IBM

eserver

iSeries

ILE 개념

버전 5 릴리스 3

SA30-0240-07

IBM

@server

iSeries

ILE 개념

버전 5 릴리스 3

SA30-0240-07

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 231 페이지의 부록 D『주의사항』정보를 반드시 읽으십시오.

제 8 판(2004년 4월)

| 이 개정판은 새 개정판에서 달리 명시하지 않는 한, IBM Operating System/400(프로그램 번호 5722-SS1), 버전 5, 릴리스 3 수정 0과 모든 후속 릴리스 및 수정판에 적용됩니다. 이 버전은 모든 축약 명령어 세트 컴퓨터(RISC) 모델에서 실행하지 않으며 또한 CISC 모델에서도 실행하지 않습니다.

| 본 제판은 SA30-0240-06을 대체합니다.

목차

ILE 개념(SA30-0240)

이 책에서는 OS/400® 사용권 프로그램의 Integrated Language Environment®(ILE) 구조에 관한 개념과 용어를 설명합니다. 본문에서 다루는 주제로는 모듈 작성, 바인딩, 프로그램의 수행 및 디버깅, 예외 처리가 있습니다.

이 책에서 설명되는 개념은 모든 ILE 언어에 해당하는 것입니다. 각 ILE 언어는 ILE 구조를 다소 다르게 구현할 수 있습니다. 각 언어가 여기에서 설명되는 개념들을 어떻게 구현시키는지 알아보려면 특수 ILE 언어에 대한 프로그래머 안내서를 참조하십시오.

또한 이 책에서는 모든 ILE 언어에 직접적으로 관련되는 OS/400 기능도 설명합니다. 특히 바인딩, 메세지 처리 및 디버깅에 관한 일반적인 정보가 설명됩니다.

이 책은 기존 OS/400 언어에서 ILE 언어로 마이그레이션하는 작업에 관해서는 설명하지 않습니다. 그와 같은 내용은 각 ILE 고급 언어(HLL) 프로그래머 안내서에 포함되어 있습니다.

이 책의 사용자

다음에 해당할 경우 이 책을 읽어야 합니다.

- 어플리케이션이나 소프트웨어 툴을 개발하는 소프트웨어 업체인 경우
- iSeries 서버에서 혼합 언어 어플리케이션을 개발해 본 경험이 있는 경우
- iSeries 서버에는 익숙하지 않지만 다른 시스템에서의 어플리케이션 프로그래밍 경험이 있는 경우
- 프로그램이 공통 프로시듀어를 공유하고 해당 프로시듀어를 갱신하거나 확장할 때 그것을 사용하는 프로그램을 재작성해야 하는 경우

주로 한 가지 언어로 프로그램을 작성하는 OS/400 어플리케이션 프로그래머인 경우에는 ILE와 그 장점에 대한 일반적인 이해를 위해 먼저 이 책의 제 4 장까지 읽어야 합니다. 그리고 나면 해당 ILE 언어용 안내서만으로도 어플리케이션을 충분히 개발할 수 있습니다.

요구사항 및 관련 정보

iSeries 기술 정보를 찾기 위한 시작점으로 iSeries Information Center를 사용하십시오.

다음 두가지 방법으로 Information Center에 액세스할 수 있습니다.

- 다음 웹 사이트에서:

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

- *iSeries Information Center*, SK3T-4091-04 CD-ROM에서: 이 CD-ROM은 새 iSeries 하드웨어 또는 IBM Operating System/400 소프트웨어 업그레이드 주문과 함께 제공됩니다. 또한 다음 IBM® Publications Center에서 CD-ROM을 주문할 수도 있습니다.

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

iSeries Information Center에는 소프트웨어 및 하드웨어 설치, Linux, WebSphere®, Java™, 고가용성, 데이터베이스, 논리 파티션, CL 명령 및 시스템 어플리케이션 프로그래밍 인터페이스(API)와 같은 신규 및 갱신된 iSeries 정보가 있습니다. 또한, iSeries 하드웨어 및 소프트웨어의 계획, 문제 해결 및 구성을 돕기 위한 어드바이저와 파인더를 제공합니다.

신규 하드웨어를 주문할 때마다 *iSeries 설정 및 조작 CD-ROM*, SK3T-4098-02를 수신합니다. 이 CD-ROM에는 IBM **@server** Windows용 IBM e(logo)server iSeries Access 및 EZ-Setup 마법사가 있습니다. iSeries Access 제품군은 PC를 iSeries™ 서버와 연결하기 위한 강력한 클라이언트 및 서버 기능을 제공합니다. EZ-Setup 마법사는 많은 iSeries 설정 타스크를 자동화합니다.

기타 관련 정보는 235 페이지의 『참고 서적』을 참조하십시오.

사용자의 의견을 보내는 방법

고객 여러분의 의견은 정확하고 우수한 품질의 정보를 제공하는 데 있어서 매우 중요합니다. 이 책이나 기타 다른 iSeries 출간물에 관해 의견이 있으시면 이 책의 맨 뒤에 있는 IBM 한글 지원에 관한 설문 양식을 작성해 주십시오.

- 우편을 이용하는 경우 IBM 한글 지원에 관한 설문 양식을 사용하십시오.
- FAX를 이용하는 경우 아래 번호를 사용하십시오.
  - 02-3787-0123
- 전자 우편을 이용하는 경우 아래 네트워크 ID를 사용하십시오.
  - 책에 관한 의견서:

    ibmkspoe@kr.ibm.com
  - iSeries Information Center에 관한 의견서:

    ibmkspoe@kr.ibm.com

보내실 때는 반드시 아래 항목을 포함시켜 주십시오.

- 책의 이름이나 iSeries Information Center 주제
- 책 번호
- 해당 페이지 또는 책의 주제

제 1 장 통합 언어 환경 소개

이 장에서는 통합 언어 환경(ILE) 모델을 정의하고, ILE의 이점을 설명하며 이전 프로그램 모델에 비하여 ILE의 개선된 사항을 설명합니다.

가능한 한 모든 내용을 RPG 또는 COBOL 프로그래머 관점에서 표현했으며, 기존 iSeries 서버 피처의 용어를 사용하여 설명했습니다.

ILE의 개념

ILE는 iSeries 시스템에서 프로그램 개발을 향상시키기 위해 설계된 툴 및 연관 시스템 지원 세트입니다.

이러한 새로운 모델의 기능은 단지 새로운 ILE 계열의 컴파일러에 의해 생성된 프로그램으로만 실행될 수 있습니다. 관련 제품군으로는 ILE RPG, ILE COBOL, ILE C, ILE C++ 및 ILE CL이 있습니다.

ILE의 이점

ILE는 이전 프로그램 모델에 비하여 많은 이점을 갖고 있습니다. 그러한 이점으로는 바인딩, 모듈화, 재사용할 수 있는 구성요소, 공통 실행시 서비스, 공존 및 소스 디버거가 있습니다. 또한 자원에 대한 보다 나은 제어, 언어 상호작용에 대한 보다 나은 제어, 보다 나은 코드 최적화, C를 위한 보다 나은 환경 및 미래를 위한 기초도 그러한 이점에 포함시킬 수 있습니다.

바인딩

바인딩의 이점은 호출 프로그램과 연관된 총 경비를 절감하는 데 도움이 된다는 점입니다. 모듈을 함께 바인딩하면 호출 속도가 빨라집니다. 이전의 호출 메커니즘을 계속 사용할 수 있으나 더 빠른 대체 방안도 있습니다. 두 호출 유형간의 차이점을 식별하기 위해 이전 방법을 동적 또는 외부 프로그램 호출이라 하며 ILE 방법은 정적 또는 바인드 프로시듀어 호출이라 합니다.

바인딩 기능은 호출 성능의 향상과 함께 고도의 모듈 방식에서 어플리케이션 개발을 보다 더 실제적으로 만듭니다. ILE 컴파일러는 실행할 수 있는 프로그램을 만드는 것이 아니라 하나의 실행가능 단위, 즉 프로그램 오브젝트(*PGM)를 형성하기 위해 다른 모듈과 바인드될 수 있는 모듈 오브젝트(*MODULE)를 만드는 것입니다.

마치 COBOL 프로그램에서 RPG 프로그램을 호출할 수 있는 것처럼, ILE는 다른 언어로 작성된 모듈을 바인드할 수 있게 합니다. 그러므로 RPG, COBOL, C, C++, 및 CL 등을 사용하여 각각 작성된 모듈로 이루어진 단일 실행가능 프로그램을 작성할 수 있습니다.

모듈성

어플리케이션 프로그래밍에 대해 모듈 접근 방식을 사용하여 얻는 이점은 다음과 같습니다.

- 보다 빠른 컴파일 시간

  컴파일하는 코드의 단위가 적을수록 컴파일러가 빨리 처리할 수 있습니다. 이러한 이점은 흔히 하나 또는 두 행만 변경해야 하는 경우의 유지보수 과정에서 특히 중요합니다. 두 행을 변경할 경우 2000행을 재컴파일해야 할 수도 있습니다. 이러한 점이 자원의 효율적 사용을 어렵게 합니다.

  만약 코드를 모듈화시키고 ILE 바인딩 기능을 사용한다면, 100 또는 200행만 재컴파일할 수도 있습니다. 바인딩 단계가 포함되어 있더라도 이러한 처리 과정이 훨씬 더 빠릅니다.

- 간단한 유지보수

  대형 프로그램을 갱신할 경우 진행 상태를 정확히 이해한다는 것은 매우 어렵습니다. 이러한 점은 원래 프로그래머가 사용자와 다른 방식으로 프로그램을 작성했을 경우 특히 그렇습니다. 좀더 작은 단위로 된 코드는 단일 기능을 나타내는 경향이 있으며, 그것의 내부 작업을 파악하는 것은 아주 쉽습니다. 그러므로, 논리 흐름은 한층 더 분명해지며, 변경할 때 원하지 않는 부작용(side effect)을 유발시킬 가능성이 훨씬 줄어듭니다.

- 간단한 테스트

  보다 작은 컴파일 단위는 기능을 분리해서 테스트할 수 있도록 합니다. 이러한 분리는 테스트 적용 범위가 완벽한지를 알 수 있도록 합니다. 즉, 모든 가능한 입력과 논리 경로가 테스트됩니다.

- 프로그래밍 자원의 효과적인 활용

  모듈화는 작업을 더 많은 부분(division)으로 세분화하도록 합니다. 매우 큰 프로그램을 작성할 경우에는 작업을 세분화하는 것이(가능하지만) 어렵습니다. 프로그램의 모든 부분을 코딩하는 것은 초급 프로그래머의 경우에는 무리이고, 전문 프로그래머의 경우에는 기술 낭비라고 할 수 있습니다.

- 다른 플랫폼에서의 보다 쉬운 코드 마이그레이션

  UNIX®와 같이 다른 플랫폼에서 작성된 프로그램은 종종 모듈화되어 있습니다. 그러한 모듈들은 OS/400으로 마이그레이트하여 ILE 프로그램으로 통합시킬 수 있습니다.

재사용 가능 구성요소

ILE는 사용자가 자신의 프로그램과 혼합하여 사용할 수 있는 루틴 패키지를 선택할 수 있도록 합니다. ILE 언어로 작성된 루틴들은 모든 iSeries ILE 컴파일러 사용자들이 사용할 수 있습니다. 프로그래머가 자신이 선택한 언어로 프로그램을 작성할 수 있다는 사실은 루틴 선택에서 폭넓은 가능성을 보장합니다.

IBM과 다른 업체에서 사용자들에게 이 패키지를 제공하기 위해 사용하는 동일한 메커니즘을 사용자 자신의 어플리케이션에도 사용할 수 있습니다. 현재 시스템에서 자체적인 표준 루틴 세트를 개발할 수 있으며, 선택한 모든 언어로도 개발할 수 있습니다.

사용자의 어플리케이션에 즉시 가용한(off-the-shelf) 루틴을 사용할 수 있을 뿐만 아니라, 선택한 ILE 언어로 루틴을 개발하여 모든 ILE 언어 사용자에게 그것을 판매할 수도 있습니다.

공통 실행시 서비스

즉시 가용한(off-the-shelf) 구성요소(**바인드 가능 API**)의 선택은 ILE의 부분으로 공급되며, 사용자의 어플리케이션에 결합되도록 준비되어 있습니다. 이 API는 다음과 같은 서비스를 제공합니다.

날짜 및 시간 조작

메세지 처리

산술 루틴

화면 처리시 보다 나은 제어

동적 기억장치 할당

시간이 지남에 따라 추가 루틴이 이러한 세트에 포함되며, 다른 것들은 제3의 업체를 통해 구할 수 있습니다.

IBM에는 ILE에서 제공하는 API에 관해 자세한 정보를 제공하는 온라인 정보가 있습니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

기존 어플리케이션과의 공존

ILE 프로그램은 현재 OPM 프로그램과 공존할 수 있습니다. ILE 프로그램은 OPM 프로그램 및 다른 ILE 프로그램을 호출할 수 있습니다. 이와 마찬가지로, OPM 프로그램은 ILE 프로그램 및 다른 OPM 프로그램을 호출할 수도 있습니다. 그러므로, 주의깊게 계획하면 ILE로의 단계적인 전환이 가능합니다.

소스 디버거

소스 디버거는 ILE 프로그램과 서비스 프로그램을 디버그할 수 있도록 합니다. 소스 디버거에 관한 정보는 145 페이지의 제 10 장『디버깅 고려사항』부분을 참조하십시오.

자원의 효율적 제어

ILE가 소개되기 전에는 프로그램에서 사용되는 자원들이(예: 열린 파일) 다음으로 한정(즉, 소유)되었습니다.

자원을 할당하는 프로그램

작업

대부분의 경우 이러한 제한사항은 어플리케이션 설계자의 선택(trade-off)을 강요합니다.

ILE는 세 번째의 대체 방안을 제공합니다. 작업 부분은 자원을 소유할 수 있습니다. 이 방법은 **활성 그룹**이라는 ILE 구조를 사용하여 이루어집니다. ILE하에서는 자원의 범위가 다음 범위까지 확장됩니다.

프로그램

활성 그룹

작업

공유 열린 자료 경로--시나리오

공유 열린 자료 경로(ODP)는 사용자들이 ILE의 새로운 단계별 범위지정을 통해 자원을 더 효율적으로 제어할 수 있도록 하는 예입니다.

iSeries 서버에서 어플리케이션의 성능을 향상시키기 위해 프로그래머가 고객 마스터 파일에 공유 ODP를 사용하기로 결정합니다. 고객 마스터 파일은 주문 입력 및 청구 어플리케이션 양식에 사용됩니다.

공유 ODP는 작업에까지 영향을 미치므로, 하나의 어플리케이션이 부주의하게 다른 어플리케이션에 문제점을 일으킬 수도 있습니다. 사실, 이러한 문제를 피하려면 어플리케이션 개발자 사이에 세심한 조정이 필요합니다. 어플리케이션을 서로 다른 업체에서 구입했다면, 문제점을 피할 방법이 없을 수도 있습니다.

어떤 문제점이 발생할 수 있는가? 다음 시나리오를 살펴보십시오.

1. 고객 마스터 파일은 계좌번호를 키로 하고 있으며, 계좌번호 A1, A2, B1, C1, C2, D1, D2에 대한 레코드가 있습니다.
2. 오퍼레이터는 마스터 파일 레코드를 검토하며 다음 레코드를 요구하기 전에 각각의 레코드를 요구에 따라 갱신합니다. 현재 표시되어 있는 레코드는 계좌번호 B1에 대한 것입니다.
3. 전화가 연결되었습니다. 고객 D1이 주문을 원합니다.
4. 오퍼레이터는 주문 입력으로 가기 기능 키를 누르고, 고객 D1 주문을 처리한 후 마스터 파일 화면으로 리턴합니다.
5. 프로그램은 여전히 B1에 있는 레코드를 올바르게 표시합니다. 그러나 오퍼레이터가 다음 레코드를 요구할 때는 어떤 레코드가 표시되겠습니까 ?

만약 D2라고 대답했다면, 정답입니다. 주문 입력 어플리케이션이 레코드 D1을 읽을 때 공유 ODP는 작업에까지 영향을 미치므로 현재 파일 위치가 변경됩니다. 그러므로, 다음 레코드를 요구하는 것은 D1 이후의 다음 레코드를 의미합니다.

