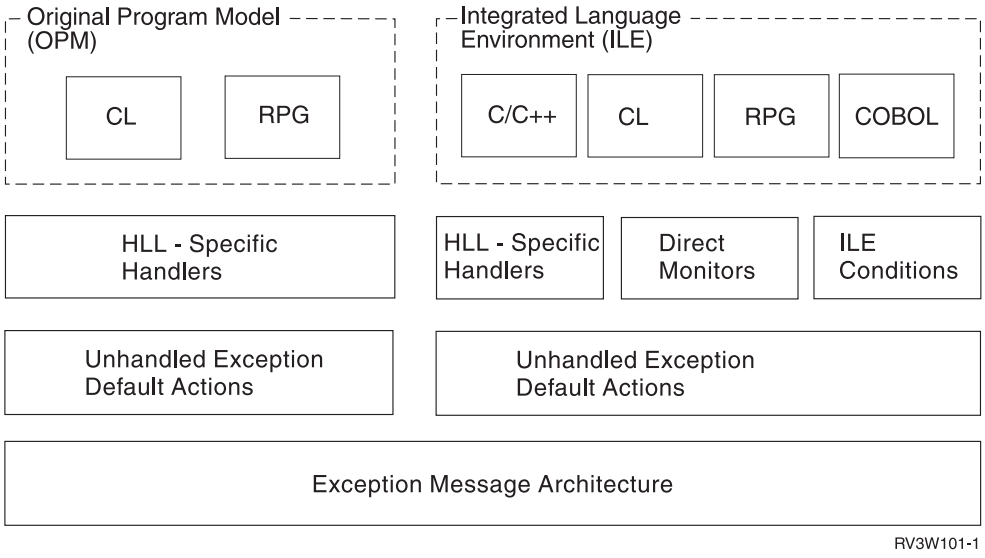
ขอบเขตการควบคุม ถูกใช้เมื่อเกิดฟังก์ชันเช็คแบบ Unhandled ขึ้นในโพรซีเดอร์ของ ILE. ขอบเขตการควบคุมจะกำหนดตำแหน่ง call stack ที่ฟังก์ชันเช็คแบบ Unhandled ถูก **promoted** เป็นสถานะที่ล้มเหลวแบบทั่วไปของ ILE. สำหรับรายละเอียดเพิ่มเติม ดูได้ในหัวข้อ “การจัดการข้อผิดพลาด (Error Handling)” ในหน้า 45.

เมื่อขอบเขตการควบคุม ที่ใกล้ที่สุดคือ call stack entry ที่เก่าที่สุดใน ILE activation group, คำสั่ง HLL end verb หรือฟังก์ชันเช็คแบบ Unhandled จะทำให้ activation group ถูกลบออกไป. เมื่อขอบเขตการควบคุมที่ใกล้ที่สุดไม่ใช่ call stack entry ที่เก่าที่สุด, การควบคุมจะย้อนกลับไปยัง call stack entry ที่อยู่ก่อนขอบเขตการควบคุม. ทำให้ activation group จะไม่ถูกลบเพราะว่า call stack entry ตัวก่อนหน้านี้ยังคงอยู่ใน activation group เดียวกัน.

รูปที่ 22 ในหน้า 43 แสดงให้เห็นว่าโพรซีเดอร์ P2 และ P3 เป็น call stack entry ที่เก่าที่สุดในแต่ละ activation group. การใช้ HLL end verb ในโพรซีเดอร์ P2, P3, P4, หรือ P5 (แต่ไม่ใช่ P6 หรือ P7) อาจทำให้ activation group A2 ถูกลบไปได้.

การจัดการข้อผิดพลาด (Error Handling)

หัวข้อนี้อธิบายความสามารถในระดับสูงของการจัดการข้อผิดพลาดของโปรแกรม OPM และโปรแกรม ILE. เพื่อที่จะเข้าใจถึงความสอดคล้องของความสามารถเหล่านี้กับสถาปัตยกรรม exception message,ดังแสดงใน รูปที่ 24. สำหรับข้อมูลอ้างอิงเฉพาะและแนวคิดเพิ่มเติมสามารถดูได้ใน บทที่ 9, “การจัดการ Exception และ Condition”, ในหน้า 133.รูปที่ 24แสดงถึงภาพโดยรวมของการจัดการข้อผิดพลาด. หัวข้อนี้จะเริ่มต้นที่เลเยอร์ล่างสุดของภาพและต่อเนื่องไปจนถึงเลเยอร์บนสุด. ซึ่งแสดงถึงฟังก์ชันที่คุณอาจจะใช้เพื่อจัดการข้อผิดพลาดในโปรแกรม OPM หรือโปรแกรม ILE.



รูปที่ 24. แสดงการจัดการข้อผิดพลาดสำหรับ ILE และ OPM

Job Message Queues

message queue จะมีในทุกๆ call entry ของ stack ในแต่ละงานของOS/400. โดย message queue จะให้ความสะดวกในการรับ-ส่งข้อมูลและข้อความ exception ระหว่างโปรแกรมและโพรซีเดอร์ที่รันอยู่บน call stack. message queue นี้มีชื่อเรียกอีกอย่างหนึ่งว่า **call message queue**.

Call message queue ถูกระบุโดยใช้ชื่อของโปรแกรม OPM หรือโพรซีเดอร์ ILE ที่อยู่บน call stack. ชื่อของ โพรซีเดอร์หรือชื่อของโปรแกรมสามารถถูกใช้ในการกำหนด call stack entry ที่เป็นเป้าหมายสำหรับการส่งข้อความ. เนื่องจากชื่อของโพรซีเดอร์ ILE อาจจะซ้ำกันได้ ดังนั้นชื่อของโมดูล ILE และชื่อของโปรแกรมหรือ เซอร์วิสโปรแกรม ILE อาจถูกใช้เป็นตัวกำหนดได้. เมื่อโปรแกรมเดียวกันหรือโพรซีเดอร์เดียวกันมี call stack entry หลายตัว call message queue ที่ใกล้ที่สุดจะถูกใช้.

นอกจากนี้ในแต่ละงานของ OS/400 จะมี **External Message Queue** หนึ่งคิว . โปรแกรมและโพรซีเดอร์ทุกตัวที่รันอยู่ในงานจะอาศัยคิวนี้ในการรับ-ส่งแมสเสจระหว่างงานโต้ตอบกับผู้ใช้.

ไอบีเอ็ม มีข้อมูลออนไลน์เกี่ยวกับการรับ-ส่งข้อความ Exception โดยการใช้ API ที่จัดการด้านแมสเสจ. โปรดดูส่วน *API* ของหมวด **Programming** สำหรับ iSeries Information Center.

ข้อความ Exception และวิธีการส่ง

ในหัวข้อนี้จะอธิบายถึงชนิดของข้อความ exception และวิธีการที่ข้อความ exception ถูกส่งไป.

การจัดการข้อผิดพลาดของ ILE และ OPM มีพื้นฐานจากชนิดของข้อความ exception. โดยที่ **ข้อความ exception** จะแสดงถึงชนิดของแมสเสจ ในแบบต่างๆ ดังนี้:

Escape (*ESCAPE)
แสดงถึงข้อผิดพลาดที่ทำให้โปรแกรมจบลงอย่างไม่ปกติ โดยที่งานของโปรแกรมนั้นยังไม่เสร็จสิ้น. คุณจะไม่ได้รับการควบคุม หลังจากการส่งข้อความ exception แบบ escape.

Status (*STATUS)
อธิบายสถานะของงานที่ทำโดยโปรแกรม. คุณอาจได้รับการควบคุมภายหลังจากการส่งแมสเสจ ชนิดนี้. โดยขึ้นอยู่กับวิธีการที่โปรแกรมทางฝั่งรับจัดการกับข้อความสถานะ.

Notify (*NOTIFY)
แสดงสถานะที่ต้องการการกระทำเพื่อแก้ไขให้ถูกต้อง หรือแสดงคำตอบจากโปรแกรมที่เป็นตัวเรียก (calling program). คุณอาจได้รับการควบคุมภายหลังจากการส่งแมสเสจชนิดนี้. โดยขึ้นอยู่กับวิธีการที่โปรแกรมทางฝั่งรับจัดการกับข้อความแจ้ง.

Function Check
แสดงสถานะในการสิ้นสุดโดยโปรแกรมไม่ได้คาดคิดไว้. ฟังก์ชันเช็คของ ILE คือ CEE9901, ซึ่งจะเป็นแมสเสจพิเศษที่ส่งโดยระบบเท่านั้น. ส่วนฟังก์ชันเช็คของ OPM คือ escape message ที่มี message ID เป็น CPF9999.

ไอบีเอ็ม มีข้อมูลออนไลน์เกี่ยวกับการรับ-ส่งข้อความ exception และข้อความชนิดอื่นๆ ของ OS/400. โปรดดู ส่วน *API* ของหมวด **Programming** สำหรับ iSeries Information Center.

การส่งข้อความ Exception มีวิธีการดังต่อไปนี้:

- สร้างขึ้นโดยระบบ

  OS/400 (รวมทั้ง HLL ของคุณ) สร้างข้อความ Exception ขึ้นเพื่อแสดงข้อผิดพลาดของโปรแกรมหรือให้ข้อมูลเกี่ยวกับสถานะของโปรแกรม.
- API ที่จัดการด้านแมสเสจ

  Send Program Message (QMHSNDPM) API สามารถใช้ในการส่ง exception message ไปยัง call message queue ที่ต้องการได้.
- ILE API

  Signal a Condition (CEESGL) bindable API สามารถใช้ในการตั้งเงื่อนไขของ ILE. ซึ่งเป็นเงื่อนไขที่ทำให้เกิด ข้อความ exception แบบ escape หรือแบบ status.
- Verb เฉพาะของแต่ละภาษา

สำหรับ ILE C และ ILE C++, ฟังก์ชัน raise() เป็นตัวส่งสัญญาณ C signal. ในขณะที่ทั้ง ILE RPG หรือ ILE COBOL จะไม่มีฟังก์ชันที่ทำงานในลักษณะนี้.

วิธีการที่ข้อความ Exception ถูกจัดการ

เมื่อคุณหรือระบบส่งข้อความ exception,กระบวนการ exception ก็จะเริ่มขึ้น. กระบวนการนี้จะดำเนินไปจนกระทั่ง exception ถูกจัดการ, นั่นคือเมื่อข้อความ exception ถูกเปลี่ยนแปลงเพื่อแสดงว่ามันถูกจัดการแล้ว.

ระบบเปลี่ยนแปลงข้อความ exception เพื่อแสดงสถานะว่าข้อความ exception ถูกจัดการแล้ว เมื่อมันเรียกตัวจัดการ exception สำหรับ call message queue ของ OPM. HLL ของ ILE จะแก้ไขข้อความ exception ก่อนที่ตัวจัดการ exception จะถูกเรียกสำหรับ call message queue ของ ILE. ผลที่ได้ก็คือ, การจัดการข้อผิดพลาดแบบเฉพาะ HLL จะพิจารณาว่าข้อความ exception ถูกจัดการเมื่อตัวจัดการของคุณถูกเรียก. ถ้าคุณไม่ใช้การจัดการข้อผิดพลาดแบบเฉพาะ HLL, HLL ของ ILE ก็สามารถที่จะจัดการกับข้อความ exception หรือยอมให้กระบวนการ exception ดำเนินต่อไปได้. ในการพิจารณาถึงการทำงานเบื้องต้นของ HLL สำหรับข้อความ exception ที่ไม่ถูกจัดการ ดูได้จากคู่มืออ้างอิงของ HLL ของ ILE ด้วย.

ความสามารถเพิ่มเติมของ ILE จะทำให้คุณผ่านข้ามการจัดการเฉพาะสำหรับแต่ละภาษา. ความสามารถนี้รวมไปถึงตัวจัดการแบบตรวจสอบโดยตรง และตัวจัดการเงื่อนไขของ ILE. เมื่อใช้ความสามารถเหล่านี้ คุณจะต้องรับผิดชอบในการเปลี่ยนข้อความ exceptionเพื่อแสดงว่า ข้อความ exception ถูกจัดการแล้ว. ถ้าคุณไม่เปลี่ยนข้อความ exception,ระบบจะยังดำเนินกระบวนการ exception ต่อไป โดยพยายามที่จะหาตำแหน่งของตัวจัดการ exception ตัวอื่น. หัวข้อ “ชนิดของ Exception Handler” ในหน้า 49 มีรายละเอียดของตัวจัดการแบบตรวจสอบโดยตรงและตัวจัดการเงื่อนไขของ ILE. ไอบีเอ็ม มีข้อมูลออนไลน์ที่อธิบายถึงวิธีการเปลี่ยนข้อความ exception. โปรดดู Change Exception Message (QMHCHGEM) API ในส่วน *API* ของหมวด **Programming** ของ iSeries Information Center.

การคืนสภาพหลังจาก Exception

คุณอาจต้องการที่จะดำเนินกระบวนการต่อหลังจากเกิด exception. การกลับคืนจากข้อผิดพลาดเป็นสิ่งที่แอ็พพลิเคชันทูลที่ดีควรมี ซึ่งทำให้โปรแกรมของคุณมีความทนต่อความผิดพลาด. สำหรับโปรแกรม ILE และ OPM ระบบมีแนวคิดหนึ่งที่เรียกว่า **resume point**. ซึ่งเป็นชุดคำสั่งแรกที่ทำงานทันทีหลังการเกิด exception. หลังจากจัดการกับ exception คุณอาจจะดำเนินโพรเซสต่อได้ที่ resume point. สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับวิธีการใช้และเปลี่ยนแปลง resume point ดูได้ใน บทที่ 9, “การจัดการ Exception และ Condition”, ในหน้า 133.

การกระทำที่เป็นดีฟอลต์สำหรับ Unhandled Exception

ถ้าคุณไม่จัดการข้อความ exception ใน HLL ของคุณ , ระบบจะปฏิบัติตามการทำงานที่เป็นดีฟอลต์ของ Unhandled Exception.

รูปที่ 24 ในหน้า 45 แสดงถึงการทำงานที่เป็นดีฟอลต์สำหรับ Unhandled Exception โดยมีพื้นฐานว่า exception ถูกส่งไปยังโปรแกรม OPM หรือโปรแกรม ILE. การทำงานที่เป็นดีฟอลต์ที่ต่างกันระหว่าง 2 โปรแกรมจะสร้างข้อแตกต่างพื้นฐานให้กับความสามารถของการจัดการข้อผิดพลาด.

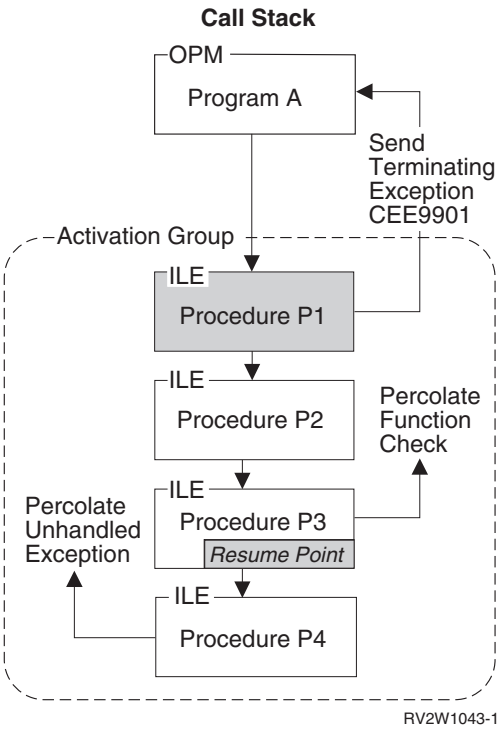
สำหรับ Unhandled Exception ในโปรแกรม OPM จะสร้างข้อความแบบ escape พิเศษที่เรียกว่าข้อความฟังก์ชันเช็ค. ซึ่งมี message ID เป็น CPF9999. มันถูกส่งไปยัง call message queue ของ call stack entry ที่ทำให้เกิดข้อความ exception แรก. ถ้าข้อความฟังก์ชันเช็คไม่ถูกจัดการ,ระบบก็จะลบ call stack entry ทั้งหมดออกไป. แล้วระบบจะส่งข้อความแบบฟังก์ชันเช็คไปยัง call stack entry ก่อนหน้านี้. กระบวนการนี้จะดำเนินไปจนกระทั่งข้อความฟังก์ชันเช็คถูกจัดการ. ถ้าข้อความฟังก์ชันเช็คไม่ถูกจัดการเลย job ก็จะสิ้นสุดลง.

สำหรับ Unhandled Exception ของโปรแกรม ILE จะถูกปล่อยให้ผ่านไปยัง call stack entry message queue ก่อนหน้านี้. **Percolation** เกิดขึ้นเมื่อข้อความ exception ถูกย้ายไปยัง call message queue ก่อนหน้าคิวปัจจุบัน. ซึ่งจะก่อให้เกิดการส่งข้อความ exception ที่เหมือนกันไปยัง call message queue ก่อนหน้านี้. และเมื่อเกิดเหตุการณ์เช่นนี้, กระบวนการ exception จะยังดำเนินต่อไปยัง call stack entry ก่อนหน้านี้.

รูปที่ 25 ในหน้า 49 แสดงข้อความ exception ที่ไม่ถูกจีดการภายใน ILE. ในตัวอย่างนี้, โพรซีเดอร์ P1 เป็นขอบเขตการควบคุม. และยังเป็น call stack entry ใน activation group ที่เก่าที่สุด. โพรซีเดอร์ P4 ทำให้เกิดข้อความ exception ที่ไม่ถูกจัดการ. Percolation ของunhandled exception ดำเนินไปจนกว่าจะถึงขอบเขตการควบคุม หรือจนกระทั่งข้อความ exception ถูกจัดการ. Unhandled exception จะถูกแปลงกลับไปเป็นฟังก์ชันเช็คเมื่อมันถูกปล่อยผ่านไปจนถึง ขอบเขตการควบคุม. ถ้า exception นั้นเป็นแบบ escape , ก็จะมีการสร้างฟังก์ชันเช็คขึ้น. ถ้า exception เป็นแบบ notify,คำตอบที่กำหนดไว้จะถูกส่งไป, และ exception จะถูกจัดการ, และส่วนที่แจ้งสถานะจะยังทำงานต่อไปได้. แต่ถ้าเป็นแบบ status แล้ว, exception นั้นก็จะถูกจัดการ, และส่วนที่แจ้งสถานะก็สามารถทำงานต่อไปได้. resume point (ที่แสดงใน โพรซีเดอร์ P3) ถูกใช้ในการกำหนด call stack entry ที่กระบวนการ exception ของฟังก์ชันเช็คจะดำเนินต่อไป. สำหรับ ILE, ขั้นตอนต่อไปคือการส่งข้อความ exception แบบฟังก์ชันเช็คพิเศษไปยัง call stack entry. ซึ่งก็คือโพรซีเดอร์ P3 ในตัวอย่างนี้นั้นเอง.

ข้อความ Exception แบบฟังก์ชันเช็คสามารถที่จะถูกจัดการ หรืออาจถูกปล่อยผ่านไปยัง ขอบเขตการควบคุม ก็ได้. ถ้ามันถูกจัดการ, กระบวนการปกติจะดำเนินต่อไปและกระบวนการ exception ก็จะสิ้นสุดลง. ถ้าข้อความฟังก์ชันเช็คถูกปล่อยผ่านไปยัง ขอบเขตการควบคุม, ILE จะพิจารณาว่าแอ็พพลิเคชันจะต้องสิ้นสุดการทำงานลงเนื่องจากข้อผิดพลาดที่ไม่คาดหวัง. ข้อความ exception ของความล้มเหลวแบบทั่วไปจะถูกกำหนดไว้โดย ILE สำหรับทุกๆ ภาษา. นั้นคือค่า CEE9901 ซึ่งข้อความนี้จะถูกส่งไปยังตัวเรียกของ ขอบเขตการควบคุม.

การกระทำที่เป็นดีฟอลต์สำหรับข้อความ exception ที่ไม่ถูกจัดการ ที่กำหนดไว้ใน ILE ยอมให้คุณกลับออกจากสถานะที่ผิดพลาดที่เกิดขึ้นในแอ็พพลิเคชันที่มีหลายภาษา. สำหรับข้อผิดพลาดที่ไม่คาดหวังนั้น, ILE จะสร้าง failure message ขึ้นสำหรับทุกๆ ภาษา. ซึ่งเป็นการพัฒนาความสามารถในการรวมแอ็พพลิเคชันจากแหล่งที่ต่างกัน.



รูปที่ 25. แสดงการกระทำที่เป็นดีฟอลต์ของ Unhandled Exception.

ชนิดของ Exception Handler

หัวข้อนี้เป็นการอธิบายโดยรวมในเรื่องชนิดของ exception handler สำหรับโปรแกรม OPM และโปรแกรม ILE. ดังแสดงใน รูปที่ 24 ในหน้า 45 ซึ่งก็คือเลเยอร์บนสุดของสถาปัตยกรรมข้อความ exception. เมื่อเปรียบเทียบกันแล้ว ILE จะมีความสามารถในการจัดการกับ exception มากกว่า OPM.

สำหรับโปรแกรม OPM, การจัดการข้อผิดพลาดแบบเฉพาะ HLL จะให้รูทีนในการจัดการตั้งแต่ 1 รูทีนขึ้นไปสำหรับแต่ละ call stack entry. ระบบจะเรียกรูทีนที่เหมาะสม เมื่อ exception ถูกส่งไปยังโปรแกรม OPM.

การจัดการข้อผิดพลาดแบบเฉพาะ HLL ใน ILE ก็มีความสามารถในลักษณะเดียวกัน. แต่ ILE ยังมีชนิดของ exception handler เพิ่มเติมอีก. ซึ่งตัวจัดการแต่ละชนิดช่วยให้คุณสามารถควบคุมโครงสร้างของข้อความ exception ได้โดยตรงและยังช่วยให้คุณสามารถข้ามการจัดการข้อผิดพลาดแบบเฉพาะ HLL ได้อีกด้วย. ชนิดของตัวจัดการสำหรับ ILE ได้แก่:

ตัวจัดการแบบตรวจสอบโดยตรง
ตัวจัดการแบบเงื่อนไขของ ILE

ในการพิจารณาว่า HLL ของคุณสนับสนุน ตัวจัดการเหล่านี้หรือไม่, ดูได้ใน ILE HLL programmer's guide.

ตัวจัดการแบบตรวจสอบโดยตรง ช่วยให้คุณกำหนดตัวมอนิเตอร์เฝ้าดู exception รอบ ๆ คำสั่ง HLL ในระดับ source ได้โดยตรง. สำหรับ ILE C ความสามารถนี้จะใช้ได้ผ่านไดเรกทีฟ #pragma. ส่วน ILE COBOL ไม่สามารถกำหนดตัวมอนิเตอร์ได้โดยตรงเหมือน ILE C. โปรแกรม ILE COBOLสามารถโค้ดการใช้หรือไม่ใช้ตัวจัดการในซอร์สโค้ดได้โดยตรง. อย่างไรก็ตาม, คำสั่งเช่น

```
ADD a TO b ON SIZE ERROR imperative
```

จะถูกโยงภายในเพื่อใช้กลไกเดียวกัน. ดังนั้น, ในเทอมของลำดับความสำคัญของตัวจัดการใดที่จะได้รับการควบคุมตัวแรก, เช่น คำสั่งที่อยู่ในของเขตของ imperative จะได้การควบคุมก่อนตัวจัดการของ ILE (ที่เรจิสเตอร์ ผ่าน CEEHDLR). ตัวควบคุมจึงดำเนินการใช้คำสั่งที่ประกาศไว้ใน COBOL.

ตัวจัดการแบบเงื่อนไขของ ILE ช่วยให้คุณสามารถ**เรจิสเตอร์** exception handler ได้ในเวลารันไทม์. ตัวจัดการเงื่อนไขแบบ ILE จะถูกเรจิสเตอร์สำหรับ call stack entry ที่เฉพาะเจาะจงไว้. ในการเรจิสเตอร์ตัวจัดการเงื่อนไขแบบ ILE, ให้ใช้ Register a User-Written Condition Handler (CEEHDLR) bindable API. ซึ่ง API นี้จะให้ระบุโพรซีเดอร์ในขณะรัน โพรซีเดอร์นี้จะได้รับการควบคุมเมื่อเกิด exception ขึ้น. API ของ CEEHDLR ต้องการความสามารถในการประกาศ (declare) และตั้งค่าพอยน์เตอร์ไปยังโพรซีเดอร์ของภาษาที่คุณใช้. CEEHDLR ถูกสร้างขึ้นให้เป็นฟังก์ชันในตัว. ดังนั้น,แอดเดรสของมันจึงไม่สามารถระบุ และไม่สามารถถูกเรียกผ่านตัวชี้ของโพรซีเดอร์ได้. ตัวจัดการแบบเงื่อนไขของ ILE อาจถูก **ถอนการเรจิสเตอร์** ได้โดยการใช้ Unregister a User-Written Condition Handler (CEEHDLU) bindable API.

ทั้ง OPM และ ILE สนับสนุนตัวจัดการแบบเฉพาะ HLL. **ตัวจัดการแบบเฉพาะ HLL** เป็นคุณลักษณะของภาษาที่กำหนดไว้สำหรับจัดการกับข้อผิดพลาด. ตัวอย่างเช่น, ใน ILE C สามารถใช้ฟังก์ชัน signal ในการจัดการข้อความ exception. HLLกำหนดการจัดการข้อผิดพลาดในภาษาRPG รวมถึงความสามารถในการจัดการ exception ในระดับข้อความเดียว (E extender),กลุ่มของข้อความ (MONITOR), หรือในระดับโพรซีเดอร์ทั้งหมด (*PSSR และรูทีนย่อย INFSR). ตัวจัดการข้อผิดพลาดแบบเฉพาะ HLL ใน COBOL ใช้คำสั่ง USE สำหรับการจัดการข้อผิดพลาดทาง I/O และข้อบังคับในคำสั่งที่เป็นเงื่อนไข เช่น ON SIZE ERROR และ AT INVALID KEY.

ลำดับความสำคัญของ exception handler เป็นสิ่งสำคัญ ถ้าคุณใช้ทั้งตัวจัดการแบบเฉพาะ HLL และ exception handler ของ ILE ชนิดอื่น.

รูปที่ 26 ในหน้า 52 แสดง call stack entry ทั้งหมดของโพรซีเดอร์ P2. ในตัวอย่างนี้, ตัวจัดการทั้ง 3 ชนิดถูกกำหนดให้กับ call stack entry เดียว. ตัวอย่างนี้อาจไม่ใช่ตัวอย่างในทางปฏิบัติ, แต่มันมีทางเป็นได้ที่มีตัวจัดการทั้งสามชนิด. เนื่องจากตัวจัดการทั้งสามชนิดได้ถูกกำหนดระดับความสำคัญไว้แล้ว. ลำดับความสำคัญแสดงได้ดังในภาพ.เมื่อข้อความ exception message ถูกส่งไป, exception handler จะถูกเรียกตามลำดับดังนี้:

1. ตัวจัดการแบบตรวจสอบโดยตรง

   เริ่มแรกเมื่อมีการร้องขอเกิดขึ้น, ตัวจัดการแบบตรวจสอบโดยตรง. และคำสั่งโคบอลที่เป็นเงื่อนไขแบบ imperative จะได้รับการควบคุมก่อนตัวจัดการแบบตัวจัดการแบบเงื่อนไขของ ILE. ในทำนองเดียวกัน, ตัวจัดการแบบเงื่อนไขของ ILE จะได้รับการควบคุมก่อนตัวจัดการแบบเฉพาะ HLL.

ถ้าตัวจัดการแบบตรวจสอบโดยตรงถูกกำหนดรอบ ๆ คำสั่งที่เกิด exception, ตัวจัดการเหล่านั้นจะถูกเรียกก่อนตัวจัดการแบบเฉพาะ HLL. ตัวอย่างเช่น ถ้าโพรซีเดอร์ P2 ใน รูปที่ 26 ในหน้า 52 มีตัวจัดการแบบเฉพาะ HLL และโพรซีเดอร์ P1 มีตัวจัดการแบบตรวจสอบโดยตรง , ตัวจัดการของ P2' จะถูกพิจารณาก่อนตัวจัดการแบบตรวจสอบโดยตรงของ P1'.

ตัวจัดการแบบตรวจสอบโดยตรงสามารถถูก nest ได้. ตัวจัดการที่ถูกกำหนดไว้ใน nested direct monitor ที่ลึกที่สุดจะถูกเลือกเป็นอันดับแรกจาก nested monitor จำนวนมากที่มีหมายเลขแสดงลำดับความสำคัญเดียวกัน.

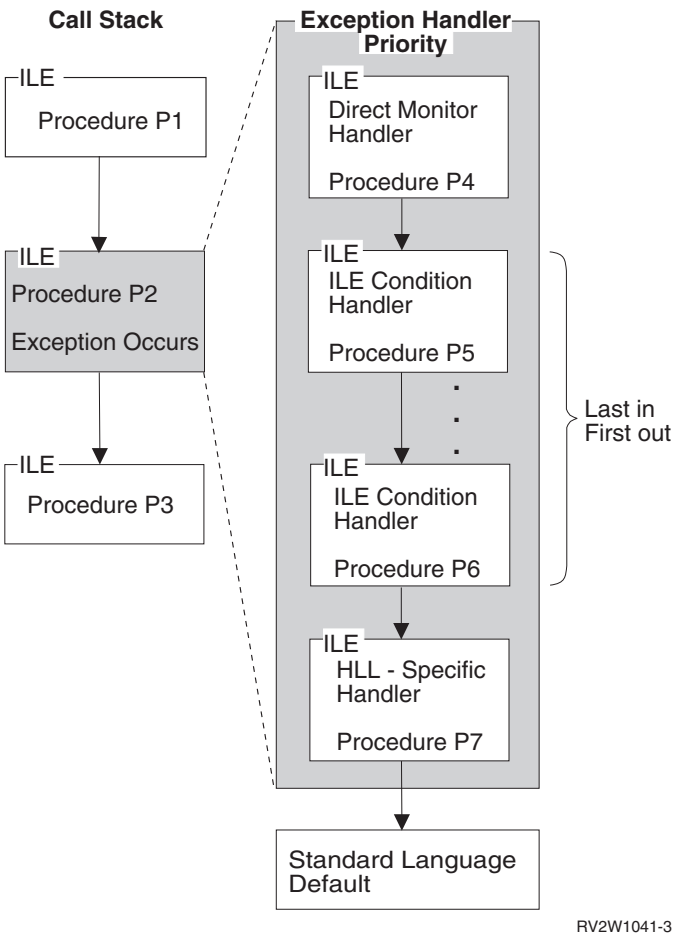
2. ตัวจัดการแบบเงื่อนไขของ ILE

   ถ้าตัวจัดการแบบเงื่อนไขของ ILE ถูกเรจิสเตอร์ call stack entry ไว้, ตัวจัดการนี้จะถูกเรียกเป็นลำดับที่สอง. อาจมีตัวจัดการ ILE condition หลายๆ ตัวที่ถูกบันทึกไว้. ทั้งโพรซีเดอร์ P5 และ P6 ต่างก็เป็นตัวจัดการแบบเงื่อนไขของ ILE. เมื่อมีตัวจัดการแบบเงื่อนไขของ ILEหลายตัวถูกบันทึก call stack entry เดียวกัน, ระบบจะเรียกตัวจัดการนั้นตามลำดับแบบ last-in-first-out (LIFO). หากคุณ แยกประเภท COBOL statement-scoped conditional imperative เป็น HLL-specific handler, imperative มีระดับความสำคัญ มากกว่า ILE condition handler. โดยทั่วไปแล้ว, HLL-specific handlers มี ระดับความสำคัญน้อยที่สุด, น้อยกว่า direct monitor handler และ condition handler. exception หนึ่งคือ COBOL statement-scoped condition imperative, ซึ่งเป็น HLL-specific handler และมีระดับความสำคัญเท่ากันกับ direct monitor handler.

3. ตัวจัดการแบบเฉพาะ HLL

   ตัวจัดการชนิดนี้ จะถูกเรียกเป็นลำดับสุดท้าย.

ระบบสิ้นสุดกระบวนการ exception เมื่อข้อความ exception ถูกเปลี่ยนแปลงเพื่อแสดงว่ามันถูกจัดการ. ถ้าคุณใช้ตัวจัดการแบบตรวจสอบโดยตรง หรือตัวจัดการแบบเงื่อนไขของ ILE , การเปลี่ยนแปลงข้อความ exception จะเป็นหน้าที่ของคุณ. Control Action หลาย ๆ อย่างยังมีอยู่. ตัวอย่างเช่น, คุณสามารถกำหนดวิธีจัดการให้เป็นแบบ Control Action. ตราบใดที่ข้อความ exception ยังคงไม่ถูกจัดการ, ระบบจะยังคงหาตัวจัดการ exception โดยใช้ลำดับความสำคัญที่กำหนดไว้ก่อนหน้านี้. ถ้า exception ยังไม่ถูกจัดการ ใน call stack entry ปัจจุบัน, percolation ก็จะถูกปล่อยผ่านไปยัง call stack entry ก่อนหน้านี้. ถ้าคุณไม่ใช้การจัดการแบบเฉพาะ HLL แล้ว , HLL ของ ILE สามารถเลือกที่จะยอมให้การจัดการ exception ดำเนินต่อไปยัง call stack entry ก่อนหน้านี้.



รูปที่ 26. Exception Handler Priority

เงื่อนไขของ ILE

เพื่อให้เกิดความเหมือนกันระหว่างระบบมากขึ้น , ILE ได้กำหนดคุณสมบัติที่ทำให้คุณทำงานกับเงื่อนไขต่างๆ ของข้อผิดพลาดได้. **เงื่อนไขของ ILE** เป็นตัวแทนของเงื่อนไขความผิดพลาดใน HLL ที่ไม่ขึ้นกับระบบ. สำหรับ OS/400 แต่ละเงื่อนไขของ ILE จะมีข้อความ exception ที่สอดคล้องกัน. เงื่อนไขของ ILE จะถูกแทนด้วย condition token. **Condition token** เป็นโครงสร้างข้อมูลขนาด 12 ไบต์ ซึ่งจะสอดคล้องกันระหว่างระบบหลายๆ ระบบ. โครงสร้างข้อมูลนี้บรรจุข้อมูลที่ยอมให้คุณสร้างเงื่อนไขด้วยข้อความ exceptionที่กล่าวมาแล้ว.

เพื่อที่จะเขียนโปรแกรมที่สามารถข้ามระบบได้, คุณจำเป็นต้องใช้ตัวจัดการแบบเงื่อนไขของ ILE และ ILE condition token. สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับเงื่อนไขของ ILE ดูจากใน บทที่ 9, “การจัดการ Exception และ Condition”, ในหน้า 133.

กฎในการจำกัดขอบเขตการบริหารข้อมูล (Data Management Scoping Rules)

กฎในการจำกัดขอบเขตการบริหารข้อมูล จะควบคุมการใช้รีซอร์สการจัดการข้อมูล (data management resource). รีซอร์สเหล่านี้เป็นอ็อบเจ็กต์ชั่วคราวที่อนุญาตให้โปรแกรมทำงานกับการบริหารข้อมูล. ตัวอย่างเช่น, เมื่อโปรแกรมเปิดไฟล์ๆ หนึ่ง อ็อบเจ็กต์จะเรียก open data path (ODP) ที่ถูกสร้างขึ้นเพื่อติดต่อระหว่างโปรแกรมกับไฟล์. เมื่อโปรแกรมต้องการเปลี่ยนกระบวนการทำงานของไฟล์ ระบบก็จะสร้างอ็อบเจ็กต์ใหม่ขึ้นมาทับของเดิม.

กฎในการจำกัดขอบเขตการบริหารข้อมูล กำหนดเมื่อรีซอร์สถูกแบ่งใช้โดยโปรแกรมหรือโพรซีเดอร์ทั้งหลายที่ทำงานอยู่บน call stack. ตัวอย่างเช่น, การเปิดไฟล์ที่สร้างด้วยพารามิเตอร์ SHARE (*YES) หรืออ็อบเจ็กต์ commitment definition สามารถถูกใช้โดยโปรแกรมหลายโปรแกรมในเวลาเดียวกัน. ความสามารถในการแบ่งใช้รีซอร์สการจัดการข้อมูล ขึ้นอยู่กับระดับของการจำกัดขอบเขตของรีซอร์สการจัดการข้อมูลนั้น.

กฎในการจำกัดขอบเขตการบริหารข้อมูล ยังกำหนดการคงอยู่ของรีซอร์ส. ระบบจะลบรีซอร์สที่ไม่ได้ใช้งานออกจากงานโดยอัตโนมัติ โดยขึ้นอยู่กับกฎเกณฑ์ในการจำกัดขอบเขต. จากผลของการลบรีซอร์สโดยอัตโนมัตินี้ ทำให้งานใช้พื้นที่ในการเก็บข้อมูลน้อยลงและเพิ่มประสิทธิภาพการทำงานของงานนั้น.

ILE จัดระเบียบให้กฎในการจำกัดขอบเขตการบริหารข้อมูล ของทั้งโปรแกรม OPM และโปรแกรม ILE ให้อยู่ในระดับของการจำกัดขอบเขต ดังนี้:

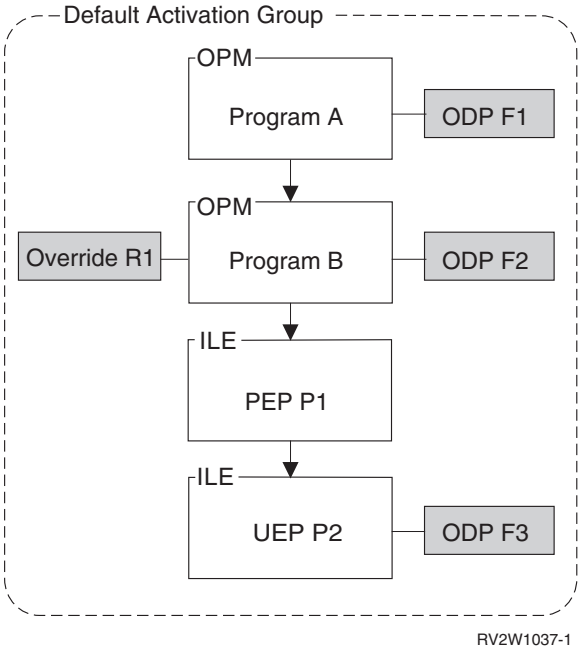
Call
Activation group
Job

การระบุระดับของการจำกัดขอบเขตว่าอยู่ในระดับใดบ้างนั้นขึ้นอยู่กับรีซอร์สการจัดการข้อมูลที่ใช้. ถ้าไม่เลือกระดับของการจำกัดขอบเขตเลย, ระบบจะเลือกระดับใดระดับหนึ่งเป็นค่าดีฟอลต์ให้.

สำหรับรายละเอียดวิธีการที่รีซอร์สการจัดการข้อมูล สนับสนุนระดับของการจำกัดขอบเขต อยู่ใน บทที่ 11, “การวางขอบเขตในการบริหารข้อมูล”, ในหน้า 149.

การวางขอบเขตการ Call

การวางขอบเขตการ Call กิดขึ้นเมื่อรีซอร์สการจัดการข้อมูลเชื่อมต่อกับ call stack entry ที่สร้างรีซอร์สนั้น. รูปที่ 27 ในหน้า 54 แสดงตัวอย่างของการวางขอบเขตการ Call. ที่มักจะเป็นระดับของการวางขอบเขตที่เป็นดีฟอลต์ของโปรแกรมที่รันใน default activation group. ในรูปนี้ โปรแกรม OPM A, โปรแกรม OPM B, หรือโพรซีเดอร์ ILE P2 อาจเลือกที่จะรีเทิร์นโดยไม่มีการปิดไฟล์ที่เกี่ยวข้อง F1, F2, หรือ F3. การจัดการข้อมูลสร้าง ODP สำหรับแต่ละไฟล์ให้สัมพันธ์กับหมายเลข call-level ที่เปิดไฟล์. คำสั่ง RCLRSC อาจถูกใช้เพื่อปิดไฟล์ขึ้นกับหมายเลข call-level เฉพาะที่ผ่านไปยังคำสั่งนั้น.

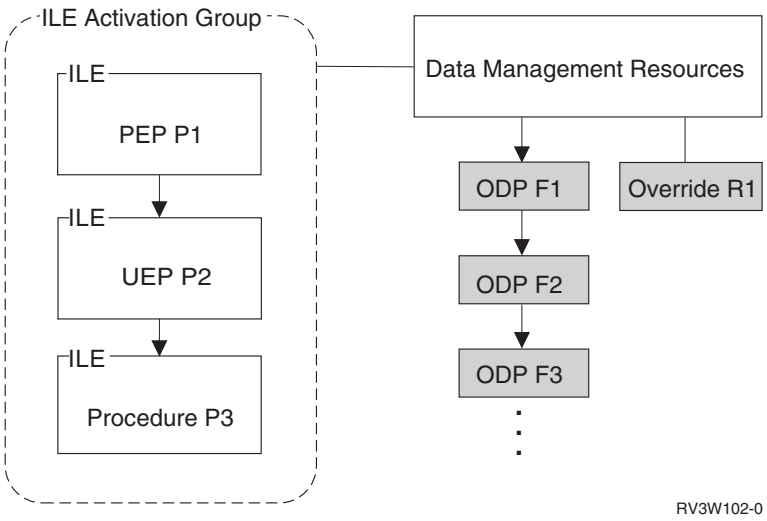


รูปที่ 27. แสดงการวางขอบเขตการ Call. ODP และการแทนที่(Overrides) อาจถูกวางขอบเขตการ call เอาไว้.

การแทนที่ที่ถูกวางขอบเขตในระดับ call ถูกลบออกเมื่อมีการรีเทิร์นของ call stack entry ที่เกี่ยวข้อง. การแทนที่อาจถูกแบ่งใช้โดย call stack entry ใดๆ ที่อยู่ใต้ระดับ call ที่สร้างการแทนที่นั้นๆ.

การวางขอบเขต Activation-Group-Level

การวางขอบเขตระดับ Activation group เกิดขึ้นเมื่อรีซอร์สการจัดการข้อมูลเชื่อมต่อกับ activation group ของโปรแกรมหรือเซอร์วิสโปรแกรมของ ILE ที่สร้างรีซอร์สนั้น. เมื่อ activation group ถูกลบ, การจัดการข้อมูลจะปิดรีซอร์สทุกตัวที่เกี่ยวข้องกับ activation group นั้นที่ยังเปิดค้างอยู่โดยโปรแกรมที่รันอยู่ใน activation group. รูปที่ 28 ในหน้า 55 แสดงตัวอย่างของการจำกัดขอบเขตระดับ activation group. ซึ่งเป็นระดับของการจำกัดขอบเขตที่เป็นดีฟอลต์ของรีซอร์สการจัดการข้อมูลส่วนใหญ่ ที่ถูกใช้โดยโพรซีเดอร์ ILE ที่ไม่ได้รันใน default activation group. จากรูปแสดงถึง ODP ของไฟล์ F1, F2, F3 และ override R1 ถูกจำกัดขอบเขตใน activation group.



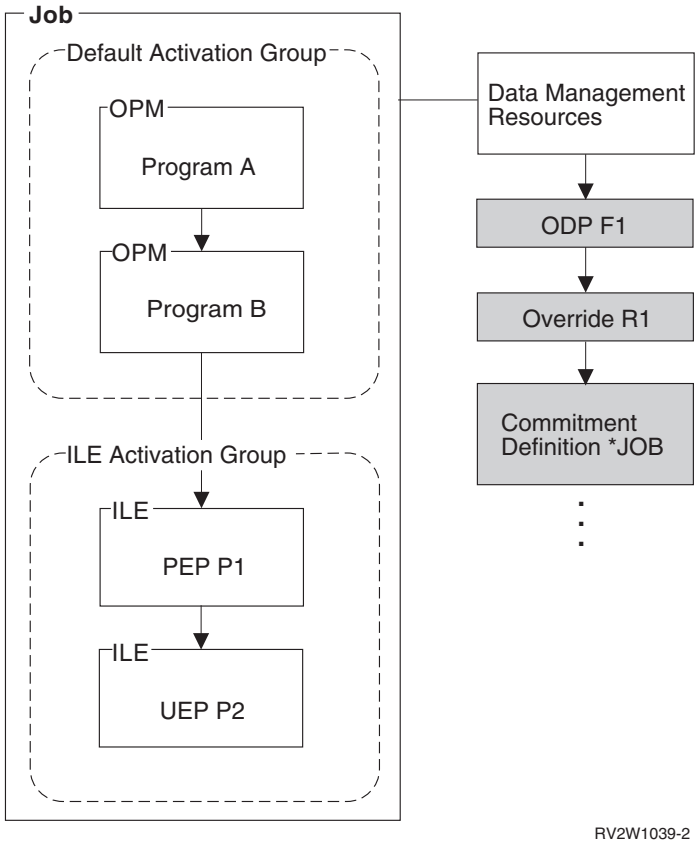
รูปที่ 28. Activation Group Level Scoping. ODP และ override อาจจะถูกจำกัดขอบเขตที่ activation group.

ความสามารถในการแบ่งใช้รีซอร์สการจัดการข้อมูลที่จำกัดขอบเขตเฉพาะใน activation group จะถูกจำกัดให้จัดสรรให้กับโปรแกรมที่ทำงานใน activation group เท่านั้น. ก่อให้เกิดการแยกกันและการปกป้องในระดับแอ็พพลิเคชัน. ตัวอย่างเช่น, สมมุติว่าไฟล์ F1 ในรูปถูกเปิดด้วยค่าพารามิเตอร์ SHARE (*YES). ไฟล์ F1 อาจถูกใช้โดยโพรซีเดอร์ ILE ใดๆ ที่ทำงานใน activation group เดียวกัน. กระบวนการเปิดไฟล์ F1 ใน activation group อื่นอาจจะทำให้เกิดการสร้าง ODP แห่งที่ 2 สำหรับไฟล์นั้น.

การวางขอบเขตระดับงาน

การวางขอบเขตระดับงานเกิดขึ้นเมื่อรีซอร์สการจัดการข้อมูลเชื่อมต่อกับงาน. การวางขอบเขตระดับงานมีทั้งในโปรแกรม OPM และโปรแกรม ILE. การวางขอบเขตระดับงานช่วยให้มีการแบ่งใช้รีซอร์สการจัดการข้อมูลระหว่างโปรแกรมที่ทำงานอยู่ใน activation group ต่างกลุ่มกันได้. เช่นเดียวกับการอธิบายในหัวข้อที่ผ่านมาคือ การจำกัดขอบเขตการแบ่งรีซอร์สให้กับ activation group จะจำกัดการแบ่งรีซอร์สไว้เฉพาะกับโปรแกรมที่ทำงานใน activation group นั้นเท่านั้น. การวางขอบเขตระดับงานการจำกัดขอบเขตระดับ job อนุญาตให้มีการแบ่งใช้รีซอร์สการจัดการข้อมูลระหว่างโปรแกรม ILE และโปรแกรม OPM ทั้งหมดที่รันในงานนั้น.

รูปที่ 29 ในหน้า 56 แสดงตัวอย่างของการวางขอบเขตระดับงาน. โปรแกรม A อาจเปิดไฟล์ F1, ซึ่งเป็นการระบุการวางขอบเขตระดับงาน. ODP ของไฟล์นั้นเชื่อมต่อกับงาน. ไฟล์จะไม่ถูกปิดโดยระบบถ้างานยังไม่สิ้นสุด. ถ้า ODP ถูกสร้างด้วยค่าพารามิเตอร์ SHARE (YES) โปรแกรม OPM หรือโพรซีเดอร์ ILE อื่นอาจจะแบ่งใช้ไฟล์นั้นได้.



รูปที่ 29. Job Level Scoping. ODP, override และ commitment definition อาจถูกจำกัดขอบเขตในระดับงาน.

การแทนที่ถูกวางขอบเขตในระดับงานมีอิทธิพลต่อกระบวนการเปิดไฟล์ทั้งหมดในงาน. ในตัวอย่างนี้, override R1 ที่ถูกสร้างด้วย โพรซีเดอร์ P2. การแทนที่ในระดับงาน ยังคงทำงานอยู่จนกว่าจะถูกลบหรืองานสิ้นสุดการทำงานลง. การแทนที่ระดับงานจะเป็นการแทนที่ที่มีลำดับความสำคัญสูงสุดเมื่อมีการแทนที่หลายระดับเกิดขึ้น. เนื่องจากการแทนที่ระดับ call ถูกรวมเข้าด้วยกัน เมื่อมีการแทนที่หลายตัวใน call stack.

ระดับในการวางขอบเขตของการจัดการข้อมูล อาจถูกกำหนดได้โดย การใช้พารามิเตอร์ในการวางขอบเขตในคำสั่ง override, คำสั่ง commitment control และผ่าน API. รายชื่อทั้งหมดของรีซอร์สการจัดการข้อมูลที่ใช้กฎเกณฑ์ในการวางขอบเขต ใน บทที่ 11, “การวางขอบเขตในการบริหารข้อมูล”, ในหน้า 149.

บทที่ 4. หน่วยเก็บข้อมูลแบบ Teraspace และ Single-level

เมื่อคุณสร้างโปรแกรม ILE คุณสามารถเลือกใช้หน่วยเก็บข้อมูลในเซิร์ฟเวอร์ iSeries iSeries ได้สองแบบคือ: แบบTeraspace และแบบ Single-level. ซึ่งในบทนี้เราจะมุ่งเน้นไปที่อ็อพชันใหม่นั้นคือ Teraspace เท่านั้น. อ็อพชันที่เป็นดีฟอลต์ของ ILE ก็คือ Single-level.

คุณลักษณะของ Teraspace

Teraspace เป็นหน่วยความจำชั่วคราวที่มีขนาดใหญ่ และสามารถเข้าถึงได้เฉพาะงานนั้นเท่านั้น. นั้นหน่วยความจำชนิดนี้มีแอดเดรสที่ต่อเนื่อง แต่ก็อาจเนื้อที่จัดสรรอาจไม่ต่อเนื่องกันก็ได้. หน่วยความจำแบบ Teraspace จะมีอายุการใช้งานแค่ช่วงระยะเวลาที่งานเริ่มต้น และงานสิ่นสุดเท่านั้น.

หน่วยความจำแบบ Teraspace ไม่ใช่อ็อบเจ็กต์ Space. ซึ่งหมายความว่ามันไม่ได้เป็นอ็อบเจ็กต์ของระบบ คุณจึงไม่สามารถอ้างถึงโดยใช้พอยเตอร์ของระบบได้. อย่างไรก็ตาม Teraspace สามารถอ้างถึงแบบแอดเดรสได้ด้วยพอยเตอร์ Space.

ตารางด้านล่างนี้แสดงการเปรียบเทียบระหว่างหน่วยเก็บข้อมูลแบบ Single-level กับแบบ Teraspace.

ตารางที่ 2. การเปรียบเทียบระหว่างหน่วยเก็บข้อมูลแบบ Single-level กับแบบ Teraspace

แอ็ตทริบิวต์	Teraspace	Single-level
ขอบเขตใช้งาน	โลคอล: เข้าถึงได้เฉพาะงานที่เป็นเจ้าของเท่านั้น.	โกลบอล: เข้าถึงได้จากงานใดก็ได้โดยการใช้พอยเตอร์.
ขนาด	รวมทั้งหมด 1 TB	ยูนิตละ 16 MB หลายยูนิต.
สนับสนุนการแม็พหน่วยความจำ?	ใช่	ไม่
พอยเตอร์ขนาด 8 ไบต์?	ใช่	ไม่
สนับสนุนการแบ่งใช้ระหว่างงาน?	ต้องใช้ API ที่เกี่ยวกับการแบ่งใช้หน่วยความจำเท่านั้น (เช่น shmat หรือ mmap).	สามารถทำได้โดยส่งค่าพอยเตอร์ไปยังงานอื่น หรือใช้ API ที่เกี่ยวกับการแบ่งใช้หน่วยความจำ.

การทำให้โปรแกรมของคุณสามารถใช้ Teraspace ได้

โปรแกรม ILE ใช้โมเดลหน่วยเก็บข้อมูลแบบ Single-level เป็นค่าดีฟอล์ต. ดังนั้นหากคุณต้องการให้โปรแกรมของคุณสามารถใช้หน่วยเก็บข้อมูลแบบ Teraspace ได้โปรแกรมนั้นต้องทำการ*เปิดใช้ Teraspace*. โปรแกรมที่เปิดใช้ Teraspace จึงจะสามารถประมวลผลแอดเดรสแบบ Teraspace ในหลายๆ รูปแบบได้ เช่น:

- เมื่อมันถูกรีเทิร์นจากโปรแกรมที่ร้องขอเพื่อจัดสรรหน่วยความจำ Heap แบบ Teraspace
- เมื่อมันถูกรีเทิร์นจากโปรแกรมที่ร้องขอเพื่อจัดสรรหน่วยความจำที่ใช้ร่วมกันแบบ Teraspace

- เมื่อมันถูกส่งจากโปรแกรมอื่น

คอมไพเลอร์ต่อไปนี้สามารถสร้างโค้ดที่ใช้ Teraspace ได้:

- ILE C (เลือก TERASPACE(*YES) เมื่อคุณสร้างโมดูลหรือโปรแกรม)
- ILE C++ (เลือก TERASPACE(*YES) เมื่อคุณสร้างโมดูลหรือโปรแกรม)
- ILE RPG (การเปิดใช้ Teraspace ถูกใช้เป็นค่าดีฟอล์ตตั้งแต่ใน V4R4)
- ILE COBOL (การเปิดใช้ Teraspace ถูกใช้เป็นค่าดีฟอล์ตตั้งแต่ใน V4R4)
- ILE CL (การเปิดใช้ Teraspace ถูกใช้เป็นค่าดีฟอล์ตตั้งแต่ใน V5R1)

คอมไพเลอร์ ILE C และ C++ ได้เตรียมคำสั่ง TERASPACE (*YES *TSIFC) ที่ช่วยทำให้คุณสามารถใช้หน่วยความจำแบบ Teraspace ได้โดยไม่ต้องเปลี่ยนแปลงซอร์สโค้ด. เช่น `malloc()` จะถูกแม็พเข้ากับฟังก์ชัน `_C_TS_malloc()` แทน.

โปรดดู WebSphere Development Studio ILE C/C++ Programmer's Guide  สำหรับรายละเอียดเกี่ยวกับคอมไพลเลอร์อ็อพชันเหล่านี้.

คุณสามารถทำให้โปรแกรม OPM ใช้งาน Teraspace ได้โดยใช้คำสั่ง CHGPGM.

การเลือกโมเดลหน่วยความจำสำหรับโปรแกรม

คุณสามารถใช้คุณลักษณะของ Teraspace ได้มากกว่านี้ โดยการสร้างโมดูล และโปรแกรมของคุณด้วย *โมเดลหน่วยความจำแบบ Teraspace*. ซึ่งโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Teraspace จะสามารถใช้ Teraspace กับหน่วยความจำแบบไดนามิก, สแตติก และค่าคงที่ได้. เมื่อคุณเลือกโมเดลหน่วยความจำแบบ Teraspace คุณก็สามารถใช้เนื้อที่หน่วยความจำขนาดใหญ่นี้ได้. โปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Teraspace สามารถใช้พอยเตอร์ขนาด 8 ไบต์ในการอ้างถึงเนื้อที่ในหน่วยความจำได้. ดูในหัวข้อ "การใช้โมเดลหน่วยความจำแบบ Teraspace" ในหน้า 65 สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับโมเดลหน่วยความจำแบบ Teraspace.

คุณสามารถเลือกโมเดลหน่วยความจำให้กับโมดูล และโปรแกรมของคุณได้สองแบบคือ แบบ Single-level (*SNGLVL) และแบบ Teraspace (*TERASPACE). คุณยังสามารถเลือกกำหนดให้เซอร์วิสโปรแกรมรับช่วงโมเดลหน่วยความจำของ Activation Group ที่รันอยู่ก็ได้. ในหัวข้อนี้เราจะอธิบายถึงโมเดลหน่วยความจำแบบ Teraspace.

การกำหนดโมเดลหน่วยความจำแบบ Teraspace

ในการเลือกโมเดลหน่วยความจำแบบ Teraspace คุณต้องระบุอ็อพชันดังต่อไปนี้เมื่อคุณทำการคอมไพล์โค้ดของคุณ:

1. ตรวจดูให้แน่ใจว่าโมดูลของคุณเปิดใช้ Teraspace แล้ว. ให้ระบุ *YES ในพารามิเตอร์ TERASPACE เมื่อคุณสร้างโมดูลของคุณ.

คุณสามารถคำสั่ง DSPMOD, DSPPGM และ DSPSRVPGM เพื่อแสดงแอ็ตทริบิวต์ Teraspace ของโมดูล, โปรแกรม และเซอร์วิสโปรแกรมตามลำดับ. คุณสามารถดูแอ็ตทริบิวต์ Teraspace จาก DETAIL(*MODULE) เมื่อคุณใช้อ็อพชัน 5 เพื่อแสดงรายละเอียดในแต่ละโมดูล.

2. เลือก *TERASPACE หรือ *INHERIT ในพารามิเตอร์ Storage model (STGMDL) ของคำสั่งสร้างโมดูลสำหรับภาษาโปรแกรมของ ILE.
3. กำหนด *TERASPACE ในพารามิเตอร์ STGMDL ของคำสั่ง CRTPGM หรือ CRTSRVPGM. ซึ่งคุณต้องเลือกคำสั่งที่สอดคล้องกับโมเดลหน่วยความจำของโมดูลที่คุณรวมโปรแกรม. ดูในหัวข้อ “กฎในการรวมโมดูลเข้าด้วยกัน” ในหน้า 61 สำหรับรายละเอียดเพิ่มเติม.

   คุณสามารถกำหนด *TERASPACE ในพารามิเตอร์ STGMDL ของคำสั่ง CRTBNDC และ CRTBNDCPP ซึ่งสร้างการรวมโปรแกรมในขั้นเดียวที่มีโมดูลเพียงอันเดียว.

   ในคำสั่ง CRTSRVPGM คุณสามารถระบุ *INHERIT ในพารามิเตอร์ STGMDL ก็ได้. ซึ่งจะทำให้เซอร์วิสโปรแกรมถูกสร้างในรูปแบบที่มันสามารถใช้ได้ทั้ง Single-level หรือ Teraspace ขึ้นอยู่กับชนิดของหน่วยความจำที่ใช้ใน Activation Group ซึ่งเซอร์วิสโปรแกรมนั้นเรียกใช้งาน.

   การใช้แอ็ตทริบิวต์ *INHERIT ช่วยให้เกิดความยืดหยุ่นมาก แต่คุณยังต้องระบุ *CALLER ในพารามิเตอร์ ACTGRP ด้วย. ซึ่งในกรณีนี้คุณควรจำไว้เสมอว่าเซอร์วิสโปรแกรมของคุณสามารถทำงานได้ทั้งในแบบ Single-level และ Teraspace คุณต้องเขียนโค้ดของคุณให้รองรับการทำงานในทั้งสองรูปแบบ. ตัวอย่างเช่น ขนาดของตัวแปรแบบสแตติกทั้งหมดต้องไม่เกินเกินค่าจำกัดที่กำหนดไว้ของหน่วยความจำแบบ Single-level.

ตารางที่ 3. โมเดลหน่วยความจำสำหรับโปรแกรมแบบต่างๆ ที่อนุญาตให้ใช้งาน.

โมเดลหน่วยความจำสำหรับโปรแกรม	**ประเภทของโปรแกรม**		
	OPM *PGM	**ILE *PGM**	**ILE *SRVPGM**
*TERASPACE	ไม่	ใช่	ใช่
*INHERIT	ไม่	ไม่	ใช่ แต่เฉพาะ ACTGRP (*CALLER) เท่านั้น)
*SNGLVL	ใช่	ใช่	ใช่

การเลือก Activation Group ที่เข้ากันได้

Activation Group จะแสดงถึงโมเดลหน่วยความจำของโปรแกรมหลักที่สร้าง Activation Group นั้นขึ้นมา. โมเดลหน่วยความจำเป็นตัวกำหนดประเภทหน่วยความจำแบบอัตโนมัติ, สแตติก และค่าคงที่ที่โปรแกรมใช้งาน.

โปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Single-level จะมีหน่วยความจำแบบอัตโนมัติ, สแตติก และค่าคงที่เป็นแบบ Single-level. ตามค่าดีฟอล์ตแล้ว โปรแกรมประเภทนี้ยังกำหนดให้หน่วยความจำแบบ Heap เป็นแบบ Single-level ด้วย.

ส่วนโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Teraspace จะมีหน่วยความจำแบบอัตโนมัติ, สแตติก และค่าคงที่เป็นแบบ Teraspace. ตามค่าดีฟอล์ตแล้ว โปรแกรมประเภทนี้ยังกำหนดให้หน่วยความจำแบบ Heap เป็นแบบ Teraspace ด้วย.

สำหรับโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Teraspace จะไม่สามารถใช้งานใน Activation Group ที่โปรแกรมหลักใช้โมเดลหน่วยความจำแบบ Single-level ได้. และโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Single-level ก็จะไม่สามารถใช้งานใน Activation Group ที่โปรแกรมหลักใช้โมเดลหน่วยความจำแบบ Teraspace เช่นกัน.

ตารางต่อไปนี้จะสรุปความสัมพันธ์ของโมเดลหน่วยความจำกับ Activation Group แบบต่างๆ.

ตารางที่ 4. ความสัมพันธ์ของโมเดลหน่วยความจำกับ Activation Group

โมเดลหน่วยความจำสำหรับโปรแกรม	**แอ็ตทริบิวต์ของ Activation Group**			
	***CALLER**	***DFTACTGRP**	***NEW**	**Named**
*TERASPACE	ใช้ได้. ตรวจดูให้แน่ใจว่าโปรแกรมไคลเอนต์ถูกสร้างโดยใช้โมเดลหน่วยความจำแบบ Teraspace.	ไม่อนุญาต. ค่าดีฟอล์ตของ Activation Group เป็นได้เฉพาะ *SNGLVL เท่านั้น.	ใช่	ใช่
*INHERIT	ใช่	ไม่อนุญาต.	ไม่อนุญาต.	ไม่อนุญาต.
*SNGLVL	ใช่	ใช่	ใช่	ใช่

เมื่อคุณเลือก Activation Group ที่โปรแกรม และเซอร์วิสโปรแกรมของคุณจะใช้งานได้แล้ว ก็ให้พิจารณาคำแนะนำดังต่อไปนี้:

- ถ้าเซอร์วิสโปรแกรมของคุณระบุ STGMDL(*INHERIT) เอาไว้ คุณก็ต้องใช้ ACTGRP (*CALLER) เท่านั้น.
- ถ้าโปรแกรมของคุณระบุ STGMDL(*TERASPACE) เอาไว้:
  - คุณก็ควรใช้ ACTGRP(*NEW) หรือ Named Activation Group.
  - หรือใช้ ACTGRP(*CALLER) ก็ได้หากคุณแน่ใจว่าทุกโปรแกรมทั้งหมดที่โปรแกรมของคุณเรียกใช้โมเดลหน่วยความจำแบบ Teraspace.

วิธีการที่โมเดลหน่วยความจำทำงานร่วมกัน

ความสอดคล้องในการใช้งานเป็นสิ่งที่จำเป็นสำหรับโมดูลและโปรแกรมที่ใช้โมเดลหน่วยความจำ. ในหัวข้อต่อไปนี้จะเป็นกฎที่ทำให้โปรแกรมต่างๆ ทำงานร่วมกันได้อย่างถูกต้อง.

- “กฎในการรวมโมดูลเข้าด้วยกัน” ในหน้า 61
- “กฎในการรวมเซอร์วิสโปรแกรมเข้าด้วยกัน” ในหน้า 61
- “กฎในการเรียกใช้งานโปรแกรมและเซอร์วิสโปรแกรม” ในหน้า 61
- “กฎในการเรียกโปรแกรมและโพรซีเดอร์” ในหน้า 62

กฎในการรวมโมดูลเข้าด้วยกัน

ตารางต่อไปนี้จะแสดงกฎในการรวมโมดูลเข้าด้วยกัน:

กฎในการรวม: รวมโมดูล M เข้ากับโปรแกรมที่ใช้โมเดลหน่วยความจำ.		โมเดลหน่วยความจำของโปรแกรมที่ถูกสร้างขึ้น.		
		Teraspace	**Inherit (เฉพาะเซอร์วิสโปรแกรมเท่านั้น)**	**Single-level**
M	**Teraspace**	Teraspace	ไม่ถูกต้อง	ไม่ถูกต้อง
	Inherit	Teraspace	Inherit	Single-level
	Single-level	ไม่ถูกต้อง	ไม่ถูกต้อง	Single-level

กฎในการรวมเซอร์วิสโปรแกรมเข้าด้วยกัน

ตารางต่อไปนี้จะแสดงกฎในการรวมโปรแกรมเข้ากับเซอร์วิสโปรแกรม.

กฎในการรวมเซอร์วิสโปรแกรม: โปรแกรมหรือเซอร์วิสโปรแกรมที่เรียกสามารถรวมกับเซอร์วิสโปรแกรมเป้าหมายได้หรือไม่?		โมเดลหน่วยความจำของเซอร์วิสโปรแกรมเป้าหมาย		
		Teraspace	**Inherit**	**Single-level**
โมเดลหน่วยความจำของโปรแกรมหรือเซอร์วิสโปรแกรมที่เรียก	**Teraspace**	ใช่	ใช่	ใช้ได้[2]
	Inherit[1]	ใช้ได้[3]	ใช่	ใช้ได้[3]
	Single-level	ใช้ได้[2]	ใช่	ใช่

หมายเหตุ:

1. เฉพาะเซอร์วิสโปรแกรมเท่านั้นที่สามารถกำหนดให้ใช้โมเดลหน่วยความจำแบบ Inherit.
2. เซอร์วิสโปรแกรมเป้าหมายต้องรันใน Activation Group ที่แยกจากกัน. ตัวอย่างเช่น เซอร์วิสโปรแกรมเป้าหมายไม่สามารถใช้แอ็ตทริบิวต์ ACTGRP(*CALLER) ได้. เนื่องจากเป็นไปไม่ได้ที่จะใช้โมเดลหน่วยความจำหลายๆ แบบใน Activation Group เดียวกัน.
3. การรวมเซอร์วิสโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Inherit เข้ากับเซอร์วิสโปรแกรมที่ใช้ Single-level หรือ Teraspace เป็นสิ่งที่สามารถทำได้ แต่ผลการทำงานอาจไม่สามารถทราบได้จนกว่าจะใช้งานจริง. ถ้าเซอร์วิสโปรแกรมเป้าหมายใช้แอ็ตทริบิวต์ ACTGRP(*CALLER) แล้วเซอร์วิสโปรแกรมที่เรียก (ซึ่งกำหนดให้ใช้โมเดลหน่วยความจำแบบ Inherit) ต้องถูกเรียกใช้ใน Activation Group ที่เข้ากันได้กับเซอร์วิสโปรแกรมเป้าหมาย. เซอร์วิสโปรแกรมเป้าหมายจะไม่สามารถระบุ ACTGRP(*CALLER) หรือระบุชื่อของ Activation Group เหมือนโปรแกรม หรือเซอร์วิสโปรแกรมที่เรียกได้.

กฎในการเรียกใช้งานโปรแกรมและเซอร์วิสโปรแกรม

เซอร์วิสโปรแกรมแบบ Teraspace ที่ระบุให้ใช้โมเดลหน่วยความจำแบบ Inherit สามารถทำงานได้ในActivation Group ซึ่งรันโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Single-level หรือ Teraspace

ได้. ไม่อย่างนั้นแล้วโมเดลหน่วยความจำของเซอร์วิสโปรแกรมต้องตรงกันกับโมเดลหน่วยความจำของโปรแกรมอื่นๆ ที่รันใน Activation Group นั้น.

กฎในการเรียกโปรแกรมและโพรซีเดอร์

โปรแกรมและเซอร์วิสโปรแกรมที่ใช้โมเดลหน่วยความจำที่แตกต่างกันสามารถทำงานร่วมกันได้. โดยโปรแกรมเหล่านี้สามารถทำงาน และใช้ข้อมูลร่วมกันได้ตราบเท่าที่โปรแกรมนั้นปฏิบัติตามกฎและคำแนะนำต่างๆ ที่อธิบายไว้ในบทนี้.

โค้ดที่ไม่ได้เปิดใช้งาน Teraspace จะไม่สามารถประมวลผลแอดเดรสแบบ Teraspace ได้. ซึ่งการทำเช่นนี้จะทำให้เกิดข้อความ Exception โดยเฉพาะ MCH0607 (การใช้เนื้อที่ที่ไม่ได้สนับสนุน).

การแปลงเซอร์วิสโปรแกรมของคุณให้ใช้โมเดลหน่วยความจำแบบ Inherit

คุณสามารถแปลงเซอร์วิสโปรแกรมของคุณให้ใช้โมเดลหน่วยความจำแบบ Inherit (ระบุ *INHERIT ในพารามิเตอร์ STGMDL) ได้ คุณยังสามารถทำให้โปรแกรมนั้นใช้สภาพแวดล้อมหน่วยความจำแบบ Teraspace หรือ Single-level ก็ได้. โดยทำตามขั้นตอนในการแปลงเซอร์วิสโปรแกรมที่คุณมีสร้างไว้แล้วให้ใช้โมเดลหน่วยความจำแบบ Teraspace ดังนี้:

1. สร้างโมดูลทั้งหมดด้วยโมเดลหน่วยความจำแบบ Inherit. คุณไม่สามารถสร้างเซอร์วิสโปรแกรมด้วยโมเดลหน่วยความจำแบบ Inherit ได้หากมีโมดูลตัวใดตัวหนึ่งถูกสร้างด้วยโมเดลหน่วยความจำแบบ Single-level หรือ Teraspace.
2. ตรวจดูให้แน่ใจว่าโค้ดของคุณต้องมีการจัดการพอยเตอร์อย่างมีประสิทธิภาพในการอ้างถึงและจากหน่วยความจำแบบ Teraspace และ Single-level. ดูในหัวข้อ “ข้อควรปฏิบัติในการใช้ Teraspace” ในหน้า 65 สำหรับรายละเอียดเพิ่มเติม.
3. สร้างเซอร์วิสโปรแกรมของคุณด้วยโมเดลหน่วยความจำแบบ Inherit. แล้วระบุ *CALLER ในพารามิเตอร์ ACTGRP เหมือนกัน.

การเปลี่ยนแปลงและอัพเดตโปรแกรมของคุณ: ข้อพิจารณาสำหรับ Teraspace

คุณสามารถเปลี่ยนแปลง หรืออัพเดตโปรแกรมของคุณให้ใช้งาน Teraspace ได้ภายใต้ข้อกำหนดและเงื่อนไขที่กำหนดไว้. ซึ่งเงื่อนไขต่างๆ จะจำเพาะสำหรับโมเดลหน่วยความจำของโมดูลที่ใช้ในการอัพเดตโปรแกรมของคุณ.

การแก้ไขโปรแกรมของคุณ:

คุณสามารถใช้คำสั่ง CHGPGM และ CHGSRVPGM ในการเปลี่ยนโปรแกรมที่ไม่ได้ใช้ Teraspace ให้เป็นโปรแกรมที่ใช้ Teraspace ได้. คุณสามารถใช้คำสั่งนี้ได้กับโปรแกรม ILE พร้อมกับโมดูลทั้งหมดที่รวมไว้, รีลีสเป้าหมาย V4R4M0 หรือสูงกว่า. คุณยังสามารถใช้คำสั่งนี้ได้กับโปรแกรม OPM ซึ่งมีรีลีสเป้าหมาย V4R4M0 หรือสูงว่าเช่นกัน.

การอัพเดตโปรแกรมของคุณ:

คุณสามรถเพิ่มหรือแทนที่โมดูลที่อยู่ในโปรแกรมตราบที่โมดูลเหล่านั้นใช้โมเดลหน่วยความจำแบบเดียวกัน อย่างไรก็ตาม คุณไม่สามารถใช้คำสั่งอัพเดตในการเปลี่ยนโมเดลของหน่วยความจำของโมดูลที่รวมหรือโปรแกรม.

การใช้ประโยชน์จากพอยเตอร์ขนาด 8 ไบต์ในโค้ด C และ C++ ของคุณ

พอยเตอร์แบบ 8 ไบต์สามารถใช้ได้เฉพาะ Teraspace เท่านั้น. ส่วนพอยเตอร์ของโพรซีเดอร์แบบ 8 ไบต์สามารถอ้างถึงโพรซีเดอร์ที่ใช้งานอยู่ผ่าน Teraspace. โดย Type ของพอยเตอร์แบบ 8 ไบต์คือ Space และพอยเตอร์ของโพรซีเดอร์.

แต่สำหรับพอยเตอร์แบบ 16 ไบต์มี Type อยู่หลายแบบ. แบบ ในตารางต่อไปนี้แสดงการเปรียบเทียบคุณสมบัติของพอยเตอร์แบบ 8 ไบต์และ 16 ไบต์.

ตารางที่ 5. การเปรียบเทียบพอยเตอร์

คุณสมบัติ	พอยเตอร์แบบ 8 ไบต์	พอยเตอร์แบบ 16 ไบต์
ขนาด (หน่วยความจำที่ต้องการ)	8 ไบต์	16 ไบต์
แท็ก	ไม่	ใช่
การวางแนว	ยอมให้วางแนวระดับไบต์ (ให้อยู่ในรูปบีบอัด). การวางแนวตามปกติ (8 ไบต์) เหมาะสมกว่าในด้านประสิทธิภาพ.	16 ไบต์เสมอ.
Atomicity	Atomic โหลดและเก็บเมื่อมีการวางแนวพอยเตอร์แบบ 8 ไบต์. ไม่รวมถึงการก็อปปี.	Atomic โหลดและเก็บ. Atomic ทำสำเนาเมื่อเป็นส่วนหนึ่งของการรวม.
ช่วงของแอดเดรส	Teraspace storage	Teraspace storage + single-level storage
ค่าในพอยเตอร์	ค่า 64 บิตที่แสดงอ็อฟเซ็ตในTeraspace. ซึ่งจะไม่มี Effective Address.	พอยเตอร์แบบ 16 ไบต์และ Effective Address 64 บิต.
การอ้างอิงเฉพาะที่	ทำงานเฉพาะการอ้างอิงแบบโลคัล. (พอยเตอร์แบบ 8 ไบต์อ้างอิงได้เฉพาะงานของ Teraspace ในหน่วยความจำเท่านั้น).	ทำงานได้ทั้งโลคัลหรืออ้างอิงแบบ Single-level ก็ได้. (พอยเตอร์แบบ 16 ไบต์สามารถอ้างถึงหน่วยความจำที่เป็นของงานอื่นได้).
การทำงานที่อนุญาต	การทำงานของพอยเตอร์สามารถใช้ได้เฉพาะ Space และพอยเตอร์ของโพรซีเดอร์ และการทำงานที่ไม่ใช่พอยเตอร์, การทำงานด้านลอจิคัลและคณิตศาสตร์ในข้อมูลแบบไบนารีสามารถใช้ได้โดยไม่ทำให้พอยเตอร์โมฆะ.	เฉพาะการทำงานของพอยเตอร์.
การอ้างอิงข้อมูลอย่างรวดเร็ว	ไม่	ใช่

ตารางที่ 5. การเปรียบเทียบพอยเตอร์ (ต่อ)

คุณสมบัติ	พอยเตอร์แบบ 8 ไบต์	พอยเตอร์แบบ 16 ไบต์
โหลด, เก็บข้อมูล และคณิตศาสตร์ของพอยเตอร์ Space อย่างรวดเร็ว	ใช่ รวมค่าโอเวอร์เฮด EAO ที่หลีกเลี่ยง.	ไม่
ขนาดของค่าไบนารีที่สงวนไว้เมื่อมีการแปลงพอยเตอร์	8 ไบต์	4 ไบต์
สามารถยอมรับพารามิเตอร์โดยโพรซีเดอร์ที่เป็นตัวจัดการ Exception หรือยกเลิกตัวจัดการ.	ไม่	ใช่

พอยเตอร์ที่สนับสนุนในคอมไพเลอร์ C และ C++

ในการใช้ประโยชน์จากพอยเตอร์แบบ 8 ไบต์อย่างเต็มที่ เมื่อคุณคอมไพล์โค้ดของคุณด้วยคอมไพเลอร์ C หรือ C++ ของIBM ให้ระบุ STGMDL(*TERASPACE) และ DTAMDL(*LLP64).

คอมไพเลอร์ C และ C++ ยังสนับสนุนการทำงานของพอยเตอร์ดังนี้:

- ไวยากรณ์ในการประกาศพอยเตอร์แบบ 8 ไบต์หรือ 16 ไบต์:
  - ประกาศพอยเตอร์แบบ 8 ไบต์เป็น `char * __ptr64`
  - ประกาศพอยเตอร์แบบ 16ไบต์เป็น `char * __ptr128`
- อ็อพชันของคอมไพเลอร์ และ Pragma สำหรับระบุ *แบบจำลองข้อมูล* ที่ใช้เฉพาะสภาวะแวดล้อมการโปรแกรมด้วยภาษา C และ C++. แบบจำลองข้อมูลจะส่งผลกระทบต่อค่าดีฟอล์ตของขนาดของพอยเตอร์ในกรณีที่ไม่ได้กำหนด Qualifier เอาไว้. ซึ่งคุณสามารถเลือกตัวเลือกสำหรับแบบจำลองข้อมูลได้ 2 แบบคือ:
  - `P128`, หรือที่รู้จักในชื่อ 4-4-16[1]
  - `LLP64` หรือที่รู้จักในชื่อ 4-4-8[2]

การแปลงพอยเตอร์

คอมไพเลอร์ C และ C++ ของไอบีเอ็ม จะทำการแปลง `__ptr128` ให้เป็น `__ptr64` และในทางกลับกันหากจำเป็นขึ้นอยู่กับการประกาศฟังก์ชัน และตัวแปร. โปรดสังเกตข้อควรระวังต่อไปนี้ให้ดีด้วย:

- `_ptr128` ที่อ้างถึงหน่วยความจำแบบ Single-level จะถูกแปลงเป็นค่า `__ptr64`
- โค้ดที่ไม่ได้เปิดใช้งาน Teraspace จะไม่สามารถเข้าถึง Teraspace ได้.
- อินเตอร์เฟสระหว่างพารามิเตอร์พอยเตอร์กับพอยเตอร์ต้องการจัดการพิเศษ.

คอมไพเลอร์จะแทรกการแปลงพอยเตอร์ที่เหมาะสมกับขนาดของพอยเตอร์ให้โดยอัตโนมัติ. ตัวอย่างเช่น การแปลงพอยเตอร์จะถูกแทรกเข้าไปเมื่ออาร์กิวเมนต์ของพอยเตอร์ไปยังฟังก์ชันไม่เหมาะตรงกับกับพอยเตอร์ในต้นแบบ (Prototype) สำหรับฟังก์ชัน. หรือหากมีการเปรียบเทียบ

1. โดย 4-4-16 คือ sizeof(int) – sizeof(long) – sizeof(pointer)
2. โดย 4-4-8 คือ sizeof(int) – sizeof(long) – sizeof(pointer)

ขนาดพอยเตอร์ที่แตกต่างกัน คอมไพเลอร์จะทำการแปลงพอยเตอร์แบบ 8 ไบต์ไปเป็น 16 ไบต์. คอมไพเลอร์ยังยอมให้คุณสั่งแปลงค่าได้. หากคุณจำเป็นต้องแปลงค่าจริงๆ ก็ขอให้ระวังตามข้อสังเกตดังต่อไปนี้:

- การแปลงพอยเตอร์จาก 16 ไบต์เป็น 8 ไบต์จะทำได้เฉพาะในกรณีที่พอยเตอร์แบบ 16 ไบต์มีค่าแอดเดรสแบบ Teraspace หรือค่า Null เท่านั้น. ไม่อย่างนั้นข้อความ Exceptoin MCH0609 จะปรากฏขึ้น หรือได้รับค่าอ็อฟเซ็ต Teraspace คืนกลับมา.
- พอยเตอร์ 16 ไบต์ไม่สามารถแปลง Type จากชนิดหนึ่งไปเป็นอีกชนิดหนึ่ง แต่พอยเตอร์ OPEN แบบ 16 ไบต์สามารถเก็บค่าพอยเตอร์ชนิดใดก็ได้. เนื่องจากไม่มีพอยเตอร์ OPEN แบบ 8 ไบต์ให้ใช้ แต่พอยเตอร์แบบ 8 ไบต์สามารถแปลงเป็นพอยเตอร์ Space และพอยเตอร์โพรซีเดอร์ได้. ดังนั้นการแปลงพอยเตอร์แบบ 8 ไบต์จึงเป็นเพียงการเปลี่ยนชนิดของพอยเตอร์เท่านั้น แต่มันจะไม่ยอมให้พอยเตอร์ Space ถูกใช้เป็นพอยเตอร์โพรซีเดอร์จนกว่าพอยเตอร์ Space จะถูกกำหนดให้อ้างถึงโพรซีเดอร์เท่านั้น.

เมื่อคุณสั่งให้แปลงพอยเตอร์เป็นค่าไบนารี อย่าลืมว่าคุณสมบัติของพอยเตอร์แบบ 8 ไบต์และ 16 ไบต์แตกต่างกัน. พอยเตอร์แบบ 8 ไบต์สามารถเก็บค่าไบนารีได้ครบทั้ง 8 ไบต์ ในขณะที่พอยเตอร์แบบ 16 ไบต์สามารถเก็บค่าไบนารีได้เพียง 4 ไบต์เท่านั้น. ในขณะที่เก็บค่าไบนารีอยู่นั้น การดำเนินงานที่กำหนดไว้เพียงอย่างเดียวของพอยเตอร์ก็คือ การแปลงค่าให้เป็นไบนารีเท่านั้น. ส่วนการดำเนินงานอื่นๆ ไม่ได้กำหนดไว้ซึ่งรวมทั้งการใช้เป็นตัวชี้, การแปลงขนาดและการเปรียบเทียบพอยเตอร์. ตัวอย่างเช่น ถ้ากำหนดค่าจำนวนเต็มให้กับพอยเตอร์แบบ 8 ไบต์และ 16 ไบต์เหมือนกัน จากนั้นก็แปลงกับพอยเตอร์แบบ 8 ไบต์ให้เป็น 16 ไบต์แล้วจึงทำการเปรียบเทียบค่าของพอยเตอร์ทั้งสอง ผลของการเปรียบเทียบจะไม่มีนิยาม รวมทั้งจะไม่แสดงผลลัพธ์ว่าพอยเตอร์ทั้งสองเท่ากัน.

การเปรียบเทียบพอยเตอร์ที่มีขนาดแตกต่างกันจะนิยามไว้เฉพาะเมื่อพอยเตอร์แบบ 8 ไบต์และพอยเตอร์แบบ 16 ไบต์เก็บค่าแอดเดรสของ Teraspace เท่านั้น (ซึ่งพอยเตอร์แบบ 8 ไบต์ไม่ได้เก็บค่าไบนารีไว้). พอยเตอร์แบบ 8 ไบต์จะถูกแปลงให้เป็นพอยเตอร์แบบ 16 ไบต์ แล้วจึงทำการเปรียบเทียบพอยเตอร์แบบ 16 ไบต์ทั้งสอง. ส่วนในกรณีอื่นนอกจากนี้ไม่มีการนิยามไว้. ตัวอย่างเช่น ถ้าพอยเตอร์แบบ 16 ไบต์ถูกแปลงให้เป็นพอยเตอร์แบบ 8 ไบต์ แล้วนำมาทำการเปรียบเทียบกับพอยเตอร์แบบ 8 ไบต์ ผลลัพธ์ที่ได้จะไม่มีนิยาม.

การใช้โมเดลหน่วยความจำแบบ Teraspace

ตามแนวคิดของสภาพแวดล้อมแบบ Teraspace โมดูล, โปรแกรม และเซอร์วิสโปรแกรมทั้งหมดของคุณน่าจะใช้โมเดลหน่วยความจำแบบ Teraspace. แต่ในทางปฏิบัติแล้ว คุณจำเป็นต้องจัดการกับสภาพแวดล้อมที่โมดูล, โปรแกรม และเซอร์วิสโปรแกรมใช้โมเดลหน่วยความจำทั้งสองแบบ (Single-level และ Teraspace).

ในหัวข้อนี้จะอธิบายถึงวิธีการที่คุณสามารถประยุกต์ใช้แนวคิดของสภาพแวดล้อมแบบ Teraspace. และยังอธิบายถึงวิธีการลดปัญหาที่เกิดจากการใช้โปรแกรมที่ใช้หน่วยความจำแบบ Single-level และ Teraspace.

ข้อควรปฏิบัติในการใช้ Teraspace

- *ใช้โมเดลหน่วยความจำแบบ Teraspace เฉพาะโมดูลเท่านั้น*

สร้างโมดูลของคุณโดยใช้โมเดลหน่วยความจำแบบ Teraspace หรือ Inherit. Inherit โมดูลที่ใช้หน่วยความจำแบบ Single-level ไม่เหมาะกับสภาพแวดล้อมแบบ Teraspace เนื่องจากคุณจะไม่สามารถรวมข้อมูลเหล่านี้เข้าในโปรแกรมของคุณ. แต่ถ้าคุณจำเป็นต้องใช้งานในลักษณะนี้จริงๆ (เช่น หากคุณไม่สามารถเข้าถึงซอร์สโค้ดของโมดูลนั้นได้) ดูในสถานการณ์ที่ 9 ของหัวข้อ "เคล็ดลับในการใช้ Teraspace" ในหน้า 69.

- *ทำการรวมเฉพาะเซอร์วิสโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Teraspace หรือ Inherit*

  โปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Teraspace ของคุณสามารถรวมได้กับเซอร์วิสโปรแกรมเกือบทุกแบบ. อย่างไรก็ตาม มันก็ควรรวมเข้ากับโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Inherit หรือ Teraspace. ถ้าคุณควบคุมเซอร์วิสโปรแกรม คุณก็ควรสร้างเซอร์วิสโปรแกรมทั้งหมดของคุณในลักษณะเดียวกันกับโปรแกรมที่คุณต้องการรวมเข้าด้วยกัน. โดยทั่วไป เซอร์วิสโปรแกรมของไอบีเอ็ม ก็ถูกสร้างในลักษณะนี้. คุณเองก็ควรทำในลักษณะเดียวกัน โดยเฉพาะอย่างยิ่งหากคุณวางแผนที่จะสร้างเซอร์วิสโปรแกรมให้โปรแกรมเมอร์รายอื่นใช้งาน. ดูในสถานการณ์ที่ 10 ของหัวข้อ "เคล็ดลับในการใช้ Teraspace" ในหน้า 69 ถ้าคุณจำเป็นต้องรวมเซอร์วิสโปรแกรมแบบ Single-level.

- *ทำการเรียกเฉพาะโปรแกรมที่เปิดใช้งาน Teraspace*

  โปรแกรมของคุณสามารถเรียกโปรแกรมภายนอกได้. แต่ถ้าคุณเรียกโปรแกรมที่ไม่ได้เปิดใช้ Teraspace และพารามิเตอร์ของคุณกำหนดเป็น Teraspace โปรแกรมที่เรียกนั้นอาจล้มเหลวได้. ซึ่งข้อแนะนำนี้ยังสามารถนำไปประยุกต์ใช้กับโปรแกรมสิ้นสุดการทำงานของผู้ใช้ (User Exit Program).

  นอกจากนี้คุณต้องแน่ใจว่าโปรแกรมที่เปิดใช้ Teraspace ที่คุณเรียกนั้นไม่ส่งส่งค่าแอดเดรส Teraspace ให้กับโปรแกรมหรือเซอร์วิสโปรแกรมที่ไม่ได้เปิดใช้ Teraspace. ไม่อย่างนั้นแล้วคุณอาจต้องปฏิบัติตามข้อแนะนำอื่นๆ ที่กล่าวถึงในหัวข้อนี้แทน. ถ้าคุณจำเป็นต้องเรียกโปรแกรมที่ไม่ได้เปิดใช้ Teraspace หรือไม่สามารถกำหนดได้ว่าโปรแกรมที่คุณเรียกเปิดใช้ Teraspace หรือไม่ คุณก็สามารถเรียกโปรแกรมนั้นได้โดยปฏิบัติขั้นตอนต่างๆ ที่กล่าวไว้ในสถานการณ์ที่ 9 ของหัวข้อ "เคล็ดลับในการใช้ Teraspace" ในหน้า 69.

- *ทำให้พอยเตอร์อ้างถึงเฉพาะโพรซีเดอร์ที่เปิดใช้ Teraspace*

  โค้ดของคุณยังสามารถรับพอยเตอร์โพรซีเดอร์ และใช้มันในการเรียกโพรซีเดอร์ได้. ควรตรวจดูให้แน่ใจว่าโพรซีเดอร์ที่คุณเรียกอยู่ในโปรแกรมหรือเซอร์วิสโปรแกรมที่เปิดใช้ Teraspace. และตรวจดูด้วยว่ามันไม่ได้ส่งแอดเดรส Teraspace ไปให้โปรแกรมที่ไม่ได้เปิดใช้ Teraspace. ถ้าคุณจำเป็นต้องทำเช่นนั้น หรือไม่สามารถกำหนดได้ว่าโปรแกรมที่คุณเรียกเปิดใช้ Teraspace หรือไม่ คุณก็สามารถเรียกโปรแกรมนั้นได้โดยปฏิบัติขั้นตอนต่างๆ ที่กล่าวไว้ในสถานการณ์ที่ 9 ของหัวข้อ "เคล็ดลับในการใช้ Teraspace" ในหน้า 69.

  โปรแกรมหรือเซอร์วิสโปรแกรมที่มีโพรซีเดอร์ต้องมีโมดูลทุกโมดูลที่เปิดใช้ Teraspace ไม่อย่างนั้นพอยเตอร์โพรซีเดอร์ที่เรียกจะล้มเหลวในขณะที่เรียกใช้และทำให้เกิด MCH4443 ได้. แต่ถ้าโมดูลทั้งหมดในโปแกรมถูกเปิดใช้ Teraspace ไว้แล้ว โพรซีเดอร์ที่ถูกเรียกก็จะถูกเปิดใช้ Teraspace ด้วย.

ถ้าคุณปฏิบัติตามคำแนะนำในหัวข้อนี้ คุณก็สามารถใช้ Teraspace ในโปรแกรมของคุณได้. ได้ อย่างไรก็ดี การใช้ Teraspace จำเป็นที่คุณต้องระวังการเขียนโปรแกรมของคุณให้ดีๆ เนื่องจากหน่วยความจำแบบ Single-level ถูกใช้เป็นค่าดีฟอล์ต. ในหัวข้อต่างๆ ต่อไปนี้จะอธิบายถึงสิ่งที่คุณไม่

สามารถทำได้ใน Teraspace และสิ่งที่คุณไม่ควรทำ. ในบางกรณีระบบจะช่วยป้องกันคุณจากปัญหายุ่งยาก แต่บางครั้งคุณต้องจัดการกับการใช้งานหน่วยความจำแบบ Teraspace และ Single-level ด้วยตนเอง.

- "ระบบจะควบคุมโปรแกรม Teraspace เมื่อโปรแกรมถูกสร้างขึ้น"
- "ระบบจะควบคุมโปรแกรม Teraspace เมื่อโปรแกรมถูกเรียกใช้งาน"
- "ระบบจะควบคุมโปรแกรม Teraspace เมื่อโปรแกรมถูกรัน"

หมายเหตุ: เซอร์วิสโปรแกรมที่ใช้ในโมเดลหน่วยความจำแบบ Inherit ต้องปฏิบัติตามข้อแนะนำเหล่านี้ เนื่องจากเซอร์วิสโปรแกรมอาจถูกสั่งให้ใช้ Teraspace ก็ได้.

ระบบจะควบคุมโปรแกรม Teraspace เมื่อโปรแกรมถูกสร้างขึ้น

ส่วนมากแล้ว ระบบจะช่วยป้องกันคุณจากการทำสิ่งต่างๆ เหล่านี้:

- การผสมผสานการใช้โมเดลหน่วยความจำแบบ Single-level และ Teraspace ร่วมกันในโปรแกรมหรือเซอร์วิสโปรแกรมเดียวกัน.
- การสร้างโมเดลหน่วยความจำ Teraspace ของโปรแกรมหรือเซอร์วิสโปรแกรมที่ถูกระบุค่าดีฟอล์ตของ Activation Group (ACTGRP(*DFTACTGRP)).
- การรวมโปรแกรมที่ใช้ Single-level เข้ากับเซอร์วิสโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Teraspace ซึ่งระบุ Activation Group เป็น *CALLER.

ระบบจะควบคุมโปรแกรม Teraspace เมื่อโปรแกรมถูกเรียกใช้งาน

ในบางครั้งที่เรียกใช้งาน ระบบจะตรวจสอบว่าคุณสร้างโปรแกรมและเซอร์วิสโปรแกรมโดยใช้ทั้งหน่วยความจำแบบ Single-level และแบบ Teraspace พร้อมๆ กันหรือไม่ หรือเซอร์วิสโปรแกรมพยายามเรียกใช้งาน Activation Group เดียวกันหรือไม่. ซึ่งระบบจะส่งข้อความ Access Violation Exception แล้วสิ้นสุดการทำงาน.

ระบบจะควบคุมโปรแกรม Teraspace เมื่อโปรแกรมถูกรัน

ระบบไม่สามารถตรวจพบปัญหาต่างๆ เหล่านี้ได้ในขณะช่วงรันไทม์:

- การเรียกโค้ดของหน่วยความจำแบบ Single-level ที่ไม่ได้เปิดใช้ Teraspace จากโค้ดที่ใช้โมเดลหน่วยความจำแบบ Teraspace.
- ความพยายามใช้พอยเตอร์อ้างถึง Teraspace ในโปรแกรมที่ไม่ได้เปิดใช้ Teraspace. โปรแกรมของคุณต้องเปิดใช้งาน Teraspace ทั้งหมด ไม่ใช่เฉพาะโมดูลที่มีโพรซีเดอร์ที่ต้องการเรียกใช้เท่านั้น.

OS/400 อินเตอร์เฟสของ OS/400 กับ Teraspace

โดยทั่วไปแล้ว OS/400 ถูกสร้างให้เปิดใช้งาน Teraspace.

อินเตอร์เฟสของ OS/400 ที่มีพารามิเตอร์พอยเตอร์มักคาดว่าจะมีแท็กพอยเตอร์แบบ 16 ไบต์ (__ptr128):

- คุณสามารถเรียกอินเตอร์เฟสกับพอยเตอร์ระดับเดียว (ตัวอย่างเช่น `void f(char*p);`) โดยใช้พอยเตอร์แบบ 8 ไบต์ (__ptr64) ได้โดยตรง คอมไพเลอร์จะแปลงพอยเตอร์ให้ตามความจำเป็น. ตรวจดูว่าคุณใช้ไฟล์ Header ของระบบ.

- อินเตอร์เฟสกับพอยเตอร์หลายระดับ (ตัวอย่างเช่น `void g(char**p);`));) ซึ่งตามปกติแล้วคุณต้องทำการประกาศพอยเตอร์แบบ 16 ไบต์ในระดับที่สองเอง. อย่างไรก็ดี เวอร์ชันที่ยอมให้ใช้พอยเตอร์แบบ 8 ไบต์ได้ถูกจัดเตรียมไว้แล้วในระบบอินเตอร์เฟสส่วนใหญ่ เพื่อทำให้มันสามารถถูกเรียกได้โดยตรงจากโค้ดที่ใช้เฉพาะพอยเตอร์แบบ 8 ไบต์เท่านั้น. อินเตอร์เฟสเหล่านี้จะถูกเปิดใช้ผ่านทางไฟล์ Header มาตรฐานเมื่อคุณเลือกอ็อพชัน datamodel(LLP64).

Bindable API สำหรับใช้งาน Teraspace:

IBM provides bindable APIs for allocating and discarding teraspace.[3]

- _C_TS_malloc() ใช้จัดสรรหน่วยความจำใน Teraspace.
- _C_TS_free() ใช้ยกเลิกหน่วยความจำที่จองไว้ของ Teraspace.
- _C_TS_realloc() ใช้เปลี่ยนขนาดของหน่วยความจำที่จองไว้แล้วใน Teraspace.
- _C_TS_calloc() ใช้จัดสรรหน่วยความจำใน Teraspace ล้วกำหนดให้มีค่าเป็นศูนย์.

ฟังก์ชัน `malloc()`, `free()`, `calloc()` และ `realloc()` ถูกใช้สำหรับจัดสรร หรือยกเลิกหน่วยความจำแบบ Single-level หรือ Teraspace ตามโมเดลหน่วยความจำของโปรแกรมที่เรียก จนกว่ามันจะถูกคอมไพล์โดยใช้อ็อพชัน TERASPACE(*YES *TSIFC).

หน่วยความจำแบบแบ่งใช้ของ POSIX และอินเตอร์เฟสไฟล์แม็พหน่วยความจำอาจใช้ Teraspace. สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับ Interprocess Communication API และอินเตอร์เฟส `shmget()`, โปรดดูหัวข้อ *UNIX-type APIs* ในหมวด iSeries Information Center (ในหมวด **Programming** และหมวดย่อย **API**).

ปัญหาที่อาจเกิดขึ้นได้เมื่อคุณใช้ Teraspace

เมื่อคุณใช้ Teraspace ในโปรแกรมของคุณ คุณควรระมัดระวังเกี่ยวกับปัญหาที่อาจเกิดขึ้นได้ดังต่อไปนี้.

- *แอดเดรส Teraspace ไม่สามารถส่งให้โปรแกรมหรือโพรซีเดอร์ที่ไม่เปิดใช้ Teraspace.* เมื่อคุณเรียกโค้ดที่ไม่ได้เปิดใช้ Teraspace พารามิเตอร์ในโปรแกรมที่เรียกไม่สามารถอยู่ใน Teraspace และพอยเตอร์ที่ส่งค่าเป็นพารามิเตอร์สำหรับโปรแกรม และโพรซีเดอร์ที่เรียกไม่สามารถเก็บค่าแอดเดรส Teraspace ได้. ซึ่งจะทำให้เกิดข้อความ Exception MCH0607, MCH3601 หรือ MCH3602 ขึ้นอยู่กับสถานการณ์.
- *คำสั่ง MI บางคำสั่งไม่สามารถประมวลผลแอดเดรส Teraspace ได้.* การใช้แอดเดรส Teraspace ในคำสั่งเหล่านี้จะทำให้เกิดข้อความ Exception MCH0607.
  - CIPHER (เฉพาะอ็อพชันบางอย่างที่จำกัด)
  - MATBPGM
  - MATPG
  - SCANX (เฉพาะอ็อพชันบางอย่างที่จำกัด)
  - SETDP

3. อ็อพชันคอมไพเลอร์ Teraspace ของ TERASPACE(*YES *TSIFC) พร้อมใช้งานสำหรับคอมไพเลอร์ ILE C และ C++ เพื่อแม็พฟังก์ชัน malloc(), free(), calloc() และ realloc() ให้เป็นเวอร์ชัน Teraspace เมื่อ STGMDL(*SNGLVL) ถูกระบุ.

- SETDPADR

- *การเข้าถึงระหว่างงานไม่สามารถคาดการณ์ได้.* ในบางกรณี พอยเตอร์ที่อ้างถึง Teraspace จะถูกส่งมาจากงานอื่นจะสามารถใช้ได้ แม้ว่า Teraspace จะถูกกำหนดให้เป็นโลคัลสำหรับงานนั้น ให้หลีกเลี่ยงการส่งพอยเตอร์ที่อ้างถึง Teraspace ไปยังงานอื่น. เพื่อหลีกเลี่ยงการล้มเหลวของแอ็พพลิเคชันเมื่อการเข้าถึงระหว่างงานไม่ทำงาน.
- *Effective Address Overflow (EAO) อาจทำให้ประสิทธิภาพลดลง.* เหตุการณ์นี้อาจเกิดขึ้นเมื่อการคำนวณแอดเดรสในพอยเตอร์แบบ 16 ไบต์ทำให้เกิดผลลัพธ์เป็นแอดเดรสที่อยู่นอกหน่วยความจำ 16 MB นับจากหน่วยความจำเริ่มต้น. การขัดจังหวะของฮาร์ดแวร์ที่ถูกสร้างขึ้นจะถูกจัดการโดยซอฟต์แวร์ของระบบ. ยิ่งมีการขัดจังหวะบ่อยครั้งก็ยิ่งมีผลกระทบต่อประสิทธิภาพ. หลีกเลี่ยงการคำนวณแอดเดรสที่มีขอบเขต 16 MB ใน Teraspace เมื่อคุณใช้การคำนวณกับพอยเตอร์แบบ 16 ไบต์.

เคล็ดลับในการใช้ Teraspace

คุณอาจพบสถานการณ์ต่างๆ เหล่านี้ เมื่อคุณใช้งานโมเดลหน่วยความจำแบบ Teraspace. ซึ่งเราได้อธิบายถึงวิธีการแก้ไขสถานการณ์ต่างไว้เป็นข้อๆ ดังนี้.

- *สถานการณ์ที่ 1: คุณจำเป็นต้องใช้หน่วยความจำแบบไดนามิกมากกว่า 16 MB ด้วยการจัดสรรเพียงครั้งเดียว*

  ให้ใช้ `_C_TS_malloc` หรือระบุ `TERASPACE(*YES *TSIFC)` ในคำสั่งสร้างของคอมไพเลอร์ก่อนที่จะใช้ `malloc`. วิธีนี้จะทำให้มีหน่วยความจำ Heap ให้โปรแกรมที่เปิดใช้ Teraspace.
- *สถานการณ์ที่ 2: คุณจำเป็นต้องใช้หน่วยความจำแบบแบ่งใช้ที่มีขนาดมากกว่า 16 MB*

  ให้ใช้หน่วยความจำแบบแบ่งใช้ (`shmget`) กับอ็อพชันของ Teraspace.
- *สถานการณ์ที่ 3: คุณจำเป็นต้องเข้าถึงไฟล์แบบ byte-stream ขนาดใหญ่อย่างมีประสิทธิภาพ*

  ให้ใช้หน่วยความจำแม็พไฟล์ (`mmap`).

  คุณสามารถเข้าถึงไฟล์ในหน่วยความจำได้จากโปรแกรมที่เปิดใช้ Teraspace แต่เพื่อให้ได้ประสิทธิภาพสูงสุดให้ใช้โมเดลหน่วยความจำแบบ Teraspace กับพอยเตอร์ขนาด 8 ไบต์.
- *สถานการณ์ที่ 4: คุณจำเป็นต้องใช้หน่วยความจำต่อเนื่องแบบอัตโนมัติหรือแบบสแตติกที่มีขนาดมากกว่า 16 MB*

  ให้ใช้โมเดลหน่วยความจำแบบ Teraspace. คุณสามารถใช้ Teraspace กับพอยเตอร์แบบ 8 ไบต์หรือ 16 ไบต์ก็ได้ แต่เพื่อให้ได้ประสิทธิภาพสูงสุดให้เลือกพอยเตอร์แบบ 8 ไบต์.
- *สถานการณ์ที่ 5: แอ็พพลิเคชันของคุณทำให้มีการใช้พอยเตอร์ Space อย่างหนัก*

  ให้ใช้โมเดลหน่วยความจำแบบ Teraspace และพอยเตอร์แบบ 8 ไบต์เพื่อลด Footprint ในหน่วยความจำ และยังช่วยเพิ่มความเร็วในการทำงานของพอยเตอร์อีกด้วย.
- *สถานการณ์ที่ 6: คุณจำเป็นต้องย้ายโค้ดจากระบบหนึ่งและต้องการหลีกเลี่ยงปัญหาในการใช้งานพอยเตอร์แบบ 16 ไบต์*

  ให้ใช้โมเดลหน่วยความจำแบบ Teraspace และพอยเตอร์แบบ 8 ไบต์.
- *สถานการณ์ที่ 7: คุณต้องการใช้หน่วยความจำแบบ Single-level ในโปรแกรม Teraspace*

  ในบางครั้งทางเลือกของคุณเหลือเพียงการใช้หน่วยความจำแบบ Single-level ในโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Teraspace ของคุณ. ตัวอย่างเช่น คุณอาจจำเป็นต้องใช้มันในการ

เก็บพารามิเตอร์สำหรับการเรียกโปรแกรมหรือเซอร์วิสโปรแกรมที่ไม่ได้เปิดใช้ Teraspace. หรือคุณอาจจำเป็นต้องเก็บข้อมูลผู้ใช้สำหรับการสื่อสารภายใน. คุณสามารถใช้หน่วยความจำ Single-level ได้จากแหล่งต่างๆ ต่อไปนี้:

- หน่วยความจำใน user space ที่ได้มาจากคำสั่ง CRTS MI
- หน่วยความจำแบบ Single-level เวอร์ชัน `malloc`
- การอ้างถึงหน่วยความจำแบบ Single-level ที่ส่งให้โปรแกรมของคุณ
- เนื้อที่ของ Heap ของหน่วยความจำแบบ Single-level ที่ได้มาจากคำสั่ง ALCHS MI

- *สถานการณ์ที่ 8:ใช้ประโยชน์จากพอยเตอร์แบบ 8 ไบต์ในโค้ดของคุณ*

  สร้างโมดูลและโปรแกรมของคุณโดยใช้ STGMDL(*TERASPACE). ให้ใช้DTAMDL (*LLP64) หรือประกาศ (`__ptr64`) เอง เพื่อใช้พอยเตอร์แบบ 8 ไบต์อ้างถึง Teraspace (ตรงข้ามกับพอยเตอร์แบบ 16 ไบต์อ้างถึง Teraspace). จากนั้นคุณก็สามารถใช้ประโยชน์พอยเตอร์ได้ตามที่กล่าวไว้ในหัวข้อ “การใช้ประโยชน์จากพอยเตอร์ขนาด 8 ไบต์ในโค้ด C และ C++ ของคุณ” ในหน้า 63.

- *สถานการณ์ที่ 9:การรวมโมดูลที่ใช้โมเดลหน่วยความจำแบบ Single-level*

  คุณไม่สามารถรวมโมดูลหน่วยความจำแบบ Single-level เข้ากับโมเดลหน่วยความจำ Teraspace ได้. ถ้าคุณจำเป็นต้องทำเช่นนั้น ขั้นแรกให้คุณลองหาโมดูลเวอร์ชันที่ใช้ (หรือ inherit) โมเดลหน่วยความจำแบบ Teraspace ดูก่อน ถ้ามีก็ให้ใช้ตามขั้นตอนในหัวข้อ “ข้อควรปฏิบัติในการใช้ Teraspace” ในหน้า 65. แต่ถ้าไม่มี คุณก็มีสองทางเลือก:

  - รวมโมดูลให้เป็นเซอร์วิสโปรแกรมแยกต่างหาก. เซอร์วิสโปรแกรมจะใช้โมเดลหน่วยความจำแบบ Single-level จากนั้นใช้วิธีในสถานการณ์ที่ 10 เพื่อเรียกใช้ต่อไป.
  - รวมโมดูลให้เป็นโปรแกรมแยกต่างหาก. โปรแกรมจะใช้โมเดลหน่วยความจำแบบ Single-level. จากนั้นใช้วิธีในสถานการณ์ที่ 11 เพื่อเรียกใช้ต่อไป.

- *สถานการณ์ที่ 10: การรวมเซอร์วิสโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Single-level*

  คุณสามารถรวมโปรแกรม Teraspace ของคุณเข้ากับเซอร์วิสโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Single-level ถ้าเซอร์วิสโปรแกรมทั้งสองทำงานอยู่ใน Activation Groups ที่แยกจากกัน. คุณไม่สามารถทำแบบนี้ได้ ถ้าเซอร์วิสโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Single-level ระบุอ็อพชันเป็น ACTGRP(*CALLER).

  ถ้าเซอร์วิสโปรแกรมที่ใช้โมเดลหน่วยความจำแบบ Single-levelไม่ได้เปิดใช้ Teraspace เอาไว้ ให้คุณลองค้นหาเวอร์ชันที่เปิดใช้ Teraspace ดูก่อน. ถ้าคุณหาไม่พบ ให้ดูในสถานการณ์ 11 ด้านล่างนี้.

- *สถานการณ์ที่ 11: การเรียกโปรแกรมหรือเซอร์วิสโปรแกรมที่ไม่เปิดใช้ Teraspace*

  ขั้นแรก พยายามเขียนโปรแกรมของคุณโดยไม่เรียกโปรแกรมที่ไม่ได้เปิดใช้ Teraspace (ถ้าทำได้). อย่างไรก็ตาม คุณสามารถเขียนโปรแกรมในลักษณะนี้ได้ หากคุณระมัดระวังในการส่งผ่านเฉพาะพารามิเตอร์ที่เก็บอยู่ในหน่วยความจำ Single-level เท่านั้น.

  ในการทำเช่นนี้ ให้คัดลอกข้อมูลจากหน่วยความจำ Teraspace ไปยัง Single level ส่งผ่านไปให้โปรแกรม และเมื่อมันถูกส่งกลับมา ให้คัดลอกผลลัพธ์ที่ได้ หรือเปลี่ยนหน่วยความจำกลับเป็น Teraspace.

  คุณไม่สามารถทำให้พอยเตอร์โพรซีเดอร์ทำการเรียกจากโมเดลหน่วยความจำแบบ Teraspace ไปยังโพรซีเดอร์ในโปรแกรมที่ไม่ได้เปิด Teraspace เอาไว้ได้.

- *สถานการณ์ที่ 12: การเรียกฟังก์ชันที่มี พารามิเตอร์แบบ pointer-to-pointer*

  การเรียกบางฟังก์ชันที่มีพารามิเตอร์แบบ pointer-to-pointer จำเป็นต้องมีการจัดการแบบพิเศษจากโมดูลซึ่งคอมไพล์ด้วย DTMDL(อ็อพชัน *LLP64). การแปลงแบบชัดเจนระหว่างพอยเตอร์แบบ 8 และ 16 ไบต์ ใช้กับพารามิเตอร์ของพอยเตอร์. ซึ่งใช้ไม่ได้กับอ็อบเจ็กต์ข้อมูลที่ถูกชี้โดย พารามิเตอร์ของพอยเตอร์, แม้ว่าเป้าหมายของพอยเตอร์จะเป็นพอยเตอร์เช่นกัน. ตัวอย่างเช่น, การใช้อินเตอร์เฟส **char**** ที่แสดงในไฟล์ส่วนหัว ซึ่งอ้างแบบจำลองข้อมูล P128 ที่ใช้โดยทั่วไป จำเป็นต้องมีโค้ดในโมดูล ที่สร้างขึ้นด้วยแบบจำลองข้อมูล LLP64. ตรวจสอบให้แน่ใจว่าส่งแอดเดรสของพอยเตอร์แบบ 16 ไบต์ สำหรับกรณีนี้แล้ว. ดังตัวอย่างต่อไปนี้:

  - ในตัวอย่างนี้, คุณได้สร้างโปรแกรมแบบจำลองหน่วยเก็บเทราสเปซโดยใช้พอยเตอร์แบบ 8– ไบต์กับอ็อพชัน `STGMDL(*TERASPACE)DTAMDL(*LLP64)` ในคำสั่งสร้าง, เช่น `CRTCMOD`. ขณะนี้คุณต้องการส่งพอยเตอร์ให้ พอยเตอร์ให้อักขระใน array จากโปรแกรมแบบจำลองหน่วยเก็บเทราสเปซ ไปยังอินเตอร์เฟส P128 **char****. ในการทำเช่นนั้น, คุณต้องแสดงพอยเตอร์แบบ 16– ไบต์อย่างชัดเจน:

    ```
    #pragma datamodel(P128)
    void func(char **);
    #pragma datamodel(pop)

    char myArray[32];
    char *_ptr128 myPtr;

    myPtr = myArray; /* assign address of array to 16-byte pointer */
    func(&myPtr);    /* pass 16-byte pointer address to the function */
    ```

  - application programming interface (API) ที่มีพารามิเตอร์แบบ pointer-to-pointer ที่ใช้ทั่วไปคือ iconv. คาดว่าจะมีเพียงพอยเตอร์แบบ 16– ไบต์ เท่านั้น. นี่คือส่วนของไฟล์ส่วนหัวสำหรับ **iconv**:

    ```
    ...
    #pragma datamodel(P128)
    ...
    size_t inconv(iconv_t cd,
                  char     **inbuf,
                  size_t   *inbytesleft,
                  char     **outbuf,
                  size_t   *outbytesleft);
    ...
    #pragma datamodel(pop)
    ...
    ```

    โค้ดต่อไปนี้จะเรียก **iconv** จากโปรแกรม ที่คอมไพล์กับอ็อพชัน DTAMDL(*LLP64):

    ```
    ...
    iconv_t myCd;
    size_t myResult;
    char *_ptr128 myInBuf, myOutBuf;
    size_t myInLeft, myOutLeft;
    ...
    myResult = inconv(myCd, &myInBuf, &myInLeft, &myOutBuf, &myOutLeft);
    ...
    ```

นอกจากนี้คุณควรทราบว่าไฟล์ส่วนหัวของอินเตอร์เฟส Retrieve Pointer to User Space (QUSPTRUS) จะระบุพารามิเตอร์ void* ซึ่งตามจริงแล้วคาดว่าจะมีพอยเตอร์ไปยังพอยเตอร์. ตรวจสอบให้แน่ใจว่าส่งแอดเดรส ของพอยเตอร์แบบ 16– ไบต์สำหรับ operand ที่สอง.

- *สถานการณ์ที่ 13: การเข้าใช้งานพารามิเตอร์สำหรับ การประมวลผลคำสั่ง, การตรวจสอบความถูกต้อง, และ prompt override programs*

  การประมวลผลคำสั่ง, การตรวจสอบความถูกต้อง, และ prompt override programs ถูกสร้างด้วยแบบจำลองหน่วยเก็บเทราสเปซจะรับพารามิเตอร์ในหน่วยเก็บ แบบระดับเดียว. โปรแกรมเช่นนี้อาจรับพารามิเตอร์ในหน่วยเก็บเทราสเปซหาก ถูกเรียกโดยใช้ CALL จากบรรทัดรับคำสั่ง.

  โปรแกรมเหล่านี้ไม่สามารถแอ็กเซส พารามิเตอร์โดยใช้พอยเตอร์แบบ 8– ไบต์หากไม่มีการก็อปปีพารามิเตอร์ไปยังเทราสเปซก่อน. วิธีการหนึ่งเพื่อให้แอ็พพลิเคชันที่เหลือ ได้รับประโยชน์สูงสุดจากฟังก์ชันเทราสเปซ คือการสร้างการประมวลผลข้อความ, การตรวจสอบความถูกต้อง, และ prompt override program โดยใช้อ็อพชัน TERASPACE(*YES *TSIFC) และ DTAMDL (*P128). การใช้อ็อพชันเหล่านี้จะทำให้แน่ใจ ว่าโปรแกรมของคุณเป็น teraspace-enabled, ให้ใช้หน่วยเก็บเทราสเปซเมื่อดำเนินการ malloc, และใช้พอยเตอร์ใช้พอยเตอร์แบบ 16– ไบต์. พารามิเตอร์ใดๆ ที่ถูกเข้าใช้งานด้วย พอยเตอร์แบบ 8– ไบต์ สามารถคัดลอกลงในหน่วยเก็บเทราสเปซซึ่ง จัดสรรด้วย malloc ได้ก่อน.

  โปรแกรมประมวลผลคำสั่งสามารถแทรกโค้ด ลักษณะเช่นนี้เพื่อส่งพารามิเตอร์จากคำสั่งไปยังแอ็พพลิเคชันที่เหลือ ซึ่งใช้แบบจำลองหน่วยเก็บเทราสเปซและพอยเตอร์แบบ 8– ไบต์:

```
#include <stdlib.h>
#include <string.h>

#define ParmLen 32

int main(int argc, char *argv[])
{
   char * myTsPtr;
   void AppFunc(char *__ptr64); /* entry to rest of the application */
   ...
   /* module created with TERASPACE(*YES *TSIFC) */
   myTsPtr = (char *)malloc(ParmLen); /* allocate teraspace storage */
   ...
   /* copy parameter to teraspace */
   memcpy(myTsPtr, argv[1], (size_t)ParmLen);
   /* pass copied parameter along to rest of the application */
   AppFunc(myTsPtr); /* 16-byte pointer implicitly converted to 8-byte */
   ...
}
```

- *สถานการณ์ที่ 14: ฟังก์ชันการประกาศซ้ำ*

  ควรเลี่ยงฟังก์ชันการประกาศซ้ำซึ่งประกาศแล้วในไฟล์ส่วนหัวที่ได้จาก IBM. การประกาศในระดับโลคัลอาจจะไม่มีการระบุความยาวที่ถูกต้องของพอยเตอร์. อินเตอร์เฟสที่ใช้กันทั่วไปเช่นนี้คือ **errno**, ซึ่งดำเนินการเป็น การเรียกฟังก์ชันใน OS/400

- *สถานการณ์ที่ 15: การใช้แบบจำลองข้อมูล *LLP64 กับโปรแกรมและฟังก์ชันที่ให้พอยเตอร์*

  หากคุณจะ ใช้แบบจำลองข้อมูล *LLP64, โปรดตรวจสอบโปรแกรมและฟังก์ชัน ที่ให้พอยเตอร์ให้ดี. หากพอยเตอร์ชี้ไปที่หน่วยเก็บแบบระดับเดียว, จะไม่สามารถกำหนดค่าที่ถูกต้องให้กับพ

อยเตอร์แบบ 8– ไบต์, ดังนั้นไคลเอ็นต์ ของอินเตอร์เฟสเหล่านี้ต้องคงค่าที่ให้ไว้ในพอยเตอร์แบบ 16– ไบต์. API หนึ่งคือ QUSPTRUS. พื้นที่ของผู้ใช้อยู่ในหน่วยเก็บแบบระดับเดียว.

ตัวอย่างของฟังก์ชันที่ให้พอยเตอร์คือฟังก์ชัน Java Native Interface (JNI) `GetStringChars` และ `GetByteArrayElements`. แบบแรกจะส่ง พอยเตอร์ให้กับสตริงของอักขระ Unicode ที่อยู่ในหน่วยเก็บแบบ ระดับเดียว, และแบบที่สองจะส่งพอยเตอร์ไปให้หรือก็อปปีของ array ขั้นต้น, ซึ่งอยู่ในหน่วยเก็บแบบระดับเดียวเช่นกัน.

- *สถานการณ์ที่ 16: การเลี่ยงปัญหาเมื่อใช้ฟังก์ชัน JNI*

  หากคุณจะใช้หน่วยเก็บเทราสเปซและจะเรียกฟังก์ชัน JNI, ให้ติดตั้ง PTF MF26929.

- *สถานการณ์ที่ 17: การดำเนินการดีบักขั้นต้น ที่มีอ็อพชัน Licensed Internal Code DetectConvertTo8BytePointerError*

  สำหรับการดีบักขั้นต้นของโปรแกรมแบบจำลองหน่วยเก็บเทราสเปซซึ่งสร้างด้วย `STGMDL (*TERASPACE)`, ให้พิจารณาการใช้อ็อพชัน Licensed Internal Code `DetectConvertTo8BytePointerError`. การใช้อ็อพชันนี้เมื่อสร้างโมดูล และโปรแกรมจะทำให้เกิดการสร้างโค้ด ซึ่งจะส่งสัญญาณ MCH0609 ในขณะรันไทม์ หากมีการพยายามแปลงแอดเดรสหน่วยเก็บแบบระดับเดียวเป็นพอยเตอร์แบบ 8 ไบต์.

- *สถานการณ์ที่ 18: การใช้อ็อพชัน Licensed Internal Code `MinimizeTeraspaceFalseEAOs`*

  ให้พิจารณาการใช้อ็อพชัน Licensed Internal Code option `MinimizeTeraspaceFalseEAOs` ซึ่งอธิบายไว้ใน Advanced Optimization Techniques, สำหรับโปรแกรม ที่ใช้พอยเตอร์แบบ 16 ไบต์ซึ่งจะแอดเดรสการจัดสรรหน่วยเก็บเทราสเปซที่ ใหญ่กว่า 16 เมกะไบต์. เมื่อใช้พอยเตอร์แบบ 8 ไบต์แทนพอยเตอร์แบบ 16 ไบต์, คุณยังอาจ ลดโอเวอร์เฮด Effective Address Overflow (EAO). โปรดระวังเนื่องจาก มีการสังเกตว่าเมื่อใช้อ็อพชันนี้โดยไม่จำเป็นจะทำให้ประสิทธิภาพ ลดลงเกือบ 15%. อย่างไรก็ตาม, เมื่อใช้อ็อพชันนี้อย่างถูกต้อง, สังเกตว่าประสิทธิภาพการทำงาน จะเพิ่มขึ้นไม่เกิน 60%.

บทที่ 5. แนวคิดในการสร้างโปรแกรม

ขั้นตอนในการสร้างโปรแกรม ILE หรือเซอร์วิสโปรแกรมช่วยให้คุณสามารถควบคุมการออกแบบและดูแลรักษาแอ็พพลิเคชันได้อย่างยืดหยุ่นมาก. โดยกระบวนการดังกล่าวจะแบ่งออกเป็นสองขั้นตอน:

1. คอมไพล์ซอร์สโค้ดให้เป็นโมดูล.
2. รวมโมดูลเข้าไปในโปรแกรม ILE หรือเซอร์วิสโปรแกรม. การรวมจะเกิดขึ้นเมื่อคำสั่ง Create Program (CRTPGM) หรือ Create เซอร์วิสโปรแกรม (CRTSRVPGM) ถูกรัน.

ในบทนี้จะอธิบายถึงแนวคิดที่เกี่ยวพันกับการรวม และกระบวนการสร้างโปรแกรม ILE หรือเซอร์วิสโปรแกรม. แต่ก่อนที่คุณจะอ่านบทนี้ คุณควรทำความรู้จักกับแนวคิดในการรวมที่อธิบายไว้ในบทที่ 2, “แนวคิด ILE ขั้นพื้นฐาน”, ในหน้า 13 ก่อน.

คำสั่งสร้างโปรแกรม และเซอร์วิสโปรแกรม

คำสั่ง Create Program (CRTPGM) และ Create เซอร์วิสโปรแกรม (CRTSRVPGM) ดูคล้ายกันและใช้พารามิเตอร์บางอย่างรวมกัน. การเปรียบเทียบพารามิเตอร์ที่ใช้ในคำสั่งทั้งสองจะช่วยให้คุณเข้าใจถึงการใช้งานคำสั่งได้ดียิ่งขึ้น.

ตารางที่ 6แสดงคำสั่ง และพารามิเตอร์พร้อมค่าดีฟอลต์ที่กำหนดไว้.

ตารางที่ 6. พารามิเตอร์สำหรับคำสั่ง CRTPGM และ CRTSRVPGM

กลุ่มพารามิเตอร์	คำสั่ง CRTPGM	คำสั่ง CRTSRVPGM
Identification	PGM(*CURLIB/WORK) MODULE(*PGM)	SRVPGM(*CURLIB/UTILITY) MODULE(*SRVPGM)
Program access	ENTMOD(*FIRST)	EXPORT(*SRCFILE) SRCFILE (*LIBL/QSRVSRC) SRCMBR (*SRVPGM)
Binding	BNDSRVPGM(*NONE) BNDDIR(*NONE)	BNDSRVPGM(*NONE) BNDDIR(*NONE)
Run time	ACTGRP(*ENTMOD)	ACTGRP(*CALLER)
Optimization	IPA(*NO) IPACTLFILE (*NONE)	IPA(*NO) IPACTLFILE (*NONE)
Miscellaneous	OPTION(*GEN *NODUPPROC *NODUPVAR *WARN *RSLVREF) DETAIL(*NONE) ALWUPD(*YES) ALWRINZ (*NO) REPLACE(*YES) AUT (*LIBCRTAUT) TEXT (*ENTMODTXT) TGTRLS (*CURRENT) USRPRF (*USER) STGMDL(*SNGLVL)	OPTION(*GEN *NODUPPROC *NODUPVAR *WARN *RSLVREF) DETAIL(*NONE) ALWUPD(*YES) ALWRINZ (*NO) REPLACE(*YES) AUT (*LIBCRTAUT) TEXT (*ENTMODTXT) TGTRLS (*CURRENT) USRPRF (*USER) STGMDL(*SNGLVL)

พารามิเตอร์ Identification สำหรับคำสั่งทั้งสองจะตั้งชื่ออ็อบเจ็กต์ที่ถูกสร้าง และโมดูลที่คัดลอก. ความแตกต่างระหว่างพารามิเตอร์ทั้งสองก็คือชื่อที่เป็นดีฟอลต์ของโมดูลที่ใช้ในการสร้างอ็อบเจ็กต์. เจ็กต์ สำหรับ CRTPGM ใช้ชื่อ สำหรับโมดูลตามที่ระบุไว้ในพารามิเตอร์ Program (*PGM). ส่วน CRTSRVPGM ใช้ชื่อสำหรับโมดูลตามที่ระบุไว้ในพารามิเตอร์ เซอร์วิสโปรแกรม (*SRVPGM). พารามิเตอร์อื่นๆ ก็ดูเหมือน และใช้งานในลักษณะเดียวกัน.

สิ่งที่เหมือนกันมากที่สุดในคำสั่งทั้งสองคือ วิธีที่ Binder แยกสัญลักษณ์ระหว่างอิมพอร์ต และเอ็กซ์พอร์ต. โดยทั้งสองกรณี Binder จะประมวลผลจากพารามิเตอร์อินพุต Module (MODULE), Bound เซอร์วิสโปรแกรม (BNDSRVPGM), และ Binding Directory (BNDDIR).

สิ่งที่แตกต่างกันที่สุดในคำสั่งก็คือพารามิเตอร์การเข้าถึงโปรแกรม (โปรดดูใน “การเข้าถึงโปรแกรม” ในหน้า 85). สำหรับคำสั่ง CRTPGM สิ่งที่จำเป็นสำหรับแยกแยะในการรวมก็คือโมดูลใดที่มีโพรซีเดอร์หลักของโปรแกรมอยู่. เมื่อโปรแกรมถูกสร้าง และ Dynamic Program Call ได้ถูกทำขึ้นกับโปรแกรมนี้ กระบวนการจะเริ่มต้นที่โมดูลที่มีโพรซีเดอร์หลักของโปรแกรมอยู่. ส่วนคำสั่ง CRTSRVPGM ต้องการข้อมูลของการเข้าถึงโปรแกรมมากกว่า เนื่องจากมันสามารถให้อินเตอร์เฟสของการเข้าถึงหลายๆ จุดสำหรับโปรแกรมหรือเซอร์วิสโปรแกรมอื่นๆ.

การใช้สิทธิที่รับมา (Use Adopted Authority - QUSEADPAUT)

ค่าระบบ QUSEADPAUT ใช้กำหนดว่าผู้ใช้คนใดสามารถสร้างโปรแกรมโดยใช้แอ็ตทริบิวต์ Adopted Authority (USEADPAUT(*YES)). ผู้ใช้ทั้งหมดที่ให้สิทธิโดยค่าระบบ QUSEADPAUT สามารถสร้างหรือเปลี่ยนแปลงโปรแกรมและเซอร์วิสโปรแกรมเพื่อใช้สิทธิที่รับมา ถ้าผู้ใช้มอบสิทธิที่จำเป็น. โปรดดู iSeries Security Reference เพื่อค้นหาว่า สิทธิใดบ้างที่จำเป็น.

ค่าระบบสามารถเก็บชื่อของ Authorization List. ซึ่งสิทธิของผู้ใช้จะถูกตรวจสอบกับรายการ. การถ้าผู้ใช้มีสิทธิ *USE ใน Authorization List เป็นอย่างน้อย ผู้ใช้ก็สามารถสร้าง, เปลี่ยนแปลง หรืออัพเดตโปรแกรมหรือเซอร์วิสโปรแกรมด้วยแอ็ตทริบิวต์ USEADPAUT(*YES). การให้สิทธิแก่ Authorization List ไม่สามารถได้รับจากสิทธิที่รับมาได้.

ถ้า Authorization List ถูกตั้งชื่อไว้แล้วในค่าระบบแต่ Authorization List หายไป ฟังก์ชันก็จะไม่ทำงาน. ซึ่งในกรณีนี้จะมีข้อความแจ้งให้คุณทราบ. อย่างไรก็ตาม ถ้าโปรแกรมที่ถูกสร้างโดยใช้ QPRCRTPG API และค่า *NOADPAUT ถูกระบุไว้ในอ็อพชันเท็บเพลต โปรแกรมก็ยังสามารถสร้างได้อย่างสมบูรณ์แม้จะไม่มี Authorization List อยู่ก็ตาม. หากมีมากกว่าหนึ่งฟังก์ชันที่จำเป็นสำหรับคำสั่งหรือ API และ Authorization List หายไป ฟังก์ชันจะไม่สามารถทำงานได้.

ตารางที่ 7. ค่าต่างๆ ที่เป็นได้สำหรับ QUSEADPAUT

ค่า	คำอธิบาย
authorizationlist name	ข้อความวินิจฉัยจะแจ้งให้ทราบว่าโปรแกรมนั้นถูกสร้างโดยใช้ USEADPAUT(*NO) หากข้อสังเกตทั้งหมดนี้เป็นจริง: • Authorization List ถูกระบุสำหรับค่าระบบของ QUSEADPAUT. • ผู้ใช้ไม่มีสิทธิใน Authorization List ที่กล่าวถึงข้างต้น. • ไม่มีความผิดพลาดอื่นๆ เกิดขึ้นเมื่อโปรแกรมหรือเซอร์วิสโปรแกรมถูกสร้างขึ้น. ถ้าผู้ใช้มีสิทธิใน Authorization List โปรแกรมหรือเซอร์วิสโปรแกรมถูกสร้างโดยใช้ USEADPAUT (*YES).
***NONE**	ผู้ใช้ทุกคนที่มีสิทธิด้วยค่าระบบของ QUSEADPAUT สามารถสร้างหรือเปลี่ยนโปรแกรมและเซอร์วิสโปรแกรมเพื่อใช้สิทธิที่ได้รับมา ถ้าผู้ใช้ได้รับสิทธิที่จำเป็น. โปรดดู iSeries Security Reference เพื่อค้นหาว่าสิทธิใดบ้างที่จำเป็น.

สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับค่ากำหนดของระบบ QUSEADPAUT, โปรดดู Security – Reference.

การใช้พารามิเตอร์ Optimization

คุณสามารถระบุพารามิเตอร์ Optimization เพื่อขยายการทำ Optimize โปรแกรม ILE หรือเซอร์วิสโปรแกรมที่รวมไว้. สำหรับรายละเอียดเพิ่มเติมในการ Optimization โปรดดูในหัวข้อ “การวิเคราะห์ระหว่างโพรซีเดอร์ (Interprocedural Analysis – IPA)” ในหน้า 169.

Symbol Resolution

Symbol resolution เป็นกระบวนการรวมที่ทำตรงกับเงื่อนไขเหล่านี้:

- การอิมพอร์ตที่ร้องขอจากชุดของโมดูลที่รวมโดยการก็อปปี้
- ชุดของเอ็กซ์พอร์ตที่จัดเตรียมไว้โดยโมดูลและเซอร์วิสโปรแกรมที่ระบุ

ชุดของเอ็กซ์พอร์ตที่ถูกใช้ในระหว่าง Symbol Resolution สามารถถูกพิจารณาได้เหมือนรายการที่มีลำดับ (เรียงตามตัวเลข). ลำดับของการเอ็กซ์พอร์ตถูกกำหนดโดยเงื่อนไขดังนี้:

- คำสั่งที่อ็อบเจ็กต์ถูกระบุในพารามิเตอร์ MODULE, BNDSRVPGM และ BNDDIR ของคำสั่ง CRTPGM หรือ CRTSRVPGM
- การเอ็กซ์พอร์ตจากภาษารูทีนแบบรันไทม์ของโมดูลที่ระบุ

Resolved และ Unresolved Imports

การอิมพอร์ต และเอ็กซ์พอร์ตจะประกอบด้วยโพรซีเดอร์หรือชนิดของข้อมูล และชื่อ. ซึ่ง **Unresolved Import** เป็นการอิมพอร์ตที่ชนิดของข้อมูล และชื่อไม่ตรงกันกับชนิดของข้อมูล และชื่อที่เอ็กซ์พอร์ตออกมา. ส่วน **Resolved Import** เป็นการอิมพอร์ตที่ชนิดของข้อมูล และชื่อตรงกันกับชนิดของข้อมูล และชื่อที่เอ็กซ์พอร์ตออกมา.

เฉพาะการอิมพอร์ตจากโมดูลที่รวมโดยการก็อปปี้เท่านั้นที่สามารถเข้าไปยังรายการ Unresolved Import ได้. ในระหว่างการทำ Symbol Resolution นั้น Unresolved Import ต่อไปจะถูกใช้เพื่อค้นหารายการเอ็กซ์พอร์ตที่มีลำดับ. ถ้า Unresolved Import ยังปรากฏอยู่หลังจากการตรวจสอบชุดของกราเอ็กซ์พอร์ตที่มีลำดับ อ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมก็จะไม่ถูกสร้างขึ้น. อย่างไรก็ตาม ถ้า *UNRSLVREF ถูกระบุไว้ในพารามิเตอร์ อ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมที่มี Unresolved Imports ก็สามารถสร้างขึ้นได้. แต่ถ้าอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมนั้นพยายามใช้ Unresolved Import ในขณะรัน ผลที่เกิดขึ้นจะเป็นดังนี้:

- ถ้าอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมถูกสร้าง หรืออัพเดตสำหรับระบบเวอร์ชัน 2 รีลีส 3 ข้อความผิดพลาด MCH3203 จะปรากฏขึ้น. ข้อความนั้นคือ “Function error in machine instruction.”
- ข้อความนั้นคือ “Attempt to use an import that was not resolved.”

การรวมโดยการก็อปปี้ (Bind by copy)

โมดูลที่ระบุไว้ในพารามิเตอร์ MODULE จะถูกรวมโดยการก็อปปี้เสมอ. โมดูลที่มีชื่ออยู่ใน Binding Directory ซึ่งระบุโดยพารามิเตอร์ BNDDIR ก็จะถูกรวมโดยการก็อปปี้ถ้าจำเป็น. โมดูลที่มีชื่อใน Binding Directory เป็นสิ่งที่จำเป็นสำหรับกรณีเหล่านี้:

- โมดูลที่เตรียมการเอ็กซ์พอร์ตสำหรับ Unresolved Import
- โมดูลที่มีชื่อเตรียมเอ็กซ์พอร์ตในบล็อกเอ็กซ์พอร์ตปัจจุบันของภาษา Binder ของไฟล์ต้นฉบับที่ถูกใช้สร้างเซอร์วิสโปรแกรม

ถ้าการเอ็กซ์พอร์ตที่ถูกพบในภาษา Binder มาจากอ็อบเจ็กต์โมดูล ซึ่งโมดูลนั้นจะถูกรวมโดยการก็อปปี้ โดยไม่สนใจว่ามันถูกจัดเตรียมให้จากบรรทัดรับคำสั่ง หรือมาจาก Binding Directory. ตัวอย่างเช่น:

```
Module M1:  imports P2
Module M2:  exports P2
Module M3:  exports P3
Binder language S1:  STRPGMEXP PGMLVL(*CURRENT)
                     EXPORT P3
ENDPGMEXP
Binding directory BNDDIR1:  M2
                            M3
CRTSRVPGM SRVPGM(MYLIB/SRV1) MODULE(MYLIB/M1) SRCFILE(MYLIB/S1)
          SRCMBR(S1) BNDDIR(MYLIB/BNDDIR1)
```

เซอร์วิสโปรแกรม SRV1 จะมีโมดูล 3 โมดูลคือ: M1, M2 และ M3. โดย M3 จะถูกก็อปปี้ เนื่องจาก P3 อยู่ในบล็อกเอ็กซ์พอร์ตปัจจุบัน

การรวมโดยการอ้างอิง (Bind by reference)

เซอร์วิสโปรแกรมที่ระบุไว้ในพารามิเตอร์ BNDSRVPGM จะถูกรวมโดยการอ้างอิง. ถ้าเซอร์วิสมีชื่ออยู่ใน Binding Directory ที่เตรียมการเอ็กซ์พอร์ตสำหรับ Unresolved Import ที่เซอร์วิสโปรแกรมถูกรวมโดยการอ้างอิง. เซอร์วิสโปรแกรมที่รวมด้วยวิธีนี้จะไม่มีการเพิ่มอิมพอร์ตใหม่.

หมายเหตุ: เพื่อควบคุมการรวมกันในโปรแกมของคุณให้ดียิ่งขึ้น ให้ระบุชื่อเซอร์วิสโปรแกรมทั่วไป หรือระบุไลบรารี. ไลบรารี ค่า *LIBL ควรใช้ในกรณีที่อยู่ในสภาพแวดล้อมที่ผู้ใช้ควบคุมได้ เมื่อคุณรู้แน่นอนว่าอะไรย้างที่ถูกรวมในโปรแกรมของคุณ. ไม่ควรระบุ BNDSRVPGM(*LIBL/*ALL) พร้อมกับ OPTION(*DUPPROC *DUPVAR). การระบุ *LIBL ด้วย *ALL อาจทำให้เกิดผลลัพธ์ที่ไม่สามารถคาดเดาได้เมื่อโปรแกรมถูกใช้งาน.

การรวมโมดูลจำนวนมากเข้าด้วยกัน

สำหรับพารามิเตอร์โมดูล (MODULE) ในคำสั่ง CRTPGM และ CRTSRVPGM ก็มีข้อจำกัดในจำนวนของโมดูลที่คุณสามารถระบุได้. ถ้าจำนวนของโมดูลที่คุณต้องการรวมเกินจำนวนที่จำกัด คุณสามารถใช้วิธีการต่อไปนี้:

1. ใช้ Binding Directory หลายชุดในการรวมโมดูลจำนวนมากที่เตรียมการเอ็กซ์พอร์ต ซึ่งจำเป็นสำหรับโมดูลอื่น.
2. ใช้หลักการตั้งชื่อโมดูลที่ช่วยให้สามารถระบุชื่อของโมดูลทั่วไปในพารามิเตอร์ MODULE ของคำสั่ง CRTPGM และ CRTSRVPGM ได้. เช่น `CRTPGM PGM(mylib/payroll) MODULE (mylib/pay*)`. โมดูลทุกตัวที่มีชื่อขึ้นต้นด้วย `pay` จะถูกรวมเข้าในโปรแกรม `mylib/payroll` โดยไม่มีเงื่อนไข. ดังนั้นคุณควรพิจารณาหลักการตั้งชื่ออย่างระมัดระวัง เพื่อให้ชื่อแบบทั่วไปที่ระบุในคำสั่ง CRTPGM หรือ CRTSRVPGM ไม่ทำการรวมโมดูลที่คุณไม่ต้องการเข้าไปด้วย.
3. จัดกลุ่มโมดูลแยกตามไลบรารี ดังนั้นค่า *ALL จึงสามารถใช้ได้พร้อมกับการระบุชื่อไลบรารีในพารามิเตอร์MODULE. เช่น `CRTPGM PGM(mylib/payroll) MODULE(payroll/*ALL)`. โมดูลทุกตัวในไลบรารี `payroll` payroll จะถูกรวมเข้าในโปรแกรม `mylib/payroll` โดยไม่มีเงื่อนไข.
4. ใช้วิธีผสมผสานระหว่างการใช้ชื่อแบบทั่วไป (Generic name) กับการระบุไลบรารีที่อธิบายไว้ในวิธีที่ 2 และ 3.
5. สำหรับเซอร์วิสโปรแกรม ให้ใช้ภาษาต้นฉบับของ Binding. การเอ็กซ์พอร์ตที่ระบุในภาษาต้นฉบับของ Binding จะทำให้โมดูลถูกรวมเข้ามาเมื่อตรงกับเงื่อนไขของเอ็กซ์พอร์ต. พอร์ต คำสั่ง RTVBNDSRC สามารถช่วยให้คุณสร้างภาษาต้นฉบับของ Binding ได้. แม้ว่าพารามิเตอร์ MODULE ในคำสั่ง RTVBNDSRC จะจำกัดจำนวนของโมดูลที่ระบุในพารามิเตอร์ MODULE ก็ตาม คุณสามารถใช้ชื่อโมดูลแบบทั่วไป และค่า *ALL พร้อมการระบุชื่อไลบรารี. คุณยังสามารถใช้คำสั่ง RTVBNDSRC ได้หลายๆ ครั้งโดยให้เอาต์พุตเป็นไฟล์ต้นฉบับเดียวกันก็ได้. อย่างไรก็ตาม คุณอาจจำเป็นต้องแก้ไขภาษาต้นฉบับของ Binding ก็ได้ในกรณีนี้.

ความสำคัญของลำดับการเอ็กซ์พอร์ต

ด้วยการเปลี่ยนแปลงคำสั่งเพียงเล็กน้อย คุณก็สามารถสร้างความแตกต่างขึ้นได้ แต่ดูใกล้เคียงกับโปรแกรมที่ถูกต้อง. ลำดับของอ็อบเจ็กต์ที่ระบุอยู่ในพารามิเตอร์ MODULE, BNDSRVPGM และ BNDDIR มีความสำคัญมากถ้าอยู่ในกรณีเหล่านี้:

- โมดูลหรือเซอร์วิสโปรแกรมหลายๆ ตัวถูกเอ็กซ์พอร์ตชื่อสัญลักษณ์ซ้ำซ้อนกัน
- โมดูลอื่นจำเป็นต้องอิมพอร์ตชื่อสัญลักษณ์

แอ็พพลิเคชันส่วนใหญ่มักไม่มีสัญลักษณ์ใดๆ ซ้ำกัน และโปรแกรมเมอร์ไม่จำเป็นต้องไปกังวลกับลำดับของอ็อบเจ็กต์ที่ระบุในพารามิเตอร์มากนัก. แต่สำหรับแอ็พพลเคชันที่มีสัญลักษณ์ที่เอ็กซ์พอร์ตออกมาซ้ำซ้อนกันก็ยังสามารถอิมพอร์ตได้โดยขึ้นอยู่กับลำดับของอ็อบเจ็กต์ที่อยู่ในคำสั่ง CRTPGM หรือ CRTSRVPGM.

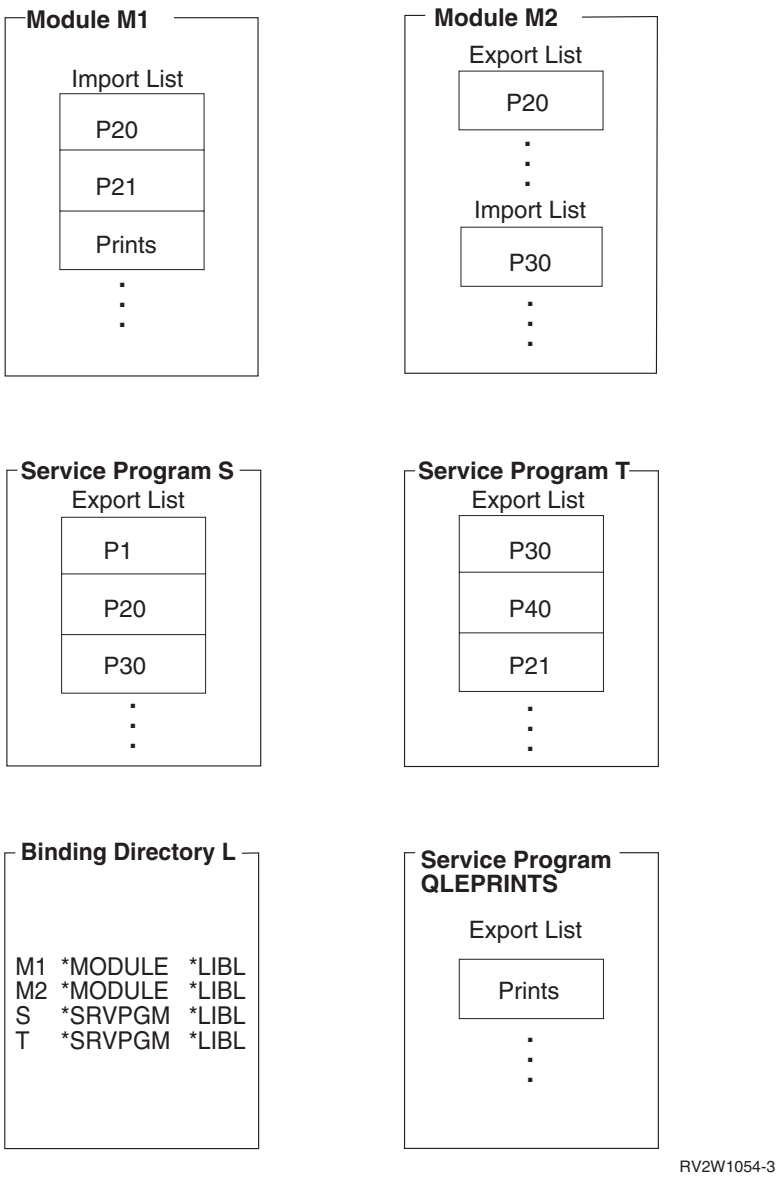
ตัวอย่างต่อไปนี้จะแสดงถึงการทำงานของ Symbol Resolution. โดยโมดูล, เซอร์วิสโปรแกรม และ Binding Directory ในหัวข้อ รูปที่ 30 ในหน้า 81 ถูกใช้สำหรับการร้องขอของ CRTPGM ใน รูปที่ 31 ในหน้า 82 และ รูปที่ 32 ในหน้า 84. สมมุติให้โพรซีเดอร์เป็นสิ่งที่อมพอร์ และเอ็กซ์พอร์ต.

ตัวอย่างยังแสดงให้เห็นถึงหน้าที่ของ Binding Directory ในขั้นตอนการสร้างโปรแกรม. สมมุติว่าไลบรารี MYLIB อยู่ในรายการของไลบรารีสำหรับคำสั่ง CRTPGM และ CRTSRVPGM. CRTSRVPGM คำสั่งต่อไปนี้จะสร้าง Binding Directory L ในไลบรารี MYLIB:

```
CRTBNDDIR BNDDIR(MYLIB/L)
```

คำสั่งต่อไปนี้จะเพิ่มชื่อของโมดูล M1 และ M2 และเซอร์วิสโปรแกรม S และ T เข้าไปใน Binding Directory L:

```
ADDBNDDIRE BNDDIR(MYLIB/L) OBJ((M1 *MODULE) (M2 *MODULE) (S) (T))
```

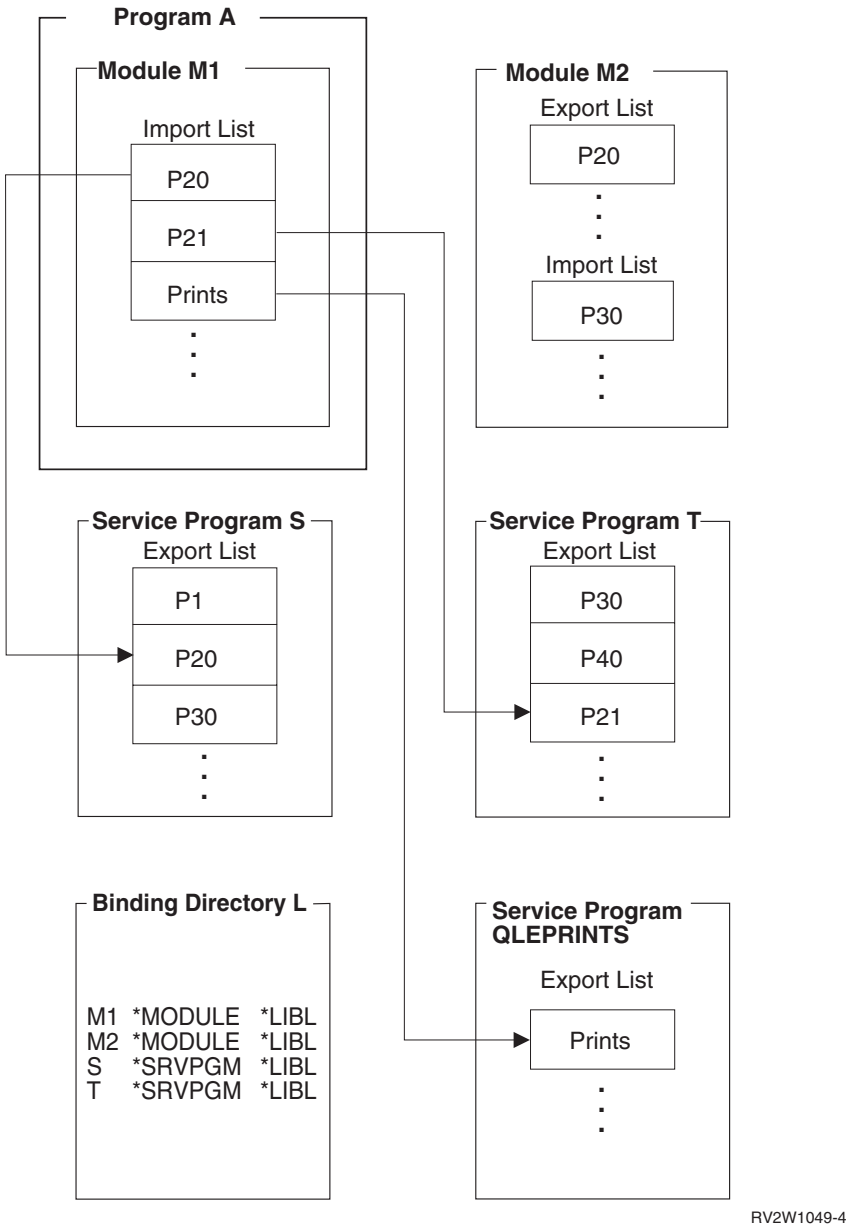


รูปที่ 30. โมดูล เซอร์วิสโปรแกรมและ Binding Directory

ตัวอย่างที่ 1 ในการสร้างโปรแกรม

สมมุติว่าคำสั่งเหล่านี้ถูกใช้ในการสร้างโปรแกรม A ในหัวข้อ รูปที่ 31 ในหน้า 82:

```
CRTPGM  PGM(TEST/A)
        MODULE(*LIBL/M1)
        BNDSRVPGM(*LIBL/S)
        BNDDIR(*LIBL/L)
        OPTION(*DUPPROC)
```



รูปที่ 31. Symbol Resolution and Program Creation: Example 1

ในการสร้างโปรแกรม A นั้น Binder จะประมวลผลอ็อบเจ็กต์ที่ระบุในพารามิเตอร์คำสั่ง CRTPGM ตามลำดับที่ระบุไว้:

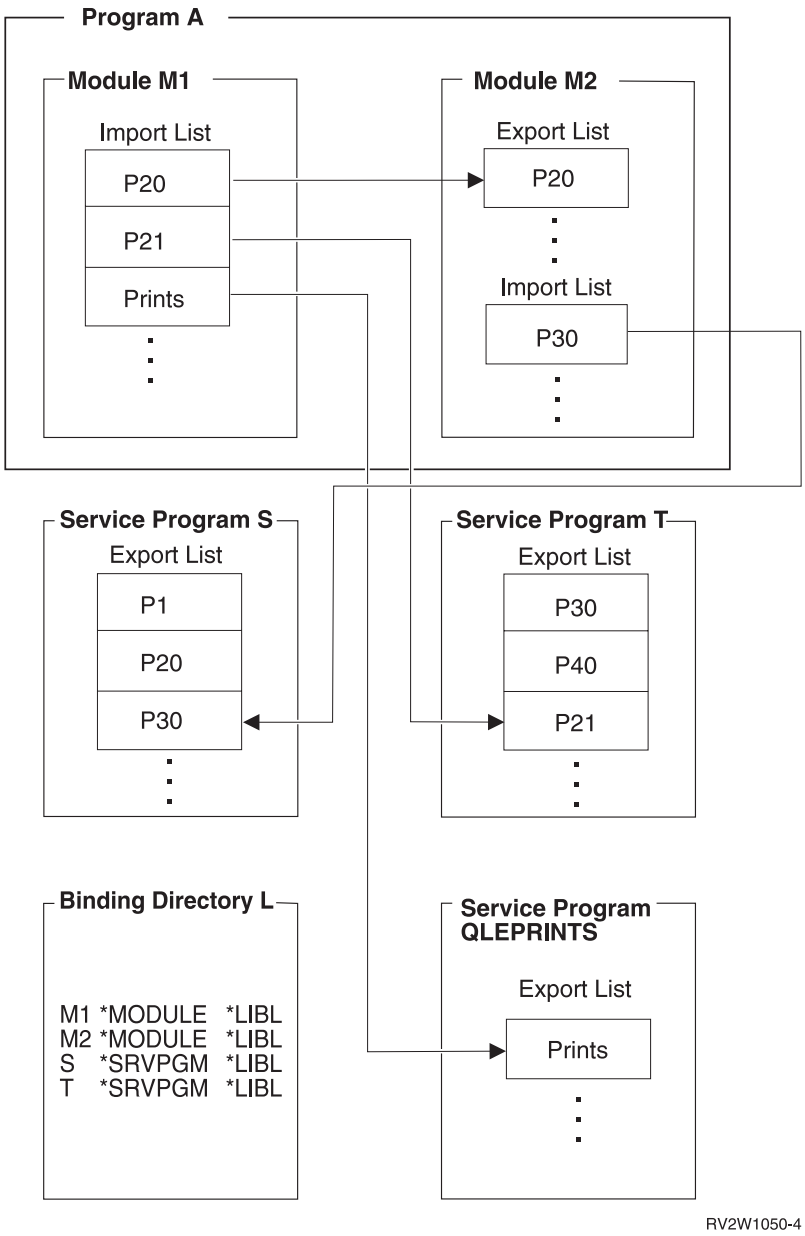
1. ค่าที่ระบุไว้ในพารามิเตอร์แรก (PGM) คือ A ซึ่งเป็นชื่อของโปรแกรมที่จะสร้าง.
2. ค่าที่ระบุไว้ในพารามิเตอร์ที่สอง (module) คือ M1. โดย Binder จะเริ่มต้นที่นี่. ซึ่งโมดูล M1 ประกอบด้วยอิมพอร์ต 3 ยูนิตที่จำเป็นต้อง Resolved ด้วยคือ: P20, P21, และ Prints.
3. ค่าที่ระบุไว้ในพารามิเตอร์ที่สาม (BNDSRVPGM) คือ S ซึ่ง Binder จะทำการสแกนรายการเอ็กซ์พอร์ตของเซอร์วิสโปรแกรม S สำหรับโพรซีเดอร์ที่ทำการ Resolve การร้องขอของ Unresolved Import. Import เนื่องจากรายการเอ็กซ์พอร์ตมีโพรซีเดอร์ P20 ที่การร้องขออิมพอร์ตได้รับการ Resolve.

4. ค่าที่ระบุไว้ในพารามิเตอร์ที่สี่ (BNDDIR) คือ L. โดย Binder จะทำการสแกน Binding Directory L. ต่อไป
   a. อ็อบเจ็กต์แรกที่ระบุใน Binding Directory คือโมดูล M1. โมดูล M1 เป็นที่รู้จักแล้ว เนื่องจากมันถูกระบุไว้ในพารามิเตอร์ของโมดูล แต่มันไม่มีการเอ็กซ์พอร์ตใดๆ.
   b. อ็อบเจ็กต์ที่สองที่ระบุใน Binding Directory คือโมดูล M2. โมดูล M2 มีการเอ็กซ์พอร์ตแต่ไม่มียูนิตใดตรงกันกับ Unresolved Import ที่มีอยู่ (P21 และ Prints).
   c. อ็อบเจ็กต์ที่สามที่ระบุใน Binding Directory คือ เซอร์วิสโปรแกรม S. ซึ่งเซอร์วิสโปรแกรม S ถูกประมวลผลแล้วในขั้นตอนที่ 3 ในหน้า 82และไม่มีเอ็กซ์พอร์ตเพิ่มเติม.
   d. อ็อบเจ็กต์ที่สี่ที่ระบุใน Binding Directory คือ เซอร์วิสโปรแกรม โดย Binder จะสแกนรายการเอ็กซ์พอร์ตของเซิร์ฟเวอร์โปรแกรม T. ซึ่งจะพบโพรซีเดอร์ P21 ทำให้สามารถ Resolve การร้องขออิมพอร์ตได้.
5. การอิมพอร์ตสุดท้ายที่จำเป็นต้อง Resolve (Prints) ไม่ได้ระบุไว้ในพารามิเตอร์ใดๆ. ใดๆ แต่ Binder ก็พบโพรซีเดอร์ Prints ในรายการเอ็กซ์พอร์ตของเซอร์วิสโปรแกรม QLEPRINTS ซึ่งเป็นรูทีนแบบรันไทม์ทั่วไปที่จัดเตรียมโดยคอมไพเลอร์ในตัวอย่างนี้. เมื่อทำการคอมไพล์โมดูล คอมไพเลอร์ระบุให้เป็นค่าดีฟอลต์ของ Binding Directory ที่มีเซอร์วิสโปรแกรมแบบรันไทม์ของตัวมันเอง และของ ILE. นั้นคือวิธีที่ Binder จะทราบว่าควรมองหาการอ้างอิงที่ยังไม่ถูก Resolved ในเซอร์วิสโปรแกรมแบบรันไทม์ที่จัดเตรียมโดยคอมไพเลอร์. ถ้าหลังจากที่ Binder ค้นหาในเซอร์วิสโปรแกรมแบบรันไทม์ แล้วพบการอ้างอิงไม่สามารถ Resolve ได้ การรวมกันก็จะล้มเหลว. อย่างไรก็ตามถ้าคุณระบุ OPTION(*UNRSLVREF) เอาไว้ในคำสั่ง Create โปรแกรมก็จะถูกสร้างขึ้น.

ตัวอย่างที่ 2 ในการสร้างโปรแกรม

รูปที่ 32 ในหน้า 84 แสดงผลของคำสั่ง CRTPGM ที่คล้ายกัน แต่เซอร์วิสโปรแกรมในพารามิเตอร์ BNDSRVPGM จะถูกลบออก:

```
CRTPGM  PGM(TEST/A)
        MODULE(*LIBL/M1)
        BNDDIR(*LIBL/L)
        OPTION(*DUPPROC)
```



รูปที่ 32. Symbol Resolution and Program Creation: Example 2

ในการเปลี่ยนลำดับของอ็อบเจ็กต์ที่ประมวลผลให้คุณเปลี่ยนลำดับการเอ็กซ์พอร์ต. ผลที่ได้จากการสร้างโปรแกรมจะแตกต่างไปจากโปรแกรมที่สร้างในตัวอย่างที่ 1. เนื่องจากเซอร์วิสโปรแกรม S ไม่ได้ถูกระบุไว้นพารามิเตอร์ BNDSRVPGM ของคำสั่ง CRTPGM ดังนั้น Binding Directory จึงประมวลผล. . โดยโมดูล M2 จะทำการเอ็กซพอร์ต P20 และถูกระบุไว้ใน Binding Directory ก่อนเซอร์วิสโปรแกรม S. ดังนั้นโมดูล M2 จึงก็อปปี้ผลลัพธ์ของอ็อบเจ็กต์โปรแกรมในตัวอย่างนี้. เมื่อคุณเปรียบเทียบ รูปที่ 31 ในหน้า 82 กับ รูปที่ 32คุณก็จะพบว่า:

- โปรแกรม A ในตัวอย่างที่ 1 มีเฉพาะโมดูล M1 และใช้โพรซีเดอร์จากเซอร์วิสโปรแกรม S, T, และ QLEPRINTS.
- ในโปรแกรม A ของตัวอย่างที่ 2 โมดูลสองโมดูลที่ชื่อ M1 และ M2 ใช้เซอร์วิสโปรแกรม T และ QLEPRINTS.

โปรแกรมในตัวอย่างที่ 2 จะถูกสร้างตามรายละเอียดดังต่อไปนี้:

1. พารามิเตอร์แรก (PGM) จะระบุชื่อของโปรแกรมที่ถูกสร้าง.
2. ค่าที่ระบุไว้ในพารามิเตอร์ที่สอง (module) คือ M1. โดย Binder จะเริ่มต้นที่นี่. ซึ่งโมดูล M1 ประกอบด้วยอิมพอร์ต 3 ยูนิตที่จำเป็นต้อง Resolved ด้วยคือ: P20, P21, และ Prints.
3. คราวนี้ค่าที่ระบุไว้ในพารามิเตอร์ที่สามไม่ใช่ BNDSRVPGM. แต่เป็น BNDDIR. . ดังนั้น Binder จึงสแกน Binding Directory ที่ระบุ (L).
   a. รายการแรกที่ระบุใน Binding Directory คือโมดูล M1 โดยโมดูล M1. จากไลบรารีนี้ถูกประมวลผลแล้วโดยพารามิเตอร์ของโมดูล
   b. รายการที่สองที่ระบุใน Binding Directory คือโมดูล M2. โดย Binder จะสแกนหารายการเอ็กซ์พอร์ตของโมดูล M2. เนื่องจากรายการเอ็กซ์พอร์ตนั้นมี P20 ซึ่งการร้องขอให้อิมพอร์ตจะได้รับการ Resolve. โมดูล M2 จะถูกรวมโดยการก็อปปี้ และต้องถูกเพิ่มเข้าไปในรายการ Unresolved Import ด้วย. ในตอนนี้รายการร้องขอ Unresolved Import คือ P21, Prints, และ P30.
   c. การประมวลผลจะทำงานไปจนถึงอ็อบเจ็กต์ต่อไปที่ถูกระบุไว้ใน Binding Directory ซึ่งก็คือเซอร์วิสโปรแกรม 'S' ซึ่งเซอร์วิสโปรแกรม S มีเอ็กซพอร์ต P30 สำหรับการร้องขอ Unresolved Import ของ P21 และ Prints. การประมวลผลทำต่อไปจนถึงอ็อบเจ็กต์ต่อไปที่แสดงใน Binding Directory นั้นคือเซอร์วิสโปรแกรม T.
   d. เซอร์วิสโปรแกรม T จะมีเอ็กซ์พอร์ต P21 สำหรับ Unresolved Import.
4. เหมือนในตัวอย่างที่ 1 การอิมพอร์ตของ Prints ไม่ได้ถูกระบุไว้. อย่างไรก็ตามโพรซีเดอร์ก็ถูกตรวจพบในรูทีนแบบรันไทม์ที่จัดเตรียมโดยภาษาที่โมดูล M1 ถูกเขียนขึ้น.