ILE하에서 이러한 문제점은 청구에 할당된 활성 그룹에 마스터 파일 유지보수를 실행하여 막을 수 있습니다. 이와 같이, 주문 입력 어플리케이션은 자체의 활성 그룹내에서 실행됩니다. 각 어플리케이션에는 여전히 공유 ODP의 이점이 있으나 각각에는 관련된 활성 그룹에 속해 있는 자체의 공유 ODP가 있습니다. 이러한 범위 레벨은 이 예에서 보여준 간섭 현상을 막습니다.

활성 그룹에까지 자원을 확장(scoping)하는 것은 프로그래머가 작업에 실행하는 어떠한 다른 어플리케이션과도 독립적으로 실행할 수 있는 어플리케이션을 자유롭게 개발하도록 합니다. 이것은 요구되는 공동 노력을 감소시킬 수 있고 현재 어플리케이션 패키지에 드롭 인(drop-in) 확장을 작성할 수 있는 능력을 향상시켜 줍니다.

확약 제어--시나리오

어플리케이션에까지 공유 ODP를 확장할 수 있는 능력은 확약 제어 영역에 유용합니다.

또한 확약 제어하에 있는 파일을 공유 ODP로 사용하려는 것으로 가정하십시오. ILE 없이 한 개의 프로그램이 확약 제어하에서 파일을 열려는 경우에는 작업내에 있는 모든 프로그램은 그렇게 해야 합니다. 확약 기능이 단지 한 개 또는 두 개의 프로그램에서만 요구되는 경우에도 그렇습니다.

또한 이런 상황에 생길 수 있는 문제점 중 한 가지는 만약 작업내에 있는 어떤 프로그램이 확약 작업을 수행하고 있다면 모든 갱신사항이 확약된다는 것입니다. 그것이 논리적으로 문제점이 있는 어플리케이션 부분이 아니라 할지라도 갱신이 확약됩니다.

이러한 문제점은 확약 제어가 필요한 어플리케이션의 각 부분을 분리 활성 그룹내에서 실행하여 피할 수 있습니다.

언어 상호작용에 대한 보다 효율적인 제어

ILE가 없으면 iSeries 서버에서 프로그램을 실행하는 방법이 다음 조합에 의해 결정됩니다.

언어 표준(예: COBOL 및 C를 위한 ANSI 표준)

컴파일러 개발자

이러한 조합은 언어를 혼합하여 사용할 경우 문제점을 일으킬 수 있습니다.

혼합 언어--시나리오

ILE에 의해 도입된 활성 그룹 없이 OPM 언어 사이의 상호작용을 예측하기는 어렵습니다. ILE 활성 그룹이 바로 이와 같은 어려움을 해결할 수 있습니다.

예를 들면, COBOL과 다른 언어를 혼합하여 사용할 경우 발생되는 문제점에 대해 생각해보십시오. COBOL 언어 표준에는 **실행 단위(run unit)**로 알려진 개념이 포함되어 있습니다. 실행 단위 그룹 프로그램은 일정한 조건하에서는 단일 엔티티로 처리되도록 함께 프로그래밍됩니다. 이것은 매우 유용한 피처가 될 수 있습니다.

예를 들어, PRGA가 PRGB를 호출하고, 차례로 PRGC를 호출하는 세 개의 ILE COBOL 프로그램(PRGA, PRGB 및 PRGC)을 가정하십시오(그림 1 참조). ILE COBOL의 규칙에서는 세 개 프로그램의 실행 단위가 같습니다. 따라서, 이 중 어느 것이라도 종료하면 세 개의 프로그램 모두 종료되어 제어가 PRGA의 호출자에게 리턴되어야 합니다.

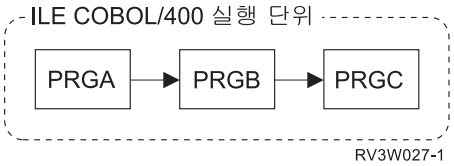

그림 1. 하나의 실행 단위에 있는 세 개의 ILE COBOL 프로그램

이제 RPG 프로그램(RPG1)을 어플리케이션에 적용시키고, 또한 RPG1을 COBOL 프로그램 PRGB가 호출하는 것으로 가정하십시오(그림 2 참조). 이 경우 RPG 프로그램은 프로그램이 최종 레코드(LR) 인디케이터에 리턴할 때까지 변수, 파일 및 다른 자원이 그대로 남아 있기를 기대합니다.

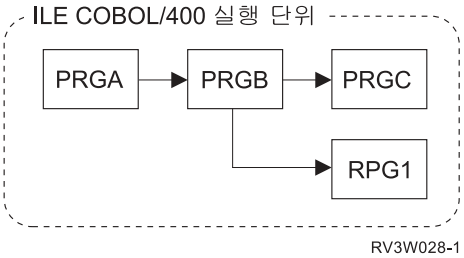

그림 2. 하나의 실행 단위에 있는 세 개의 ILE COBOL 프로그램 및 하나의 ILE RPG 프로그램

그러나 프로그램 RPG1이 RPG로 작성되었다는 사실이 RPG1이 COBOL 실행 단위의 부분으로서 실행될 때 모든 PRG 의미가 적용된다는 것을 보장하는 것은 아닙니다. 만약 실행 단위가 종료되면 RPG1은 LR 인디케이터 세트와 관계 없이 사라집니다. 대부분의 경우 이러한 상황은 사용자가 원하는 것일 수도 있습니다. 그러나 만약 RPG1이 송장 번호 발행을 제어하는 유틸리티 프로그램이라면, 이러한 상황은 받아들이기 어렵습니다.

이러한 상황이 발생하지 않도록 COBOL 실행 단위에서 분리 활성 그룹 내에 RPG 프로그램을 실행시킬 수 있습니다(그림 3 참조). ILE COBOL 실행 단위 자체는 활성 그룹입니다.

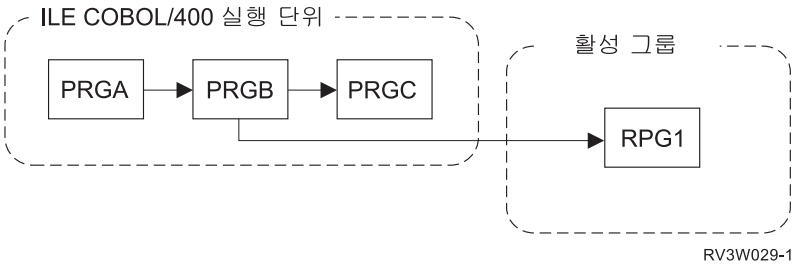

그림 3. 분리 활성 그룹내의 ILE RPG 프로그램

OPM 실행 단위와 ILE 실행 단위의 차이점에 관한 정보는 ILE COBOL for AS/400® Programmer's Guide 를 참조하십시오.

효율적인 코드 최적화

ILE 변환 프로그램은 기본 프로그램 모델(OPM) 변환 프로그램보다 더 많은 유형의 최적화를 수행합니다. 각 컴파일러가 일부 최적화를 수행하지만 OS/400에 대한 대부분의 최적화는 ILE 변환 프로그램에 의해 이루어집니다.

ILE 가능 컴파일러는 모듈을 직접 생성하지 않습니다. 처음에는 컴파일러가 모듈의 중간 형태를 생성하며 그 다음에는 중간 코드를 실행가능한 명령으로 변환하는 ILE 변환 프로그램을 호출합니다.

C를 위한 더 좋은 환경

C는 툴 빌더를 위한 일반적인 언어입니다. 이러한 이유로 인해 더 좋은 C 언어란 더 많은 최신 어플리케이션 개발 툴을 OS/400으로 마이그레이트하는 것을 의미합니다. 한편, 사용자 입장에서는 다음과 같은 폭넓은 선택이 가능하다는 것을 의미합니다.

CASE 툴

4세대 언어(4GL)

추가 프로그래밍 언어

편집기

디버거

미래를 위한 토대

ILE가 제공하는 이점과 기능이 미래에는 더욱 중요할 것입니다. 미래의 ILE 컴파일러는 상당한 개선점을 제공할 것입니다. 현재 우리는 오브젝트 지향 프로그래밍 언어와 비주얼 프로그래밍 툴 방향으로 나아가고 있기 때문에 ILE의 필요성은 더욱 분명해지고 있습니다.

점차, 프로그래밍 방법이 고도로 모듈화한 접근 방법에 의지하고 있습니다. 완전한 어플리케이션을 형성하기 위해서는 수천 개의 재사용이 가능한 작은 구성요소를 조합해야 합니다. 이들 구성요소가 그들 사이에서 빠르게 제어를 전송하지 못하면 최종 어플리케이션은 작업할 수 없습니다.

ILE의 발전 과정

ILE는 OS/400 프로그램 모델이 발전하는 스테이지입니다. 각 단계는 어플리케이션 프로그래머의 변화하는 요구에 따라 발전해 왔습니다.

AS/400 시스템이 처음 소개되었을 때 제공된 프로그램 환경을 기본 프로그램 모델(OPM)이라고 합니다. 버전 1 릴리스 2에서는 확장 프로그램 모델(EPM)이 소개되었습니다.

기본 프로그램 모델 설명

iSeries 서버에서 어플리케이션 개발자가 소스 코드를 소스 파일로 입력하여 그 소스를 컴파일합니다. 컴파일이 성공적으로 수행되면 프로그램 오브젝트가 작성됩니다. 프로그램을 작성하고 실행하기 위해 OS/400이 제공하는 기능, 프로세스, 규칙 세트를 **기본 프로그램 모델(OPM)**이라고 합니다.

OPM 컴파일러는 프로그램 오브젝트를 생성할 때 추가 코드를 생성합니다. 추가 코드는 프로그램 변수를 초기화하며 특정 언어에 필요한 특수 처리에 대해 필요한 코드를 제공합니다. 특수 처리는 해당 프로그램에 의해 예상되는 모든 입력 매개변수를 처리할 수 있습니다. 프로그램이 실행을 시작할 때 추가 컴파일러 생성 코드는 프로그램의 시작 위치(입력점)가 됩니다.

프로그램은 보통 OS/400이 호출 요구를 만날 때 활성화됩니다. 실행시, 다른 프로그램을 호출하는 것이 **동적 프로그램 호출**입니다. 동적 프로그램 호출에 필요한 자원은 중요할 수 있습니다. 어플리케이션 개발자는 종종 어플리케이션이 동적 프로그램 호출 횟수를 최소화하는 몇 개의 대형 프로그램으로 이루어지도록 설계합니다.

그림 4에서는 OPM과 오퍼레이팅 시스템 사이의 관계를 보여줍니다. 다음과 같이 RPG, COBOL, CL, BASIC 및 PL/I 모두가 이 모델내에서 작업합니다.

OPM 경계를 구성하는 점선이 OPM이 OS/400의 완벽한 부분임을 나타냅니다. 이 통합은 컴파일러 출력기에 의해 정상적으로 제공된 많은 기능이 오퍼레이팅 시스템에 내장됨을 의미합니다. 결과로 나타난 호출 규약의 표준은 한 개의 언어로 작성된 프로그

램이 다른 언어로 작성된 프로그램을 자유롭게 호출하는 것을 허용합니다. 예를 들면, RPG로 작성된 어플리케이션은 파일 대체를 발행하거나 스트링 조작을 수행하거나 메세지를 송신하기 위해 다수의 CL 프로그램을 일반적으로 포함합니다.

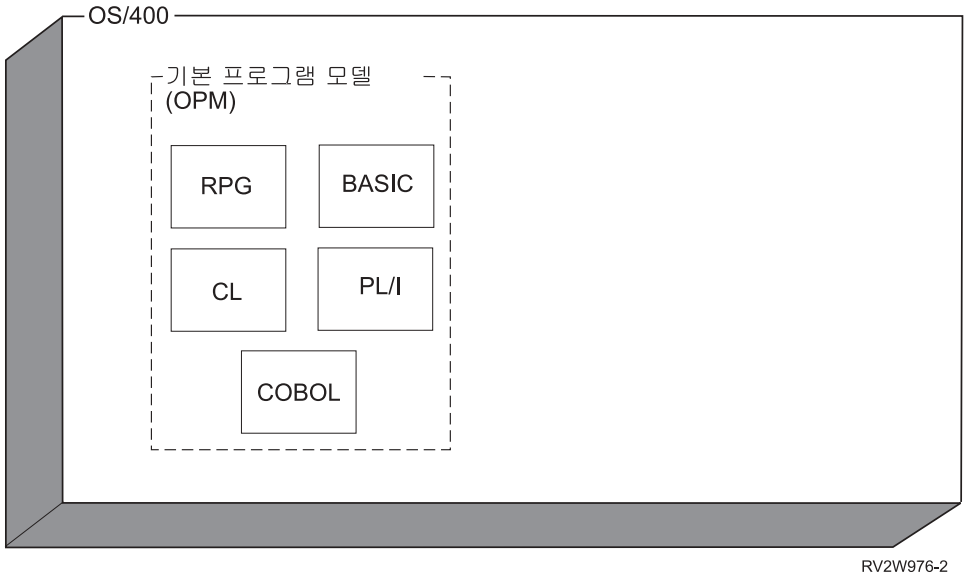

그림 4. OPM과 OS/400의 관계

OPM의 기본 특성

다음 리스트는 OPM의 기본 특성입니다.

- 기존 RPG와 COBOL 프로그램에 효과적임

  OPM은 기존 RPG와 COBOL 프로그램(즉, 상대적으로 크고 기능이 다양한 프로그램)을 지원하는 데 이상적입니다.

- 동적 바인딩

  프로그램 A가 프로그램 B 호출을 원하면 곧바로 수행합니다. 이 동적 프로그램 호출은 단순하고 강력한 기능이 있습니다. 실행시, 오퍼레이팅 시스템은 프로그램 B를 찾아내어 사용자가 그것을 사용할 권리를 보장합니다.

  OPM 프로그램에는 단 하나의 입력점이 있으며, 반면에 ILE 프로그램의 각 프로시듀어가 입력점이 될 수 있습니다.

- 제한된 자료 공유

  OPM에서는 내부 프로시듀어가 전체 프로그램과 변수를 공유해야 하며 반면에 ILE에서는 각 프로시듀어가 내부에 국한된 변수를 가질 수 있습니다.

확장 프로그램 모델 설명

OPM은 유용한 목적을 위해 지속적으로 사용되고 있습니다. 그러나 OPM은 C와 같은 언어로 정의된 프로시듀어에 대해서는 직접적인 지원을 제공하지 않습니다. **프로시듀어**

는 특정 타스크를 실행한 후 호출자에게 리턴하는 자체내에 포함된 고급 언어(HLL) 명령문 세트입니다. 개별 언어는 프로시듀어가 정의되는 방식에 따라 다릅니다. C에서는 프로시듀어를 기능이라고 합니다.

컴파일 단위 사이에서 프로시듀어 호출을 정의하거나 로컬 변수로 프로시듀어를 정의하는 언어가 iSeries 서버에서 실행되도록 OPM이 향상되었습니다. 이와 같이 OPM 기능을 강화시킨 것을 **확장 프로그램 모델(EPM)**이라고 합니다. ILE 이전에 EPM은 Pascal 및 C와 같이 프로시듀어를 기초로 한 언어의 중간 솔루션으로서 사용되었습니다.

iSeries 서버는 임의의 EPM 컴파일러를 더 이상 제공하지 않습니다.

통합 언어 환경 설명

그림 5에서 볼 수 있듯이 ILE는 OPM과 마찬가지로 OS/400에 단단히 통합되어 있습니다. ILE는 프로시듀어를 기초로 한 언어에 대해 EPM이 제공하는 것과 같은 유형의 지원을 제공하지만 ILE는 훨씬 더 완전하고 일관성이 있습니다. ILE의 설계는 RPG 및 COBOL과 같은 보다 통상적인 언어와 향후 언어 발전에 대비하고 있습니다.

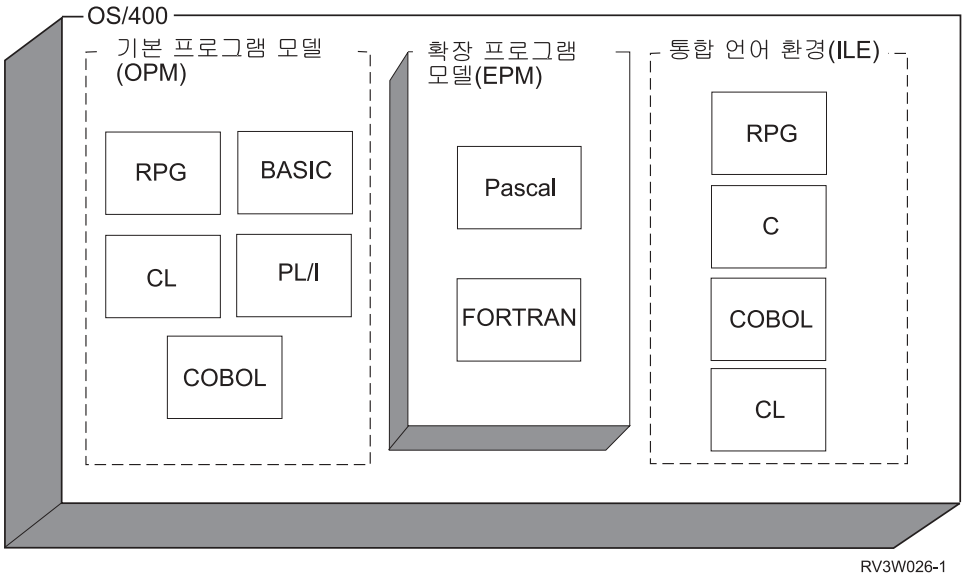

그림 5. OPM, EPM 및 ILE와 OS/400의 관계

프로시듀어를 기초로 한 언어의 기본 특성

프로시듀어를 기초로 한 언어에는 다음과 같은 특성이 있습니다.

- 로컬 범위 변수

로컬 범위 변수는 그것을 정의하는 프로시듀어내에만 알려집니다. 상응하는 로컬 범위 변수는 두 개의 분리 자료 부분을 참조하는 같은 이름이 있는 두 개의 변수를 정의할 수 있습니다. 예를 들면, 변수 COUNT에는 서브루틴 CALCYR에서 4자릿수의 길이 및 서브루틴 CALCDAY에서 6자릿수의 길이가 있습니다.

로컬 범위 변수는 사용자가 몇 개의 다른 프로그램으로 복사되도록 한 서브루틴을 작성할 때 상당한 이점이 있습니다. 로컬로 범위가 정해진 변수가 없으면 프로그래머는 반드시 서브루틴명에 기초한 명명 변수 등의 구성표를 반드시 사용해야 합니다.

- 자동 변수

  자동 변수는 프로시듀어가 시작될 때마다 작성됩니다. 자동 변수는 프로시듀어가 종료될 때 소멸됩니다.

- 외부 변수

  외부 자료는 프로그램간 자료를 공유하기 위한 하나의 방법입니다. 만일 프로그램 A가 외부로 자료 항목을 선언하면 프로그램 A는 자료를 공유하려는 다른 프로그램으로 해당 자료 항목을 **내보내기**했다고 합니다. 그러고 나서, 프로그램 D는 전혀 프로그램 B와 C의 개입 없이 항목을 **가져오기**할 수 있습니다. 가져오기와 내보내기에 관한 자세한 정보는 14 페이지의 『모듈 오브젝트』 부분을 참조하십시오.

- 다수 입력점

  COBOL 및 RPG 프로그램에는 단지 하나의 입력점이 있습니다. COBOL 프로그램에서는 PROCEDURE DIVISION 시작이고, RPG 프로그램에서는 첫 페이지(1P) 출력입니다. 이것이 OPM이 지원하는 모델입니다.

  반면에 프로시듀어를 기초로 한 언어에는 다수 입력점을 가집니다. 예를 들면, C 프로그램은 다른 프로그램에 의해 사용될 수 있도록 완전히 서브루틴으로 구성될 수 있습니다. 이러한 프로시듀어는 필요하다면 가져오기할 다른 프로그램을 위해 관련 자료와 함께 내보내기될 수 있습니다.

  ILE에서는 이러한 유형의 프로그램을 **서비스 프로그램**이라고 합니다. 서비스 프로그램 임의의 ILE 언어에서 모듈을 포함할 수 있습니다. 서비스 프로그램은 개념상으로 Windows® 또는 OS/2®의 동적 링크 라이브러리(DLL)와 유사합니다. 서비스 프로그램은 18 페이지의 『서비스 프로그램』에서 더 자세히 설명됩니다.

- 빈번한 호출

  프로시듀어를 기초로 한 언어는 원래 호출 집약적입니다. EPM이 호출의 오버헤드를 최소화하기 위한 기능을 제공함에도 불구하고, 개별적으로 컴파일된 단위 사이의 프로시듀어 호출에는 아직도 비교적 많은 오버헤드가 발생합니다. ILE는 이러한 유형의 호출을 상당히 향상시킵니다.

제 2 장 ILE 기본 개념

표 1에서는 기본 프로그램 모델(OPM)과 통합 언어 환경(ILE) 모델을 비교 및 대비하여 설명합니다. 이 장에서는 아래의 표에 나열된 유사점과 차이점을 간단하게 설명합니다.

표 1. OPM과 ILE간의 유사점과 차이점

OPM	ILE
프로그램	프로그램
	서비스 프로그램
컴파일의 결과 실행가능 프로그램이 됨	컴파일의 결과 실행불능 모듈 오브젝트가 됨
컴파일, 실행	컴파일, 바인드, 실행
각 언어에 대해 시뮬레이트된 실행 단위	활성 그룹
동적 프로그램 호출	동적 프로그램 호출
	정적 프로시듀어 호출
단일 언어 중심	혼합 언어 중심
언어 특정 오류 처리	공통 오류 처리
	언어 특정 오류 처리
OPM 디버거	소스 레벨 디버거

ILE 프로그램의 구조

ILE 프로그램은 하나 또는 여러 개의 모듈을 포함합니다. 또한 모듈에는 하나 또는 여러 개의 프로시듀어가 포함되어 있습니다(그림 6 참조).

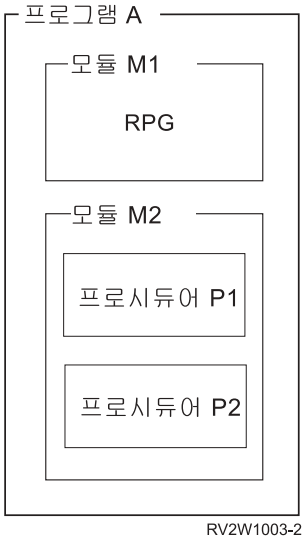

그림 6. ILE 프로그램의 구조

프로시듀어

*프로시듀어*란 특정 타스크를 수행한 후 호출자에게 리턴하는 자체내에 포함된 (self-contained) 고급 언어문 세트입니다. 예를 들어, ILE C 함수는 ILE 프로시듀어입니다.

모듈 오브젝트

모듈 오브젝트는 ILE 컴파일러의 출력인 *실행불능* 오브젝트입니다. 모듈 오브젝트는 기호 *MODULE로 시스템에 표시됩니다. 모듈 오브젝트는 실행가능한 ILE 오브젝트를 빌드하기 위한 기본 구성 블록입니다. 이것이 ILE와 OPM간의 중요한 차이점입니다. OPM 컴파일러의 출력은 *실행가능* 프로그램입니다.

모듈 오브젝트는 하나 이상의 프로시듀어와 자료 항목 스펙으로 구성될 수 있습니다. 다른 ILE 오브젝트에서 한 개의 모듈에 있는 프로시듀어 또는 자료 항목에 직접 액세스할 수 있습니다. 다른 ILE 오브젝트에 의해 직접 액세스될 수 있는 프로시듀어와 자료 항목의 코딩에 관한 세부사항은 ILE HLL 프로그래머 안내서를 참조하십시오.

ILE RPG, ILE COBOL, ILE C 및 ILE C++ 모두 다음과 같은 공통 개념을 갖습니다.

- 내보내기

  내보내기는 다른 ILE 오브젝트에 의해 사용이 가능한 모듈 오브젝트로 코드화된 프로시듀어 또는 자료 항목입니다. 내보내기는 이름과 연관된 유형, 프로시듀어 또는 자료에 의해 식별됩니다.

  내보내기를 **정의**라고도 합니다.

- 가져오기

  가져오기는 현재 모듈 오브젝트에 정의되지 않은 프로시듀어명 또는 자료 항목에 대한 사용 또는 참조입니다. 가져오기는 이름과 연관된 유형, 프로시듀어 또는 자료에 의해 식별됩니다.

  가져오기를 **참조**라고도 합니다.

모듈 오브젝트는 실행가능한 ILE 오브젝트의 기본 구성 블록입니다. 그러므로, 모듈 오브젝트가 작성될 때 다음이 생성될 수도 있습니다.

- 디버그 자료

  디버그 자료는 실행 중인 ILE 오브젝트 디버깅에 필요한 자료입니다. 이 자료는 선택적입니다.

- PEP(프로그램 입력 프로시듀어)

프로그램 입력 프로시듀어는 동적 프로그램 호출에서 ILE 프로그램을 위한 입력점인 컴파일러 생성 코드입니다. 이것은 OPM 프로그램의 입력점을 위해 제공되는 코드와 유사합니다.

- UEP(사용자 입력 프로시듀어)

  프로그래머가 작성하는 **사용자 입력 프로시듀어**는 동적 프로그램 호출의 목표입니다. 이것은 PEP가 제어하는 프로시듀어입니다. C 프로그램의 기본() 기능은 ILE에서 해당 프로그램의 UEP가 됩니다.

그림 7에서는 모듈 오브젝트의 개념적인 뷰를 보여줍니다. 이 예에서 모듈 오브젝트 M1은 두 개의 프로시듀어(Draw_Line 및 Draw_Arc)와 한 개의 자료 항목(rtn_code)을 내보내기합니다. 모듈 오브젝트 M1은 Draw_Plot이라 불리는 프로시듀어를 가져오기합니다. 이 특정 모듈 오브젝트에는 PEP, 해당 UEP(프로시듀어 Draw_Arc) 및 디버그 자료가 있습니다.

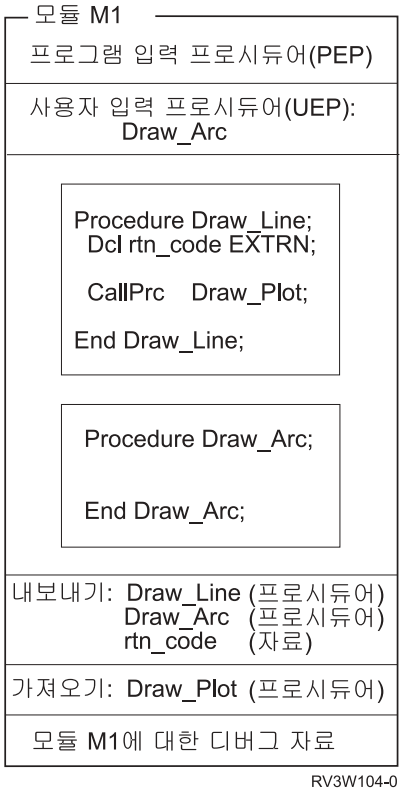

그림 7. 모듈의 개념적인 뷰

*MODULE 오브젝트의 특성

- *MODULE 오브젝트는 ILE 컴파일러의 결과입니다.
- ILE 실행가능 오브젝트의 기본 구성 블록입니다.
- 실행가능 오브젝트가 아닙니다.
- 정의된 PEP를 가질 수 있습니다.

- PEP가 정의되면 UEP도 정의됩니다.
- 프로시듀어 및 자료 항목명을 내보내기할 수 있습니다.
- 프로시듀어 및 자료 항목명을 가져오기할 수 있습니다.
- 정의된 디버그 자료를 가질 수 있습니다.

ILE 프로그램

ILE 프로그램은 OPM 프로그램과 함께 다음과 같은 특성을 공유합니다.

- 이 프로그램은 동적 프로그램 호출을 통해 제어를 획득합니다.
- 프로그램에는 하나의 입력점만 있습니다.
- 이 프로그램은 기호 *PGM 표시로 시스템에서 식별됩니다.

ILE 프로그램에는 OPM 프로그램이 없는 다음과 같은 특성이 있습니다.

- ILE 프로그램은 하나 이상의 복사된 모듈 오브젝트로 작성됩니다.
- 하나 이상의 복사된 모듈은 PEP를 포함할 수 있습니다.
- 어떤 모듈의 PEP가 ILE 프로그램 오브젝트에 대한 PEP로 사용될 것인가에 대한 제어권은 사용자에게 있습니다.

  프로그램 작성(CRTPGM) 명령이 지정될 경우 ENTMOD 매개변수는 PEP가 있는 모듈 중에서 사용자가 프로그램의 입력점을 선택할 수 있도록 해줍니다.

  프로그램에 대한 입력점으로 선택되지 않은 모듈과 연관된 PEP는 무시됩니다. 모듈의 다른 모든 프로시듀어와 자료 항목은 지정된 대로 사용됩니다. 단지 PEP만 무시될 뿐입니다.

동적 프로그램 호출이 ILE 프로그램에 대해 이루어질 때 프로그램 작성시 선택된 모듈의 PEP로 제어가 부여됩니다. 이 PEP는 관련 UEP를 호출합니다.

ILE 프로그램 오브젝트가 작성될 때 디버그 자료가 있는 복사된 모듈에 연관된 프로시듀어만 ILE 디버거에 의해 디버그될 수 있습니다. 디버그 자료는 실행 중인 ILE 프로그램의 성능에 영향을 미치지 않습니다.

17 페이지의 그림 8에서는 ILE 프로그램 오브젝트의 개념적인 뷰를 보여줍니다. 프로그램 PGMEXAMP가 호출될 때 복사된 모듈 오브젝트 M3에 정의된 프로그램의 PEP에 제어가 주어집니다. 또한 복사된 모듈 M2에는 정의된 PEP가 있지만 무시되고 프로그램에 의해 절대로 사용되지 않습니다.

이 프로그램 예에서는 M1과 M3 모듈에만 새로운 ILE 디버거에 필요한 자료가 있습니다. 모듈 M2와 M4의 프로시듀어는 새로운 ILE 디버거를 사용하여 디버그될 수 없습니다.

가져오기 프로시듀어인 print와 SIN은 서비스 프로그램인 PRINTS와 MATHFUNC 각각에서 내보내기된 프로시듀어로 결정됩니다.

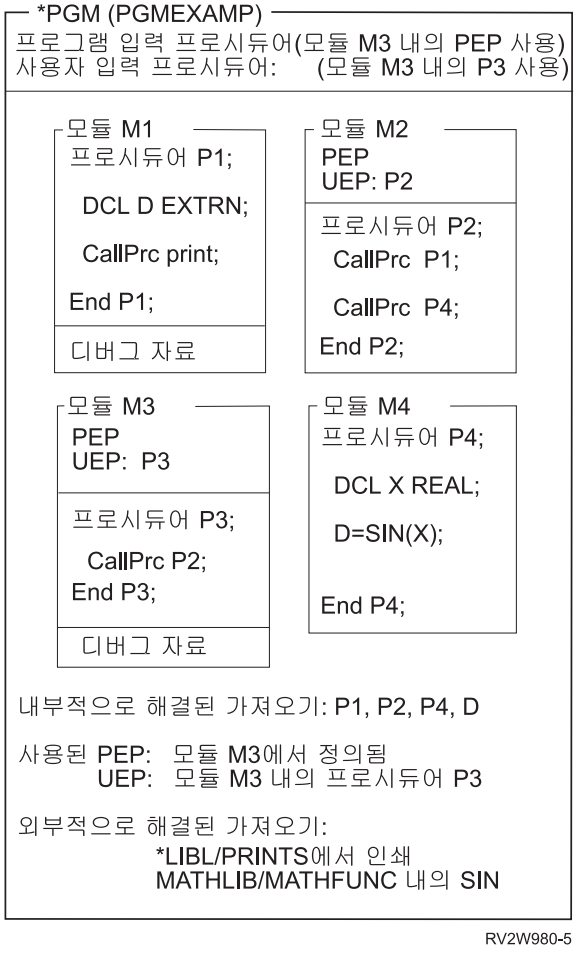

그림 8. ILE 프로그램의 개념적인 뷰

ILE *PGM 오브젝트의 특성:

- 임의의 ILE 언어에서 하나 이상의 모듈이 복사되어 *PGM 오브젝트가 작성됩니다.
- 프로그램 작성자에게 프로그램을 위한 유일한 PEP 모듈을 결정하기 위한 제어권이 있습니다.
- 동적 프로그램 호출의 경우 프로그램을 위한 PEP로 선택된 모듈의 PEP에 제어권이 있습니다.
- 선택된 PEP와 연관된 UEP가 프로그램에 대한 사용자의 입력점입니다.
- 프로시듀어 및 자료 항목명은 프로그램에서 내보내기될 수 없습니다.
- 프로시듀어 또는 자료 항목명은 모듈과 서비스 프로그램에서 가져오기할 수 있지만 프로그램 오브젝트에서는 가져오기할 수 없습니다. 서비스 프로그램에 관한 정보는 18 페이지의 『서비스 프로그램』 부분을 참조하십시오.
- 모듈은 디버그 자료를 가질 수 있습니다.