Symbol Resolution จะมีผลกระทบต่อ Strength ของเอ็กซ์พอร์ต. พอร์ต สำหรับรายละเอียดเกี่ยวกับการเอ็กซ์พอร์ตแบบ Strong และ Weak ดูได้จากในหัวข้อ "แนวคิดในการอิมพอร์ตและเอ็กซ์พอร์ต" ในหน้า 88.

การเข้าถึงโปรแกรม

เมื่อคุณสร้างโปรแกรม อ็อบเจ็กต์ หรือเซอร์วิสโปรแกรมของ ILE คุณจำเป็นต้องระบุวิธีการที่โปรแกรมอื่นสามารถเข้าถึงโปรแกรมของคุณ. โดยคำสั่ง CRTPGM คุณสามารถทำได้โดยใช้พารามิเตอร์ Entry Module (ENTMOD). สำหรับคำสั่ง CRTSRVPGM คุณสามารถทำได้โดยใช้พารามิเตอร์ Export (EXPORT) (โปรดดูในหัวข้อ ตารางที่ 6 ในหน้า 75).

พารามิเตอร์ Program Entry โพรซีเดอร์ Module ในคำสั่ง CRTPGM

พารามิเตอร์ Program Entry โพรซีเดอร์ Module (ENTMOD) จะบอกให้ Binder ทราบถึงชื่อของโมดูลที่อยู่ในตำแหน่งต่อไปนี้:

Program entry procedure (PEP)

User entry procedure (UEP)

ข้อมูลนี้ช่วยให้ทราบว่าโมดูลใดมี PEP ที่ต้องใช้ในการควบคุมเมื่อมีการเรียกแบบไดนามิกไปยังโปรแกรมที่ถูกสร้างขึ้น.

ค่าดีฟอลต์สำหรับพารามิเตอร์ ENTMOD คือ *FIRST. ค่านี้จะบอกให้ Binder ใช้ Entry Module ของโมดูลแรกที่มันหาพบในรายชื่อโมดูลที่ระบุไว้ในพารามิเตอร์ของโมดูลที่มี PEP.

ถ้าเงื่อนไขเหล่านี้เป็นจริง:

*FIRST ถูกระบุไว้ในพารามิเตอร์ ENTMOD
โมดูลที่สองถูกตรวจพบ PEP

Binder จะก็อปปี้โมดูลที่สองลงในอ็อบเจ็กต์โปรแกรม และทำการรวมต่อไป. ซึ่งBinder จะไม่สนใจ PEP ที่เกินมา.

ถ้า *ONLY ถูกระบุไว้ใน ENTMOD แสดงว่าจะมีเพียงโมดูลเดียวในโปรแกรมเท่านั้นที่สามารถมี PEP อยู่ได้. ถ้าหาก *ONLY ถูกระบุไว้ แต่มีการตรวจพบ PEP ในโมดูลที่สอง อ็อบเจ็กต์จะไม่ถูกสร้างขึ้น.

ในการสั่งควบคุมด้วยตัวเอง คุณสามารถระบุชื่อของโมดูลที่มี PEP อยู่เองได้. โดย PEP อื่นๆ ที่ตรวจพบจะถูกละทิ้ง. แต่ถ้าโมดูลที่ระบุไว้ไม่มี PEP คำสั่ง CRTPGM ก็จะล้มเหลว.

ในการดูว่าโมดูลใดที่มี Program Entry โพรซีเดอร์ อยู่ ก็ให้คุณใช้คำสั่ง display module (DSPMOD). ซึ่งรายละเอียดจะปรากฏอยู่ในฟิลด์ *Program entry procedure name* ของ Display Module Information. Information ถ้า *NONE ถูกระบุไว้ในฟิลด์ก็แสดงว่าโมดูลนี้ไม่มี PEP. แต่ถ้ามีชื่อปรากฏอยู่ในฟิลด์ก็แสดงว่า โมดูลนั้นมี PEP อยู่.

พารามิเตอร์ Export ในคำสั่ง CRTSRVPGM

พารามิเตอร์ export (EXPORT), source file (SRCFILE), และ source member (SRCMBR) ใช้สำหรับแยกแยะอินเตอร์เฟสพับลิกไปยังเซอร์วิสโปรแกรมที่ถูกสร้าง. พารามิเตอร์ระบุเอ็กซ์พอร์ต (โพรซิเดอร์ หรือข้อมูล) ซึ่งเซอร์วิสโปรแกรมทำให้พร้อมใช้งานสำหรับโปรแกรมหรือเซอร์วิสโปรแกรมของ ILE อื่นๆ.

ค่าดีฟอลต์สำหรับพารามิเตอร์เอ็กซ์พอร์ตคือ *SRCFILE. ซึ่งค่านี้จะสั่งให้ Binder ใช้ไฟล์ที่ระบุไว้ในพารามิเตอร์ SRCFILE ในการอ้างอิงข้อมูลเกี่ยวกับการเอ็กซ์พอร์ตเซอร์วิสโปรแกรม. โปรแกรมข้อมูลที่อยู่ในไฟล์ต้นฉบับนั้นคือซอร์สของภาษา Binder (โปรดดูในหัวข้อ “ภาษา Binder” ในหน้า 90).). โดย Binder จะค้นหาซอร์สของภาษา Binder จากชื่อไฟล์ที่ระบุให้เอ็กซ์พอร์ต, สร้าง Signature เพิ่มขึ้น. ภาษา Binder ยังอนุญาตให้คุณระบุ Signature ที่คุณเลือกแทนการให้ Binder สร้างให้.

| คำสั่ง Retrieve Binder Source (RTVBNDSRC) สามารถใช้ในการสร้างไฟล์ต้นฉบับที่มีซอร์สของ
| ภาษา Binder. ซอร์สอาจจะใช้หลักอย่างใดอย่างหนึ่งระหว่างเซอร์วิสโปรแกรมที่มีอยู่ก่อนหรือกลุ่ม
| ของโมดูล. ซอร์สที่ใช้หลักของ เซอร์วิสโปรแกรมเหมาะสำหรับการสร้างใหม่หรือการปรับปรุงเซอร์
| วิสโปรแกรมนั้นๆ ซอร์สที่ใช้หลักของกลุ่มของโมดูล ประกอบไปด้วยสัญลักษณ์ที่สามาถเอ็กซปอร์
| ตออกจากโมดูลได้ . คุณสามารถแก้ไขไฟล์เพื่อเพิ่มเฉพาะสัญลักษณ์ที่คุณต้องการเอ็กซ์พอร์ตก็ได้
| แล้วจึงระบุชื่อไฟล์นี้ในพารามิเตอร์ SRCFILE ของคำสั่ง CRTSRVPGM หรือคำสั่ง
| UPDSRVPGM.

ค่าอื่นๆ ที่เป็นไปได้สำหรับพารามิเตอร์ export ก็คือ *ALL. เมื่อ EXPORT(*ALL) ถูกระบุ สัญลักษณ์ต่างๆ ทั้งหมดที่เอ็กซ์พอร์ตจากโมดูลที่ก็อปปี้จะถูกเอ็กซ์พอร์ตจากเซอร์วิสโปรแกรม. โดย Signature ที่ถูกสร้างตามเงื่อนไขนี้:

- ตามจำนวนของสัญลักษณ์ที่เอ็กซ์พอร์ต
- เรียงลำดับตัวอักษรของสัญลักษณ์ที่เอ็กซ์พอร์ต

ถ้า EXPORT(*ALL) ถูกระบุ ก็จะไม่มีภาษา Binder ที่จำเป็นสำหรับนิยามเอ็กซ์พอร์ตจากเซอร์วิสโปรแกรม. ค่านี้เป็นค่าที่ใช้งานง่ายที่สุด เนื่องจากคุณไม่ต้องสร้างซอร์สของภาษา Binder. อย่างไรก็ตามเซอร์วิสโปรแกรมที่ระบุ EXPORT(*ALL) เอาไว้จะอัพเดต หรือแก้ไขเอ็กซ์พอร์ตที่ถูกใช้โดยโปรแกรมอื่นได้ยาก. ยาก ถ้าเซอร์วิสโปรแกรมมีการเปลี่ยนแปลง ลำดับ หรือจำนวนของเอ็กซ์พอร์ตอาจมีการเปลี่ยนแปลง. แปลง ดังนั้น Signature ของเซอร์วิสโปรแกรมก็ต้องเปลี่ยนแปลงด้วย. และถ้า Signature เปลี่ยนแปลง โปรแกรมหรือเซอร์วิสโปรแกรมทั้งหมดที่ใช้เซอร์วิสโปรแกรมที่เปลี่ยนแปลงนั้นก็ต้องถูกสร้างใหม่ด้วย.

EXPORT(*ALL) จะแสดงว่าสัญลักษณ์ทั้งหมดที่เอ็กซ์พอร์ตจากโมดูลที่ใช้เซอร์วิสโปรแกรมถูกเอ็กซ์พอร์มาจาก เซอร์วิสโปรแกรม. ILE C สามารถนิยามการเอ็กซ์พอร์ตให้เป็นแบบ Global หรือ Static เฉพาะตัวแปรภายานอกเท่านั้นที่สามารถประกาศใน ILE C ให้เป็นแบบ global ที่พร้อมใช้งานกับ EXPORT(*ALL). สำหรับใน ILE RPG สิ่งเหล่านี้พร้อมใช้งานกับ EXPORT(*ALL):

- ชื่อโพรซีเดอร์หลัก RPG
- ชื่อของโพรซีเดอร์ย่อยที่จะถูกเอ็กซปอร์ต
- ตัวแปรนิยามโดยคีย์เวิร์ด EXPORT

ส่วนใน ILE COBOL องค์ประกอบพื้นฐานของภาษาเป็นแบบการเอ็กซ์พอร์ตโมดูล:

- ชื่อในพารากราฟ PROGRAM-ID ในโปรแกรม COBOL (อย่าสับสนกับอ็อบเจ็กต์ *PGM) ของยูนิตคอมไพล์. การแม็พนี้จะเป็นการเอ็กซ์พอร์ตโพรซีเดอร์แบบ Strong.
- ชื่อที่สร้างจากการคอมไพล์ของ COBOL นำมาจากชื่อที่อยู่ในพารากราฟ PROGRAM-ID ถ้าโปรแกรมนั้นไม่มีแอ็ตทริบิวต์ INITIAL. การแม็พนี้จะเป็นการเอ็กซ์พอร์ตโพรซีเดอร์แบบ Strong. พอร์ต สำหรับรายละเอียดเกี่ยวกับการเอ็กซ์พอร์ตแบบ Strong และ Weak ดูได้จากในหัวข้อ “แนวคิดในการอิมพอร์ตและเอ็กซ์พอร์ต” ในหน้า 88.
- ข้อมูลหรือไฟล์ใดๆที่ถูกประกาศเป็น EXTERNAL. การแม็พนี้จะเป็นการเอ็กซ์พอร์ตโพรซีเดอร์แบบ Weak.

พารามิเตอร์ Export ถูกใช้กับซอร์สไฟล์และพารามิเตอร์ Source Member

ค่าดีฟอลต์ของพารามิเตอร์ export ก็คือ *SRCFILE. ถ้า *SRCFILE ถูกระบุในพารามิเตอร์ export Binder ก็จะใช้พารามิเตอร์ SRCFILE และ SRCMBR เพื่อค้นหาซอร์สของภาษา Binder.

ตัวอย่างต่อไปนี้เป็นการสั่งให้รวมเซอร์วิสโปรแกรมที่ชื่อ UTILITY โดยใช้ค่าดีฟอลต์ในการหาซอร์สภาษา Binder:

```
CRTSRVPGM SRVPGM(*CURLIB/UTILITY)
          MODULE(*SRVPGM)
            EXPORT(*SRCFILE)
          SRCFILE(*LIBL/QSRVSRC)
            SRCMBR(*SRVPGM)
```

สำหรับคำสั่งในการสร้างเซอร์วิสโปรแกรมนี้ สมาชิกที่ชื่อ UTILITY ต้องอยู่ในซอร์สไฟล์ QSRVSRC. โดยสมาชิกนี้ต้องมีซอร์สของภาษา Binderอยู่ ซึ่ง Binder จะทำการแปลให้เป็น Signature และชุดของเอ็กซ์พอร์ต. ค่าดีฟอลต์ในการรับซอร์สของภาษา Binder จากสมาชิกที่มีชื่อเดียวกันกับชื่อของเซอร์วิสโปรแกรม นั้นคือ UTILITY. . ถ้าหากไฟล์, สมาชิก, หรือ ซอร์สของภาษา Binder ที่มีค่าระบุในพารามิเตอร์เหล่านี้ แต่ค้นหาไม่พบ เซอร์วิสโปรแกรมก็จะไม่ถูกสร้างขึ้น

ความกว้างสูงสุดของไฟล์สำหรับพารามิเตอร์ SRCFILE

ใน V3R7 หรือรีลีสใหม่ๆ ความกว้างสูงสุดของไฟล์สำหรับพารามิเตอร์ Source File (SRCFILE) ในคำสั่ง CRTSRVPGM หรือ UPDSRVPGM คือ 240 ตัวอักษร. . ถ้าไฟล์ใหญ่เกินกว่าความกลว้างสูงสุด ข้อความ CPF5D07 ก็จะปรากฏขึ้น. สำหรับใน V3R2 ความกว้างสูงสุดคือ 80 ตัวอักษร. อักษร สำหรับ V3R6, V3R1 และ V2R3 ไม่มีข้อจำกัดของความกว้างสูงสุด.

แนวคิดในการอิมพอร์ตและเอ็กซ์พอร์ต

ภาษา ILE สนับสนุนการอิมพอร์ตและเอ็กซ์พอร์ตแบบต่างๆ ดังต่อไปนี้:

- เอ็กซ์พอร์ตข้อมูลแบบ Weak
- อิมพอร์ตข้อมูลแบบ Weak
- เอ็กซ์พอร์ตข้อมูลแบบ Strong
- อิมพอร์ตข้อมูลแบบ Strong
- เอ็กซ์พอร์ตโพรซีเดอร์แบบ Strong
- เอ็กซ์พอร์ตโพรซีเดอร์แบบ Weak
- อิมพอร์ตโพรซีเดอร์

อ็อบเจ็กต์โมดูลของ ILE สามารถเอ็กซ์พอร์ตโพรซีเดอร์หรือตัวข้อมูลไปยังโมดูลอื่นได้. ได้ และอ็อบเจ็กต์โมดูลของ ILE ก็สามารถอิมพอร์ต (อ้างถึง) โพรซีเดอร์หรือตัวข้อมูลจากโมดูลอื่นได้. เมื่อใช้อ็อบเจ็กต์โมดูลในคำสั่ง CRTSRVPGM เพื่อสร้างเซอร์วิสโปรแกรม มันจะทำการอิมพอร์ตยูนิตจากเซอร์วิสโปรแกรม. (โปรดดูในหัวข้อ “พารามิเตอร์ Export ในคำสั่ง CRTSRVPGM” ในหน้า 86.) ค่า Strength (Strong หรือ Weak) ของเอ็กซ์พอร์ตขึ้นอยู่กับภาษที่ใช้ในการเขียนโปรแกรม. ค่า Strength แตกต่างกันตามช่วงเวลาในการใช้งานลักษณะพิเศษของมัน เช่น ขนาดของข้อมูล. ลักษณะพิเศษของการเอ็กซ์พอร์ตแบบคือการทำงานในเวลาที่รวม. ค่า Strength ของการเอ็กซ์จะมีผลกระทบต่อ Symbol Resolution.

- Binder ใช้ลักษณะพิเศษของการเอ็กซ์พอร์ตแบบ Strong ถ้ามีการเอ็กซ์พอร์ตแบบ Weak ที่มีชื่อเหมือนกัน.
- ถ้าการเอ็กซ์พอร์ตแบบ Weak ไม่มีชื่อเหมือนกับการเอ็กซ์พอร์ตแบบ Strong คุณก็ไม่สามารถกำหนดลักษณะพิเศษได้จนกว่าจะใช้งาน. ซึ่งในช่วงเวลานั้น ถ้ามีการเอ็กซ์พอร์ตแบบ Weak หลายตัวที่มีชื่อเหมือนกัน โปรแกรมจะเลือกใช้ที่มีขนาดใหญ่ที่สุด. ซึ่งเป็นสิ่งที่ถูกต้อง จนกว่าการเอ็กซ์พอร์ตแบบ Weak จะถูกใช้งานโดยมีชื่อเหมือนกันปรากฏอยู่
- ในช่วงเวลารวม ถ้า Binding Directory ถูกใช้และการเอ็กซ์พอร์ตแบบ Weak ตรงกันกับการอิมพอร์ตแบบ Weak พวกมันก็จะถูกรวมกัน. เมื่ออิมพอร์ตทั้งหมดถูก Resolve แล้ว การค้นหาผ่าน

ทาง Binding Directory ก็จะยุติลง. . การเอ็กซ์พอร์ตแบบ Weak ที่ซ้ำกันก็จะไม่ถูกตรวจพบว่าเป็นตัวแปร หรือโพรซีเดอร์ที่ซ้ำกัน. ลำดับของไอเท็มใน Binding Directory จึงมีความสำคัญมาก.

คุณสามารถทำการการเอ็กซ์พอร์ตแบบ Weak ออกนอกอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมสำหรับ Resolution ในเวลาใช้งานได้. ซึ่งตรงข้ามกับการเอ็กซ์พอร์ตแบบ Strong ที่คุณสามารถเอ็กซ์พอร์ตได้เฉพาะนอก เซอร์วิสโปรแกรม และเฉพาะเวลาที่รวมเท่านั้น.

อย่างไรก็ตาม คุณไม่สามารถทำการการเอ็กซ์พอร์ตแบบ Strong นอกอ็อบเจ็กต์โปรแกรมได้. คุณสามารถเอ็กซ์พอร์ตโพรซีเดอร์แบบ Strong นอกเซอร์วิสโปรแกรมได้เงื่อนไขข้อใดข้อหนึ่งเป็นจริง:

- อิมพอร์ตในโปรแกรมที่รวมเซอร์วิสโปรแกรมโดยอ้างอิง.
- อิมพอร์ตในเซอร์วิสโปรแกรมอื่นที่รวมโดยอ้างอิงกับโปรแกรม.

เซอร์วิสโปรแกรมนิยามอินเตอร์เฟสพับลิกผ่านทางซอร์สภาษา Binding.

คุณสามารถทำให้การเอ็กซ์พอร์ตโพรซีเดอร์แบบ Weak กลายเป็นส่วนหนึ่งของอินเตอร์เฟสพับลิกสำหรับเซอร์วิสโปรแกรมได้โดยผ่านทางซอร์สภาษา Binding. อย่างไรก็ตาม การเอ็กซ์พอร์ตโพรซีเดอร์แบบ Weak จากเซอร์วิสโปรแกรมผ่านทางซอร์สภาษา Binding จะไม่จัดว่าเป็นแบบ Weak อีกต่อไป. โดยมันจะกลายเป็นการเอ็กซ์พอร์ตโพรซีเดอร์แบบ Strong.

คุณสามารถเอ็กซ์พอร์ตข้อมูลแบบ Weak ไปยัง Activation Group เท่านั้น. คุณไม่สามารถทำให้มันกลายเป็นส่วนหนึ่งของอินเตอร์เฟสพับลิกที่เอ็กซ์พอร์ตมาจากเซอร์วิสโปรแกรมผ่านทางซอร์สภาษา Binding. การระบุข้อมูลแบบ Weak ในซอร์สภาษา Binding จะทำให้การรวมกันล้มเหลว.

ตารางที่ 8สรุปประเภทของการอิมพอร์ต และเอ็กซ์พอร์ตที่สนับสนุนโดยภาษา ILE:

ตารางที่ 8. อิมพอร์ตและเอ็กซ์พอร์ตที่สนับสนุนโดยภาษา ILE

ภาษา ILE	การเอ็กซ์พอร์ตแบบ Weak	การอิมพอร์ตแบบ Strong	การเอ็กซ์พอร์ตแบบ Strong	การอิมพอร์ตข้อมูลแบบ Strong	การเอ็กซ์พอร์ตโพรซีเดอร์แบบ Strong	การเอ็กซ์พอร์ตโพรซีเดอร์แบบ Weak	การอิมพอร์ตโพรซีเดอร์
RPG IV	ไม่	ไม่	ใช่	ใช่	ใช่	ไม่	ใช่
COBOL[2]	ใช่[3]	ใช่[3]	ไม่	ไม่	ใช่[1]	ไม่	ใช่
CL	ไม่	ไม่	ไม่	ไม่	ใช่ [1]	ไม่	ใช่
C	ไม่	ไม่	ใช่	ใช่	ใช่	ไม่	ใช่
C++	ไม่	ไม่	ใช่	ใช่	ใช่	ใช่	ใช่

หมายเหตุ:

1. COBOL และ CL อนุญาตให้โพรซีเดอร์เดียวเท่านั้นที่สามารถเอ็กซ์พอร์ตออกจากโมดูลได้.
2. COBOL ใช้โมเดลข้อมูลแบบ Weak. ข้อมูลที่ประกาษเป็น External จะกลายเป็นการเอ็กซ์พอร์ต และอิมพอร์ตแบบ Weak สำหรับโมดูล.
3. COBOL ต้องใช้อ็อพชัน NOMONOPRC. ถ้าไม่ใช้อ็อพชันนี้อักษรตัวเล็กก็จะถูกแปลงเป็นอักษรตัวใหญ่โดยอัตโนมัติ.

สำหรับข้อมูลเกี่ยวกับการประกาศแบบใดที่เป็นการอิมพอร์ต หรือเอ็กซ์พอร์ตในแต่ละภาษา ให้ดูได้จากหนังสือเหล่านี้:

- WebSphere Development Studio: ILE RPG Programmer's Guide
- WebSphere Development Studio: ILE COBOL Programmer's Guide
- WebSphere Development Studio ILE C/C++ Programmer's Guide

ภาษา Binder

ภาษา Binder คือชุดคำสั่งขนาดเล็กที่ไม่สามารถรันได้ ซึ่งนิยามการเอ็กซ์พอร์ตสำหรับเซอร์วิสโปรแกรม. ภาษา Binder เปิดใช้งานการตรวจสอบไวยากรณ์ Source Entry Utility (SEU) เพื่อตรวจสอบอินพุตเมื่อประเภทซอร์ส BND ถูกระบุ.

หมายเหตุ: คุณไม่สามารถใช้ไวยากรณ์ SEU ในการตรวจสอบชนิดของ BND สำหรับซอร์สไฟล์ของ Binder ที่ใช้ wildcard ได้. คุณยังไม่สามารถใช้มันสำหรับซอร์สไฟล์ของ Binder ที่ชื่อยาวเกิน 254 ตัวอักษร.

ภาษา Binder ประกอบด้วยคำสั่งดังต่อไปนี้:

1. คำสั่ง Start Program Export (STRPGMEXP) ใช้บอกให้ทราบว่าเป็นจุดเริ่มต้นของรายการเอ็กซ์พอร์ตจากเซอร์วิสโปรแกรม
2. คำสั่ง Export Symbol (EXPORT) ใช้บอกให้ทราบว่าชื่อสัญลักษณ์ที่ถูกเอ็กซ์พอร์ตจากเซอร์วิสโปรแกรม.
3. คำสั่ง End Program Export (ENDPGMEXP) ใช้บอกให้ทราบว่าเป็นจุดสิ้นสุดของรายการเอ็กซ์พอร์ตจากเซอร์วิสโปรแกรม

รูปที่ 33 ในหน้า 91 เป็นตัวอย่างของภาษา Binder ในซอร์สไฟล์:

```
STRPGMEXP PGMLVL(*CURRENT) LVLCHK(*YES)
   .
   .
EXPORT SYMBOL(p1)
EXPORT SYMBOL('p2')
EXPORT SYMBOL('P3')
   .
   .
ENDPGMEXP
   .
   .
   .

STRPGMEXP  PGMLVL(*PRV)
   .
   .
EXPORT SYMBOL(p1)
EXPORT SYMBOL('p2')
   .
   .
ENDPGMEXP
```

รูปที่ 33. ตัวอย่างของภาษา Binder ในซอร์สไฟล์

คำสั่ง Retrieve Binder Source (RTVBNDSRC) สามารถช่วยสร้างซอร์สของภาษา Binder ตามเอ็กซ์พอร์ตจากโมดูล.

Signature

สัญลักษณ์ที่ใช้แยกแยะระหว่าง STRPGMEXP PGMLVL(*CURRENT) และ ENDPGMEXP ในการนิยามอินเตอร์เฟสพับลิกให้กับเซอร์วิสโปรแกรม. ซึ่งอินเตอร์เฟสพับลิกนั้นแสดงด้วย **Signature**. ซึ่ง Signature เป็นค่าที่ใช้แยกอินเตอร์เฟสที่สนับสนุนโดยเซอร์วิสโปรแกรม.

หมายเหตุ: อย่าสับสนกับ signatures ที่อธิบายในหัวข้อนี้กับ *Digital Object Signatures*. . ซึ่ง Digital signatures ของอ็อบเจ็กต์บน OS/400 ใช้สำหรับ Integrity ของซอฟต์แวร์และข้อมูล. และยังใช้เป็นเครื่องมือขัดขวางการเจาะข้อมูล, ไวรัสพื้นฐาน, หรือการเปลี่ยนแปลงอ็อบเจ็กต์ที่ไม่ได้รับอนุญาต. Signature ยังช่วยให้ทราบถึงจุดเริ่มต้นของโปรแกรมอีกด้วย. สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับ Digital Object Signatures ให้ดูที่หมวด **Security** ใน iSeries Information Center.

ถ้าคุณเลือกไม่ระบุ signature ด้วยตัวเอง Binder ก็จะสร้าง signature ขึ้นจากรายชื่อโพรซีเดอร์ และข้อมูลที่ถูกเอ็กซ์พอร์ต และตามลำดับที่ระบุไว้. ดังนั้น signature จึงเป็นวิธีที่ง่าย และสะดวกในการตรวจสอบอินเตอร์เฟสพับลิกของเซอร์วิสโปรแกรม . Signature ไม่สามารถตรวจสอบอินเตอร์เฟสเฉพาะโพรซีเดอร์ในเซอร์วิสโปรแกรมได้.

หมายเหตุ: เพื่อหลีกเลี่ยงการเปลี่ยนแปลงที่เข้ากันไม่ได้กับเซอร์วิสโปรแกรม โพรซีเดอร์ที่มีอยู่และชื่อของตัวข้อมูลต้องถูกลบหรือจัดเรียงใหม่ในซอร์สของภาษา Binder. นอกจากนี้

บล็อกเอ็กซ์พอร์ตต้องมีสัญลักษณ์ที่มีชื่อ และการเรียงลำดับเหมือนกันกับบล็อกเอ็กซ์พอร์ตที่มีอยู่แล้ว. แล้ว อีกทั้งสัญลักษณ์เหล่านี้ต้องเพิ่มเข้าไปที่ส่วนท้ายของรายการเท่านั้น.

ไม่มีวิธีใดที่จะสามารถลบเอ็กซ์พอร์ตของเซอร์วิสโปรแกรมให้เข้ากันได้กับโปรแกรมและเซอร์วิสโปรแกรมที่มีอยู่แล้วได้ เนื่องจากเอ็กซ์พอร์ตนั้นอาจจำเป็นสำหรับโปรแกรมหรือเซอร์วิสโปรแกรมที่รวมกับเซอร์วิสโปรแกรมก็ได้.

ถ้าการเปลี่ยนแปลงที่ทำกับเซอร์วิสโปรแกรมไม่สามารถเข้ากันได้โปรแกรมเก่าที่รวมกับโปรแกรมเซอร์วิสอาจทำงานผิดพลาดได้. การเปลี่ยนแปลงในลักษณะนี้กับเซอร์วิสโปรแกรมสามารถทำได้เฉพาะในกรณีที่คุณมั่นใจว่าโปรแกรมและเซอร์วิสโปรแกรมทั้งหมดที่รวมอยู่กับมันถูกสร้างใหม่ด้วยคำสั่ง**CRTPGM** หรือ **CRTSRVPGM** หลังจากที่ได้ทำการเปลี่ยนแปลงเรียบร้อยแล้ว.

คำสั่ง Start Program Export และ End Program Export

คำสั่ง Start Program Export (STRPGMEXP) ใช้บอกให้ทราบว่าเป็นจุดเริ่มต้นของรายการเอ็กซ์พอร์ตจากเซอร์วิสโปรแกรม. ส่วนคำสั่ง End Program Export (ENDPGMEXP) ใช้บอกให้ทราบว่าเป็นจุดสิ้นสุดของรายการเอ็กซ์พอร์ตจากจากเซอร์วิสโปรแกรม.

คำสั่ง STRPGMEXP และ ENDPGMEXP หลายๆ คู่ที่ระบุไว้ในซอร์สไฟล์จะทำให้เกิด Signature หลายอันด้วย. ลำดับของคำสั่ง STRPGMEXP และ ENDPGMEXP ไม่มีความสำคัญใดๆ.

พารามิเตอร์ Program Level ในคำสั่ง STRPGMEXP

มีเพียงคำสั่ง STRPGMEXP คำสั่งเดียวเท่านั้นที่สามารถระบุ PGMLVL(*CURRENT) ได้ แต่มันไม่จำเป็นต้องเป็นคำสั่ง STRPGMEXP คำสั่งแรก. ส่วนคำสั่ง STRPGMEXP อื่นๆ ในซอร์สไฟล์ต้องระบุ PGMLVL(*PRV). Signature ปัจจุบันจะแสดงโดยคำสั่ง STRPGMEXP ที่มี PGMLVL (*CURRENT) ระบุไว้.

พารามิเตอร์ Level Check ในคำสั่ง STRPGMEXP

พารามิเตอร์ Level Check (LVLCHK) ที่ระบุในคำสั่ง STRPGMEXP จะบอกให้ Binder ทำการตรวจสอบอินเตอร์เฟสพับลิกของเซอร์วิสโปรแกรมโดยอัตโนมัติ. ซึ่งการระบุ LVLCHK(*YES) หรือปล่อยให้ค่าดีฟอลต์เป็น LVLCHK(*YES) จะทำให้ Binder ตรวจสอบพารามิเตอร์ Signature. โดยพารามิเตอร์ Signature ทำให้ทราบว่า Binder ใช้ค่า Signature แบบกำหนดเอง หรือสร้างค่า Signature ที่ไม่เท่ากับศูนย์ขึ้น. ถ้า Binder สร้างค่า Signature ขึ้น ระบบจะตรวจดูว่าค่านั้นตรงกันกับค่าที่พบในโปรแกรมไคลเอนต์ของเซอร์วิสโปรแกรมหรือไม่. ถ้าค่านั้นตรง โปรแกรมไคลเอนต์ของเซอร์วิสโปรแกรมสามารถใช้อินเตอร์เฟสพับลิกได้โดยไม่ต้องทำการคอมไพล์ใหม่.

การระบุ LVLCHK(*NO) จะปิดการตวจสอบ Signature แบบอัตโนมัติ. อัตโนมัติ คุณอาจจำเป็นต้องใช้คุณลักษณะนี้ หากเกิดเหตุการณ์เหล่านี้ขึ้น:

- คุณทราบถึงการเปลี่ยนแปลงในอินเตอร์เฟสของเซอร์วิสโปรแกรมว่าจะไม่ก่อให้เกิดความไม่เข้ากัน.

- คุณต้องการหลีกเลี่ยงการอัพเดตซอร์สไฟล์ของภาษา Binder หรือการคอทไพล์โปรแกรมไคลเอนต์ใหม่.

คุณควรใช้ค่า LVLCHK(*NO) ด้วยความระมัดระวัง เนื่องจากมันอาจหมายความว่า คุณต้องรับผิดชอบการตรวจสอบอินเตอร์เฟสพับลิกว่าเข้ากันได้กับเลเวลเก่าหรือไม่ด้วยตัวของคุณเอง. ให้ระบุ LVLCHK(*NO) ก็ต่อเมื่อคุณสามารถควบคุมได้ว่าโพรซีเดอร์ใดในเซอร์วิสโปรแกรมที่ถูกเรียกและตัวแปรใดบ้างที่ถูกใช้โดยโปรแกรมไคลเอนต์. ไคลเอนต์ ถ้าคุณไม่สามารถควบคุมอินเทอร์เฟสพับลิก, รันไทม์ หรือ Activation ความผิดพลาดก็อาจเกิดขึ้นได้. โปรดดูในหัวข้อ “ข้อผิดพลาดของ Binder Language” ในหน้า 202 สำหรับคำอธิบายเกี่ยวกับตวามผิดพลาดทั่วไปที่อาจเกิดขึ้นได้จากการใช้ภาษา Binder.

พารามิเตอร์ Signature ในคำสั่ง STRPGMEXP

พารามิเตอร์ Signature (SIGNATURE) ช่วยให้คุณสามารถระบุ Signature สำหรับเซอร์วิสโปรแกรมได้ด้วยตัวเอง. ค่าที่ระบุด้วยตัวเองนี้สามารถเป็นเป็นสตริงของเลขฐานสิบหก หรือสตริงของอักขระก็ได้. ได้ คุณอาจจำเป็นต้องเลือกใช้วิธีการระบุด้วยตัวเองด้วยสาเหตุต่อไปนี้:

- Binder สร้าง Signature จากความเข้ากันได้ แต่คุณไม่ต้องการ. ค่า Signature จะสร้างจากชื่อ และลำดับของเอ็กซ์พอร์ตที่ระบุ. ดังนั้นถ้าบล็อกเอ็กซ์พอร์ต 2 บล็อกทีมีชื่อและลำดับของเอ็กซ์พอร์ตเหมือนกัน พวกมันก็จะมี Signature เหมือนกัน. ในฐานะของผู้จัดเตรียมเซอร์วิสโปรแกรม คุณอาจทราบว่าอินตอร์เฟสทั้งสองไม่เข้ากัน (เนื่องจากพารามิเตอร์ของมันอาจแตกต่างกัน).). ซึ่งในกรณีนี้คุณก็สามารถระบุ Signature ใหม่แทนค่าที่ Binder สร้างขึ้นจากความเข้ากันได้. ถ้าคุณทำเช่นนั้น คุณก็จะทำให้เกิดความไม่เข้ากันในเซอร์วิสโปรแกรม ซึ่งทำให้ต้องคอมไพล์ปรแกรมไคลเอนต์บางส่วน หรือทั้งหมดใหม่.
- Binder สร้าง Signature จากความเข้ากันไม่ได้ แต่คุณไม่ต้องการ. ต้องการ ดังนั้นถ้าบล็อกเอ็กซ์พอร์ต 2 บล็อกทีมีชื่อและลำดับของเอ็กซ์พอร์ตแตกต่างกัน. พวกมันก็จะมี Signature แตกต่างกันด้วย ในฐานะของผู้จัดเตรียมเซอร์วิสโปรแกรม คุณอาจทราบว่าอินตอร์เฟสทั้งสองเข้ากันได้ (เนื่องจากชื่อของฟังก์ชันอาจเปลี่ยนแปลงไปแต่ก็ยังคงฟังก์ชันดียวกัน) คุณสามารถกำหนดให้ใช้ Signature เหมือนกันแทน Signature ที่สร้างจากความเข้ากันไม่ได้ ของ Binder. ถ้าคุณระบุให้ใช้ Signature เดียวกัน คุณก็ต้องคอยระวังความเข้ากันได้ในเซอร์วิสโปรแกรม เพื่อช่วยให้โปรแกรมไคลเอนต์ของคุณที่ใช้เซอร์วิสโปรแกรมไม่จำเป็นต้องทำการรวมใหม่.

*GEN เป็นค่าดีฟอลต์สำหรับพารามิเตอร์ Signature ที่บอกให้ Binder สร้าง Signature ขึ้นจากสัญลักษณ์ที่อิมพอร์ต.

คุณสามารถตรวจดูค่า Signature ของเซอร์วิสโปรแกรมได้โดยใช้คำสั่ง Display Service Program (DSPSRVPGM) แล้วให้ระบุ DETAIL(*SIGNATURE).

คำสั่ง Export Symbol

คำสั่ง Export Symbol (EXPORT) ใช้แสดงชื่อสัญลักษณ์ที่สามารถเอ็กซ์พอร์ตจากจากเซอร์วิสโปรแกรม.

ถ้าสัญลักษณ์ที่เอ็กซ์พอร์ตมีอักษรแบบตัวพิมพ์เล็กอยู่ด้วย ชื่อสัญลักษณ์นั้นก็ควรใส่เครื่องหมาย Apostrophe ล้อมไว้ ดังในรูป รูปที่ 33 ในหน้า 91. แต่ถ้าไม่สามารถใช้เครื่องหมาย Apostrophe ได้ ชื่อสัญลักษณ์นั้นก็จะถูกแปลงให้เป็นตัวพิมพ์ใหญ่. ในตัวอย่าง Binder จะค้นหาเอ็กซ์พอร์ตที่ชื่อ P1 ไม่ใช่ p1.

ชื่อสัญลักษณ์สามารถทำการเอ็กซ์พอร์ตโดยใช้อักขระตัวแทน (Wildcard Character) (<<< or >>>). ถ้าชื่อสัญลักษณ์นั้นมีอยู่ก่อนแล้วและตรงกันกับอักขระตัวแทนที่ระบุ ชื่อสัญลักษณ์ก็จะถูกเอ็กซ์พอร์ต. แต่ถ้าหากเงื่อนไขข้อใดข้อหนึ่งด้านล่างนี้เป็นจริง สัญญาณความผิดพลาดก็จะปรากฏขึ้น และเซอร์วิสโปรแกรมก็จะไม่ถูกสร้าง:

- ไม่มีชื่อสัญลักษณ์ที่ตรงกันกับอักขระตัวแทนที่ระบุ
- มีชื่อสัญลักษณ์ที่ตรงกันกับอักขระตัวแทนที่ระบุมากกว่าหนึ่งตัว
- ชื่อสัญลักษณ์ที่ตรงกันกับอักขระตัวแทนที่ระบุ แต่มันไม่สามารถเอ็กซ์พอร์ตได้

ซับสตริงที่ระบุอยู่ในอักขระตัวแทนต้องล้อมรอบด้วยเครื่องหมายคำพูด.

Signature จะถูกตรวจสอบจากอักขระที่ระบุอยู่ในอักขระตัวแทน. ซึ่งการเปลี่ยนแปลงอักขระตัวแทนจะทำให้ Signature เปลี่ยนไปด้วย ถ้าอักขระตัวแทนที่เปลี่ยนแปลงตรงกันกับเอ็กซ์พอร์ตเดียวกัน. . ตัวอย่างเช่น อักขระตัวแทน 2 แบบคือ “r”>>> และ “ra”>>> จะทำการเอ็กซ์พอร์ตสัญลักษณ์ “rate” แต่มันจะสร้าง Signature ที่ไม่เหมือนกัน. ดังนั้นเราขอแนะนำให้คุณใช้อักขระตัวแทนที่คล้ายกับสัญลักษณ์ที่เอ็กซ์พอร์ตให้มากที่สุด.

หมายเหตุ: คุณไม่สามารถใช้ไวยากรณ์ SEU ตรวจสอบ Type แบบ BND ของซอร์สไฟล์ Binder ที่มีอักขระตัวแทนได้.

ตัวอย่างการใช้ Wildcard Export Symbol

สำหรับตัวอย่างต่อไปนี้ สมมุติว่ารายการสัญลักษณ์ของเอ็กซ์พอร์ตประกอบด้วย:

interest_rate
international
prime_rate

ตัวอย่างต่อไปนี้จะแสดงว่าเอ็กซ์พอร์ตใดที่ถูกเลือก หรือทำไมจึงเกิดความผิดพลาด:

EXPORT SYMBOL (“interest”>>>)

ทำการเอ็กซ์พอร์ตสัญลักษณ์ “interest_rate” เนื่องจากมีเพียงสัญลักษณ์เดียวเท่านั้นที่ขึ้นต้นด้วย “interest”.

EXPORT SYMBOL (“i”>>>“rate”>>>)

ทำการเอ็กซ์พอร์ตสัญลักษณ์ “interest_rate” เนื่องจากมีเพียงสัญลักษณ์เดียวเท่านั้นที่ขึ้นต้นด้วย “i” และตามหลังด้วย “rate”.

EXPORT SYMBOL (<<<“i”>>>“rate”)

แสดงข้อความผิดพลาด “Multiple matches for wildcard specification”. เนื่องจากทั้ง “prime_rate” และ “interest_rate” ต่างก็มี “i” และต่อท้ายด้วย “rate” เหมือนกัน.

EXPORT SYMBOL ("inter">>>"prime")

แสดงข้อความผิดพลาด "No matches for wildcard specification". เนื่องจากไม่มีสัญลักษณ์ตัวใดขึ้นต้นด้วย "inter" และลงท้ายด้วย "prime".

EXPORT SYMBOL (<<<)

แสดงข้อความผิดพลาด "Multiple matches for wildcard specification". เนื่องจากสัญลักษณ์นี้ตรงกับสัญลักษณ์อื่นถึงสามตัว ซึ่งเป็นการผิดเงื่อนไข. การเอ็กซ์พอร์ตคำสั่งสามารถทำได้เพียงสัญลักษณ์เดียวเท่านั้น.

ตัวอย่างภาษา Binder

ตัวอย่างของการใช้ภาษา Binder สมมุติให้คุณกำลังพัฒนาแอ็พพลิเคชันทางการเงินอย่างง่าย ซึ่งประกอบด้วยโพรซีเดอร์ดังต่อไปนี้:

- โพรซีเดอร์ Rate

  คำนวณหา Interest_Rate โดยมีอินพุตเป็น Loan_Amount, Term_of_Payment, และ Payment_Amount.
- โพรซีเดอร์ Amount

  คำนวณหา Loan_Amount โดยมีอินพุตเป็น Interest_Rate, Term_of_Payment, และ Payment_Amount.
- โพรซีเดอร์ Payment

  คำนวณหา Payment_Amount โดยมีอินพุตเป็น Interest_Rate, Term_of_Payment, และ Loan_Amount.
- โพรซีเดอร์ Term

  คำนวณหา Term_of_Payment โดยมีอินพุตเป็น Interest_Rate, Loan_Amount, และ Payment_Amount.

เอาต์พุตบางส่วนของแอ็พพลิเคชันนี้แสดงไว้ใน ภาคผนวก A, "Output Listing จากคำสั่ง CRTPGM, CRTSRVPGM, UPDPGM, หรือ UPDSRVPGM", ในหน้า 191.

ในตัวอย่างภาษา Binder แต่ละโมดูลจะมีมากกว่าหนึ่งโพรซีเดอร์. ตัวอย่างต่างๆ สามารถประยุกต์ได้แม้ว่าเป็นโมดูลที่มีเพียงหนึ่งโพรซีเดอร์.

ตัวอย่างการใช้ภาษา Binder ชุดที่ 1

ภาษา Binder สำหรับโพรซีเดอร์ Rate, Amount, Payment, และ Term จะเหมือนในตัวอย่างข้างล่างนี้:

```
FILE: MYLIB/QSRVSRC  MEMBER: FINANCIAL

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
ENDPGMEXP
```

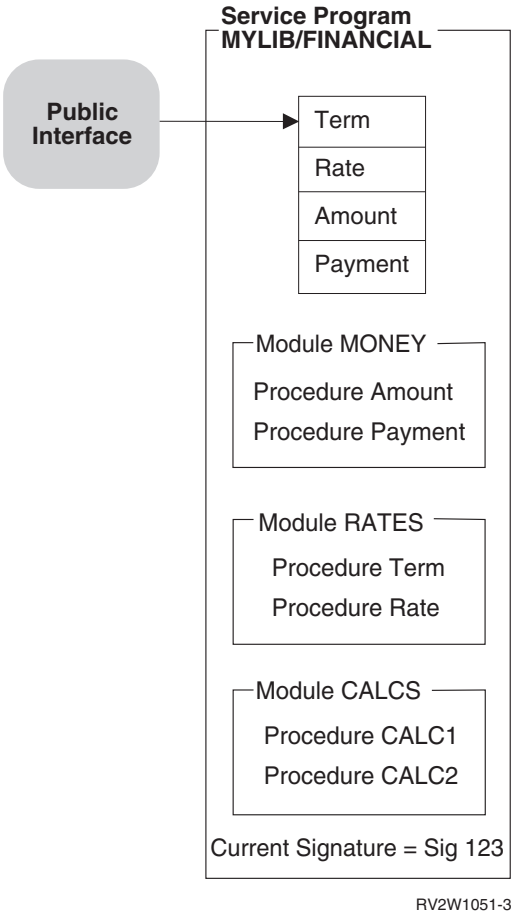
จากนั้นเริ่มต้นออกแบบขั้นตอนการเลือก และกำหนดโมดูล 3 โมดูล (MONEY, RATES, และ CALCS) จัดเตรียมไว้สำหรับโพรซีเดอร์.

ในการสร้างเซอร์วิสโปรแกรมดังใน รูปที่ 34ภาษา Binder จะถูกระบุในคำสั่ง CRTSRVPGM ดังนี้:

```
CRTSRVPGM  SRVPGM(MYLIB/FINANCIAL)
           MODULE(MYLIB/MONEY MYLIB/RATES MYLIB/CALCS)
           EXPORT(*SRCFILE)
           SRCFILE(MYLIB/QSRVSRC)
           SRCMBR(*SRVPGM)
```

โปรดสังเกตว่า ซอร์สไฟล์ QSRVSRC อยู่ในไลบรารี MYLIB โดยระบุอยู่ในพารามิเตอร์ SRCFILE ซึ่งเป็นไฟล์ที่มีซอร์สของภาษา Binder อยู่.

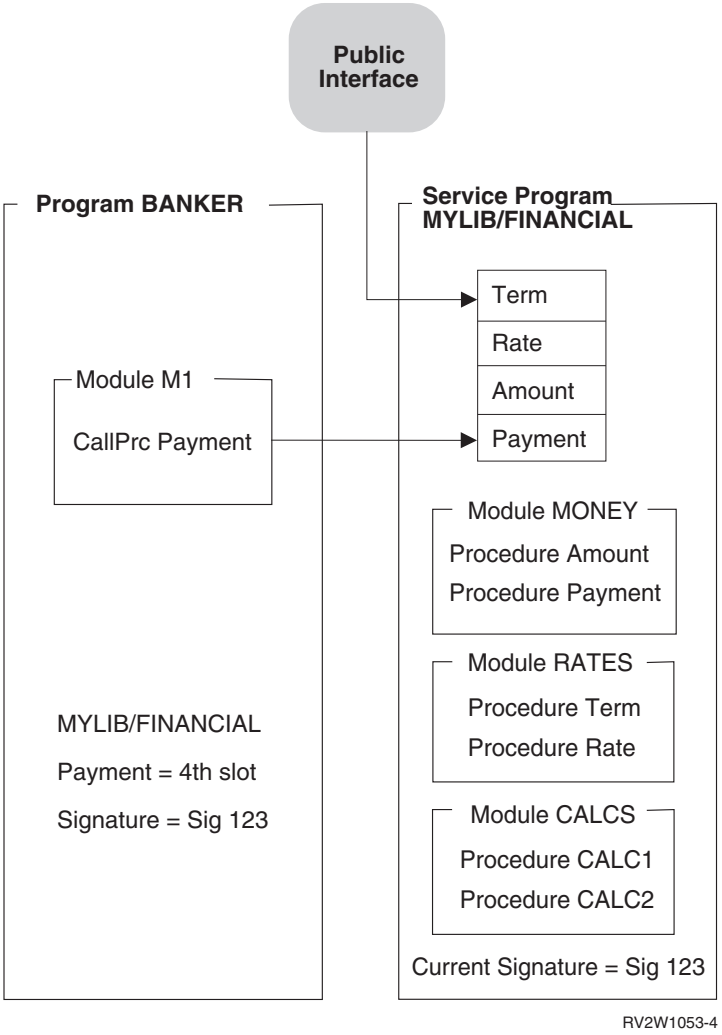
และโปรดสังเกตอีกว่า ไม่มีการกำหนด Binding Directory ไว้เลย เนื่องจากโมดูลทั้งหมดที่จำเป็นต่อการเซอร์วิสโปรแกรมถูกระบุไว้ในพารามิเตอร์ MODULE หมดแล้ว.



รูปที่ 34. Creating a Service Program by Using the Binder Language

ตัวอย่างการใช้ภาษา Binder ชุดที่ 2

เมื่อการพัฒนาแอ็พพลิเคชันผ่านไปได้ระยะหนึ่งก็มีโปรแกรมที่ชื่อ BANKER ถูกเขียนขึ้น. ซึ่งโปรแกรม BANKER จำเป็นต้องเรียกใช้โพรซีเดอร์ Payment ที่อยู่ในเซอร์วิสโปรแกรมชื่อ FINANCIAL. . ผลลัพธ์ของแอ็พพลิเคชันพร้อมโปรแกรม BANKER แสดงไว้ใน รูปที่ 35.



รูปที่ 35. การใช้เซอร์วิสโปรแกรม FINANCIAL

เมื่อโปรแกรม BANKER ถูกสร้าง เซอร์วิสโปรแกรม MYLIB/FINANCIAL ถูกจัดเตรียมโดยพารามิเตอร์ BNDSRVPGM. สัญลักษณ์ Payment ถูกตรวจพบว่าโดนเอ็กซ์พอร์ตมาจากสล็อตที่สี่ของอินเตอร์เฟสพับลิกของ เซอร์วิสโปรแกรม FINANCIAL. และ Signature ปัจจุบันของ MYLIB/FINANCIAL ตามด้วยสล็อตที่สัมพันธ์กับอินเตอร์เฟส Payment ถูกบันทึกไว้ด้วยโปรแกรม BANKER.

ในระหว่างกระบวนการเตรียม BANKER ให้พร้อมใช้งาน Activation จะทำการตรวจสอบดังต่อไปนี้:

- ตรวจหาเซอร์วิสโปรแกรม FINANCIAL ในไลบรารี MYLIB.

- เซอร์วิสโปรแกรมยังคงสนับสนุน Signature (SIG 123) ที่บันทึกในโปรแกรม BANKER.

การตรวจสอบ Signature นี้จะทำการตรวจสอบอินเตอร์เฟสพับลิกที่ใช้โดยโปรแกรม BANKER เมื่อถูกสร้างขึ้นว่าสามารถใช้งานได้ในช่วงรันไทม์หรือไม่.

ดังในรูป รูปที่ 35 ในหน้า 97เมื่อโปรแกม BANKER ถูกเรียก MYLIB/FINANCIAL จะยังคงสนับสนุนอินเตอร์เฟสพับลิกที่ใช้โดยโปรแกรม BANKER. ถ้า Activation ไม่พบ Signature ที่ตรงกันใน MYLIB/FINANCIAL หรือเซอร์วิสโปรแกรม MYLIB/FINANCIAL เหตุการณ์เหล่านี้จะเกิดขึ้น:

โปรแกรม BANKER จะล้มเหลวในการเรียกใช้งาน
ข้อความแสดงความผิดพลาดจะปรากฏขึ้น

ตัวอย่างการใช้ภาษา Binder ชุดที่ 3

เมื่อแอ็พพลิเคชันเริ่มขยายขนาดขึ้นเรื่อยๆ ก็ต้องการเพิ่มโพรซีเดอร์ขึ้นอีก 2 โพรซีเดอร์เพิ่มความสามารถในการทำงาน. โพรซีเดอร์ทั้งสองคือ OpenAccount และ CloseAccount, ซึ่งทำหน้าที่เปิดและปิดบัญชีตามลำดับ. ขั้นตอนต่อไปนี้ป็ขั้นตอนที่จำเป็นสำหรับการอัพเดต MYLIB/FINANCIAL โดยที่ไม่จำเป็นต้องไปอัพเดตโปรแกรม BANKER:

1. เขียนโพรซีเดอร์ OpenAccount และ CloseAccount.
2. อัพเดตภาษา Binder เพื่อระบุโพรซีเดอร์ใหม่.

ภาษา Binder ที่อัพเดตสนับสนุนการทำงานของโพรซีเดอร์ใหม่. มันยังช่วยให้โปรแกรม ILE ที่มีอยู่หรือเซอร์วิสโปรแกรมที่ใช้เซอร์วิสโปรแกรม FINANCIAL ไม่จำเป็นต้องเปลี่ยนแปลงใดๆ. ภาษา Binder ที่อัพเดตจะเหมือนกับที่แสดงไว้ข้างล่างนี้:

```
FILE: MYLIB/QSRVSRC  MEMBER: FINANCIAL

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
  EXPORT SYMBOL('OpenAccount')
  EXPORT SYMBOL('CloseAccount')
ENDPGMEXP

STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
ENDPGMEXP
```

เมื่อทำการอัพเดตเซอร์วิสโปรแกรม คุณต้องคำนึงถึงสิ่งต่างๆ เหล่านี้:

- สนับสนุนโพรซีเดอร์หรือตัวข้อมูลใหม่
- อนุญาตให้โปรแกรมที่มีอยู่แล้วและเซอร์วิสโปรแกรมสามารรถใช้เซอร์วิสโปรแกรมที่ทำการอัพเดตได้โดยไม่ต้องทำการเปลี่ยนแปลงใดๆ

คุณต้องเลือกใช้วิธีใดวิธีหนึ่งจากหัวข้อทั้งสองนี้. ถ้าคุณเลือกใช้วิธีการในหัวข้อแรก คุณก็ต้องทำดังนี้:

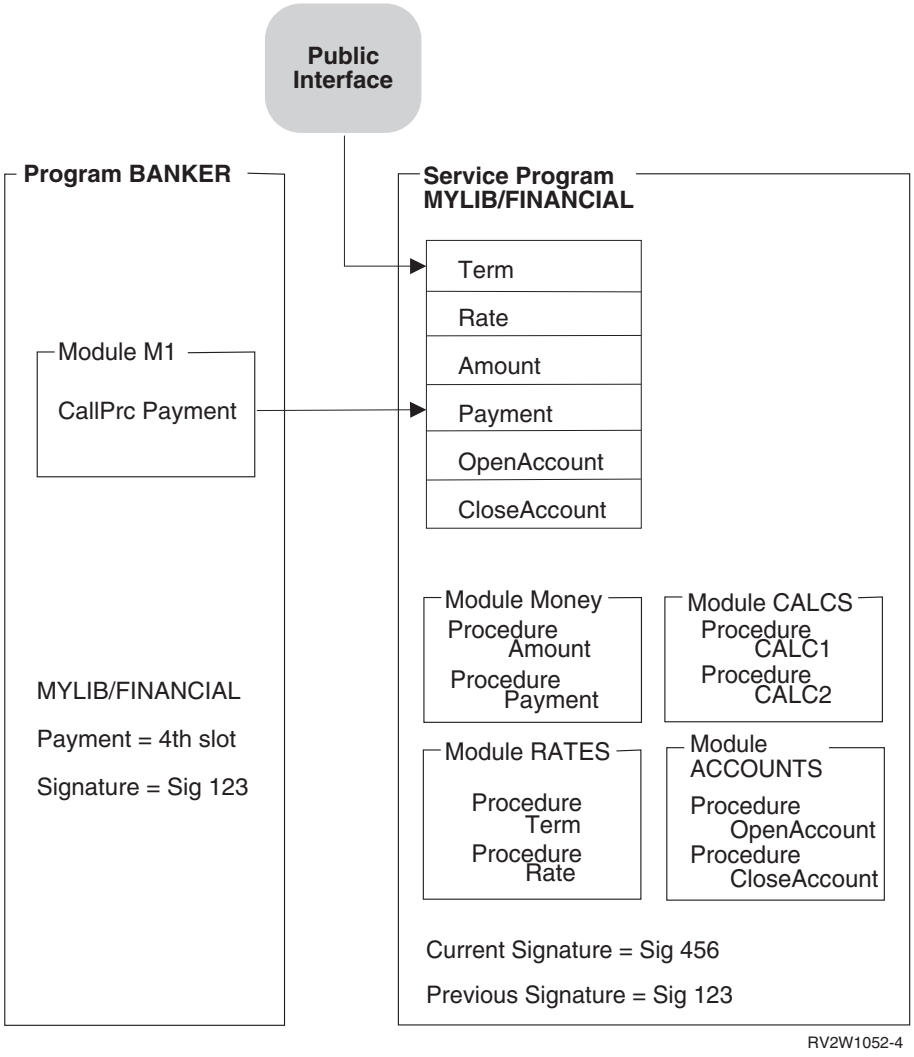
1. ทำสำเนาบล็อก STRPGMEXP, ENDPGMEXP ที่มีคำสั่ง PGMLVL(*CURRENT) อยู่.
2. เปลี่ยนคำสั่ง PGMLVL(*CURRENT) ที่อยู่ในสำเนาให้เป็น PGMLVL(*PRV).
3. ในคำสั่ง STRPGMEXP ที่มี PGMLVL(*CURRENT) ให้เพิ่มโพรซีเดอร์หรือตัวข้อมูลใหม่ที่เอ็กซ์พอร์ตเข้าไปที่ส่วนท้ายของรายการ.
4. บันทึกการเปลี่ยนแปลงลงในซอร์สไฟล์.
5. สร้างโมดูลใหม่ หรือเปลี่ยนแปลงโมดูลที่มีอยู่.
6. สร้างเซอร์วิสโปรแกรมจากโมดูลใหม่หือโมดูลที่เปลี่ยนแปลง โดยการใช้ภาษา Binder ที่อัพเดต.

สำหรับวิธีการที่สองเป็นการใช้ประโยชน์จากพารามิเตอร์ Signature ในคำสั่ง STRPGMEXP และการเพิ่มสัญลักษณ์ใหม่ลงที่ส่วนท้ายของบล็อกที่เอ็กซ์พอร์ต:

```
STRPGMEXP PGMVAL(*CURRENT) SIGNATURE('123')
  EXPORT SYMBOL('Term')
    .
    .
    .
  EXPORT SYMBOL('OpenAccount')
  EXPORT SYMBOL('CloseAccount')
ENDPGMEXP
```

ในการสร้างเซอร์วิสโปรแกรมที่สมบูรณ์อย่างในรูป รูปที่ 36 ในหน้า 100 ภาษา Binder ที่อัพเดตซึ่งระบุอยู่ในหน้าที่ 98 จะถูกนำมาใช้กับคำสั่ง CRTSRVPGM:

```
 CRTSRVPGM  SRVPGM(MYLIB/FINANCIAL)
            MODULE(MYLIB/MONEY MYLIB/RATES MYLIB/CALCS MYLIB/ACCOUNTS))
            EXPORT(*SRCFILE)
            SRCFILE(MYLIB/QSRVSRC)
            SRCMBR(*SRVPGM)
```



รูปที่ 36. Updating a Service Program by Using the Binder Language

โปรแกรม BANKER ไม่มีการเปลี่ยนแปลงใดๆ เนื่องจาก Signature ก่อนหน้ายังคงได้รับการสนับสนุน. (ดูในSignature ก่อนหน้าในเซอร์วิสโปรแกรม MYLIB/FINANCIAL และ Signature ที่บันทึกไว้ใน BANKER).) ถ้าโปรแกรม BANKER ถูกสร้างขึ้นใหม่โดยคำสั่ง CRTPGM ค่า Signature ที่ถูกบันทึกไว้ในโปรแกรม BANKER ก็จะเป็น Signature ปัจจุบันของเซอร์วิสโปรแกรม FINANCIAL. สาเหตุเดียวที่จะทำให้ต้องสร้างโปรแกรม BANKER ก็คือโปรแกรมนั้นใช้โพรซีเดอร์ใหม่ที่อยู่ในเซอร์วิสโปรแกรม FINANCIAL. ภาษา Binder ช่วยให้คุณสามารถเพิ่มคุณลักษณะของ เซอร์วิสโปรแกรมได้โดยไม่ต้องเปลี่ยนแปลงโปรแกรมหรือเซอร์วิสโปรแกรมที่เรียกใช้งานเซอร์วิส.

ตัวอย่างการใช้ภาษา Binder ชุดที่ 4

หลังจากการส่งมอบเซอร์วิสโปรแกรม FINANCIAL ที่อัพเดตไปแล้วนั้น คุณก็ได้รับคำร้องขอให้สร้างอัตราดอกเบี้ยตามเงื่อนไขดังต่อไปนี้:

พารามิเตอร์ปัจจุบันของโพรซีเดอร์ Rate
ประวัติด้านเครดิตของลูกค้า

พารามิเตอร์ตัวที่ห้าที่ชื่อ Credit_History ต้องถูกเพิ่มในการเรียกไปยังโพรซีเดอร์ Rate. โดย Credit_History จะทำการอัพเดตพารามิเตอร์ Interest_Rate ที่ได้รับการคืนค่ามาจากโพรซีเดอร์ Rate. ข้อจำกัดอีกประการหนึ่งก็คือโปรแกรม ILE ที่มีอยู่หรือเซอร์วิสโปรแกรมที่ใช้เซอร์วิสโปรแกรม FINANCIAL ต้องไม่มีการเปลี่ยนแปลงใดๆ. ถ้าภาษาไม่สนับสนุนการส่งค่าพารามิเตอร์หลายๆ ตัว มันก็เป็นเรื่องยากที่จะทำตามเงื่อนไขเหล่านี้:

- อัพเดตเซอร์วิสโปรแกรม
- หลีกเลี่ยงการสร้างอ็อบเจ็กต์ทั้งหมดที่ใช้เซอร์วิสโปรแกรม FINANCIAL ใหม่.

แต่โชคดีที่ยังพอมีทางทำได้. โดยภาษา Binder ต่อไปนี้สนับสนุนการอัพเดตโพรซีเดอร์ Rate. ซึ่งมันทำให้โปรแกรม ILE ที่มีอยู่หรือเซอร์วิสโปรแกรมที่ใช้ซอร์วิสโปรแกรม FINANCIAL ไม่ต้องมีการเปลี่ยนแปลงใดๆ.

```
FILE: MYLIB/QSRVSRC  MEMBER: FINANCIAL

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Old_Rate')  /* Original Rate procedure with four parameters */
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
  EXPORT SYMBOL('OpenAccount')
  EXPORT SYMBOL('CloseAccount')
  EXPORT SYMBOL('Rate')           /* New Rate procedure that supports +
                                     a fifth parameter, Credit_History */
ENDPGMEXP

STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
  EXPORT SYMBOL('OpenAccount')
  EXPORT SYMBOL('CloseAccount')
ENDPGMEXP

STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
ENDPGMEXP
```

สัญลักษณ์ Rate เดิมถูกเปลี่ยนชื่อเป็น Old_Rate แต่ยังคงตำแหน่งที่สัมพันธ์กับสัญลักษณ์ที่ถูกเอ็กซ์พอร์ตไป. ซึ่งนี้เป็นสิ่งสำคัญที่คุณต้องจำไว้.

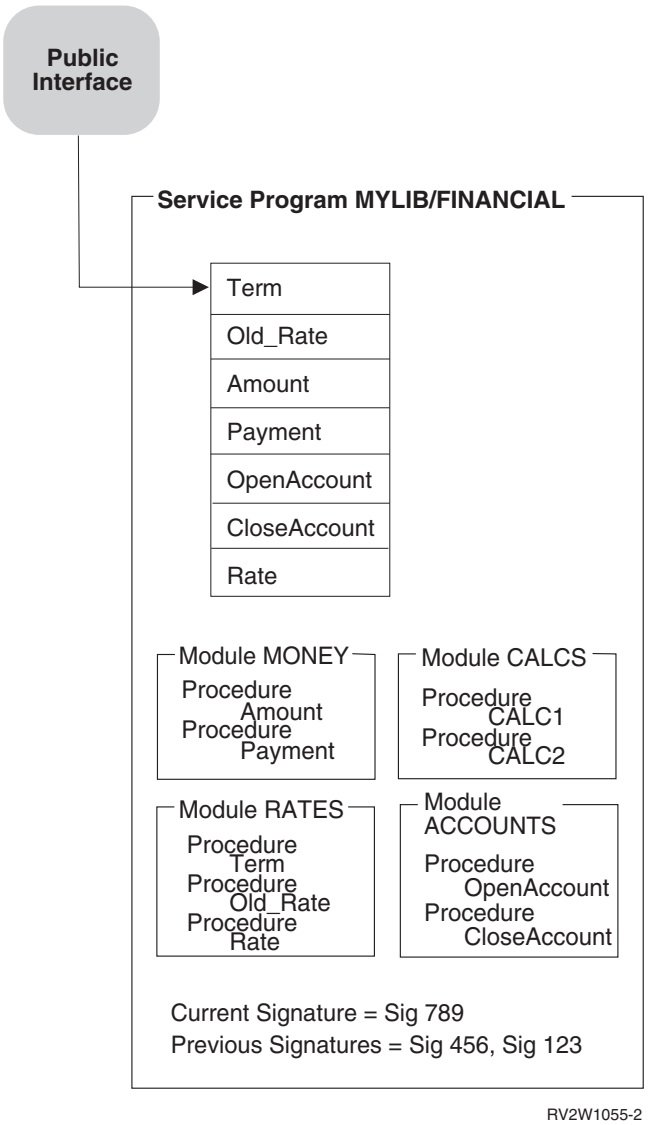
Comment ที่เกี่ยวข้องกับสัญลักษณ์ Old_Rate ก็จะปรากฏให้เห็น. Comment คือข้อความที่ปรากฏอยู่ระหว่างเครื่องหมาย `/*` และ `*/`. โดย Binder จะไม่สนใจ Comment ต่างๆ ที่อยู่ใน ซอร์สของภาษา Binder เมื่อคุณทำการสร้างเซอร์วิสโปรแกรม.

โพรซีเดอร์ Rate ใหม่สนับสนุนการใช้พารามิเตอร์ Credit_History ต้องถูกเอ็กซ์พอร์ต. โพรซีเดอร์ที่อัพเดตนี้จะถูกเพิ่มเข้าไปที่ส่วนท้ายของราการเอ็กซ์พอร์ต.

วิธีการต่อไปนี้สามารถนำไปใช้จัดการกับโพรซีเดอร์ Rate เดิมได้:

- เปลี่ยนชื่อโพรซีเดอร์ Rate เดิมที่สนับสนุนพารามิเตอร์สี่ตัวให้เป็น Old_Rate. และทำสำเนาโพรซีเดอร์ Old_Rate (ตั้งชื่อเป็น Rate). ทำการอัพเดตโค้ดให้สนับสนุนพารามิเตอร์ Credit_History.
- อัพเดตโพรซีเดอร์ Rate เดิมให้สนับสนุนพารามิเตอร์ Credit_History. และสร้างโพรซีเดอร์ใหม่ชื่อ Old_Rate ขึ้น. โดยให้ Old_Rate สนับสนุนการใช้พารามิเตอร์สี่ตัวของ Rate. มันยังสามารถเรียกไปยังโพรซีเดอร์ Rate ที่อัพเดตได้โดยให้พารามิเตอร์ตัวที่ห้าเป็น Dummy.
  นี่เป็นวิธีที่นิยมใช้กัน เนื่องจากการบำรุงรักษาง่ายกว่า และขนาดอ็อบเจ็กต์ก็เล็กกว่าด้วย.

การใช้ภาษา Binder ที่อัพเดต และโมดูล RATES ตัวใหม่ที่สนับสนุนโพรซีเดอร์ Rate, Term, และ Old_Rate คุณก็สามารถสร้างเซอร์วิสโปรแกรม FINANCIAL ได้:



รูปที่ 37. Updating a Service Program by Using the Binder Language

โปรแกรม ILE และเซอร์วิสโปรแกรมที่ใช้โพรซีเดอร์ Rate ต้นฉบับของ FINANCIAL เซอร์วิสโปรแกรมจะไปยังสล็อตที่ 2 ซึ่งเป็นการเรียกโพรซีเดอร์ Old_Rate โดย Old_Rate จะจัดการกับพารามิเตอร์ 4 ตัว. ถ้าหากมีโปรแกรม ILE หรือเซอร์วิสโปรแกรมใดที่ใช้โพรซีเดอร์ Rate เดิมก็จำเป็นต้องสร้างโพรซีเดอร์นั้นใหม่ โดยทำตามขั้นตอนต่อไปนี้:

- ในการใช้โพรซีเดอร์ Rate ตัวเดิม ซึ่งมีพารามิเตอร์สี่ตัว ก็ให้เรียกโพรซีเดอร์ Old_Rate แทนการเรียกโพรซีเดอร์ Rate.
- ในการใช้โพรซีเดอร์ Rate ตัวใหม่ ซึ่งมีพารามิเตอร์ Credit_History เป็นตัวที่ห้า ก็ให้เรียกโพรซีเดอร์ Rate.

เมื่อคุณทำการอัพเดตเซอร์วิสโปรแกรมก็ให้ตามข้อกำหนดดังต่อไปนี้:

- สนับสนุนโพรซีเดอร์ที่เปลี่ยนจำนวนของพารามิเตอร์ที่มันประมวลผล
- อนุญาตให้โปรแกรมที่มีอยู่และเซอร์วิสโปรแกรมที่ใช้เซอร์วิสโปรแกรมที่อัพเดตไม่มีการเปลี่ยนแปลงใดๆ

ขั้นตอนต่อไปนี้เป็นสิ่งที่คุณต้องปฏิบัติ:

1. ทำสำเนาบล็อก STRPGMEXP, ENDPGMEXP ที่มีคำสั่ง PGMLVL(*CURRENT) อยู่.
2. เปลี่ยนคำสั่ง PGMLVL(*CURRENT) ที่อยู่ในสำเนาให้เป็น PGMLVL(*PRV).
3. ในคำสั่ง STRPGMEXP ที่มี PGMLVL(*CURRENT) ให้เปลี่ยนชื่อโพรซีเดอร์เดิม แต่ให้ปล่อยมันไว้ในตำแหน่งเดิม.

   ในตัวอย่างนี้ Rate ถูกเปลี่ยนชื่อเป็น Old_Rate แต่ปล่อยให้ตำแหน่งเดิมในรายการสัญลักษณ์ที่เอ็กซ์พอร์ต.
4. ในคำสั่ง STRPGMEXP ที่มีตำสั่ง PGMLVL(*CURRENT) อยู่ ให้คุณนำชื่อโพรซีเดอร์เดิมไปใส่ไว้ที่ส่วนล่างของรายการที่สนับสนุนพารามิเตอร์ที่ไม่เท่ากัน.

   ในตัวอย่างนี้ Rate ถูกเพิ่มเข้าไปที่ส่วนท้ายรายการสัญลักษณ์ที่เอ็กซ์พอร์ต แต่โพรซีเดอร์ Rate นี้สนับสนุนพารามิเตอร์ Credit_History เพิ่มเติม.
5. บันทึกซอร์สของภาษา Binder ที่เปลี่ยนแปลง.
6. ในไฟล์ที่มีซอร์สโค้ดอยู่ ให้อัพเดตโพรซีเดอร์เดิมให้สนับสนุนพารามิเตอร์ใหม่.

   ในตัวอย่าง นี่หมายถึงการเปลี่ยนแปลงโพรซีเดอร์ Rate ที่มีอยู่ให้สนับสนุนพารามิเตอร์ตัวที่ห้า ซึ่งก็คือ Credit_History.
7. โพรซีเดอร์ใหม่ถูกสร้างขึ้นจะจัดการให้พารามิเตอร์เดิมเป็อินพุต และเรียกโพรซีเดอร์ใหม่ที่มีพารามิเตอร์พิเศษ (Dummy).

   ในตัวอย่าง นี่หมายความว่าการเพิ่มโพรซีเดอร์ Old_Rate ขึ้นเพื่อใช้จัดการกับพารามิเตอร์เดิม และเรียกโพรซีเดอร์ Rate ตัวใหม่โดยให้พารามิเตอร์ตัวที่ห้าเป็น Dummy.
8. บันทึกซอร์สโค้ดของภาษา Binder ที่เปลี่ยนแปลง.
9. สร้างอ็อบเจ็กต์โมดูลด้วยโพรซีเดอร์ใหม่ และโพรซีเดอร์ที่เปลี่ยนแปลง.
10. สร้างเซอร์วิสโปรแกรมจากโมดูลใหม่และโมดูลที่ปลี่ยนแปลงโดยใช้ภาษา Binder ที่อัพเดต.

การเปลี่ยนแปลงโปรแกรม

คำสั่ง Change Program (CHGPGM) ใช้เปลี่ยนแอ็ตทริบิวต์ของโปรแกรมโดยไม่จำเป็นต้องคอมไพล์ใหม่. แอ็ตทริบิวต์บางตัวที่สามารถเปลี่ยนแปลงได้คือ:

- แอ็ตทริบิวต์ Optimization.
- แอ็ตทริบิวต์User profile.
- แอ็ตทริบิวต์ Use adopted authority.
- แอ็ตทริบิวต์ Performance collection.
- แอ็ตทริบิวต์ Profiling data.
- ข้อความของโปรแกรม.
- อ็อพชัน Licensed Internal Codes.
- แอ็ตทริบิวต์ Teraspace.

ผู้ใช้สามารถบังคับให้ทำการสร้างโปรแกรมขึ้นใหม่ก็ได้ แม้ว่าแอ็ตทริบิวต์ที่ระบุจะเหมือนกันกับแอ็ตทริบิวต์ที่ใช้อยู่. ซึ่งคุณสามารถทำได้โดยระบุค่า *YES ที่พารามิเตอร์ Force Program Recreation (FRCCRT).

ค่า *NO และ *NOCRT เป็นค่าที่สามารถใช้ระบุในพารามิเตอร์ Force Program Recreation (FRCCRT). ค่าเหล่านี้เป็นตัวกำหนดว่าแอ็ตทริบิวต์ของโปรแกรมที่ถูกร้องขอควรจะถูกเปลี่ยนหรือไม่เมื่อการเปลี่ยนกำหนดว่าโปรแกรมต้องถูกสร้างขึ้นใหม่. การแก้ไขแอ็ตทริบิวต์ของโปรแกรมต่อไปนี้อาจเป็นสาเหตุให้โปรแกรมถูกสร้างขึ้นใหม่:

- พร้อมต์ของ Optimize program (พารามิเตอร์ OPTIMIZE).
- พร้อมต์ของ Use adopted authority (พารามิเตอร์ USEADPAUT).
- พร้อมต์ของ Enable performance collection (พารามิเตอร์ ENBPFRCOL).
- พร้อมต์ของ Profiling data (พารามิเตอร์ PRFDTA).
- พร้อมต์ของ User profile (พารามิเตอร์ USRPRF).
- อ็อพชัน Licensed Internal Codes (พารามิเตอร์ LICOPT).
- พร้อมต์ของ Teraspace (พารามิเตอร์ TERASPACE)

ค่า *NO ในพารามิเตอร์ Force Program Recreation (FRCCRT) หมายความว่าไม่บังคับการสร้างใหม่ของโปรแกรม แต่ถ้ามีการเปลี่ยนแปลงแอ็ตทริบิวต์ของโปรแกรมที่มีผลต่อการสร้างใหม่โปรแกรมจะถูกสร้างขึ้นใหม่. อ็อพชันนี้อนุญาตให้ระบบเป็นผู้กำหนดว่าการเปลี่ยนนั้นจำเป็นหรือไม่.

การสร้างโปรแกรมใหม่โดยใช้คำสั่ง CHGPGM หรือ CHGSRVPGM ในขณะที่ยังมีงานบางงานกำลังใช้โปรแกรมนี้อยู่ ก็จะทำให้เกิดข้อความ “Object Destroyed” ขึ้น ซึ่งงานเหล่านี้อาจล้มเหลว. โดยการเปลี่ยนค่าดีฟอลต์ของคำสั่งสำหรับ พารามิเตอร์ Force Program Recreation (FRCCRT) เป็น *NOCRT, สามารถช่วยป้องกันการสร้างโปรแกรมใหม่โดยไม่ได้ตั้งใจ.

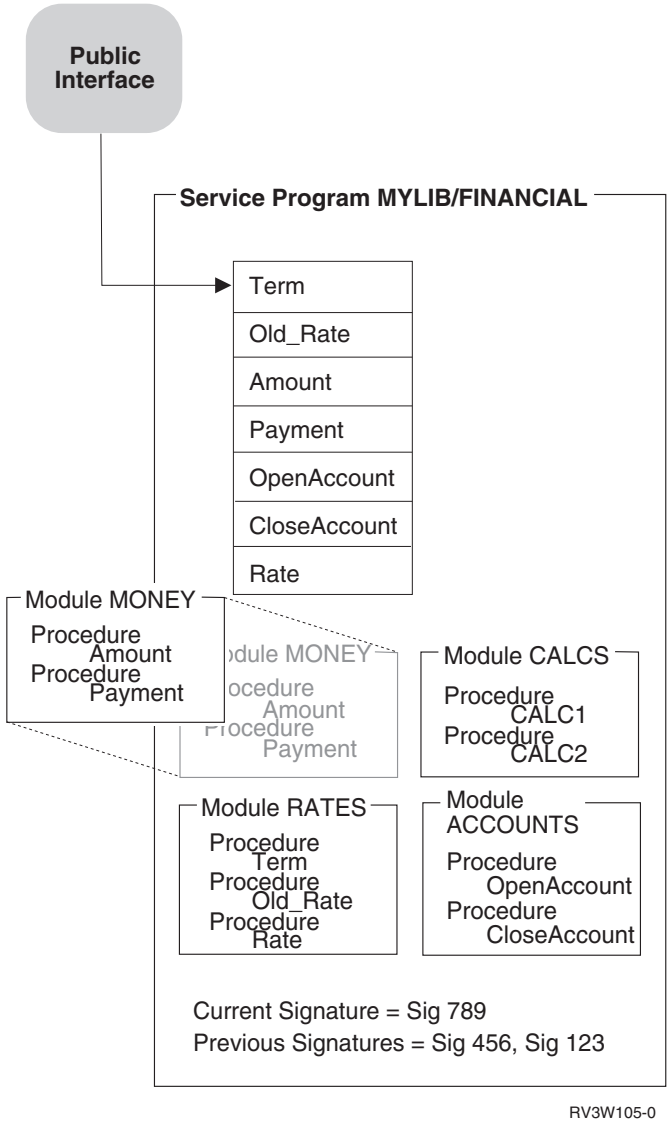
การอัพเดตโปรแกรม

หลังจากที่อ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมของ ILE ถูกสร้างขึ้น คุณอาจต้องทำการแก้ไขความผิดพลาดของโปรแกรมนั้นหรือ เพิ่มคุณลักษณะบางอย่างลงไป. อย่างไรก็ตามหลังจากที่คุณจัดการกับอ็อบเจ็กต์เรียบร้อยแล้ว มันก็อาจมีขนาดใหญ่เกินไป ซึ่งทำให้การส่งอ็อบเจ็กต์ทั้งหมดไปให้ลูกค้าของคุณเป็นเรื่องยาก และมีค่าใช้จ่ายสูง.

ดังนั้นคุณสามารถลดขนาดของโปรแกรมที่จะส่งให้ลูกค้าได้โดยใช้คำสั่ง Update Program (UPDPGM) หรือ Update Service Program (UPDSRVPGM). คำสั่งเหล่านี้จะทำการแทนที่เฉพาะโมดูลที่ระบุ หรือโมดูลมีการเปลี่ยนแปลง หรือเพิ่มขึ้นเท่านั้นที่คุณต้องการส่งให้ลูกค้า.

ถ้าคุณใช้กระบวนการ PTF โปรแกรมทางออกที่มีการเรียกไปยังคำสั่ง UPDPGM หรือ UPDSRVPGM ก็จะสามารถใช้ฟังก์ชันอัพเดตได้. การรวมโมดูลเดียวกันเข้ากับอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมหฃายๆ ตัวจำเป็นต้องใช้คำสั่ง UPDPGM หรือ UPDSRVPGM กับอ็อบเจ็กต์ *PGM และ *SRVPGM.

ตัวอย่างเช่น รูปที่ 38 ในหน้า 106



รูปที่ 38. การแทนที่โมดูลในเซอร์วิสโปรแกรม

ถ้าโปรแกรมหรือเซอร์วิสโปรแกรมถูกอัพเดตในขณะที่มันกำลังปฏิบัติงานในงานอื่นอยู่ งานนั้นจะดำเนินต่อโดยใช้เวอร์ชันเก่าของโปรแกรมหรือเซอร์วิสโปรแกรมนั้น. การเริ่มงานใหม่จะใช้เวอร์ชันของโปรแกรมหรือเซอร์วิสโปรแกรมที่ถูกอัพเดตแล้ว.

พารามิเตอร์ allow update (ALWUPD) และ allow *SRVPGM library update (ALWLIBUPD) ในคำสั่ง CRTPGM หรือ CRTSRVPGM สามารถตรวจดูได้ว่าอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมนั้นสามารถทำการอัพเดตได้หรือไม่. โดยการระบุ ALWUPD(*NO) โมดูลในอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมจะไม่สามารถแทนที่ได้โดยใช้คำสั่ง UPDPGM หรือ UPDSRVPGM. โดยระบุ ALWUPD(*YES) และ ALWLIBUPD(*YES) คุณสามารถอัพเดตโปรแกรมของคุณให้ใช้เซอร์วิสโปรแกรมจากไลบรารีที่ไม่ได้ระบุไว้ก่อน. โดยการระบุ ALWUPD (*YES) และ ALWLIBUPD(*NO) คุณสามารถอัพเดตโมดูล แต่ไม่ใช่การรวมไลบรารีของเซอร์วิสโปรแกรม. คุณไม่สามารถระบุ ALWUPD(*NO) และ ALWLIBUPD(*YES) ได้ในเวลาเดียวกัน.

พารามิเตอร์ในคำสั่ง UPDPGM และ UPDSRVPGM

ในแต่ละโมดูลที่ระบุไว้ในพารามิเตอร์ของโมดูละแทนที่โมดูลที่มีชื่อเหมือนกันที่รวมเข้าไปในอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรม. ถ้ามีโมดูลมากกว่าหนึ่งโมดูลรวมเข้าไปในอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมที่มีชื่อเหมือนกัน พารามิเตอร์ replacement library (RPLLIB) ก็จะถูกใช้งาน. งาน พารามิเตอร์นี้จะระบุวิธีการที่ใช้เลือกโมดูลในการแทนที่. ถ้าไม่มีโมดูลใดที่มีชื่อเหมือนกันในอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรม อ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมนั้นจะไม่มีการอัพเดต.

พารามิเตอร์ bound service program (BNDSRVPGM) ใช้ระบุเซอร์วิสโปรแกรมเพิ่มเติมที่นอกเหนือจากนี้ ซึ่งอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมถูกรวมไว้เรียบร้อยแล้ว. ถ้าโมดูลที่ใช้แทนที่มีการอิมพอร์ตมากกว่าหรือการเอ็กซ์พอร์ตน้อยกว่าโมดูลที่ถูกแทนที่ เซอร์วิสโปรแกรมอาจจำเป็นต้องทำการ Resolve อิมพอร์ตเหล่านี้ก่อน.

| ด้วยการใช้พารามิเตอร์ Service program library (SRVPGMLIB) คุณก็สามารถระบุไลบรารีที่รวม
| กับเซอร์วิสโปรแกรมได้. ทุกครั้งที่คุณเรียกใช้คำสั่ง UPDPGM หรือ UPDSRVPGM เซอร์วิส
| โปรแกรมจากไลบรารีที่ระบุจะถูกใช้งาน. คำสั่ง UPDPGM หรือ UPDSRVPGM อนุญาตให้คุณ
| เปลี่ยนไลบรารีได้ ถ้า ALWLIBUPD(*YES) ถูกใช้.

พารามิเตอร์ Binding Directory (BNDDIR) ใช้ระบุ Binding Directory ที่มีโมดูลหรือเซอร์วิสโปรแกรม ซึ่งอาจจำเป็นสำหรับ Resolve การอิมพอร์ตพิเศษ.

| พารามิเตอร์ Activation Group (ACTGRP) ใช้ระบุ ชื่อของ Activation Group ที่ถูกใช้งานเมื่อ
| โปรแกรมหรือเซอร์วิสโปรแกรมถูกเรียกทำงาน. พารามิเตอร์นี้ยังอนุญาตให้คุณเปลี่ยนชื่อของ
| Activation Group ได้.

โมดูลถูกแทนที่ด้วยโมดูลที่มีการอิมพอร์ตน้อยกว่า

ถ้าโมดูลถูกแทนที่ด้วยโมดูลอื่นที่มีการอิมพอร์ตน้อยกว่า อ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมใหม่จะถูกสร้างขึ้นเสมอ. อย่างไรก็ตาม อ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมที่อัพเดตจะมีโมดูลแบบ Isolate เกิดขึ้น ถ้าเงื่อนไขต่างๆ เหล่านี้เป็นจริง:

- เนื่องจากมีการอิมพอร์ตที่หายไป โมดูลที่รวมเข้ากับอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมจะไม่สามารถ Resolves การอิมพอร์ตได้
- เริ่มแรกโมดูลนั้นมาจาก Binding Directory ที่ใช้คำสั่ง CRTPGM หรือ CRTSRVPGM

โปรแกรมที่มีโมดูลแบบ Isolate อาจขยายขนาดขึ้นเมื่อเวลาผ่านไป. ในการลบโมดูลที่ไม่สามารถ Resolve การอิมพอร์ตและมาจาก Binding Directory คุณสามารถระบุ OPTION(*TRIM) เมื่อทำการอัพเดตอ็อบเจ็กต์. เจ็กต์ อย่างไรก็ตาม ถ้าคุณใช้อ็อพชันนี้ การเอ็กซ์พอร์ตที่โมดูลนั้นมีอยู่อาจจไม่สามารถอัพเดตได้ในอนาคต.

โมดูลถูกแทนที่ด้วยโมดูลที่มีการอิมพอร์ตมากกว่า

ถ้าโมดูลถูกแทนที่ด้วยโมดูลอื่นที่มีการอิมพอร์ตมากกว่า อ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมสามารถอัพเดตได้ ถ้าอิมพอร์ตที่เกินมานั้นสามารถ Resolve ได้ดังต่อไปนี้:

- ชุดของโมดูลที่มีอยู่รวมเข้ากับอ็อบเจ็กต์.
- เซอร์วิสโปรแกรมรวมเข้ากับอ็อบเจ็กต์.
- Binding Directoriy ถูกระบุอยู่ในคำสั่ง. ถ้าโมดูลใน Binding Directory เหล่านั้นมีต้องการเอ็กซ์พอร์ต โมดูลก็จะถูกเพิ่มเข้าไปในโปรแกรมหรือเซอร์วิสโปรแกรม. แต่ถ้าเซอร์วิสโปรแกรมใน Binding Directory เหล่านั้นมีต้องการเอ็กซ์พอร์ต เซอร์วิสโปรแกรมก็จะถูกรวมโดยอ้างอิงเข้าไปในโปรแกรมหรือเซอร์วิสโปรแกรม.
- การรวมไดเรกทอรีโดยนัย. ซึ่ง**การรวมไดเรกทอรีโดยนัย** คือ Binding Directory ที่มีเอ็กซ์พอร์ตที่จำเป็นต่อการสร้างโปรแกรมที่มีโมดูล. คอมไพเลอร์ของ ILE ตัวตัวจะสร้างรายชื่อของการรวมไดเรกทอรีโดยในเข้าไปในแต่ละโมดูลที่สร้าง.

ถ้าอิมพอร์ตที่เกินมาไม่สามารถ Resolve ได้ การอัพเดตก็จะล้มเหลวจนกว่าคุณจะระบุ OPTION (*UNRSLVREF) ไว้ในคำสั่งอัพเดต.

โมดูลถูกแทนที่ด้วยโมดูลที่มีการเอ็กซ์พอร์ตน้อยกว่า

ถ้าโมดูลถูกแทนที่ด้วยโมดูลอื่นที่มีการเอ็กซ์พอร์ตน้อยกว่าแล้ว การอัพเดตจะเกิดขึ้นได้ก็ต่อเมื่อเงื่อนไขเหล่านี้เป็นจริง:

- เอ็กซ์พอร์ตที่หายไปไม่จำเป็นต่อการรวม.
- เอ็กซ์พอร์ตที่หายไปไม่ไม่ได้ถูกเอ็กซ์พอร์ตออกจากเซอร์วิสโปรแกรมเซอร์วิสโปรแกรมในกรณีของ UPDSRVPGM.

ถ้าเซอร์วิสโปรแกรมถูกอัพเดตโดยระบุ EXPORT(*ALL) รายการเอ็กซปอร์ตใหม่จะถูกสร้างขึ้น. รายการเอ็กซปอร์ตใหม่จะแตกต่างไปจากรายการเอ็กซปอร์ตเดิม.

การอัพเดตจะไม่เกิดขึ้นถ้าเงื่อนไขเหล่านี้เป็นจริง:

- การอิมพอร์พอร์ตบางตัวไม่สามารถ Resolve ได้ เนื่องจากมีเอ็กซ์พอร์ตหายไป.
- เอ็กซ์พอร์ตที่หายไปเหล่านั้นไม่สามารถพบได้จากเซอร์วิสโปรแกรม และ Binding Directory พิเศษที่ระบุไว้ในคำสั่ง.
- ภาษา Binder แสดงการเอ็กซ์พอร์ตเป็นสัญลักษณ์ แต่การเอ็กซ์พอร์ตนั้นหายไป.

โมดูลถูกแทนที่ด้วยโมดูลที่มีการเอ็กซ์พอร์ตมากกว่า

ถ้าโมดูลถูกแทนที่ด้วยโมดูลอื่นที่มีการเอ็กซ์พอร์ตมากกว่าแล้ว การอัพเดตจะเกิดขึ้นถ้าเอ็กซ์พอร์ตที่เกินมานี้มีชื่อไม่ซ้ำกัน. เซอร์วิสโปรแกรมที่เอ็กซ์พอร์ตมาจะต่างกันถ้า EXPORT(*ALL) ถูกระบุไว้.

อย่างไรก็ตาม ถ้าเอ็กซ์พอร์ตที่เกินมาตัวใดตัวหนึ่งมีชื่อซ้ำ ชื่อที่ซ้ำนั้นก็อาจก่อให้เกิดปัญหาได้ดังนี้:

- ถ้า OPTION(*NODUPPROC) หรือ OPTION(*NODUPVAR) ถูกระบุไว้ในคำสั่งอัพเดตอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมจะไม่ถูกอัพเดต.
- ถ้า OPTION(*DUPPROC) หรือ OPTION(*DUPVAR) ถูกระบุไว้ การอัพเดตก็จะเกิดขึ้นแต่เอ็กซ์พอร์ตพิเศษอาจถูกใช้แทนที่จะเป็นเอ็กซปอร์ตต้นฉบับที่มีชื่อเหมือนกัน.

คำแนะนำในการสร้างโมดูล โปรแกรม และเซอร์วิสโปรแกรม

เมื่อคุณต้องการสร้างและบำรุงรักษาโมดูล, โปรแกรม ILE, และเซอร์วิสโปรแกรมอย่างสะดวก ขอให้พิจารณาเรื่องต่อไปนี้:

- ปฏิบัติตามหลักการตั้งชื่อของโมดูลที่ทำสำเนาเพื่อสร้างโปรแกรมหรือเซอร์วิสโปรแกรม. วิธีการตั้งชื่อโดยใช้ Prefix จะช่วยทำให้เข้าใจลักษณะของโมดูล และพารามิเตอร์ได้ง่ายขึ้น.
- เพื่อความสะดวกในการบำรุงรักษาให้ใช้โมดูลหนึ่งโมดูลกับโปรแกรมหรือเซอร์วิสโปรแกรมเพียงตัวเดียว. . ถ้ามีโปรแกรมมากกว่าหนึ่งตัวต้องการใช้โมดูล ก็ให้นำโมดูลไปใส่ไว้ในเซอร์วิสโปรแกรม. ด้วยวิธีนี้ หากคุณต้องออกแบบโมดูลใหม่ คุณก็แค่ออกแบบใหม่เพียงที่เดียว.
- เพื่อให้แน่ใจใน signature ของคุณ ให้ใช้ภาษา Binder ทุกครั้งที่คุณสร้างเซอร์วิสโปรแกรม.

  ภาษา Binder ช่วยให้เซอร์วิสโปรแกรมถูกอัพเดตได้ง่าย โดยไม่จำเป็นต้องสร้างโปรแกรมและเซอร์วิสโปรแกรมใหม่.

  คำสั่ง Retrieve Binder Source (RTVBNDSRC) สามารถช่วยสร้างซอร์สของภาษา Binder ตามการเอ็กซ์พอร์ตจากโมดูล หรือเซอร์วิสโปรแกรม.

  ถ้าเงื่อนไขใดเงื่อนไขหนึ่งเป็นจริง:

  - เซอร์วิสโปรแกรมไม่มีการเปลี่ยนแปลง
  - – ผู้ใช้เซอร์วิสโปรแกรมไม่สนใจที่จะเปลียนโปรแกรมของพวกเขาเมื่อ signature เปลี่ยนแปลง.

  คุณไม่จำเป็นต้องใช้ภาษา Binder. เนื่องจากในกรณีนี้ต่างจากแอ็พพลิเคชันส่วนใหญ่ คุณควรพิจรณาการใช้ภาษา Binder สำหรับเซอร์วิสโปรแกรมทั้งหมด.
- หากคุณได้รับข้อความ CPF5D04 เมื่อใช้คำสั่งการสร้างโปรแกรม เช่น CRTPGM, CRTSRVPGM, หรือ UPDPGM, แต่ยังมีการสร้างโปรแกรมหรือเซอร์วิสโปรแกรมของคุณอยู่, อาจมีคำอธิบายที่เป็นไปได้สองประการ:
  1. โปรแกรมของคุณถูกสร้างด้วย OPTION(*UNRSLVREF) และมีการอ้างอิงที่ยังไม่ลงตัว .
  2. คุณกำลังเชื่อมเข้ากับ *SRVPGM ซึ่งแสดงรายการใน *BNDDIR QSYS/QUSAPIBD ที่ถูกส่ง ด้วยสิทธิ *PUBLIC *EXCLUDE, และคุณไม่มีสิทธิในการใช้งาน. เพื่อดูว่า ผู้ใดมีสิทธิในการใช้งานอ็อบเจ็กต์, ให้ใช้คำสั่ง DSPOBJAUT. ระบบ *BNDDIR QUSAPIBD มีชื่อของ *SRVPGMs ที่ให้ API ระบบ. บาง API ไวต่อระบบรักษาความปลอดภัย, ดังนั้น *SRVPGMs ที่ API อยู่จะถูกส่งด้วย *PUBLIC *EXCLUDE authority. *SRVPGM เหล่านี้ถูกจัดกลุ่มเมื่อสิ้นสุด QUSAPIBD. เมื่อ คุณจะใช้เซอร์วิสโปรแกรม *PUBLIC *EXCLUDE ในรายการนี้, ตามปกติตัวเชื่อม จะต้องตรวจสอบ *PUBLIC *EXCLUDE *SRVPGM อื่นก่อนเซอร์วิสโปรแกรมของคุณ, และ ใช้ CPF5D04.

  เพื่อหลีกเลี่ยงข้อความ CPF5D04, ให้ใช้วิธีใดวิธีหนึ่งต่อไปนี้:

  - ระบุอย่างชัดเจนถึง *SRVPGM ใดๆ ที่โปรแกรมหรือเซอร์วิสโปรแกรมของคุณ เชื่อมโยงอยู่. เพื่อดูรายการของ *SRVPGMS ที่โปรแกรมหรือเซอร์วิสโปรแกรมของคุณเชื่อมโยงอยู่ , ให้ใช้ DSPPGM หรือ DSPSRVPGM DETAIL(*SRVPGM). *SRVPGM เหล่านี้สามารถระบุได้บน พารามิเตอร์ CRTPGM หรือ CRTSRVPGM BNDSRVPGM. อีกทั้งยังสามารถใส่ใน binding directory ที่ให้บนพารามิเตอร์ CRTBNDRPG, CRTRPGMOD,

CRTBNDCBL, CRTPGM หรือ CRTSRVPGM BNDDIR, หรือจาก RPG H-spec. การดำเนินการเช่นนี้ทำให้แน่ใจได้ว่าการอ้างอิงทั้งหมดจะ resolve ได้ก่อน จำเป็นต้องตรวจสอบ *PUBLIC *EXCLUDE *SRVPGM ใน *BNDDIR QUSAPIBD.

- ให้สิทธิ *PUBLIC หรือสิทธิเฉพาะตัวแก่ *SRVPGM ที่แสดงรายการใน ข้อความ CPF5D04. ซึ่งจะมีข้อเสียของการให้สิทธิแก่ผู้ใช้เข้าใช้งานอินเตอร์เฟสที่อาจจะ ไวต่อระบบรักษาความปลอดภัยโดยไม่จำเป็น.
- หากมีการใช้ OPTION(*UNRSLVREF) และโปรแกรมของคุณมีการอ้างอิงที่ยังไม่ลงตัว, โปรดตรวจสอบให้แน่ใจว่าการอ้างอิงทั้งหมดลงตัว.

- ถ้ามีบุคคลอื่นต้องการใช้อ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมที่คุณสร้างขึ้น ให้ระบุ OPTION(*RSLVREF) ในเวลาที่สร้างมัน. เมื่อคุณพัฒนาแอ็พพลิเคชัน คุณอาจต้องการสร้างอ็อบเจ็กต์โปรแกรมหรือเซอร์วิสโปรแกรมด้วย Unresolved Imports. อย่างไรก็ดี เมื่อถึงขั้นตอนการผลิต อิมพอร์ตทั้งหมดต้องดีรับการ Resolve.

  ถ้า OPTION(*WARN) ถูกระบุไว้ การอ้างถุงที่ไม่ได้รับการ Resolve จะแสดงไว้ในบันทึกการใช้งาที่มีการรร้องขอ CRTPGM หรือ CRTSRVPGM . ถ้าคุณระบุรายการไว้ในพารามิเตอร์ DETAIL พวกมันก็จะถูกรวมเข้าไว้ในรายการของโปรแกรม. โปรแกรม คุณควรตรวจดูในบันทึกการใช้งาน หรือรายการของโปรแกรมเสมอ.
- เมื่อคุณออกแบบแอ็พพลิเคชันใหม่ การกำหนดโพรซีเดอร์หลักที่ควรอยู่ในเซอร์วิสโปรแกรมเป็นเรื่องง่าย.

  มันเป็นการง่ายกว่าที่จะออกแบบโพรซีเดอร์หลักสำหรับแอพพลิเคชันใหม่. ถ้าคุณทำการแปลงแอพพลิเคชันที่มีอยู่ให้ใช้ ILE มันอาจยกกว่าที่จะกำหนดโพรซีเดอร์หลักสำหรับเซอร์วิสโปรแกรม. แต่คุณก็ควรลองกำหนดโพรซีเดอร์หลักที่จำเป็นต่อแอ็พพลิเคชัน และลองสร้างเซอร์วิสโปรแกรมที่มีโพรซีเดอร์หลัก.
- เมื่อคุณแปลงแอ็พพลิเคชันที่มีอยู่ให้ใช้ ILE ขอให้พิจารณาสร้างโปรแกรมขนาดใหญ่จำนวนน้อยๆ.

  ด้วยโปรแกรมขนาดใหญ่จำนวนน้อยๆ ซึ่งมักมีการเปลี่นรแปลงเล็กน้อย คุณสามารถแปลงแอ็พพลิเคชันที่อยู่ให้ใช้ประโยชน์จากคุณลักษณะของ ILE ได้ง่าย. หลังจากที่คุณสร้างโมดูลแล้ว การนำมันมารวมกันเป็นโปรแกรมขนาดใหญ่จำนวนน้อยๆ จะเป็นวิธีการแปลงให้เป็น ILE ที่ง่ายและประหยัดที่สุด.

  นอกจากนี้การใช้โปรแกรมขนาดใหญ่จำนวนน้อยๆ แทนการใช้โปรแกรมขนาดเล็กจำนวนมากยังช่วยคุณให้ที่เก็บข้อมูลน้อยกว่าอีกด้วย.
- พยายามจำกัดจำนวนของเซอร์วิสโปรแกรมที่แอ็พพลิเคชันของคุณใช้อยู่.

  ซึ่งอาจต้องการให้เซอร์วิสโปรแกรมสร้างมากกว่าหนึ่งโมดูล. ข้อดีของมันคือเวลาในการเรียกใช้และรวมรวดเร็วขึ้น.

  มันเป็นการยากที่จะเจาะจงว่าจำนวนที่เหมาะสมของเซอร์วิสโปรแกรมที่แอ็พพลิเคชันใช้ควรเป็นเท่าไร. ถ้าหากโปรแกรมใช้เซอร์วิสโปรแกรมเป็นร้อยตัว มันก็ดูเหมือนว่าจะมากเกินไป. ในทางกลับกัน การใช้เซอร์วิสโปรแกรมเพียงตัวเดียวก็ดูไม่ค่อยเหมาะสมนัก.

  ตัวอย่างเช่น เซอร์วิสโปรแกรมประมาณ 10 ตัวสำหรับรูทีนที่ระบุภาษาและรูทีนหลักซึ่งจัดเตรียมไว้ให้โดย OS/400. มากกว่า 70 โมดูลที่ใช้ในการสร้างเซอร์วิสโปรแกรมทั้ง 10 ตัวเหล่านี้. ซึ่งเป็นอัตราส่วนที่เหมาะสมระหว่างประสิทธิภาพ ความเข้าใจ และการดูแลรักษา.

บทที่ 6. การบริหาร Activation Group

ในบทนี้ประกอบด้วยตัวอย่างของวิธีการในการจัดโครงสร้างของแอ็พพลิเคชันที่ใช้ activation group. ซึ่งประกอบด้วยหัวข้อดังต่อไปนี้:

- การสนับสนุนแอ็พพลิเคชันหลายตัวในงานเดียวกัน
- การใช้คำสั่ง Reclaim Resources (RCCRSC) กับโปรแกรม OPM และ ILE
- การลบ activation group ด้วยคำสั่ง Reclaim Activation Group (RCLACTGRP)
- เซอร์วิสโปรแกรมและ activation group

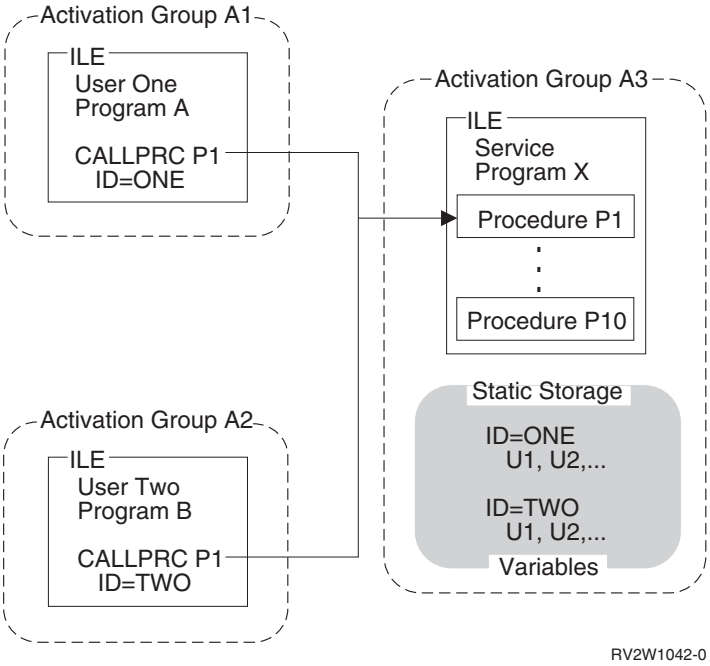
แอ็พพลิเคชันหลายตัวที่รันอยู่ในงานเดียวกัน

Activation group ที่ผู้ใช้กำหนดชื่อให้ จะยอมให้ทิ้ง activation group ใน job หนึ่ง ๆ เพื่อไว้ใช้ในภายหลัง. คำสั่งในรีเทิร์นหรือคำสั่งในการกระโดดข้าม (เช่น longjump() ใน ILE C) ผ่านขอบเขตการควบคุมจะไม่ลบ activation group ของคุณ.

การทำเช่นนี้คือการยอมให้คุณทิ้งแอ็พพลิเคชันของคุณไว้ในสถานะที่ถูกใช้งานครั้งสุดท้าย. ตัวแปรแบบคงที่และไฟล์ที่เปิดไว้ยังคงไม่มีการเปลี่ยนแปลงระหว่างการเรียกไปยังแอ็พพลิเคชันของคุณ. วิธีนี้จะสามารถประหยัดเวลาในการทำงานและอาจจำเป็นต่อการการสร้างฟังก์ชันที่คุณพยายามสร้างขึ้น.

อย่างไรก็ตามคุณต้องเตรียมรับคำร้องขอ (request) จากไคลเอ็นต์หลายๆ รายที่กำลังรันอยู่ในงานเดียวกัน. ระบบจะไม่จำกัดจำนวนของโปรแกรม ILE ที่จะถูกรวมเข้ากับเซอร์วิสโปรแกรมของคุณ. เป็นผลให้คุณจำเป็นต้องสนับสนุนไคลเอ็นต์หลายราย.

รูปที่ 39 ในหน้า 112 แสดงเทคนิคที่ใช้ในการแบ่งใช้ฟังก์ชั่นบริการแบบทั่วไป ในขณะที่ยังรักษาข้อได้เปรียบทางด้านประสิทธิภาพของ activation group ที่ผู้ใช้กำหนดชื่อให้ไว้ได้.



รูปที่ 39. แสดงแอ็พพลิเคชันหลายตัวที่รันอยู่ในงานเดียวกัน

การเรียกใช้โพรซิเดอร์ในเซอร์วิสโปรแกรม X แต่ละครั้งต้องมีผู้ใช้รับผิดชอบ.ในตัวอย่างนี้ ฟิลด์ ID แทนค่าการจัดการจากผู้ใช้. ผู้ใช้แต่ละคนจะรับผิดชอบในการกำหนดชื่อนี้. โปรแกรมเมอร์จะเป็นผู้สร้างรูทีน เพื่อให้ชื่อที่ไม่ซ้ำกันสำหรับผู้ใช้แต่ละคน.

เมื่อมีการเรียกไปยังเซอร์วิสโปรแกรมของคุณ ชื่อของผู้ใช้จะถูกใช้ในการกำหนดตำแหน่งที่เก็บตัวแปรที่เกี่ยวข้องกับผู้ใช้นี้. ในช่วงที่มีการบันทึกค่าในการสร้าง activation group คุณสามารถสนับสนุนไคลเอ็นต์หลายรายได้ในเวลาเดียวกัน.

คำสั่ง Reclaim Resources

คำสั่ง Reclaim Resources (RCLRSC) ขึ้นอยู่กับแนวคิดที่เรียกว่า **หมายเลขระดับ**. ซึ่งเป็นค่าที่ไม่ซ้ำกันที่กำหนดโดยระบบให้กับรีซอร์สที่ใช้ภายในงานๆ หนึ่ง. การกำหนดหมายเลขลำดับแบ่งออกเป็น 3 ระดับคือ:

หมายเลขระดับการเรียก

Call stack entry แต่ละตัวจะได้รับหมายเลขระดับเฉพาะของตัวเอง

หมายเลขระดับ Program-activation

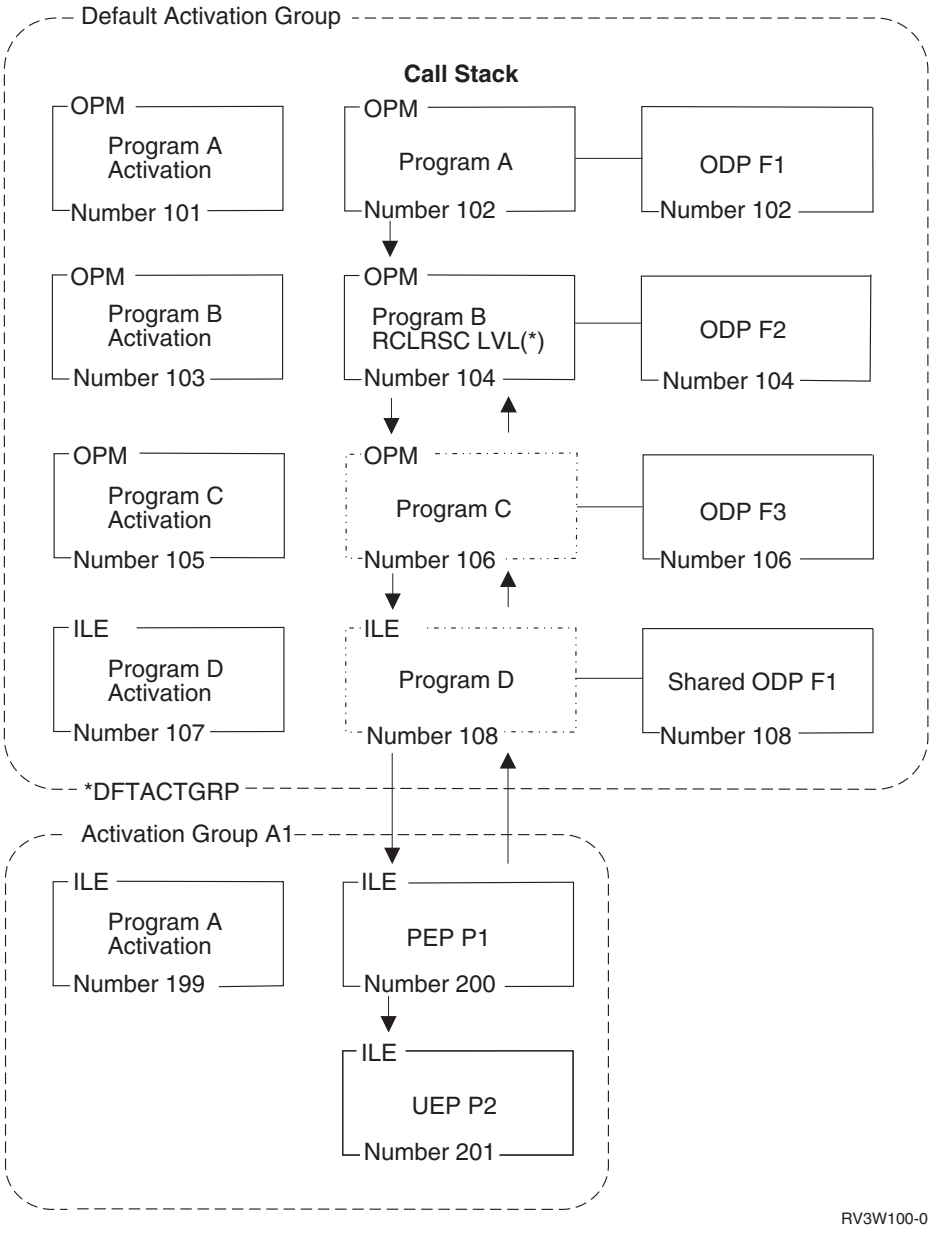
Program activation ของโปรแกรม OPM และ ILE แต่ละตัวจะได้รับหมายเลขระดับเฉพาะของตัวเอง

หมายเลขระดับ Activation-group

Activation group แต่ละตัวจะได้รับหมายเลขระดับเฉพาะของตัวเอง

ในขณะที่งานของคุณรันอยู่ ระบบจะยังคงกำหนดค่าหมายเลขระดับเฉพาะให้กับรีซอร์สที่เกิดขึ้นใหม่ในแต่ละครั้ง. โดยค่าหมายเลขระดับที่กำหนดเป็นค่าที่เพิ่มขึ้น. ดังนั้นรีซอร์สที่มีค่าหมายเลขระดับสูงจะถูกสร้างหลังจากรีซอร์สที่มีค่าหมายเลขระดับต่ำกว่า.

รูปที่ 40แสดงตัวอย่างของการใช้คำสั่ง RCLRSC ในโปรแกรม OPM และ ILE. จากรูป call-level scoping ถูกใช้สำหรับไฟล์ที่เปิดอยู่. เมื่อ call-level scoping ถูกใช้ทำให้รีซอร์สในการจัดการข้อมูลแต่ละตัวจะได้รับ หมายเลขระดับ ค่าเดียวกับของ call stack entry ที่สร้างรีซอร์สเหล่านั้น.



รูปที่ 40. Reclaim Resources

ในตัวอย่างนี้ ลำดับการเรียกคือ โปรแกรม A, B, C, และ D โปรแกรม D และ C ย้อนกลับไปยังโปรแกรม B โปรแกรม B ใช้คำสั่ง RCLRSC ที่มีอ็อพชัน LVL(*). คำสั่ง RCLRSC ใช้พารามิเตอร์ LVL ในการลบรีซอร์ส. รีซอร์สทั้งหมดที่มีหมายเลขระดับการเรียกมากกว่าหมายเลขระดับการเรียก ของ call stack entry ปัจจุบันจะถูกลบออกไป. ในตัวอย่างนี้หมายเลขระดับการเรียกที่ 104 ถูกใช้เป็นจุดเริ่มต้น. รีซอร์สทุกตัวที่มีค่าหมายระดับการเรียกมากกว่า 104 จะถูกลบทิ้งทั้งหมด.

จะสังเกตได้ว่ารีซอร์สในระดับการเรียกที่ 200 และ 201 จะไม่ถูกผลกระทบจากคำสั่ง RCLRSC เพราะอยู่ใน activation group ของ ILE. เนื่องจากคำสั่ง RCLRSC ทำงานเฉพาะใน activation group ที่กำหนดไว้เท่านั้น.

นอกจากนี้ที่เก็บข้อมูลของโปรแกรม C และ D และ open data path (ODP) สำหรับไฟล์ F3 จะถูกปิด. ไฟล์ F1 ถูกแบ่งใช้กับ ODP ที่เปิดอยู่ในโปรแกรม A. จะถูกปิดแต่ไฟล์ F1 จะยังคงเปิดอยู่.

คำสั่ง Reclaim Resources สำหรับโปรแกรม OPM

คำสั่ง Reclaim Resources อาจใช้ในการปิดไฟล์ที่เปิดอยู่และยกเลิกการใช้ที่เก็บข้อมูลแบบสแตติกของโปรแกรม OPM ที่รีเทิร์นโดยที่ยังไม่จบการทำงาน. ภาษา OPM บางภาษา เช่น อาร์พีจี ยอมให้คุณรีเทิร์นโดยไม่ต้องจบการทำงานได้. ถ้าคุณต้องการปิดไฟล์ของโปรแกรมและยกเลิกการใช้พื้นที่ของมันในอนาคต คุณอาจใช้คำสั่ง RCLRSC.

คำสั่ง Reclaim Resources สำหรับโปรแกรม ILE

สำหรับ ILE ที่สร้างขึ้นด้วยคำสั่ง CRTBNDxxx ที่มีการระบุค่า DFTACTGRP(*YES) คำสั่ง RCLRSC จะยกเลิกการใช้ที่เก็บข้อมูลแบบคงที่ เช่นเดียวกับในโปรแกรม OPM. สำหรับโปรแกรม ILE ที่ไม่ได้สร้างขึ้นด้วยคำสั่ง CRTBNDxxx และระบุค่า DFTACTGRP(*YES) คำสั่ง RCLRSC จะให้ค่าเริ่มต้นอีกครั้งแก่การ activate ใดๆ ที่ถูกสร้างขึ้นใน default activation group แต่จะไม่ยกเลิกการใช้ที่เก็บข้อมูลแบบสแตติก. โปรแกรม ILE ที่ใช้ที่เก็บข้อมูลแบบสแตติกเป็นจำนวนมากควรถูก acivate ใน activation group ของ ILE. การลบ activation group จะคืนที่เก็บข้อมูลนี้ให้กับระบบ. คำสั่ง RCLRSC จะปิดไฟล์ที่เปิดโดยเซอร์วิสโปรแกรมหรือโปรแกรม ILE ที่รันใน activation group ที่กำหนดไว้. RCLRSC จะไม่ให้ค่าเริ่มต้นอีกครั้งแก่ที่เก็บข้อมูลแบบสแตติกของเซอร์วิสโปรแกรมและไม่ส่งผลกระทบต่อ activation group ที่ไม่ใช่ activation group ที่กำหนดไว้.

เพื่อใช้คำสั่ง RCLRSC จาก ILE โดยตรง คุณสามารถใช้ทั้ง QCAPCMD API และโพรซีเดอร์ CL ของ ILE. QCAPCMD API ยอมให้คุณเรียกคำสั่งของระบบได้โดยตรงโดยไม่ต้องใช้โปรแกรม CL. ใน รูปที่ 40 ในหน้า 113การเรียกคำสั่งของระบบโดยตรงมีความสำคัญ เนื่องจากคุณอาจต้องการใช้ call-level number ของ โพรซีเดอร์ ILE ตัวใดตัวหนึ่งโดยเฉพาะ. ในบางภาษา เช่น ILE C, 400 มีฟังก์ชันของระบบที่ยอมให้รันคำสั่งของ OS/400 ได้โดยตรง.

คำสั่ง Reclaim Activation Group

คำสั่ง Reclaim Activation Group (RCLACTGRP) สามารถใช้ในการลบ activation group ที่ไม่ใช่ดีฟอลต์ และไม่ได้ใช้งานอยู่ได้. คำสั่งนี้อนุญาตให้เลือกได้ทั้งการลบ activation group ที่ไม่ต้องการออกทั้งหมดและการลบเพียง activation group เดียวโดยการระบุชื่อ.

เซอร์วิสโปรแกรมและ Activation Group

เมื่อคุณสร้างเซอร์วิสโปรแกรม ILE คุณต้องเลือกว่าจะระบุอ็อพชันเป็นแบบใดระหว่าง *CALLER หรือชื่อสำหรับ ACTGRP. อ็อพชันนี้เป็นการกำหนดว่า เซอร์วิสโปรแกรมของคุณจะถูก activate เข้าไปใน activation group ของผู้เรียกหรือไปยัง activation group อีกกลุ่มหนึ่งที่มีชื่อตามที่ระบุไว้. ซึ่งทั้ง 2 แบบต่างมีทั้งข้อดีและข้อเสียแตกต่างกัน. ในหัวข้อนี้จะกล่าวถึงผลที่ได้จากทางเลือกแต่ละแบบ.

การกำหนดพารามิเตอร์ ACTGRP(*CALLER) จะทำให้เซอร์วิสโปรแกรมทำงานดังนี้:

- การเรียกโพรซิเดอร์แบบสแตติกจะเร็วขึ้น

  การเรียกโพรซิเดอร์แบบสแตติก ที่ไปยังเซอร์วิสโปรแกรมจะถูก optimize เมื่อรันใน activation group เดียวกัน.
- การแบ่งใช้ข้อมูลภายนอก

  เซอร์วิสโปรแกรมอาจ export ข้อมูลที่ถูกใช้โดยโปรแกรมอื่น และเซอร์วิสโปรแกรมอื่นใน activation group เดียวกัน.
- ท การแบ่งใช้รีซอร์สจัดการข้อมูล

  ไฟล์ที่เปิดอยู่และ รีซอร์สจัดการข้อมูลอื่นๆ อาจถูกแบ่งใช้ระหว่างเซอร์วิสโปรแกรมและโปรแกรมอื่นใน activation group. เซอร์วิสโปรแกรมอาจจะเกิดการ commit หรือ rollback ซึ่งส่งผลกระทบต่อโปรแกรมอื่นใน activation group.
- ไม่มีขอบเขตการควบคุม

  Exception ที่ไม่ถูกจัดการในเซอร์วิสโปรแกรมจะผ่านไปยังโปรแกรมของไคลเอ็นต์. HLL end verb ที่ใช้ในเซอร์วิสโปรแกรมสามารถลบ activation group ของโปรแกรมของไคลเอ็นต์ได้.

การกำหนดพารามิเตอร์ ACTGRP(name) จะทำให้ เซอร์วิสโปรแกรม ทำงานดังนี้:

- แยกแอดเดรสสเปซสำหรับตัวแปร

  โปรแกรมของไคลเอ็นต์ไม่สามารถย้ายพอยน์เตอร์ไปยังตำแหน่งของที่เก็บข้อมูลที่คุณต้องการได้. ซึ่งอาจเป็นเรื่องสำคัญถ้า เซอร์วิสโปรแกรม รันด้วย adopted authority.
- แยกรีซอร์สจัดการข้อมูล

  คุณมีไฟล์ที่เปิดอยู่และมีการกำหนด commitment ที่เป็นของตัวเอง. เป็นการป้องกันข้อผิดพลาดในการแบ่งใช้ไฟล์ที่เปิดอยู่อย่างไม่ตั้งใจ.
- แสดงสถานะที่ถูกควบคุม

  คุณสามารถควบคุมเวลาในการลบพื้นที่เก็บแอ็พพลิเคชัน. โดยใช้ HLL end verbs หรือคำสั่ง return คุณสามารถตัดสินใจได้ว่าเมื่อไรจะลบแอ็พพลิเคชัน. แต่คุณจำต้องจัดการข้อมูลแสดงสถานะสำหรับไคลเอ็นต์หลายราย.

บทที่ 7. การเรียกโพรซีเดอร์และโปรแกรม

การใช้ call stack ใน ILE และวิธีการส่งผ่านอากิวเมนต์ทำให้เกิดการสื่อสารระหว่างภาษาขึ้น ซึ่งช่วยให้คุณสามารถเขียนแอ็พพลิเคชันที่ประกอบด้วยหลายภาษาได้ง่ายขึ้น. ในบทนี้จะกล่าวถึงตัวอย่างของ dynamic program calls และ การเรียกโพรซิเดอร์แบบสแตติก ซึ่งเคยกล่าวนำไปแล้วใน “การเรียกไปยังโปรแกรมและโพรซีเดอร์” ในหน้า 25. และจะแนะนำการเรียกชนิดที่ 3 คือ procedure pointer call.

นอกจากนี้ ยังจะกล่าวถึงการสนับสนุนของ original program model (OPM) ต่อ application programming interface (API) ของ OPM และ ILE.

Call Stack

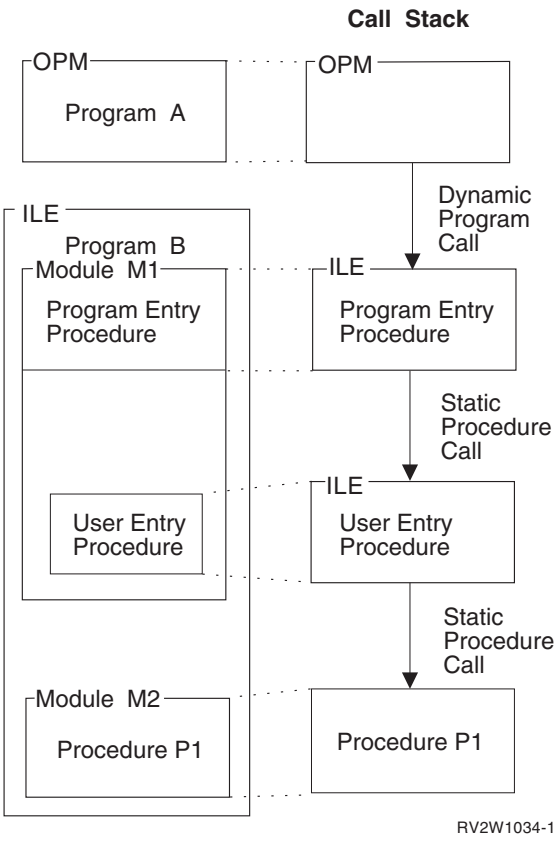
Call stack เป็นรายการของ **call stack entries** แบบ last-in-first-out (LIFO) โดยหนึ่งรายการ (entry) ใช้สำหรับการเรียกโพรซีเดอร์หรือโปรแกรมในแต่ละครั้ง. แต่ละรายการของ call stack มีข้อมูลเกี่ยวกับตัวแปรแบบอัตโนมัติสำหรับโพรซีเดอร์หรือโปรแกรม และข้อมูลเกี่ยวกับรีซอร์สอื่นที่มีขอบเขตอยู่ใน call stack entry เช่น ตัวจัดการแบบ condition และ ตัวจัดการแบบ cancel.

ในงานแต่ละงานจะมีเพียง call stack เดียวเท่านั้น. การเรียกหนึ่งครั้งจะเพิ่มรายการใหม่เข้าไปใน call stack สำหรับโพรซีเดอร์หรือโปรแกรมที่ถูกเรียก และผ่านการควบคุมไปยังอ็อบเจ็กต์ที่ถูกเรียก. ส่วนรีเทิร์นจะเป็นการลบรายการในสแต็กและส่งการควบคุมกลับไปยังโพรซีเดอร์หรือโปรแกรมที่เรียกในลำดับก่อนหน้านี้ของรายการในสแต็ก.

ตัวอย่างของ Call Stack

รูปที่ 41 ในหน้า 118 ประกอบด้วยส่วนหนึ่งของ call stack กับโปรแกรม 2 โปรแกรม ได้แก่ โปรแกรม OPM (โปรแกรม A) และโปรแกรม ILE (โปรแกรม B). ในโปรแกรม B ประกอบด้วย 3 โพรซีเดอร์ คือ program entry procedure (PEP), user entry procedure (UEP) และโพรซีเดอร์อื่น (P1). แนวคิดของ PEP และ UEP อยู่ใน “โมดูลอ็อบเจ็กต์ (Module Object)” ในหน้า 14. ขั้นตอนของการเรียกมีดังนี้:

1. เรียกโปรแกรม A แบบ dynamic program call.
2. โปรแกรม A เรียกโปรแกรม B และผ่านการควบคุมไปยัง PEP ของโปรแกรม B. ซึ่งการเรียกจะเป็นแบบ dynamic program call.
3. PEP เรียก UEP. การเรียกแบบนี้เป็นแบบ การเรียกโพรซิเดอร์แบบสแตติก.
4. UEPเรียกโพรซีเดอร์ P1. การเรียกแบบนี้เป็นแบบ การเรียกโพรซิเดอร์แบบสแตติก.



รูปที่ 41. แสดง Dynamic Program Calls และ การเรียกโพรซิเดอร์แบบสแตติกบน Call Stack.

รูปที่ 41แสดง call stack ของตัวอย่างนี้. โดยที่ stack entry ตัวล่าสุดจะอยู่ที่ตำแหน่งล่างสุด. ซึ่งเป็นรายการ ที่กำลังทำงานอยู่ในปัจจุบัน. Call stack entry ปัจจุบันอาจทำสิ่งต่างๆ ดังนี้:

- เรียกโพรซีเดอร์หรือโปรแกรมอื่น ซึ่งจะเพิ่มรายการเข้าไปที่ส่วนล่างสุดของสแต็ก.
- ส่งคืนการควบคุมไปยังผู้เรียกหลังจากทำงานเสร็จแล้ว ซึ่งจะเป็นการลบรายการออกจากสแต็ก.

สมมุติว่า หลังจากโพรซีเดอร์ P1 ทำงานเสร็จแล้ว และไม่มีกระบวนการใดที่โปรแกรม B ต้องการอีก โพรซีเดอร์ P1 จะรีเทิร์นการควบคุมไปยัง UEP และ P1 จะถูกลบออกจากสแต็ก. จากนั้น UEP จะรีเทิร์นการควบคุมไปยัง PEP และ UEP ก็ถูกลบไป. ในที่สุด PEP จะรีเทิร์นการควบคุมไปยังโปรแกรม A และ PEP ก็จะถูกลบออกจากสแต็ก. มีเพียงโปรแกรม A เท่านั้นที่จะยังอยู่ในส่วนนี้ของ call stack. และโปรแกรม A จะทำงานต่อไปจากจุดที่สร้าง dynamic program call ไปยังโปรแกรม B.

การเรียกไปยังโปรแกรมและการเรียกไปยังโพรซีเดอร์

มีการเรียก 3 ประเภทที่สามารถถูกสร้างขึ้นได้ในระหว่างการรันของ ILE คือ dynamic program calls, การเรียกโพรซิเดอร์แบบสแตติก, และ procedure pointer calls.

เมื่อโปรแกรม ILE ถูก activate โพรซีเดอร์ทั้งหมดของโปรแกรมยกเว้น PEP จะสามารถใช้ได้กับการเรียกโพรซิเดอร์แบบสแตติก และ procedure pointer call. การ activate โปรแกรมเกิดขึ้นเมื่อโปรแกรมถูกเรียกโดย dynamic program call และยังไม่เคยเกิด activation. เมื่อโปรแกรมถูก activate เซอร์วิสโปรแกรมของทุกโปรแกรมที่ถูกรวมเข้ากับโปรแกรมนี้จะถูก activate ไปด้วย. โพร

ซีเดอร์ในเซอร์วิสโปรแกรมของ ILE สามารถถูกเข้าถึงได้โดย การเรียกโพรซิเดอร์แบบสแตติก หรือ procedure pointer call เท่านั้น (ไม่ใช่โดย dynamic program call).

การเรียกโพรซิเดอร์แบบสแตติก

การเรียกโพรซีเดอร์ของ ILE จะเป็นการเพิ่ม stack entry ใหม่เข้าไปที่ส่วนล่างสุดของ stack และผ่านการควบคุมไปยังโพรซีเดอร์ที่กำหนด. โดยมีตัวอย่างดังนี้:

1. การเรียกโพรซีเดอร์ในโมดูลเดียวกัน
2. การเรียกโพรซีเดอร์ในโมดูลคนละโมดูล แต่อยู่ในโปรแกรมหรือเซอร์วิสโปรแกรม ILE เดียวกัน.
3. การเรียกโพรซีเดอร์ที่ถูก export จากเซอร์วิสโปรแกรมของ ILE ที่อยู่ใน activation group เดียวกัน.
4. การเรียกโพรซีเดอร์ที่ถูก export จากเซอร์วิสโปรแกรมของ ILE ที่อยู่ใน activation group คนละกลุ่มกัน.

ในตัวอย่าง 1, 2, และ 3การเรียกโพรซิเดอร์แบบสแตติกจะไม่มีการเรียกข้ามขอบเขตของ activation group. ทำให้ความยาวของ call path ซึ่งมีผลต่อประสิทธิภาพของตัวอย่างทั้งสามจะเหมือนกัน. call path นี้จะสั้นกว่า call path ของ dynamic program call ที่ไปยังโปรแกรม ILE หรือ OPM. ในตัวอย่าง 4เป็นการเรียกข้ามขอบเขตของ activation group และมีโพรเซสเพิ่มเติมเพื่อสลับรีซอร์สของ activation group ด้วย. ทำให้ call path จะมีความยาวมากกว่า call path ของ การเรียกโพรซิเดอร์แบบสแตติก ที่อยู่ใน activation group แต่ก็ยังสั้นกว่า call path ของ dynamic program call.

สำหรับ การเรียกโพรซิเดอร์แบบสแตติก โพรซีเดอร์ที่ถูกเรียกต้องถูกรวมเข้ากับโพรซีเดอร์ที่เป็นผู้เรียกในขณะที่มีการรวม. ดังนั้นการเรียกแบบนี้จะไปยังโพรซีเดอร์เดิมเสมอ. ซึ่งจะตรงกันข้ามกับการเรียกโพรซีเดอร์โดยผ่านพอยน์เตอร์ ที่เป้าหมายของการเรียกสามารถเปลี่ยนไปในการเรียกแต่ละครั้ง.

Procedure Pointer Calls

Procedure pointer calls มีการเรียกโพรซีเดอร์แบบไดนามิก. ตัวอย่างคือ จากการเปลี่ยนแปลงอาร์เรย์ของชื่อโพรซีเดอร์หรือตำแหน่งที่อยู่ของโพรซีเดอร์ ทำให้คุณสามารถที่จะส่งการเรียกโพรซีเดอร์แบบไดนามิกไปยังโพรซีเดอร์ ต่างๆ ได้.

Procedure pointer call จะเพิ่มรายการให้กับ call stack ในลักษณะเดียวกันกับ การเรียกโพรซิเดอร์แบบสแตติก. โพรซีเดอร์ใดที่ถูกเรียกโดยการใช้ การเรียกโพรซิเดอร์แบบสแตติก ได้ก็สามารถถูกเรียกโดยใช้ procedure pointer call ได้เช่นกัน. ถ้าโพรซีเดอร์ที่ถูกเรียกอยู่ใน activation group เดียวกัน รีซอร์สที่ใช้ของ procedure pointer call ในลักษณะนี้จะเท่ากับการเรียกโพรซีเดอร์แบบสแตติก. นอกจากนี้ procedure pointer call ยังสามารถเข้าถึงโพรซีเดอร์ที่อยู่ในโปรแกรม ILE ใดๆ ที่ถูก activate แล้วได้ด้วย.

การผ่านค่าอากิวเมนต์ไปยังโพรซีเดอร์ของ ILE

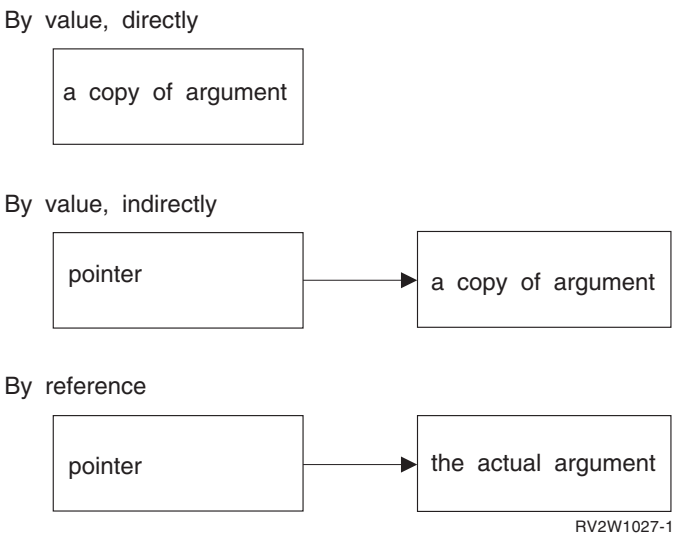
ใน ILE procedure call **อากิวเมนต์** คือเครื่องหมายแสดงค่าที่โพรซีเดอร์ที่เป็นผู้เรียกส่งไปยังโพรซีเดอร์ที่ระบุอยู่ในการเรียกนั้น. ภาษา ILE มีวิธีในการส่งผ่านอากิวเมนต์ 3 วิธี คือ:

ผ่านค่าโดยตรง (By Value, directly)
ค่าของอ็อบเจ็กต์ข้อมูลถูกใส่ไว้โดยตรงในรายการของอากิวเมนต์.

ผ่านค่าโดยอ้อม (By Value, indirectly)
ค่าของอ็อบเจ็กต์ข้อมูลถูกก็อปปี้ไว้ในตำแหน่งชั่วคราวจุดหนึ่ง. ตำแหน่งของตัวที่เป็นก็อปปี้ (พอยน์เตอร์) ถูกใส่ไว้ในรายการของอากิวเมนต์.

ผ่านค่าโดยการอ้างอิง (by reference)
พอยน์เตอร์ที่ชี้ไปยังอ็อบเจ็กต์ข้อมูลจะถูกใส่ไว้ในรายการของอากิวเมนต์. การเปลี่ยนแปลงที่ทำโดยโพรซีเดอร์ที่ถูกเรียกและที่เกิดกับอากิวเมนต์จะส่งผลกลับไปยังโพรซีเดอร์ที่เป็นตัวเรียก.

รูปที่ 42แสดงถึงรูปแบบของการส่งผ่านอากิวเมนต์แต่ละแบบ. ไม่ใช่ภาษา ILE ทุกภาษาที่จะสนับสนุนการส่งผ่านแบบผ่านค่าโดยตรง. สำหรับรูปแบบของการส่งผ่านอากิวเมนต์ที่สามารถใช้ได้ถูกอธิบายอยู่ในหนังสือ ILE HLL programmer's guides.



รูปที่ 42. Methods for Passing Arguments to ILE Procedures

ซีแมนทิกส์ของ HLL จะกำหนดว่าเมื่อไรที่ข้อมูลถูกส่งผ่านแบบผ่านค่าโดยตรงและเมื่อไรจะถูกผ่านค่าโดยการอ้างอิง. เช่น ILE C ผ่านและยอมรับอากิวเมนต์แบบผ่านค่าโดยตรง ในขณะที่อากิวเมนต์ของ ILE COBOL และ ILE RPG, ถูกผ่านค่าโดยการอ้างอิง. คุณต้องแน่ใจว่าโปรแกรมหรือโพรซีเดอร์ที่เรียกมีการส่งผ่านอากิวเมนต์เป็นไปตามลักษณะที่โพรซีเดอร์ที่ถูกเรียกคาดไว้. หนังสือ ILE HLL programmer's guides มีรายละเอียดเพิ่มเติมเกี่ยวกับการส่งผ่านอากิวเมนต์ไปยังภาษาที่แตกต่างกัน.

ในการเรียกโพรซิเดอร์แบบสแตติก จะมีจำนวนอากิวเมนต์ของ 400 ได้สูงสุด. ในแต่ละภาษาของ ILE อาจมีการจำกัดจำนวนสูงสุดของอากิวเมนต์ไว้. ภาษา ILE สนับสนุนรูปแบบในการส่งผ่านอากิวเมนต์ ดังนี้:

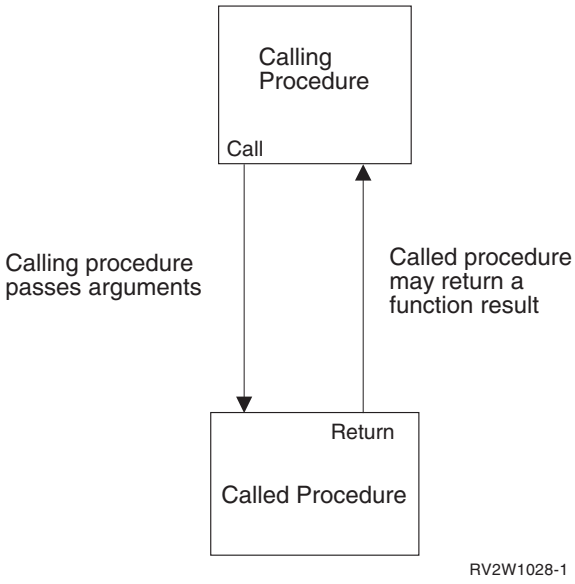
- ILE C และ C++ ส่งผ่านและยอมรับอากิวเมนต์จากค่าโดยตรง, การกระจายค่าของจำนวนเต็มและทศนิยมลอยตัวโดยค่าดีฟอลต์. อากิวเมนต์สามารถถูกส่งผ่านแบบ ไม่กระจายตัวโดยการ

ระบุ #pragma argument(name, nowiden). อากิวเมนต์ยังสามารถถูกส่งผ่านแบบผ่านค่าโดยอ้อมได้โดยการระบุอากิวเมนต์ไดเรกทีฟ #pragma สำหรับการเรียกฟังก์ชัน.

- ILE COBOL ผ่านและยอมรับอากิวเมนต์โดยค่า, โดยการอ้างอิง, หรือโดยค่าทางอ้อม.
- ILE RPG ผ่านและยอมรับอากิวเมนต์โดยค่า, หรือโดยการอ้างอิง. RPG ไม่ได้กระจายค่าจำนวนเต็ม และทศนิยมลอยตัวโดยค่าดีฟอลต์, แต่สิ่งนี้จัดเตรียมไว้สำหรับ พารามิเตอร์ให้ผ่านโดยค่า, โดยการโค้ด EXTPROC(*CWIDEN).
- ILE CL ผ่านและยอมรับอากิวเมนต์โดยการอ้างอิง และโดยกำหนดค่า.

ผลลัพธ์ของฟังก์ชัน

เพื่อสนับสนุนภาษาระดับสูง (HLL) ที่ยอมให้มีการกำหนดฟังก์ชัน (โพรซีเดอร์ที่สามารถส่งคืนอากิวเมนต์ที่เป็นผลลัพธ์) โมเดลจะถือว่าจะมีการรีเทิร์นอากิวเมนต์ที่เป็นผลลัพธ์ของฟังก์ชัน ดังใน รูปที่ 43. ดังที่อธิบายใน ILE HLL programmer's guides ว่า ภาษา ILE ที่สนับสนุนผลลัพธ์ของฟังก์ชัน ใช้กลไกทั่วไปในการรีเทิร์นค่าผลลัพธ์ของฟังก์ชัน.



รูปที่ 43. Program Call Argument Terminology

อากิวเมนต์ที่ถูกละเลย (Omitted Argument)

ภาษา ILE ทุกภาษาสามารถ ยอมให้มีการละเลยอากิวเมนต์ได้ ซึ่งช่วยให้คุณสามารถใช้กลไกผลสะท้อนสำหรับตัวจัดการเงื่อนไขของ ILE และโพรซีเดอร์แบบรันไทม์. ตัวอย่างเช่น เช่น การผ่านค่าอากิวเมนต์โดยการอ้างอิง (by reference) ของโพรซีเดอร์ ILE C หรือ an ILE bindable API บางครั้งคุณสามารถละเลยอากิวเมนต์ได้โดยผ่าน null pointer ไปแทน. สำหรับข้อมูลเกี่ยวกับการกำหนดอากิวเมนต์ที่ถูกละเลยในภาษา ILE ใดโดยเฉพาะ สามารถดูได้จาก programmer's guide ของภาษานั้นๆ. ส่วน *API* ของหมวด **Programming** ของ iSeries Information Center จะระบุว่าอากิวเมนต์ใดที่ข้ามได้สำหรับแต่ละ API.

สำหรับภาษา ILE ที่ไม่มีวิธีการสำหรับโพรซีเดอร์ที่ถูกเรียกในการทดสอบอากิวเมนต์ที่ถูกละเลย สามารถทำได้โดยการใช้ bindable API ชื่อ Test for Omitted Argument (CEETSTA).

Dynamic Program Calls

Dynamic program call เป็นการเรียกไปยังโปรแกรมอ็อบเจ็กต์. ตัวอย่างเช่น เมื่อคุณใช้คำสั่ง CALL ในภาษา CL นั่นคือคุณกำลังสร้าง dynamic program call.

โปรแกรม OPM ถูกเรียกโดยการใช้ dynamic program call. และถูกจำกัดให้สร้างได้แต่ dynamic program call เท่านั้น.

โปรแกรม ILE ก็ถูกเรียกแบบ dynamic program call. โพรซีเดอร์ที่อยู่ในโปรแกรม ILE ที่ถูก activate สามารถถูกเข้าถึงได้โดยการใช้ การเรียกโพรซิเดอร์แบบสแตติก หรือ procedure pointer call. ส่วนโปรแกรม ILE ที่ยังไม่ถูก activate จะต้องถูกเรียกแบบ dynamic program call.

ในทางตรงกันข้ามกับ การเรียกโพรซิเดอร์แบบสแตติก ซึ่งจะถูกรวมในเวลาคอมไพล์ symbol ของ dynamic program call จะถูก resolve เพื่อบอกตำแหน่งที่อยู่เมื่อมีการเรียกเกิดขึ้น. ผลที่ได้คือ dynamic program call จะใช้รีซอร์สของระบบมากกว่า การเรียกโพรซิเดอร์แบบสแตติก. ตัวอย่าง ของ dynamic program call ได้แก่:

- การเรียกใช้โปรแกรม ILE หรือโปรแกรม OPM
- การเรียกไปยัง non-bindable API

การเรียกแบบ dynamic program call ไปยังโปรแกรม ILE จะส่งผ่านการควบคุมไปยัง PEP ของโปรแกรมที่กำหนดและจะผ่านการควบคุมไปยัง UEP ของโปรแกรม. หลังจากโปรแกรมที่ถูกเรียกทำงานเสร็จแล้ว การควบคุมจะถูกส่งผ่านกลับไปยังคำสั่งที่ต่อจากคำสั่งที่เรียกโปรแกรม.

การผ่านค่าอากิวเมนต์บน Dynamic Program Call

การเรียกไปยังโปรแกรม ILE หรือ OPM(ตรงข้ามกับการใช้โพรซีเดอร์ ILE) มักจะส่งผ่านอากิวเมนต์โดยการอ้างอิง นั่นหมายความว่าโปรแกรมที่ถูกเรียกจะได้รับแอดเดรสของอากิวเมนต์.

เมื่อใช้ dynamic program call คุณต้องการทราบถึงวิธีการผ่าน อากิวเมนต์ที่คาดหวังจากโปรแกรมที่ถูกเรียกและวิธีการที่จะจำลองแบบมันหากมีความจำเป็น. ในการเรียกแบบ การเรียก dynamic program call จะมีจำนวนอากิวเมนต์ได้สูงสุดถึง 255 ตัว. ILE language แต่ละภาษาอาจมีการกำหนดจำนวนอากิวเมนต์สูงสุดที่มากกว่านี้. ภาษา ILE บางภาษาสนับสนุนฟังก์ชันในตัว CALLPGMV, ซึ่งอนุญาตให้ มีอากิวเมนต์สูงสุดได้ 16383 อากิวเมนต์. ข้อมูลเกี่ยวกับวิธีการใช้วิธีการส่งผ่านที่แตกต่างกัน อยู่ในหนังสือ ILE HLL programmer's guides.

ความเข้ากันได้ของข้อมูลหลายภาษา (Interlanguage Data Compatibility)

การเรียกของ ILE อนุญาตให้อากิวเมนต์ถูกส่งผ่านระหว่างโพรซีเดอร์ที่เขียนไว้ใน HLL คนละตัว. เพื่อให้การแบ่งใช้ข้อมูลระหว่าง HLL, ภาษา ILE บางภาษาจะมีชนิดของข้อมูลเพิ่มขึ้น. ตัวอย่างเช่น ILE COBOL มี USAGE PROCEDURE-POINTER เป็นข้อมูลชนิดใหม่.

เพื่อที่จะส่งผ่านอากิวเมนต์ระหว่าง HLL คุณจำเป็นต้องทราบรูปแบบของอากิวเมนต์ที่ HLL แต่ละตัวคาดว่าจะได้รับ. โพรซีเดอร์ที่เรียกจะต้องแน่ใจว่าอากิวเมนต์มีขนาดและชนิดตรงกับที่โพรซีเดอร์ที่ถูกเรียกคาดไว้. ตัวอย่างเช่น ฟังก์ชันของ ILE C อาจจะทำการ export จำนวนเต็มขนาด 4 ไบต์แม้ว่ารายการของพารามิเตอร์จะกำหนดไว้เป็น short integer (ขนาด 2 ไบต์) ก็ตาม. ตามสำหรับวิธีการทำตามข้อกำหนดเกี่ยวกับชนิดของข้อมูลในการผ่านค่าอากิวเมนต์จะอยู่ในหนังสือ ILE HLL programmer's guides.

ไวยากรณ์สำหรับการผ่านค่าอากิวเมนต์ในแอ็พพลิเคชันที่เขียนด้วยหลายภาษา

ภาษา ILE บางภาษาเตรียมไวยากรณ์สำหรับการส่งผ่านอากิวเมนต์ไปยังโพรซีเดอร์ในภาษา ILE อื่น. ตัวอย่างเช่น, ILE C ได้ให้อากิวเมนต์ #pragma เพื่อ ส่งผ่านค่าอากิวเมนต์ไปยังโพรซีเดอร์ ILE อื่นๆโดยกำหนดค่าทางอ้อม; RPG มีค่าพิเศษ สำหรับคีย์เวิร์ดต้นแบบ EXTPROC .

Operational Descriptors

Operational descriptors อาจมีประโยชน์สำหรับคุณถ้าคุณเขียนโพรซีเดอร์หรือ API ที่สามารถรับอากิวเมนต์จากโพรซีเดอร์ที่เขียนใน HLL คนละตัว. **Operational descriptors** ให้ข้อมูลเชิงบรรยายกับโพรซีเดอร์ที่ถูกเรียก ในกรณีที่โพรซีเดอร์ที่ถูกเรียกไม่สามารถคาดหมายรูปแบบของอากิวเมนต์ได้ (ตัวอย่างเช่น ชนิดของ string ที่ต่างกัน). ข้อมูลที่เพิ่มขึ้นทำให้โพรซีเดอร์ตีความอากิวเมนต์ได้ถูกต้อง.

อากิวเมนต์จะให้ค่าและ operational descriptors จะให้ข้อมูลเกี่ยวกับขนาดและชนิดของอากิวเมนต์. ตัวอย่างเช่น ความยาวของสตริงอัชระและชนิดของสตริง.

Operational Descriptor ทำให้ฟังก์ชันบริการ เช่น bindable API ไม่ต้องการจะมีการรวมโมดูลที่แตกต่างกันสำหรับ HLL แต่ละตัว และ HLL ไม่จำเป็นต้องเลียนแบบชนิดของข้อมูลที่ไม่เหมือนกัน. ILE bindable API ใช้ operational descriptors ในการปรับปรุงส่วนที่ขาดไปของชนิดข้อมูลที่เป็นสตริง ระหว่าง HLL. การทำงานของ Operational Descriptor จะถูกซ้อนไว้จากผู้ใช้ API.

Operational descriptors สนับสนุนซีแมนทิกส์ของ HLL ในขณะที่มองไม่เห็นโพรซีเดอร์ที่ไม่ใช้หรือคาดเดาไม่ได้. ภาษา ILE แต่ละภาษาสามารถใช้ชนิดของข้อมูลที่เหมาะสมกับภาษานั้น. นั้น คอมไพเลอร์ของภาษา ILE แต่ละตัวจะให้การสร้าง operational descriptors อย่างน้อยหนึ่งวิธี. สำหรับข้อมูลเพิ่มเติมของซีแมนทิกส์ของ HLL สำหรับ operational descriptors ดูได้ในคู่มืออ้างอิงสำหรับ ILE HLL.

Operational descriptors จะต่างจาก Data Descriptors ที่คุณคุ้นเคย. เช่น Operation Descriptor ไม่มีส่วนสัมพันธ์กับข้อมูล หรือไฟล์แบบกระจาย.

ข้อกำหนดของ Operational Descriptor

คุณต้องใช้ operational descriptors เมื่อถูกคาดหวังโดยโพรซีเดอร์ที่ถูกเรียกที่เขียนใน ILE language ที่แตกต่างกัน และเมื่อถูกคาดหวังโดย ILE bindable API. โดยทั่วไปแล้ว bindable API

ต้องการ descriptor สำหรับสตริงอากิวเมนต์ส่วนใหญ่. ข้อมูล เกี่ยวกับ bindable API ในส่วน *API* ของหมวด **Programming** ของ iSeries Information Center จะระบุว่า bindable API ใด ต้องการ operational descriptor.

การขาดไปของ Descriptor ที่จำเป็น

การขาดหายไปของ descriptor ที่จำเป็นถือเป็นข้อผิดพลาด. ถ้าโพรซีเดอร์ต้องการ descriptor สำหรับพารามิเตอร์เฉพาะ ความต้องการนี้เป็นส่วนหนึ่งของอินเตอร์เฟสสำหรับโพรซีเดอร์นี้. ถ้าไม่มี descriptor ที่ต้องการก็จะเกิดความล้มเหลวในขณะรัน.

การปรากฏของ Descriptor ที่ไม่จำเป็น

การปรากฏของ descriptor ที่ไม่เป็นที่ต้องการ จะไม่เกิดการรบกวนต่อการเข้าถึงโพรซีเดอร์ที่ถูกเรียกไปยังอากิวเมนต์. ถ้า operational descriptor ไม่เป็นที่ต้องการหรือถูกคาดหวังโพรซีเดอร์ที่ถูกเรียกก็จะไม่สนใจ.

หมายเหตุ: descriptor สามารถเป็นอุปสรรคต่อการสื่อสารระหว่างภาษาได้ เมื่อมันถูกสร้างขึ้นโดยไม่สนใจถึงความต้องการใช้งาน. เพราะ descriptor จะเพิ่มความยาวของ call path ซึ่งจะเป็นตัวลดประสิทธิภาพในการทำงานลง.

Bindable API สำหรับการเข้าถึง Operational Descriptor

โดยปกติแล้ว descriptor จะถูกเข้าถึงโดยตรงจากโพรซีเดอร์ที่ถูกเรียก เนื่องจาก semantics ของ HLL ที่เขียนโพรซีเดอร์นั้น. เมื่อโพรซีเดอร์ถูกโปรแกรมให้คาดหวัง operational descriptor โปรแกรมเมอร์มักจะไม่ต้องการการปฏิบัติแบบอื่นๆ อีก. ในบางครั้งโพรซีเดอร์ที่ถูกเรียกยังต้องการการกำหนด descriptor ที่ต้องการให้ถูกแสดงก่อนที่จะถูกเข้าถึง. เพื่อจุดประสงค์นี้จึงมีการใช้ bindable APIs ดังนี้:

- Retrieve Operational Descriptor Information (CEEDOD) bindable API
- Get String Information (CEESGI) bindable API

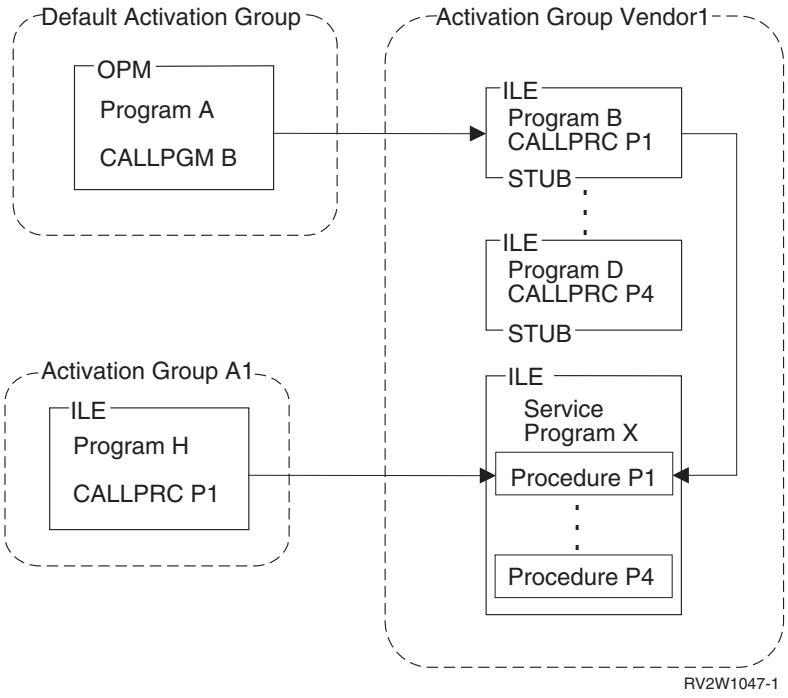
การสนับสนุน API ของ OPM และ ILE

เมื่อคุณพัฒนาฟังก์ชันใหม่ใน ILE หรือแปลงแอ็พพลิเคชันที่มีอยู่แล้วไปเป็น ILE คุณอาจยังต้องการสนับสนุน call-level API จาก OPM. หัวข้อนี้จะอธิบายถึงเทคนิคหนึ่งที่อาจใช้ในการสร้างการสนับสนุนทั้ง 2 แบบ ในขณะที่ยังรักษาแอ็พพลิเคชันของคุณไว้ใน ILE.

เซอร์วิสโปรแกรม ILE มีวิธีการในการพัฒนาและส่ง bindable API ที่อาจถูกเข้าถึงโดยภาษา ILE ทุกภาษา. เพื่อให้การทำงานที่เหมือนกันกับโปรแกรม OPM คุณต้องพิจารณาความจริงที่ว่า เซอร์วิสโปรแกรม ILE ไม่สามารถถูกเรียกโดยตรงจากโปรแกรม OPM.

เทคนิคที่ใช้คือการพัฒนา stub ของโปรแกรม ILE สำหรับ bindable API แต่ละตัวที่คุณวางแผนที่จะสนับสนุน คุณอาจต้องการตั้งชื่อ bindable API ให้เหมือนกับ stub ของโปรแกรม ILE หรือคุณอาจใช้ชื่อที่ต่างกันก็ได้. stub ของโปรแกรม ILE แต่ละโปรแกรมจะมี การเรียกโพรซิเดอร์แบบสแตติกไปยัง bindable API ที่แท้จริง.

ตัวอย่างของเทคนิคนี้แสดงใน รูปที่ 44.



รูปที่ 44. แสดงการสนับสนุน API ของ OPM และ ILE

จากรูป โปรแกรม B ถึง D คือ แกนของโปรแกรม ILE. เซอร์วิสโปรแกรม X ประกอบด้วยการทำงานที่แท้จริงของ bindable API แต่ละตัว. stub ของโปรแกรม ILE และเซอร์วิสโปรแกรมแต่ละตัวถูกกำหนดให้มีชื่อของ activation group เหมือนกัน. ในตัวอย่างนี้ activation group ชื่อ VENDOR1 เป็นตัวที่ถูกเลือก.

Activation group VENDOR1 ถูกสร้างขึ้นโดยระบบเมื่อจำเป็นต้องใช้งาน. dynamic program call จากโปรแกรม A (โปรแกรม OPM) สร้าง activation group ในการเรียกครั้งแรกของโปรแกรม OPM. ส่วนการเรียกโพรซิเดอร์แบบสแตติก จากโปรแกรม H (โปรแกรม ILE) จะสร้าง activation group เมื่อโปรแกรม H ถูก activate. เมื่อมี activation group เกิดขึ้นแล้ว มันอาจถูกใช้งานจากทั้งโปรแกรม A และโปรแกรม H.

คุณควรเขียนการทำงานของ API ของคุณในโพรซีเดอร์ ILE (ในตัวอย่างคือ โพรซีเดอร์ P1). โพรซีเดอร์นี้อาจถูกเรียกโดยตรงโดยผ่าน procedure call หรือถูกเรียกโดยอ้อมโดยผ่าน dynamic program call. คุณต้องไม่สร้างฟังก์ชันใดๆ เช่น การส่ง exception message ที่ขึ้นอยู่กับโครงสร้างของ call stack ที่มีแบบเฉพาะ. การรีเทิร์น แบบปกติจาก stub ของโปรแกรมหรือการสร้างโพรซีเดอร์จะทิ้ง activation group ไว้ในงานเพื่อการใช้ในภายหลัง. คุณสามารถสร้างโพรซีเดอร์ API ของคุณด้วยความรู้ที่ว่า ขอบเขตการควบคุม ถูกสร้างขึ้นสำหรับ stub ของโปรแกรมและการสร้างโพรซีเดอร์ในการเรียกแต่ละครั้ง. HLL end verbs จะลบ activation group ที่การเรียกมีต้นกำเนิดมาจากโปรแกรม OPM หรือ โปรแกรม ILE

บทที่ 8. การบริหารหน่วยเก็บข้อมูล

ระบบปฏิบัติการจะให้การสนับสนุนในเรื่องที่เก็บข้อมูลสำหรับภาษา ILE ระดับสูง. การสนับสนุนนี้เป็นการตัดความต้องการของตัวจัดการที่เก็บข้อมูลเฉพาะ สำหรับสภาพแวดล้อมในขณะรันของแต่ละภาษา. ซึ่งจะเป็นการหลีกเลี่ยงความไม่เข้ากันระหว่างตัวจัดการที่เก็บข้อมูลกับกลไกในภาษาระดับสูงแต่ละภาษา.

ระบบปฏิบัติการมีที่เก็บข้อมูลอยู่ 3 แบบคือ แบบอัตโนมัติ, แบบสแตติก และแบบไดนามิก ซึ่งจะถูกใช้โดยโปรแกรมและโพรซีเดอร์ในขณะรัน. หน่วยเก็บข้อมูลแบบอัตโนมัติและแบบสแตติกจะถูกจัดการโดยระบบปฏิบัติการ. ความต้องการที่เก็บข้อมูลแบบอัตโนมัติและแบบคงที่ จะถูกรับรู้เมื่อมีการคอมไพล์โดยการประกาศค่าของตัวแปรในโปรแกรม. ส่วนที่เก็บข้อมูลแบบไดนามิก จะถูกจัดการโดยโปรแกรมหรือโพรซีเดอร์. และความต้องการหน่วยความจำไดนามิกจะเกิดขึ้นในขณะรันเท่านั้น.

เมื่อโปรแกรมเริ่มต้นทำงาน หน่วยความจำแบบสแตติกสำหรับตัวแปรของโปรแกรมจะถูกจัดสรรและกำหนดค่าเริ่มต้น.

เมื่อโปรแกรมหรือโพรซีเดอร์เริ่มต้นรัน หน่วยเก็บข้อมูลแบบอัตโนมัติจะถูกกำหนดขึ้น. สแต็กสำหรับเก็บตัวแปรจะขยายตัวขึ้นเมื่อโปรแกรมหรือโพรซีเดอร์ถูกเพิ่มเข้าไปใน call stack.

ในขณะที่รันโปรแกรมหรือโพรซีเดอร์นั้น หน่วยเก็บข้อมูลแบบไดนามิกจะถูกกำหนดขึ้นภายใต้การควบคุมของโปรแกรม. หน่วยเก็บข้อมูลนี้จะถูกขยายออกเมื่อมีความต้องการที่เก็บข้อมูลเพิ่มมากขึ้น. คุณสามารถควบคุมที่เก็บข้อมูลแบบไดนามิกได้. ส่วนที่จะกล่าวต่อไปในบทนี้จะเน้นในเรื่องหน่วยเก็บข้อมูลแบบไดนามิก และวิธีการควบคุม.

Single-Level Store Heap

Heap heap คือพื้นที่ของหน่วยเก็บข้อมูลที่ใช้สำหรับจัดสรรพื้นที่ให้กับหน่วยเก็บข้อมูลแบบไดนามิก.จำนวนของหน่วยเก็บข้อมูลแบบ ไดนามิก ที่แอ็พพลิเคชันต้องการจะขึ้นอยู่กับ ข้อมูลที่ถูกโพรเซสโดยโปรแกรมและโพรซีเดอร์ที่ใช้ Heap นั้น. ระบบปฏิบัติการอนุญาตให้มีการใช้ heap ได้หลายตัวซึ่งสามารถสร้างและยกเลิกได้แบบไดนามิก. คำสั่ง ALCHSS จะเป็นคำสั่งที่บังคับให้ใช้ single-level store เสมอ. แต่ในบางภาษาก็สนับสนุนการใช้ Teraspace สำหรับหน่วยความจำไดนามิกด้วย.