- 프로그램은 실행가능한 오브젝트입니다.

서비스 프로그램

서비스 프로그램은 다른 ILE 프로그램이나 서비스 프로그램이 쉽게 직접 액세스할 수 있는 실행가능 프로시듀어와 사용가능한 자료 항목의 콜렉션입니다. 여러 면에서 서비스 프로그램은 서브루틴 라이브러리 또는 프로시듀어 라이브러리와 유사합니다.

서비스 프로그램은 다른 ILE 오브젝트에 필요한 공통 서비스를 제공하므로, 이름이 서비스 프로그램입니다. OS/400에서 제공하는 서비스 프로그램 세트의 예가 언어에 대한 실행시 프로시듀어입니다. 이러한 실행시 프로시듀어에는 종종 연산 프로시듀어와 공통 입/출력 프로시듀어와 같은 항목이 있습니다.

서비스 프로그램의 **공용 인터페이스**는 내보내기된 프로시듀어명과 다른 ILE 오브젝트에 의해 액세스가 가능한 자료 항목으로 구성되어 있습니다. 서비스 프로그램을 구성하는 모듈 오브젝트에서 내보내기된 항목만 서비스 프로그램에서 내보내기될 수 있습니다.

프로그래머는 어떤 프로시듀어나 자료 항목을 다른 ILE 오브젝트에게 알려줄 것인지를 지정할 수 있습니다. 그러므로, 서비스 프로그램은 다른 ILE 오브젝트에는 사용되지 않는 숨겨진 또는 사적인 프로시듀어와 자료를 가질 수 있습니다.

갱신된 서비스 프로그램을 사용하는 다른 ILE 프로그램이나 서비스 프로그램을 재작성하지 않고도 서비스 프로그램을 갱신하는 것이 가능합니다. 서비스 프로그램을 변경하는 프로그래머는 변경이 현행 지원 체제와 호환되는지의 여부를 제어합니다.

ILE가 호환될 수 있는 변경을 제어하도록 ILE에서 사용자에게 제공하는 방식은 **바인더 언어**를 사용하는 것입니다. 바인더 언어는 내보내기될 수 있는 프로시듀어명과 자료 항목명의 리스트를 정의할 수 있게 합니다. **서명**은 프로시듀어명과 자료 항목, 바인더 언어에 지정되어 있는 프로시듀어명의 순서에서 생성됩니다. 서비스 프로그램에 호환가능한 변경사항을 작성하려면 새로운 프로시듀어 또는 자료 항목명이 내보내기 리스트의 끝에 추가되어야 합니다. 서명, 바인더 언어, 서비스 프로그램에서 고객의 투자 보호에 관한 자세한 정보는 90 페이지의 『바인더 언어』 부분을 참조하십시오.

19 페이지의 그림 9에서는 서비스 프로그램의 개념적인 뷰를 보여줍니다. 서비스 프로그램을 구성하는 모듈은 17 페이지의 그림 8에 있는 ILE 프로그램 오브젝트 PGMEXAMP를 구성하는 모듈과 같음에 주의하십시오. 서비스 프로그램 SPGMEXAMP용인 이전 서명 Sigyy에는 프로시듀어 P3과 P4의 이름이 있습니다. 서비스 프로그램에 상향 호환이 가능한 변경을 하고 나면, 현재 서명 Sigxx에 프로시듀어 P3 및 P4의 이름 뿐만 아니라 자료 항목 D의 이름도 포함됩니다. 프로시듀어 P3 또는 P4를 사용하는 다른 ILE 프로그램이나 서비스 프로그램을 재작성하지 않아도 됩니다.

서비스 프로그램의 모듈에는 PEP가 있을 수도 있지만 이러한 PEP는 무시됩니다. 서비스 프로그램 자체에는 PEP가 없습니다. 그러므로, 프로그램 오브젝트와는 달리 서비스 프로그램은 동적으로 호출될 수 없습니다.

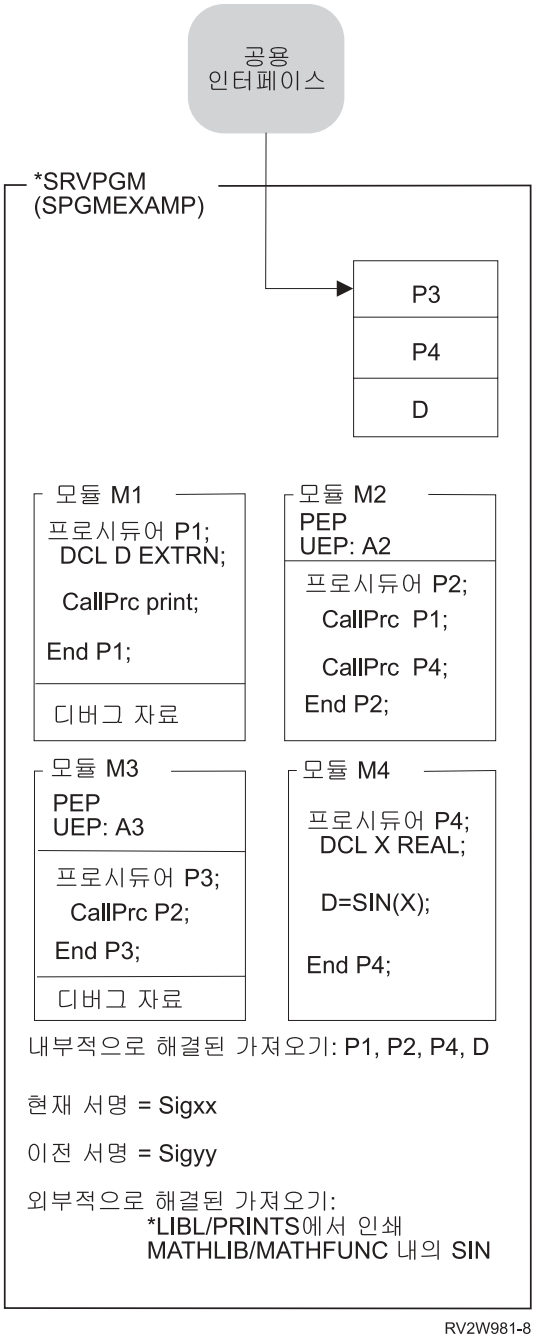

그림 9. ILE 서비스 프로그램의 개념적인 뷰

ILE *SRVPGM 오브젝트의 특성

- 임의의 ILE 언어에서 하나 이상의 모듈이 복사되어 *SRVPGM 오브젝트가 됩니다.

- 어떠한 PEP도 서비스 프로그램과 연관되지 않습니다. PEP가 없기 때문에, 서비스 프로그램에 대한 동적 프로그램 호출은 유효하지 않습니다. 모듈의 PEP는 무시됩니다.
- 다른 ILE 프로그램이나 서비스 프로그램은 공용 인터페이스에 의해 식별되는 이 서비스 프로그램의 내보내기를 사용할 수 있습니다.
- 서명은 서비스 프로그램에서 내보내기된 프로시듀어 또는 자료 항목명에서 생성될 수 있습니다.
- 서비스 프로그램은 이전의 서명이 계속 지원되는 한, ILE 프로그램이나 서비스 프로그램에 영향을 주지 않으면서 교체될 수 있습니다.
- 모듈은 디버그 자료를 가질 수 있습니다.
- 서비스 프로그램은 실행가능한 프로시듀어와 자료 항목의 콜렉션입니다.
- 약 자료는 단지 활성 그룹으로만 내보내기될 수 있습니다. 그것은 서비스 프로그램에서 내보내기되는 공용 인터페이스의 일부가 될 수 없습니다. 약 자료에 관한 정보는 88 페이지의 『가져오기 및 내보내기 개념』에서 내보내기 부분을 참조하십시오.

바인딩 디렉토리

바인딩 디렉토리에는 ILE 프로그램이나 서비스 프로그램을 작성할 때 필요할 수도 있는 모듈과 서비스 프로그램의 이름이 들어 있습니다. 바인딩 디렉토리에 있는 모듈이나 서비스 프로그램은 현재 해결되지 않은 가져오기 요구를 해결할 수 있는 내보내기를 제공할 때에만 사용됩니다. 바인딩 디렉토리는 시스템 오브젝트로서 시스템에서는 *BNDDIR로 표시됩니다.

바인딩 디렉토리는 선택적입니다. 바인딩 디렉토리는 편리함과 프로그램 크기 때문에 사용됩니다.

- 바인딩 디렉토리는 ILE 프로그램이나 서비스 프로그램을 작성할 때 필요한 모듈이나 서비스 프로그램을 패키징하는 편리한 방법을 제공합니다. 예를 들면, 한 개의 바인딩 디렉토리에는 산술 기능을 제공하는 모든 모듈과 서비스 프로그램이 들어 있습니다. 그러한 기능을 사용하려면 사용할 모든 모듈과 서비스 프로그램을 지정하지 않고 하나의 바인딩 디렉토리만 지정하면 됩니다.

  주: 바인딩 디렉토리에 모듈 또는 서비스 프로그램이 많을수록 프로그램을 바인드하는 시간이 더 길어집니다. 그러므로, 바인딩 디렉토리에 필요한 모듈 또는 서비스 프로그램만 포함되도록 해야 합니다.
- 사용하지 않는 모듈이나 서비스 프로그램을 지정하지 않아도 되기 때문에 바인딩 디렉토리는 프로그램 크기를 줄일 수 있습니다.

바인딩 디렉토리에 입력하는 것에는 거의 제한이 없습니다. 모듈이나 서비스 프로그램의 이름은 해당 오브젝트가 아직 존재하지 않더라도 바인딩 디렉토리에 추가될 수 있습니다.

바인딩 디렉토리와 함께 사용되는 CL 명령의 리스트는 227 페이지의 부록 C『ILE 오브젝트와 함께 사용되는 CL 명령』 부분을 참조하십시오.

그림 10에서는 바인딩 디렉토리의 개념적인 뷰를 보여줍니다.

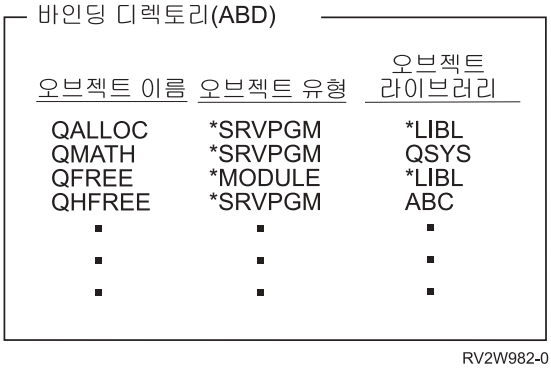

그림 10. 바인딩 디렉토리의 개념적인 뷰

*BNDDIR 오브젝트의 특성:

- ILE 프로그램이나 서비스 프로그램 작성에 필요한 서비스 프로그램과 모듈의 이름을 그룹화하는 데 편리한 방법입니다.
- 바인딩 디렉토리 입력은 단지 이름 뿐이므로 나열된 오브젝트가 아직 시스템에 존재하지 않아도 됩니다.
- 유효한 유일한 라이브러리명은 *LIBL 또는 특정 라이브러리입니다.
- 리스트에 있는 오브젝트는 선택적입니다. 명명된 오브젝트는 해결되지 않은 가져오기가 존재하는 경우 또는 명명된 오브젝트가 해결되지 않은 가져오기 요구를 충족시키기 위해 내보내기를 제공하는 경우에만 사용됩니다.

바인딩 디렉토리 처리

바인딩 중에 다음 순서로 처리가 발생합니다.

1. MODULE 매개변수에 지정된 모든 모듈은 검사됩니다. 바인더가 오브젝트에 의해 가져오고 내보내지는 기호의 리스트를 판별합니다. 모듈은 검사된 후 나열된 순서대로 작성될 프로그램에 바인드됩니다.
2. BNDSRVPGM 매개변수의 모든 서비스 프로그램이 나열된 순서대로 검사됩니다. 서비스 프로그램은 가져오기를 분석하기 위해 필요한 경우에만 바인드됩니다.
3. BNDDIR 매개변수의 모든 바인딩 디렉토리가 나열된 순서대로 처리됩니다. 이들 바인딩 디렉토리에 나열되는 모든 오브젝트가 나열된 순서대로 검사되지만, 가져오기를 분석하기 위해 필요한 경우에만 바인드됩니다. 바인딩 디렉토리의 중복 항목은 자동으로 무시됩니다.
4. 각 모듈은 **참조 시스템 오브젝트**의 리스트를 갖습니다. 이 리스트는 단순히 바인딩 디렉토리의 리스트입니다. 바인드된 모듈의 참조 시스템 오브젝트는 첫 번째 모듈의

모든 참조 시스템 오브젝트가 처리된 후 두 번째 모듈의 오브젝트가 처리되는 방식의 순서로 처리됩니다. 이들 바인딩 디렉토리에 나열되는 오브젝트는 필요할 때만 나열된 순서대로 검사되고, 필요한 경우에만 바인드됩니다. 이 처리는 OPTION(*UNRSLVREF)이 사용되는 경우에도 분석되지 않은 가져오기가 존재하는 경우만 계속됩니다. 따라서, 모든 가져오기가 분석될 때 오브젝트 처리가 중단됩니다.

오브젝트가 검사되는 동안, 오브젝트가 최종적으로 작성될 프로그램에 바인드되지 않는 경우에도 메세지 CPD5D03, 『기호에 대한 정의가 여러번 제공됨』이 나타날 수 있습니다.

모듈은 대개 모듈의 소스 코드에서 명백하지 않은 가져오기를 갖고 있음을 주의하십시오. 이들은 서비스 프로그램에서 런타임 지원이 필요한 다양한 언어 피처를 구현하기 위해 컴파일러에 의해 추가됩니다. 이들 가져오기를 보려면 DSPMOD DETAIL (*IMPORT)를 사용하십시오.

모듈 또는 서비스 프로그램에 대한 가져오고 내보내기 된 기호의 리스트를 보려면, CRTPGM 또는 CRTSRVPGM DETAIL(*EXTENDED) 리스트의 바인더 정보 리스트 섹션을 보십시오. 바인딩 중에 검사되는 오브젝트가 나열되어 있습니다.

작성될 프로그램 또는 서비스 프로그램에 바인드되는 모듈 또는 서비스 프로그램 오브젝트는 CRTPGM 또는 CRTSRVPGM DETAIL(*EXTENDED) 리스트의 바인더 정보 리스트 섹션에 표시됩니다. 일단 오브젝트가 작성되면, 또한 DSPPGM 또는 DSPSRVPGM 명령 DETAIL(*MODULE)을 사용하여 바인드된 *MODULE 오브젝트를 보고, DETAIL(*SRVPGM)을 사용하여 바인드된 *SRVPGM 오브젝트의 리스트를 볼 수 있습니다.

DSPMOD DETAIL(*REFSYSOBJ)를 사용하여 바인딩 디렉토리인 참조 시스템 오브젝트의 리스트를 볼 수 있습니다. 이 바인딩 디렉토리에는 일반적으로 오퍼레이팅 시스템이나 언어 런타임 지원에 의해 제공되는 서비스 프로그램 API의 이름이 들어 있습니다. 이 방법으로 프로그래머가 특별한 명령을 지정하지 않고도 모듈을 언어 런타임 지원 및 시스템 API에 바인드할 수 있습니다.