คุณลักษณะของ Heap

Heap แต่ละตัวมีคุณลักษณะดังนี้:

- ระบบกำหนดค่าเฉพาะของหมายเลขประจำ heap ให้กับ heap แต่ละตัวภายใน activation group.

  Heap Identifier สำหรับ Heap แบบดีฟอล์ตมักเป็นค่า 0 เสมอ.

  Storage management-bindable API ซึ่งถูกเรียกโดยโปรแกรมหรือโพรซีเดอร์ใช้ Heap Identifier ในการระบุ Heap ที่มันทำงานด้วย. bindable API จะต้องรันอยู่ภายใน activation group ที่เป็นเจ้าของ heap เท่านั้น.

- activation group ที่สร้าง heap จะเป็นเจ้าของ heap นั้นด้วย.

  เนื่องจาก activation group เป็นเจ้าของ heap ดังนั้นช่วงอายุของ heap ก็จะไม่ยาวเกินอายุของ activation group ที่เป็นเจ้าของมัน . Heap Identifier มีความหมายและมีค่าเฉพาะภายใน activation group ที่เป็นเจ้าของเท่านั้น.
- คุณสามารถขยายขนาดของ heap เพื่อตอบสนองการร้องขอการจัดสรรพื้นที่ได้.

  ได้ ขนาดที่ใหญ่ที่สุดของ heap คือ 4 กิกะไบต์ลบด้วย 512 กิโลไบต์. ถ้าขนาดของการจัดสรรพื้นที่ใน 1 ครั้ง มีค่าไม่เกิน 128,000.
- ขนาดที่ใหญ่ที่สุดของการจัดสรรพื้นที่จาก heap หนึ่งครั้งจะมีค่าไม่เกิน 16 เมกกะไบต์ลบด้วย 64 กิโลไบต์.

Default Heap

คำร้องขอสำหรับหน่วยเก็บข้อมูลแบบไดนามิกลำดับแรก จาก default heap ที่อยู่ใน activation group มีผลต่อการสร้าง default heap จากการจัดสรรพื้นที่ในการเก็บข้อมูล. ถ้าพื้นที่เก็บข้อมูลใน heap ไม่เพียงพอที่จะตอบสนองคำร้องขอสำหรับหน่วยเก็บข้อมูลแบบไดนามิกแล้ว ระบบจะขยาย heap และจัดสรรพื้นที่เก็บข้อมูลเพิ่มขึ้น.

หน่วยเก็บข้อมูลแบบไดนามิกที่ถูกกำหนดไว้จะยังคงอยู่จนกว่าจะถูกปลดปล่อยหรือระบบจะยกเลิก heap นั้น. default heap จะถูกยกเลิกเมื่อ activation group ที่เป็นเจ้าของสิ้นสุดการทำงานเท่านั้น.

โปรแกรมใน activation group เดียวกัน จะมีการแบ่งใช้หน่วยเก็บข้อมูลแบบไดนามิกที่เกิดจากการจองพื้นที่ของ default heap โดยอัตโนมัติ. อย่างไรก็ตาม คุณสามารถแยกหน่วยเก็บข้อมูลแบบไดนามิกที่ถูกใช้โดยบางโปรแกรมหรือบางโพรซีเดอร์ที่อยู่ใน activation group ได้. โดยการสร้าง heap ขึ้นตัวหนึ่งหรือมากกว่านั้น.

Heap ที่สร้างโดยผู้ใช้ (user-created heap)

คุณสามารถสร้างและยกเลิก heap ได้โดยใช้ bindable API. ซึ่งทำให้คุณสามารถจัดการ heap และหน่วยเก็บข้อมูลแบบไดนามิก ที่ถูกจองพื้นที่จาก heap เหล่านั้นได้.

ตัวอย่างเช่น ระบบอาจจะมีการแบ่งใช้หน่วยเก็บข้อมูลแบบไดนามิก ที่ถูกจัดสรรพื้นที่ใน heap ที่สร้างโดยผู้ใช้ของโปรแกรมใน activation group. การแบ่งใช้หน่วยเก็บข้อมูลแบบไดนามิก ขึ้นอยู่กับหมายเลขประจำ heap ที่ถูกอ้างถึงโดยโปรแกรม. คุณสามารถใช้ heap ตั้งแต่ 1 ตัวขึ้นไป เพื่อหลีกเลี่ยงการแบ่งใช้หน่วยเก็บข้อมูลแบบไดนามิกโดยอัตโนมัติ. ซึ่งจะทำให้คุณสามารถแยกข้อมูลออกเป็นกลุ่มได้แบบลอจิคัล. เหตุผลเพิ่มเติมในการใช้ heap ที่สร้างโดยผู้ใช้ได้แก่:

- คุณสามารถจับกลุ่มอ็อบเจ็กต์หน่วยเก็บข้อมูลเข้าด้วยกันเพื่อตอบสนองความต้องการได้ในครั้งเดียว. เมื่อคุณทำตามความต้องการนั้นแล้ว คุณสามารถยกเลิกการใช้พื้นที่ของหน่วยเก็บข้อมูลแบบไดนามิกที่เคยกำหนดไว้โดยการเรียก (call) เพียงครั้งเดียวไปยัง bindable API ชื่อ Discard Heap (CEEDSHP). กระบวนการนี้จะยกเลิกการใช้พื้นที่ของหน่วยเก็บข้อมูลแบบไดนามิกและ heap. ซึ่งวิธีนี้จะทำให้หน่วยเก็บข้อมูลแบบไดนามิกว่างสำหรับคำร้องขออื่น.
- คุณสามารถยกเลิกการใช้พื้นที่ของหน่วยเก็บข้อมูลแบบไดนามิกหลายจุดได้ในครั้งเดียว โดยการใช้ bindable API ชื่อ Mark Heap (CEEMKHP) และ Release Heap (CEERLHP) bindable

API. CEEMKHP ยอมให้คุณทำเครื่องหมายที่ heap. เมื่อคุณพร้อมที่จะยกเลิกการใช้พื้นที่ที่ทำขึ้นไว้เป็นกลุ่ม เมื่อ heap ถูกทำเครื่องหมายแล้ว ให้ใช้ bindable API CEERLHP เพื่อยกเลิกการใช้พื้นที่เหล่านั้น. การใช้ฟังก์ชัน mark และ release เป็นการปล่อย heap ไว้โดยไม่เปลี่ยนแปลง แต่ทำให้พื้นที่ของหน่วยเก็บข้อมูลแบบไดนามิกที่ถูกกำหนดไว้ว่างลง. ด้วยวิธีนี้คุณสามารถหลีกเลี่ยงโอเวอร์เฮดของระบบที่เกิดจากการสร้าง heap โดยการใช้ heap ที่มีอยู่แล้วในการตอบสนองความต้องการใช้หน่วยเก็บข้อมูลแบบไดนามิก.

- ท ความต้องการหน่วยเก็บข้อมูลของคุณ อาจไม่ตรงกับแอ็ตทริบิวต์ที่กำหนดค่าของ default heap. ตัวอย่าง เช่น ขนาดเริ่มต้นของ default heap คือ 4 กิโลไบต์. หากคุณต้องการจองพื้นที่สำหรับ หน่วยเก็บข้อมูลแบบไดนามิกที่รวมกันแล้วมีขนาดเกิน 4 กิโลไบต์. กิโลไบต์ คุณก็สามารถทำได้โดยการสร้าง heap 1 ตัวที่มีขนาดเริ่มต้นใหญ่กว่า 4 กิโลไบต์. ด้วยวิธีนี้จะเป็นการช่วยลดโอเวอร์เฮดของระบบที่เกิดขึ้น ทั้งในกรณีที่ขยาย heap และการเข้าถึง heap ที่ขยายออกไป. ในทำนองเดียวกัน คุณก็จะสามารถมี heap ที่ขยายตัวเกิน 4 กิโลไบต์ได้เช่นกัน. สำหรับข้อมูลเกี่ยวกับการกำหนดขนาดของ heap ดูหัวข้อ “Heap Allocation Strategy”และรายละเอียดของแอ็ตทริบิวต์ของ heap.

คุณอาจมีเหตุผลอื่นในการใช้ heap หลายๆ ตัวมากกว่าการใช้ bindable API ที่จัดการเกี่ยวกับ default heap. ทำให้คุณสามารถที่จะจัดการ heap ที่คุณสร้างขึ้นและจัดการหน่วยเก็บข้อมูลแบบไดนามิกที่จองพื้นที่ใน heap เหล่านั้น. ไอบีเอ็ม ไอบีเอ็มมีข้อมูลออนไลน์ที่อธิบายเกี่ยวกับ storage management–bindable API. โปรดดูส่วน *API* ซึ่งอยู่ในหมวด **Programming** ของ iSeries Information Center.

การสนับสนุน Single-Heap

ภาษาที่ไม่สนับสนุนการใช้ heap หลายตัวจะใช้ heap ที่เป็น default heap. คุณไม่สามารถใช้ bindable API Discard Heap (CEEDSHP), Mark Heap (CEEMKHP), หรือ Release Heap (CEERLHP) กับ default heap ได้. คุณสามารถยกเลิกพื้นที่เก็บข้อมูลแบบไดนามิกที่ถูกกำหนดโดย default heap โดยการใช้ free operation หรือการสิ้นสุดการทำงานของ activation group ที่เป็นเจ้าของ heap นั้น.

ข้อจำกัดในการใช้ default heap ช่วยป้องกันการยกเลิกการจองพื้นที่ของหน่วยเก็บข้อมูลแบบไดนามิก อย่างไม่ตั้งใจในแอ็พพลิเคชันที่มีหลายภาษาผสมกันได้. จำไว้ว่าการปลดปล่อย heap และยกเลิก heap เป็นสิ่งที่ไม่ปลอดภัยสำหรับแอ็พพลิเคชันขนาดใหญ่ ที่มีการใช้โค้ดเดิมซ้ำกับหน่วยเก็บข้อมูลหลายๆ ที่. จำไว้ว่าต้องไม่ใช้กระบวนการปลดปล่อย heap กับ default heap. ซึ่งจะเป็นสาเหตุให้หลายส่วนของแอ็พพลิเคชันที่ mark ไว้ซึ่งสามารถทำงานได้อย่างถูกต้องเมื่อใช้งานแต่ละส่วนแยกกันอาจล้มเหลวเมื่อมาใช้งานร่วมกัน.

Heap Allocation Strategy

แอ็ตทริบิวต์ที่เกี่ยวข้องกับ default heap ถูกกำหนดโดยระบบผ่านทาง default allocation strategy. ซึ่งจะกำหนดแอ็ตทริบิวต์ต่างๆ เช่น heap ที่สร้างขึ้นมีขนาด 4 กิโลไบต์ และขนาดของส่วนที่ขยายเพิ่มเติมเท่ากับ 4 กิโลไบต์. คุณไม่สามารถเปลี่ยนค่าดีฟอลต์ของ allocation strategy ได้.

อย่างไรก็ตาม คุณสามารถควบคุม heap ที่สร้างขึ้นโดยใช้ API ชื่อ Create a Heap (CEECRHP). คุณยังสามารถกำหนด allocation strategy สำหรับ heap ที่สร้างขึ้นโดยใช้ API ชื่อ Define Heap

Allocation Strategy (CEE4DAS). เมื่อคุณสร้าง heap ขึ้น แอ็ตทริบิวต์ของ heap จะมาจาก allocation strategy ตามที่คุณกำหนดไว้. ด้วยวิธีนี้คุณสามารถกำหนด allocation strategy เฉพาะของ heap ที่คุณสร้างขึ้น.

คุณสามารถใช้ bindable API ที่ชื่อ CEECRHP โดยไม่มีการกำหนด allocation strategy. ในกรณีนี้ heap ถูกกำหนดโดยแอ็ตทริบิวต์ของ _CEE4ALC allocation strategy type. type ซึ่งจะระบุขนาดของ heap ที่สร้างขึ้นว่าเท่ากับ 4 กิโลไบต์และขนาดของส่วนที่ขยายเพิ่มเท่ากับ 4 กิโลไบต์. _CEE4ALC allocation strategy type ประกอบด้วยแอ็ตทริบิวต์ ดังนี้:

```
Max_Sngl_Alloc = 16MB - 64K  /* maximum size of a single allocation */
Min_Bdy        = 16          /* minimum boundary alignment of any allocation */
Crt_Size       = 4K          /* initial creation size of the heap */
Ext_Size       = 4K          /* the extension size of the heap */
Alloc_Strat    = 0           /* a choice for allocation strategy */
No_Mark        = 1           /* a group deallocation choice */
Blk_Xfer       = 0           /* a choice for block transfer of a heap */
PAG            = 0           /* a choice for heap creation in a PAG */
Alloc_Init     = 0           /* a choice for allocation initialization */
Init_Value     = 0x00        /* initialization value */
```

แอ็ตทริบิวต์ดังกล่าวอธิบายถึงโครงสร้างของ _CEE4ALC allocation strategy type. ไอบีเอ็ม ได้จัดเตรียมข้อมูลแบบออนไลน์ที่อธิบายถึงแอ็ตทริบิวต์ต่างๆ ของ _CEE4ALC ไว้ให้แล้ว. โปรดดูส่วน *API* ที่พบในหมวด **Programming** ของ iSeries Information Center.

อินเตอร์เฟสของหน่วยเก็บข้อมูลHeap แบบ Single-Level

Bindable API ได้ถูกจัดเตรียมไว้สำหรับการดำเนินการเกี่ยวกับ heap. คุณสามารถสร้างแอ็พพลิเคชันโดยใช้ทั้ง Bindable API และฟังก์ชันที่มีอยู่ในภาษานั้นๆพร้อมๆ กัน หรือเลือกใช้อย่างใดอย่างหนึ่งก็ได้.

Bindable API จะแบ่งออกเป็นประเภทต่างๆ ได้ดังนี้:

- การดำเนินการ heap เบื้องต้น. การดำเนินการเหล่านี้สามารถใช้ได้ทั้งใน heap แบบดีฟอล์ต หรือแบบผู้ใช้กำหนดเอง.
  คำสั่ง Free Storage (CEEFRST) ใน Bindable API ใช้สำหรับลบหน่วยเก็บข้อมูลที่จองไว้แล้วก่อนหน้านี้.
  คำสั่ง Get Heap Storage (CEEGTST) ใน Bindable API ใช้สำหรับจัดสรรหน่วยเก็บข้อมูลภายใน heap.
  คำสั่ง Reallocate Storage (CEECZST) ใน Bindable API ใช้สำหรับเปลี่ยนขนาดของหน่วยเก็บข้อมูลที่จองไว้แล้วก่อนหน้านี้.
- การดำเนินการ heap เพิ่มเติม. การดำเนินการเหล่านี้สามารถใช้ได้เฉพาะ heap แบบผู้ใช้กำหนดเองเท่านั้น.
  คำสั่ง Create Heap (CEECRHP) ใน Bindable API ใช้สำหรับสร้าง heap ใหม่.
  คำสั่ง Discard Heap (CEEDSHP) ใน Bindable API ใช้สำหรับลบล้าง heap ที่สร้างไว้แล้ว.
  คำสั่ง Mark Heap (CEEMKHP) ใน Bindable API ใช้สำหรับคืนค่าโทเค็นที่สามารถใช้จำแนกหน่วยเก็บข้อมูล heap ที่ถูกลบล้างโดยคำสั่ง CEERLHP.

คำสั่ง Release Heap (CEERLHP) ใน Bindable API ใช้สำหรับลบล้างหน่วยเก็บข้อมูลทั้งหมดที่อยู่ใน heap.

- Heap allocation strategies

  คำสั่ง Define Heap Allocation Strategy (CEE4DAS) ใน Bindable API ใช้สำหรับกำหนด Allocation Strategy ซึ่งเป็นตัวกำหนดแอ็ตทริบิวต์ของ heap ที่สร้างด้วยคำสั่ง CEECRHP.

ไอบีเอ็มได้จัดเตรียมข้อมูลแบบออนไลน์เกี่ยวกับ Bindable API ด้านการบริหารหน่วยเก็บข้อมูลไว้ให้แล้ว. โปรดดูส่วน *API* ของหมวด **Programming** ของ iSeries Information Center.

การสนับสนุน Heap

ตามที่กำหนดไว้เป็นดีฟอล์ต คุณสามารถใช้คำสั่ง `malloc`, `calloc`, `realloc` และ `new` ในการจัดการกับหน่วยความจำไดนามิก ซึ่งหน่วยความจำนั้นจะเป็นชนิดเดียวกับโมเดลหน่วยความจำของโปรแกรมรากที่อยู่ใน Activation Group. อย่างไรก็ตาม เมื่อมีการใช้โมเดลหน่วยความจำแบบ single-level คุณก็สามารถใช้หน่วยความจำในแบบTeraspace ได้โดยใช้อินเตอร์เฟสเหล่านี้ถ้าหาอ็อพชันของคอมไพเลอร์ TERASPACE(*YES *TSIFC) compiler ถูกระบุไว้. คำสั่งต่างๆ จะคล้ายคลึงกับหน่วยความจำแบบ single-level มาก ซึ่งคุณสามารถใช้คำสั่งใน Bindable API ในการทำงานกับ Teraspace ได้ เช่น `_C_TS_malloc`, `_C_TS_free`, `_C_TS_realloc` และ `_C_TS_calloc`.

สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับวิธีใช้หน่วยความจำ Teraspace ให้ดูที่ บทที่ 4, “หน่วยเก็บข้อมูลแบบ Teraspace และ Single-level”, ในหน้า 57.

ถ้าคุณต้องการใช้ทั้งคำสั่งด้านหน่วยความจำใน Bindable API อย่าง CEExxxx และฟังก์ชัน ILE C `malloc()`, `calloc()`, `realloc()`, และ `free()` ก็ควรปฏิบัติตามกฎต่างๆ เหล่านี้:

- หน่วยความจำไดนามิกที่ถูกกำหนดโดยผ่านฟังก์ชันชื่อ malloc(), calloc(), และ realloc() ของภาษาซีจะไม่สามารถถูกยกเลิกหรือจองพื้นที่ใหม่โดยการใช้ bindable API ชื่อ CEEFRST และ CEECZST ได้.
- หน่วยความจำไดนามิกที่ถูกกำหนดโดย bindable API ชื่อ CEEGTST สามารถยกเลิกการใช้พื้นที่ได้ด้วยฟังก์ชัน free().
- หน่วยความจำไดนามิกที่เริ่มต้นถูกจองพื้นที่ด้วย bindable API ชื่อ CEEGTST สามารถถูกจองพื้นที่ใหม่ได้ด้วยฟังก์ชัน realloc().

ในภาษาอื่นๆ เช่น COBOL และ RPG ไม่มีการใช้โมเดลหน่วยความจำแบบ heap. ดังนั้นภาษาเหล่านี้สามารถเข้าถึงโมเดลหน่วยความจำไดนามิกของ ILE ได้โดยผ่านทาง Bindable API.

ในภาษา RPG มีคำสั่ง ALLOC, REALLOC และ DEALLOC, และ ฟังก์ชันติดมาด้วย %ALLOC และ %REALLOC สำหรับการเข้าถึง default heap. ในภาษา RPG สนับสนุนการใช้งานคำสั่ง CEEGTST, CEECZST, และ CEEFRST ผ่านทาง bindable APIs.

บทที่ 9. การจัดการ Exception และ Condition

บทนี้จะให้รายละเอียดเพิ่มเติมของการจัดการ exception และ condition. ก่อนที่คุณจะอ่านบทนี้ ควรอ่านแนวคิดระดับสูงที่อธิบายอยู่ในหัวข้อ “การจัดการข้อผิดพลาด (Error Handling)” ในหน้า 45 เสียก่อน.

สถาปัตยกรรมของข้อความ Exception ของ OS/400 ถูกใช้ในการสร้างการจัดการ exception และ condition. มีหลายกรณีที่การจัดการ exception และการจัดการ condition มีการโต้ตอบกัน. เช่น ตัวจัดการ ILE condition ที่เรจิสเตอร์ด้วย Register a User-Written Condition Handler (CEEHDLR) bindable API ถูกใช้ในการจัดการกับข้อความ exception ที่ส่งด้วย Send Program Message (QMHSNDPM) API. การโต้ตอบนี้จะถูกอธิบายอยู่ในบทนี้. คำว่า **exception handler** ที่ใช้ในบทนี้จะหมายความถึงตัวจัดการ exception และ ตัวจัดการ ILE condition ของ OS/400.

Handle Cursors และ Resume Cursors

เพื่อที่จะโพรเซส exception ระบบใช้พอยน์เตอร์ 2 ตัวที่เรียกว่า handle cursor และ resume cursor. พอยน์เตอร์นี้จะติดตามการทำงานของการจัดการ exception. คุณจำเป็นต้องเข้าใจถึงการใช้ handle cursor และ resume cursor ภายใต้สถานการณ์ของการจัดการข้อผิดพลาดในระดับสูง. แนวคิดนี้จะใช้ในการอธิบายถึงลักษณะสำคัญของการจัดการข้อผิดพลาดเพิ่มเติมในหัวข้อต่อไป.

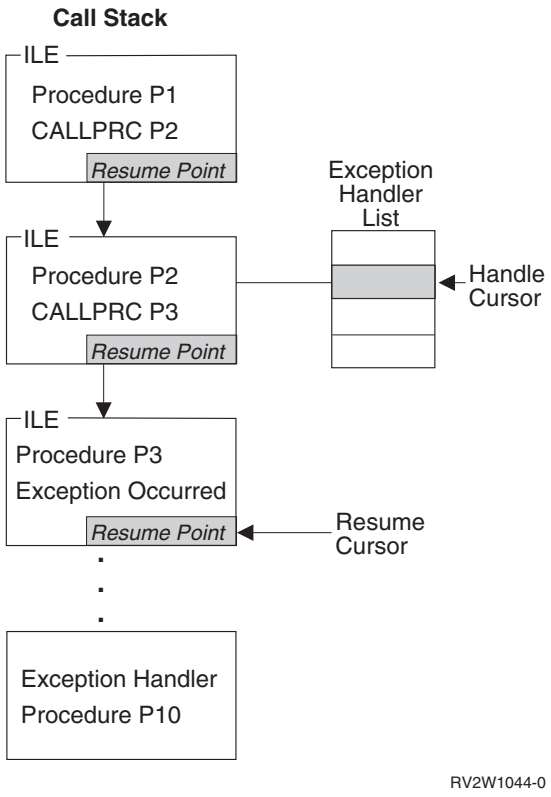
Handle Cursor คือ พอยน์เตอร์ที่แสดงตำแหน่งของ exception handler ตัวปัจจุบัน. เมื่อระบบค้นหาตัวจัดการ exception ที่มีอยู่ มันจะย้าย handle cursor ไปยังตัวจัดการตัวต่อไปตามรายการของตัวจัดการ exception ที่กำหนดโดย call stack entry. รายการเหล่านี้จะประกอบด้วย:

- ตัวจัดการแบบตรวจสอบโดยตรง
- ตัวจัดการแบบเงื่อนไขของ ILE
- ตัวจัดการแบบเฉพาะ HLL

Handle cursor จะเลื่อนไปตามรายการของตัวจัดการ exception ลงไปยังตัวจัดการที่มีลำดับความสำคัญต่ำกว่าจนกระทั่ง exception นั้นจะถูกจัดการ. ถ้า exception ไม่ถูกจัดการโดยตัวจัดการ exception ที่ถูกกำหนดไว้สำหรับ call stack entry handle cursor ก็จะย้ายไปที่ตัวจัดการตัวแรก (ซึ่งมีลำดับความสำคัญสูงสุด) สำหรับ call stack entry ก่อนหน้านี้.

Resume Cursor คือ พอยน์เตอร์ที่แสดงตำแหน่งของตัวจัดการ exception ปัจจุบัน ซึ่งจะสามารถโพรเซสต่อไปได้หลังจากจัดการกับ exception แล้ว. แล้วโดยปกติระบบจะตั้งค่า resume cursor ไปที่คำสั่งที่ต่อจากจุดที่เกิด exception. สำหรับ call stack entry ที่อยู่เหนือโพรซีเดอร์ที่เกิด exception ส่วน resume point จะอยู่ต่อจากการเรียกโพรซีเดอร์หรือโปรแกรม ที่หยุดการทำงานของโพรซีเดอร์หรือโปรแกรมในปัจจุบัน. คุณสามารถใช้ฟังก์ชั่น Move Resume Cursor (CEEMRCR) bindable API ในการย้าย resume cursor ไปยัง resume point ก่อนหน้านี้.

รูปที่ 45 ในหน้า 134 แสดงตัวอย่างของ handle cursor และ resume cursor.



รูปที่ 45. แสดงตัวอย่างของ Handle Cursor และ Resume Cursor

Handle cursor มีตำแหน่งปัจจุบันอยู่ที่ตัวจัดการ exception ลำดับที่ 2 ที่กำหนดไว้ในรายการของตัวจัดการ exception สำหรับโพรซีเดอร์ P2. ส่วน P10 คือ โพรซีเดอร์ตัวจัดการที่ระบบเรียกใช้ในปัจจุบัน. ปัจจุบัน ถ้าโพรซีเดอร์ P10 จัดการกับ exception และรีเทิร์น การควบคุมจะไปยังตำแหน่งปัจจุบันของ resume cursor ที่กำหนดไว้ในโพรซีเดอร์ P3. ในตัวอย่างนี้สมมุติว่าโพรซีเดอร์ P3 ปล่อยให้ exception ผ่านไปยังโพรซีเดอร์ P2.

โพรซีเดอร์ที่เป็นตัวจัดการ exception P10 สามารถเปลี่ยนแปลง resume cursor ได้ ด้วยการใช้ Move Resume Cursor (CEEMRCR) bindable API. โดยใน API มีอ็อพชันให้เลือก 2 แบบ. ตัวจัดการ exception สามารถดัดแปลง resume cursor ไปยังตำแหน่งใดตำแหน่งหนึ่งดังนี้:

- Call stack entry ที่มี handle cursor
- Call stack entry ตำแหน่งก่อน handle cursor

ใน รูปที่ 45คุณสามารถเปลี่ยนแปลง resume cursor ไปยัง โพรซีเดอร์ P2 หรือ P1 ก็ได้. หลังจาก resume cursor ถูกเปลี่ยนแปลงและ exception ถูกกำหนดว่าถูกจัดการแล้ว การรีเทิร์นแบบปกติจากตัวจัดการ exception ของคุณจะรีเทิร์นการควบคุมให้กับ resume point จุดใหม่.

การทำงานของตัวจัดการ Exception

เมื่อตัวจัดการ exception ถูกเรียกโดยระบบ คุณสามารถจัดการกับ exception ได้หลายวิธี. ตัวอย่าง เช่น ส่วนขยายของ ILE C สนับสนุน control action, จัดการ branch point และมอนิเตอร์ด้วย message ID. แอ็คชันที่เป็นไปได้ที่กล่าวถึงนี้เกี่ยวข้องกับชนิดของตัวจัดการต่อไปนี้:

- ตัวจัดการแบบตรวจสอบโดยตรง
- ตัวจัดการแบบเงื่อนไขของ ILE
- ตัวจัดการแบบเฉพาะ HLL

วิธีการดำเนินกระบวนการต่อไป

ถ้าคุณกำหนดว่ากระบวนการหนึ่งจะสามารถดำเนินต่อไปได้ คุณสามารถทำได้ที่ตำแหน่งปัจจุบันของ resume cursor. แต่ก่อนที่คุณจะสามารถดำเนินกระบวนการนั้นต่อไป จะต้องมีการเปลี่ยน exception message เพื่อแสดงว่ามันได้ถูกจัดการไปแล้ว. มีตัวจัดการหลายชนิดที่ต้องการให้คุณเปลี่ยน ข้อความ exception. แต่ในตัวจัดการชนิดอื่นๆ ระบบจะสามารถเปลี่ยนข้อความ exception ก่อนที่ตัวจัดการของคุณจะถูกเรียกใช้.

สำหรับตัวจัดการแบบตรวจสอบโดยตรง คุณอาจกำหนดวิธีการที่จะทำกับข้อความ exception. วิธีการนั้นอาจเป็นการเรียกไปยังตัวจัดการ เพื่อจัดการกับ exception ที่เกิดขึ้นก่อนการเรียกใช้ตัวจัดการ หรือเพื่อจัดการกับ exception และดำเนินโปรแกรมต่อไป. ถ้าวิธีการนั้นเป็นเพียงการเรียกไปยังตัวจัดการ คุณยังคงจัดการกับ exception ได้โดยการใช้ Change Exception Message (QMHCHGEM) API หรือใช้ bindable API CEE4HC (Handle Condition).) คุณสามารถเปลี่ยน resume point ภายในตัวจัดการแบบตรวจสอบโดยตรงได้โดยการใช้ Move Resume Cursor (CEEMRCR) bindable API. และหลังจากที่มีการเปลี่ยนแปลงแล้ว คุณยังดำเนินโพรเซสต่อไปโดยการรีเทิร์นจากตัวจัดการ exception ของคุณ.

สำหรับตัวจัดการ ILE condition คุณยังดำเนินโพรเซสต่อไปโดยการตั้งค่า return code value และย้อนกลับไปยังระบบ. ไอบีเอ็ม มีข้อมูลออนไลน์ที่อธิบายถึง actual return code values สำหรับ Register a User-Written Condition Handler (CEEHDLR) bindable API. โปรดดูส่วน *API* ที่พบในหมวด **Programming** ของ iSeries Information Center.

สำหรับตัวจัดการเฉพาะ HLL จะมีการเปลี่ยน exception message เพื่อแสดงสถานะว่ามันถูกจัดการก่อนที่ตัวจัดการของคุณจะถูกเรียกใช้. เพื่อศึกษาถึงวิธีการที่คุณจะสามารเปลี่ยนแปลง resume cursor จากตัวจัดการเฉพาะ HLL สามารถดูได้จากหนังสือ ILE HLL programmer's guide.

วิธีการปล่อยผ่าน Message

ถ้าคุณกำหนดให้ตัวจัดการของคุณไม่ให้จัดการกับข้อความ exception คุณสามารถปล่อยข้อความ exception นั้นผ่านไปยังตัวจัดการตัวต่อไปที่มีอยู่ได้. สำหรับการเกิดการปล่อยผ่านนั้นข้อความ exception จะไม่ถูกพิจารณาว่าเป็นแมสเสจที่ถูกจัดการแล้ว. ทำให้ตัวจัดการ exception อื่นใน call stack entry เดียวกันหรือลำดับก่อนหน้านี้ จะมีโอกาสในการจัดการกับข้อความ exception. เทคนิคในการปล่อยผ่านข้อความ exception นั้นมีได้หลายรูปแบบโดยขึ้นอยู่กับชนิดของตัวจัดการ exception.

สำหรับตัวจัดการแบบตรวจสอบโดยตรงไม่ต้องมีการเปลี่ยนข้อความ exception เพื่อแสดงว่ามันถูกจัดการแล้ว (handle). การรีเทิร์นแบบปกติจากตัวจัดการ exception ของคุณ ทำให้ระบบปล่อยผ่านแมสเสจไป. แมสเสจที่ถูกปล่อยผ่านจะไปยังตัวจัดการ exception ตัวต่อไปที่อยู่ในรายการของตัวจัดการ exception สำหรับ call stack entry ของคุณ. ถ้าตัวจัดการของคุณอยู่ที่ส่วนท้ายของรายการตัวจัดการ exception แมสเสจจะถูกปล่อยผ่านไปยังตัวจัดการ exception ลำดับแรกใน call stack entry ก่อนหน้านี้.

สำหรับตัวจัดการ ILE condition คุณแจ้งการปล่อยผ่านแมสเสจได้โดยการตั้งค่า return code value และย้อนกลับไปยังระบบ. ไอบีเอ็ม มีข้อมูลออนไลน์ที่อธิบายถึง actual return code values สำหรับ Register a User-Written Condition Handler (CEEHDLR) bindable API. โปรดดูส่วน *API* ที่พบในหมวด **Programming** ของ iSeries Information Center.

สำหรับตัวจัดการเฉพาะ HLL อาจจะเป็นไปไม่ได้ที่จะมีการปล่อยผ่านข้อความ exception. ขึ้นอยู่กับว่า HLL ของคุณกำหนดว่าข้อความนั้นถูกจัดการก่อนที่ตัวจัดการของคุณจะถูกเรียกใช้หรือไม่ ถ้าคุณไม่มีการกำหนดตัวจัดการเฉพาะ HLL ไว้ HLL ของคุณจะสามารถปล่อยผ่าน exception message ที่ไม่ถูกจัดการไปได้. กรุณาศึกษาจากคู่มืออ้างอิง HLL ของ ILE เพื่อกำหนดข้อความ exception ที่ตัวจัดการเฉพาะ HLL ของคุณจะสามารถจัดการได้.

วิธีการ Promote แมสเสจ

ภายใต้สถานการณ์ที่ถูกจำกัดไว้อย่างแน่นอน คุณสามารถเลือกที่จะเปลี่ยน exception message ไปเป็นข้อความชนิดอื่น. การกระทำนี้จะแสดงให้เห็นว่าข้อความ exception เดิมได้ถูกจัดการไปแล้ว และเริ่มกระบวนการ exception ใหม่ที่มีข้อความ exception ใหม่. การกระทำนี้จะได้รับอนุญาตจากตัวจัดการแบบตรวจสอบโดยตรง และตัวจัดการ ILE condition เท่านั้น.

สำหรับตัวจัดการตรวจสอบโดยตรงใช้ Promote Message (QMHPRMM) API ในการ promote แมสเสจ. ระบบจะสามารถ promote แมสเสจชนิด status และ escape เท่านั้น. ด้วย API นี้คุณจะมีการควบคุมเหนือการวางตำแหน่งของ handle cursor ที่ใช้ในการดำเนินกระบวนการ exception ต่อไป. โปรดดูส่วน *API* ที่พบในหมวด **Programming** ของ iSeries Information Center.

สำหรับตัวจัดการ ILE condition คุณแจ้งการ promote ข้อมูลได้โดยการตั้งค่า return code value และย้อนกลับไปยังระบบ. ไอบีเอ็ม มีข้อมูลออนไลน์ที่อธิบายถึง actual return code values สำหรับ Register a User-Written Condition Handler (CEEHDLR) bindable API. โปรดดูส่วน *API* ที่พบในหมวด **Programming** ของ iSeries Information Center.

การกระทำที่เป็นดีฟอลต์สำหรับ Unhandled Exception

ถ้า exception message ถูกปล่อยผ่านไปยัง ขอบเขตการควบคุม ระบบจะทำตามการกระทำที่เป็นดีฟอลต์. ถ้า exception คือ notify message ระบบจะส่งคำตอบที่เป็นดีฟอลต์, จัดการกับ exception และอนุญาตให้ผู้ส่ง notify message ดำเนินกระบวนการต่อไปได้. ถ้า exception คือ status message ระบบจะจัดการกับ exception และอนุญาตให้ผู้ส่งข้อความนั้นดำเนินกระบวนการต่อไป. ถ้า exception คือ escape message ระบบจะจัดการกับ escape message และส่ง function check กลับไปยังตำแหน่งของ resume cursor ในปัจจุบัน. และถ้า exception ที่ไม่ถูกจัดการคือ function check ราย

การทั้งหมดใน stack ไปจนถึง ขอบเขตการควบคุม จะถูกยกเลิกและ escape message เลขที่ CEE9901จะถูกส่งไปยังอันดับแรกของ stack entry ตัวต่อไป.

ตารางที่ 9แสดงถึงการตอบสนองที่เป็นค่าดีฟอลต์ของระบบเมื่อ exception ไม่ถูกจัดการที่ ขอบเขตการควบคุม.

ตารางที่ 9. การตอบสนองที่เป็นดีฟอลต์สำหรับ Unhandled Exceptions

ชนิดของแมสเสจ	ระดับความรุนแรงของ condition	เหตการณ์ที่เกิดขึ้นจาก Signal a Condition (CEESGL) Bindable API	Exception ที่เกิดขึ้นจากแหล่งใดๆ
Status	0 (Informative message)	รีเทิร์น condition ที่ไม่ถูกจัดการ	ทำงานต่อไปโดยไม่มีการบันทึกแมสเสจ.
Status	1 (Warning)	รีเทิร์น condition ที่ไม่ถูกจัดการ	ทำงานต่อไปโดยไม่มีการบันทึกแมสเสจ.
Notify	0 (Informative message)	Not applicable.	บันทึกแมสเสจแบบ notify และส่งคำตอบที่เป็นดีฟอลต์.
Notify	1 (Warning)	Not applicable.	บันทึกแมสเสจแบบ notify และส่งคำตอบที่เป็นดีฟอลต์.
Escape	2 (Error)	รีเทิร์น condition ที่ไม่ถูกจัดการ	บันทึกแมสเสจแบบ escape และส่งข้อความแบบฟังก์ชันเช็คไปยัง call stack entry ที่ตำแหน่ง resume point ปัจจุบัน.
Escape	3 (Severe error)	รีเทิร์น condition ที่ไม่ถูกจัดการ	บันทึกแมสเสจแบบ escape และส่งข้อความแบบฟังก์ชันเช็คไปยัง call stack entry ที่ตำแหน่ง resume point ปัจจุบัน.
Escape	4 (Critical ILE error)	บันทึกแมสเสจแบบ escape และส่งข้อความแบบฟังก์ชันเช็คไปยัง call stack entry ที่ตำแหน่ง resume point ปัจจุบัน.	บันทึกแมสเสจแบบ escape และส่งข้อความแบบฟังก์ชันเช็คไปยัง call stack entry ที่ตำแหน่ง resume point ปัจจุบัน.
Function check	4 (Critical ILE error)	Not applicable	สิ้นสุดแอ็พพลิเคชันและส่งแมสเสจ CEE9901 ไปยังตัวเรียก (caller) ของ ขอบเขตการควบคุม.

หมายเหตุ: เมื่อแอ็พพลิเคชันสิ้นสุดลงโดยฟังก์ชันเช็คที่ไม่สามารถจัดการได้ activation group จะถูกลบไปถ้า ขอบเขตการควบคุม เป็น call stack entry ที่เก่าที่สุดใน activation group.

Nested Exceptions

Nested Exception คือ exception ที่เกิดขึ้นในขณะที่มี exception อื่นกำลังถูกจัดการอยู่. เมื่อเกิดเหตุการณ์เช่นนี้ขึ้น กระบวนการของ exception ที่เกิดก่อนจะหยุดลงชั่วคราว. ระบบจะบันทึกข้อมูลที่เกี่ยวข้องทั้งหมดไว้ ได้แก่ ตำแหน่งของ handle cursor และ resume cursor. การจัดการ exception จะเริ่มต้นอีกครั้งกับ exception ที่เกิดขึ้นล่าสุด. ตำแหน่งใหม่ของ handle cursor และ resume cursor จะถูกตั้งโดยระบบ. เมื่อ exception ตัวใหม่ถูกจัดการเรียบร้อยแล้ว การจัดการกับ exception ตัวแรกจึงจะกระทำต่อไป.

เมื่อเกิด nested exceptions ขึ้น จะมี 2 สิ่งนี้ค้างอยู่ใน call stack:

- Call stack entry ที่เกี่ยวข้องกับ exception ตัวแรก.
- Call stack entry ที่เกี่ยวข้องกับตัวจัดการ exception ตัวแรก.

เพื่อที่จะลดความเป็นไปได้ในการเกิด exception handling loop ระบบจะหยุดการปล่อยผ่านของ nested exception ที่ call stack entry ของตัวจัดการ exception ตัวแรก. แล้วระบบจะ promote nested exception ให้เกิด function check message และปล่อยให้ function check message ผ่านไปยัง call stack entry เดียวกัน. ถ้าคุณไม่จัดการกับ nested exception หรือ function check message ระบบจะหยุดการทำงานของแอ็พพลิเคชันโดยการเรียกใช้ Abnormal End (CEE4ABN) bindable API. ในกรณีนี้ข้อความ CEE9901 จะถูกส่งไปยังตัวเรียกของ ขอบเขตการควบคุม.

ถ้าคุณย้าย resume cursor ในขณะที่กำลังโพรเซส nested exception อยู่ คุณจะสามารถเปลี่ยนแปลง exception ตัวแรกได้. เพื่อให้เกิดเหตุการณ์นี้ ต้องกระทำดังนี้:

1. ย้าย resume cursor ไปยัง call stack entry ที่อยู่ก่อน call stack entry ที่เกิด exception ตัวแรก.
2. ดำเนินโพรเซสต่อไปโดยการรีเทิร์นจากตัวจัดการของคุณ

Condition Handling

ILE condition คือข้อความ exception ของ OS/400 ที่แสดงลักษณะที่ไม่ขึ้นอยู่กับระบบ. ILE condition token ถูกใช้เป็นตัวแทนของ ILE condition. **Condition handling** หมายถึงฟังก์ชัน ILE ที่อนุญาตให้คุณจัดการกับข้อผิดพลาด โดยทำงานแยกจากการจัดการข้อผิดพลาดเฉพาะภาษา. ในระบบอื่นๆ ก็มีฟังก์ชันเหล่านี้. คุณสามารถใช้ condition handling ในการเพิ่มความสะดวกให้กับแอ็พพลิเคชันของคุณในระบบที่มี condition handling.

ILE condition handling ประกอบด้วยฟังก์ชันต่างๆ คือ:

- ความสามารถในการเรจิสเตอร์ตัวจัดการ ILE condition แบบไดนามิก
- ความสามารถในการสร้างสัญญาณของ ILE condition
- Condition token architecture
- Optional condition token feedback codes for bindable ILE APIs

รายละเอียดของฟังก์ชันเหล่านี้จะถูกอธิบายในหัวข้อต่อไป.

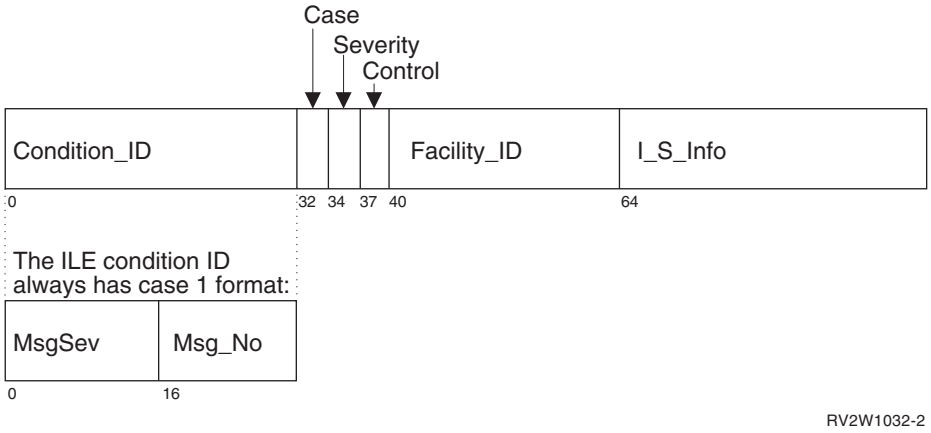
วิธีการในการแสดง Condition

ILE **Condition Token** เป็นข้อมูลแบบผสมขนาด 12 ไบต์ ที่โครงสร้างของฟิลด์ แสดงถึงลักษณะของ condition. ลักษณะเหล่านั้นได้แก่ ระดับของความรุนแรง (severity), ตัวเลขของข้อความที่เกี่ยวข้อง (associated message number), และข้อมูลที่แสดงลักษณะของ condition. condition token ใช้ข้อมูลเหล่านี้ในการสื่อสารกับระบบ, กับ message service, กับ bindable API และกับโพรซีเดอร์. ข้อมูลที่ถูกรีเทิร์นในพารามิเตอร์ fc ที่เป็นอ็อพชันของ ILE bindable API ทุกตัวเป็นตัวอย่างของการสื่อสารโดยการใช้ condition token.

ถ้า exception ถูกพบโดยระบบปฏิบัติการหรือโดยฮาร์ดแวร์ condition token ที่เกี่ยวข้องจะถูกระบบสร้างขึ้นโดยอัตโนมัติ. คุณยังสามารถสร้าง condition token โดยการใช้ Construct a Condition Token (CEENCOD) bindable API. และสามารถสร้าง signal ของ condition ให้กับระบบโดยการรีเทิร์น token ด้วยการใช้ Signal a Condition (CEESGL) bindable API.

รูปแบบของ Condition Token

รูปที่ 46แสดงแผนภาพของ condition token. ตำแหน่งของบิตเริ่มต้นถูกแสดงในแต่ละฟิลด์.



รูปที่ 46. ILE Condition Token Layout

Condition Token ทุกตัวประกอบด้วยองค์ประกอบที่แสดงใน รูปที่ 46

Condition_ID

เป็น identifier ขนาด 4 ไบต์เมื่อรวมกับ Facility_ID จะอธิบายถึง condition ที่ token สื่อสาร. ILE bindable API และแอ็พพลิเคชันส่วนใหญ่จะสร้าง condition แบบ case 1.

Case เป็นฟิลด์ขนาด 2 บิตที่กำหนดฟอร์แมตของ Condition_ID ซึ่งเป็นส่วนหนึ่งของ token. ILE ส่วนใหญ่มักจะเป็น case 1.

Severity

เป็นเลขฐานสองขนาด 3 บิตที่แสดงระดับความรุนแรงของ condition. โดยฟิลด์ Severity และ MsgSev จะบรรจุข้อมูลเหมือนกัน สำหรับระดับความรุนแรงของ condition. ดูจากตา

ราง ตารางที่ 9 ในหน้า 137 สำหรับระดับความรุนแรงของแมสเสจของ ILE. ดู ตารางที่ 11 ในหน้า 141 และ ตารางที่ 12 ในหน้า 141 สำหรับระดับความรุนแรงของ OS/400 ที่เกี่ยวข้อง.

Control

เป็นฟิลด์ขนาด 3 บิตที่บรรจุ flag ที่อธิบายหรือควบคุมลักษณะของการจัดการ condition. บิตที่ 3 จะแสดงถึง Facility_ID ที่ถูกกำหนดโดยไอบีเอ็ม.

Facility_ID

เป็น alphanumeric string ขนาด 3 ตัวอักษรที่แสดงถึงสิ่งที่สร้าง condition. condition Facility_ID จะชี้ให้เห็นว่าข้อความถูกสร้างโดยระบบหรือ HLL run time. ตารางที่ 10 แสดงรายการของ facility ID ที่ใช้ใน ILE.

I_S_Info

เป็นฟิลด์ขนาด 4 ไบต์ที่แสดงถึงข้อมูลเฉพาะที่สัมพันธ์กับ condition. condition ฟิลด์นี้จะบรรจุ reference key ของข้อความที่เกี่ยวข้องกับ condition token. ถ้า message reference key เท่ากับ 0 นั่นแสดงว่าไม่มีข้อความใดๆ.

MsgSev

เป็นเลขฐานสองขนาด 2 ไบต์ ที่แสดงถึงระดับความรุนแรงของ condition. MsgSev และ Severity จะมีข้อมูลที่เหมือนกัน. ดูจากตาราง ตารางที่ 9 ในหน้า 137 สำหรับระดับความรุนแรงของแมสเสจของ ILE. ดู ตารางที่ 11 ในหน้า 141 และ ตารางที่ 12 ในหน้า 141 สำหรับระดับความรุนแรงของ OS/400 ที่เกี่ยวข้อง.

Msg_No

เป็นเลขฐานสองขนาด 2 ไบต์ ที่แสดงถึงข้อความที่เกี่ยวข้องกับ condition. การรวมกันของ Facility_ID กับ Msg_No แสดงถึง condition ที่มีค่าเฉพาะตัว.

ตารางที่ 10แสดงถึง facility ID ที่ใช้ใน ILE และในส่วนที่เป็นคำเสริมหน้า (prefix) ของข้อความของ OS/400.

ตารางที่ 10. Facility ID ที่ใช้ใน Message และ ILE Condition Tokens

Facility ID	Facility
CEE	ILE common library
CPF	OS/400 XPF message
MCH	OS/400 machine exception message

การทดสอบ Condition Token

คุณสามารถทดสอบ condition token ที่รีเทิร์นจาก bindable API ได้ในลักษณะดังนี้:

Success เพื่อทดสอบถึงความสำเร็จให้พิจารณา condition token ว่า 4 ไบต์แรกมีค่าทั้งหมดเป็น 0 หรือไม่. ถ้าค่าทั้งหมดของ condition token เป็น 0 หมายความว่าการเรียกใช้ bindable API นั้นประสบความสำเร็จ.

Equivalent Tokens

เพื่อพิจารณาว่า condition token 2 ชุด equivalent กันหรือไม่ (คือมี*ชนิด*ของ condition

token เดียวกันแต่มี *instance* of the condition token ต่างกัน) โดยการเปรียบเทียบ 8 ไบต์แรกของ condition token. แต่ละชุดซึ่งจะมีค่าเหมือนกันสำหรับ instance ทุกตัวของ condition ที่ให้มา.

Equal Tokens

เพื่อพิจารณาว่า condition token 2 ชุด มีความเหมือนกันหรือไม่ (นั่นคือมี instance condition ที่เหมือนกันด้วย) โดยการเปรียบเทียบข้อมูลทั้ง 12 ไบต์ของ condition token ทั้ง 2 ชุด. ค่าของ 4 ไบต์สุดท้ายจะเปลี่ยนไปตาม instance แต่ละตัวของ condition.

ความสัมพันธ์ของ ILE Conditions กับแมสเสจของ OS/400

แมสเสจ หนึ่งๆ จะมีความสัมพันธ์กับ condition ทุกตัวที่เกิดขึ้นใน ILE. จะมี ID เฉพาะตัวที่ ILE ใช้ในการเขียนแมสเสจที่เกี่ยวข้องกับ condition ลงในไฟล์แมสเสจ.

ฟอร์แมตของแมสเสจขณะรัน (runtime message) คือ FFFxxxx

FFF คือ facility ID เป็น ID ขนาด 3 ตัวอักษรที่ถูกใช้โดยทุกข้อความที่สร้างขึ้นภายใต้ ILE และ ILE language. ดูใน ตารางที่ 10 ในหน้า 140สำหรับรายชื่อของ ID และ facility ที่สัมพันธ์กัน.

xxxx คือตัวเลขของข้อความแสดงความผิดพลาด. เป็นเลขฐาน 16 ที่แสดงถึงข้อความแสดงความผิดพลาดที่สัมพันธ์กับ condition.

ตารางที่ 11and ตารางที่ 12แสดงถึงระดับความรุนแรงของ ILE condition เทียบกับระดับความรุนแรงของข้อความของ OS/400.

*ตารางที่ 11. เปรียบเทียบระดับความรุนแรงของ OS/400 *ESCAPE Message กับ ILE Condition*

จาก OS/400 Message Severity	ไปยัง ILE Condition Severity	ไปยัง OS/400 Message Severity
0-29	2	20
30-39	3	30
40-99	4	40

*ตารางที่ 12. เปรียบเทียบระดับความรุนแรงของ OS/400 *STATUS และ *NOTIFY Message กับ ILECondition*

จาก OS/400 Message Severity	ไปยัง ILE Condition Severity	ไปยัง OS/400 Message Severity
0	0	0
1-99	1	10

OS/400 Messages และ Bindable API Feedback Code

เหมือนเป็นอินพุตของ bindable API คุณมีอ็อพชันในการเขียน feedback code และการใช้ feedback code ให้เป็น return code check ในโพรซีเดอร์. feedback code เป็นค่าของ condition token ที่มีเพื่อความยืดหยุ่นในการตรวจสอบรีเทิร์นจากการเรียกใช้ไปยังโพรซีเดอร์อื่น. อื่น คุณสามารถใช้

feedback code ให้เป็นอินพุตของ condition token ได้. ถ้า feedback code บนการเรียกใช้ไปยัง bindable API ถูกตัดออกและเกิด condition ขึ้นข้อความ exception จะถูกส่งไปยังตัวเรียกของ bindable API.

ถ้าคุณเขียนพารามิเตอร์ feedback code ในแอ็พพลิเคชันของคุณเพื่อรับข้อมูลที่เป็น feedback จาก bindable API เมื่อมี condition เกิดขึ้นจะเกิดเหตุการณ์ตามลำดับดังนี้:

1. Informational message ถูกส่งไปยังตัวเรียกของ bindable API เพื่อส่งข้อความที่เกี่ยวข้องกับ condition.
2. bindable API จะสร้าง condition token สำหรับ condition ที่เกิดขึ้น. bindable API จะใส่ข้อมูลลงในพื้นที่ของ instance specific information. instance specific information ของ condition token คือ message reference key ของ informational message. ซึ่งระบบใช้ในการตอบสนองกับ condition.
3. ถ้า condition ที่ถูกพบอยู่ในระดับอันตราย(มีระดับความรุนแรงเท่ากับ 4) ระบบจะส่ง exception message ไปยังตัวเรียกของ bindable API.
4. ถ้า condition ที่ถูกพบไม่อยู่ในระดับอันตราย(มีระดับความรุนแรงน้อยกว่า 4) condition token จะย้อนกลับไปยังรูทีน ที่เรียก bindable API.
5. เมื่อ condition token ย้อนกลับไปยังแอ็พพลิเคชันของคุณ คุณมีทางเลือกดังนี้:
   - ไม่สนใจและยังคงดำเนินโพรเซสต่อไป.
   - สร้างสัญญาณจาก condition โดยใช้ Signal a Condition (CEESGL) bindable API.
   - ใช้ Get, format, และ dispatch แมสเสจสำหรับการแสดงผลโดยใช้ Get, Format, and Dispatch a Message (CEEMSG) bindable API.
   - เก็บข้อความไว้ในพื้นที่สำหรับเก็บข้อมูลโดยใช้ Get a Message (CEEMGET) bindable API.
   - ใช้ Dispatch a Message (CEEMOUT) bindable API เพื่อส่งข้อความที่ผู้ใช้กำหนดไปยังเป้าหมายที่คุณกำหนดไว้อย่างรวดเร็ว.
   - เมื่อตัวเรียกของ API ได้รับการควบคุมอีกครั้ง informational message จะถูกลบออกไปและไม่ปรากฏอยู่ในบันทึกการใช้งาน.

ถ้าคุณตัดพารามิเตอร์ feedback code ออก เมื่อคุณเรียก bindable API, จะมี exception message จาก bindable API ส่งไปยังตัวเรียก bindable API นั้น.

บทที่ 10. ข้อพิจารณาในการดีบักโปรแกรม

ซอร์สดีบักเกอร์ (source debugger) ถูกใช้ในการดีบักโปรแกรม OPM, ILE, และเซอร์วิสโปรแกรม. และยังมีคำสั่ง CL ที่สามารถใช้ในการดีบัก โปรแกรม OPM ได้.

บทนี้จะแสดงถึงสิ่งที่ต้องพิจารณาหลายประการเกี่ยวกับซอร์สดีบักเกอร์. แต่สำหรับข้อมูลเกี่ยวกับวิธีการใช้งาน จะอยู่ในข้อมูลแบบออนไลน์และเกี่ยวกับคู่มือโปรแกรมเมอร์ราษาระดับสูง ILE (HLL) ในคู่มือโปรแกรมเมอร์. ส่วนข้อมูลเกี่ยวกับคำสั่งที่ใช้ในงานแบบใดแบบหนึ่งโดยเฉพาะ (เช่น การสร้างโมดูล) คุณสามารถพบได้ใน ILE HLL programmer's guide.

ดีบักโหมด

ในการใช้ซอร์สดีบักเกอร์ คุณจำเป็นจะต้องอยู่ในดีบักโหมด. ซึ่ง**ดีบักโหมด** เป็นสภาพแวดล้อมพิเศษที่โปรแกรมดีบักสามารถใช้ฟังก์ชันของระบบที่นอกเหนือจากฟังก์ชันปกติได้.

การเข้าสู่ดีบักโหมดได้ด้วยการรันคำสั่ง Start Debug (STRDBG).

สภาพแวดล้อมในการดีบัก

โปรแกรมสามารถถูกดีบักได้ในสภาพแวดล้อม 2 แบบนี้:

- สภาพแวดล้อมในการดีบักสำหรับ OPM. โปรแกรม OPM ทุกตัวจะถูกดีบักในสภาพแวดล้อมแบบนี้ ยกเว้นโปรแกรม OPM ที่ถูกรวมไว้ในสภาพแวดล้อมในการดีบัก ILE.
- สภาพแวดล้อมในการดีบักสำหรับ ILE. โปรแกรม ILE ทุกโปรแกรมจะถูกดีบักในสภาพแวดล้อมแบบนี้. นอกจากนี้โปรแกรม OPM ก็ถูกดีบักในสภาพแวดล้อมแบบนี้ได้ ถ้าเป็นไปตามกฎดังนี้:
  - เป็นโปรแกรมภาษา CL,COBOL, หรือ RPG.
  - ถูกคอมไพล์ด้วย OPM debug source data.
  - การตั้งค่าพารามิเตอร์ OPMSRC ของคำสั่ง STRDBG เป็น *YES.

สภาพแวดล้อมในการดีบัก ILE สนับสนุนการดีบักในระดับซอร์ส. ความสามารถในการดีบักจะมาจาก statement, source หรือ list views ของโค้ดโดยตรง.

เมื่อโปรแกรม OPM อยู่ในสภาพแวดล้อมในการดีบัก ILE ระบบจะทำการดีบักทั้งโปรแกรม ILE และ OPM โดยผ่านส่วนการติดต่อกับผู้ใช้เดียวกัน. สำหรับข้อมูลของวิธีการใช้ซอร์สดีบักเกอร์สำหรับโปรแกรม OPM ใน สภาพแวดล้อมในการดีบัก ILE ดูได้จากคำอธิบายออนไลน์หรือคู่มือโปรแกรมเมอร์สำหรับภาษาที่เทียบเท่ากับภาษาระดับสูงสำหรับ ILE (HLL) ที่ใช้ กับภาษาใน OPM (ได้แก่ ภาษา CL, โคบอลหรืออาร์พีจี).

การเพิ่มโปรแกรมเข้าไปในโหมดการดีบัก

ก่อนที่โปรแกรมจะถูกดีบักมันจะต้องถูกเพิ่มเข้าไปในดีบักโหมด. โปรแกรม OPM, โปรแกรม ILE, และ เซอร์วิสโปรแกรม ILE สามารถอยู่ในโหมดการดีบักได้ในเวลาเดียวกัน. จำนวนโปรแกรม OPM สูงสุดที่สามารถจะอยู่ในโหมดการดีบักได้ในเวลาเดียวกัน ภายใต้สภาพแวดล้อมในการดีบัก OPM คือ 20 โปรแกรม. ส่วนจำนวนของโปรแกรม ILE, เซอร์วิสโปรแกรม และโปรแกรม OPM ในสภาพแวดล้อมในการดีบัก ILE นั้นสามารถอยู่ในโหมดการดีบักในเวลาเดียวกันได้โดยไม่จำกัดจำนวน. แต่อย่างไรก็ตาม จำนวนสูงสุดของข้อมูลสำหรับการดีบักที่จะมีได้ในเวลาเดียวกันนั้นก็คือ 16 เมกกะไบต์ต่อ 1 โมดูล.

คุณต้องมีสิทธิ์ *CHANGE ในโปรแกรมหรือเซอร์วิสโปรแกรม เพื่อจะเพิ่มโปรแกรมเหล่านี้เข้าไปในดีบักโหมด. โปรแกรมหรือเซอร์วิสโปรแกรม สามารถถูกเพิ่มเข้าไปในดีบักโหมดได้เมื่อมันหยุดอยู่บน call stack.

ซอร์สดีบักเกอร์จะเข้าถึงโปรแกรม ILE และเซอร์วิสโปรแกรม 1 โมดูลต่อครั้ง. เมื่อคุณดีบักโปรแกรม ILE หรือเซอร์วิสโปรแกรม คุณอาจต้องการดีบักโมดูลหนึ่งที่อยู่ในโปรแกรมหรือเซอร์วิสโปรแกรมอื่น. ทำให้โปรแกรมหรือเซอร์วิสโปรแกรมนั้นต้องถูกเพิ่มเข้าไปในโหมดการดีบักก่อนที่โมดูลที่อยู่ในนั้นจะถูกดีบัก.

เมื่อโหมดการดีบักสิ้นสุดลงโปรแกรมทุกตัวจะถูกย้ายออกจากโหมดการดีบัก.

ผลกระทบของ Observability และ Optimization ต่อการดีบัก

การที่โมดูล observable และถูก optimize อย่างเต็มที่จะมีผลกระทบต่อความสามารถในการดีบักโมดูลนั้น.

Module **Observability** หมายถึง การที่ข้อมูลสามารถอยู่ในโมดูลที่ยอมให้มันถูกเปลี่ยนแปลงได้โดยไม่ต้องถูกคอมไพล์ใหม่. และ **Optimization** คือโพรเซสที่ระบบหาวิธีการทำงานที่จะลดการใช้รีซอร์สที่จำเป็นของระบบให้ได้มากที่สุดโดยที่ยังให้ผลลัพธ์เหมือนเดิม.

Observability

Module observability ประกอบด้วยข้อมูล 2 ชนิด คือ:

ข้อมูลสำหรับการดีบัก (Debug Data)
แทนค่าด้วย *DBGDTA. ข้อมูลนี้จำเป็นต่อการอนุญาตให้โมดูลถูก ดีบักได้.

ข้อมูลที่สร้างขึ้น (Creation Data)
แทนค่าด้วย *CRTDAT. ข้อมูลนี้จำเป็นต่อการแปลงรหัสไปเป็นคำสั่งเครื่อง. โมดูลจะต้องมีข้อมูลชนิดนี้ให้คุณใช้ในการเปลี่ยนระดับ optimization ของโมดูล.

เมื่อโมดูลถูกคอมไพล์ คุณทำได้เพียงการลบข้อมูลนี้เท่านั้น. การใช้คำสั่ง Change Module (CHGMOD) ทำให้คุณสามารถลบข้อมูลชนิดใดชนิดหนึ่งหรือทั้ง 2 ชนิดเลยก็ได้ การลบ observability ทั้งหมดจะทำให้โมดูลมีขนาดเล็กที่สุด (ด้วยการ compression). เมื่อข้อมูลนี้ถูกลบ

ออกไป คุณไม่สามารถเปลี่ยนโมดูลได้ด้วยวิธีอื่น นอกจากคุณจะคอมไพล์โมดูลอีกครั้ง และใส่ข้อมูลใหม่. เพื่อที่จะคอมไพล์มันอีกครั้งคุณจะต้องมีสิทธิใน ซอร์สโค้ดนั้นด้วย.

ระดับของ Optimization

โดยทั่วไปแล้วถ้าโมดูลมีข้อมูลที่สร้างขึ้น (creation data) คุณสามารถเปลี่ยนระดับที่ซอร์สโค้ดจะถูก optimize เพื่อจะรันในระบบได้. Processing shortcuts จะถูกแปลงไปเป็นภาษาเครื่อง เพื่อให้โพรซีเดอร์ในโมดูลรันได้อย่างมีประสิทธิภาพมากขึ้น. ยิ่งมีระดับของ optimization สูงขึ้นเท่าไรก็จะทำให้โพรซีเดอร์ในโมดูลรันได้มีประสิทธิภาพมากขึ้นเท่านั้น.

อย่างไรก็ตามถ้าโมดูลมี optimization สูง คุณจะไม่สามารถเปลี่ยนตัวแปรและไม่สามารถดูค่าที่แท้จริงของตัวแปรในระหว่างดีบักได้. เมื่อคุณกำลังดีบักให้ตั้งค่าระดับ optimization เท่ากับ 10 (*NONE). ซึ่งจะให้ประสิทธิภาพในระดับต่ำสุดสำหรับโพรซีเดอร์ในโมดูล แต่ยอมให้คุณแสดงค่าเปลี่ยนค่าตัวแปรได้. หลังจากการดีบักสิ้นสุดลงแล้วให้ตั้งค่าระดับ optimization เป็น 30 (*FULL) หรือ 40. ค่านี้จะให้ประสิทธิภาพในระดับสูงสุดสำหรับโพรซีเดอร์ในโมดูล.

การสร้างและลบข้อมูลสำหรับการดีบัก

ข้อมูลสำหรับการดีบักถูกเก็บไว้กับโมดูลแต่ละโมดูลและถูกสร้างขึ้นเมื่อโมดูลถูกสร้าง. เพื่อดีบักโพรซีเดอร์ในโมดูลที่ถูกสร้างขึ้นโดยไม่มีข้อมูลสำหรับการดีบัก คุณต้องสร้างโมดูลที่มีข้อมูลสำหรับการดีบักใหม่อีกครั้งและรวมโมดูลเข้ากับโปรแกรม ILE หรือเซอร์วิสโปรแกรมอีกครั้งหนึ่ง. โดยที่คุณไม่ต้องคอมไพล์โมดูลอื่นในโปรแกรมหรือเซอร์วิสโปรแกรมที่มีข้อมูลสำหรับการดีบักอยู่แล้วใหม่อีกครั้ง.

เพื่อลบข้อมูลสำหรับการดีบักออกจากโมดูล ให้สร้างโมดูลอีกครั้งโดยไม่มีข้อมูลสำหรับการดีบัก หรือใช้คำสั่ง Change Module (CHGMOD).

Module Views

ระดับของข้อมูลสำหรับการดีบักที่ใช้งานได้อาจมีค่าแตกต่างกันสำหรับโมดูลที่อยู่ในโปรแกรมหรือเซอร์วิสโปรแกรม ILE โปรแกรมหนึ่ง. โมดูลถูกคอมไพล์โดยแยกกันและอาจถูกสร้างด้วยคอมไพเลอร์และอ็อพชันที่ต่างกัน. ระดับของข้อมูลสำหรับการดีบักเหล่านี้จะกำหนดค่า **View**ที่ถูกสร้างจากคอมไพเลอร์และค่า view ที่ถูกแสดงโดยซอร์สดีบักเกอร์. ซึ่งมีค่าที่เป็นไปได้คือ:

*__NONE__
: หมายถึง ไม่มี debug views ถูกสร้างขึ้น.

*__STMT__
: หมายถึง ไม่มีซอร์สถูกแสดงโดยดีบักเกอร์ แต่สามารถเพิ่มจุดเปลี่ยนแปลงได้โดยใช้ชื่อโพรซีเดอร์ และ statement number ที่พบในรายการของคอมไพเลอร์. จำนวนของข้อมูลสำหรับการดีบักที่ถูกเก็บด้วยค่า view นี้เป็นจำนวนที่น้อยที่สุดของข้อมูลที่จำเป็นต่อการดีบัก.

*__SOURCE__
: หมายถึง ซอร์สดีบักเกอร์จะแสดงแหล่งที่มา ถ้าซอร์สไฟล์ที่ใช้คอมไพล์โมดูลยังคงอยู่ในระบบ.

*LIST หมายถึง list view ถูกสร้างขึ้นและเก็บไว้กับโมดูล. ซึ่งเป็นการอนุญาตให้ซอร์สดีบักเกอร์แสดงแหล่งที่มาของมัน ถึงแม้ว่าซอร์สไฟล์ที่ใช้ในการสร้างโมดูล จะไม่อยู่ในระบบแล้วก็ตาม. view นี้มีประโยชน์ในการเป็นก็อบปี้สำรอง ถ้าโปรแกรมจะถูกเปลี่ยนแปลง. อย่างไรก็ตามจำนวนของข้อมูลสำหรับการดีบักอาจมีขนาดใหญ่มาก โดยเฉพาะถ้าไฟล์อื่นถูกขยายเข้าไปในรายการด้วย. อ็อพชันของคอมไพเลอร์จะถูกใช้เมื่อโมดูลนั้นถูกสร้างโดยไม่คำนึงว่าข้อมูลถูกรวมเข้ามาด้วยหรือไม่. ไฟล์สามารถขยายไฟล์ DDS และไฟล์ข้อมูล (เช่น ILE C includes, ILE RPG /COPY files และ ILE COBOL COPY files).

*ALL หมายถึง debug view ทั้งหมดจะถูกสร้างขึ้น. เช่นเดียวกับ list view คือ จำนวนของ ดีบัก data อาจมีขนาดใหญ่มาก.

ILE RPG ยังมีข้อมูลสำหรับการดีบัก อ็อพชัน *COPY ที่สร้างทั้ง source view และ copy view. โดยที่ copy view คือ view ที่มี /COPY source member ทั้งหมดรวมอยู่ด้วย.

การดีบักข้ามงาน

คุณอาจต้องการใช้งานที่แยกออกไปอีกงานหนึ่งเพื่อดีบักโปรแกรมที่กำลังรันอยู่ในงาน ของคุณหรืองานเป็นแบ็ตช์. วิธีนี้มีประโยชน์มากเมื่อคุณต้องการสังเกตการทำงานของโปรแกรมโดยไม่มีการรบกวนของหน้าจอของดีบักเกอร์. ตัวอย่างเช่น หน้าจอหรือวินโดวส์ที่แสดงแอ็พพลิเคชันอาจซ้อนกันหรือถูกทำให้ซ้อนกัน โดยหน้าจอของดีบักเกอร์ในระหว่างทำงานไปที่ละขั้นหรือ ณ จุดพัก (breakpoint). คุณสามารถเลี่ยงปัญหานี้ได้โดยการเริ่มต้นงานเซอร์วิสและเริ่มต้นโปรแกรมดีบักในงานคนละงานกับอีกงานหนึ่งที่กำลังถูก ดีบัก. สำหรับข้อมูลในเรื่องนี้, สามารถดู ในภาคผนวกเกี่ยวกับการทดสอบในหนังสือคู่มือ CL Programming  .

การสนับสนุนโปรแกรมดีบักเกอร์ของ OPM และ ILE

การสนับสนุนโปรแกรมดีบักเกอร์ของ OPM และ ILE ทำให้เกิดการดีบักโปรแกรม OPM ในระดับ source โดยผ่าน ILE debugger API. สำหรับข้อมูลเกี่ยวกับ ILE Debugger API, สามารถดูได้ในส่วนของ *API* ของหมวด **Programming** ของ iSeries Information Center.โปรแกรมดีบักของ OPM และ ILE ทำให้เกิดการดีบักโปรแกรม ILE และ OPM โดยใช้ส่วนการติดต่อกับผู้ใช้เดียวกัน. และเพื่อที่จะใช้การสนับสนุนนี้ คุณต้องคอมไพล์โปรแกรม OPM ด้วยคอมไพเลอร์ของ AS/400 RPG, COBOL หรือ CL. คุณต้องตั้งค่าพารามิเตอร์ OPTION เป็น *SRCDBG หรือ *LSTDBG.

สนับสนุน Watch

คุณลักษณะ Watch ให้ความสามารถในการหยุดการ execute ของโปรแกรม. เมื่อพบว่าค่าในตำแหน่งเก็บข้อมูลที่กำหนดมีการเปลี่ยนแปลง. ตำแหน่งดังกล่าวสามารถกำหนดได้โดยใช้ชื่อของตัวแปรในโปรแกรมซึ่งจะถูก resolve ไปเป็นตำแหน่งที่เก็บข้อมูล และค่าของข้อมูลที่ตำแหน่งนี้จะถูกเฝ้าดูการเปลี่ยนแปลง. ถ้าค่าในตำแหน่งเก็บข้อมูลถูกเปลี่ยน การ execute ก็จะหยุดลง. ซอร์สโค้ดของโปรแกรมที่ถูกขัดจังหวะก็จะแสดง ณ ตำแหน่งที่ถูกอินเตอร์รัปต์และซอร์สโค้ดในบรรทัดที่ถูกไฮไลต์ จะถูกรันต่อจากคำสั่งที่ไปเปลี่ยนค่าในตำแหน่งที่เก็บข้อมูลนั้น.

Exception ที่ไม่ได้ถูกมอนิเตอร์

เมื่อเกิด unmonitored exception ขึ้น โปรแกรมที่กำลังรันอยู่จะให้ฟังก์ชันเช็ค และส่งข้อมูลไปยังบันทึกการใช้งาน. ถ้าคุณอยู่ในดีบักโหมดและโมดูลของโปรแกรมถูกสร้างขึ้นด้วยข้อมูลสำหรับการดีบัก ซอร์สดีบักเกอร์จะแสดงหน้าจอ Display Module Source. และถ้ามีความจำเป็น โปรแกรมก็จะถูกเพิ่มเข้าไปในดีบักโหมด. โมดูลที่เหมาะสมจะถูกแสดงหน้าจอและไฮไลต์บนบรรทัดที่ส่งผลกระทบ. ในที่สุดคุณก็สามารถดีบักโปรแกรมได้.

ข้อกำหนดในการสนับสนุนทางภาษาสำหรับการดีบัก

หากมีเหตุการณ์ใดเหตุการณ์หนึ่งเกิดขึ้น:

- Coded character set identifier (CCSID) ของงานที่ดีบัก เท่ากับ 290, 930, หรือ 5026 (Japan Katakana).
- Code page ของคำอธิบาย อุปกรณ์ (device) ที่ใช้ในการ ดีบัก คือ กับ 290, 930, หรือ 5026 (Japan Katakana).

คำสั่งดีบัก, ฟังก์ชัน, และตัวเลขฐาน 16 จะต้องเขียนเป็นอักษรตัวใหญ่. ดังตัวอย่างเช่น:

```
BREAK 16 WHEN var=X'A1B2'
```

ข้อกำหนดข้างต้นสำหรับ code page ภาษาญี่ปุ่นตัวอักษรคาตาคานะ จะใช้ไม่ได้เมื่อใช้ identifier name ในคำสั่งดีบัก (ตัวอย่างเช่น EVAL). อย่างไรก็ตามเมื่อดีบักโมดูลของ ILE RPG, ILE COBOL, หรือ ILE CL ชื่อของ Identifier ในคำสั่งดีบักจะถูกเปลี่ยนเป็นตัวอักษรตัวใหญ่โดยซอร์สดีบักเกอร์ และอาจมีการแสดงผลใหม่ที่แตกต่างออกไป.

บทที่ 11. การวางขอบเขตในการบริหารข้อมูล

บทนี้ประกอบด้วยข้อมูลที่เกี่ยวกับรีซอร์สการจัดการข้อมูล (data management resource) ที่อาจถูกใช้โดยโปรแกรม ILE หรือเซอร์วิสโปรแกรม. ก่อนจะอ่านบทนี้คุณควรมีความเข้าใจในแนวคิดของการจำกัดขอบเขตในการบริหารข้อมูลที่อธิบายใน “กฎในการจำกัดขอบเขตการบริหารข้อมูล (Data Management Scoping Rules)” ในหน้า 53.

รายละเอียดของรีซอร์สแต่ละชนิดจะอยู่ในหนังสือ ILE HLL programmer’s guide.

รีซอร์สการจัดการข้อมูลทั่วไป

หัวข้อนี้แสดงถึงรีซอร์สการจัดการข้อมูลทุกชนิดที่เป็นไปตามกฎของการวางขอบเขตในการบริหารข้อมูล รีซอร์สแต่ละชนิดที่กล่าวต่อไปนี้. เป็นคำอธิบายโดยสรุปของวิธีการในการวางขอบเขต. ขอบเขต ส่วนรายละเอียดเพิ่มเติมสามารถหาได้จากส่วนที่จะอ้างถึงในแต่ละหัวข้อ.

การเปิดไฟล์

การเปิดไฟล์ มีผลต่อการสร้างรีซอร์สชั่วคราวที่เรียกว่า open data path (ODP). คุณสามารถเริ่มต้นฟังก์ชัน open ได้โดยใช้ HLL open verbs, คำสั่ง Open Query File (OPNQRYF), หรือคำสั่ง Open Data Base File (OPNDBF). ODP ถูกจำกัดอยู่ใน activation group ของโปรแกรมที่เปิดไฟล์. สำหรับโปรแกรม OPM หรือ ILE ที่รันใน default activation group ODP จะมีขอบเขตที่ call-level number. คุณสามารถใช้ override เพื่อที่จะเปลี่ยนการจำกัดขอบเขตของ HLL open verbs ได้. โดยการกำหนดพารามิเตอร์ open scope (OPNSCOPE) ในคำสั่ง override, คำสั่ง OPNDBF, และคำสั่ง OPNQRYF.

การ Override

การ Override ถูกวางขอบเขตอยู่ในระดับ call, activation-group, หรือ job. เพื่อกำหนดการวางขอบเขตของ override ให้ใช้พารามิเตอร์override scope (OVRSCOPE) . ถ้าคุณไม่กำหนดการจำกัดขอบเขตอย่างชัดเจน ขอบเขตของ override จะขึ้นอยู่กับตำแหน่งที่ระบบให้ override. ถ้าระบบให้ override จาก default activation group มันจะถูกจำกัดขอบเขตอยู่ที่ระดับ call. แต่ถ้าระบบให้ override จาก activation group กลุ่มอื่นๆ มันก็จะถูกจำกัดขอบเขตอยู่ที่ระดับ activation group.

การกำหนด Commitment

การกำหนด Commitment สนับสนุนการกำหนดขอบเขตในระดับ activation group และระดับ job. ระดับในการกำหนดขอบเขตจะถูกกำหนดด้วยพารามิเตอร์ control scope (CTLSCOPE) ในคำสั่ง Start Commitment Control (STRCMTCTL). สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ commitment definition, โปรดดู หัวข้อ Backup and Recovery.

SQL cursor แบบโลคัล

คุณสามารถสร้างโปรแกรม SQL สำหรับผลิตภัณฑ์ที่เป็นคอมไพเลอร์ของ ILE. SQL cursor ที่ถูกใช้โดยโปรแกรม ILE อาจมีขอบเขตอยู่ในโมดูลหรือ activation group ก็ได้. คุณอาจกำหนดขอบเขตของ SQL cursor โดยการใช้พารามิเตอร์ end SQL (ENDSQL) ในคำสั่ง Create SQL Program.

การเชื่อมต่อ SQL แบบรีโมต

การเชื่อมต่อแบบรีโมตใช้ร่วมกับ SQL Cursor จะถูกกำหนดขอบเขตในระดับ activation group โดยถือเป็นส่วนหนึ่งของกระบวนการปกติของ SQL. ซึ่งยอมให้มีการติดต่อกันระหว่าง source job และหลาย target job หรือระหว่าง job ในหลายระบบ.

ตัวจัดการส่วนการติดต่อกับผู้ใช้ (User interface manager)

คำสั่ง Open Print Application (QUIOPNPA) และ Open Display Application API สนับสนุนพารามิเตอร์กำหนดขอบเขตของแอ็พพลิเคชัน. API เหล่านี้สามารถใช้วางขอบเขตตัวจัดการส่วนการติดต่อกับผู้ใช้ (UIM) ได้ทั้งในระดับ activation group และ job. สำหรับข้อมูลเพิ่มเติมเกี่ยวกับตัวจัดการส่วนการติดต่อกับผู้ใช้, โปรดดูส่วน *API* ในหมวด **Programming** ของ iSeries Information Center.

การเปิดการเชื่อมต่อข้อมูล (การจัดการการเปิดไฟล์)

คำสั่งใน Enable Link (QOLELINK) API จะทำให้เกิดการเชื่อมต่อข้อมูล. ถ้าคุณใช้ API นี้ภายใน ILE activation group การเชื่อมต่อข้อมูลจะถูกวางขอบเขตอยู่ใน activation group นั้น. แต่ถ้าคุณใช้ API นี้ภายใน default activation group การเชื่อมต่อข้อมูลจะถูกกำหนดขอบเขตอยู่ในระดับ call. สำหรับข้อมูลเพิ่เติมเกี่ยวกับ การเชื่อมต่อข้อมูลแบบเปิด, โปรดดูส่วน *API* ในหมวด **Programming** ของ iSeries Information Center.

การติดต่อแบบ Common Programming Interface (CPI)

Activation group ที่ก่อให้เกิดการติดต่อจะเป็นเจ้าของการติดต่อนั้น. ส่วน activation group ที่ทำให้เกิดการเชื่อมต่อโดยใช้ Enable Link (QOLELINK) API จะเป็นเจ้าของการเชื่อมต่อนั้นด้วย. ไอบีเอ็มมีข้อมูลออนไลน์เกี่ยวกับการติดต่อแบบ Common Programming Interface (CPI). โปรดดูส่วน *API* ในหมวด **Programming** ของ iSeries Information Center.

Hierarchical file system

Open System File (OHFOPNSF) API มีหน้าที่จัดการกับไฟล์ที่เป็น hierarchical file system (HFS). คุณสามารถใช้พารามิเตอร์ open information (OPENINFO) ใน API นี้ควบคุมการวางขอบเขตให้อยู่ในระดับ activation group หรือในระดับ job ได้. สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ Hierarchical File System, โปรดดูส่วน *API* ในหมวด **Programming** ของ iSeries Information Center.