바인더 기능

바인더 기능은 링크 편집기에 의해 제공된 기능과 유사하지만 약간 다른 부분도 있습니다. **바인더**는 지정된 모듈에서 프로시듀어명과 자료 항목명을 위한 가져오기 요구를 처리합니다. 그런 다음 바인더는 지정된 모듈, 서비스 프로그램 및 바인더 디렉토리에서 가져오기에 맞는 내보내기 찾기를 시도합니다.

ILE 프로그램이나 서비스 프로그램 작성에 있어, 바인더는 다음과 같은 유형의 바인딩을 수행합니다.

- 복사에 의한 바인드

  ILE 프로그램이나 서비스 프로그램을 작성하는 경우 다음 사항이 복사됩니다.

  모듈 매개변수에 지정된 모듈

  미해결 가져오기에 대한 내보내기를 제공하는 바인딩 디렉토리에서 선택된 모든 모듈

  복사된 모듈내에서 사용되는 필요한 프로시듀어와 자료 항목의 실제 주소는 ILE 프로그램이나 서비스 프로그램이 작성될 때 설정됩니다.

  예를 들어, 19 페이지의 그림 9에서 모듈 M3의 프로시듀어 P3은 모듈 M2의 프로시듀어 P2를 호출합니다. 주소가 직접 액세스되도록 모듈 M2에 있는 프로시듀어 P2의 실제 주소를 프로시듀어 M3에 알려줍니다.

- 참조에 의한 바인드

  미해결 가져오기 요구에 대해 내보내기를 제공하는 서비스 프로그램의 기호 링크는 작성된 프로그램이나 서비스 프로그램에 저장됩니다. 기호 링크는 내보내기를 제공하는 서비스 프로그램을 참조합니다. 이 링크는 서비스 프로그램이 바인드된 프로그램 오브젝트가 활성화될 때 실제 주소로 변환됩니다.

  19 페이지의 그림 9에서는 서비스 프로그램 *MATHLIB/MATHFUNC의 SIN에 대한 기호 링크의 예를 보여줍니다. SIN에 대한 기호 링크는 서비스 프로그램 SPGMEXAMP가 바인드된 프로그램 오브젝트가 활성화될 때 실제 주소로 변환됩니다.

실행시, 프로시듀어와 자료 항목에 설정된 실제 링크가 사용되면서 다음 두 작업 사이에는 성능상의 차이가 거의 없게 됩니다.

- 로컬 프로시듀어 또는 자료 항목에의 액세스
- 같은 프로그램에 바인드된 다른 모듈. 서비스 프로그램의 프로시듀어 또는 자료 항목의 액세스

24 페이지의 그림 11 및 24 페이지의 그림 12에서는 ILE 프로그램 PGMEXAMP와 서비스 프로그램 SPGMEXAMP가 어떻게 작성되었는지에 관한 개념적인 뷰를 보여줍니다. 바인더는 ILE 프로그램 PGMEXAMP와 서비스 프로그램 SPGMEXAMP를 작성하기 위해 모듈 M1, M2, M3 및 M4와 서비스 프로그램 PRINTS 및 MATHFUNC를 사용합니다.

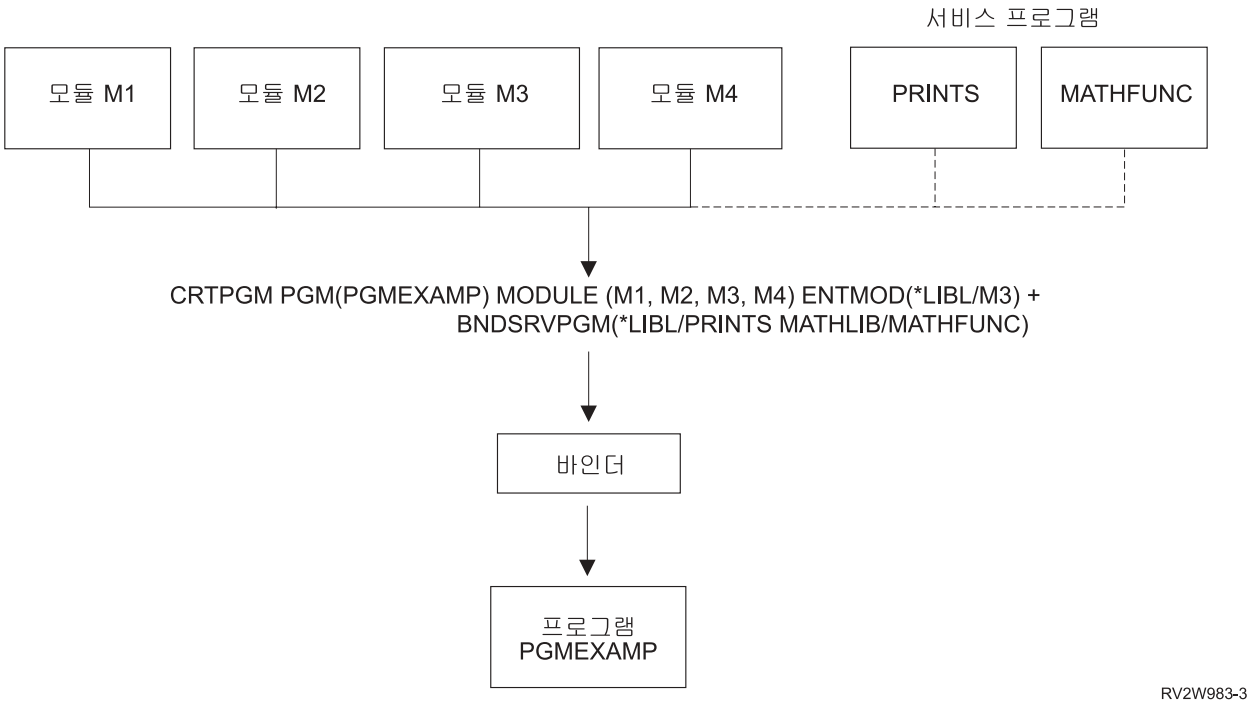

그림 11. ILE 프로그램의 작성. 점선 부분은 서비스 프로그램이 복사에 의해 바인드되는 대신 참조에 의해 바인드됨을 나타냅니다.

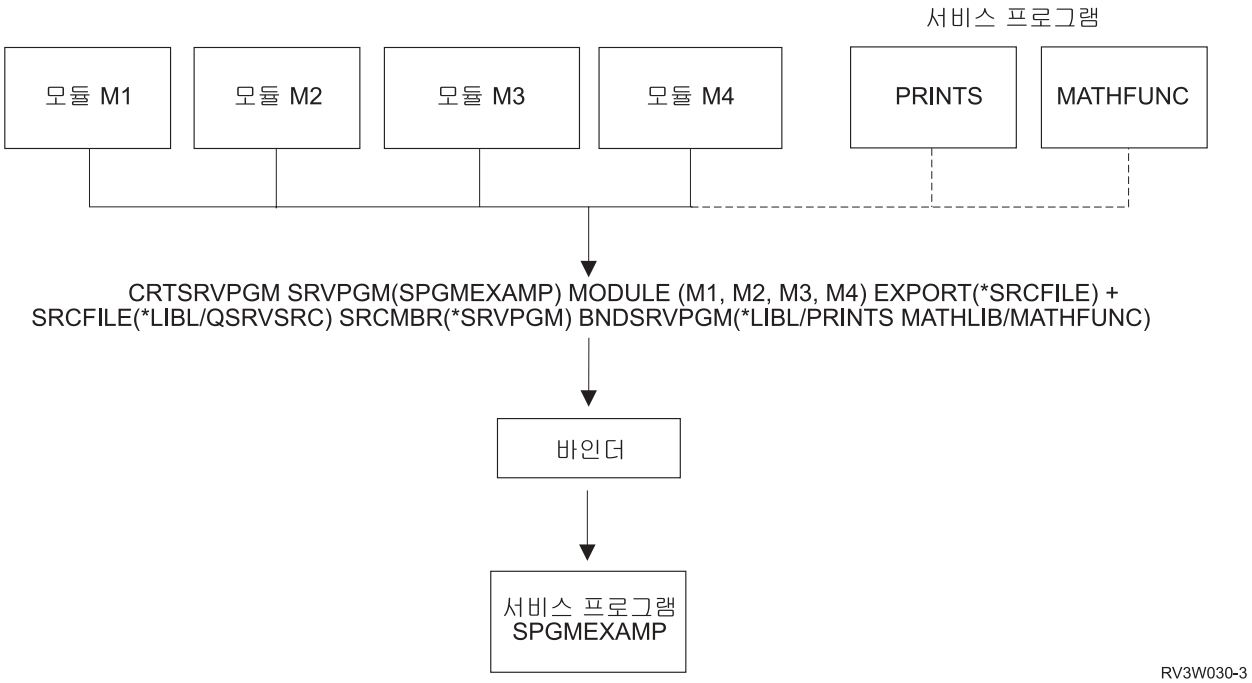

그림 12. 서비스 프로그램의 작성. 점선 부분은 서비스 프로그램이 복사에 의해 바인드되는 대신 참조에 의해 바인드됨을 나타냅니다.

ILE 프로그램이나 서비스 프로그램의 작성에 관한 추가 정보는 75 페이지의 제 5 장 『프로그램 작성 개념』 부분을 참조하십시오.

프로그램 및 프로시듀어 호출

ILE에서는 프로그램이나 프로시듀어를 호출할 수 있습니다. ILE는 호출자가 호출 명령문의 목표가 프로그램인지 또는 프로시듀어인지 식별할 것을 요구합니다. ILE 언어는 프로그램용과 프로시듀어용의 분리 호출 명령문을 사용하여 이 요구를 충족시킵니다. 그러므로, ILE 프로그램을 작성할 때에는 프로그램을 호출하는지 프로시듀어를 호출하는지를 알아야 합니다.

각각의 ILE 언어에는 동적 프로그램 호출과 정적 프로시듀어 호출을 구분할 수 있게 하는 고유 구문이 있습니다. 각 ILE 언어의 표준 호출 명령문은 동적 프로그램 호출 또는 정적 프로시듀어 호출을 디폴트로 합니다. RPG와 COBOL의 경우 디폴트는 동적 프로그램 호출이며, C의 경우 디폴트 상태는 정적 프로시듀어 호출입니다. 따라서, 표준 언어 호출은 OPM 또는 ILE에서 모두 같은 유형의 기능을 수행합니다. 이러한 규칙 때문에 OPM 언어를 ILE 언어로 마이그레이트하기가 비교적 쉽습니다.

바인더는 최대 256자까지의 프로시듀어명을 처리할 수 있습니다. 프로시듀어명의 길이를 결정하려면 ILE HLL 프로그래머 안내서를 참조하십시오.

동적 프로그램 호출

동적 프로그램 호출은 ILE 프로그램 오브젝트 또는 OPM 프로그램 오브젝트 둘다에 제어를 전송할 수 있습니다. 동적 프로그램 호출은 다음 사항을 포함합니다.

- OPM 프로그램은 다른 OPM 프로그램이나 ILE 프로그램을 호출할 수는 있지만 서비스 프로그램은 호출할 수 없습니다.
- ILE 프로그램은 OPM 프로그램이나 다른 ILE 프로그램을 호출할 수는 있지만 서비스 프로그램은 호출할 수 없습니다.
- 서비스 프로그램은 OPM 프로그램이나 ILE 프로그램을 호출할 수는 있지만 다른 서비스 프로그램은 호출할 수 없습니다.

정적 프로시듀어 호출

정적 프로시듀어 호출은 ILE 프로시듀어로 제어를 전송합니다. 정적 프로시듀어 호출은 ILE 언어로만 코드화됩니다. 정적 프로시듀어 호출은 다음 사항을 호출하는 데 사용됩니다.

- 같은 모듈내의 프로시듀어
- 같은 ILE 프로그램이나 서비스 프로그램내 분리 모듈의 프로시듀어
- 분리 ILE 서비스 프로그램의 프로시듀어

26 페이지의 그림 13에서는 정적 프로시듀어 호출의 예를 보여줍니다. 그림에서는 다음을 보여줍니다.

- ILE 프로그램의 프로시듀어는 같은 프로그램내 또는 서비스 프로그램내에서 내보내기된 프로시듀어를 호출할 수 있습니다. 프로그램 A의 프로시듀어 P1은 다른 복사

된 모듈에 있는 프로시듀어 P2를 호출합니다. 프로그램 C의 프로시듀어 P3은 서비스 프로그램 D에 있는 프로시듀어 P4를 호출합니다.

- 서비스 프로그램의 프로시듀어는 같은 서비스 프로그램 또는 또다른 서비스 프로그램내에 내보내기된 프로시듀어를 호출할 수 있습니다. 서비스 프로그램 B의 프로시듀어 P6은 또다른 복사된 모듈에 있는 프로시듀어 P7을 호출합니다. 서비스 프로그램 E의 프로시듀어 P5는 서비스 프로그램 F에 있는 프로시듀어 P4를 호출합니다.

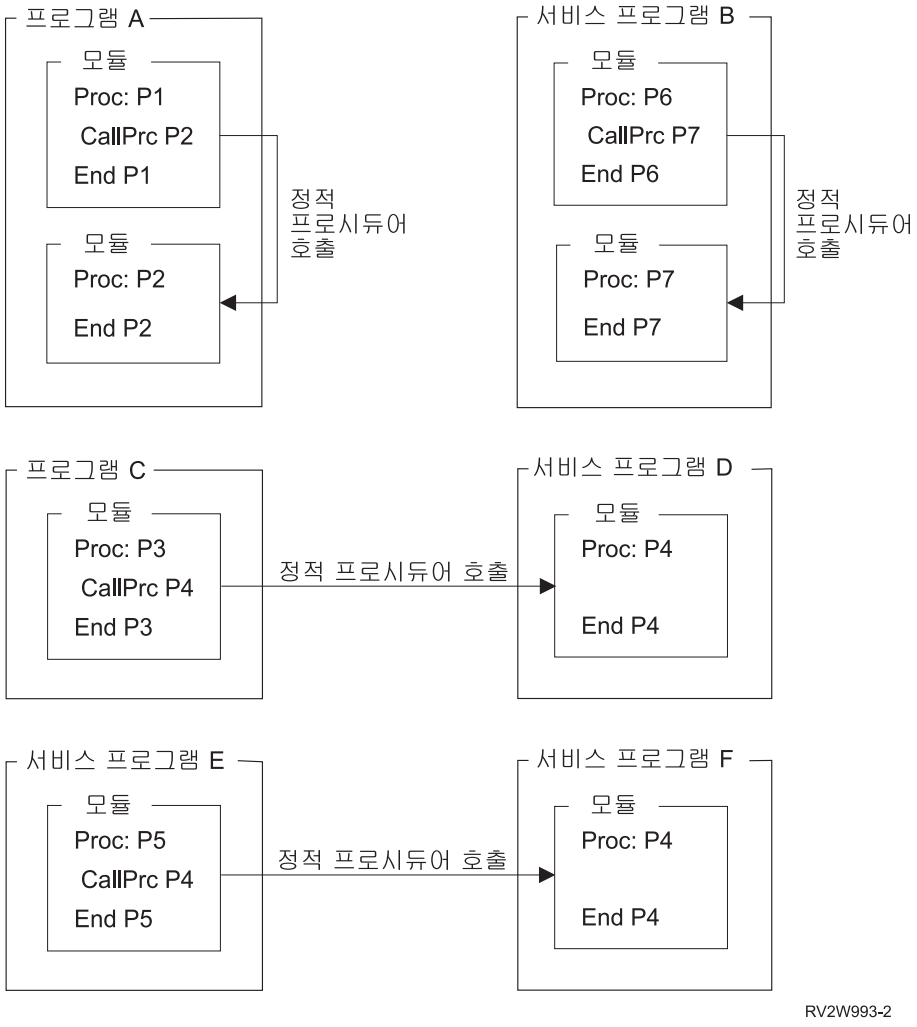

그림 13. 정적 프로시듀어 호출

활성화

ILE 프로그램이 성공적으로 작성되면 코드를 실행시키고자 할 것입니다. 프로그램이나 서비스 프로그램이 실행 준비를 갖추는 것을 **활성화**라고 합니다. 프로그램을 활성화시키기 위해 명령을 발행할 필요는 없습니다. 프로그램이 호출될 때 시스템에 의해 활성화가 수행됩니다. 서비스 프로그램은 호출되지 않기 때문에 서비스에 직접 또는 간접적으로 필요한 프로그램이 호출되는 동안 활성화됩니다.

활성화는 다음과 같은 기능을 수행합니다.

- 프로그램이나 서비스 프로그램에 필요한 정적 자료를 고유하게 할당함
- 내보내기를 제공하는 서비스 프로그램에 대한 기호 링크를 실제 주소로 링크되도록 변경함

프로그램이나 서비스 프로그램을 실행하는 작업 수에 관계 없이 기억장치에는 오브젝트 명령어의 사본 한 개만 존재합니다. 그러나 각 프로그램 활성화에는 자체의 정적 기억장치가 있습니다. 하나의 프로그램 오브젝트가 많은 작업에 의해 동시에 사용될 때에도 정적 변수는 각 활성화에 대해 분리됩니다. 같은 작업 내에서도 한 프로그램이 여러 활성 그룹을 활성화시킬 수 있으나 활성화는 특정 활성 그룹에 대해 로컬로 이루어집니다.

다음 중 어느 하나라도 해당하는 경우:

- 활성화가 필요한 서비스 프로그램을 찾을 수 없습니다.
- 서비스 프로그램이 더이상 서명으로 표시된 프로시듀어 또는 자료 항목을 지원하지 않습니다.

오류가 발생하여 어플리케이션을 실행시킬 수 없습니다.

프로그램 활성화에 관한 자세한 내용은 32 페이지의 『프로그램 활성화 작성』 부분을 참조하십시오.

활성화가 프로그램이 사용하는 정적 변수에 필요한 기억장치를 할당할 때 공간은 활성 그룹에서 할당됩니다. 프로그램이나 서비스 프로그램이 작성될 때 실행시에 사용될 활성 그룹을 지정할 수 있습니다.

활성 그룹에 관한 자세한 정보는 33 페이지의 『활성 그룹』 부분을 참조하십시오.

오류 처리

28 페이지의 그림 14에서는 OPM과 ILE 프로그램 모두를 위한 완벽한 오류 처리 구조를 보여줍니다. 이 그림은 확장된 오류 처리 기능을 설명하기 위해 이 책 전반에 걸쳐 사용됩니다. 이 주제는 표준 언어 오류 처리 기능에 대한 간단한 개요를 제공합니다. 오류 처리에 관한 추가 정보는 45 페이지의 『오류 처리』 부분을 참조하십시오.

그림에서는 예외 메세지 구조라고 하는 기본 계층을 보여줍니다. OPM 프로그램이나 ILE 프로그램이 오류를 만날 때마다 예외 메세지가 시스템에 의해 생성될 수 있습니다. 또한 예외 메세지는 프로그램 오류로 간주되지 않는 상태 정보와 통신하기 위해 사용될 수 있습니다. 예를 들면, 데이터베이스 레코드를 찾을 수 없는 상황은 상태 예외 메세지를 전송하여 통신할 수 있습니다.

각각의 고급 언어는 언어 특정 오류 처리 기능을 정의합니다. 이 기능은 언어에 따라 다르지만 일반적으로 각각의 HLL 사용자가 특정 오류 상황을 처리하려는 의도를 선언하는 것은 가능합니다. 이와 같이 의도를 선언하는 것에는 오류 처리 루틴을 식별하는 것이 포함됩니다. 예외가 발생할 때 시스템은 오류 처리 루틴을 찾아서 사용자 작성 명령어에 제어를 전달합니다. 프로그램 종료 또는 오류에서의 회복 후 계속 등을 포함하여 다양한 조치를 취할 수 있습니다.

그림 14에서는 ILE가 OPM 프로그램에 의해 사용되는 것과 같은 예외-메세지 구조 사용을 보여줍니다. 시스템에 의해 생성된 예외 메세지는 그것이 OPM 프로그램내에서 수행하는 것처럼 ILE 프로그램 내에서 언어 특정 오류 처리를 개시합니다. 그림의 맨 아래 계층에는 사용자가 예외 메세지를 송수신할 수 있는 기능이 포함되어 있습니다. 이는 메세지 핸들러 API 또는 명령으로 수행될 수 있습니다. 예외 메세지는 ILE와 OPM 프로그램 사이에서 송수신될 수 있습니다.

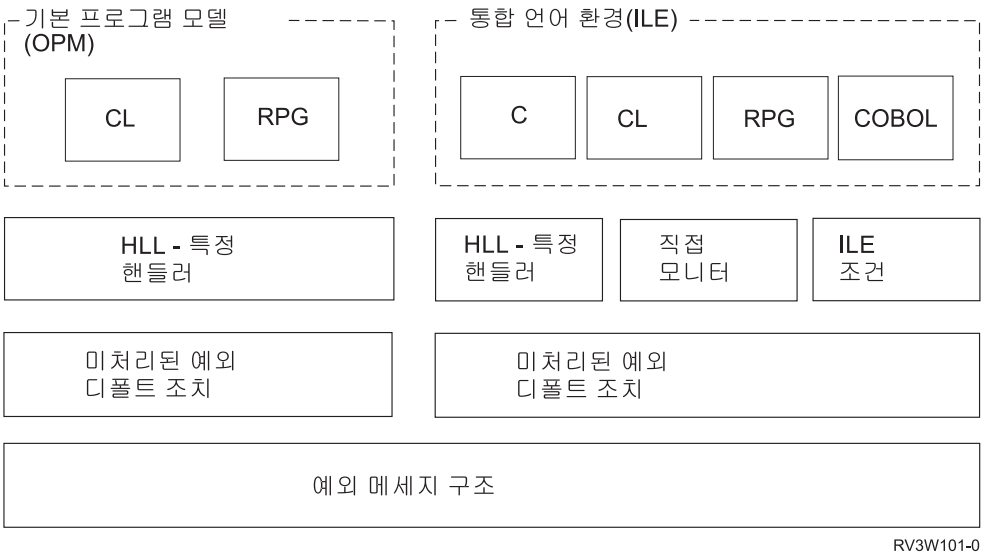

그림 14. OPM 및 ILE를 위한 오류 처리

ILE 프로그램과 OPM 프로그램간 언어 특정 오류 처리 방식은 유사하지만 다음과 같은 기본적인 차이가 있습니다.

- 시스템이 ILE 프로그램에 예외 메세지를 송신할 때는 프로시듀어명과 모듈명이 예외 메세지를 규정하는 데 사용됩니다. 예외 메세지를 송신하면 이들 동일 규정이 지정될 수 있습니다. 예외 메세지가 ILE 프로그램의 작업 기록부에 표시될 때 시스템이 보통은 프로그램명, 모듈명 및 프로시듀어명을 제공합니다.

- ILE 프로그램에 대한 광범위한 최적화로 복수의 HLL문을 동일하게 생성된 명령어와 연관시킬 수 있습니다. 최적화의 결과로 작업 기록부에 표시된 예외 메세지에는 다수의 HLL문이 들어 있을 수 있습니다.

추가 오류 처리 기능은 45 페이지의 『오류 처리』 부분에서 설명합니다.

최적화 변환 프로그램

OS/400에서 **최적화**는 오브젝트의 실행시 성능을 최대화하는 것을 의미합니다. 모든 ILE 언어에는 ILE 최적화 변환 프로그램이 제공하는 최적화 기법에 대한 액세스가 있습니다. 일반적으로, 최적화 요구가 높을수록 오브젝트를 작성하는 데 소요되는 시간은 길어집니다. 실행시, 고도로 최적화된 프로그램 또는 서비스 프로그램은 더 낮은 레벨로 작성된 그에 대응하는 프로그램 또는 서비스 프로그램보다 더 빠르게 실행됩니다.

모듈, 프로그램 오브젝트 및 서비스 프로그램에 최적화를 지정할 수는 있지만 최적화 기법은 개별 모듈에 적용됩니다. 최적화의 레벨은 다음과 같습니다.

10 또는 *NONE

20 또는 *BASIC

30 또는 *FULL

40(레벨 30보다 더 큰 최적화)

실행(in production) 상태에서 모듈을 사용할 때에는 성능상의 이유로 높은 레벨의 최적화를 원할 수 있습니다. 코드가 사용되기를 기대하는 최적화 레벨에서 코드를 테스트하십시오. 모든 것이 예상한 대로 작동하는지 확인한 후 사용자가 코드를 사용할 수 있도록 만드십시오.

레벨 30(*FULL) 또는 레벨 40의 최적화는 프로그램 명령에 상당한 영향을 줄 수 있으므로, 다른 주소지정 예외 검출과 디버깅 제한사항에 대한 주의가 필요합니다. 디버그 고려사항은 145 페이지의 제 10 장 『디버깅 고려사항』 부분을 참조하십시오. 주소지정 오류 고려사항은 225 페이지의 부록 B 『최적화 프로그램의 예외』 부분을 참조하십시오.

디버거

ILE는 소스 레벨 디버깅을 허용하는 디버거를 제공합니다. 디버거는 리스트 파일에 대한 작업을 수행하고 중단점을 설정하고, 변수를 표시하고, 명령어에 개입하거나 건너뛸 수 있게 합니다. 명령 행에 명령을 입력하지 않아도 이를 수행할 수 있습니다. 디버거 작업중에도 명령 행을 사용할 수 있습니다.

소스 레벨 디버거는 시스템 제공 API를 사용하여 프로그램 또는 서비스 프로그램을 디버그할 수 있게 합니다. 이들 API는 모든 사용자에게 사용가능하며 사용자의 디버거를 작성할 수 있게 해줍니다.

OPM 프로그램을 위한 디버거는 계속해서 iSeries 서버에 존재하지만 OPM 프로그램의 디버그에만 사용할 수 있습니다.

최적화된 모듈을 디버그할 때 혼란이 생길 수 있습니다. ILE 디버거를 사용하여 실행 중인 프로그램 또는 프로시듀어가 사용하는 변수를 보거나 변경할 때는 다음이 발생합니다. 디버거가 이러한 변수의 저장 장소에 있는 자료를 검색하거나 갱신합니다. 레벨 20(*BASIC), 30(*FULL) 또는 40의 최적화에서 자료 변수의 현재 값은 하드웨어 레지스터에 있을 수 있지만 디버거는 이 레지스터에서 값을 액세스할 수 없습니다(자료 변수가 하드웨어 레지스터에 있는지의 여부는 몇 가지 요소에 의존합니다. 이들 요소에는 변수의 사용법, 크기 및 코드에서 자료 변수를 조사하거나 변경하는 것을 멈춘 위치 등이 포함됩니다). 따라서, 변수에 대해 표시된 값이 현재 값이 아닐 수 있습니다. 이러한 이유로 개발시에는 10(*NONE)의 최적화 레벨을 사용해야 합니다. 그런 다음, 실행시 최상의 성능을 위해 최적화 레벨을 30(*FULL) 또는 40으로 변경해야 합니다.

ILE 디버거에 관한 자세한 정보는 145 페이지의 제 10 장『디버깅 고려사항』부분을 참조하십시오.

제 3 장 ILE 확장 개념

이 장에서는 ILE 모델의 확장 개념에 대해 설명합니다. 이 부분을 읽기 전에 13 페이지의 제 2 장『ILE 기본 개념』에서 설명하는 개념을 이해하도록 하십시오.

프로그램 활성화

활성화는 프로그램이 실행 준비를 갖추도록 하는 데 사용되는 처리입니다. ILE 프로그램과 ILE 서비스 프로그램은 둘다 시스템에 의해 활성화되어야 실행될 수 있습니다.

프로그램 활성화에는 다음 두 가지 주요 단계가 있습니다.

1. 프로그램의 정적 기억장치를 할당하고 초기설정합니다.
2. 서비스 프로그램에 대한 프로그램의 바인딩을 완료합니다.

여기에서는 40 페이지의『서비스 프로그램 활성화』에 나오는 1단계와 2단계를 중점적으로 설명합니다.

32 페이지의 그림 15에서는 영구 디스크 기억장치에 저장된 두 개의 ILE 프로그램 오브젝트를 보여줍니다. OS/400 오브젝트와 마찬가지로 이 프로그램 오브젝트도 다른 OS/400 작업에서 실행하는 복수 사용자들이 동시에 공유할 수 있습니다. 프로그램 코드의 사본은 하나만 존재합니다. 그러나 이들 ILE 프로그램 중 하나가 호출될 때에는 각 프로그램 활성화를 위해 프로그램 안에 선언된 일부 변수를 반드시 할당 및 초기화시켜야 합니다.

그림 15에 나오는 것처럼 각 프로그램 활성화는 이러한 변수 중 최소한 하나의 고유한 사본을 지원합니다. 하나의 프로그램 활성화 안에는 같은 이름을 가진 여러 사본의 변수가 존재할 수 있습니다. 이것은 사용자의 HLL이 개별 프로시듀어로 범위가 정해진 정적 변수를 선언할 수 있도록 하는 경우에 발생합니다.

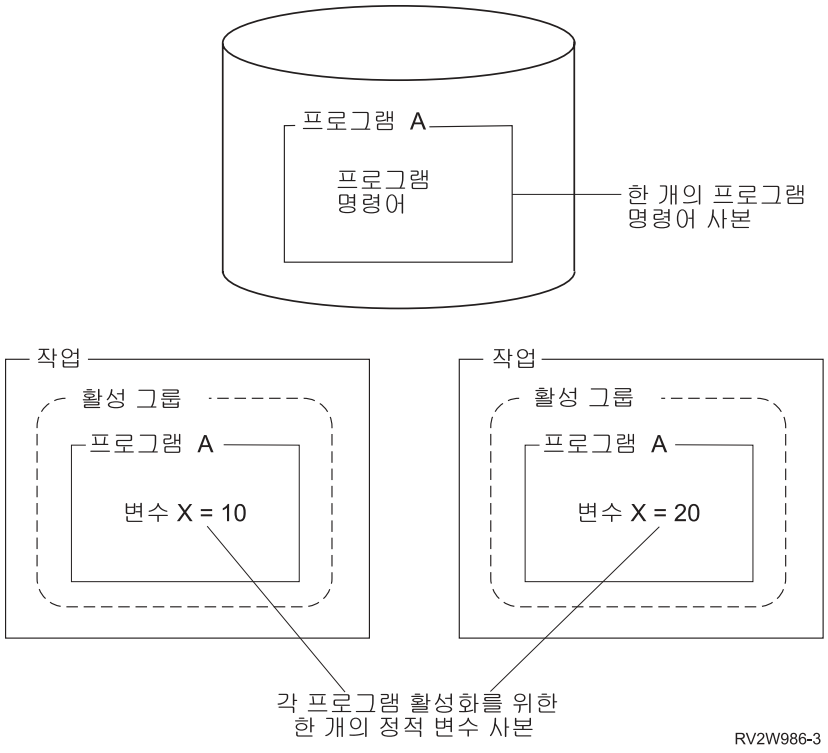

그림 15. 각 프로그램 활성화를 위한 한 개의 정적 변수의 사본

프로그램 활성화 작성

ILE는 활성 그룹내의 프로그램 활성화를 지속적으로 추적하여 프로그램 활성화 처리를 관리합니다. 활성 그룹의 정의는 33 페이지의 『활성 그룹』 부분을 참조하십시오. 활성 그룹에는 특정 프로그램 오브젝트에 대한 하나의 활성화만 존재합니다. 이 규칙이 적용될 경우 다른 OS/400 라이브러리에 상주하는 같은 이름의 프로그램이 다른 프로그램 오브젝트로 간주됩니다.

HLL 프로그램에서 동적 프로그램 호출 명령문을 사용할 때 ILE는 프로그램이 작성될 때 지정된 활성 그룹을 사용합니다. 이 속성은 프로그램 작성(CRTPGM) 명령이나 서비스 프로그램 작성(CRTSRVPGM) 명령에 활성 그룹(ACTGRP) 매개변수를 사용하여 지정됩니다. 프로그램 활성화가 이미 이 매개변수로 지정된 활성 그룹내에 있으면 그것이 사용됩니다. 프로그램이 이 활성 그룹내에서 활성화된 적이 없는 경우에는 먼저 활성화된 후 실행됩니다. 명명된 활성 그룹이 있으면 UPDPGM 및 UPDSRVPGM 명령의 ACTGRP 매개변수로 이름을 변경할 수 있습니다.