การวางขอบเขตของ Commitment Control

ILE มีการเปลี่ยนแปลง 2 แบบสำหรับ commitment control คือ:

- การมี commitment definition หลายตัวที่เป็นอิสระต่อกันในแต่ละงาน. transaction แต่ละตัวสามารถถูก commit และ roll back อย่างเป็นอิสระต่อกัน. ก่อนที่จะมี ILE นั้น ในแต่ละงานจะมี commitment definition ได้เพียง 1 ตัวเท่านั้น.
- ถ้าการเปลี่ยนแปลงยังค้างอยู่เมื่อ activation groupนั้นสิ้นสุดการทำงานแบบปกติแล้ว ระบบจะ commit การเปลี่ยนแปลงนั้น. ในขณะที่ก่อนที่จะมี ILE ระบบจะไม่ commit การเปลี่ยนแปลงที่เกิดขึ้น.

Commitment control ยอมให้คุณกำหนดและโพรเซสการเปลี่ยนแปลงของรีซอร์ส เช่น ไฟล์หรือตารางฐานข้อมูลให้เป็น transaction เดียว. **Transaction** คือกลุ่มของการเปลี่ยนแปลงของอ็อบเจ็กต์ในระบบที่จะต้องปรากฏต่อผู้ใช้ว่าเป็นการเปลี่ยนแปลงเพียงเล็กน้อยเท่านั้น. commitment control ทำให้เราแน่ใจว่าจะมีสิ่งใดสิ่งหนึ่งต่อไปนี้เกิดขึ้นในระบบ:

- มีการเปลี่ยนแปลงเกิดขึ้นทุกกลุ่ม (**commit** operation)
- ไม่มีการเปลี่ยนแปลงใดๆ เกิดขึ้น (**rollback** operation)

รีซอร์สหลายๆ ตัวสามารถถูกเปลี่ยนแปลงได้ภายใต้ commitment control ที่ใช้ทั้งโปรแกรม OPM และโปรแกรม ILE.

คำสั่ง Start Commitment Control (STRCMTCTL) ทำให้โปรแกรมที่รันอยู่ในงานทำการเปลี่ยนแปลงภายใต้ commitment control ได้. เมื่อ commitment control เริ่มทำงานโดยการใช้คำสั่ง STRCMTCTL ระบบจะสร้าง **commitment definition**ขึ้น. ขึ้น ซึ่ง commitment definition แต่ละตัวจะถูกรู้จักโดยงานที่ใช้คำสั่ง STRCMTCTL เท่านั้น. commitment definition จะบรรจุข้อมูลที่เกี่ยวข้องกับรีซอร์สที่ถูกเปลี่ยนแปลงภายใต้ commitment control ภายในงานนั้น. ข้อมูลของ commitment control ใน commitment definition จะถูกรักษาไว้โดยระบบเมื่อรีซอร์ส commitment เปลี่ยนไป. การใช้คำสั่ง End Commitment Control (ENDCMTCTL) จะเป็นการจบการทำงานของ commitment definition. สำหรับข้อมูลเพิ่มเติมเกี่ยวกับ commitment control, โปรดดูหัวข้อ Backup and Recovery.

Commitment Definitions และ Activation Groups

Commitment definition หลายตัวสามารถเกิดขึ้นและถูกใช้โดยโปรแกรมที่กำลังรันอยู่ใน job. commitment definition แต่ละตัวสำหรับหนึ่ง job จะระบุถึง transaction หนึ่งที่มีรีซอร์สที่เกี่ยวข้องกัน. รีซอร์สเหล่านี้สามารถถูก commit หรือถูก roll back อย่างเป็นอิสระจาก commitment definition อื่นที่เกิดขึ้นในงานนั้น.

หมายเหตุ: มีเพียงโปรแกรม ILE เท่านั้นที่สามารถก่อให้เกิด commitment control ให้กับ activation group อื่นที่ไม่ใช่ default activation group. ดังนั้น job จะสามารถใช้ commitment definition ได้หลายตัวถ้า job นั้นรันอยู่ในโปรแกรม ILE เท่านั้น.

สำหรับโปรแกรม OPM จะรันใน default activation group. โดยปกติแล้วโปรแกรม OPM จะใช้ commitment definition *DFTACTGRP. แต่คุณสามารถใช้ commitment definition *JOB โดยการกำหนดค่า CMTSCOPE(*JOB) ลงในคำสั่ง STRCMTCTL.

เมื่อคุณใช้คำสั่ง Start Commitment Control (STRCMTCTL) คุณกำหนดขอบเขตของ commitment definition ลงในพารามิเตอร์ commitment scope (CMTSCOPE). **ขอบเขต** สำหรับ commitment definition แสดงถึงโปรแกรมที่รันภายใน job ที่ใช้ commitment definition นั้น. ขอบเขตที่เป็นดีฟอลต์ของ commitment definition คือ activation group ของโปรแกรมที่ใช้คำสั่ง STRCMTCTL. มีเพียงโปรแกรมที่รันภายใน activation group เท่านั้นที่ใช้ commitment definition นั้น. commitment definition ที่ถูกกำหนดขอบเขตไว้ที่ activation group จะถือว่าเป็น commitment definition ในแบบ **activation-group level**. สำหรับ OPM default activation group, commitment definition จะถูกเรียกว่า default activation-group (*DFTACTGRP) commitment definition.

commitment definition ที่มีระดับ activation–group หลายระดับสามารถเกิดขึ้นและถูกใช้โดยโปรแกรมที่รันอยู่ใน activation group หลายๆ กลุ่มในงานๆ หนึ่งได้.

Commitment definition ยังสามารถถูกจำกัดขอบเขตให้อยู่ใน job หนึ่งๆ ได้. โดยมีการการกำหนดขอบเขตในระดับงานหรือหรือ *JOB commitment definition. โปรแกรมใดๆ ที่กำลังรันอยู่ใน activation group ที่ไม่มี commitment definition ที่เกิดขึ้นในระดับ activation group จะใช้ commitment definition ในระดับงาน. เหตุการณ์นี้จะเกิดขึ้นถ้า commitment definition ในระดับงานถูกสร้างขึ้นไว้แล้วโดยโปรแกรมอื่นที่อยู่ในงานนั้น. แต่ในงานหนึ่งๆ จะมี commitment definition ในระดับงานได้เพียงตัวเดียวเท่านั้น.

สำหรับ activation group ที่ให้มานั้น จะมี commitment definition เพียงตัวเดียวเท่านั้นที่ถูกใช้โดยโปรแกรมที่รันอยู่ใน activation group นั้น. โปรแกรมที่รันอยู่ใน activation group สามารถใช้ commitment definition ทั้งในระดับงานหรือ activation group ก็ได้. อย่างไรก็ตามโปรแกรมเหล่านั้นก็ไม่สามารถใช้ commitment definition ทั้งสองได้ในเวลาเดียวกัน.

เมื่อโปรแกรมดำเนินกระบวนการ commitment control โปรแกรมจะไม่บ่งชี้ไปโดยตรงว่า จะใช้ commitment definition ตัวใดสำหรับการร้องขอ. ระบบจะพิจารณา commit definition โดยดูจาก activation group ที่มีโปรแกรมที่ร้องขอนั้นรันอยู่. ซึ่งเป็นวิธีการที่เป็นไปได้ เนื่องจาก ณ เวลาใดเวลาหนึ่งโปรแกรมที่รันอยู่ภายใน activation group จะสามารถใช้ commitment definition ได้เพียงตัวเดียว.

การจบการทำงานของ Commitment Control

Commitment control ทั้งที่อยู่ในระดับ job และระดับ activation group อาจจบการทำงานลงได้โดยการใช้คำสั่ง End Commitment Control (ENDCMTCTL). คำสั่ง ENDCMTCTL จะแจ้งไปยังระบบว่า commitment definition สำหรับ activation group ของโปรแกรมทำการร้องขอเพื่อจะจบการทำงาน. คำสั่ง ENDCMTCTL จะจบการทำงานของ commitment definition เพียง 1 ตัวเท่านั้น. แต่ commitment definition ตัวอื่นๆ ของงานนั้นจะยังคงไม่เปลี่ยนแปลง.

ถ้า commitment definition ที่ระดับ activation–groupจบการทำงาน โปรแกรมใดๆ ที่กำลังรันอยู่ภายใต้ activation group นั้นจะไม่สามารถทำการเปลี่ยนแปลงภายใต้ commitment control. อีกต่อไป. แต่ถ้า commitment definition ระดับงานนั้นเริ่ม หรือมีอยู่แล้ว การเปิดไฟล์ใหม่ก็จะใช้ commitment control แบบระดับงาน.

ถ้า commitment definition ระดับ job จบการทำงานลง โปรแกรมใดๆ ที่กำลังรันอยู่ในงานนั้นซึ่งใช้ commitment definition ระดับ job จะไม่สามารถทำการเปลี่ยนแปลงภายใต้ commitment control อีกต่อไป. แต่ถ้า commitment control เกิดขึ้นอีกด้วยคำสั่ง STRCMTCTL มันก็จะสามารถทำการเปลี่ยนแปลงได้.

Commitment Control ในขณะที่ activation group สิ้นสุดลง

เมื่อเหตุการณ์ต่อไปนี้เกิดขึ้นในเวลาเดียวกัน:

- Activation group สิ้นสุดลง
- Job ยังไม่จบการทำงาน

ระบบจะยุติการทำงานของ commitment definition ในระดับ activation group โดยอัตโนมัติ. ถ้าเกิดเหตุการณ์ทั้งสองนี้:

- การเปลี่ยนแปลงที่ไม่ถูก commit ยังมีอยู่สำหรับ commitment definition ในระดับ activation group
- Activation group จบการทำงานโดยปกติ

ระบบจะดำเนินกระบวนการ commit สำหรับ commitment definition ก่อนที่มันจะยุติการทำงานของ commitment definition ลง. มิฉะนั้นหากมีเหตุการณ์ใดเหตุการณ์หนึ่งต่อไปนี้:

- Activation group สิ้นสุดการทำงานแบบไม่ปกติ
- ระบบพบข้อผิดพลาดเมื่อปิดไฟล์ใดๆ ที่เปิดอยู่ภายใต้ commitment control ที่จำกัดวงอยู่ใน activation group.

จะเกิด rollback operation ขึ้นสำหรับ commitment definition ในระดับ activation group ก่อนที่จะจบการทำงานลง. เนื่องจาก activation group จบการทำงานแบบไม่ปกติ ระบบจะปรับปรุงค่า notify object ด้วยผลของ commitment operation สุดท้ายที่ประสบความสำเร็จ. การเกิด commit และ rollback จะขึ้นอยู่กับว่ามีการหยุดลงชั่วคราวของการเปลี่ยนแปลงหรือไม่. ไม่ ถ้าไม่มีแสดงว่าไม่มี rollback แต่อ็อบเจ็กต์ notify ยังคงมีการอัพเดตค่า. ถ้า activation group จบการทำงานแบบไม่ปกติด้วยการเปลี่ยนแปลงที่มีการหยุดลงชั่วคราวระบบก็จะ rollback การเปลี่ยนแปลงนั้น. แต่ถ้า activation group จบการทำงานแบบปกติด้วยการเปลี่ยนแปลงที่มีการหยุดลงชั่วคราว ระบบก็จะ commit การเปลี่ยนแปลงที่เกิดขึ้น.

Commit operation หรือ rollback operation จะไม่เคยเกิดขึ้น ในระหว่างที่ activation group จบกระบวนการสำหรับ *JOB หรือ *DFTACTGRP commitment definition. เนื่องจาก commitment definition ทั้งสองไม่เคยจบการทำงานลงเพราะการสิ้นสุดการทำงานของ activation group. เพราะ commitment definition เหล่านี้จะจบการทำงานลงด้วยคำสั่ง ENDCMTCTL หรืออาจจบลงโดยระบบเมื่อ job จบการทำงานลง.

ระบบจะปิดไฟล์ใดๆ ที่จำกัดขอบเขตอยู่ใน activation group โดยอัตโนมัติเมื่อ activation group จบการทำงาน. ซึ่งรวมไปถึงไฟล์ฐานข้อมูลที่มีขอบเขตอยู่ใน activation group ที่เปิดอยู่ภายใต้ commitment control ด้วย. กระบวนการปิดไฟล์สำหรับไฟล์หลายๆ ไฟล์เกิดขึ้นก่อนที่ commit operation จะถูกปฏิบัติตาม commitment definition ในระดับ activation group. ดังนั้น record ใดๆ ที่อยู่ในบัฟเฟอร์ I/O จะเป็นลำดับแรกที่ไปยังฐานข้อมูลก่อนที่ commit operation ใดๆ จะถูกกระทำ.

เหมือนเป็นส่วนหนึ่งของ commit operation หรือ rollback operation ระบบจะเรียก commit และ rollback exit program API สำหรับ commitment resource API. แต่ละตัว และ commitment resource API แต่ละตัวจะต้องสัมพันธ์กับ commitment definition ในระดับ activation group. หลังจาก commit และ rollback exit program API ถูกเรียกใช้ ระบบจะลบ commitment resource API ออกไปโดยอัตโนมัติ.

หากมีเหตุการณ์ใดเหตุการณ์หนึ่งเกิดขึ้น:

- Rollback operation ทำงานตาม commitment definition ที่จะต้องสิ้นสุดลงเพราะว่า activation group จะถูกทำให้จบการทำงาน.

- อ็อบเจ็กต์ Notify ถูกกำหนดสำหรับ commitment definition.

อ็อบเจ็กต์ Notify ก็จะถูกอ็อบเจ็กต์.

บทที่ 12. Bindable Application Programming Interface ของ ILE

ILE bindable application programming interface (bindable APIs) เป็นส่วนสำคัญของ ILE. ในบางกรณี ILE จะให้ฟังก์ชันเพิ่มเติมนอกเหนือจากฟังก์ชันที่มาจากภาษาชั้นสูงเฉพาะ.ตัวอย่างเช่น, ไม่ใช่ HLL ในทุกภาษาที่จะให้วิธีจัดการกับหน่วยความจำไดนามิก. ในกรณีนั้น, คุณสามารถเพิ่มฟังก์ชันของภาษาชั้นสูงโดยการใช้ bindable APIs โดยเฉพาะ. ถ้า HLL ของคุณมีฟังก์ชันที่เหมือนกับ bindable API โดยเฉพาะแล้ว, ให้ใช้ HLL ที่ระบุนั้น.

Bindable API ทำงานเป็นอิสระจาก HLL. ซึ่งเป็นประโยชน์ต่อแอ็พพลิเคชันที่มีหลายภาษาผสมกัน. ตัวอย่าง เช่น, ถ้าคุณใช้เพียงการบริหารสภาวะของ bindable APIs กับแอ็พพลิเคชัน หลายภาษา, คุณจะมีซีแมนทิกส์การจัดการสภาวะที่เป็นแบบเดียวกันสำหรับแอ็พพลิเคชันนั้น. วิธีนี้จะทำให้ condition management มีความสอดคล้องกันมากกว่าการใช้ตัวจัดการ condition เฉพาะ HLL หลายตัว.

Bindable APIs ให้ฟังก์ชันมากมายดังนี้ :

- ฟังก์ชันเกี่ยวกับ Activation group และ control flow
- ฟังก์ชันการจัดการ Condition
- ฟังก์ชันเกี่ยวกับวันและเวลา
- ฟังก์ชันการจัดการหน้าจอแบบไดนามิก
- ฟังก์ชันด้านคณิตศาสตร์
- ฟังก์ชันการจัดการแมสเสจ
- ฟังก์ชันการเรียกโปรแกรมหรือโพรซีเดอร์และการเข้าถึง operational descriptor
- ฟังก์ชันสำหรับซอร์สดีบักเกอร์
- ฟังก์ชันการจัดการหน่วยเก็บข้อมูล

สำหรับข้อมูลอ้างอิงเกี่ยวกับ ILE bindable API, โปรดดูส่วน *API* ในหมวด **Programming** ของ iSeries Information Center.

Bindable APIs ของ ILE ที่สามารถใช้ได้

Bindable APIs ส่วนใหญ่สามารถใช้ได้กับ HLL บางตัวที่ ILE ให้การสนับสนุน. ILE มี bindable APIs ดังต่อไปนี้:

Bindable API เกี่ยวกับ Activation Group และ Control Flow

Abnormal End (CEE4ABN)
Find a Control Boundary (CEE4FCB)
Register Activation Group Exit Procedure (CEE4RAGE)
Register Call Stack Entry Termination User Exit Procedure (CEERTX)
Signal the Termination-Imminent Condition (CEETREC)
Unregister Call Stack Entry Termination User Exit Procedure (CEEUTX)

Bindable API ที่เกี่ยวกับการจัดการ Condition

Construct a Condition Token (CEENCOD)
Decompose a Condition Token (CEEDCOD)
Handle a Condition (CEE4HC)
Move the Resume Cursor to a Return Point (CEEMRCR)
Register a User-Written Condition Handler (CEEHDLR)
Retrieve ILE Version and Platform ID (CEEGPID)
Return the Relative Invocation Number (CEE4RIN)
Signal a Condition (CEESGL)
Unregister a User Condition Handler (CEEHDLU)

Bindable API ที่เกี่ยวกับวันและเวลา

Calculate Day-of-Week from Lilian Date (CEEDYWK)
Convert Date to Lilian Format (CEEDAYS)
Convert Integers to Seconds (CEEISEC)
Convert Lilian Date to Character Format (CEEDATE)
Convert Seconds to Character Timestamp (CEEDATM)
Convert Seconds to Integers (CEESECI)
Convert Timestamp to Number of Seconds (CEESECS)
Get Current Greenwich Mean Time (CEEGMT)
Get Current Local Time (CEELOCT)
Get Offset from Universal Time Coordinated to Local Time (CEEUTCO)
Get Universal Time Coordinated (CEEUTC)
Query Century (CEEQCEN)
Return Default Date and Time Strings for Country or Region (CEEFMDT)
Return Default Date String for Country or Region (CEEFMDA)
Return Default Time String for Country or Region (CEEFMTM)
Set Century (CEESCEN)

API ทางคณิตศาสตร์

ตัวอักษร x ในชื่อของฟังก์ชันทางคณิตศาสตร์ แสดงถึงชนิดของข้อมูลชนิดใดชนิดหนึ่งดังนี้:

I คือ เลขจำนวนเต็มที่เป็นเลขฐานสอง ความยาว 32 บิต

S คือ เลขทศนิยม 1 ตำแหน่ง ความยาว 32 บิต

D คือเลขทศนิยม 2 ตำแหน่ง ความยาว 64 บิต

T คือ single floating-complex number (ทั้งส่วนที่มองเห็นได้และส่วนที่อยู่ในความคิด) ความยาว 32 บิต

E คือ double floating-complex number (ส่วนที่มองเห็นได้และส่วนที่อยู่ในความคิด) ความยาว 64 บิต

Absolute Function (CEESxABS)
Arccosine (CEESxACS)
Arcsine (CEESxASN)
Arctangent (CEESxATN)
Arctangent2 (CEESxAT2)
Conjugate of Complex (CEESxCJG)
Cosine (CEESxCOS)
Cotangent (CEESxCTN)
Error Function and Its Complement (CEESxERx)
Exponential Base e (CEESxEXP)
Exponentiation (CEESxXPx)
Factorial (CEE4SIFAC)
Floating Complex Divide (CEESxDVD)
Floating Complex Multiply (CEESxMLT)
Gamma Function (CEESxGMA)
Hyperbolic Arctangent (CEESxATH)
Hyperbolic Cosine (CEESxCSH)
Hyperbolic Sine (CEESxSNH)
Hyperbolic Tangent (CEESxTNH)
Imaginary Part of Complex (CEESxIMG)
Log Gamma Function (CEESxLGM)
Logarithm Base 10 (CEESxLG1)
Logarithm Base 2 (CEESxLG2)
Logarithm Base e (CEESxLOG)
Modular Arithmetic (CEESxMOD)
Nearest Integer (CEESxNIN)
Nearest Whole Number (CEESxNWN)
Positive Difference (CEESxDIM)
Sine (CEESxSIN)
Square Root (CEESxSQT)
Tangent (CEESxTAN)
Transfer of Sign (CEESxSGN)
Truncation (CEESxINT)

API ทางด้านคณิตศาสตร์เพิ่มเติม:

Basic Random Number Generation (CEERAN0)

Bindable API ด้านการจัดการแมสเสจ

Dispatch a Message (CEEMOUT)
Get a Message (CEEMGET)
Get, Format, and Dispatch a Message (CEEMSG)

Bindable API ด้านการเรียกโปรแกรมหรือโพรซีเดอร์

Get String Information (CEEGSI)
Retrieve Operational Descriptor Information (CEEDOD)
Test for Omitted Argument (CEETSTA)

Bindable API ด้านซอร์สดีบักเกอร์

Allow a Program to Issue Debug Statements (QteSubmitDebugCommand)
Enable a Session to Use the Source Debugger (QteStartSourceDebug)
Map Positions from One View to Another (QteMapViewPosition)
Register a View of a Module (QteRegisterDebugView)
Remove a View of a Module (QteRemoveDebugView)
Retrieve the Attributes of the Source Debug Session (QteRetrieveDebugAttribute)
Retrieve the List of Modules and Views for a Program (QteRetrieveModuleViews)
Retrieve the Position Where the Program Stopped (QteRetrieveStoppedPosition)
Retrieve Source Text from the Specified View (QteRetrieveViewText)
Set the Attributes of the Source Debug Session (QteSetDebugAttribute)
Take a Job Out of Debug Mode (QteEndSourceDebug)

Bindable API ด้านการจัดการหน่วยเก็บข้อมูล

Create Heap (CEECRHP)
Define Heap Allocation Strategy (CEE4DAS)
Discard Heap (CEEDSHP)
Free Storage (CEEFRST)
Get Heap Storage (CEEGTST)
Mark Heap (CEEMKHP)
Reallocate Storage (CEECZST)
Release Heap (CEERLHP)

API ด้านการจัดการหน้าจอแบบไดนามิก

API ด้านการจัดการหน้าจอแบบไดนามิก (DSM) คือชุดของอินเตอร์เฟส I/O ของหน้าจอ ที่สามารถสร้างและจัดการหน้าจอแสดงผลแบบไดนามิก ใช้สำหรับภาษา ILE ระดับสูง.

API ด้าน DSM สามารถแบ่งกลุ่มตามลักษณะการทำงานได้ดังนี้:

- **Low-level services**

  Low-level services APIs ให้การอินเตอร์เฟสโดยตรงไปยัง 5250 data stream commands API. ถูกใช้ในการ query และจัดการสถานะของหน้าจอที่แสดงผล; เพื่อสร้าง, สอบถาม, และจัดการอินพุตและคำสั่งบัฟเฟอร์ในการโต้ตอบกับการแสดงผลของหน้าจอ; และเพื่อกำหนดฟิลด์และเขียนข้อมูลลงในหน้าจอที่แสดงผล.

- **Window services**

Window services API ถูกใช้ในการสร้าง, ลบ, ย้าย, และเปลี่ยนขนาดของวินโดวส์; และเพื่อจัดการกับ วินโดวส์หลายๆ วินโดวส์ที่เปิดอยู่ในหนึ่งเซสชั่น.

- **Session services**

  Session service APIs มีอินเตอร์เฟส paging ทั่วไปที่สามารถใช้ในการสร้าง, สอบถาม, และจัดการกับเซสชั่น และดำเนินการกับอินพุตและเอาท์พุตของเซสชั่น.

ไอบีเอ็มมีข้อมูลออนไลน์เกี่ยวกับ DSM bindable API. โปรดดู ส่วน *API* ในหมวด **Programming** ของ iSeries Information Center.

บทที่ 13. เทคนิคขั้นสูงของการทำ optimization

ในบทนี้จะอธิบายถึงเทคนิคต่างๆ ที่คุณสามารถนำไปใช้ในการ optimize โปรแกรมและเซอร์วิสโปรแกรม ILE ของคุณ:

- "การจัดทำโปรไฟล์ (Program Profiling)"
- "การวิเคราะห์ระหว่างโพรซีเดอร์ (Interprocedural Analysis – IPA)" ในหน้า 169
- "Licensed Internal Code Options" ในหน้า 176

การจัดทำโปรไฟล์ (Program Profiling)

การจัดทำโปรไฟล์เป็นเทคนิคขั้นสูงของการทำ optimization เพื่อจัดลำดับโพรซีเดอร์, หรือโค้ดภายในโพรซีเดอร์, ที่อยู่ในโปรแกรม ILE และเซอร์วิสโปรแกรมเสียใหม่ โดยมีพื้นฐานการจัดลำดับอยู่บนการรวบรวมข้อมูลแบบสถิติ ในขณะที่กำลังรันโปรแกรม. การจัดลำดับใหม่นี้สามารถปรับปรุงยูทิไลเซชันของ instruction cache และลด paging ที่ต้องการโดยโปรแกรม ,ซึ่งจะเป็นการเพิ่มประสิทธิภาพในการทำงาน. ลักษณะการทำงานแบบ semantic (semantic behavior) ของโปรแกรมจะไม่รับผลกระทบจากการจัดทำโปรไฟล์.

การพัฒนาประสิทธิภาพในการทำงานที่เกิดจากการจัดทำโปรไฟล์ขึ้นอยู่กับชนิดของแอ็พพลิเคชัน. โดยทั่วไป, คุณสามารถคาดหวังได้ว่าการพัฒนาโปรแกรมให้ดีขึ้นคือใช้เวลาส่วนใหญ่ในโค้ดของแอ็พพลิเคชันเองมากกว่าการใช้เวลาในการรันหรือการโพรเซสอินพุต/เอาท์พุต. ประสิทธิภาพการทำงานของโค้ดโปรแกรมที่ได้หลังจากการทำโปรไฟล์ขึ้นกับความแม่นยำของ optimizing translator ในการระบุส่วนที่สำคัญที่สุดของโปรแกรมระหว่างการใช้งาน. จำเป็นอย่างยิ่งที่ต้องรวบรวมข้อมูลของโปรไฟล์ในระหว่างที่ผู้ใช้กำลังทำงาน, และจำเป็นที่ต้องใช้ข้อมูลอินพุตที่ใกล้เคียงกับสภาพแวดล้อมจริงการทำงานของการรันโปรแกรม.

การจัดทำโปรไฟล์สามารถใช้ได้เฉพาะกับโปรแกรม ILE และเซอร์วิสโปรแกรมที่ตรงตามเงื่อนไขเหล่านี้:

- โปรแกรมถูกสร้างขึ้นโดยเฉพาะสำหรับ V4R2M0 หรือรีลีสหลังจากนั้น.
- หากโปรแกรมถูกสร้างมาสำหรับรีลีสก่อนหน้า V5R2M0, รีลีสเป้าหมายของโปรแกรมต้องตรงกับรีลีสของระบบ.
- โปรแกรมที่ถูกคอมไพล์โดยใช้การ optimization ระดับ *FULL (30) หรือสูงกว่านั้น. ส่วนระบบที่เป็น V5R2M0 หรือหลังจากนั้น, สามารถใช้โมดูลที่ถูกรวมเข้ากันด้วยการoptimizationที่ต่ำกว่าระดับ 30 ได้, แต่ต้องเป็นโมดูลที่ไม่เกี่ยวข้องกับการทำโปรไฟล์แอ็พพลิเคชัน.

หมายเหตุ: เนื่องจากข้อกำหนดของ optimization, คุณจะต้องดีบักโปรแกรมของคุณให้เรียบร้อยก่อนการจัดทำโปรไฟล์.

ชนิดของการจัดทำโปรไฟล์

คุณสามารถจัดทำโปรไฟล์ให้โปรแกรมของคุณได้ 2 วิธีดังนี้:

- แบบ block order
- แบบ procedure order และ block order

การจัดทำโปรไฟล์แบบ block order จะจดจำจำนวนครั้งที่มีการเรียก condition. เมื่อใช้ข้อมูลโปรไฟล์ของ block order เข้ากับโปรแกรม, จะเกิดการ optimization แบบขึ้นกับโปรไฟล์ภายในโพรซีเตอร์จำนวนมากมายโดย optimizining translator. หนึ่งในการ optimization ที่เกิดขึ้นก็คือการจัดเรียงลำดับให้พาธของโค้ดที่มีการเรียกใช้งานบ่อยที่สุดในโพรซีเตอร์อยู่ติดกับอ็อบเจ็กต์ของโปรแกรม. การจัดเรียงลำดับดังกล่าวเพิ่มประสิทธิภาพได้ เพราะทำให้สามารถใช้งานส่วนประกอบของหน่วยประมวลผลเช่นแคชคำสั่งและส่วนคำสั่ง prefetch ได้ดีขึ้น.

การจัดทำโปรไฟล์แบบ procedure order จะบันทึกจำนวนครั้งของการเรียกโพรซีเดอร์อื่นของโพรซีเตอร์แต่ละตัวภายในโปรแกรม. โพรซิเตอร์ที่อยู่ในโปรแกรมจะถูกจัดเรียงใหม่โดยรวมเอาโพรซิเตอร์ที่ถูกเรียกใช้มากที่สุดมาอยู่รวมกัน. การจัดเรียงนี้เพิ่มประสิทธิภาพการทำงานได้เพราะทำให้จำนวนการเพจหน่วยความจำลดลง.

ถึงแม้ว่าคุณจะสามารถเลือกที่จะใช้เพียงแบบ block order แบบเดียวกับโปรแกรมของคุณ, มันจะดีกว่าถ้าคุณทำโปรไฟล์ทั้งสองแบบเพื่อให้ได้ประสิทธิภาพสูงสุด.

วิธีการในการจัดทำโปรไฟล์โปรแกรม

การจัดทำโปรไฟล์โปรแกรม ประกอบด้วย 5 ขั้นตอน คือ:

1. เริ่มให้โปรแกรมรวบรวมข้อมูลการทำโปรไฟล์.
2. เริ่มทำการรวบรวมข้อมูลการทำโปรไฟล์ในระบบโดยการใช้คำสั่ง Start Program Profiling (STRPGMPRF).
3. เก็บรวบรวมข้อมูลโปรไฟล์โดยการรันโปรแกรมผ่าน code path ของโปรแกรมที่มีการใช้งานสูง. เนื่องจากการจัดทำโปรไฟล์ใช้ข้อมูลถิติที่รวบรวมในขณะที่รันโปรแกรม เพื่อให้เกิด optimization, จึงจำเป็นที่ข้อมูลจะต้องถูกเก็บให้ครอบคลุมสิ่งที่เป็นตัวแทนของการใช้งานของแอ็พพลิเคชันของคุณ.
4. จบการรวบรวมข้อมูลการทำโปรไฟล์ของระบบด้วยคำสั่ง End Program Profiling (ENDPGMPRF).
5. ใช้ข้อมูลการจัดทำโปรไฟล์ที่เก็บมากับโปรแกรมโดยการร้องขอให้โค้ดของโปรแกรมนั้นถูกจัดลำดับใหม่ เพื่อประสิทธิภาพสูงสุดโดยมีพื้นฐานมาจากข้อมูลการทำโปรไฟล์ที่รวบรวมมา.

การกำหนดให้โปรแกรมรวบรวมข้อมูลการทำโปรไฟล์

โปรแกรมจะถูกกำหนดให้รวบรวมข้อมูลการทำโปรไฟล์ถ้ามีโมดูลอย่างน้อย 1 โมดูลถูกเชื่อมเข้ากับโปรแกรมที่กำหนดให้รวบรวมข้อมูล. การกำหนดโปรแกรมให้รวบรวมข้อมูลการทำโปรไฟล์นี้ทำได้โดยการเปลี่ยนอ็อบเจ็กต์ *MODULE ตั้งแต่ 1 อ็อบเจ็กต์ขึ้นไปให้เก็บรวบรวมข้อมูลการทำ

โปรไฟล์ แล้วสร้างหรืออัพเดตโปรแกรมด้วยโมดูลเหล่านี้, หรือโดยการเปลี่ยนโปรแกรมหลังจากที่มันถูกสร้างให้รวบรวมข้อมูลการทำโปรไฟล์ก็ได้. เทคนิคทั้ง 2 แบบจะทำให้เกิดโปรแกรมที่มีโมดูลที่ให้เก็บข้อมูลการทำโปรไฟล์.

อาจมีอ็อพชันในคำสั่งของคอมไพเลอร์ในการสร้างโมดูลที่อนุญาตให้เก็บข้อมูลการทำโปรไฟล์ได้ ซึ่งจะขึ้นอยู่กับภาษา ILE ที่คุณใช้. คำสั่ง change module (CHGMOD) ก็สามารถใช้ได้โดยการระบุค่า *COL ลงในพารามิเตอร์ profiling data (PRFDTA) เพื่อเปลี่ยนให้โมดูลใดๆ ของ ILE เก็บข้อมูลการทำโปรไฟล์, ตราบใดที่ภาษา ILE นั้นยังสนับสนุนระดับการ optimization อย่างน้อยเท่ากับ *FULL (30).

เพื่อให้โปรแกรมเก็บรวบรวมข้อมูลการทำโปรไฟล์หลังจากการใช้คำสั่ง Change Program (CHGPGM) หรือ Change Service Program (CHGSRVPGM), ให้ปฏิบัติดังนี้

- ระบุค่าเป็น *COLในพารามิเตอร์ profiling data (PRFDTA). ค่าดังกล่าวจะมีผลกับทุกโมดูลที่มีการเชื่อมโยงในโปรแกรมถ้าโปรแกรมนั้น:
  - ถูกสร้างเพื่อใช้กับรีลีส V4R2M0 หรือหลังจากนั้น. ถ้าคุณใช้ระบบที่เก่ากว่าV5R2M0, คุณต้องใช้งานโปรแกรมบนระบบที่มีรีลีสระดับเดียวกับตอนสร้างโปรแกรม เพื่อให้เกิดการรวบรวมข้อมูลการทำโปรไฟล์ .ข้อกำหนดนี้มีผลถึงกับโมดูลที่ถูกเชื่อมโยงด้วย .
  - มีระดับของ optimization อย่างน้อยเท่ากับ 30.
    ส่วนระบบที่มีรีลีสV5R2M0หรือหลังจากนั้น, สามารถใช้โมดูลที่ถูกรวมเข้ากันด้วยการoptimizationที่ต่ำกว่าระดับ 30 ได้, แต่ต้องเป็นโมดูลที่ไม่เกี่ยวข้องกับการทำโปรไฟล์แอ็พพลิเคชัน.

หมายเหตุ: โปรแกรมที่กำหนดให้รวบรวมข้อมูลการทำโปรไฟล์ ของแอพพลิเคชันที่อยู่บนระบบที่มีรีลีสก่อนหน้า V5R2M0 สามารถนำข้อมูลที่ได้มาใช้บนระบบที่มีรีลีส V5R2M0 หรือใหม่กว่าได้, แต่จะไม่ได้ผลดีสุด. ถ้าคุณต้องการใช้ข้อมูลของการทำโปรไฟล์หรือผลลัพธ์ของโปรแกรมบนระบบที่เป็น V5R2M0 หรือมีรีลีสหลังจากนั้น, คุณต้องให้ระบบรวบรวมข้อมูลการทำโปรไฟล์ของโปรแกรมเริ่มทำงานหรือให้กลับมาทำงานอีกครั้ง .

การทำให้โมดูลหรือโปรแกรมเริ่มทำการรวบรวมข้อมูลการทำโปรไฟล์ได้จะต้องให้อ็อบเจ็กต์นั้นถูกสร้างขึ้นใหม่. ดังนั้น, เวลาที่ใช้เพื่อทำให้โมดูลหรือโปรแกรมเริ่มเก็บข้อมูลการทำโปรไฟล์ จะเทียบได้กับเวลาที่ใช้ในการสร้างอ็อบเจ็กต์ขึ้นใหม่อีกครั้ง (พารามิเตอร์ FRCCRT). นอกจากนี้, ขนาดของอ็อบเจ็กต์จะใหญ่ขึ้นเพราะคำสั่งเครื่องพิเศษที่ถูกสร้างขึ้นโดย optimizing translator.

เมื่อคุณเริ่มให้โปรแกรมหรือโมดูลเก็บข้อมูลการทำโปรไฟล์ ข้อมูลชนิดobservability ที่สร้างขึ้นไม่สามารถถูกลบออกไปจนกว่าจะมีเหตุการณ์ใดเหตุการณ์หนึ่งเกิดขึ้น:

- ข้อมูลการทำโปรไฟล์ที่เก็บได้ถูกใช้กับโปรแกรม.
- มีการแก้ไขโปรแกรมหรือโมดูลเพื่อทำให้ไม่สามารถเก็บข้อมูลการทำโปรไฟล์ได้.

ใช้คำสั่ง Display Module (DSPMOD), Display Program (DSPPGM), หรือ Display Service Program (DSPSRVPGM) และระบุค่า DETAIL(*BASIC), เพื่อแสดงผลถ้าโมดูลหรือโปรแกรมถูกกำหนดให้รวบรวมข้อมูลการทำโปรไฟล์. สำหรับโปรแกรมหรือเซอร์วิสโปรแกรม ที่ใช้อ็อพชัน 5 (แสดงคำอธิบาย) จากค่า DETAIL(*BASIC) จะแสดงว่าโมดูลที่ถูกรวมตัวใดถูก enable ให้รวบ

รวมข้อมูลการทำโปรไฟล์. ดูในหัวข้อ “วิธีการในการแสดงว่าโปรแกรมหรือโมดูลถูกทำโปรไฟล์หรือถูกตั้งค่าให้รวบรวมข้อมูลโปรไฟล์” ในหน้า 168 สำหรับรายละเอียดเพิ่มเติม.

หมายเหตุ: ถ้าโปรแกรมมีการรวบรวมข้อมูลการทำโปรไฟล์ (ข้อมูลแบบสถิติที่รวบรวมได้ในขณะที่โปรแกรมกำลังรันอยู่) ไว้เรียบร้อยแล้ว ,ข้อมูลนี้จะถูกลบไปเมื่อโปรแกรมมีการ enable ให้รวบรวมข้อมูลการทำโปรไฟล์ใหม่อีกครั้ง. ดูในหัวข้อ “การจัดการกับโปรแกรมที่ตั้งค่าให้รวบรวมข้อมูลโปรไฟล์” ในหน้า 166 สำหรับรายละเอียดเพิ่มเติม.

การรวบรวมข้อมูลการทำโปรไฟล์

เพื่อให้ค่าของจำนวนครั้งการทำโปรไฟล์ถูกต้อง ระบบที่มีโปรแกรมถูกตั้งค่าให้เก็บข้อมูลการทำโปรไฟล์จะต้องมีการจัดทำโปรไฟล์อยู่. วิธีนี้เป็นการทำให้แอ็พพลิเคชันที่มีขนาดใหญ่และใช้เวลานานในการรันสามารถเริ่มต้นและไปยังสถานะคงที่ได้ก่อนที่จะรวบรวมข้อมูลการทำโปรไฟล์. และวิธีนี้ให้คุณได้ควบคุมทุกอย่างเมื่อการรวบรวมข้อมูลเกิดขึ้น.

ใช้คำสั่ง Start Program Profiling (STRPGMPRF) เมื่อเริ่มการทำโปรไฟล์บนเครื่อง. และใช้คำสั่ง End Program Profiling (ENDPGMPRF) เมื่อจบการทำโปรไฟล์. ไอบีเอ็มได้สร้างคำสั่งทั้งสองให้มีสิทธิพับลิกแบบ *EXCLUDE. การทำโปรไฟล์จะจบลงอย่างแน่นอนเมื่อเครื่องถูก IPL.

เมื่อเริ่มการทำโปรไฟล์, โปรแกรมหรือเซอร์วิสโปรแกรม ใดๆ ที่รันอยู่และถูกกำหนดให้รวบรวมข้อมูลการทำโปรไฟล์ด้วยจะอัพเดตจำนวนนับการทำโปรไฟล์ของตัวเอง. เหตุการณ์นี้จะเกิดขึ้นโดยไม่คำนึงว่าโปรแกรมได้ถูก activate ก่อนที่จะมีการใช้คำสั่ง STRPGMPRF หรือไม่.

ถ้าโปรแกรมที่คุณรวบรวมข้อมูลโปรไฟล์ไว้ถูกเรียกโดยงานหลายๆ งานบนเครื่อง, งานเหล่านั้นทั้งหมดจะ อัพเดตจำนวนนับการทำโปรไฟล์ด้วย. แต่ถ้าสิ่งนี้ไม่เป็นที่ต้องการ, จะมีการก็อปปีโปรแกรมในไลบรารีที่แยกออกไป และตัวก็อปปีนั้นจะถูกใช้งานแทน.

หมายเหตุ:

1. เมื่อการทำโปรไฟล์เริ่มทำงานบนเครื่อง, จำนวนนับการทำโปรไฟล์จะเพิ่มขึ้นในขณะที่โปรแกรมที่ถูก enable ให้รวบรวมข้อมูลโปรไฟล์กำลังทำงาน. ดังนั้นจึงเป็นไปได้ที่จำนวนนับการทำโปรไฟล์ ″ตัวเก่า″ จะถูกเพิ่มเข้าไปด้วย ถ้าก่อนหน้านั้นโปรแกรมนี้รันโดยไม่มีการลบค่าเหล่านั้นออกไปก่อน. คุณสามารถกำหนดให้ข้อมูลการทำโปรไฟล์ถูกลบออกไปได้หลายวิธี. ดูในหัวข้อ “การจัดการกับโปรแกรมที่ตั้งค่าให้รวบรวมข้อมูลโปรไฟล์” ในหน้า 166 สำหรับรายละเอียดเพิ่มเติม.
2. จะไม่มีการบันทึกข้อมูลจำนวนนับการทำโปรไฟล์ลงในDASDทุกครั้งที่มันมีค่าเพิ่มขึ้นเพราะการทำเช่นนั้นจะทำให้โปรแกรมทำงานช้าลงอย่างมาก. ค่าจำนวนนับจะถูกเขียนลงใน DASD เมื่อโปรแกรมถูก page out ตามปกติเท่านั้น. เพื่อให้แน่ใจว่าค่าจำนวนนับการทำโปรไฟล์ถูกเขียนลงใน DAS ให้ใช้คำสั่ง Clear Pool (CLRPOOL) เพื่อล้างพูลของหน่วยความจำที่โปรแกรมนั้นทำงานอยู่.

การใช้ข้อมูลการทำโปรไฟล์ที่รวบรวมได้

การใช้ข้อมูลการทำโปรไฟล์ที่รวบรวมมา กระทำได้ดังนี้:

1. สั่งให้เครื่องใช้ข้อมูลการทำโปรไฟล์ที่รวบรวมได้ในการจัดลำดับใหม่ให้กับโพรซีเดอร์ (procedure order profiling data) ในโปรแกรม เพื่อให้เกิดประสิทธิภาพที่ดีที่สุด.
2. สั่งให้เครื่องใช้ข้อมูลข้อมูลการทำโปรไฟล์ที่รวบรวมได้ (basic block profiling data)เพื่อจัดลำดับโค้ดของโพรซีเดอร์ในโปรแกรมเพื่อให้เกิดประสิทธิภาพที่ดีที่สุด
3. ลบคำสั่งเครื่องจากโปรแกรมที่เพิ่มเข้ามาตอนที่ถูกตั้งค่าให้รวบรวมข้อมูลข้อมูลการทำโปรไฟล์. โปรแกรมดังกล่าวจะไม่สามารถรวบรวมข้อมูลโปรไฟล์ได้อีก .
4. บันทึกข้อมูลโปรไฟล์ที่รวบรวมได้ไว้ในโปรแกรมให้เป็นแบบobservableอันได้แก่:
   - *BLKORD (basic block profiling observability)
   - *PRCORD (procedure order profiling observability)

เมื่อข้อมูลที่รวบรวมมาถูกใช้กับโปรแกรม, มันจะไม่สามารถใช้ซ้ำได้. เพื่อที่จะใช้ข้อมูลได้อีกครั้ง คุณจำเป็นจะต้องทำตามขั้นตอนที่กำหนดไว้ในหัวข้อ “วิธีการในการจัดทำโปรไฟล์โปรแกรม” ในหน้า 162. **ข้อมูลใดๆ ที่ถูกป้อนไว้ก่อนหน้านี้จะถูกลบทิ้งไปเมื่อโปรแกรมถูก enableให้รวบรวมข้อมูลโปรไฟล์อีก.**

ถ้าคุณต้องการใช้ข้อมูลที่คุณรวบรวมเอาไว้อีกครั้งหนึ่ง, คุณควรจะทำสำเนาโปรแกรมไว้ก่อนที่จะลงข้อมูลโปรไฟล์ .วิธีนี้อาจจำเป็นต้องใช้ถ้าคุณกำลังทดลองหาประโยชน์ที่ได้รับจากการทำโปรไฟล์แต่ละแบบ (ทั้งแบบ block order และแบบ block and procedure order).

เพื่อใช้ข้อมูลโปรไฟล์, ให้ใช้คำสั่ง Change Program (CHGPGM) หรือ Change Service Program (CHGSRVPGM). สำหรับค่าพารามิเตอร์ profiling data (PRFDTA) ให้ใช้คำสั่ง:

- Block order profiling data (*APYBLKORD)
- ใช้คำสั่งสำหรับblock orderและ procedure profiling data(*APYALL) หรือใช้ (*APYPRCORD)

ไอบีเอ็ม แนะนำว่าควรจะใช้ *APYALL.

การใช้ข้อมูลโปรไฟล์จะเป็นการสร้างและบันทึกobservabilityสองชนิดลงในโปรแกรม . คุณสามารถลบobservabilityที่เกิดขึ้นด้วยคำสั่ง Change Program (CHGPGM) และ Change Service Program (CHGSRVPGM).

- *BLKORD observability ถูกเพิ่มเข้ามาเมื่อข้อมูลโปรไฟล์แบบblock order ถูกป้อนให้กับโปรแกรม. มันเป็นการอนุญาตให้เครื่องเก็บข้อมูลโปรไฟล์ block order ที่ลงให้โปรแกรมไว้ในกรณีที่ต้องมีการทำโปรแกรมใหม่อีก.
- การป้อนข้อมูลโปรไฟล์แบบ procedure order ให้กับโปรแกรมจะเป็นการเพิ่มobservabilityชนิด *PRCORD และ *BLKORDโดยทางอ้อม. มันเป็นการอนุญาตให้เครื่องเก็บข้อมูลโปรไฟล์procedure order ที่ลงให้โปรแกรมไว้ในกรณีที่ต้องมีการเปลี่ยนแลงหรือทำโปรแกรมใหม่ .

ตัวอย่างเช่น, คุณป้อนข้อมูลโปรไฟล์ block order ให้กับโปรแกรมของคุณ แล้วลบ *BLKORD observability ออกไป. ถือว่าโปรแกรมยังถูกทำโปรไฟล์แบบ block order อยู่. แต่ถ้ามีการเปลี่ยนแปลงใดๆ ที่ทำให้โปรแกรมถูกสร้างใหม่ มันจะพ้นจากสภาวะที่ถูกทำโปรไฟล์แบบ block order ไป.

หมายเหตุ: การลบ *CRTDTA observability จะทำให้ * BLKORD observability ถูกลบออกไปด้วย. นั่นก็เพราะค่า observalibity แบบ *BLKORD จะถูกใช้เมื่อโปรแกรมถูกทำใหม่

เท่านั้น. แต่เนื่องจากโปรแกรมไม่สามารถทำใหม่ได้ถ้าค่าobservabilityแบบ *CRTDTA ถูกลบออก, จึงทำให้ค่า *BLKORD ไม่มีความจำเป็นและถูกลบไปด้วย. แต่ *PRCORD observability จะไม่ถูกลบออกไป.

การจัดการกับโปรแกรมที่ตั้งค่าให้รวบรวมข้อมูลโปรไฟล์

การเปลี่ยนแปลงโปรแกรมที่ถูกทำให้รวบรวมข้อมูลโปรไฟล์ด้วยคำสั่ง Change Program (CHGPGM) หรือ Change Service Program(CHGSRVPGM) มีผลทางอ้อมทำให้จำนวนนับข้อมูลโปรไฟล์มีค่าเป็นศูนย์ถ้าการเปลี่ยนแปลงนั้นทำให้เกิดการทำใหม่กับโปรแกรม. ตัวอย่างเช่น, ถ้าคุณเปลี่ยนโปรแกรมที่ถูกทำให้รวบรวมข้อมูลโปรไฟล์จากระดับ optimization เป็น *FULL ไปยังระดับที่ 40, จะมี ข้อมูลโปรไฟล์ ที่ถูกรวบรวมไว้บางตัวถูกลบออกไป. เหตุการณ์นี้ยังจะเป็นจริงถ้าโปรแกรมที่ถูกทำให้รวบรวมข้อมูลโปรไฟล์ถูกrestore และมีการกำหนดค่า FRCOBJCVN(*YES *ALL) ไว้ในคำสั่ง Restore Object (RSTOBJ).

เช่นเดียวกัน, การอัพเดตโปรแกรมที่ถูกทำให้รวบรวมข้อมูลโปรไฟล์โดยการใช้คำสั่ง Update Program (UPDPGM) หรือ Update Service Program (UPDSRVPGM) จะทำให้จำนวนนับข้อมูลโปรไฟล์ถูกลบไปถ้าโปรแกรมที่เป็นผลลัพธ์นั้นยังคงถูกให้รวบรวมข้อมูลโปรไฟล์. ตัวอย่างเช่น, โปรแกรม P1 ประกอบด้วยโมดูล M1 และ M2. โมดูล M1 ที่ถูกโยงอยู่ใน P1 ถูกทำให้รวบรวมข้อมูลโปรไฟล์ แต่โมดูล M2 ไม่. ดังนั้นตราบใดที่มีโมดูลหนึ่งถูก enable อยู่, การอัพเดตโปรแกรม P1 ด้วยโมดูล M1 หรือ M2 จะส่งผลให้โปรแกรมยังถูก enable ให้รวบรวมข้อมูลโปรไฟล์. และจำนวนนับข้อมูลโปรไฟล์ทั้งหมดจะถูกลบออกไป. อย่างไรก็ตาม,ถ้าโมดูล M1 ถูกเปลี่ยนให้ไม่ให้รวบรวมข้อมูลโปรไฟล์อีกต่อไปโดยการกำหนดค่า *NOCOL ลงในพารามิเตอร์ profiling data (PRFDTA) ของคำสั่ง Change Module CHGMOD) , การอัพเดตโปรแกรม P1 ด้วย M1 จะส่งผลให้โปรแกรม P1ไม่ต้องรวบรวมข้อมูลโปรไฟล์ อีกต่อไป.

คุณสามารถลบจำนวนนับโปรไฟล์จากโปรแกรมได้โดยการกำหนดอ็อพชัน *CLR ลงในพารามิเตอร์ profiling data (PRFDTA) ของคำสั่ง Change Program (CHGPGM) หรือ Change Service Program (CHGSRVPGM). โปรดจำไว้ว่าโปรแกรมที่ใช้อ็อพชัน *CLR จะต้องไม่ถูก activate.

ถ้าคุณไม่ต้องการให้โปรแกรมรวบรวมข้อมูลโปรไฟล์อีกต่อไปคุณจะต้องทำตามวิธีการใดวิธีการหนึ่งดังนี้:

- ระบุค่าพารามิเตอร์ profiling data(PRFDTA) ในคำสั่ง Change Program (CHGPGM) เป็น *NOCOL.
- ระบุค่าพารามิเตอร์ profiling data(PRFDTA) ในคำสั่ง Change Service Program (CHGSRVPGM) เป็น *NOCOL.

การทำเช่นนี้จะเปลี่ยนโปรแกรมให้กลับไปสู่สถานะเดิมก่อนที่มันจะรวบรวมข้อมูลโปรไฟล์. และคุณสามารถเปลี่ยนค่า PRFDTA ของโมดูลไปเป็น *NOCOL ด้วยคำสั่ง CHGMOD หรือโดยการคอมไพล์โมดูลและรวมโมดูลเข้ากับโปรแกรมใหม่อีกครั้ง.

การจัดการกับโปรแกรมที่ได้รับข้อมูลโปรไฟล์

ถ้าโปรแกรมที่มีข้อมูลโปรไฟล์ถูกเปลี่ยนแปลงโดยการใช้คำสั่ง Change Program (CHGPGM) หรือ Change Service Program (CHGSRVPGM) คุณจะสูญเสียข้อมูลโปรไฟล์ที่ป้อนไว้หากเหตุการณ์ทั้งสองต่อไปนี้เป็นจริง:

- การเปลี่ยนแปลงที่บังคับให้โปรแกรมถูกทำซ้ำ.

  หมายเหตุ: ระดับการทำ optimization ของโปรแกรมที่มีการลงข้อมูลโปรไฟล์ไว้ จะไม่สามารถกำหนดให้ต่ำกว่าระดับที่ 30 ได้. เนื่องจากข้อมูลโปรไฟล์ขึ้นอยู่กับระดับของ optimization.

- Profiling observability ที่ต้องการถูกลบออกไป.

นอกจากนี้ ข้อมูลโปรไฟล์ที่ป้อนไว้ทั้งหมดยังจะถูกลบไป ถ้าการร้องขอเพื่อเปลี่ยนแปลงนี้เป็นการ enable โปรแกรม ให้รวบรวมข้อมูลโปรไฟล์, โดยไม่สนใจว่า โปรไฟล์แบบ observability จะถูกลบไปแล้วหรือไม่. ดังนั้นคำร้องขอนี้จะมีผลให้โปรแกรมถูก enable ให้รวบรวมข้อมูลโปรไฟล์.

ดังตัวอย่างต่อไปนี้:

- โปรแกรม A มีข้อมูลของโปรไฟล์แบบ procedure order และ block order ลงไว้. มี *BLKORD observability ถูกลบออกจากโปรแกรมแต่ *PRCORD observability ยังคงอยู่. ใช้คำสั่ง CHGPGM เพื่อปรับค่าแอ็ตทริบิวต์การเก็บข้อมูลการทำงานของโปรแกรม A ,ซึ่งเป็นผลให้โปรแกรมถูกทำซ้ำ. การเปลี่ยนแปลงนี้จะทำให้โปรแกรม A ไม่ถูกทำโปรไฟล์แบบ block order อีกต่อไป. แต่อย่างไรก็ตามยังคงมีข้อมูลโปรไฟล์แบบ procedure อยู่กับโปรแกรม.
- โปรแกรม A มีข้อมูลโปรไฟล์แบบ procedure order และ block order ลงไว้. ทั้ง *BLKORD และ *PRCORD observability ถูกลบออกจากโปรแกรม A. ใช้คำสั่ง CHGPGM เพื่อปรับค่าแอ็ตทริบิวต์โปรไฟล์ผู้ใช้ของโปรแกรมA ,ซึ่งเป็นผลให้โปรแกรมถูกทำซ้ำ. การเปลี่ยนแปลงนี้จะทำให้โปรแกรม A ไม่ถูกจัดทำโปรไฟล์แบบblock order หรือแบบ procedure order อีกต่อไป. โปรแกรม A จะกลับไปสู่สถานะที่เคยเป็นก่อนที่จะมีการป้อนข้อมูลโปรไฟล์.
- โปรแกรม A มีข้อมูลโปรไฟล์แบบ block order ลงอยู่. มี *BLKORD observability ถูกลบออกจากโปรแกรม. ใช้คำสั่ง CHGPGM เพื่อเปลี่ยนข้อความในโปรแกรมA , ซึ่งจะ**ไม่**เป็นผลให้โปรแกรมถูกทำซ้ำ.หลังจากนั้น, โปรแกรม A จะยังอยู่ในสถานะที่ถูกโปรไฟล์แบบ block order.
- โปรแกรม A มีการลงข้อมูลโปรไฟล์แบบ procedure order และ block order. ซึ่งค่า observability แบบ *PRCORD และ *BLKORD จะไม่ถูกลบ ออกจากโปรแกรม. ใช้คำสั่ง CHGPGM เพื่อให้โปรแกรมเริ่มรวบรวมข้อมูลโปรไฟล์(จะมีการทำซ้ำโปรแกรม). จะทำให้โปรแกรม A ไม่ถูกทำโปรไฟล์ทั้งแบบ block order และ procedure order อีกต่อไป. เป็นการทิ้งให้โปรแกรมอยู่ในสถานะเหมือนกับว่าไม่เคยมีการป้อนข้อมูลโปรไฟล์เลย. การเปลี่ยนแปลงนี้เป็นการ enable โปรแกรมให้รวบรวมข้อมูลโปรไฟล์โดยที่ล้างจำนวนนับข้อมูลโปรไฟล์ออกหมด.

โปรแกรมที่มีข้อมูลโปรไฟล์ (*APYALL, *APYBLKORD หรือ *APYPRCORD) ลงไว้ จะไม่สามารถเปลี่ยนสถานะทันที ให้เป็นโปรแกรมที่ไม่ถูกทำโปรไฟล์ด้วยการกำหนดค่า PRFDTA (*NOCOL) ในคำสั่ง CHGPGM หรือคำสั่ง CHGSRVPGM ได้. วิธีนี้มีไว้เพื่อป้องกันการสูญหายของข้อมูลการทำโปรไฟล์. แต่ถ้าต้องการทำจริงๆ ต้องตั้งค่าโปรแกรมเป็น PRFDTA(*COL), ซึ่งมีผลให้ข้อมูลการทำโปรไฟล์เดิมถูกลบไปก่อน, แล้วค่อยเปลี่ยนค่าเป็น PRFDTA(*NOCOL).

วิธีการในการแสดงว่าโปรแกรมหรือโมดูลถูกทำโปรไฟล์ หรือถูกตั้งค่าให้รวบรวมข้อมูลโปรไฟล์

ใช้คำสั่ง Display Program (DSPPGM) หรือ Display Service Program (DSPSRVPGM), และระบุค่า DETAIL(*BASIC) เพื่อพิจารณาแอ็ตทริบิวต์ program profiling data ของโปรแกรม. ค่าของ ″Profiling data″ จะเป็นค่าใดค่าหนึ่งดังนี้:

- *NOCOL – หมายถึง โปรแกรมไม่ถูก enable ให้รวบรวมข้อมูลโปรไฟล์.
- *COL – หมายถึง มีโมดูลตั้งแต่ 1 โมดูลขึ้นไปในโปรแกรมที่ถูก enable ให้รวบรวมข้อมูลโปรไฟล์. จะไม่มีการแสดงค่านี้ถ้าข้อมูลโปรไฟล์ ถูกรวบรวมไว้แล้ว.
- *APYALL – หมายถึงมีข้อมูลโปรไฟล์แบบ block order และ procedure order ถูกป้อนให้กับโปรแกรม. และการรวบรวมข้อมูลโปรไฟล์จะไม่ถูก enable อีกต่อไป.
- *APYBLKORD – หมายถึงมีข้อมูลโปรไฟล์แบบ block order ถูกป้อนให้กับโพรซีเดอร์ของโมดูลอย่างน้อย 1 โมดูลที่ถูกรวมเข้ากับโปรแกรม. ค่านี้ถูกใช้เมื่อโมดูลที่ถูกรวมนี้เคยถูก enable ให้รวบรวมข้อมูลโปรไฟล์มาก่อนเท่านั้น. และการรวบรวมข้อมูลการทำโปรไฟล์จะไม่ถูกenableอีกต่อไป.
- *APYPRCORD– หมายถึงข้อมูลโปรไฟล์แบบ procedure orderถูกป้อนให้กับโปรแกรมนี้. การรวบรวมข้อมูลโปรไฟล์จะไม่ถูก enable อีกต่อไป.

ถ้าต้องการให้มีเฉพาะโปรไฟล์แบบ procedure order ใช้กับโปรแกรม:

- เริ่มต้นด้วยการระบุค่า *APYALL หรือ *APYPRCORD (ซึ่งมีค่าเท่ากับ *APYALL).
- ลบค่าobservability *BLKORD ออกแล้วทำการสร้างดปรแกรมใหม่.

เพื่อแสดงค่าแอ็ตทริบิวต์ profiling data ของโมดูลที่เชื่อมโยงอยู่ในโปรแกรม, ให้ใช้คำสั่ง DSPPG หรือ DSPSRVPGM DETAIL(*MODULE).ระบุอ็อพชัน 5 ในโมดูลที่เชื่อมโยงกับโปรแกรมเพื่อแสดงค่าพารามิเตอร์นี้ในระดับของโมดูล.ค่าของ ″Profiling data″ จะเป็นค่าใดค่าหนึ่งดังนี้:

- *NOCOL – หมายถึง โมดูลที่ถูกรวมนี้ไม่ถูก enable ให้รวบรวมข้อมูลโปรไฟล์.
- *COL – หมายถึง โมดูลที่ถูกรวมนี้ถูก enable ให้รวบรวมข้อมูลโปรไฟล์. จะไม่มีการแสดงค่านี้ถ้าข้อมูลโปรไฟล์ถูกรวบรวมไว้แล้ว.
- *APYBLKORD – หมายถึงข้อมูลโปรไฟล์แบบ block order ถูกป้อนให้กับโพรซีเดอร์อย่างน้อย 1 โพรซีเดอร์ ที่เป็นของโมดูลที่ถูกรวมเข้ากับโปรแกรม. และการรวบรวมข้อมูลโปรไฟล์จะไม่ถูก enable อีกต่อไป.

นอกจากนี้พารามิเตอร์ DETAIL(*MODULE) จะแสดงผลของ field ต่างๆ ที่แสดงถึงจำนวนของโพรซีเดอร์ที่ได้รับผลกระทบจากแอ็ตทริบิวต์ของ program profiling data ดังนี้.

- จำนวนของโพรซีเดอร์ หมายถึง จำนวนของโพรซีเดอร์ทั้งหมดในโมดูล.
- จำนวนprocedures block reordered หมายถึงจำนวนของโพรซิเดอร์ในโมดูลที่เป็นbasic block reordered.
- จำนวนของ procedures block order measured หมายถึงจำนวนของโพรซีเดอร์ในโมดูลนี้ที่มีข้อมูลโปรไฟล์ block order ที่รวบรวมไว้เมื่อมีการป้อนข้อมูล block order. เมื่อเบนช์มาร์กถูกรัน,

มันจะต้องเป็นกรณีที่ไม่มีข้อมูลถูกรวบรวมไว้สำหรับโพรซีเดอร์ หนึ่งโดยเฉพาะ เนื่องจากโพรซีเดอร์จะไม่ถูกกระทำในเบนช์มาร์ก. ดังนั้นจำนวนที่แสดงนี้จะหมายถึงจำนวนที่แท้จริงของโพรซีเดอร์ที่ถูกกระทำด้วยเบนช์มาร์ก.

ใช้คำสั่ง DSPMOD เพื่อพิจารณาแอ็ตทริบิวต์ในการทำโปรไฟล์ของโมดูล. ค่าของ "Profiling data" จะเป็นค่าใดค่าหนึ่งที่แสดงไว้ต่อไปนี้. โดยจะไม่แสดงค่า *APYBLKORD เพราะว่า basic block data สามารถถูกป้อนให้กับโมดูลที่รวมอยู่ในโปรแกรมเท่านั้น.

- *NOCOL – หมายถึง โมดูลไม่ถูก enable ให้รวบรวมข้อมูลโปรไฟล์.
- *COL – หมายถึง โมดูลถูก enable ให้รวบรวมข้อมูลโปรไฟล์.

การวิเคราะห์ระหว่างโพรซีเดอร์ (Interprocedural Analysis - IPA)

ในหัวข้อนี้เราจะกล่าวถึงภาพรวมของกระบวนการ Interprocedural Analysis (IPA) ที่ได้จัดเตรียมไว้โดยผ่านทางอ็อพชัน IPA ในคำสั่ง CRTPGM และ CRTSRVPGM.

ในเวลาคอมไพล์,optimizing translator จะทำการวิเคราะห์ระหว่างโพรซีเดอร์และภายในโพรซีเดอร์. การวิเคราะห์ภายในโพรซีเดอร์ (Intraprocedural Analysis) เป็นกลไกสำหรับการทำ Optimization สำหรับฟังก์ชันแต่ละฟังก์ชันในยูนิตคอมไพล์จะใช้เฉพาะข้อมูลที่มีอยู่สำหรับฟังก์ชันและยูนิตคอมไพล์เท่านั้น. การวิเคราะห์ระหว่างโพรซีเดอร์เป็นกลไกสำหรับการทำข้ามขอบเขตของฟังก์ชัน. โดย optimizing translator จะทำการวิเคราะห์ระหว่างโพรซีเดอร์ แต่ทำเฉพาะภายในยูนิตคอมไพล์เท่านั้น. การวิเคราะห์ระหว่างโพรซีเดอร์ถูกเรียกใช้งานโดยอ็อพชันคอมไพเลอร์ของ IPA เพื่อเพิ่มประสิทธิภาพให้กับข้อจำกัดของการวิเคราะห์ระหว่างโพรซีเดอร์ที่กล่าวถึงข้างต้น. เมื่อคุณรันการวิเคราะห์ระหว่างโพรซีเดอร์ผ่านทางอ็อพชัน IPA แล้ว IPA ก็จะทำการ optimize ในโปรแกรมทั้งหมด. มันยังสามารถทำการ optimize นอกจากในเวลาคอมไพล์ได้โดยใช้ด้วยตัวแปล optimizing translator. ตัวแปล optimizing translator หรืออ็อพชัน IPA สามารถทำการ Optimization ได้หลายแบบดังนี้:

- *การทำ inline ตลอดหน่วยการทำคอมไพล์.* การ Inline เป็นการแทนที่ฟังก์ชันที่เรียกใช้ด้วยโค้ดของฟังก์ชันโดยตรง. วิธีการ Inline ไม่เพียงแต่จะช่วยลดโอเวอร์เฮดของการเรียกเท่านั้น, แต่ยังทำให้ทั้งฟังก์ชันสามารถเห็นได้โดยผู้เรียก ซึ่งช่วยให้คอมไพเลอร์สร้างโค้ดที่ optimize ให้คุณได้ดียิ่งขึ้น.
- *การพาร์ติชันโปรแกรม.* พาร์ติชันของโปรแกรมเป็นการเพิ่มประสิทธิภาพด้วยการจัดลำดับฟังก์ชันเพื่อใช้ประโยชน์จากการอ้างถึงแบบโลคัล. พาร์ติชันจะนำฟังก์ชันที่มีการเรียกใช้บ่อยๆ มาอยู่ติดๆ กันในหน่วยความจำ. สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับพาร์ติชันของโปรแกรม โปรดดูในหัวข้อ "พาร์ติชันที่สร้างโดย IPA" ในหน้า 175.
- *การรวมตัวแปรแบบโกลบอล.* คอมไพเลอร์นำตัวแปรโกลบอลไปไว้ในโครงสร้างข้อมูล และเข้าถึงตัวแปรนี้ด้วยการคำนวณอ็อฟเซ็ตจากจุดเริ่มต้นของโครสร้างข้อมูล. ซึ่งช่วยลดเวลาในการเข้าถึงตัวแปร และทำให้ตัวแปรเป็นแบบโลคัล.
- *การทำให้โปรแกรมทำงานกระชับขึ้น.* การทำให้การทำงานของโปรแกรมของคุณเป็นแบบตรงไปตรงมาที่สุด.
- *การกำจัดโค้ดที่ไม่ได้ใช้.* การกำจัดโค้ดที่ไม่ได้ใช้งานโดยลบโค้ดที่ไม่ได้ใช้ในฟังก์ชันทิ้งไป.

- *การเรียกกราฟที่ตัดทอนฟังก์ชันที่ไม่ได้ใช้งาน.* การเรียกกราฟที่ตัดทอนฟังก์ชันที่ไม่ได้ใช้งานออกจากโค้ดโดยทำการ Inline แบบ 100 % หรือไม่มีการอ้างถึงเลย.
- *การกระจายค่าคงที่แบบIntraprocedural และการกระจายเซ็ต.* IPA จะกระจายค่าคงที่แบบ Floating Point และจำนวนเต็มที่ใช้ในการคำนวณนิพจน์คงที่ในเวลาคอมไพล์. นอกจากนี้ ตัวแปรที่ใช้ซึ่งทราบว่าเป็นหนึ่งในค่าคงที่หลายๆ ตัวสามารถให้ผลลัพธ์ในเงื่อนไข และสวิตซ์ที่เคลื่อนย้ายได้.
- *การวิเคราะห์ค่า alias ของพอยเตอร์แบบ Intraprocedural.* IPA จะตรวจสอบนิยามของพอยเตอร์ที่ใช้อยู่ เพื่อหาข้อมูลที่แน่ชัดเกี่ยวกับตำแหน่งของหน่วยความจำที่การแปลงพอยเตอร์อาจถูกใช้ หรือทำการนิยามไว้. ซึ่งช่วยให้ส่วนอื่นๆ ของคอมไพเลอร์สามารถสร้างโค้ดที่ optimize ในจุดที่มีการแปลงพอยเตอร์. IPA ยังตรวจสอบนิยามของข้อมูลและพอยเตอร์ของฟังก์ชันด้วย. เมื่อพอยเตอร์สามารถอ้างถึงได้เฉพาะในตำแหน่งหน่วยความจำหรือ ฟังก์ชันเพียงแห่งเดียว, IPA จะทำการเปลี่ยนแปลงโดยการอ้างถึงตำแหน่งหน่วยความจำหรือ ฟังก์ชันนั้นด้วยตัวเอง.
- *การกระจายตัวของ Intraprocedural copy* IPA จะกระจายนิพจน์ และนิยามตัวแปรบางตัวสำหรับการใช้ตัวแปรดังกล่าว. เพื่อช่วยเพิ่มโอกาสสำหรับการเคลื่อนย้ายของนิพจน์ที่คงที่. มันยังกำจัดความซ้ำซ้อนของตัวแปรที่คัดลอกด้วย.
- *การกำจัดโค้ดแบบ intraprocedural ที่ไม่สามารถเข้าถึง และการกำจัด store.* IPA จะลบนิยามของตัวแปรที่ไม่สามารถเข้าถึงได้ออกไป พร้อมการคำนวณที่เกี่ยวข้องกับนิยามมนั้น.
- *การแปลงอากิวเมนต์อ้างอิง(แอดเดรส)ให้เป็น value arguments.* IPA จะแปลงอากิวเมนต์อ้างอิง (แอดเดรส) ให้เป็นค่าอากิวเมนต์ เมื่อพารามิเตอร์ที่คุณกำหนดไม่ได้เขียนไว้เพื่อเรียกโพรซีเดอร์.
- *การแปลงตัวแปรสแตติกให้เป็นตัวแปร(stack) variables.* IPA จะแปลงตัวแปรสแตติกให้เป็นตัวแปร (stack) อัตโนมัติ เมื่อการใช้งานตัวแปรเหล่านั้นจำกัดเฉพาะในการเรียกโพรซีเดอร์เพียงอันเดียว.

โค้ดสำหรับรันไทม์ที่ถูก Optimize โดยใช้ IPA มักจะเร็วกว่าโค้ดโค้ดที่ทำการ Optimize เฉพาะในเวลาคอมไพล์. แต่ไม่ใช่ว่าแอ็พพลิเคชันทุกประเภทจะเหมาะกับการทำ Optimize โดย IPA อย่างไรก็ตาม ประสิทธิภาพที่เพิ่มขึ้นซึ่งได้รับจากการใช้ IPA อาจแตกต่างกัน. สำหรับในบางแอ็พพลิเคชันประสิทธิภาพของแอ็พพลิเคชันอาจไม่เพิ่มขึ้นเมื่อใช้การวิเคราะห์ระหว่างโพรซีเดอร์. และในบางกรณี (ค่อนข้างหายาก), ประสิทธิภาพของแอ็พพลเคชันอาจลดลงได้เมื่อการวิเคราะห์ระหว่างโพรซีเดอร์. ถ้าเกิดเหตุการณ์เช่นนี้ขึ้น, เราขอแนะนำว่าคุณไม่ควรใช้การวิเคราะห์ระหว่างโพรซีเดอร์. การเพิ่มประสิทธิภาพที่ได้จากการวิเคราะห์ระหว่างโพรซีเดอร์จะขึ้นอยู่กับประเภทของแอ็พพลิเคชัน. โดยแอ็พพลิเคชันที่น่าจะได้รับประสิทธิภาพสูงขึ้นก็คือแอ็พพลิเคชันที่มีลักษณดังนี้:

- มีฟังก์ชันจำนวนมาก
- มียูนิตคอมไพล์จำนวนมาก
- มีฟังก์ชันจำนวนมากที่ไม่อยู่ในยูนิตคอมไพล์เดียวกันกับผู้เรียก
- ไม่ควรใช้งานกับโปรแกรมที่มีการทำงานด้านอินพุตและเอาต์พุตมากๆ

การ Optimization ระหว่างโพรซีเดอร์สามารถใช้งานได้เฉพาะกับโปรแกรม ILE และ เซอร์วิสโปรแกรมที่มีเงื่อนไขตามนี้:

- คุณทำการสร้างโมดูลที่รวมกับโปรแกรมหรือเซอร์วิสโปรแกรมโดยเฉพาะสำหรับ V4R4M0 หรือรีลีสใหม่กว่า.

- คุณทำการคอมไพล์โมดูลที่รวมโปรแกรมหรือเซอร์วิสโปรแกรมที่มีระดับ Optimization เป็น 20 (*BASIC) หรือสูงกว่านั้น.
- โมดูลที่รวมกับโปรแกรมหรือเซอร์วิสโปรแกรมมีข้อมูล IL ที่สัมพันธ์กับมัน. ให้ใช้อ็อพชันในการสร้างโมดูล MODCRTOPT(*KEEPILDTA) เพื่อรักษาข้อมูล Intermediate language (IL) ไว้กับโมดูล.

หมายเหตุ: เนื่องจากข้อกำหนดในการ optimize คุณควรทำการดีบักโปรแกรมของคุณก่อนที่จะใช้การวิเคราะห์ระหว่างโพรซีเดอร์.

วิธีการ Optimize โปรแกรมของคุณด้วย IPA

เมื่อคุณต้องการใช้ IPA เพื่อ Optimize โปรแกรมหรือเซอร์วิสโปรแกรมของคุณ, ให้ทำตามขั้นตอนดังต่อไปนี้:

1. ตรวจดูให้แน่ใจว่าคุณได้ทำการคอมไพล์โมดูลที่จำเป็นสำหรับโปรแกรมหรือเซอร์วิสโปรแกรมด้วย MODCRTOPT(*KEEPILDTA) และเลือก Optimization Level เป็น 20 หรือสูงกว่านั้น (แนะนำให้เลือก 40). คุณสามารถใช้คำสั่ง DSPMOD กับพารามิเตอร์ DETAIL (*BASIC) เพื่อตรวจสอบว่าโมดูลเดี่ยวได้ถูกคอมไพล์กับ อ็อพชันที่ถูกต้องหรือไม่. ฟิลด์ **Intermediate language data** ต้องกำหนดให้เป็น *YES ถ้าคุณมีข้อมูล IL อยู่ด้วย. ส่วนฟิลด์ **Optimization level** จะแสดงระดับของการ Optimization ในโมดูลนั้นๆ.
2. หากคุณระบุ IPA(*YES) ในคำสั่ง CRTPGM หรือ CRTSRVPGM. เมื่อส่วนการรวมของ IPA ทำงาน, ระบบจะแสดงข้อความสถานะเพื่อรายงานความคืบหน้าของ IPA.

คุณสามารถกำหนดเพิ่มเติมเกี่ยวกับวิธีที่ IPA optimize โปรแกรมของคุณโดยใช้ พารามิเตอร์ต่อไปนี้:

- ระบุ IPACTLFILE(*IPA-control-file*) เพื่อจัดเตรียมข้อมูลของซับอ็อพชันของ IPA เพิ่มเติม. โปรดดูในหัวข้อ “ไวยากรณ์ของไฟล์ควบคุมของ IPA”สำหรับรายการอ็อพชันทั้งหมดที่คุณสามารถกำหนดได้ในไฟล์ควบคุม.

เมื่อคุณระบุ IPA(*YES) ในคำสั่ง CRTPGM แล้ว คุณไม่สามารถอัพเดตโปรแกรม (คุณไม่สามารถระบุ ALWUPD(*YES)). ซึ่งเป็นจริงสำหรับพารามิเตอร์ ALWLIBUPD ในคำสั่ง CRTSRVPGM. ถ้าระบุพร้อมกับ IPA(*YES), พารามิเตอร์นี้ต้องเป็น ALWLIBUPD(*NO).

ไวยากรณ์ของไฟล์ควบคุมของ IPA

ไฟล์ควบคุมของ IPA กระแสของไฟล์ที่มีไดเรกทีฟของ IPA เพิ่มเติม. ไฟล์ควบคุมอาจเป็น สมาชิกของไฟล์, และใช้หลักการตั้งชื่อ QSYS.LIB (ตัวอย่างเช่น, `/qsys.lib/mylib.lib/xx.file/yy.mbr`). พารามิเตอร์ IPACTLFILE จะระบุชื่อพาธของไฟล์นี้.

IPA จะแสดงข้อความแสดงความผิดพลาด ถ้าไดเรกทีฟของไฟล์ควบคุมมีไวยากรณ์ที่ไม่ถูกต้อง.

คุณสามารถระบุไดเรกทีฟต่อไปนี้ได้ในไฟล์ควบคุม:

exits=name[,name]

ใช้ระบุรายชื่อฟังก์ชันที่เป็นจุดสิ้นสุดของโปรแกรม. คุณสามารถทำการ optimize การเรียก

ฟังก์ชัน (เช่น โดยการจัดลำดับของการบันทึกและเรียกคืน) เนื่องจากการเรียกนั้นจะไม่มีการคืนค่ากลับมาให้โปรแกรม. ฟังก์ชันเหล่านี้ต้องไม่เรียกส่วนอื่นของโปรแกรมที่มีข้อมูล IL ซึ่งสัมพันธ์กับมัน.

inline=attribute

ใช้ระบุวิธีการที่คุณต้องการให้คอมไพเลอร์ตรวจสอบฟังก์ชันที่คุณต้องการทำ Inline. คุณสามารถระบุแอ็ตทริบิวต์ดังต่อไปนี้ได้ในไดเรกทีฟ:

auto ใช้ระบุให้การทำ Inline ควรตรวจดูว่า ถ้าฟังก์ชันสามารถ inline ในแบบพื้นฐานของค่า inline-limit และ inline-threshold. ไดเรกทีฟ noinline จะลบล้างการทำ Inline แบบอัตโนมัติ. นี่เป็นค่าดีฟอล์ต.

noauto ใช้ระบุให้ IPA พิจารณาการทำ Inline เฉพาะฟังก์ชันที่คุณระบุเท่านั้น ตามรายชื่อพร้อมกับไดเรกทีฟ.

name[,name]

ใช้ระบุรายชื่อฟังก์ชันที่คุณต้องการให้ทำ Inline. โดยฟังก์ชันนั้นอาจเป็น Inline หรือไม่ก็ได้.

name[,name] from name[,name]

ใช้ระบุรายชื่อฟังก์ชันที่คุณต้องการเสนอให้ทำ Inline ถ้าฟังก์ชันจำเพาะ หรือรายการฟังก์ชันที่เรียก. โดยฟังก์ชันนั้นอาจเป็น Inline หรือไม่ก็ได้.

inline-limit=num

ใช้ระบุขนาดสัมพัทธ์สูงสุด (ในยูนิตโค้ดแบบ abstract) ที่ฟังก์ชันสามารถขยายได้ก่อนการทำ Inline จะหยุด. โค้ดแบบ Abstract จะสัมพันธ์กับขนาดของโค้ดแบบ Executable ในฟังก์ชันนี้. ยิ่งสูงก็จะยิ่งช่วยให้คอมไพเลอร์สามารถทำการ Inline โปรแกรมย่อยขนาดใหญ่ได้, หรือเรียกหลายๆ โปรแกรมย่อยได้ หรือทั้งสองแบบ. ไดเรกทีฟเหล่านี้สามารถใช้งานได้ก็ต่อเมื่อกำหนด inline=auto. ขนาดดีฟอล์ตคือ 8192.

inline-threshhold=size

ใช้ระบุขนาดสูงสุด (ในยูนิตโค้ดแบบ abstract) ของฟังก์ชันที่ได้รับการเสนอให้ทำ Inline แบบอัตโนมัติ. ไดเรกทีฟเหล่านี้สามารถใช้งานได้ก็ต่อเมื่อกำหนด inline=auto. ขนาดดีฟอล์ตคือ 1024.

isolated=name[,name]

ใช้ระบุชื่อของฟังก์ชัน "isolated". ซึ่งฟังก์ชัน Isolated เป็นฟังก์ชันที่ไม่มีการเรียกโดยตรง (หรือโดยอ้อมผ่านทางฟังก์ชันอื่นภายในการเรียงแบบต่อเนื่อง) อ้างถึงหรือเปลี่ยนแปลงตัวแปรโกลบอลที่สามารถเข้าถึงได้จากฟังก์ชันที่มองเห็น. IPA เข้าใจว่าฟังก์ชันที่รวมจากเซอร์วิสโปรแกรมถูก Isolate.

lowfreq=name[,name]

ใช้ระบุชื่อของฟังก์ชันที่คุณคาดว่าจะเรียกใช้งานไม่บ่อยนัก. ซึ่งเป็นการป้องกันข้อพิดพลาด และติดตามฟังก์ชัน. IPA สามารถทำให้ส่วนอื่นๆ ของโปรแกรมทำงานได้รวดเร็วยิ่งขึ้นโดนการทำการ optimize ฟังก์ชันเหล่านี้เพียงเล็กน้อย.

missing=attribute

ใช้ระบุพฤติกรรมระหว่างโพรซีเดอร์ของฟังก์ชัน *Missing*. ฟังก์ชัน missing ฟังก์ชันที่หาย

ไปคือฟังก์ชันที่ไม่มีข้อมูล IL เกี่ยวข้อง และไม่ได้กำหนดชื่อในไดเรกทีฟ unknown, safe, isolated, หรือ pure. ซึ่งไดเรกทีฟจะกำหนดว่า IPA สามารถทำ Optimize ได้มากน้อยแค่ไหนถึงปลอดภัยในการเรียกไปยังรูทีนของไลบรารีที่ไม่มีข้อมูล IL ที่สัมพันธ์กับมัน.