프로그램은 일단 활성화되면 활성 그룹이 삭제될 때까지 활성화된 상태로 남아 있습니다. 이와 같은 규칙에 따라 활성 그룹내의 호출 스택상에 없는 활동 프로그램을 소유하는 것이 가능합니다. 33 페이지의 그림 16에서는 활성 그룹 안에 있는 세 개의 활동 프로그램의 예 뿐만 아니라, 세 개의 프로그램 중 단지 두 개에만 호출 스택 프로시듀어

가 있음을 보여줍니다. 이 예에서 프로그램 A는 프로그램 B를 호출하여 프로그램 B가 활성화되도록 합니다. 그런 다음, 프로그램 B가 프로그램 A를 리턴합니다. 또한 프로그램 A는 프로그램 C를 호출합니다. 그 결과 호출 스택에는 프로그램 B용 프로시듀어를 제외한 프로그램 A와 C용 프로시듀어가 포함됩니다. 호출 스택에 대한 논의는 119 페이지의 『호출 스택』 부분을 참조하십시오.

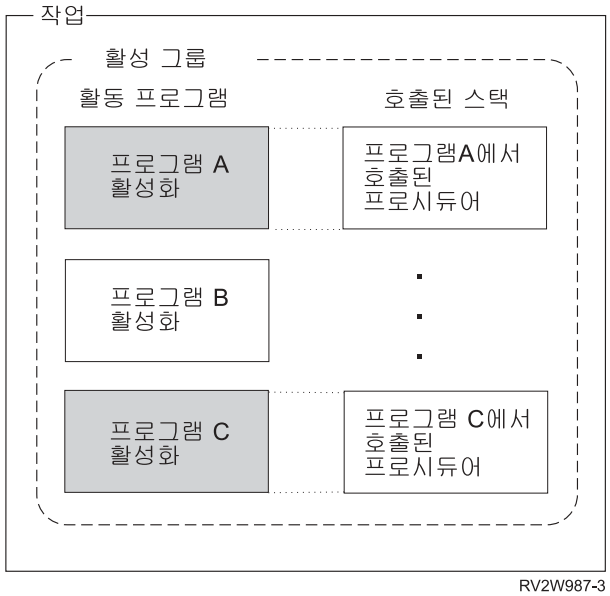

그림 16. 활동중이지만 호출 스택에 없을 수 있는 프로그램

활성 그룹

모든 ILE 프로그램과 서비스 프로그램은 **활성 그룹**이라고 하는 작업의 하부구조 안에서 활성화됩니다. 이 하부구조에는 프로그램을 실행하는 데 필요한 자원이 들어 있습니다. 이들 자원은 다음과 같은 일반적인 범주에 속합니다.

정적 프로그램 변수

동적 기억장치

임시 자료 관리 자원

특정 유형의 예외 핸들러와 종료 프로시듀어

정적 프로그램 변수에 대한 기억장치를 제공하기 위해 활성 그룹은 단일 레벨 저장이나 테라스페이스 중 하나를 사용합니다. 자세한 정보는 57 페이지의 제 4 장 『테라스페이스 및 단일 레벨 저장』을 참조하십시오. 단일 레벨 저장이 사용되면 정적 및 자동 프로그램 변수와 동적 기억장치에는 각 활성 그룹을 위한 분리 주소 공간이 할당되어 우발적인 액세스로부터 어느 정도 프로그램 분리 및 보호를 제공합니다. 테라스페이스가 사

용되면 정적 프로그램 변수와 동적 기억장치에는 테라스페이스내의 각 주소 범위가 할당되어 우발적인 액세스로부터 다소 더 적은 정도의 프로그램 분리 및 보호를 제공합니다.

임시 자료 관리 자원에는 다음이 있습니다.

- 열린 파일(열린 자료 경로 또는 ODP)
- 확약 정의
- 로컬 SQL 커서
- 리모트 SQL 커서
- 계층 파일 시스템(HFS)
- 사용자 인터페이스 관리 프로그램(UIM)
- 조회 관리 인스턴스
- 열린 통신 링크
- 공통 프로그래밍 인터페이스(CPI) 통신

활성 그룹간 이러한 자원의 분리는 기본적인 개념을 지원합니다. 즉, 하나의 활성 그룹 안에서 활성화된 모든 프로그램은 하나의 공통 어플리케이션으로 개발된다는 개념입니다.

소프트웨어 업체들은 다른 활성 그룹을 선택하여 같은 작업에서 실행 중인 다른 업체 어플리케이션에서 프로그램을 분리할 수 있습니다. 이러한 업체 분리는 35 페이지의 그림 17에서 볼 수 있습니다. 이 그림에서는 서로 다른 네 개 업체의 소프트웨어 패키지를 통합하여 완전한 고객 솔루션을 제공합니다. 활성 그룹은 각 업체 패키지와 연관된 자원을 분리하여 통합의 용이성을 증가시킵니다.

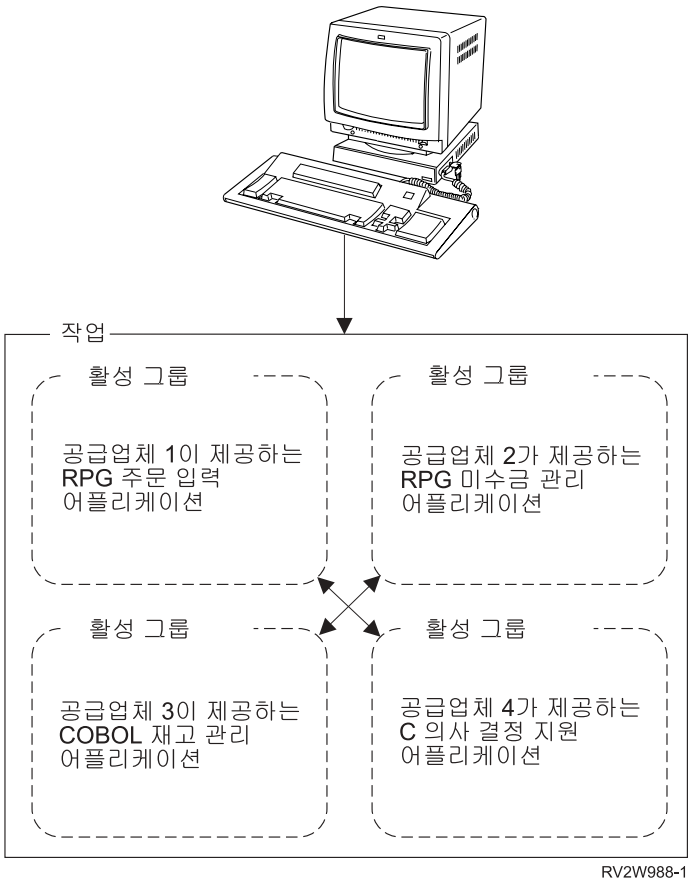

그림 17. 각 공급업체의 어플리케이션을 분리시킨 활성 그룹

위의 자원을 활성 그룹에 할당하면 중요한 결과가 발생합니다. 그것은 활성 그룹이 삭제될 때 위의 모든 자원이 시스템으로 리턴한다는 것입니다. 활성 그룹이 삭제될 때 열린 상태로 있는 임시 자료 관리 자원은 시스템에 의해 닫힙니다. 할당 해제되지 않은 정적 및 자동 프로그램 변수용 기억장치와 동적 기억장치는 시스템으로 리턴합니다.

활성 그룹 작성

프로그램이나 서비스 프로그램을 작성할 때 활성 그룹 속성을 지정하게 되면 ILE 활성 그룹의 실행시 작성을 제어할 수 있습니다. 속성은 CRTPGM 명령 또는 CRTSRVPGM 명령에 ACTGRP 매개변수를 사용하여 지정됩니다. 활성 그룹 작성 명령은 없습니다.

모든 ILE 프로그램에는 다음과 같은 활성 그룹 속성 중 하나가 있습니다.

- 사용자 명명 활성 그룹

  ACTGRP(이름) 매개변수로 지정됨. 이 속성을 사용하면 ILE 프로그램과 ILE 서비스 프로그램의 콜렉션을 하나의 어플리케이션으로 관리할 수 있습니다. 활성 그룹은 처음 요구될 때 작성됩니다. 그런 다음, 같은 활성 그룹명을 지정하는 모든 프로그램과 서비스 프로그램에 의해 사용됩니다.
- 시스템 명명 활성 그룹

CRTPGM 명령에 ACTGRP(*NEW) 매개변수로 지정됨. 이 속성을 사용하면 프로그램이 호출될 때마다 새로운 활성 그룹을 작성할 수 있습니다. ILE는 이 활성 그룹을 위해 이름을 선택합니다. ILE에 의해 할당되는 이름은 작업내에서 고유합니다. 시스템 명명 활성 그룹에 지정되는 이름은 사용자가 사용자 명명 활성 그룹을 위해 선택하는 어떤 이름과도 일치해서는 안됩니다. ILE 서비스 프로그램은 이 속성을 지원하지 않습니다.

- 호출하는 프로그램의 활성 그룹을 사용하기 위한 속성

  ACTGRP(*CALLER) 매개변수로 지정됨. 이 속성을 사용하면 호출하는 프로그램의 활성 그룹내에서 활성화될 ILE 프로그램 또는 ILE 서비스 프로그램을 작성할 수 있습니다. 이 속성이 있는 새로운 활성 그룹은 프로그램 또는 서비스 프로그램이 활성화될 때 절대로 작성되지 않습니다.

- 프로그래밍 언어 및 기억장치 모델에 적합한 활성 그룹을 선택하는 속성

  CRTPGM 명령에 ACTGRP(*ENTMOD) 매개변수로 지정됨. ACTGRP(*ENTMOD)가 지정될 때, ENTMOD 매개변수에 의해 지정되는 프로그램 항목 프로시듀어 모듈이 검사됩니다. 다음 중 하나가 발생합니다.

  – 모듈 속성이 RPGLE 또는 CBLLE인 경우, QILE가 활성 그룹으로 사용됩니다.
  – 모듈 속성이 CLLE이고,
    - STGMDL(*SNGLVL)가 지정되면, QILE가 활성 그룹으로 사용됩니다.
    - STGMDL(*TERASPACE)가 지정되면, QILETS가 활성 그룹으로 사용됩니다.
  – 모듈 속성이 RPGLE, CBLLE 또는 CLLE가 아닌 경우 *NEW가 활성 그룹으로 사용됩니다.

  ACTGRP(*ENTMOD)가 CRTPGM 명령의 매개변수에 대한 디폴트 값입니다.

한 작업내의 모든 활성 그룹에는 하나의 이름이 있습니다. 활성 그룹이 작업내에 있으면 그 그룹은 ILE에 의해 사용되어 해당 이름을 지정하는 프로그램과 서비스 프로그램을 활성화합니다. 이같은 설계의 결과, 중복된 활성 그룹명은 하나의 작업내에 존재할 수 없습니다. 그러나 UPDPGM 및 UPDSRVPGM의 ACTGRP 매개변수를 사용하여 활성 그룹의 이름을 변경할 수는 있습니다.

디폴트 활성 그룹

OS/400 작업이 시작되면 시스템은 다른 모든 OPM 프로그램에서 사용할 두 개의 활성 그룹을 작성합니다. 디폴트 활성 그룹은 정적 프로그램 변수에 대한 단일 레벨 저장을 사용합니다. 사용자는 OPM 디폴트 활성 그룹을 삭제할 수 없습니다. 이들 그룹은 작업이 종료될 때 시스템에 의해 삭제됩니다.

ILE 프로그램 및 ILE 서비스 프로그램은 다음 조건이 충족되는 경우 OPM 디폴트 활성 그룹에서 활성화될 수 있습니다.

- ILE 프로그램 또는 ILE 서비스 프로그램이 활성 그룹 *CALLER 옵션으로 작성된 경우
- ILE 프로그램 또는 ILE 서비스 프로그램에 대한 호출이 OPM 디폴트 활성 그룹에서 시작된 경우
- ILE 프로그램 또는 서비스 프로그램은 테라스페이스 기억장치 모델을 사용하지 않는 경우

디폴트 활성 그룹을 삭제할 수 없기 때문에 ILE HLL 종료 동사가 완전한 종료 처리를 제공할 수 없습니다. 열린 파일은 작업이 종료될 때까지 시스템이 닫지 못합니다. ILE 프로그램이 사용하는 정적 기억장치와 힙 기억장치는 작업이 종료할 때까지 시스템으로 리턴할 수 없습니다.

그림 18에서는 ILE 활성 그룹과 OPM 디폴트 활성 그룹을 가진 일반적인 OS/400 작업을 보여줍니다. 두 개의 OPM 디폴트 활성 그룹은 두 그룹을 나타내기 위해 사용된 특수 값 *DFTACTGRP로 인해 조합됩니다. 각 활성 그룹내의 상자는 프로그램 활성화를 나타냅니다.

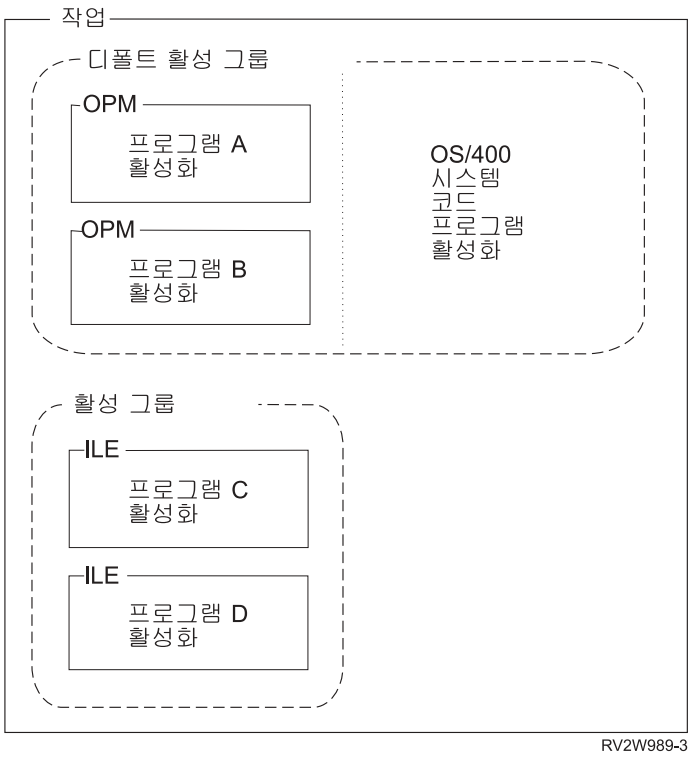

그림 18. 디폴트 활성 그룹과 ILE 활성 그룹

ILE 활성 그룹 삭제

활성 그룹은 자원이 작업내에서 작성될 것을 요구합니다. 활성 그룹이 어플리케이션에 의해 재사용될 수 있는 경우에는 처리 시간이 절약됩니다. ILE는 활성 그룹을 종료하거나 삭제하지 않고도 활성 그룹에서 리턴할 수 있는 몇 개의 옵션을 제공합니다. 활성 그룹의 삭제 여부는 활성 그룹의 유형과 어플리케이션이 종료되는 방법에 달려 있습니다.

어플리케이션은 다음과 같은 방식으로 활성 그룹에 남아서 다른 활성 그룹에서 실행 중인 호출 스택 항목(119 페이지의 『호출 스택』 참조)으로 리턴합니다.