IPA จะไม่สามารถเข้าถึงโค้ดที่อยู่ในฟังก์ชันเหล่านี้. คุณต้องแน่ใจว่าผู้ใช้ทั้งหมดที่อ้างถึงได้รับการ Resolve ด้วยไลบรารีของผู้ใช้ หรือรันไทม์ไลบรารี.

การกำหนดค่าดีฟอล์ตสำหรับไดเรกทีฟคือ `unknown`. ซึ่งค่า *Unknown* จะสั่งให้ IPA ระมัดระวังเกี่ยวกับการใช้ และเปลี่ยนแปลงข้อมูลผ่านการเรียกฟังก์ชันที่หายไป และเกี่ยวกับฟังก์ชันที่อาจถูกเรียกทางอ้อมเช่นกัน. คุณสามารถระบุแอ็ตทริบิวต์ดังต่อไปนี้ได้ในไดเรกทีฟ:

unknown
ใช้ระบุฟังก์ชัน "unknown" ที่หายไป. โปรดดูคำอธิบายเกี่ยวกับไดเรกทีฟ unknown ด้านล่าง. นี่เป็นค่าดีฟอล์ตของแอ็ตทริบิวต์.

safe ใช้ระบุฟังก์ชัน "safe" ที่หายไป. โปรดดูคำอธิบายเกี่ยวกับไดเรกทีฟ save ด้านล่าง.

isolated
ใช้ระบุฟังก์ชัน "isolate" ที่หายไป. โปรดดูคำอธิบายเกี่ยวกับไดเรกทีฟ Isolate ด้านบน.

pure ใช้ระบุฟังก์ชัน "pure" ที่หายไป. โปรดดูคำอธิบายเกี่ยวกับไดเรกทีฟ pure ด้านล่าง.

noinline=name[,name]
ใช้ระบุรายชื่อของฟังก์ชันที่คอมไพเลอร์ไม่ต้องทำ Inline.

noinline=name[,name] from name[,name]
ใช้ระบุรายชื่อของฟังก์ชันที่คอมไพเลอร์ไม่ต้องทำ Inline, ถ้าฟังก์ชันถูกเรียกจากฟังก์ชันที่จำเพาะ หรือรายการของฟังก์ชัน.

partition=small| medium|large|unsigned-integer
ใช้ระบุขนาดของแต่ละโปรแกรมพาร์ติชันที่ IPA สร้างขึ้น. ขนาดของพาร์ติชันจะขึ้นอยู่กับเวลาที่ใช้ในการลิงค์ และคุณภาพของโค้ดที่สร้าง. เมื่อพาร์ติชันมีขนาดใหญ่, เวลาที่ใช้ในการลิงค์ก็จะนานกว่า และคุณภาพของโค้ดที่ดีกว่า.

ค่าดีฟอล์ตสำหรับไดเรกทีฟคือ Medium.

สำหรับการควบคุมที่ต้องการความละเอียด, คุณก็สามารถใช้ค่า unsigned-integer ในการระบุขนาดของพาร์ติชัน. จำนวนเต็มอยู่ในหน่วยของค่าสมบูรณ์, และอาจเปลี่ยนแปลงได้ในระหว่างรีลีส. คุณควรใช้เฉพาะจำนวนต็มสำหรับเทอมเล็กๆ, หรือในสถานการณ์ที่จำนวนของพาร์ทิชันคงที่.

pure=name[,name]
ใช้ระบุรายชื่อฟังก์ชัน *pure*. ฟังก์ชันเหล่านี้ Safe และ Isolate. ซึ่งฟังก์ชัน pure จะไม่มีช่วงเริ่มที่สังเกตได้. ซึ่งหมายความว่าค่าที่ส่งกลับมาในการเรียกฟังก์ชันจะไม่ขึ้นกับการเรียกฟังก์ชันในอดีต และอนาคต.

safe=name[,name]

ใช้ระบุรายชื่อฟังก์ชัน *safe*. ฟังก์ชันเหล่านี้ไม่ได้ถูกเรียกโดยตรง หรือเรียกฟังก์ชันอื่นที่ข้อมูล IL สัมพันธ์กับมันโดยอ้อม. ฟังกชัน safe อาจอ้างถึงและเปลี่ยนแปลงตัวแปรโกลบอลก็ได้.

unknown=name[,name]

ใช้ระบุรายชื่อฟังก์ชัน *unknown*. ซึ่งฟังก์ชันเหล่านี้ไม่ปลอดภัย, Isolated หรือ Pure.

ข้อสังเกตในการใช้ IPA

- การใช้ IPA สามารถเพิ่มเวลาในการรวม. ซึ่งขึ้นอยู่กับขนาดของแอ็พพลิเคชัน และความเร็วของโพรเซสเซอร์ของคุณ ช่วงเวลาในการรวมอาจเพิ่มคุรจนคุณสังเกตได้.
- IPA สามารถสร้างการรวมโปรแกรมและเซอร์วิสโปรแกรมที่มีขนาดใหญ่กว่าการรวมในแบบปกติ.
- ในขณะที่การ Optimize ในระหว่างโพรซิเดอร์ของ IPA สามารถเพิ่มประสิทธิภาพของโปรแกรมได้อย่างชัดเจน, แต่มันก็อาจสามารถทำให้การทำงานของโปรแกรมที่มีข้อผิดพลาดล้มเหลวได้.
- เนื่องจาก IPA จะคอมไพล์ฟังก์ชันแบบ Inline, ดังนั้นให้คุณระวังเมื่อใช้ API ที่ยอมรับการใช้ Relative Stack Frame Offset (เช่น QMHRCVPM).
- ในการคอมไพล์ฟังก์ชัน inline, IPA ใช้ inliner ของ IPA เองแทนที่จะใช้ ตัวแทรกภายในส่วนเสริมหลัง. พารามิเตอร์ใดๆ ที่ให้สำหรับ backend inliner, เช่นการใช้ อ็อพชัน INLINE ในคำสั่งคอมไพล์, จะถูกข้าม. พารามิเตอร์สำหรับ ตัวแทรกภายใน IPA จะให้ในไฟล์ควบคุม IPA.

ข้อกำหนดและข้อจำกัดของ IPA

- คุณไม่สามารถใช้ UPDPGM หรือ UPDSRVPGM บนบาวน์โปรแกรมหรือเซอร์วิสโปรแกรมที่ถูก IPA ทำการ Optimize แล้วได้.
- คุณไม่สามารถดีบักโปรแกรมหรือเซอร์วิสโปรแกรมที่ IPA ได้ optimize ด้วย source debug facilities แบบปกติ. นี่เป็นเพราะ IPA ไม่ได้ดูแลข้อมูลในการดีบักที่อยู่ภายในข้อมูล IL ซึ่งจริงๆ แล้วมันลบข้อมูลในการดีบักทิ้งไปหมดเมื่อมันทำการสร้างพาร์ติชันเอาต์พุต. ด้วยเหตุนี้, โปรแกรมดีบักจึงไม่สามารถจัดการกับโปรแกรมหรือเซอร์วิสโปรแกรมที่ใช้ IPA ได้.
- พาร์ติชันเอาต์พุต จำกัดจำนวนอยู่ที่ 10,000 พาร์ติชัน. ถ้าคุณสร้างพาร์ติชันจนมาถึงข้อจำกัดนี้ ,การรวมกันก็จะล้มเหลว, และระบบจะส่งข้อความมาแจ้งให้คุณทราบ. ซึ่งทางแก้ไขสำหรับกรณีนี้, คุณควรรันคำสั่ง CRTPGM หรือ CRTSRVPGM อีกครั้ง และระบุขนาดของพาร์ติชันให้ใหญ่ขึ้น. โปรดดูในเกี่ยวกับไดเรกทีฟ partition ได้จากหัวข้อ “ไวยากรณ์ของไฟล์ควบคุมของ IPA” ในหน้า 171.
- ข้อจำกัดบางอย่างของ IPA ที่อาจถูกประยุกต์เข้ากับโปรแกรมของคุณ ถ้าโปรแกรมนั้นมีข้อมูล SQL อยู่. แต่ถ้าคอมไพเลอร์ที่คุณใช้อยู่ มีอ็อพชันที่ให้เก็บข้อมูลของ IL, ข้อจำกัดนี้ก็จะไม่มีผลใดๆ กับโปรแกรมของคุณ. ถ้าคอมไพเลอร์ที่คุณใช้อยู่ไม่มีอ็อพชันที่ให้เก็บข้อมูลของ IL, คุณก็ต้องทำตามขั้นตอนที่แสดงไว้ด้านล่างเพื่อทำให้ IPA ในโปรแกรมเก็บข้อมูล SQL เอาไว้. ตัวอย่างเช่น,ในโปรแกรมที่ใช้ภาษา C ที่มีคำสั่ง SQL แทรกอยู่. คุณควรคอมไพล์ซอร์สนี้แบบปกติด้วยคำสั่ง CRTSQLCI; อย่างไรก็ตาม, คำสั่งนี้ไม่มีอ็อพชัน MODCRTOPT(*KEEPILDTA). ปฏิบัติตามขั้นตอนด้านล่างนี้เพื่อสร้าง *MODULE ที่มีทั้งข้อมูล SQL และ IL.

1. คอมไพล์ไฟล์ต้นฉบับ SQL C ด้วยคำสั่ง CRTSQLCI. ระบุ OPTION(*NOGEN) และคอมไพลเลอร์อ็อพชัน TOSRCFILE(QTEMP/QSQLTEMP). ในขั้นตอนนี้เตรียมคอมไพล์คำสั่ง SQL และนำข้อมูล SQL ที่เตรียมไว้ไปใส่ไว้ในที่เก็บข้อมูลของซอร์สไฟล์เดิมที่เกี่ยวข้อง. มันยังนำซอร์สภาษา C เข้าไปเป็นสมาชิกที่มีชื่อเดียวกันในซอร์สไฟล์ชั่วคราว QTEMP/QSQLTEMP.
2. คอมไพล์ซอร์สโค้ด C ใน QTEMP/QSQLTEMP ด้วยอ็อพชัน MODCRTOPT(*KEEPILDTA) ในคำสั่งของคอมไพเลอร์. แอ็คชันนี้จะสร้างอ็อบเจ็กต์ C *MODULE ของ SQL, และกระจายข้อมูลที่เตรียมไว้จากที่เก็บข้อมูลของซอร์สไฟล์เดิมที่เกี่ยวข้องไปยังอ็อบเจ็กต์โมดูล. อ็อบเจ็กต์ *MODULE นี้ยังมีข้อมูล IL อีกด้วย. ณ. จุดนี้, คุณสามารถระบุอ็อบเจ็กต์ *MODULE ในคำสั่ง CRTPGM หรือ CRTSRVPGM ด้วยพารามิเตอร์ IPA(*YES).

- IPA ไม่สามารถ optimize โมดูลที่คุณคอมไพล์ที่ระดับการ optimization ระดับ 10 (*NONE). IPA ต้องการข้อมูลภายในของข้อมูล IL ที่พร้อมใช้งานเฉพาะที่optimization ในเลเวลสูงเท่านั้น.
- IPA ไม่สามารถ optimize โมดูลที่ไม่มีข้อมูล IL. ด้วยเหตุนี้, IPA จึงสามารถ optimize เฉพาะโมดูลที่คุณสร้างด้วยคอมไพลเลอร์ซึ่งให้ อ็อพชัน MODCRTOPT(*KEEPILDTA). ในขณะนี้รวมถึงคอมไพเลอร์ของภาษา C และ C++.
- สำหรับโปรแกรม, โมดูลที่มี entry point ของโปรแกรม, ซึ่งปกติแล้วจะเป็นฟังก์ชันหลัก, จะต้องมีแอ็ตทริบิวต์ที่ถูกต้อง ดังที่แสดงไว้ด้านบน, มิฉะนั้น IPA จะล้มเหลว. สำหรับเซอร์วิสโปรแกรม, อย่างน้อย โมดูลหนึ่งที่มีฟังก์ชันซึ่งเอ็กซปอร์ตต้องมีแอ็ตทริบิวต์ที่ถูกต้อง ดังที่แสดงไว้ด้านบน, มิฉะนั้น IPA จะล้มเหลว. โมดูลอื่นภายในโปรแกรมหรือเซอร์วิสโปรแกรม ควรจะมีแอ็ตทริบิวต์ที่ถูกต้องด้วยเช่นกัน, แต่ ไม่จำเป็น. IPA จะรับโมดูลใดๆ ที่ไม่มีแอ็ตทริบิวต์ที่ถูกต้อง, แต่โมดูลเหล่านั้นจะไม่ได้รับการ optimize.

พาร์ติชันที่สร้างโดย IPA

โปรแกรมหรือเซอร์วิสโปรแกรมสุดท้ายที่สร้างโดย IPA จะประกอบไปด้วยพาร์ติชันหลายๆ อัน. IPA สร้าง *MODULE สำหรับแต่ละพาร์ติชัน. โดยจุดประสงค์ของพาร์ติชันมี 2 ประการคือ:

- พาร์ติชันช่วยเพิ่มการอ้างอิงแบบโลคัลในโปรแกรมโดยเน้นที่สัมพันธ์กับโค้ดที่อยู่ในที่เก็บข้อมูลชุดเดียวกัน.
- พาร์ติชันช่วยลดหน่วยความจำที่ต้องการในระหว่างการสร้างอ็อบเจ็กต์โค้ดสำหรับพาร์ติชันนั้น.

พาร์ติชันแบ่งออกเป็น 3 ชนิดคือ:

- พาร์ติชันเริ่มต้น. ในส่วนนี้จะมีโค้ด และข้อมูลเริ่มต้นเก็บไว้อยู่.
- พาร์ติชันหลัก. ในส่วนนี้จะมีข้อมูลเกี่ยวกับ Primary Entry Point ของโปรแกรม.
- พาร์ติชันรอง และพาร์ติชันอื่นๆ.

IPA สามารถหาจำนวนของพาร์ติชันแต่ละประเภทได้ด้วยวิธีต่อไปนี้:

- ไดเรกทีฟ 'partition' ที่อยู่ในไฟล์ควบคุมที่ระบุไว้โดยพารามิเตอร์ IPACTLFILE. ไดเรกทีฟนี้ยังแสดงขนาดของพาร์ติชันแต่ละอันด้วย.
- การเชื่อมต่อภายในโปรแกรมที่เรียกกราฟ. การเชื่อมต่อจะอ้างถึงวอลุ่มของการเรียกระหว่างฟังก์ชันในโปรแกรม.

- ความขัดแย้งของ Resolution ระหว่างอ็อพชันคอมไพเลอร์ที่ระบุไว้สำหรับยูนิตคอมไพล์ต่างๆ. IPA จะพยายามแก้ไขความขัดแย้งนั้นโดยประยุกต์ใช้อ็อพชันร่วมกันระหว่างยูนิตคอมไพล์ทั้งหมด. ถ้าไม่สามารถทำได้, มันก็จะบังคับให้ยูนิตคอมไพล์ที่มีผลกระทบต่ออ็อพชันเดิมแยกออกมาเป็นอีกพาร์ติชันหนึ่ง.

  ตัวอย่างหนึ่งสำหรับเรื่องนี้คือ *Licensed Internal Code Options* (LICOPTs). ถ้ายูนิตคอมไพล์ 2 ตัวมี LICOPT ที่ขัดแย้งกัน IPA จะไม่สามารถรวมฟังก์ชันจากยูนิตคอมไพล์เหล่านี้ให้อยู่ในพาร์ติชันเดียวกันได้. โปรดดู “Partition Map” ในหน้า 200 สำหรับตัวอย่างของส่วน Partition Map listing . ซึ่ง IPA จะสร้างพาร์ติชันขึ้นในไลบรารีชั่วคราว, และรวม *MODULE ที่สัมพันธ์กันเข้าด้วยกันเพื่อสร้างโปรแกรมหรือเซอร์วิสโปรแกรมสุดท้าย. IPA จะสร้าง ชื่อพาร์ติชัน *MODULE โดยใช้อักษรนำหน้าแบบสุ่ม (ตัวอย่างเช่น, QDO068xxxx ซึ่ง xxxx มีตั้งแต่ 0000 ถึง 9999).

  เนื่องจากฟิลด์บางตัวใน DSPPGM หรือ DSPSRVPGM อาจไม่ได้คาดไว้. ดังนั้น 'Program entry procedure module' จะแสดงชื่อของพาร์ติชัน *MODULE และไม่ใช่ชื่อเดิมของ *MODULE. ฟิลด์ 'Library' ของโมดูลก็จะแสดงชื่อของไลบรารีชั่วคราวเอาไว้แทนที่จะเป็นชื่อเดิมของไลบรารี. นอกจากนี้, ชื่อของโมดูลที่รวมเข้ากับโปรแกรมหรือเซอร์วิสโปรแกรมจะสร้างชื่อของพาร์ติชันด้วย. สำหรับโปรแกรมหรือเซอร์วิสโปรแกรมใดที่ถูกทำการ Optimize โดย IPA, ส่วนฟลิด์ 'Program attribute' จะถูกแสดงโดย DSPPGM หรือ DSPSRVPGM ที่เป็น IPA, เช่นเดียวกับแอ็ตทริบิวต์ของฟิลด์ของโมดูลทั้งหมดที่รวมกับ โปรแกรมหรือเซอร์วิสโปรแกรม.

หมายเหตุ: เมื่อ IPA กำลังทำพาร์ติชัน, IPA อาจใส่อักษรนำหน้า ฟังก์ชันหรือชื่อข้อมูลด้วย @nnn@ หรือ XXXX@nnn@, ซึ่ง XXXX เป็นชื่อพาร์ติชัน, และ nnn เป็น หมายเลขไฟล์ต้นฉบับ. เพื่อให้แน่ใจได้ว่าชื่อฟังก์ชัน และข้อมูลแบบ Static เหล่านั้นไม่มีชื่อใดซ้ำกัน.

Licensed Internal Code Options

Licensed Internal Code Options (LICOPTs) เป็นอ็อพชันของคอมไพเลอร์ที่ถูกผ่านไปยัง Licensed Internal Code เพื่อส่งผลต่อวิธีการที่โค้ดถูกสร้างขึ้นหรือถูกทำให้เป็นแพคเกจ. อ็อพชันของคอมไพเลอร์ที่ถูกผ่านไปนี้จะมีผลกับโค้ดที่ถูกสร้างขึ้นสำหรับโมดูล ,โปแกรมอ็อบเจ็กต์ ILE ,หรือโปรแกรมที่คอมไพล์ด้วยจาวา. คุณสามารถใช้อ็อพชันบางอย่างในการปรับแต่งการ optimization ของโค้ดของคุณ. อ็อพชันบางอย่างช่วยในการดีบักโปรแกรม ในส่วนนี้เราจะกล่าวเฉพาะอ็อพชันของ Licensed Internal Code ที่เกี่ยวข้องกับ ILE เท่านั้น.สำหรับข้อมูลที่เกี่ยวข้องกับ Java, โปรดดูข้อมูลความช่วยเหลือแบบออนไลน์สำหรับพารามิเตอร์ LICOPT ของคำสั่ง CRTJVAPGM. หรืออาจจะดูได้จาก IBM Developer Kit for Javaใน Information Center.

อ็อพชันที่ใช้กำหนดในปัจจุบัน

Licensed Internal Code Options ที่ปัจจุบันถูกกำหนดไว้สำหรับ ILE คือ:

[No]AlwaysTryToFoliate

กำหนดให้ตัวแปร Optimizing เมื่อคอมไพล์ที่ระดับ Optimization เท่ากับ 40 เพื่อเพิ่มประสิทธิภาพในการ Optimize ที่เรียกว่า **call foliation**,ซึ่งเป็นการลดจำนวนของ Stack Frames ที่ต้องดูแลใน Runtime Call Stack. ข้อดีของวิธีนี้คือ (ในบางกรณี), ต้องการสแตกเฟรมน้อย ซึ่งช่วยเพิ่มการอ้างอิงในแบบโลคัลและ (สถานการณ์ที่เกิดได้ยาก) ,ลดการโอกาสเกิด

Runtime Stack Overflow . ข้อเสียของวิธีนี้คือ, ในกรณีที่โปรแกรมล้มเหลว คุณอาจไม่พบร่องรอยใดๆ ใน call stack เมื่อคุณทำการดีบัก. อ็อพชันนี้จะถูกปิดเป็นค่าดีฟอลต์.

[No]CallTracingAtHighOpt

ใช้อ็อพชันนี้เพื่อร้องขอการเรียกและคืนค่าที่แทรกไว้ก่อน และหลังโพรซีเดอร์ตามลำดับ ซึ่งต้องการ Stack แม้ใน optimize ระดับ 40. ตามค่าดีฟอล์ต, ไม่มีการเรียกและคืนค่าใดที่จะถูกแทรกลงไปโพรซีเดอร์ที่ optimize ระดับ 40. ข้อดีในการแทรกการเรียกและ return trap คือความสามารถในการใช้ job trace (TRCJOB), แต่มีข้อเสียคืออาจทำให้ประสิทธิภาพของรันไทม์ลดลง. อ็อพชันนี้จะถูกปิดเป็นค่าดีฟอล์ต.

[No]Compact

ใช้อ็อพชันนี้ในการลดขนาดของโค้ดในจุดที่สามารถทำได้, ซึ่งอาจทำให้ความเร็วในการทำงานลดลงได้. โดยจะทำการระงับการ Optimize ในส่วนที่ซ้ำกัน หรือขยายโค้ดแบบ Inline. อ็อพชันนี้จะถูกปิดเป็นค่าดีฟอล์ต.

[No]CreateSuperblocks

ใช้อ็อพชันนี้เพื่อควบคุมการสร้าง superblock , ซึ่งก็คือบล้อกขนาดใหญ่ที่ขยายมาจาก basic block แต่ไม่มีหน่วย control flow ยกเว้นที่ส่วนหัวของมัน. อ็อพชันนี้ยังใช้ควบคุมการทำ optimization บางชนิดที่ทำกับ superblock เช่น trace unrolling และ trace peeling. การสร้างsuperblock และการทำ optimization จะทำให้เกิดความซ้ำซ้อนของโค้ดขึ้นมากมาย; ดังนั้นอ็อพชันนี้จึงถูกใช้เพื่อหยุดการทำ optimization. อ็อพชันนี้จะมีผลต่อเมื่อได้ป้อนข้อมูลการทำโปรไฟล์ไปแล้ว. และอ็อพชันนี้จะถูกเปิดทำงานโดยดีฟอล์ต.

[No]DetectConvertTo8BytePointerError

อ็อพชันนี้ใช้ได้กับโปรแกรมโมเดลหน่วยเก็บเทราสเปซเท่านั้น (ตัวอย่างเช่น, เมื่อมีการใช้ STGMDL(*TERASPACE) ในคำสั่ง CRT ที่เหมาะสม สำหรับภาษาต้นฉบับที่กำลังคอมไพล์). เมื่อใช้อ็อพชันนี้, โค้ดพิเศษ จะถูกสร้างเป็นส่วนหนึ่งของทุกๆ การแปลงจากตัวชี้แบบ 16–ไบต์ เป็น ตัวชี้ 8–ไบต์เพื่อตรวจหาในสถานการณ์รันไทม์ซึ่งตัวชี้ 16–ไบต์ มีแอดเดรส single level store (SLS). แอดเดรส SLS จะไม่ ถูกเก็บเป็นตัวชี้แบบ 8–ไบต์เนื่องจากตัวชี้แบบ 8–ไบต์สามารถชี้ไป ยังเทราสเปซเท่านั้น. เมื่อการตรวจหา SLS ไม่มีผลสำหรับการแปลงจากตัวชี้แบบ 16–ไบต์เป็นตัวชี้แบบ 8–ไบต์, และตัวชี้แบบ 16–ไบต์มี แอดเดรส SLS, การใช้ตัวชี้แบบ 8–ไบต์ในภายหลังอาจอ้างถึง arbritrary location ในเทราสเปซ, หรืออาจทำให้เกิด MCH0601 exception. ในทางตรงกันข้าม, เมื่อการตรวจหามีผล, MCH0609 exception จะถูกส่งสัญญาณ เพื่อบ่งชี้ปัญหาอย่างชัดเจน. การตรวจหานี้มีผลตามดีฟอลต์ตลอด SLS และโปรแกรมหน่วยเก็บแบบ inherit. ในโปรแกรมโมเดลหน่วยเก็บเทราสเปซ, การตรวจหานี้มีผลตามดีฟอลต์เฉพาะเมื่ออยู่ใน program entry procedure (PEP) เท่านั้น, ซึ่งถูกเรียกขึ้นเป็นส่วนหนึ่งของการเรียกโปรแกรม, แต่ไม่อยู่ใน โพรซีเดอร์อื่น

การตรวจหาสำหรับการดำเนินการแปลงตัวชี้บางครั้งอาจ สำเร็จได้โดยใช้ retrieve teraspace address (RETTSADR) Machine Interface instruction เป็นฟังก์ชันในตัวของภาษาเพื่อดำเนินการ แปลงตัวชี้.

อ็อพชันนี้จะถูกปิดเป็นค่าดีฟอล์ต.

[No]EnableInlining

อ็อพชันนี้ใช้ควบคุมการทำ inline ของโพรซีเดอร์ด้วยวิธีการ optimizing translator.การ

ทำinlineของโพรซีเดอร์หมายถึงการแทนที่การเรียกใช้งานโพรซีเดอร์ด้วยสำเนา
แบบinlineของโพรซีเดอร์นั้น .และอ็อพชันนี้จะถูกเปิดทำงานโดยดีฟอล์.

[No]FoldFloat

ใช้ระบุให้ระบบอาจประเมินนิพจน์แบบ Floating-Point ในเวลาคอมไพล์. ซึ่ง LICOPT นี้จะลบล้างอ็อพชันในการสร้างโมดูล 'Fold float constants'. เมื่อ LICOPT ไม่ถูกระบุ อ็อพชันในการสร้างโมดูลจะได้รับการยอมรับ.

LoopUnrolling=<option>

ใช้อ็อพชัน LoopUnrolling เพื่อควบคุมจำนวนครั้งในการทำ loop unrolling ของ optimizing translator. ระบุอ็อพชันเป็น 0 ถ้าไม่ต้องการให้ทำ loop unrolling ,ค่า1 ถ้าต้องการให้ทำ loop unrolling ในปริมาณน้อยโดยเน้นไปที่การลดจำนวนโค้ดที่ซ้ำซ้อน และ 2 ถ้าต้องการให้ทำ loop unrolling อย่างหนักหน่วง. การใช้อ็อพชัน 2 อาจมีผลทำให้โค้ดที่ได้มีขนาดใหญ่มาก. ค่าดีฟอล์ตคือ1.

[No]Maf

อนุญาตให้สร้างคำสั่งการคูณ-บวกเลขแบบ Floating-Point. คำสั่งเหล่านี้จะรวมการคูณและการบวกโดยไม่การปัดเศษ. ซึ่งความเร็วในการทำงานจะเพิ่มขึ้น, แต่ผลการคำนวณอาจได้รับผลกระทบ. ซึ่ง LICOPT นี้จะลบล้างอ็อพชันในการสร้างโมดูล 'Use multiply add'. เมื่อ LICOPT ไม่ถูกระบุ, อ็อพชันในการสร้างโมดูลจะได้รับการยอมรับ.

[No]MinimizeTeraspaceFalseEAOs

ฮาร์ดแวร์ปัจจุบันโหลดและเก็บคำสั่งตรวจพบการข้ามขอบเขต 16 MB หรือที่เรียกอีกอย่างหนึ่งว่า **effective address overflows** (EAOs). การตรวจสอบ EAO จะถูกทำให้เป็นส่วนหนึ่งของการคำนวณแอดเดรส. โค้ดที่ถูกสร้างต้องจัดการกับแอดเดรสแบบ Teraspace และ Single Level, ดังนั้นการใช้ Traspace อาจก่อให้เกิดค่าเท็จใน EAO ได้. เงื่อนไขของ EAO เหล่านี้จะไม่แสดงถึงปัญหาที่เกิดขึ้น, แต่การจัดการกับปัญหาเหล่านี้อาจทำให้โอเวอร์เฮดสูงขึ้น. โดยMinimizeTeraspaceFalseEAOs ของ LICOPT อาจมีชุดคำสั่งแตกต่างกันเมื่อสร้างด้วยฮาร์แวร์ที่แตกต่างกัน. เริ่มแรก คำสั่งในการคำนวณแอดเดรสที่แตกต่างกันจะถูกสร้างขึ้น ซึ่งจะช้ากว่าในกรณีปกติเล็กน้อย แต่สามารถกำจัด EAO ที่เกิดขึ้น. นอกจากนี้, การทำ optimization จะทำให้โค้ดเร็วขึ้นก็จริง แต่สามารถเพิ่มค่าเท็จของ EAOs ได้บ่อยขึ้น. ตัวอย่างว่าเมื่อใดจึงควรใช้ LICOPT นี้ คำตอบก็คือเมื่อมีการคำนวณแอดเดรสส่วนใหญ่ในโมดูลเป็การคำนวณแอดเดรส Teraspace จากแอดเดรสฐานรวมกับอ็อฟเซ็ตที่มีมากกว่า 16 MB. อ็อพชันนี้จะถูกปิดเป็นค่าดีฟอล์ต.

[No]OrderedPtrComp

ใช้อ็อพชันนี้ในการเปรียบเทียบพอยเตอร์เป็นค่าจำนวนเต็มแบบ Unsign และสร้างผลลัพธ์ตามลำดับ (เท่ากับ, น้อยกว่า, หรือมากกว่า). เมื่อคุณใช้อ็อพชันนี้, พอยเตอร์ที่อ้างถึงที่เก็บข้อมูลหลายๆ แห่งจะไม่ถูกเปรียบเทียบตามลำดับ. อ็อพชันนี้จะถูกปิดเป็นค่าดีฟอล์ต.

[No]PredictBranchesInAbsenceOfProfiling

เมื่อโปรไฟล์ข้อมูลไม่ได้ถูกเตรียมไว้, ให้คุณใช้อ็อพชันนี้ในการทำ Static Branch Prediction เพื่อช่วยในการ Optimize โปรแกรม. ถ้าโปรไฟล์ข้อมูลถูกเตรียมไว้ให้แล้ว, โปรไฟล์ข้อมูลจะถูกใช้ในการทำนายคสามน่าจะเป็นของ Branch ต่างๆ โดยไม่สนใจอ็อพชันนี้. อ็อพชันนี้จะถูกปิดเป็นค่าดีฟอล์ต.

[No]PtrDisjoint

อ็อพชันนี้ทำให้เกิดการแบ่ง alias ตามประเภทโดยละเอียดในแบบ aggressive ซึ่ง ทำให้ optimizing translator สามารถขจัดงานซ้ำจำนวนมาก, ซึ่งอาจช่วยเพิ่มประสิทธิภาพรันไทม์ได้. แอ็พพลิเคชันสามารถใช้อ็อพชันนี้ได้อย่างปลอดภัย หากเนื้อหาของตัวชี้ไม่ได้ถูกเข้าใช้งานผ่านประเภท ที่ไม่ใช่ตัวชี้. นิพจน์ต่อไปนี้ใน C แสดงวิธีการเข้าใช้งาน ค่าของตัวชี้แบบไม่ปลอดภัย:

```
void* spp;
... = ((long long*) &spp) [1]; // เข้าใช้ 8 ไบต์หลังของตัวชี้ขนาด 16 ไบต์
```

Default: NoPtrDisjoint

TargetProcessorModel=<option>

อ็อพชัน `targetProcessorModel`ออกคำสั่งให้translator ในการดำเนินการ optimize สำหรับรุ่นของโพรเซสเซอร์ที่ระบุ. โปรแกรมซึ่งสร้าง ด้วยอ็อพชันนี้จะรันในฮาร์ดแวร์รุ่นที่สนับสนุนทุกรุ่น, แต่ตามปกติแล้วจะรันได้เร็วกว่าใน ในโพรเซสเซอร์รุ่นที่ระบุ. ค่าที่ถูกต้องคือ 0 สำหรับ Star processors และ 2 สำหรับ POWER4™ processors. ค่าดีฟอลต์ขึ้นอยู่รีลีสเป้าหมายที่สัมพันธ์กับ กับโปรแกรมอ็อบเจ็กต์. เมื่อเริ่มต้นด้วย V5R2M0, ค่าดีฟอลต์คือ is 2. สำหรับรีลีสก่อนหน้านี้, ค่าดีฟอลต์คือ 0.

สังเกตได้ว่าสำหรับอ็อพชันแต่ละตัวจะมีค่าตัวแปรทั้งด้านบวกและลบ ค่าที่เป็นด้านลบจะเริ่มต้นด้วย 'no'. ค่าที่เครื่องหมายลบนำหน้าหมายความว่าอ็อพชันนั้นจะไม่ถูกใช้งาน. ในอ็อพชันของ Boolean มักจะมีตัวแปร 2 ค่าแบบนี้, เพื่ออนุญาตให้ผู้ใช้เลือกที่จะ 'ปิด' หรือ 'เปิด'. การใช้อ็อพชันเหล่านั้น และยังจำเป็นต่อการที่จะ 'ปิด'การใช้อ็อพชันที่มีค่าดีฟอลต์เป็น 'เปิด' อีกด้วย. แต่ค่าดีฟอลต์ของอ็อพชันในระบบปฏิบัติการแต่ละรีลีสอาจจะเปลี่ยนไปได้.

การประยุกต์ใช้

คุณสามารถระบุอ็อพชัน Licensed Internal Code เป็นอ็อพชันคอมไพเลอร์เมื่อคุณทำการคอมไพล์โปรแกรมของคุณ.

คุณสามารถกำหนด Licensed Internal Code Options ลงในคำสั่ง CHGMOD (Change Module), CHGPGM (Change Program), และ CHGSRVPGM (Change Service Program) บน AS/400 เพื่อป้อนให้กับอ็อบเจ็กต์ที่มีอยู่. ชื่อของพารามิเตอร์คือ LICOPT. ตัวอย่างของการป้อน Licensed Internal Code Options ให้กับโมดูลหนึ่ง คือ:

```
> CHGMOD MODULE(TEST) LICOPT('maf')
```

เมื่อใช้บนคำสั่ง CHGPGM หรือ CHGSRVPGM ระบบจะป้อน Licensed Internal Code ที่กำหนดให้กับทุกโมดูลที่อยู่ในโปรแกรม ILE. ตัวอย่างของการป้อน Licensed Internal Code ให้กับโปรแกรม ILE คือ:

```
> CHGPGM PGM(TEST) LICOPT('nomaf')
```

ตัวอย่างของการประยุกต์ใช้อ็อพชัน Licensed Internal Code กับเซอร์วิสโปรแกรมคือ:

```
> CHGSRVPGM SRVPGM(TEST) LICOPT('maf')
```

ข้อจำกัด

ข้อจำกัดหลายประการของชนิดของโปรแกรมและโมดูลที่คุณจะสามารถใช้อ็อพชัน Licensed Internal Code.

- คุณไม่สามารถประยุกต์ใช้อ็อพชัน Licensed Internal Code กับโปรแกรม OPM ได้.
- โมดูลหรือโปรแกรม ILE หรืออ็อบเจ็กต์เซอร์วิสโปรแกรมต้องถูกสรางขึ้นมา.เพื่อใช้กับ.รีลีส V4R5M0 หรือใหม่กว่า.
- คุณไม่สามารถใช้อ็อพชัน Licensed Internal Code กับโมดูลที่มาจากรีลีสก่อนรีลีส V4R5 ที่รวมเข้ากับโปรแกรมหรือเซอร์วิสโปรแกรมในรีลีส V4R5 หรือหลังจากนั้น. แต่จะไม่มีผลกระทบต่อโมดูลอื่นๆ ในโปรแกรมที่สามารถใช้ LICOPTs ได้.

ไวยากรณ์

ในคำสั่ง CHGMOD, CHGPGM, และ CHGSRVPGM ขนาดของตัวอักษรที่แสดงค่าของพารามิเตอร์ LICOPT นั้นไม่มีความสำคัญ. ดังตัวอย่างข้างล่างนี้, การใช้คำสั่งทั้ง 2 แบบจะให้ผลลัพธ์ที่เหมือนกัน:

```
> CHGMOD MODULE(TEST) LICOPT('nomaf')
> CHGMOD MODULE(TEST) LICOPT('NoMaf')
```

เมื่อมีการกำหนดอ็อพชัน Licensed Internal Code หลายตัว คุณต้องแยกอ็อพชันด้วยการใช้เครื่องหมายจุลภาค (,). และระบบจะไม่สนใจช่องว่างที่อยู่ก่อนหน้าหรือตามหลังอ็อพชัน. ดังตัวอย่างต่อไปนี้:

```
> CHGMOD MODULE(TEST) LICOPT('Maf,NoFoldFloat')
> CHGMOD MODULE(TEST) LICOPT('Maf,   NoFoldFloat')
> CHGMOD MODULE(TEST) LICOPT('   Maf   ,   NoFoldFloat   ')
```

สำหรับอ็อพชันของ Boolean, ระบบจะไม่อนุญาตให้มีการกำหนดตัวแปรที่ตรงข้ามกัน 2 ตัวในเวลาเดียวกัน. จากตัวอย่าง, ระบบจะไม่อนุญาตให้ใช้คำสั่งนี้:

```
> CHGMOD MODULE(TEST) LICOPT('Maf,NoMaf')   <- NOT ALLOWED!
```

อย่างไรก็ตาม, คุณสามารถระบุอ็อพชันเดิมได้มากกว่าหนึ่งครั้ง. ตัวอย่างเช่น, คำสั่งนี้ถูกต้องตามกฎ:

```
> CHGMOD MODULE(TEST) LICOPT('Maf, NoFoldFloat, Maf')
```

ความเข้ากันได้ของรีลิส

ระบบจะไม่ยอมให้ผู้ใช้ย้ายโมดูล, โปรแกรม ILE, หรือเซอร์วิสโปรแกรม ที่มี Licensed Internal Code ซึ่งมีรีลีสก่อน V4R5M0. ความจริงแล้ว, ระบบจะป้องกันผู้ใช้จากการกำหนดเป้าหมายให้เป็นรีลีสหน้านี้ เมื่อพยายามที่จะ save อ็อบเจ็กต์ลงในมีเดียหรือไฟล์.

OS/400 อาจกำหนดอ็อพชัน Licensed Internal Code ได้ในรีลีสต่อไป (หรือในรีลีสที่ให้มาโดยผ่าน PTF). คุณสามารถใช้อ็อพชันใหม่บนระบบที่มีรีลีสแรกที่สนับสนุนมันหรือรีลีสหลังจากนั้น. คุณสามารถย้ายโมดูล, โปรแกรม ILE, หรือเซอร์วิสโปรแกรมที่มีอ็อพชันใหม่ไปยังรีลีสที่ไม่สนับสนุนอ็อพชันนั้น. อย่างไรก็ตาม, รีลีสนั้นจะต้องเป็น V4R5M0 หรือหลังจากนั้น. ระบบไม่สน

ใจและไม่ป้อนอ็อพชันที่ Licensed Internal Code ไม่สนับสนุนให้กับอ็อบเจ็กต์ที่ถูกแปลงใหม่ ถ้าพารามิเตอร์ LICOPT ของคำสั่งไม่ได้กำหนดอ็อพชัน. การเปลี่ยนแปลงประเภทนี้สามารถทำได้เมื่อระบบมีการสร้างอ็อบเจ็กต์ด้วยการระบุ LICOPT(*SAME) ในคำสั่ง OD, CHGPGM, หรือ GSRVPGM และการเปลี่ยนแปลงนี้จะเกิดเมื่อระบบทำการแปลงอ็อบเจกต์แบบอัตโนมัติ. แต่ไม่ได้ป้องกันการสร้างใหม่แต่อย่างไร. ในทางตรงกันข้าม, ความพยายามใดๆ ที่จะกำหนดอ็อพชันที่ไม่ถูกสนับสนุนในพารามิเตอร์ LICOPT ในคำสั่ง CHGMOD, CHGPGM,หรือ CHGSRVPGM จะล้มเหลว.

การแสดงอ็อพชัน Licensed Internal Code ของโมดูลและโปรแกรม ILE

คำสั่ง DSPMOD, DSPPGM, และ DSPSRVPGM จะแสดงถึงอ็อพชัน Licensed Internal Code ที่ถูกใช้งาน. DSPMOD แสดงอ็อพชันเหล่านั้นในส่วนของ Module Informational section. ตัวอย่างเช่น:

```
    Licensed Internal Code options  . . . . . . . . . . . :   maf
```

คำสั่ง DSPPGM และ DSPSRVPGM แสดงอ็อพชัน Licensed Internal Code ที่ถูกใช้กับโมดูลแต่ละโมดูลภายในโปรแกรมในส่วนของ Module Attribute ของแต่ละโมดูล.

ในการกำหนด Licensed Internal Code ที่เหมือนกันมากกว่า 1 ครั้ง การเกิดขึ้นทุกครั้งของอ็อพชันนั้นจะนำหน้าด้วยเครื่องหมายบวก (+) ยกเว้นครั้งสุดท้าย. ดังตัวอย่างของคำสั่งให้ป้อนอ็อพชัน Licensed Internal Code ให้กับโมดูลอ็อบเจ็กต์ ดังนี้:

```
> CHGMOD MODULE(TEST) LICOPT('maf, maf,   Maf')
```

จากนั้น DSPMOD จะแสดงผลดังนี้:

```
    Licensed Internal Code options  . . . . . . . . . . . :   +maf,+maf,Maf
```

เครื่องหมาย '+' หมายความว่าผู้ใช้กำหนดการเกิดขึ้นที่ซ้ำกันของอ็อพชันที่เหมือนกัน.

ถ้ามีเครื่องหมาย '*' นำหน้าอ็อพชัน Licensed Internal Code Options แสดงว่ามันจะไม่ถูกป้อนให้กับโมดูลหรือโปรแกรม ILE อีกต่อไป. นั่นก็เพราะระบบสุดท้ายที่ทำการสร้างอ็อบเจ็กต์ใหม่ไม่สนับสนุนอ็อพชันเหล่านั้น. สำหรับรายละเอียดเพิ่มเติม ให้ดูในหัวข้อ “ความเข้ากันได้ของรีลิส” ในหน้า 180. ตัวอย่างเช่น, สมมุติให้อ็อพชันใหม่ถูกป้อนบนระบบที่เป็นรีลีส N+1 โดยการใช้คำสั่งดังนี้:

```
  > CHGMOD MODULE(TEST) LICOPT('NewOption')
```

โมดูลนั้นจะถูกถอยหลังกลับเป็นรีลีสเก่าที่ไม่สนับสนุนอ็อพชันนั้น จากนั้นก็จะมีการสร้างโมดูลอ็อบเจ็กต์ใหม่โดยใช้คำสั่ง:

```
  > CHGMOD MODULE(TEST) FRCCRT(*YES) LICOPT(*SAME)
```

อ็อพชัน Licensed Internal Code ที่แสดงใน DSPMOD เป็นดังนี้:

```
    Licensed Internal Code options  . . . . . . . . . . . :   *NewOption
```

เครื่องหมาย '*' แสดงว่าอ็อพชันนั้นไม่ได้ใช้งานในโมดูลนี้อีกต่อไป.

บทที่ 14. การ Synchronize ของหน่วยเก็บข้อมูลแบบแบ่งใช้

Shared storage ให้วิธีการที่มีประสิทธิผลสำหรับการสื่อสารระหว่าง thread ที่กำลังรันอยู่ตั้งแต่ 2 ตัวขึ้นไป. บทนี้จะเป็นการอธิบายถึงประเด็นต่างๆ ที่เกี่ยวข้องกับแบ่งใช้หน่วยความจำ. โดยจะเน้นในเรื่องปัญหาของการประสานเวลาของข้อมูลที่สามารถเกิดขึ้นในขณะเข้าถึงหน่วยความจำที่แบ่งใช้และวิธีการที่จะแก้ไขปัญหานั้น.

แม้ว่าปัญหาจะไม่ได้เกิดขึ้นกับ ILE เพียงอย่างเดียว แต่ปัญหาในการเขียนโปรแกรมที่เกี่ยวกับการแบ่งใช้หน่วยความจำ ซึ่งมักจะเกิดขึ้นในภาษา ILE มากกว่าในภาษา MI เดิม. เนื่องมาจากการสนับสนุน Application Programming Interface แบบมัลติโปรแกรมมิงใน ILE ที่มีมากกว่าภาษาอื่น.

หน่วยความจำแบบแบ่งใช้

คำว่า *หน่วยความจำแบบแบ่งใช้* ในที่นี้หมายถึง พื้นที่เก็บข้อมูลใดๆ ที่ถูกเข้าถึงจาก Thread ตั้งแต่ 2 ตัวขึ้นไป. ความหมายนี้รวมไปถึงหน่วยเก็บข้อมูลใดๆ ที่สามารถเข้าถึงได้โดยตรงไปยังแต่ละไบต์ของหน่วยเก็บข้อมูลนั้น และสามารถรวมเป็นลำดับชั้นของหน่วยเก็บข้อมูลได้ดังนี้:

- MI space objects
- Primary associated spaces of other MI objects
- POSIX shared memory segments
- Implicit process spaces: ได้แก่ หน่วยเก็บข้อมูลแบบอัตโนมัติ, หน่วยเก็บข้อมูลแบบสแตติก และ Heap แบบ activation-base
- Teraspace

ระบบกำหนดพื้นที่เหล่านี้โดยไม่คำนึงถึงช่วงอายุของพื้นที่เหล่านั้น แต่ถือว่าเป็นหน่วยความจำแบบแบ่งใช้เมื่อถูกเข้าถึงโดย Thread หลายๆ ตัวที่มาจากโพรเซสที่ดำเนินอยู่ในปัจจุบัน.

ปัญหาของหน่วยความจำแบบแบ่งใช้

เมื่อสร้างแอ็พพลิเคชันที่ใช้ประโยชน์จากหน่วยความจำแบบแบ่งใช้ นักพัฒนาต้องการที่จะเลี่ยงปัญหา 2 ชนิดที่สามารถทำให้เกิดผลลัพธ์ที่ไม่สามารถคาดเดาค่าของข้อมูลได้ นั่นคือ: *Race Condition* และ *Storage Access Ordering Problem*.

- Race Cnditions จะเกิดขึ้นเมื่อเกิดผลลัพธ์ของโปรแกรมที่แตกต่างกัน เนื่องจากเวลาที่สัมพันธ์กันของ Thread ที่ทำงานร่วมกันตั้งแต่ 2 ตัวขึ้นไป.

  คุณสามารถเลี่ยง Race Condition ได้โดยการซิงโครไนสกระบวนการของ Thread ที่แย่งกัน เพื่อให้ Thread โต้ตอบกันในลักษณะของการปฏิบัติที่ดีและสามารถคาดเดาได้. แม้ว่าจุดสำคัญของบทนี้จะอยู่ที่ Storage Synchronization แต่เทคนิคสำหรับการทำซิงโครไนสThread และการทำซิงโครไนสหน่วยเก็บข้อมูลจะซ้ำซ้อนกันอยู่มาก. ดังนั้นตัวอย่างของปัญหาที่จะกล่าวในบทนี้จะพูดถึง race condition อย่างคร่าวๆ.

- ส่วนปัญหาเรื่อง Storage Access Ordering หรือเป็นที่รู้จักในชื่ออื่นว่าปัญหาการซิงโครไนส หน่วยเก็บข้อมูล หรือหน่วยความจำไม่สอดคล้องกัน. การที่ Thread ที่ทำงานร่วมกันตั้งแต่ 2 ตัวขึ้นไปโดยอาศัยการเรียงลำดับของการอัพเดตหน่วยเก็บข้อมูลที่ใช้ร่วมกันไม่มีการเข้าถึงพื้นที่เก็บข้อมูลแบบตามลำดับ เป็นสาเหตุของปัญหาเหล่านี้. ตัวอย่างเช่น ถ้ามีเหตุการณ์ดังนี้: Thread 1 ตัวเก็บค่าของตัวแปรที่มีการแบ่งใช้ 2 ตัวแปร และ Thread อีกตัวหนึ่งขึ้นอยู่กับการปฏิบัติตามการอัพเดตในลำดับที่แน่นอน.

  คุณสามารถหลีกเลี่ยงปัญหาเรื่อง Shared Storage Access Ordering โดยการทำให้แน่ใจว่าระบบได้ทำ Storage Synchronization โดยการใช้ Thread ที่อ่านและเขียนลงในหน่วยเก็บข้อมูลแบบแบ่งใช้. คุณสามารถอ่านเกี่ยวกับวิธีแก้ปัญหาเหล่านี้ได้จากหัวข้อด้านล่าง.

ลำดับในการเข้าถึงหน่วยความจำแบบแบ่งใช้

เมื่อ Thread แบ่งใช้หน่วยเก็บข้อมูล ไม่ได้รับประกันว่าการเข้าถึงหน่วยความจำแบบแบ่งใช้ ที่ถูกกระทำโดย Thread ตัวหนึ่งจะถูกปฏิบัติในลำดับหนึ่งโดยเฉพาะโดย Thread ตัวอื่น . คุณสามารถป้องกันเหตุการณ์นี้ได้โดยการมีรูปแบบที่ชัดเจนของ Storage Synchronization ที่กระทำโดย Thread ที่อ่านหรือเขียนในหน่วยความจำแบบแบ่งใช้.

มีเพียงเหตุการณ์ต่อไปนี้เท่านั้นที่ต้องการ Storage Synchronization: Thread ตั้งแต่ 2 ตัวขึ้นไปพยายามที่จะเข้าถึง shared storage พร้อมๆ กัน และซีแมนทิกส์ของลอจิกของ Thread ต้องการการเรียงลำดับในการเข้าถึงหน่วยความจำแบบแบ่งใช้. เมื่อลำดับของการอัพเดตของหน่วยความจำแบบแบ่งใช้ไม่มีความสำคัญ Storage Synchronization ก็ไม่มีความจำเป็นเช่นกัน. Thread ที่ให้มามักจะปฏิบัติตามการอัพเดตหน่วยเก็บข้อมูลของตัวเอง (ในการแบ่งหรือไม่แบ่งใช้หน่วยเก็บข้อมูล) ตามลำดับ. การเข้าถึงของ thread ทั้งหมดที่*ซ้ำซ้อนกัน*ในตำแหน่งของหน่วยความจำแบบแบ่งใช้จะปฏิบัติตามการเข้าถึงเหล่านั้นในลำดับที่เหมือนกัน.

พิจารณาตัวอย่างต่อไปนี้ที่แสดงถึงวิธีที่ Race Conditions และปัญหาเรื่อง Storage Access Ordering สามารถนำไปสู่ผลลัพธ์ที่สามารถคาดเดาได้.

```
                volatile int X = 0;
                volatile int Y = 0;
   Thread A                          Thread B
-------------                     -------------
    Y = 1;                            print(X);
    X = 1;                            print(Y);
```

ตารางข้างล่างนี้เป็นการสรุปผลลัพธ์ที่เป็นไปได้ที่จะถูกพิมพ์โดย B.

X	Y	ชนิดของปัญหา	คำอธิบาย
0	0	Race Condition	Thread B อ่านตัวแปรก่อนการเปลี่ยนแปลงของ Thread A.
0	1	Race Condition	Thread B ทำการอัพเดต ค่า Y แต่พิมพ์ค่า X ก่อนที่จะมีการอัพเดตของ Thread A.
1	1	Race Condition	Thread B read both variables after the updates of Thread A.

X	Y	ชนิดของปัญหา	คำอธิบาย
1	0	Storage Access Ordering	Thread B อัพเดตค่า X แต่จะต้องเห็นว่า Thread A อัพเดตค่า Y เนื่องจากไม่มีการทำ data synchronization ทำให้การเข้าถึงหน่วยเก็บข้อมูลแบบไม่เป็นไปตามลำดับเกิดขึ้นได้.

ตัวอย่างของปัญหาแบบที่ 1: ตัวเขียน 1 ตัว ตัวอ่านหลายตัว

โดยส่วนใหญ่แนวโน้มสำหรับการเข้าถึงหน่วยความจำแบบแบ่งใช้โดยไม่เป็นไปตามลำดับจะไม่มีผลกระทบต่อความถูกต้องของลอจิกของโปรแกรมแบบ multi-threaded. อย่างไรก็ตามในบางกรณีลำดับของ thread มองว่าการอัพเดตหน่วยเก็บข้อมูลของ thread ตัวอื่นมีความสำคัญต่อความถูกต้องของโปรแกรม.

โปรแกรม ลองพิจารณาถึงเหตุการณ์ที่ต้องการรูปแบบของการ synchronization ข้อมูลอย่างชัดเจน. เมื่อสถานะของตำแหน่งของหน่วยความจำแบบแบ่งใช้ตำแหน่งหนึ่งถูกใช้ (โดยกฎในลอจิกของโปรแกรม) ในการควบคุมการเข้าถึงตำแหน่งที่ 2 ของหน่วยความจำแบบแบ่งใช้ (โดยไม่ซ้ำซ้อนกัน).) ตัวอย่างเช่น สมมุติให้ thread ตัวหนึ่งให้ค่าเริ่มต้นแก่ข้อมูลที่มีการแบ่งใช้ (DATA). นอกจากนี้ thread นั้นยังตั้งค่า shared flag (FLAG) เพื่อแจ้งไปยัง thread ทั้งหมดว่าข้อมูลที่มีการแบ่งใช้นี้ถูกให้ค่าเริ่มต้นแล้ว.

```
Initializing Thread                 All Other Threads
-------------------                 -------------------------
     DATA = 10                      loop until FLAG has value 1
     FLAG = 1                       use DATA
```

ในกรณีนี้ thread ที่มีการแบ่งใช้จะต้องดำเนินการให้มีการเรียงลำดับในการเข้าถึงหน่วยความจำแบบแบ่งใช้. มิฉะนั้นแล้ว thread อื่นๆ จะมองว่าหน่วยความจำแบบแบ่งใช้ของ thread ที่ให้ค่าเริ่มต้นนั้นอัพเดตโดยไม่มีลำดับ. ซึ่งอาจจะเป็นการยอมให้ thread อื่นๆ อ่านค่าที่ยังไม่ได้ถูกให้ค่าเริ่มต้นจาก DATA.

ข้อแก้ไขของตัวอย่างที่ 1

วิธีการที่ดีในการแก้ปัญหาในตัวอย่างข้างต้นคือ หลีกเลี่ยงการสัมพันธ์กันระหว่างค่าของ data และ flag. คุณสามารถทำได้โดยการใช้แบบแผนในการทำ thread synchronization ที่เข้มแข็งกว่านี้. แม้ว่าคุณอาจจะใช้ Thread Synchronization ได้หลายเทคนิค เทคนิคหนึ่งที่เหมาะสมกับปัญหานี้คือ semaphore. (การสนับสนุนสำหรับ semaphore มีตั้งแต่ AS/400 เวอร์ชัน 3, รีลีส 2.)

สำหรับลอจิกที่เหมาะสมต่อไปนี้ คุณต้องสมมุติให้:

- โปรแกรมสร้าง semaphore ก่อนการเกิด thread ที่ทำงานร่วมกัน.
- โปรแกรม initialize semaphore ให้ count เท่ากับ 1.

```
Initializing Thread                 All Other Threads
-------------------                 -------------------------
    DATA = 10                   Wait for semaphore count to reach 0
    Decrement semaphore         use DATA
```

Storage Synchronizing Actions

เมื่อมีความต้องการการเรียงลำดับในการเข้าถึงหน่วยความจำแบบแบ่งใช้ คุณต้องพบเหตุการณ์นี้: Thread ทั้งหมดที่ต้องการการดำเนินการให้มีการเรียงลำดับ จะต้องทำการซิงโครไนสการเข้าถึงหน่วยความจำแบบแบ่งใช้. ซึ่งเรียกการกระทำนี้ว่า *Storage Synchronizing Actions*.

Synchronizing action ถูกกระทำโดย thread ตัวหนึ่งเพื่อให้แน่ใจว่าการเข้าถึงหน่วยความจำแบบแบ่งใช้ นั้นเกิดขึ้นก่อนจนกว่า synchronizing action ใน logical flow ของ thread จะเสร็จสิ้นลงก่อนการเข้าถึงเหล่านั้นเกิดขึ้นใน logical flow ของโค้ดหลัง synchronizing action. นี่คือมุมมองของ thread อื่น ในขณะที่เกิด synchronizing action ของ thread เหล่านั้น. ในทางตรงกันข้าม ถ้า thread มีการเขียน 2 ครั้งไปยังตำแหน่งที่มีการแบ่งใช้ 2 ตำแหน่ง และ synchronizing action แยกการเขียนเหล่านั้นออกจากกัน ระบบจะกระทำดังนี้: การเขียนครั้งแรกถูกรับประกันให้ใช้ได้กับ thread อื่นๆ ในขณะที่เกิดหรือก่อนที่จะเกิด synchronizing action ครั้งต่อไป และไม่มีการให้ใช้การเขียนครั้งที่สอง.

เมื่อการอ่าน 2 ครั้งจากตำแหน่งที่มีการแบ่งใช้ 2 ตำแหน่ง ถูกแยกจากกันด้วย storage synchronizing action การอ่านครั้งที่ 2 จะอ่านค่าที่ไม่เป็นปัจจุบันกว่าการอ่านครั้งแรก. เหตุการณ์นี้เป็นจริงเมื่อ thread ตัวอื่นดำเนินการให้มีการเรียงลำดับเมื่อมีการเขียนไปยังหน่วยความจำแบบแบ่งใช้.

Thread synchronization action ต่อไปนี้ก่อให้เกิดการประสานเวลาของ storage action:

Mechanism	Synchronizing Action	First Available in VRM
Object Locks	ล็อค, ไม่ล็อค	All
Space Location Locks	ล็อค, ไม่ล็อค	All
Mutex	ล็อค, ไม่ล็อค	V3R1M0
Semaphores	โพสต์, รอ	V3R2M0
Pthread Conditions	รอ, Signal, Broadcast	V4R2M0
Data Queues	ใส่ในคิว, เอาออกจากคิว	All
Message Queues	ใส่ในคิว, เอาออกจากคิว	V3R2M0
Compare-and-Swap	เก็บข้อมูลไปที่เป้าหมายได้สำเร็จ	V3R1M0
Check Lock Value (CHKLKVAL)	บันทึกข้อมูลสู่เป้าหมายได้สำเร็จ	V5R3M0
Clear Lock Value (CLRLKVAL)	Always	V5R3M0

นอกจากนี้ ยังมีการกระทำที่ไม่ synchronization กับ thread แต่ก่อให้เกิด synchronization ของ storage action:

Mechanism	Synchronizing Action	First Available in VRM
SYNCSTG MI Instruction	Always	V4R5M0

ข้อควรจำ: เพื่อที่จะดำเนินการให้เกิดการเรียงลำดับในการเข้าถึง shared storage ระหว่าง thread ตั้งแต่ 2 ตัวขึ้นไปโดยสมบูรณ์ จำเป็นที่ thread ทุกตัวจะต้องขึ้นกับลำดับของการเข้าถึงที่ใช้ synchronizing action ที่เหมาะสม. ทั้งสำหรับตัวอ่านและตัวเขียนของข้อมูลที่มีการแบ่งใช้. ข้อตกลงระหว่างตัวอ่านและตัวเขียนทำให้แน่ใจว่าลำดับของการเข้าถึงจะยังคงไม่เปลี่ยนแปลงจากการ optimization ใดๆ ที่เกิดขึ้นโดยเครื่องที่มีสิทธิเหนือกว่า.

ตัวอย่างของปัญหาแบบที่ 2 : ตัวเขียนหรือตัวอ่าน 2 ตัว

ปัญหาธรรมดาแบบอื่นที่ต้องการ synchronization เพิ่มเติม คือปัญหาที่มี thread ตั้งแต่ 2 ตัวขึ้นไปที่พยายามดำเนินการให้มี informal locking protocol. ดังตัวอย่างข้างล่างนี้ ในตัวอย่างนี้มี thread 2 ตัวเปลี่ยนแปลงข้อมูลใน shared storage thread. ทั้งสองพยายามที่จะอ่านและเขียนข้อมูลที่มีการแบ่งใช้ 2 ข้อมูลโดยทำงานซ้ำกัน การใช้ flag ที่มีการแบ่งใช้ในการทำงานของ thread เพื่อที่จะเข้าถึงแบบตามลำดับ.

```
                Thread A
    ------------------------------------
 /* Do some work on the shared data */
  for (int i=0; i<10; ++i) {
    /* Wait until the locked flag is clear */
    while (locked == 1) {
     sleep(1);
    }

    locked = 1;  /* Set the lock */

   /* Update the shared data */
   data1 += 5;
   data2 += 10;

    locked = 0;  /* Clear the lock */
  }
```

```
                Thread B
------------------------------------
   /* Do some work on the shared data */
for (int i=0; i<10; ++i) {
  /* Wait until the locked flag is clear */
  while (locked == 1) {
       sleep(1);
  }

  locked = 1;     /* Set the lock */

     /* Update the shared data */
     data1 += 4;
     data2 += 6;

  locked = 0;     /* Clear the lock */
}
```

ตัวอย่างนี้แสดงถึง shared memory pitfall ทั้งคู่

Race Conditions

Locking protocol ที่ใช้ในที่นี้ไม่สามารถเลี่ยงปัญหา data race condition ได้. งานทั้ง 2 งานจะเห็นว่า locked นั้นว่างอยู่และทั้งคู่จะเข้าสู่ลอจิกในการอัพเดตข้อมูล. ณ สภาวะนั้น, ไม่สามารถรับประกันได้ว่าค่าของข้อมูลจะถูกอ่าน, ถูกเพิ่มขึ้น, และถูกเขียน — เป็นโอกาสให้เกิดผลลัพธ์ที่เป็นไปได้หลายกรณี.

ความกังวลในเรื่อง Storage Access Ordering

เมื่อหยุดสนใจในเรื่อง race condition ที่กล่าวไปแล้วชั่วคราว. จะสังเกตว่าลอจิกที่ใช้โดยงานทั้งสองในการอัพเดต lock และข้อมูลที่มีการแบ่งใช้มีสมมุติฐานเกี่ยวกับการเรียงลำดับของฟิลด์ที่อัพเดต. โดยเฉพาะมีสมมติฐานเกี่ยวกับส่วนหนึ่งของ thread แต่ละตัวที่ thread ตัวอื่นจะปฏิบัติตาม นั่นคือ locked flag ถูกตั้งค่าเป็น 1 ก่อนการปฏิบัติตามการเปลี่ยนแปลงของข้อมูล. นอกจากนี้ยังถือได้ว่า thread แต่ละตัวจะปฏิบัติตามการเปลี่ยนแปลงของข้อมูลก่อนหน้าที่จะทำตามค่าของ locked flag ที่เป็น 0. ดังที่กล่าวไว้ตั้งแต่แรกแล้วว่าสมมุติฐานนี้ไม่สามารถใช้งานได้.

ข้อแก้ไขของตัวอย่างที่ 2

เพื่อหลีกเลี่ยง race condition และทำให้เกิด storage ordering คุณควรเข้าถึงข้อมูลที่แบ่งใช้อย่างเป็นลำดับ โดยใช้กลไกของ synchronization แบบใดแบบหนึ่งจากที่กล่าวมาแล้ว. ในตัวอย่างนี้ ที่ๆ thread หลายตัวแข่งขันกันสำหรับรีซอร์สที่มีการแบ่งใช้ ต้องปรับตัวมันเองให้เหมาะกับรูปแบบของล็อก. จะมีการอธิบายถึงการแก้ปัญหาโดยการใช้ space location lock และตามด้วยทางเลือกหนึ่งของการแก้ปัญหาคือการใช้กลไก compare-and-swap.

```
              THREAD A                                    THREAD B
-------------------------------------       -------------------------------------
for (i=0; i<10; ++i) {                        for (i=0; i<10; ++i) {
  /* Get an exclusive lock on the shared       /* Get an exclusive lock on the shared
     data.  We go into a wait state until         data.  We go into a wait state until
     the lock is granted. */                      the lock is granted. */
  locksl( LOCK_LOC, _LENR_LOCK );             locksl( LOCK_LOC, _LENR_LOCK );

  /* Update the shared data */                    /* Update the shared data */
  data1 += 5;                                    data1 += 4;
  data2 += 10;                                   data2 += 6;

  /* Unlock the shared data */                   /* Unlock the shared data */
  unlocksl( LOCK_LOC, _LENR_LOCK );           unlocksl( LOCK_LOC, _LENR_LOCK );
}                                             }
```

การจำกัดการเข้าถึงข้อมูลที่มีการแบ่งใช้ด้วย lock รับประกันว่ามี thread เพียง 1 ตัวเท่านั้นที่จะสามารถเข้าถึงข้อมูลได้ใน 1 ครั้ง. วิธีนี้จะแก้ปัญหาเรื่อง race condition ได้. และยังแก้ปัญหาในเรื่อง storage access ordering ได้ด้วย ดังนั้นจะไม่มีการพึ่งพาในการเรียงลำดับระหว่างตำแหน่งของ shared storage 2 ตำแหน่งอีกต่อไป.

ทางเลือก ในการแก้ปัญหา: การใช้ Check Lock Value / Clear Lock Value

Space location lock เหมือนกับวิธีการที่ใช้ในข้อแก้ไขข้อแรกคือ เต็มไปด้วยคุณลักษณะที่ไม่เป็นที่ต้องการในตัวอย่างธรรมดาเช่นนี้. ตัวอย่างเช่น space location lock สนับสนุนค่า time-out ซึ่งยอมให้โพรเซส ดำเนินต่อไป ถ้าไม่สามารถที่จะได้รับ lock ภายในช่วงเวลาหนึ่ง. Space locations locks ยังสนับสนุนการรวมกันในบางลักษณะของ shared locks. สิ่งเหล่านี้เป็นคุณลักษณะที่สำคัญ, แต่จะมาพร้อมราคาของโอเวอร์เฮดในด้านประสิทธิภาพที่สูงบางอย่าง.

ทางเลือกหนึ่งคือการใช้ *Check Lock Value* และ *Clear Lock Value*. พร้อมกันนี้, คำสั่ง MI ทั้งสองนี้ได้ให้แนวทาง ในการปฏิบัติที่ง่ายมาก และโปรโตคอลในการล็อกที่รวดเร็ว, โดยเฉพาะ ถ้าไม่มี contention ที่มากไปบนล็อก.

ในการแก้ปัญหานี้, ระบบจะใช้ CHKLKVAL เพื่อพยายามที่จะเข้าครอบครองการล็อก. ถ้าการพยายามนั้นล้มเหลว (สาเหตุจากระบบได้พบว่ามีการใช้ล็อกแล้ว), thread จะรออยู่ชั่วขณะ และจะพยายามใหม่อีกครั้ง, จะทำซ้ำจนกว่าจะเข้าครอบครองการล็อก ได้. หลังจากการอัพเดตข้อมูลที่แชร์, ระบบจะใช้ CLRLKVAL เพื่อถอดล็อก. ในตัวอย่างนี้, สมมติว่า, นอกจากตัวข้อมูลที่แชร์แล้ว, threads ยังร่วมแชร์แอดเดรสของตำแหน่ง 8-byte. โค้ด นี้อ้างถึงตำแหน่งนั้นเทียบกับตัวแปร LOCK. ต่อไป, สมมติว่า ล็อกถูก initialize เป็นศูนย์, ไม่ว่าทั้งการ initialize แบบ static หรือ การ initialize บางอย่างแบบ prior synchronized อย่างใดอย่างหนึ่ง.

```
              THREAD A                                    THREAD B
--------------------------------            --------------------------------
/* Do some work on the shared data */       /* Do some work on the shared data */
for (i=0; i<10; ++i) {                      for (i=0; i<10; ++i) {

  /* Attempt to acquire the lock using        /* Attempt to acquire the lock using
     CHKLKVAL.  By convention, use value         CHKLKVAL.  By convention, use value
     1 to indicate locked, 0 to indicate         1 to indicate locked, 0 to indicate
     unlocked. */                                unlocked. */
  while (_CHKLKVAL(&LOCK, 0, 1) == 1) {       while (_CHKLKVAL(&LOCK, 0, 1) == 1) {
```

```
    sleep(1);  /* wait a bit and try again */
  }

  /* Update the shared data */
  data1 += 5;
  data2 += 10;

  /* Unlock the shared data.  Use of
     CLRLKVAL ensures other jobs/threads
     see update to shared data prior to
     release of the lock. */
  _CLRLKVAL(&LOCK, 0);
}
```

```
    sleep(1);  /* wait a bit and try again */
  }

  /* Update the shared data */
  data1 += 4;
  data2 += 6;

  /* Unlock the shared data.  Use of
     CLRLKVAL ensures other jobs/threads
     see update to shared data prior to
     release of the lock. */
  _CLRLKVAL(&LOCK, 0);
}
```

ดังนี้, threads ใช้ Check Lock Value เพื่อปฏิบัติการทดสอบ race-free และ อัพเดตตัวแปรล็อก, และ Clear Lock Value เพื่อปรับค่าตัวแปรล็อก ให้เป็นสถานะปลดล็อก. ซึ่งเป็นการแก้ปัญหา race condition ที่เคยเกิดขึ้นในส่วนที่เป็นปัญหาในครั้งแรก. และยังชี้ให้เห็นปัญหาในเรื่อง storage access ordering อีกด้วย. ตามหมายเหตุก่อนหน้า, ถูกนำมาใช้ในสมัยนิยมนี้, Check Lock Value และ Clear Lock Value เป็น synchronizing actions. การใช้ Check Lock Value เพื่อตั้งค่าล็อกก่อนหน้า ในการอ่านข้อมูลที่แชร์ เพื่อให้แน่ใจว่า thread ได้อ่านข้อมูลที่อัพเดต ปัจจุบันที่สุด. การใช้ Clear Lock Value ในการลบค่าล็อกหลังจากการอัพเดตข้อมูลที่แชร์ เพื่อให้แน่ใจว่า การอัพเดตพร้อมสำหรับ subsequent ที่อ่านโดย thread ใดๆ หลังจากการ synchronizing action ถัดมา.

ภาคผนวก A. Output Listing จากคำสั่ง CRTPGM, CRTSRVPGM, UPDPGM, หรือ UPDSRVPGM

ในภาคผนวกนี้จะแสดงตัวอย่างของ binder listing และอธิบายถึงข้อผิดพลาดที่อาจเกิดขึ้นเป็นผลลัพธ์ของการใช้ binder language.

Binder Listing

Binder listing ของคำสั่ง Create Program (CRTPGM), Create Service Program (CRTSRVPGM), Update Program (UPDPGM), และ Update Service Program (UPDSRVPGM) ส่วนใหญ่จะมีลักษณะเหมือนกัน . หัวข้อนี้จะแสดง binder listing จากคำสั่ง CRTSRVPGM ที่ใช้ในการสร้างเซอร์วิสโปรแกรม FIANCIAL ในหัวข้อ “ตัวอย่างภาษา Binder” ในหน้า 95.

คุณสามารถกำหนดชนิดของ listing ได้ 3 ชนิด ลงในพารามิเตอร์ detail (DETAIL) ของคำสั่ง CRTPGM, CRTSRVPGM, UPDPGM, หรือ UPDSRVPGM:

*BASIC
*EXTENDED
*FULL

Basic Listing

ถ้าคุณกำหนดค่า DETAIL (*BASIC) ลงในคำสั่ง CRTPGM, CRTSRVPGM, UPDPGM, หรือ UPDSRVPGM listing จะประกอบด้วย:

- ค่าที่กำหนดในคำสั่ง CRTPGM, CRTSRVPGM, UPDPGM, หรือ UPDSRVPGM
- ตารางสรุป
- ข้อมูลที่แสดงถึงระยะเวลาที่กระบวนการรวมบางส่วนใช้ในการโพรเซส

รูปที่ 47, รูปที่ 48, and รูปที่ 49 ในหน้า 193 แสดงถึงข้อมูลเหล่านี้.

```
                                    Create Service Program                          Page     1

 Service program  . . . . . . . . . . . . . . . . . :   FINANCIAL
   Library  . . . . . . . . . . . . . . . . . . . . :     MYLIB
 Export . . . . . . . . . . . . . . . . . . . . . . :   *SRCFILE
 Export source file . . . . . . . . . . . . . . . . :   QSRVSRC
   Library  . . . . . . . . . . . . . . . . . . . . :     MYLIB
 Export source member . . . . . . . . . . . . . . . :   *SRVPGM
 Activation group . . . . . . . . . . . . . . . . . :   *CALLER
 Allow update  . . . . . . . . . . . . . . . . . . :   *YES
 Allow bound *SRVPGM library name update . . . . . :   *NO
 Creation options . . . . . . . . . . . . . . . . . :   *GEN        *NODUPPROC  *NODUPVAR   *DUPWARN
 Listing detail . . . . . . . . . . . . . . . . . . :   *FULL
 User profile . . . . . . . . . . . . . . . . . . . :   *USER
 Replace existing service program . . . . . . . . . :   *YES
 Target release . . . . . . . . . . . . . . . . . . :   *CURRENT
 Allow reinitialization . . . . . . . . . . . . . . :   *NO
Authority   . . . . . . . . . . . . . . . . . . . . :   *LIBCRTAUT
 Text . . . . . . . . . . . . . . . . . . . . . . . :

 Module      Library         Module      Library         Module      Library         Module      Library

 MONEY       MYLIB           CALCS       MYLIB
 RATES       MYLIB           ACCTS       MYLIB

 Service                     Service                     Service                     Service
 Program     Library         Program     Library         Program     Library         Program     Library

 *NONE

 Binding                     Binding                     Binding                     Binding
 Directory   Library         Directory   Library         Directory   Library         Directory   Library

 *NONE
```

รูปที่ 47. แสดงค่าที่กำหนดในคำสั่ง CRTSRVPGM

```
                                    Create Service Program                          Page     3

                                            Brief Summary Table

 Program entry procedures . . . . . . . . . . . :   0

 Multiple strong definitions  . . . . . . . . . :   0

 Unresolved references  . . . . . . . . . . . . :   0


                          * * * * *  E N D  O F  B R I E F  S U M M A R Y  T A B L E  * * * * *
```

รูปที่ 48. แสดง Brief Summary Table

```
                                   Create Service Program                       Page    23

                                          Binding Statistics

 Symbol collection CPU time . . . . . :                .018
 Symbol resolution CPU time . . . . . :                .006
 Binding directory resolution CPU time  . . . . . :    .403
 Binder language compilation CPU time . . . . . :      .040
 Listing creation CPU time  . . . . . :               1.622
 Program/service program creation CPU time  . . . . . :.178

 Total CPU time . . . . . :                           2.761
 Total elapsed time . . . . . :                      11.522

                          *****  E N D  O F  B I N D I N G  S T A T I S T I C S  *****

*CPC5D0B - Service program FINANCIAL created in library MYLIB.

                  *****  E N D  O F  C R E A T E  S E R V I C E  P R O G R A M  L I S T I N G  *****
```

รูปที่ 49. แสดง Binding Statistics

Extended Listing

ถ้าคุณกำหนดค่า DETAIL (*EXTENDED) ในคำสั่ง CRTPGM, CRTSRVPGM, UPDPGM, หรือ UPDSRVPGM listing นั้นจะประกอบด้วยข้อมูลทั้งหมดของ DETAIL (*BASIC) และเพิ่มเติมตารางสรุป. extended ตารางสรุป extended จะแสดงจำนวนของ import (reference) ที่ถูก resolve และจำนวนของ export (definition) ที่ถูกโพรเซส. สำหรับคำสั่ง CRTSRVPGM หรือ UPDSRVPGM listing ยังคงแสดงถึง binder language ที่ใช้, signature ที่ถูกสร้างขึ้น, และ import (reference) ที่เหมาะกับ export (definition). รูปที่ 50, รูปที่ 51 ในหน้า 194, และ รูปที่ 52 ในหน้า 195 แสดงถึงตัวอย่างของข้อมูลที่เพิ่มขึ้นมาเหล่านั้น.

```
                                   Create Service Program                       Page     2

                                          Extended Summary Table

   Valid definitions  . . . . . :          418
     Strong . . . . . :                    418
     Weak . . . . . :                         0
   Resolved references  . . . . . :         21
     To strong definitions  . . . . . :     21
     To weak definitions  . . . . . :        0

                          *****  E N D  O F  E X T E N D E D  S U M M A R Y  T A B L E  *****
```

รูปที่ 50. แสดง Extended Summary Listing

```
                                    Create Service Program                          Page     4

                                        Binder Information Listing
Module . . . . . :   MONEY
  Library  . . . . . :     MYLIB
  Bound  . . . . . :     *YES

     Number    Symbol      Ref       Identifier                      Type      Scope       Export       Key

     00000001  Def                   main                            Proc      Module      Strong
     00000002  Def                   Amount                          Proc      SrvPgm      Strong
     00000003  Def                   Payment                         Proc      SrvPgm      Strong
     00000004  Ref         0000017F  Q LE AG_prod_rc                 Data
     00000005  Ref         0000017E  Q LE AG_user_rc                 Data
     00000006  Ref         000000AC  _C_main                         Proc
     00000007  Ref         00000180  Q LE leDefaultEh                Proc
     00000008  Ref         00000181  Q LE mhConversionEh             Proc
     00000009  Ref         00000125  _C_exception_router             Proc

Module . . . . . :   RATES
  Library  . . . . . :     MYLIB
  Bound  . . . . . :     *YES

     Number    Symbol      Ref       Identifier                      Type      Scope       Export       Key

     0000000A  Def                   Term                            Proc      SrvPgm      Strong
     0000000B  Def                   Rate                            Proc      SrvPgm      Strong
     0000000C  Ref         0000017F  Q LE AG_prod_rc                 Data
     0000000D  Ref         0000017E  Q LE AG_user_rc                 Data
     0000000E  Ref         00000180  Q LE leDefaultEh                Proc
     0000000F  Ref         00000181  Q LE mhConversionEh             Proc
     00000010  Ref         00000125  _C_exception_router             Proc

Module . . . . . :   CALCS
  Library  . . . . . :     MYLIB
  Bound  . . . . . :     *YES

     Number    Symbol      Ref       Identifier                      Type      Scope       Export       Key

     00000011  Def                   Calc1                           Proc      Module      Strong
     00000012  Def                   Calc2                           Proc      Module      Strong
     00000013  Ref         0000017F  Q LE AG_prod_rc                 Data
     00000014  Ref         0000017E  Q LE AG_user_rc                 Data
     00000015  Ref         00000180  Q LE leDefaultEh                Proc
     00000016  Ref         00000181  Q LE mhConversionEh             Proc
     00000017  Ref         00000125  _C_exception_router             Proc

Module . . . . . :   ACCTS
  Library  . . . . . :     MYLIB
  Bound  . . . . . :     *YES

     Number    Symbol      Ref       Identifier                      Type      Scope       Export       Key

     00000018  Def                   OpenAccount                     Proc      SrvPgm      Strong
     00000019  Def                   CloseAccount                    Proc      SrvPgm      Strong
     0000001A  Ref         0000017F  Q LE AG_prod_rc                 Data
     0000001B  Ref         0000017E  Q LE AG_user_rc                 Data
     0000001C  Ref         00000180  Q LE leDefaultEh                Proc
     0000001D  Ref         00000181  Q LE mhConversionEh             Proc
     0000001E  Ref         00000125  _C_exception_router             Proc
```

รูปที่ 51. แสดง Binder Information Listing (Part 1 of 2) (ส่วนที่ 1 ของ 2)

```
Service program  . . . . . :   QC2SYS
  Library  . . . . . . . . . :     *LIBL
  Bound  . . . . . . . . . . :     *NO

      Number    Symbol      Ref       Identifier                        Type      Scope       Export       Key

      0000001F  Def                   system                            Proc                  Strong

Service program  . . . . . :   QLEAWI
  Library  . . . . . . . . . :     *LIBL
  Bound  . . . . . . . . . . :     *YES

      Number    Symbol      Ref       Identifier                        Type      Scope       Export       Key

      0000017E  Def                   Q LE AG_user_rc                   Data                  Strong
      0000017F  Def                   Q LE AG_prod_rc                   Data                  Strong
      00000180  Def                   Q LE leDefaultEh                  Proc                  Strong
      00000181  Def                   Q LE mhConversionEh               Proc                  Strong
```

รูปที่ 51. แสดง Binder Information Listing (Part 1 of 2) (ส่วนที่ 2 ของ 2)

```
                                        Create Service Program                              Page    14

                                              Binder Language Listing

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
  EXPORT SYMBOL('OpenAccount')
  EXPORT SYMBOL('CloseAccount')
ENDPGMEXP
******** Export signature: 00000000ADCEFEE088738A98DBA6E723.
STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
ENDPGMEXP
******** Export signature: 00000000000000000000ADC89D09E0C6E7.


                        *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 52. แสดง Binder Language Listing

Full Listing

ถ้าคุณกำหนดค่า DETAIL (*FULL) ในคำสั่ง CRTPGM, CRTSRVPGM, UPDPGM, หรือ UPDSRVPGM listing จะแสดงรายละเอียดทั้งหมดที่แสดงใน DETAIL (*EXTENDED) และเพิ่มด้วย cross-reference listing. รูปที่ 53 ในหน้า 197 แสดงตัวอย่างในส่วนที่เพิ่มเติมขึ้นมา.

Cross-Reference Listing

Identifier	Defs	Refs Ref	Refs Ref	Type	Library	Object
.	.			.	.	.
.	.			.	.	.
.	.			.	.	.
xlatewt	000000DD			*SRVPGM	*LIBL	QC2UTIL1
yn	00000140			*SRVPGM	*LIBL	QC2UTIL2
y0	0000013E			*SRVPGM	*LIBL	QC2UTIL2
y1	0000013F			*SRVPGM	*LIBL	QC2UTIL2
Amount	00000002			*MODULE	MYLIB	MONEY
Calc1	00000011			*MODULE	MYLIB	CALCS
Calc2	00000012			*MODULE	MYLIB	CALCS
CloseAccount	00000019			*MODULE	MYLIB	ACCTS
CEECRHP	000001A0			*SRVPGM	*LIBL	QLEAWI
CEECZST	0000019F			*SRVPGM	*LIBL	QLEAWI
CEEDATE	000001A9			*SRVPGM	*LIBL	QLEAWI
CEEDATM	000001B1			*SRVPGM	*LIBL	QLEAWI
CEEDAYS	000001A8			*SRVPGM	*LIBL	QLEAWI
CEEDCOD	00000187			*SRVPGM	*LIBL	QLEAWI
CEEDSHP	000001A1			*SRVPGM	*LIBL	QLEAWI
CEEDYWK	000001B3			*SRVPGM	*LIBL	QLEAWI
CEEFMDA	000001AD			*SRVPGM	*LIBL	QLEAWI
CEEFMDT	000001AF			*SRVPGM	*LIBL	QLEAWI
CEEFMTM	000001AE			*SRVPGM	*LIBL	QLEAWI
CEEFRST	0000019E			*SRVPGM	*LIBL	QLEAWI
CEEGMT	000001B6			*SRVPGM	*LIBL	QLEAWI
CEEGPID	00000195			*SRVPGM	*LIBL	QLEAWI
CEEGTST	0000019D			*SRVPGM	*LIBL	QLEAWI
CEEISEC	000001B0			*SRVPGM	*LIBL	QLEAWI
CEELOCT	000001B4			*SRVPGM	*LIBL	QLEAWI
CEEMGET	00000183			*SRVPGM	*LIBL	QLEAWI
CEEMKHP	000001A2			*SRVPGM	*LIBL	QLEAWI
CEEMOUT	00000184			*SRVPGM	*LIBL	QLEAWI
CEEMRCR	00000182			*SRVPGM	*LIBL	QLEAWI
CEEMSG	00000185			*SRVPGM	*LIBL	QLEAWI
CEENCOD	00000186			*SRVPGM	*LIBL	QLEAWI
CEEQCEN	000001AC			*SRVPGM	*LIBL	QLEAWI
CEESCEN	000001AB			*SRVPGM	*LIBL	QLEAWI
CEESECI	000001B2			*SRVPGM	*LIBL	QLEAWI
CEESECS	000001AA			*SRVPGM	*LIBL	QLEAWI
CEESGL	00000190			*SRVPGM	*LIBL	QLEAWI
CEETREC	00000191			*SRVPGM	*LIBL	QLEAWI
CEEUTC	000001B5			*SRVPGM	*LIBL	QLEAWI
CEEUTCO	000001B7			*SRVPGM	*LIBL	QLEAWI
CEE4ABN	00000192			*SRVPGM	*LIBL	QLEAWI
CEE4CpyDvfb	0000019A			*SRVPGM	*LIBL	QLEAWI
CEE4CpyIofb	00000199			*SRVPGM	*LIBL	QLEAWI
CEE4CpyOfb	00000198			*SRVPGM	*LIBL	QLEAWI
CEE4DAS	000001A4			*SRVPGM	*LIBL	QLEAWI
CEE4FCB	0000018A			*SRVPGM	*LIBL	QLEAWI
OpenAccount	00000018			*MODULE	MYLIB	ACCTS
Payment	00000003			*MODULE	MYLIB	MONEY
Q LE leBdyCh	00000188			*SRVPGM	*LIBL	QLEAWI
Q LE leBdyEpilog	00000189			*SRVPGM	*LIBL	QLEAWI
Rate	0000000B			*MODULE	MYLIB	RATES
Term	0000000A			*MODULE	MYLIB	RATES

รูปที่ 53. แสดง Cross-Reference Listing

IPA Listing Components

ในหัวข้อเหล่านี้จะอธิบายถึงส่วนประกอบของ IPA ในรายการ:

- Object File Map
- Compiler Options Map
- Inline Report
- Global Symbols Map
- Partition Map
- Source File Map
- Messages
- Message Summary

คำสั่ง CRTPGM หรือ CRTSRVPGM เป็นคำสั่งที่ทำให้เกิดส่วนประกอบที่กล่าวถึงนี้ทั้งหมด เว้นแต่ Inline Report เท่านั้นที่เกิดจากคุณกำหนดคำสั่งเป็น IPA(*YES) และ DETAIL(*BASIC หรือ *EXTENDED). คำสั่ง CRTPGM หรือ CRTSRVPGM จะทำให้เกิด Inline Report ก็ต่อเมื่อคุณกำหนดให้ IPA(*YES) และ DETAIL(*FULL) เท่านั้น.

Object File Map

ในหัวข้อของ Object File Map จะแสดงรายชื่อของอ็อบเจ็กต์ไฟล์ที่ถูกใช้เป็นอินพุตให้กับ IPA. ส่วนในหัวข้ออื่น เช่น Source File Map จะใช้หมายเลข FILE ID ที่ปรากฏในหัวข้อนี้.

Compiler Options Map

ในหัวข้อ Compiler Options Map จะแสดงอ็อพชันของคอมไพเลอร์ที่ระบุไว้ในข้อมูลของ IL สำหรับยูนิตย่อยแต่ละตัวที่ถูกโพรเซสอยู่. สำหรับแต่ละยูนิตย่อยจะแสดงอ็อพชันที่เกี่ยวข้องกับการโพรเซสของ IPA. คุณสามารถกำหนดอ็อพชันต่างๆ ได้โดยผ่านอ็อพชันของคอมไพเลอร์ ได้แก่ ไดเรกทีฟ #pragma หรือค่าดีฟอล์ต.

Inline Report

ในหัวข้อ Inline Report จะแสดงแอ็คชันที่ถูกกระทำโดย IPA. ในรายงานนี้ คำว่า 'subprogram' จะมีความหมายเหมือนฟังก์ชันในภาษา C/C++ หรือเมธอดในภาษา C++. ข้อสรุปจะประกอบไปด้วย:

- ชื่อของโปรแกรมย่อยที่ถูกระบุไว้. IPA จะจัดเรียงรายชื่อของโปรแกรมย่อยตามลำดับอักษร.
- เหตุผลที่เกิดแอ็คชันขึ้นในโปรแกรมย่อย:
  - คุณระบุ #pragma แบบ noinline ไว้ในโปรแกรมย่อย.
  - คุณระบุ #pragma แบบ inline ไว้ในโปรแกรมย่อย.
  - IPA ทำการประมวลผลแบบ inline ในโปรแกรมย่อยโดยอัตโนมัติ.
  - ไม่มีเหตุผลที่จะประมวลผลแบบ inline ในโปรแกรมย่อย.
  - มีความขัดแย้งในพาร์ติชัน.
  - IPA ไม่สามารถทำการ inline โปรแกรมย่อยได้เนื่องจากไม่มีข้อมูลของ IL อยู่.

- แอ็คชันที่เกิดขึ้นในโปรแกรมย่อย:
  - IPA ทำการ inline โปรแกรมย่อยอย่างน้อยหนึ่งครั้ง.
  - IPA ไม่ทำการ inline โปรแกรมย่อยเนื่องจากขนาดเริ่มต้นมีข้อจำกัด.
  - IPA ไม่ทำการ inline โปรแกรมย่อยเนื่องจากมีการขยายขนาดเกินกว่าที่กำหนด.
  - โปรแกรมย่อยถูกเสนอให้ทำการ inline แต่ IPA ไม่ทำการ inline ให้
  - โปรแกรมย่อยถูกเสนอให้ทำการ inline แต่ไม่ได้ถูกอ้างถึง.
  - โปรแกรมย่อยถูกเรียกซ้ำโดยตรง หรือการเรียกบางอย่างไม่ตรงกันกับพารามิเตอร์.
- สถานะของโปรแกรมย่อยหลังจากการ inline:
  - IPA จะยกเลิกโปรแกรมย่อยเนื่องจากไม่มีการอ้างถึง และถูกกำหนดให้เป็นแบบ static internal.
  - IPA จะไม่ยกเลิกโปรแกรมแกรมย่อยด้วยสาเหตุเหล่านี้:
    - โปรแกรมย่อยเป็นแบบ external. (ซึ่งมันไม่สามารถเรียกจากยูนิตที่อยู่ภายนอกได้.)
    - โปรแกรมย่อยเรียกตัวมันเอง.
    - โปรแกรมย่อยถูกนำแอดเดรสไปใช้งาน.
- ขนาดเริ่มต้นแบบ Relative ของโปรแกรมย่อย (ใน Abstract Code Units).
- ขนาดสุดท้ายแบบ Relative ของโปรแกรมย่อย (ใน Abstract Code Units) หลังจากการทำ Inline.
- จำนวนการเรียกภายในโปรแกรมย่อย และจำนวนของการเรียกเหล่านี้ที่ IPA ทำการ Inline เข้าไปในโปรแกรมย่อย.
- จำนวนครั้งที่โปรแกรมย่อยถูกเรียกโดยโปรแกรมย่อยอื่นใน Compile Unit และจำนวนครั้งที่ IPA ทำการ Inline โปรแกรมย่อย.
- โหมดที่ถูกเลือก และค่า Threshold และ Limit ที่ระบุไว้. ฟังก์ชันสแตติกซึ่งอาจมีชื่อไม่เฉพาะภายใน แอ็พพลิเคชันโดยรวม จะมีชื่อที่ขึ้นต้นด้วย `@nnn@` หรือ `XXXX@nnn@`, ซึ่ง `XXXX` เป็นชื่อพาร์ติชัน, และ `nnn` เป็นหมายเลขซอร์สไฟล์.

ในรายละเอียดของการเรียกจะมีข้อมูลเหล่านี้ปรากฏอยู่ในแต่ละโปรแกรมย่อย ซึ่งได้แก่:

- โปรแกรมย่อยที่ถูกเรียก.
- โปรแกรมย่อยที่เรียกมัน.
- โปรแกรมย่อยของมันที่ถูกทำการ Inline.

ข้อมูลเหล่านี้จะช่วยให้คุณสามารถวิเคราะห์โปรแกรมได้ดียิ่งขึ้น หากคุณต้องการใช้วิธี Inline ในโหมด Selective. ค่าที่นับได้ในรายงานนี้จะไม่รวมการเรียกจากโปรแกรมที่ไม่ใช่แบบ IPA.

Global Symbols Map

ในหัวข้อ Global Symbols Map จะแสดงวิธีที่สัญลักษณ์โกลบอลแม็พเข้าเป็นสมาชิกของโครงสร้างข้อมูลแบบโกลบอลโดยกระบวนการ Global Variable Coalescing Optimization. ซึ่งจะแสดงทั้งข้อมูลของสัญลักษณ์ และชื่อไฟล์ (ข้อมูลเกี่ยวกับไฟล์โดยย่อ). และยังแสดงหมายเลขบรรทัดไว้อีกด้วย.

Partition Map

ในหัวข้อ Partition Map จะแสดงพาร์ติชันของอ็อบเจ็กต์โค้ดที่สร้างโดย IPA. ซึ่งมีรายละเอียดดังต่อไปนี้:

- เหตุผลในการสร้างอ็อบเจ็กต์โค้ดในแต่ละพาร์ติชัน.
- อ็อพชันที่ใช้ในการสร้างอ็อบเจ็กต์โค้ด.
- ฟังก์ชันและข้อมูลแบบโกลบอลที่รวมอยู่ในพาร์ติชัน.
- ซอร์สไฟล์ที่ถูกใช้เพื่อสร้างพาร์ติชัน.

Source File Map

ในหัวข้อ Source File Map จะแสดงซอร์สไฟล์ที่อยู่ในอ็อบเจ็กต์ไฟล์.

Messages

ถ้า IPA ตรวจพบข้อผิดพลาด หรือความผิดพลาดที่อาจเกิดขึ้น มันก็จะสร้างข้อความวินิจฉัยปัญหาและแสดงไว้ในหัวข้อ Messages. ในหัวข้อนี้จะแสดงข้อความต่างๆ ที่เกิดขึ้นในระหว่างกระบวนการ IPA. ซึ่งข้อความเหล่านี้จะถูกเรียงไว้ตามลำดับความสำคัญ. ในหัวข้อ Messages ยังแสดงหมายเลขหน้าที่ข้อความเหล่านั้นปรากฏอยู่ด้วย. อีกทั้งยังบอกข้อความ, อ็อพชัน, ข้อมูลที่เกี่ยวข้องกับไฟล์ เช่น ชื่อไฟล์, บรรทัด (หากมีข้อมูล), และคอลัมน์ (หากมีข้อมูล).

Message Summary

ในหัวข้อ Message Summary จะแสดงจำนวนของข้อความทั้งหมด และจำนวนข้อความในแต่ละระดับความสำคัญ.

รายการสำหรับเซอร์วิสโปรแกรมตัวอย่าง

รูปที่ 49 ในหน้า 193, รูปที่ 51 ในหน้า 194 และ รูปที่ 53 ในหน้า 197 แสดงถึง listing data บางส่วนที่ถูกสร้างขึ้นเมื่อค่า DETAIL (*FULL) ถูกกำหนดขึ้นเพื่อสร้างเซอร์วิสโปรแกรม FINANCIAL ใน รูปที่ 36 ในหน้า 100. รูปทั้งหมดแสดงถึงสถิติในการรวม, binder information listing, และ cross-reference listing.

Binder Information Listing สำหรับเซอร์วิสโปรแกรมตัวอย่าง

Binder Information Listing (รูปที่ 51 ในหน้า 194) ประกอบด้วยข้อมูลและหัวข้อในคอลัมน์ต่างๆ ดังนี้:

- ไลบรารีและชื่อของโมดูลหรือเซอร์วิสโปรแกรมที่ถูกโพรเซส.

  ถ้าฟิลด์ *Bound* แสดงค่าเป็น *YES สำหรับโมดูลอ็อบเจ็กต์ แสดงว่าโมดูลนั้นถูกกำหนดให้ถูกรวมโดยการก็อปปี. และถ้าฟิลด์ *Bound* แสดงค่าเป็น *YES สำหรับเซอร์วิสโปรแกรม ก็แสดงว่าเซอร์วิสโปรแกรมนั้นถูกรวมโดยการอ้างอิง. แต่ถ้าฟิลด์ *Bound* แสดงค่าเป็น *NO ทั้งกับโมดูลอ็อบเจ็กต์และเซอร์วิสโปรแกรม นั่นคืออ็อบเจ็กต์นั้นไม่ต้องถูกรวม. เนื่องมาจากอ็อบเจ็กต์นั้นไม่มี export ที่ตรงกับ import ที่ไม่ถูก resolve.

- Number

  เป็นตัวเลขสำหรับโมดูลหรือเซอร์วิสโปรแกรมแต่ละตัวที่ถูกโพรเซส จะเป็น ID เฉพาะที่เกี่ยวข้องกับ export (definition) หรือ import (reference) แต่ละตัว.

- Symbol

  คอลัมน์นี้แสดงถึงชื่อของ symbol ที่เป็น export (Def) หรือ import (Ref).

- Ref

  ตัวเลขที่กำหนดในคอลัมน์นี้เป็น ID เฉพาะของ export (Def) ที่ตอบสนองการร้องขอของ import. ดังตัวอย่างจาก รูปที่ 51 ในหน้า 194 ID เฉพาะของ import 00000005 จะเหมาะสมกับ ID เฉพาะของ export 0000017E.

- Identifier

  เป็นชื่อของ symbol ที่ถูก export หรือ import. ชื่อของ symbol import สำหรับ ID เฉพาะ 00000005 คือ Q LE AG_user_rc. และชื่อของ symbol export สำหรับ ID เฉพาะ 0000017E ก็คือ Q LE AG_user_rc เช่นกัน.

- Type

  ถ้าชื่อของ symbol เป็นโพรซีเดอร์ คอลัมน์นี้จะแสดงค่าเป็น Proc. และถ้าชื่อของ symbol เป็นตัวข้อมูล คอลัมน์นี้จะแสดงค่าเป็น Data.

- Scope

  สำหรับโมดูล คอลัมน์นี้แสดงถึงชื่อของ symbol ที่ถูก export ถูกเข้าถึงในระดับโมดูลหรือในระดับ public interface ของเซอร์วิสโปรแกรม. ถ้าโปรแกรมถูกสร้างขึ้น ชื่อของ symbol ที่ถูก export จะถูกเข้าถึงได้ที่ระดับโมดูลเท่านั้น. แต่ถ้าเซอร์วิสโปรแกรมถูกสร้างขึ้น ชื่อของ symbol ที่ถูก export จะถูกเข้าถึงได้ที่ระดับโมดูลและระดับเซอร์วิสโปรแกรม (SrvPgm). และถ้า symbol ที่ถูก export เป็นส่วนหนึ่งของ public interface ค่าในคอลัมน์ *Scope* จะต้องเป็น SrvPgm.

- Export

  คอลัมน์นี้แสดงถึงความแข็งแกร่งของตัวข้อมูลที่ถูก export จากโมดูลหรือเซอร์วิสโปรแกรม.

- Key

  คอลัมน์นี้จะมีข้อมูลเพิ่มเติมเกี่ยวกับ weak export บางตัว. ซึ่งโดยปกติแล้วคอลัมน์นี้จะว่าง.

Cross-Reference Listing สำหรับเซอร์วิสโปรแกรมตัวอย่าง.

Cross-reference listing จาก รูปที่ 53 ในหน้า 197 เป็นอีกวิธีหนึ่งในการดูข้อมูลที่แสดงใน binder information. cross-reference listing ประกอบด้วยหัวข้อในคอลัมน์ต่างๆ ดังนี้:

- Identifier

  แสดงชื่อของ export ที่ถูกโพรเซสในขณะเกิด symbol resolution.

- Defs

  แสดง ID เฉพาะของ export แต่ละตัว.

- Refs

  ตัวเลขที่แสดงในคอลัมน์นี้คือ ID เฉพาะสำหรับ import(Ref) ที่ถูก resolve ให้กับ export (Def).

- Type

  แสดงให้เห็นว่า export มาจากอ็อบเจ็กต์ *MODULE หรือ *SRVPGM.

- Library

  แสดงชื่อของไลบรารีที่ถูกกำหนดไว้ในคำสั่งหรือใน binding directory.

- Object

แสดงชื่อของอ็อบเจ็กต์ที่ให้ export (Def).

Binding Statistics สำหรับเซอร์วิสโปรแกรมตัวอย่าง.

รูปที่ 49 ในหน้า 193 แสดงชุดของสถิติต่างๆ ในการสร้างเซอร์วิสโปรแกรม FINANCIAL. สถิติเหล่านี้ แสดงให้เห็นถึงแต่ละจุดที่ตัวรวมใช้เวลาไปเมื่อมันโพรเซสการร้องขอ. คุณสามารถควบคุมข้อมูลที่แสดงในหัวข้อนี้ได้ทางอ้อมเท่านั้น. จำนวนของโอเวอร์เฮดในการโพรเซสไม่สามารถถูกวัดได้. ดังนั้นค่าที่แสดงในฟิลด์ *Total CPU time* จะมีค่ามากกว่าเวลาของฟิลด์ก่อนหน้านี้ทั้งหมดรวมกัน.

ข้อผิดพลาดของ Binder Language

ในขณะที่ระบบกำลังโพรเซส binder language ระหว่างการสร้างเซอร์วิสโปรแกรมอาจมีข้อผิดพลาดเกิดขึ้นได้. ถ้ามีการกำหนด DETAIL(*EXTENDED) หรือ DETAIL(*FULL) ในคำสั่ง Create Service Program คุณสามารถเห็นข้อผิดพลาดที่เกิดขึ้นได้ใน spool file.

อาจมีข้อความแสดงข้อมูล (information message) เกิดขึ้นได้ดังนี้:

- Signature padded
- Signature truncated

ข้อความแสดงความผิดพลาดในระดับเตือน (warning error) ที่อาจเกิดขึ้น มีได้ดังนี้:

- Current export block limits interface
- Duplicate export block
- Duplicate symbol on previous export
- Level checking cannot be disabled more than once, ignored
- Multiple current export blocks not allowed, previous assumed

ข้อความแสดงความผิดพลาดในระดับรุนแรง (serious error) ที่อาจเกิดขึ้น มีได้ดังนี้:

- Current export block is empty
- Export block not completed, end-of-file found before ENDPGMEXP
- Export block not started, STRPGMEXP required
- Export blocks cannot be nested, ENDPGMEXP missing
- Exports must exist inside export blocks
- Identical signatures for dissimilar export blocks, must change exports
- Multiple wildcard matches
- No current export block
- No wildcard match
- Previous export block is empty
- Signature contains variant characters
- SIGNATURE(*GEN) required with LVLCHK(*NO)

- Signature syntax not valid
- Symbol name required
- Symbol not allowed as service program export
- Symbol not defined
- Syntax not valid

Signature Padded

รูปที่ 54แสดงถึง binder language listing ที่มีข้อความดังนี้:

```
                                Binder Language Listing

             STRPGMEXP  SIGNATURE('Short signature')
******** Signature padded
             EXPORT     SYMBOL('Proc_2')
             ENDPGMEXP

******** Export signature: E2889699A340A289879581A3A4998540.

           * * * * *  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  * * * * *
```

รูปที่ 54. แสดง Signature Padded เนื่องจาก signature ที่ให้มีความยาวน้อยกว่า 16 ไบต์.

นี่คือข้อความแสดงข้อมูล.

การแก้ไขที่เหมาะสม

ไม่มีความต้องการการเปลี่ยนแปลงใดๆ.

ถ้าคุณต้องการจะหลีกเลี่ยงข้อความชนิดนี้ จะต้องแน่ใจว่า signature ที่ได้มีความยาว 16 ไบต์.

Signature Truncated

รูปที่ 55แสดงถึง binder language listing ที่มีข้อความดังนี้.

```
                                Binder Language Listing

             STRPGMEXP  SIGNATURE('This signature is very long')
******** Signature truncated
             EXPORT     SYMBOL('Proc_2')
             ENDPGMEXP

******** Export signature: E38889A240A289879581A3A499854089.

           * * * * *  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  * * * * *
```

รูปที่ 55. แสดง Signature Truncated เนื่องจาก Signature มีความยาวเกิน 16 ไบต์.

นี่คือข้อความแสดงข้อมูล.

การแก้ไขที่เหมาะสม

ไม่มีความต้องการการเปลี่ยนแปลงใดๆ.

ถ้าคุณต้องการจะหลีกเลี่ยงข้อความชนิดนี้ จะต้องแน่ใจว่า signature ที่ได้มีความยาว 16 ไบต์.

Current Export Block Limits Interface

รูปที่ 56แสดงถึง binder language listing ที่มีข้อผิดพลาดนี้.

```
                        Binder Language Listing

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000CD2.
STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
  EXPORT SYMBOL(C)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000CDE3.
******** Current export block limits interface.


          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

*รูปที่ 56. แสดง PGMLVL(*PRV) Exporte Symbol มากกว่า PGMLVL(*CURRENT)*

นี่คือ ข้อความแสดงความผิดพลาดในระดับเตือน.

PGML (*PRV) export block ได้ระบุสัญลักษณ์มากกว่า PGMLVL(*CURRENT) export block.

หากไม่มีข้อผิดพลาดอื่นๆ เกิดขึ้น เซอร์วิสโปรแกรมจะถูกสร้างขึ้น.

ถ้าข้อความทั้งสองนี้เป็นจริง:

- PGMLVL(*PRV) เคยสนับสนุนโพรซีเดอร์ C.
- โพรซีเดอร์ C ไม่ถูกสนับสนุนอีกต่อไป ภายใต้เซอร์วิสโปรแกรมใหม่.

โปรแกรมหรือเซอร์วิสโปรแกรม ILE ที่เรียกโพรซีเดอร์ C ในเซอร์วิสโปรแกรมนี้ จะมีข้อผิดพลาดในขณะรัน.

การแก้ไขที่เหมาะสม

1. แน่ใจว่า PGMLVL(*CURRENT) export block มี symbol ที่ถูก export มากกว่า PGMLVL (*PRV) export block.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ EXPORT SYMBOL(C) ถูกเพิ่มให้กับ STRPGMEXP PGMLVL(*PRV) export block แทนที่จะถูกเพิ่มให้กับ PGMLVL(*CURRENT) block ซึ่งไม่ถูกต้อง.

Duplicate Export Block

รูปที่ 57แสดงถึง binder language listing ที่มีข้อผิดพลาดนี้.

```
                        Binder Language Listing

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000CD2.
STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000CD2.
******** Duplicate export block.


          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 57. แสดง STRPGMEXP/ENDPGMEXP Block ที่เหมือนกัน

นี่คือ ข้อความแสดงความผิดพลาดในระดับเตือน.

มี block ของ STRPGMEXP และ ENDPGMEXP มากกว่า 1 block ที่ทำการ export symbol ตัวเดียวกันในลำดับที่เหมือนกัน.

หากไม่มีข้อผิดพลาดอื่นๆ เกิดขึ้นเซอร์วิสโปรแกรมจะถูกสร้างขึ้น. signature ที่ซ้ำกันจะถูกรวมในเซอร์วิสโปรแกรมเพียงครั้งเดียว.

การแก้ไขที่เหมาะสม

1. ทำการเปลี่ยนแปลงตามข้อใดข้อหนึ่งดังนี้:
   - แน่ใจว่า PGMLVL(*CURRENT) export block มีค่าถูกต้อง. โดยอัพเดตมันให้มีค่าที่เหมาะสม.
   - ลบ export block ที่ซ้ำกันออกไป.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ คำสั่ง STRPGMEXP ที่กำหนดค่า PGMLVL(*CURRENT) ต้องการการเพิ่มเติมบรรทัดคำสั่งต่อจาก EXPORT SYMBOL(B) ดังนี้:

```
EXPORT  SYMBOL(C)
```

Duplicate Symbol on Previous Export

รูปที่ 58แสดงถึง binder language listing ที่มีข้อผิดพลาดที่เกิดจากการมี symbol ซ้ำกัน.

```
                                Binder Language Listing

 STRPGMEXP  PGMLVL(*CURRENT)
   EXPORT SYMBOL(A)
   EXPORT SYMBOL(B)
   EXPORT SYMBOL(A)
 ******** Duplicate symbol on previous export
   EXPORT SYMBOL(C)
 ENDPGMEXP
 ******** Export signature: 00000000000000000000000000CDED3.


           *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 58. แสดง Duplicate Exported Symbols

นี่คือ ข้อความแสดงความผิดพลาดในระดับเตือน.

มีการกำหนดให้ symbol ตัวหนึ่งถูก export จากเซอร์วิสโปรแกรมมากกว่า 1 ครั้งใน STRPGMEXP และ ENDPGMEXP block.

หากไม่มีข้อผิดพลาดอื่นๆ เกิดขึ้น เซอร์วิสโปรแกรมจะถูกสร้างขึ้น. สำหรับ symbol ที่ซ้ำกัน จะมีเพียงชุดแรกเท่านั้นที่จะถูก export. แต่ symbol ที่ซ้ำกันทั้งหมดจะส่งผลกระทบกับ signature ที่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ลบบรรทัดคำสั่งในซอร์สไฟล์ของ binder language ที่ซ้ำกันออก 1 บรรทัด.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ เป็นการลบ EXPORT SYMBOL(A) บรรทัดที่ 2 ออกไป.

Level Checking Cannot Be Disabled More than Once, Ignored

รูปที่ 59 ในหน้า 207 แสดงถึง binder language listing ที่มีข้อผิดพลาดนี้.

```
STRPGMEXP  PGMLVL(*CURRENT) LVLCHK(*NO)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000000.
STRPGMEXP  PGMLVL(*PRV) LVLCHK(*NO)
******** Level checking cannot be disabled more than once, ignored
  EXPORT SYMBOL(A)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000C1.

          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

*รูปที่ 59. แสดงคำสั่ง STRPGMEXP หลายตัวที่มีการกำหนด LVLCHK(*NO).*

นี่คือ ข้อความแสดงความผิดพลาดในระดับเตือน.

มี STRPGMEXP block มากกว่า 1 block ถูกกำหนดเป็น LVLCHK(*NO).

ถ้าไม่มีข้อผิดพลาดอื่นเกิดขึ้น เซอร์วิสโปรแกรมจะถูกสร้างขึ้น ค่า LVLCHK(*NO). ลำดับที่ 2 จะถูกถือว่าเป็น LVLCHK(*YES).

การแก้ไขที่เหมาะสม

1. แน่ใจว่ามี STRPGMEXP block ที่กำหนดค่า LVLCHK(*NO) เพียงแค่ block เดียว.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ PGMLVL(*PRV) export block เป็น block เดียวที่กำหนดค่า LVLCHK(*NO). ดังนั้นค่า LVLCHK(*NO) อีกตัวหนึ่งใน PGMLVL(*CURRENT) export block จะต้องถูกลบออก.

Multiple Current Export Blocks Not Allowed, Previous Assumed

รูปที่ 60 ในหน้า 208 แสดงถึง binder language listing ที่มีข้อผิดพลาดนี้.

```
                                Binder Language Listing

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
  EXPORT SYMBOL(C)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000CDE3.
STRPGMEXP
  EXPORT SYMBOL(A)
******** Multiple 'current' export blocks not allowed, 'previous' assumed.
  EXPORT SYMBOL(B)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000CD2.


          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

*รูปที่ 60. แสดงการกำหนดค่า PGMLVL(*CURRENT) มากกว่าหนึ่งครั้ง*

นี่คือ ข้อความแสดงความผิดพลาดในระดับเตือน.

ค่า PGMLVL(*CURRENT) ที่ถูกกำหนดหรือยอมให้เป็นค่าดีฟอลต์ในคำสั่ง STRPGMEXP มากกว่า 1 คำสั่ง. export block ที่ 2 ที่มีค่าเป็น PGMLVL(*CURRENT) จะถูกถือว่าเป็น PGMLVL (*PRV).

ถ้าไม่มีข้อผิดพลาดอื่นเกิดขึ้น เซอร์วิสโปรแกรมจะถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. เปลี่ยนค่า PGMLVL ที่ต้องแก้ไขให้เป็น PGMLVL(*PRV).
2. รันคำสั่ง CRTSRVPGM อีกครั้ง

ในตัวอย่างนี้ คำสั่ง STRPGMEXP ลำดับที่ 2 จะต้องถูกแก้ไข.

Current Export Block Is Empty

รูปที่ 61 ในหน้า 209 แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                                Binder Language Listing

STRPGMEXP  PGMLVL(*CURRENT)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000000.
***ERROR Current export block is empty.

          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

*รูปที่ 61. ไม่มี Symbol ถูก Export จาก STRPGMEXP PGMLVL(*CURRENT) Block.*

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

ไม่มีการกำหนด symbol ให้ถูก export จาก *CURRENT export block.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่ง ดังนี้:
   - เพิ่มชื่อของ symbol ที่จะถูก export.
   - ท ลบ STRPGMEXP–ENDPGMEXP export block ที่ว่างออกไปและสร้าง block ใหม่ที่มีค่า PGMLVL(*CURRENT).
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ บรรทัดคำสั่งต่อไปนี้จะถูกเพิ่มเข้าไปใน binder language source file ระหว่างคำสั่ง STRPGMEXP และ ENDPGMEXP:

```
EXPORT SYMBOL(A)
```

Export Block Not Completed, End-of-File Found before ENDPGMEXP

รูปที่ 62แสดง binder language listing ที่มีข้อผิดพลาดนี้

```
                                Binder Language Listing

STRPGMEXP  PGMLVL(*CURRENT)
***ERROR Syntax not valid.
***ERROR Export block not completed, end-of-file found before ENDPGMEXP.

          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 62. ไม่พบคำสั่ง ENDPGMEXP แต่พบจุดสิ้นสุดของซอร์สไฟล์.

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

ไม่พบคำสั่ง ENDPGMEXP ก่อนที่จะสิ้นสุดไฟล์.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่ง ดังนี้:
   - เพิ่มคำสั่ง ENDPGMEXP ลงในตำแหน่งที่เหมาะสม.
   - ลบคำสั่ง STRPGMEXP ใดๆ ที่ไม่มีคำสั่ง ENDPGMEXP ที่เข้าคู่กัน และลบชื่อของ symbol ที่จะต้องถูก export ออกไปด้วย.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ จะมีการเพิ่มบรรทัดต่อไปนี้ต่อจากคำสั่ง STRPGMEXP:

```
  EXPORT  SYMBOL(A)
ENDPGMEXP
```

Export Block Not Started, STRPGMEXP Required

รูปที่ 63แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                        Binder Language Listing

ENDPGMEXP
***ERROR Export block not started, STRPGMEXP required.
***ERROR No 'current' export block


          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 63. ไม่พบคำสั่ง STRPGMEXP.

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

ไม่พบคำสั่ง STRPGMEXP ก่อนที่จะพบคำสั่ง ENDPGMEXP.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่ง ดังนี้:
   - เพิ่มคำสั่ง STRPGMEXP ลงไป.
   - ลบ symbol ที่ถูก export และคำสั่ง ENDPGMEXP ออกไป.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ จะมีการเพิ่ม 2 บรรทัดต่อไปนี้ลงใน binder language source file ก่อนหน้าคำสั่ง ENDPGMEXP.

```
STRPGMEXP
  EXPORT  SYMBOL(A)
```

Export Blocks Cannot Be Nested, ENDPGMEXP Missing

รูปที่ 64แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                                Binder Language Listing

 STRPGMEXP  PGMLVL(*CURRENT)
   EXPORT SYMBOL(A)
   EXPORT SYMBOL(B)
 STRPGMEXP  PGMLVL(*PRV)
 ***ERROR Export blocks cannot be nested, ENDPGMEXP missing.
   EXPORT SYMBOL(A)
 ENDPGMEXP
 ******** Export signature: 00000000000000000000000000000C1.


            *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 64. ไม่พบคำสั่ง ENDPGMEXP

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

ไม่พบคำสั่ง ENDPGMEXP ก่อนที่จะพบคำสั่ง STRPGMEXP อีกตัวหนึ่ง.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่ง ดังนี้:
   - เพิ่มคำสั่ง ENDSTRPGM ก่อนคำสั่ง STRPGMEXP ตัวต่อไป.
   - ลบคำสั่ง STRPGMEXP และชื่อของ symbol ที่ถูก export ออกไป.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ คำสั่ง ENDPGMEXP จะถูกเพิ่มลงไปใน binder source file ก่อนหน้าคำสั่ง STRPGMEXP ลำดับที่ 2.

Exports Must Exist inside Export Blocks

รูปที่ 65 ในหน้า 212 แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                                Binder Language Listing

 STRPGMEXP  PGMLVL(*CURRENT)
   EXPORT SYMBOL(A)
   EXPORT SYMBOL(B)
 ENDPGMEXP
 ******** Export signature: 00000000000000000000000000000CD2.
   EXPORT SYMBOL(A)
 ***ERROR Exports must exist inside export blocks.


           *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 65. แสดงชื่อของ Symbol ที่ถูก Export อยู่ภายนอก STRPGMEXP-ENDPGMEXP Block.

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

Symbol ที่จะต้องถูก export ไม่ได้ถูกกำหนดไว้ใน STRPGMEXP-ENDPGMEXP block.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่งดังนี้:
   - ย้าย symbol ที่จะถูก export. ให้อยู่ใน STRPGMEXP-ENDPGMEXP block.
   - ลบ symbol ออกไป.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ บรรทัดที่เป็นข้อผิดพลาดจะถูกลบออกจาก binder language source file.

Identical Signatures for Dissimilar Export Blocks, Must Change Exports

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

Signature ที่เหมือนกันถูกสร้างขึ้นจาก STRPGMEXP-ENDPGMEXP block ที่ทำการ export symbol ที่แตกต่างกัน. ข้อผิดพลาดเช่นนี้ไม่น่าจะเกิดขึ้นได้. สำหรับกลุ่มของ symbol ที่ถูก export ที่มีความสำคัญมาก ใดๆ ข้อผิดพลาดนี้มีโอกาสเกิดขึ้นได้ 1 ใน 3.4E28 ครั้ง.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่งดังนี้:
   - เพิ่ม symbol ที่จะถูก export จาก PGMLVL(*CURRENT) block.
     วิธีที่ควรใช้คือกำหนด symbol ที่เคยถูก export แล้ว. วิธีนี้จะทำให้เกิด ข้อความแสดงความผิดพลาดในระดับเตือน ของการมี symbol ที่ซ้ำกัน แต่ก็จะช่วยให้แน่ใจว่า signature จะไม่ซ้ำกัน. และอีกวิธีหนึ่งคือเพิ่ม symbol ตัวอื่นที่ยังไม่เคยถูก export.

- เปลี่ยนชื่อของ symbol ที่จะถูก export ในโมดูลและ binder language source file ในส่วนที่เกี่ยวข้องกัน.
- กำหนด signature โดยใช้พารามิเตอร์ SIGNATURE ในคำสั่ง Start Program Export (STRPGMEXP).

2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

Multiple Wildcard Matches

รูปที่ 66แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                        Binder Language Listing

STRPGMEXP PGMLVL(*CURRENT)
EXPORT ("A"<<<)
***ERROR Multiple matches of wildcard specification
EXPORT ("B"<<<)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000FFC2.


   * * * * *  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G
```

รูปที่ 66. แสดง Multiple Matches of Wildcard Specification

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

เครื่องหมาย wildcard ที่กำหนดสำหรับ export ตรงกับ symbol ที่จะ export มากกว่า 1 ตัว.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. กำหนด wildcard ที่มีรายละเอียดมากขึ้น เพื่อให้ export ที่เป็นไปตามข้อกำหนดเป็น export ที่ต้องการเท่านั้น.
2. 2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

No Current Export Block

รูปที่ 67 ในหน้า 214 แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                                Binder Language Listing

 STRPGMEXP  PGMLVL(*PRV)
   EXPORT SYMBOL(A)
 ENDPGMEXP
 ******** Export signature: 00000000000000000000000000000C1.
 ***ERROR No 'current' export block

           *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

*รูปที่ 67. แสดง No PGMLVL(*CURRENT) Export Block*

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

ไม่พบ STRPGMEXP PGMLVL(*CURRENT) ใน binder language source file.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่ง ดังนี้:
   - เปลี่ยน PGMLVL(*PRV) ให้เป็น PGMLVL(*CURRENT).
   - เพิ่ม STRPGMEXP–ENDPGMEXP block ที่เป็น *CURRENT export block.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ ค่า PGMLVL(*PRV) ถูกเปลี่ยนให้เป็น PGMLVL(*CURRENT).

No Wildcard Matches

รูปที่ 68แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                                Binder Language Listing

STRPGMEXP PGMLVL(*CURRENT)
EXPORT ("Z"<<<)
***ERROR No matches of wildcard specification
EXPORT ("B"<<<)
ENDPGMEXP
******** Export signature: 0000000000000000000000000000FFC2.

    *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G
```

รูปที่ 68. แสดง No Matches of Wildcard Specification.

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

เครื่องหมาย wildcard ที่กำหนดสำหรับ export ไม่ตรงกับ symbol ใดๆ ที่สามารถ export ได้.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. กำหนด wildcard ให้ตรงกับ symbol ที่ต้องการจะ export.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

Previous Export Block Is Empty

รูปที่ 69แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                        Binder Language Listing

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000CD2.
STRPGMEXP  PGMLVL(*PRV)
ENDPGMEXP
******** Export signature: 00000000000000000000000000000000.
***ERROR Previous export block is empty.

          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

*รูปที่ 69. ไม่มีการกำหนด Symbol ไว้ใน PGMLVL(*PRV) Export Block*

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

พบ STRPGMEXP PGMLVL(*PRV) แต่ไม่มีการกำหนด symbol ไว้.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่ง ดังนี้:
   - เพิ่ม symbol ใน STRPGMEXP–ENDPGMEXP block ที่ว่างอยู่.
   - ลบ STRPGMEXP–ENDPGMEXP block ที่ว่างนั้นออกไป.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ STRPGMEXP–ENDPGMEXP block ที่ว่างอยู่จะถูกลบออกไปจาก binder language source file.

Signature Contains Variant Characters

รูปที่ 70 ในหน้า 216 แสดง binder language listing ที่แสดงข้อผิดพลาดนี้.

```
                                Binder Language Listing

            STRPGMEXP  SIGNATURE('\!cdefghijklmnop')
***ERROR Signature contains variant characters
            EXPORT     SYMBOL('Proc_2')
            ENDPGMEXP

******** Export signature: E05A838485868788899191929394959697.

          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 70. แสดง Signature Contains Variant Characters

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

Signature **มีตัวอักษรที่ไม่ได้อยู่ใน** coded character set identifiers(CCSIDs).

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. **ลบตัวอักษรตัวที่ใช้ไม่ได้ออกไป.**
2. **รันคำสั่ง** CRTSRVPGM **อีกครั้ง.**

ในกรณีนี้ ตัว\! **จะต้องถูกลบออกไป.**

SIGNATURE(*GEN) Required with LVLCHK(*NO)

รูปที่ 71 **แสดง** binder language listing **ที่แสดงข้อผิดพลาดนี้.**

```
                                Binder Language Listing

            STRPGMEXP  SIGNATURE('ABCDEFGHIJKLMNOP')  LVLCHK(*NO)
            EXPORT     SYMBOL('Proc_2')
***ERROR SIGNATURE(*GEN) required with LVLCHK(*NO)
            ENDPGMEXP

******** Export signature: C1C2C3C4C5C6C7C8C9D1D2D3D4D5D6D7.

          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

*รูปที่ 71. ถ้ามีการกำหนด LVLCHK(*NO) แล้ว Signature จะใช้งานไม่ได้.*

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

ถ้ามีการกำหนด LVLCHK(*NO) **จะต้องมีการกำหนดค่า** SIGNATURE(*GEN) **ด้วย.**

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่ง ดังนี้:
   - กำหนด SIGNATURE(*GEN)
   - กำหนด LVLCHK(*YES)
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

Signature Syntax Not Valid

รูปที่ 72แสดง binder language listing ที่แสดงข้อผิดพลาดนี้.

```
                                Binder Language Listing

             STRPGMEXP  SIGNATURE('"abcdefghijkl "')
 ***ERROR Signature syntax not valid
 ***ERROR Signature syntax not valid
 ***ERROR Syntax not valid.
 ***ERROR Syntax not valid.
             EXPORT     SYMBOL('Proc_2')
             ENDPGMEXP

            * * * * *  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  * * * * *
```

รูปที่ 72. แสดงการกำหนด Signature ที่ไม่ถูกต้อง

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

Signature มีตัวอักษรที่ไม่สามารถใช้ได้.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ลบตัวอักษรที่ใช้ไม่ได้ออกจากค่าของ signature.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในกรณีนี้ เป็นการลบเครื่องหมาย " ออกจากฟิลด์ signature.

Symbol Name Required

รูปที่ 73 ในหน้า 218 แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                                Binder Language Listing

 STRPGMEXP  PGMLVL(*CURRENT)
   EXPORT SYMBOL(A)
   EXPORT SYMBOL(')
 ***ERROR Symbol name required.
 ENDPGMEXP
 ******** Export signature: 00000000000000000000000000000C1.

           *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 73. ไม่มี Symbol ที่ถูก Export

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

ไม่พบชื่อของ symbol ที่ถูก export จากเซอร์วิสโปรแกรม.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่ง ดังนี้:
   - ลบบรรทัดที่มีข้อผิดพลาดออกจากซอร์สไฟล์ของ binder language.
   - เพิ่มชื่อของ symbol ที่จะถูก export จากเซอร์วิสโปรแกรมเข้าไป.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ บรรทัดของ EXPORT SYMBOL("") จะถูกลบออกจากซอร์สไฟล์ของ binder language.

Symbol Not Allowed as Service Program Export

รูปที่ 74แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                                Binder Language Listing

 STRPGMEXP  PGMLVL(*CURRENT)
   EXPORT SYMBOL(A)
 ***ERROR Symbol not allowed as service program export.
   EXPORT SYMBOL(D)
 ENDPGMEXP
 ******** Export signature: 00000000000000000000000000000CD4.

           *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 74. แสดงชื่อของ Symbol ไม่สามารถถูก Export จาก Service Program

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

Symbol ที่จะถูก export จากเซอร์วิสโปรแกรมนั้นไม่ได้ถูก export จากโมดูลที่ถูกรวมโดยการก็อปปี. โดยทั่วไป symbol ที่กำหนดให้ถูก export จากเซอร์วิสโปรแกรม คือ symbol ที่ต้องการถูก import โดยเซอร์วิสโปรแกรม.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่ง ดังนี้:
   - ลบ symbol ที่ผิดพลาดออกจากซอร์สไฟล์ของ binder language.
   - กำหนดโมดูลที่มี symbol ที่ต้องการ export ไว้ในพารามิเตอร์ MODULE ของคำสั่ง CRTSRVPGM.
   - เพิ่ม symbol ลงในโมดูลที่จะถูกรวมแบบ by copy และสร้างโมดูลอ็อบเจ็กต์ใหม่อีกครั้ง.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ บรรทัดของ EXPORT SYMBOL(A) จะถูกลบออกจากซอร์สไฟล์ของ binder language

Symbol Not Defined

รูปที่ 75แสดง binder language listing ที่มีข้อผิดพลาดนี้.

```
                        Binder Language Listing

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(Q)
***ERROR Symbol not defined.
ENDPGMEXP
******** Export signature: 00000000000000000000000000000CE8.


          *****  E N D  O F  B I N D E R  L A N G U A G E  L I S T I N G  *****
```

รูปที่ 75. ไม่พบ Symbol ในโมดูลที่ถูกรวมโดยการก็อปปี.

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

Symbol ที่จะถูก export จากเซอร์วิสโปรแกรม ไม่ถูกพบในโมดูลที่ถูกรวมโดยการก็อปปี.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. ทำการแก้ไขตามข้อใดข้อหนึ่ง ดังนี้:
   - ลบ symbol ที่ไม่ถูกกำหนดออกจากซอร์สไฟล์ของ binder language.

- กำหนดโมดูลที่มี symbol ที่ต้องการจะ export ในพารามิเตอร์ MODULE ของคำสั่ง CRTSRVPGM.
- เพิ่ม symbol ลงในโมดูลที่จะถูกรวมแบบ by copy และสร้างโมดูลอ็อบเจ็กต์ใหม่อีกครั้ง.

2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ในตัวอย่างนี้ บรรทัดของ EXPORT SYMBOL (Q) จะถูกลบออกจากซอร์สไฟล์ของ binder language.

Syntax Not Valid

นี่คือ ข้อความแสดงความผิดพลาดในระดับรุนแรง.

คำสั่งที่อยู่ใน source member ไม่ได้เป็นคำสั่งของ binder language.

เซอร์วิสโปรแกรมจะไม่ถูกสร้างขึ้น.

การแก้ไขที่เหมาะสม

1. แก้ไข source member ให้ถูกต้อง เพื่อให้มีแต่คำสั่งของ binder language ที่ถูกต้อง.
2. รันคำสั่ง CRTSRVPGM อีกครั้ง.

ภาคผนวก B. Exception ในโปรแกรมที่ถูก Optimize

ในสภาพแวดล้อมที่หาได้ยาก MCH3601 exception message อาจเกิดขึ้นในโปรแกรมที่คอมไพล์ในระดับ optimization เท่ากับ 30(*FULL) หรือ 40. ภาคผนวกนี้จะอธิบายตัวอย่างที่มีข้อความชนิดนี้เกิดขึ้น. โปรแกรมเดียวกันนี้อาจไม่ได้รับข้อความ Exception หมายเลข MCH3601 เมื่อถูกคอมไพล์ที่ระดับ optimization เท่ากับ 10(*NONE) หรือ 20(*BASIC). ข้อความในตัวอย่างนี้เกิดขึ้นโดยขึ้นอยู่กับวิธีการที่คอมไพเลอร์ HLL ของ ILE จัดสรรพื้นที่เก็บข้อมูลสำหรับอาร์เรย์. ดังนั้นตัวอย่างนี้อาจไม่เคยเกิดขึ้นกับภาษาที่คุณใช้.

เมื่อคุณต้องการทำ optimization ในระดับ 30(*FULL) หรือ 40 ILE พยายามที่จะพัฒนาประสิทธิภาพโดยการคำนวณอาร์เรย์ที่เป็นดัชนีอ้างอิงที่อยู่ภายนอก loop. เมื่อคุณอ้างถึงอาร์เรย์ใน loop คุณจะเข้าถึงทุกส่วนของอาร์เรย์นั้นโดยเป็นไปตามลำดับ. ประสิทธิภาพสามารถพัฒนาขึ้นได้โดยการ save แอดเดรสของอาร์เรย์ตัวสุดท้ายจากการวนซ้ำกันของ loop ก่อนหน้านี้. เพื่อที่บรรลุการพัฒนาประสิทธิภาพ ILE จะคำนวณแอดเดรสของอาร์เรย์ตัวแรกที่อยู่ภายนอก loop และเก็บค่าไว้สำหรับใช้ภายใน loop.

ดังตัวอย่างต่อไปนี้:

```
DCL ARR[1000] INTEGER;
DCL I INTEGER;

I = init_expression;   /* Assume that init_expression evaluates
                          to -1 which is then assigned to I */

/* More statements */

WHILE ( I < limit_expression )

  I = I + 1;

  /* Some statements in the while loop */

  ARR[I] = some_expression;

  /* Other statements in the while loop */

END;
```

ถ้าการอ้างอิงไปยัง ARR[init_expression] ทำให้เกิดดรรชนีของอาร์เรย์ที่ไม่ถูกต้อง ซึ่งตัวอย่างนี้ทำให้เกิดข้อความ Exception หมายเลข MCH3601. เนื่องจาก ILE พยายามที่จะคำนวณแอดเดรสของอาร์เรย์ตัวแรก ก่อนที่จะเข้าไปใน WHILE loop.

ถ้าคุณได้รับ MCH3601 ที่ระดับ optimization เท่ากับ 30(*FULL) หรือ 40 ให้คุณมองหาเหตุการณ์เหล่านี้:

1. คุณมี loop หนึ่งที่เพิ่มค่าของตัวแปรก่อนที่มันจะใช้ตัวแปรเป็นดรรชนีของอาร์เรย์.
2. ค่าเริ่มต้นของตัวแปรที่เป็นดรรชนีบนทางเข้าของ loop มีค่าเป็นลบ.

3. ตัวอ้างอิงถึงอาร์เรย์มีค่าเริ่มต้นของตัวแปรที่ไม่ถูกต้อง.

แม้ว่าจะมีเหตุการณ์เช่นนี้เกิดขึ้น แต่คุณยังสามารถใช้ระดับ optimization ที่ 30(*FULL) หรือ 40 ต่อไปได้โดยการทำตามวิธีการเหล่านี้:

1. ย้ายส่วนของโปรแกรมที่เพิ่มค่าของตัวแปรไปยังส่วนท้ายของ loop.
2. เปลี่ยนตัวอ้างอิงไปยังตัวแปรให้เป็นไปตามที่ต้องการ.

ตัวอย่างก่อนหน้านี้จะถูกเปลี่ยนไปดังนี้:

```
I = init_expression + 1;

WHILE ( I < limit_expression + 1 )

  ARR[I] = some_expression;

  I = I + 1;

END;
```

ถ้าการเปลี่ยนแปลงนี้ไม่สามารถทำได้ให้ลดระดับของ optimization ลงจาก 30(*FULL) หรือ 40 ไปเป็น 20(*BASIC) หรือ 10(*NONE).

ภาคผนวก C. คำสั่ง CL ที่ใช้กับอ็อบเจ็กต์ ILE

ตารางต่อไปนี้แสดงถึงคำสั่ง CL ที่สามารถใช้กับอ็อบเจ็กต์ ILE แต่ละชนิด.

คำสั่ง CL ที่ใช้กับโมดูล

ตารางที่ 13. คำสั่ง CL ที่ใช้กับโมดูล

คำสั่ง	รายละเอียด
CHGMOD	เปลี่ยนแปลงโมดูล
CRTCBLMOD	สร้างโมดูลภาษาโคบอล
CRTCLMOD	สร้างโมดูลภาษา CL
CRTCMOD	สร้างโมดูลภาษา C
CRTCPPMOD	สร้างโมดูลภาษา C++
CRTRPGMOD	สร้างโมดูลภาษาอาร์พีจี
DLTMOD	ลบโมดูล
DSPMOD	แสดงโมดูล
RTVBNDSRC	เรียกค้น Binder Source
WRKMOD	ทำงานกับโมดูล

คำสั่ง CL ที่ใช้กับโปรแกรมอ็อบเจ็กต์

ตารางที่ 14. คำสั่ง CL ที่ใช้กับโปรแกรมอ็อบเจ็กต์

คำสั่ง	รายละเอียด
CHGPGM	เปลี่ยนแปลงโปรแกรม
CRTBNDC	สร้างโปรแกรมภาษาซีที่ถูกเชื่อมโยง
CRTBNDCBL	สร้างโปรแกรมภาษาโคบอลที่ถูกเชื่อมโยง
CRTBNDCL	สร้างโปรแกรมภาษา CL ที่ถูกเชื่อมโยง
CRTBNDCPP	สร้างโปรแกรมภาษาซีพลัสพลัสที่ถูกเชื่อมโยง
CRTBNDRPG	สร้างโปรแกรมภาษาอาร์พีจีที่ถูกเชื่อมโยง
CRTPGM	สร้างโปรแกรม
DLTPGM	ลบโปรแกรม

ตารางที่ 14. คำสั่ง CL ที่ใช้กับโปรแกรมอ็อบเจ็กต์ (ต่อ)

DSPPGM	แสดงผลโปรแกรม
DSPPGMREF	แสดงผลการอ้างอิงโปรแกรม
UPDPGM	อัพเดตโปรแกรม
WRKPGM	ทำงานกับโปรแกรม

คำสั่ง CL ที่ใช้กับเซอร์วิสโปรแกรม

ตารางที่ 15. คำสั่ง CL ที่ใช้กับเซอร์วิสโปรแกรม

คำสั่ง	**รายละเอียด**
CHGSRVPGM	เปลี่ยนแปลงเซอร์วิสโปรแกรม
CRTSRVPGM	สร้างเซอร์วิสโปรแกรม
DLTSRVPGM	ลบเซอร์วิสโปรแกรม
DSPSRVPGM	แสดงผลเซอร์วิสโปรแกรม
RTVBNDSRC	เรียกค้น Binder Source
UPDSRVPGM	อัพเดตเซอร์วิสโปรแกรม
WRKSRVPGM	ทำงานกับเซอร์วิสโปรแกรม

คำสั่ง CL ที่ใช้กับ Binding Directory

ตารางที่ 16. คำสั่ง CL ที่ใช้กับ Binding Directory

คำสั่ง	**รายละเอียด**
ADDBNDDIRE	เพิ่มรายการของ Binding Directory
CRTBNDDIR	สร้าง Binding Directory
DLTBNDDIR	ลบ Binding Directory
DSPBNDDIR	แสดง Binding Directory
RMVBNDDIRE	ลบรายการของ Binding Directory
WRKBNDDIR	ทำงานกับ Binding Directory
WRKBNDDIRE	ทำงานกับรายการของ Binding Directory

คำสั่ง CL ที่ใช้กับ Structured Query Language

ตารางที่ 17. คำสั่ง CL ที่ใช้กับ Structured Query Language

คำสั่ง	รายละเอียด
CRTSQLCI	สร้าง Structured Query Language ของอ็อบเจ็กต์ ILE C
CRTSQLCBLI	สร้าง Structured Query Language ของอ็อบเจ็กต์ ILE COBOL
CRTSQLRPGI	สร้าง Structured Query Language ของอ็อบเจ็กต์ ILE RPG

คำสั่ง CL ที่ใช้กับ CICS

ตารางที่ 18. คำสั่ง CL ที่ใช้กับ CICS

คำสั่ง	รายละเอียด
CRTCICSC	สร้าง CICS® ของอ็อบเจ็กต์ ILE C
CRTCICSCBL	สร้าง CICS ของโปรแกรม COBOL

คำสั่ง CL ที่ใช้กับซอร์สดีบักเกอร์

ตารางที่ 19. คำสั่ง CL ที่ใช้กับซอร์สดีบักเกอร์

คำสั่ง	รายละเอียด
DSPMODSRC	แสดงผลของโมดูลซอร์ส
ENDDBG	สิ้นสุดการดีบัก
STRDBG	เริ่มการดีบัก

คำสั่ง CL ที่ใช้ในการแก้ไข Binder Language Source File

ตารางที่ 20. คำสั่ง CL ที่ใช้ในการแก้ไข Binder Language Source

คำสั่ง	รายละเอียด
EDTF	แก้ไขไฟล์
STRPDM	เริ่มการทำงานของ Programming Development Manager
STRSEU	เริ่มการทำงานของ Source Entry Utility
หมายเหตุ: คำสั่งที่ไม่สามารถรันได้ต่อไปนี้ สามารถใส่ลงใน binder language source file ได้:	
ENDPGMEXP	จบการทำงานของ Program Export

ตารางที่ 20. คำสั่ง CL ที่ใช้ในการแก้ไข Binder Language Source (ต่อ)

EXPORT	นำออก

ภาคผนวก D. ประกาศ

ข้อมูลนี้ถูกพัฒนาขึ้นสำหรับผลิตภัณฑ์และบริการที่เสนอขายในประเทศไทย.

ไอบีเอ็มอาจไม่เสนอขายผลิตภัณฑ์ บริการ หรือ คุณลักษณะต่างๆ ที่กล่าวถึงในเอกสารนี้ ในประเทศอื่น. ปรึกษาตัวแทนไอบีเอ็มในท้องที่นั้นๆ ของคุณสำหรับข้อมูลเกี่ยวกับผลิตภัณฑ์และบริการที่เสนอขายในท้องที่ของคุณ. การอ้างถึงผลิตภัณฑ์ โปรแกรม หรือบริการ ของไอบีเอ็มไม่ได้หมายความว่าต้องเฉพาะผลิตภัณฑ์ โปรแกรม หรือบริการ ที่เป็นของไอบีเอ็มเท่านั้นที่จะถูกใช้. ผลิตภัณฑ์ โปรแกรม หรือบริการ อื่นที่เทียบเท่าและไม่ละเมิดลิขสิทธิ์ของไอบีเอ็มก็สามารถใช้ได้. อย่างไรก็ตามเป็นความรับผิดชอบของผู้ใช้ที่จะประเมินและตรวจสอบ ผลิตภัณฑ์ โปรแกรม หรือบริการที่ไม่ใช่ของไอบีเอ็ม.

ไอบีเอ็มอาจมีสิทธิบัตรหรือเอกสารการขอสิทธิบัตร ที่ครอบคลุมสิ่งที่อธิบายในเอกสารนี้. การตกแต่งเอกสารใหม่ไม่ได้ทำให้คุณได้สิทธิบัตร. คุณสามารถส่งการสอบถามเกี่ยวกับสิทธิในการใช้สิทธิบัตรไปที่:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594-1785
U.S.A.

สำหรับการสอบถามเกี่ยวกับสิทธิในการใช้สิทธิบัตรในเรื่องข้อมูล double-byte (DBCS) ให้ติดต่อ IBM Intellectual Property Department ในประเทศของคุณหรือส่งคำถามโดยการเขียนไปที่:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

ย่อหน้าต่อไปนี้ ไม่ใช้กับประเทศสหราชอาณาจักรหรือประเทศอื่น ที่ข้อกำหนดไม่สอดคล้องกับกฎหมายท้องถิ่น: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. บางรัฐไม่อนุญาตให้มีแสดงคำพูดเชิงปฏิเสธหรือกล่าวถึง การรับประกันโดยนัยใน transaction ดังนั้นคำพูดนี้อาจไม่ใช้กับคุณ.

ข้อมูลนี้อาจมีความไม่ถูกต้องทางเทคนิคหรือความผิดพลาดในการเรียงพิมพ์. โดยจะมีการเปลี่ยนแปลงข้อมูลในนี้เป็นระยะๆ ซึ่งจะรวบรวมไว้ในการตีพิมพ์ครั้งใหม่. ไอบีเอ็มอาจทำการปรับปรุงและ/หรือ เปลี่ยนแปลงในผลิตภัณฑ์ และ/หรือโปรแกรมที่อธิบายในสิ่งตีพิมพ์นี้โดยไม่แจ้งให้ทราบ.

การอ้างถึงเว็บไซต์ที่ไม่ใช่ของไอบีเอ็มนั้นถูกจัดหามาเพื่อความสะดวกเท่านั้น ไม่ได้มีการรับรองเว็บไซต์เหล่านั้น. ส่วนเนื้อหาในเว็บไซต์เหล่านั้นไม่ใช่เนื้อหา สำหรับผลิตภัณฑ์ของไอบีเอ็ม และการใช้เว็บไซต์เหล่านั้น เป็นความเสี่ยงของตัวคุณเอง.

IBM อาจใช้หรือเผยแพร่ข้อมูลใดๆ ที่คุณให้ไว้ในทางที่ไอบีเอ็มเชื่อว่าเหมาะสม โดยไม่มีข้อผูกมัดใดๆ กับคุณ.

สำหรับผู้ที่มีไลเซนส์ของโปรแกรมนี้ที่ต้องการมีข้อมูลเกี่ยวกับมันสำหรับจุดประสงค์ให้ทำงานได้: (i) การแลกเปลี่ยนข้อมูลระหว่างโปรแกรมที่ถูกสร้างขึ้นอย่างเป็นอิสระและโปรแกรมอื่น (รวมอันนี้) และ (ii) การใช้ข้อมูลร่วมกันที่ซึ่งมีการแลกเปลี่ยน ควรติดต่อ:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

ข้อมูลดังกล่าวอาจใช้ประโยชน์ได้ขึ้นอยู่กับช่วงเวลาและเงื่อนไขที่เหมาะสม รวมทั้งการจ่ายค่าธรรมเนียม (ในบางกรณี).

โปรแกรมไลเซนส์ที่อธิบายในข้อมูลนี้และ ปัจจัยที่มีไลเซนส์ทั้งหมดถูกจัดหามาโดยบริษัท ไอบีเอ็ม ภายใต้ IBM Customer Agreement, IBM International Program Licensed Agreement, หรือข้อตกลงอื่นที่เทียบเท่า.

ข้อมูลนี้มีตัวอย่างข้อมูลและรายงานที่ใช้ในการดำเนินธุรกิจ ประจำวัน. เพื่อแสดงตัวอย่างให้สมบูรณ์ที่สุดเท่าที่จะเป็นไปได้, ตัวอย่างจึงมี ชื่อบุคคล, บริษัท, ตราสินค้า, และผลิตภัณฑ์. ชื่อเหล่านี้ทั้งหมด เป็นชื่อสมมติ อีกทั้งการที่ชื่อและที่อยู่ซึ่งใช้คล้ายกับกิจการทางธุรกิจจริงนั้น เป็นเหตุบังเอิญทั้งสิ้น.

COPYRIGHT LICENSE:

ข้อมูลบรรจุตัวอย่างของโปรแกรมแอ็พพลิเคชันในภาษาต้นฉบับ ซึ่งแสดงเทคนิคในการเขียนโปรแกรมบนหลายๆ แพล็ตฟอร์ม. คุณอาจคัดลอก ดัดแปลง และกระจายโปรแกรมตัวอย่างเหล่านี้ในรูปแบบใดๆโดยไม่ต้องจ่ายเงินแก่ไอบีเอ็ม สำหรับจุดประสงค์ในการพัฒนา การใช้งาน การตลาดหรือการกระจายแอ็พพลิเคชันโปรแกรม ที่สอดคล้องกับ application programming interface สำหรับแพล็ตฟอร์มปฏิบัติการที่ซึ่งโปรแกรมตัวอย่างถูกเขียน. ตัวอย่างเหล่านี้ไม่ได้ผ่านการทดสอบภายใต้ทุกสถานการณ์. ดังนั้นไอบีเอ็มไม่สามารถรับประกันหรือ กล่าวถึงความเชื่อถือได้, ความสามารถในการบริการ, หรือการทำงาน ของโปรแกรมเหล่านี้. คุณอาจคัดลอก ดัดแปลง และกระจายโปรแกรมตัวอย่างเหล่านี้ในรูปแบบใดๆ โดยไม่ต้องจ่ายเงินแก่ไอบีเอ็ม สำหรับจุดประสงค์ในการพัฒนา การใช้งาน การตลาดหรือการกระจายแอ็พพลิเคชันโปรแกรม ที่สอดคล้องกับ application programming interface ของไอบีเอ็ม.

แต่ละสำเนาหรือบางส่วนของโปรแกรมตัวอย่าง หรืองานใดๆ ที่มาจากโปรแกรมเหล่านี้ ต้องมีข้อความแสดงลิขสิทธิ์ ดังนี้:

Programming Interface Information

จุดประสงค์ของคู่มือเล่มนี้คือเพื่อช่วยคุณในการใช้ Integrated Language Environment. ส่วนหนังสือ General-Use Programming Interface และ Associated Guidance Information ได้จัดเตรียมไว้ให้แล้วโดย OS/400.

General-Use programming interfaces จะช่วยให้ลูกค้าสามารถเขียนโปรแกรมที่ใช้เซอร์วิสของ OS/400.

เครื่องหมายการค้า

คำต่อไปนี้ เป็นเครื่องหมายการค้าของ International Business Machines Corporation ในประเทศสหรัฐ, หรือในประเทศอื่น, หรือทั้งสองกรณี:

400
AS/400
CICS
IBM
iSeries
OS/2
OS/400POWER4
WebSphere

โลโก้ Microsoft, Windows, Windows NT, และ Windows เป็นเครื่องหมายการค้าของ Microsoft Corporation ในสหรัฐ, ประเทศอื่น, หรือทั้งสองกรณี.

Java และเครื่องหมายการค้าที่มีคำว่า Java เป็นเครื่องหมายการค้าของ บริษัท Sun Microsystems Inc. ในประเทศสหรัฐ, หรือในประเทศอื่น, หรือทั้งสองกรณี.

UNIX เป็นเครื่องหมายการค้าจดทะเบียนของ The Open Group ในสหรัฐและ ประเทศอื่นๆ.

ชื่ออื่นๆ ของบริษัท, ผลิตภัณฑ์, และบริการ อาจเป็นเครื่องหมายการค้าหรือเครื่องหมายการบริการของผู้อื่น.

รายชื่อเอกสารอ้างอิง

สำหรับข้อมูลเพิ่มเติมในหัวข้อที่เกี่ยวข้องกับสภาพแวดล้อมของ ILE บนเซิร์ฟเวอร์ iSeries, สามารถอ้างอิงได้จากเอกสารเหล่านี้:

- หัวข้อ การสำรองข้อมูลและการกู้คืน ของศูนย์ข้อมูลจะให้ข้อมูลเกี่ยวกับการวางแผนกลยุทธ์การสำรองข้อมูลและการกู้คืน, สื่อประเภทต่างๆ ที่มีเพื่อบันทึกและ เรียกคืนข้อมูลระบบ, รวมทั้งรายละเอียดวิธีการบันทึกความเปลี่ยนแปลงที่เกิดขึ้น กับไฟล์ฐานข้อมูลโดยใช้การทำเจอร์นัล และวิธีการที่ข้อมูลถูกใช้สำหรับ การกู้คืนระบบ. หนังสือเล่มนี้อธิบายถึงการวางแผนและการติดตั้ง Auxiliary Storage Pools (ASP), การทำ Mirrored Protection และการ Checksum. นอกจากนี้ ยังอธิบายถึงการติดตั้งระบบจากที่สำรองข้อมูล (Backup) ไว้.

- CL Programming จะมีการอภิปรายอย่างกว้างขวาง เกี่ยวกับหัวข้อการโปรแกรมมิง, รวมทั้งการอภิปรายทั่วไปเกี่ยวกับ อ็อบเจ็กต์และไลบรารี, CL โปรแกรมมิง, การ control flow และการสื่อสารระหว่างโปรแกรม, การทำงานกับอ็อบเจ็กต์ในโปรแกรม CL, และการสร้างโปรแกรม CL.นอกจากนี้หัวข้ออื่นประกอบด้วย แมสเสจที่มีการกำหนดไว้แล้ว (Predefined Message) และการจัดการเกี่ยวกับแมสเสจ , การกำหนดและสร้างคำสั่งที่ผู้ใช้กำหนดขึ้นเอง และเมนู, การทดสอบแอ็พพลิเคชัน รวมถึงดีบักโหมด, จุดพัก.

- Communications Management ให้ข้อมูลเกี่ยวกับการจัดการระบบงานในสภาวะแวดล้อมของการสื่อสาร, สถานะการสื่อสาร, การ trace และวินิจฉัยปัญหาของการสื่อสาร, การจัดการข้อผิดพลาดและการกู้คืน, ประสิทธิภาพการทำงาน, และความเร็วของสายเฉพาะ และ ข้อมูลหน่วยเก็บของระบบย่อย.

- ICF Programming ให้ข้อมูลที่จำเป็นในการเขียนแอ็พพลิเคชันโปรแกรม ที่ใช้ในการสื่อสาร และฟังก์ชันการสื่อสารระหว่างระบบของ OS/400 (OS/400-ICF). หนังสือเล่มนี้ยังมีข้อมูลเกี่ยวกับคีย์เวิร์ดของ Data Description Specification (DDS), ฟอร์แมตที่ระบบกำหนด, ค่าส่งคืน, การสนับสนุนการโอนย้ายไฟล์, และโปรแกรมตัวอย่าง.

- WebSphere Development Studio ILE C/C++ Programmer's Guide อธิบายถึงวิธีการสร้าง, คอมไพล์, แก้ไข, และรัน ILE C และโปรแกรม ILE C++. คู่มือแนะนำจะบอกข้อมูลเกี่ยวกับ ILE และคุณลักษณะของโปรแกรมมิงบน OS/400 ; iSeries ระบบไฟล์, อุปกรณ์, และคุณลักษณะต่างๆ; การปฏิบัติการของ I/O ; localization; ประสิทธิภาพการทำงานโปรแกรม; และข้อมูลชนิดของรันไทม์ C++ (RTTI).

- WebSphere Development Studio C/C++ Language Reference อธิบายถึงโครงร่างของภาษาที่สอดคล้องกับ *ภาษาโปรแกรม - มาตรฐาน C* (ISO/IEC 9899:1990) และ *ภาษาโปรแกรม - มาตรฐาน C++* (ISO/IEC 14882:1998).

- WebSphere Development Studio ILE C/C++ Compiler Reference ประกอบด้วยข้อมูลอ้างอิงสำหรับคอมไพเลอร์ ILE C/C++ , ที่รวมถึงข้อความ preprocessor, แมโคร, pragmas, บรรทัดรับคำสั่งที่ใช้สำหรับ iSeries และสภาพแวดล้อม QShell , และข้อควรพิจารณาของ I/O .

- ILE C/C++ Run-time Library Reference ให้ข้อมูลอ้างอิงที่เกี่ยวกับ ILE C/C++, ฟังก์ชันไลบรารีของรันไทม์, ที่รวมไฟล์, และข้อควรพิจารณาของรันไทม์.

- WebSphere Development Studio: ILE COBOL Programmer's Guide อธิบายวิธีการเขียน, คอมไพล์, เชื่อมโยง, รัน, แก้ไข, และรักษาไว้ ILE COBOL โปรแกรม บน ระบบ AS/400. นอกจากนี้ยังมีข้อมูลเกี่ยวกับการเรียกโปรแกรม ILE COBOL อื่น และโปรแกรมที่ไม่ใช่ ILE COBOL, การใช้ข้อมูลร่วมกันกับโปรแกรมอื่น, การใช้พอยน์เตอร์และการจัดการ

Exception. หนังสือเล่มนี้ยังอธิบายเกี่ยวกับการอินพุต/เอาท์พุตบนอุปกรณ์ภายนอกที่ต่ออยู่, ไฟล์ฐานข้อมูล, ไฟล์หน้าจอ และ ไฟล์ ICF.

- WebSphere Development Studio: ILE COBOL Reference  อธิบายถึงภาษาโปรแกรม ILE COBOL . และโครงสร้างของโปรแกรม ILE COBOL และโครงสร้างของซอร์สโปรแกรม ILE COBOL. หนังสือเล่มนี้ยังอธิบายถึง Identification Division paragraphs, Environment Division clauses, Data Division paragraphs, Procedure Division statement และ Compiler-Directing statements.
- WebSphere Development Studio: ILE RPG Programmer's Guide  เป็นคู่มือแนะนำสำหรับการใช้ภาษาโปรแกรม RPG IV, ซึ่งเป็นการนำไปปฏิบัติ ของ ILE RPG ใน Integrated Language Environment (ILE) บน iSeries เซิร์ฟเวอร์. นอกจากนี้ยังกล่าวถึงการสร้างและการรันโปรแกรมพร้อมข้อพิจารณาสำหรับการเรียกโพรซีเดอร์ และการโปรแกรมระหว่างภาษา. คู่มือนี้ยังครอบคลุมถึงการดีบัก และการจัดการ exception และอธิบายวิธีการใช้ไฟล์และอุปกรณ์ของ AS/400 ในโปรแกรม RPG.ในส่วนภาคผนวกจะประกอบด้วยการโอนย้ายระบบ (Migration) ไปยัง RPG IV และรายชื่อคอมไพเลอร์ตัวอย่าง. หนังสือเล่มนี้เหมาะสำหรับผู้อ่านที่มีความเข้าใจขั้นพื้นฐานเกี่ยวกับการโพรเซสข้อมูลของภาษาอาร์พีจี.

- คู่มือ WebSphere Development Studio ILE RPG  ประกอบด้วยข้อมูลที่จำเป็นในการเขียนโปรแกรมสำหรับเซิร์ฟเวอร์ iSeries โดยใช้ภาษาโปรแกรม RPG IV. คู่มือเล่มนี้อธิบายตำแหน่งต่อตำแหน่ง, คีย์เวิร์ดต่อคีย์เวิร์ด, และรายละเอียดเกี่ยวกับโค้ดและฟังก์ชันในตัว (Build-In) ด้วย. คู่มือนี้ยังประกอบด้วยข้อมูลเกี่ยวกับวงจร RPG logic, อาร์เรย์และตาราง, ฟังก์ชันในการ edit, และตัวบ่งชี้.

- Intrasystem Communications Programming  ให้ข้อมูลเกี่ยวกับการสื่อสารแบบโต้ตอบระหว่าง สองแอ็พพลิเคชันโปรแกรมบนเซิร์ฟเวอร์ iSeries ตัวเดียวกัน. คู่มือนี้อธิบายถึงการติดต่อสื่อสารที่สามารถโค้ดไว้ในโปรแกรมซึ่งสนับสนุน Intrasystem Communication กับอีกโปรแกรมหนึ่ง. นอกจากนี้ยังมีข้อมูลเกี่ยวกับการพัฒนาโปรแกรมแอ็พพลิเคชัน Intrasystem Communication ที่ใช้ OS/400 Intersystem Communications Function (OS/400-ICF).

- iSeries Security Reference  บอกถึงวิธีสนับสนุนความปลอดภัยของระบบ สามารถนำมาใช้ป้องกันระบบและข้อมูลจากการถูกใช้โดยบุคคลที่ไม่มีการให้สิทธิอย่างเหมาะสม, ปกป้องข้อมูลจากความเสียหาย หรือการถูกทำลายโดยตั้งใจ หรือไม่ตั้งใจ, เก็บรักษาข้อมูลที่เป็นความลับให้ทันสมัยอยู่ตลอด, และตั้งค่าความปลอดภัยบนระบบ.
- ในหัวข้อ Work Management, ซึ่งอยู่ในหมวด **Systems Management** ของ iSeries Information Center, ให้ข้อมูลเกี่ยวกับวิธีการสร้างและเปลี่ยนสภาพแวดล้อมการจัดการระบบงาน. หัวข้ออื่นจะประกอบด้วยการปรับจูนระบบ, การเก็บรวบรวมข้อมูลด้าน performance รวมถึงข้อมูลบนฟอร์แมตเร็กคอร์ด, การทำงานกับ system value เพื่อควบคุมหรือเปลี่ยนแปลงการทำงานทั้งหมดของระบบ และวิธีการเก็บข้อมูลเพื่อตรวจว่าใครกำลังใช้ระบบ และรีซอร์สอะไรกำลังถูกใช้อยู่.

ดัชนี

อักขระพิเศษ

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

ก

ช

ซ

ด

ต

ป

ส

ห

ความคิดเห็นจากผู้อ่าน — เราต้องการฟังความคิดเห็นจากคุณ

iSeries
แนวคิดเรื่อง ILE
เวอร์ชัน 5 รีลีส 3

หมายเลขสิ่งตีพิมพ์ SC09-3449-03

กรุณาตอบแบบสอบถามข้อคิดเห็นนี้ เพื่อช่วยให้ไอบีเอ็มตอบสนองต่อความต้องการของคุณได้ดียิ่งขึ้น

โดยรวมแล้ว, คุณพึงพอใจเพียงไรกับข้อมูลในหนังสือเล่มนี้

	พึงพอใจมาก	พึงพอใจ	เฉยๆ	ไม่พอใจ	ไม่พอใจมาก
ความพึงพอใจโดยรวม	☐	☐	☐	☐	☐

คุณพึงพอใจเพียงไรกับข้อมูลในหนังสือเล่มนี้

	พึงพอใจมาก	พึงพอใจ	เฉยๆ	ไม่พอใจ	ไม่พอใจมาก
ความถูกต้อง	☐	☐	☐	☐	☐
ความสมบูรณ์	☐	☐	☐	☐	☐
ความง่ายในการค้นหา	☐	☐	☐	☐	☐
ความง่ายในการเข้าใจ	☐	☐	☐	☐	☐
การจัดเรียงลำดับ	☐	☐	☐	☐	☐
การมีส่วนช่วยในงานของคุณ	☐	☐	☐	☐	☐

โปรดแนะนำเราในการทำหนังสือเล่มนี้ให้ดีขึ้น:

_____

_____

_____

ขอขอบคุณสำหรับความคิดเห็นของคุณ คุณจะอนุญาตให้เราติดต่อคุณได้หรือไม่ ? ☐ ได้ ☐ ไม่ได้

เมื่อคุณส่งความคิดเห็นให้กับไอบีเอ็ม, เท่ากับว่าคุณได้ให้สิทธิ์ต่อไอบีเอ็มในการใช้หรือส่งต่อความคิดเห็นของคุณด้วยวิธีการใดๆ ที่ไอบีเอ็มคิดว่าเหมาะสมโดยไม่ต้องมีพันธะผูกพันต่อคุณ.

_____ _____

ชื่อ ที่อยู่

_____ _____

บริษัทหรือองค์กร

_____ _____

หมายเลขโทรศัพท์

(โปรดส่งข้อมูลนี้กลับมายังศูนย์ลูกค้าสัมพันธ์, บริษัท ไอบีเอ็ม ประเทศไทย จำกัด, โทรสาร: 0-2273-0188 หรือตามที่อยู่ในหน้าถัดไป)

ความคิดเห็นจากผู้อ่าน — เราต้องการฟังความคิดเห็นจากคุณ

SC09-3449-03

IBM

ตัดหรือพับตามเส้นนี้

พับและปิดผนึก **กรุณาหลีกเลี่ยงการเย็บลวด** พับและปิดผนึก

กรุณาติด
ตรา
ไปรษณียากร
ที่นี่

ศูนย์ลูกค้าสัมพันธ์
บริษัท ไอบีเอ็ม ประเทศไทย จำกัด
388 ถนนพหลโยธิน พญาไท
กรุงเทพฯ
10400

พับและปิดผนึก **กรุณาหลีกเลี่ยงการเย็บลวด** พับและปิดผนึก

ตัดหรือพับตามเส้นนี้

พิมพ์ในสหรัฐอเมริกา

SC09-3449-03