- HLL 종료 동사

  예를 들면, COBOL의 STOP RUN 또는 C의 exit()
- 미처리 예외

  미처리 예외는 시스템에 의해 다른 활성 그룹의 호출 스택 항목으로 이동될 수 있습니다.
- 언어별 HLL 리턴 명령문

  예를 들면, C의 RETURN 명령문, COBOL의 EXIT PROGRAM 명령문 또는 RPG의 RETURN 명령문
- 건너뜀 조작

  예를 들면, 예외 메세지를 보내거나 활성 그룹에 없는 호출 스택 항목으로 분기하는 것

HLL 종료 동사를 사용하여 어플리케이션에서 활성 그룹을 삭제할 수 있습니다. 또한 미처리 예외는 활성 그룹을 삭제되도록 할 수 있습니다. 이러한 조작은 가장 가까운 제어 경계가 활성 그룹에서 가장 오래된 호출 스택 항목인 경우(때로는 하드 제어 경계라고 함)에는 항상 활성 그룹을 삭제합니다. 가장 가까운 제어 경계가 가장 오래된 호출 스택 항목이 아닌 경우(때로는 소프트 제어 경계라고 함), 제어는 제어 경계 앞의 호출 스택으로 전달됩니다. 그러나 활성 그룹은 삭제되지 않습니다.

제어 경계는 어플리케이션에 경계를 나타내는 호출 스택 항목입니다. ILE는 사용자가 활성 그룹간을 호출할 때마다 제어 경계를 정의합니다. 제어 경계의 정의는 42 페이지의 『제어 경계』 부분을 참조하십시오.

사용자 명명 활성 그룹은 나중에 사용될 수 있도록 작업에 남아 있을 수 있습니다. 이와 같은 유형의 활성 그룹의 경우 정상 리턴이나 하드 제어 경계를 지나친 건너뜀 조작은 활성 그룹을 삭제하지 않습니다. 시스템 명명 활성 그룹내에서 사용되는 같은 조작은 활성 그룹을 삭제합니다. 시스템 명명 활성 그룹은 시스템이 생성한 이름을 지정하여 재사용할 수 없기 때문에 항상 삭제됩니다. 활성 그룹의 가장 오래된 호출 스택 항목에서의 정상 리턴에 관한 언어 종속 규칙은 ILE HLL 프로그래머 안내서를 참조하십시오.

그림 19에서는 활성 그룹을 남겨두는 방법에 대한 예를 보여줍니다. 이 그림에서는 프로시듀어 P1이 가장 오래된 호출 스택 항목입니다. 시스템 명명 활성 그룹 (ACTGRP(*NEW) 옵션으로 작성된)의 경우 P1에서의 정상 리턴이 활성 그룹을 삭제합니다. 사용자 명명 활성 그룹(ACTGRP(이름) 옵션으로 작성된)의 경우 P1에서의 정상 리턴이 활성 그룹을 삭제하지 않습니다.

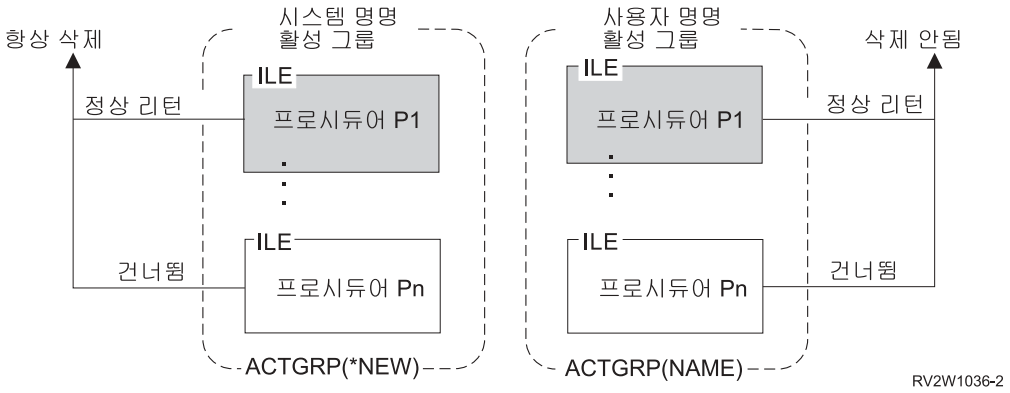

그림 19. 사용자 명명 활성 그룹 및 시스템 명명 활성 그룹 남김

사용자 명명 활성 그룹이 작업에 남아 있으면 활성 그룹 재생(RCLACTGRP) 명령을 사용하여 삭제할 수 있습니다. 이 명령을 사용하면 어플리케이션이 리턴한 후 명명된 활성 그룹을 삭제할 수 있습니다. 사용 중이 아닌 활성 그룹만 이 명령으로 삭제할 수 있습니다.

40 페이지의 그림 20에서는 현재 사용되지 않는 하나의 활성 그룹과 현재 사용되는 하나의 활성 그룹을 가진 OS/400 작업을 보여줍니다. 활성 그룹은 활성 그룹내에서 활성화된 ILE 프로시듀어용 호출 스택 항목이 있는 경우 사용 중으로 간주됩니다. 프로그램 A 또는 프로그램 B에서 RCLACTGRP 명령을 사용하면 프로그램 C와 프로그램 D의 활성 그룹이 삭제됩니다.

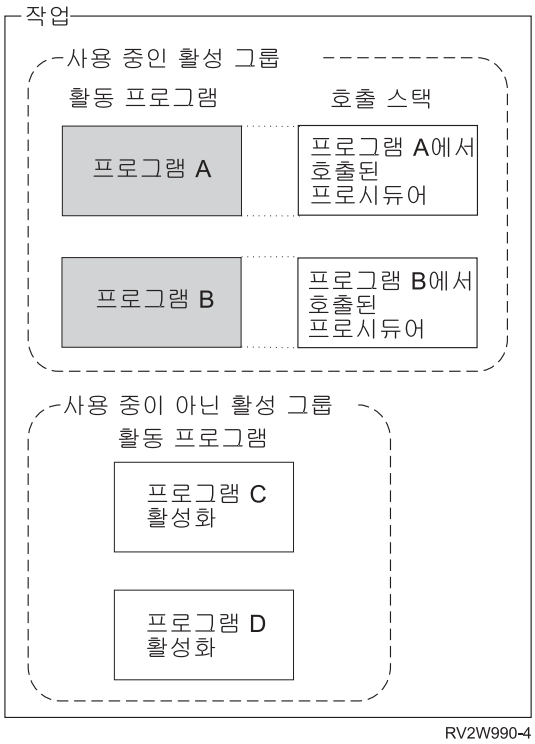

그림 20. 호출 스택 항목이 있는 사용 중인 활성 그룹

활성 그룹이 ILE에 의해 삭제될 때 특정 종료 조작 처리가 발생합니다. 이 처리에는 사용자가 등록한 나감 프로시듀어 호출, 자료 관리 지우기 및 언어 지우기(파일을 닫는 것과 같은)가 포함됩니다. 활성 그룹이 삭제될 때 발생하는 자료 관리 처리에 관한 자세한 내용은 53 페이지의 『자료 관리 범위지정 규칙』 부분을 참조하십시오.

서비스 프로그램 활성화

여기에서는 시스템이 서비스 프로그램을 활성화하기 위해 사용하는 고유한 단계를 논의합니다. 프로그램과 서비스 프로그램에 사용되는 공통적인 단계는 31 페이지의 『프로그램 활성화』 부분에서 설명합니다. 다음의 활성화 작업은 서비스 프로그램마다 고유합니다.

- 서비스 프로그램 활성화는 ILE 프로그램에 대한 동적 프로그램 호출의 일부로서 간접적으로 시작됩니다.
- 서비스 프로그램 활성화에는 기호 링크를 실제 링크에 맵핑하여 프로그램간 바인딩 연결을 완성하는 것이 포함됩니다.
- 서비스 프로그램 활성화에는 서명 검사 처리가 포함됩니다.

활성 그룹내에서 처음으로 활성화된 ILE 프로그램은 ILE 서비스 프로그램으로의 바인딩을 위해 검사됩니다. 서비스 프로그램이 활성화중인 프로그램에 바인드된 경우에는 동일한 동적 호출 처리의 일부로도 활성화됩니다. 이 처리는 필요한 모든 서비스 프로그램이 활성화될 때까지 반복됩니다.

그림 21에서는 ILE 프로그램 A가 ILE 서비스 프로그램 B, C, D에 바인드되는 것을 보여줍니다. 또한 ILE 서비스 프로그램 B와 C도 ILE 서비스 프로그램 E로 바인드됩니다. 각 프로그램과 서비스 프로그램을 위한 활성 그룹 속성이 표시됩니다.

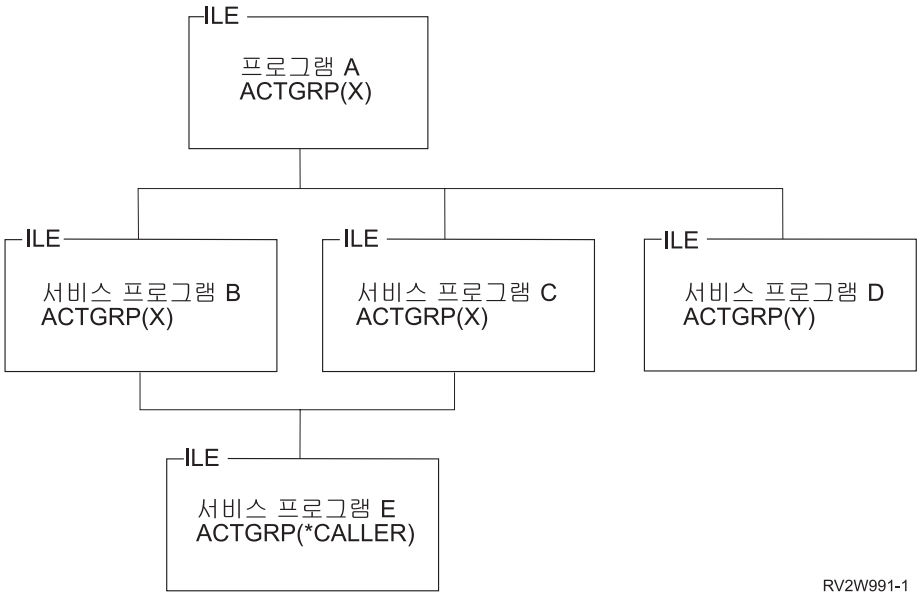

그림 21. 서비스 프로그램 활성화

ILE 프로그램 A가 활성화될 때 다음 사항이 발생합니다.

- 서비스 프로그램은 명시적 라이브러리명이나 현재 라이브러리 리스트를 사용하여 위치가 정해집니다. 이 옵션은 프로그램과 서비스 프로그램이 작성될 때 사용자에 의해 제어됩니다.
- 프로그램과 마찬가지로 서비스 프로그램 활성화도 활성 그룹내에서 한 번만 발생합니다. 그림 21에서는 서비스 프로그램 E가 서비스 프로그램 B와 C에서 사용되더라도 단 한 번만 활성화됩니다.
- 두 번째 활성 그룹(Y)은 서비스 프로그램 D용으로 작성됩니다.
- 서명 검사는 모든 프로그램과 서비스 프로그램간에 발생합니다.

개념적으로 이 처리는 프로그램과 서비스 프로그램이 작성될 때 시작되는 바인딩 처리의 완료로 볼 수 있습니다. CRTPGM 명령과 CRTSRVPGM 명령이 참조되는 각 서비스 프로그램의 이름과 라이브러리를 저장했습니다. 내보내기된 프로시듀어와 자료 항목의 표에 있는 색인도 프로그램 작성시 클라이언트 프로그램이나 서비스 프로그램에 저장되었습니다. 서비스 프로그램 활성화의 처리는 이러한 기호 참조를 실행시에 사용될 수 있는 주소로 변경하여 바인딩 단계를 완료합니다.

일단 서비스 프로그램이 활성화되면 다른 서비스 프로그램내의 모듈에 대한 정적 프로시듀어 호출과 정적 자료 항목 참조가 처리됩니다. 처리량은 모듈이 같은 프로그램 안

으로 복사하여 바인드되는 경우 요구되는 것과 같습니다. 그러나 복사에 의해 바인드되는 모듈에는 서비스 프로그램보다 적은 활성화 시간 처리가 필요합니다.

프로그램과 서비스 프로그램의 활성화에는 ILE 프로그램과 모든 ILE 서비스 프로그램 오브젝트에 대한 실행 권한이 필요합니다. 41 페이지의 그림 21에서는 프로그램 A 호출자의 현재 권한이 프로그램 A와 모든 서비스 프로그램 권한을 검사하는 데 사용됩니다. 또한 프로그램 A의 권한도 모든 서비스 프로그램에 대한 권한을 검사하는 데 사용됩니다. 서비스 프로그램 B, C, D의 권한은 서비스 프로그램 E에 대한 권한을 검사하는 데 사용되지 않는다는 것에 유의하십시오.

제어 경계

ILE는 미처리 기능 검사가 발생하거나 HLL 종료 동사가 사용될 때 다음과 같은 조치를 취합니다. ILE는 사용자 어플리케이션의 경계를 나타내는 호출 스택 항목의 호출자에게 제어를 전달합니다. 이 호출 스택 항목은 **제어 경계**로서 알려집니다.

제어 경계에는 두 가지 정의가 있습니다. 『ILE 활성 그룹을 위한 제어 경계』 및 43 페이지의 『OPM 디폴트 활성 그룹을 위한 제어 경계』에서 다음 정의를 설명합니다.

제어 경계는 다음 중 하나일 수 있습니다.

- 바로 앞의 호출 스택 항목이 디폴트가 아닌 다른 활성 그룹에 있는 ILE 호출 스택 항목입니다.
- 바로 앞의 호출 스택 항목이 OPM 프로그램인 ILE 호출 스택 항목입니다.

ILE 활성 그룹을 위한 제어 경계

이 예에서는 제어 경계가 ILE 활성 그룹 사이에서 어떻게 정의되는지를 보여줍니다.

43 페이지의 그림 22에서는 다양한 호출에 의해 이루어진 두 개의 ILE 활성 그룹과 제어 경계를 보여줍니다. 프로시듀어 P2, P3 및 P6은 잠재 제어 경계입니다. 예를 들어, 프로시듀어 P7에서 실행 중일 때는 프로시듀어 P6이 제어 경계입니다. 프로시듀어 P4나 P5에서 실행 중일 때는 프로시듀어 P3이 제어 경계입니다.

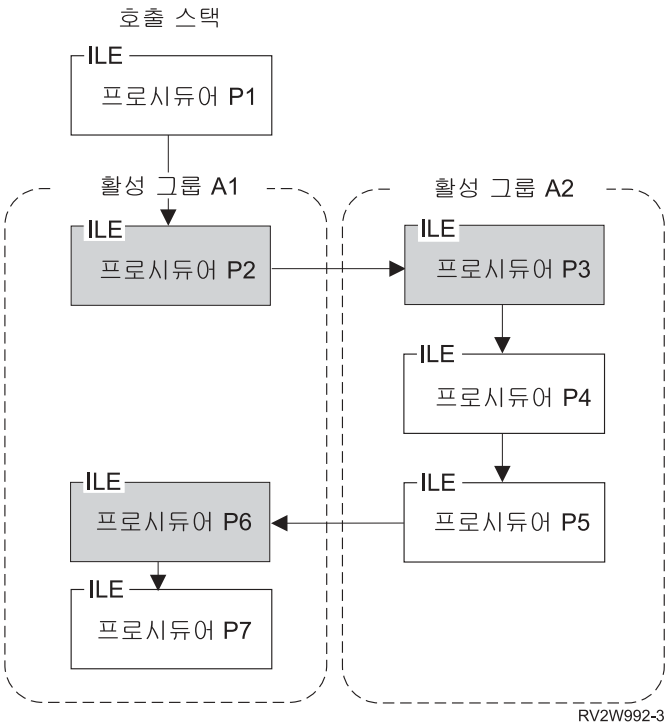

그림 22. 제어 경계. 명암 처리된 프로시듀어가 제어 경계입니다.

OPM 디폴트 활성 그룹을 위한 제어 경계

이 예에서는 ILE 프로그램이 OPM 디폴트 활성 그룹에서 실행 중일 때 제어 경계가 정의되는 방법을 보여줍니다.

44 페이지의 그림 23에서는 OPM 디폴트 활성 그룹에서 실행되는 세 개의 ILE 프로시듀어(P1, P2, P3)를 보여줍니다. 이 예는 ACTGRP(*CALLER) 매개변수 값과 함께 CRTPGM 명령 또는 CRTSRVPGM 명령을 사용하여 작성되었을 수 있습니다. 프로시듀어 P1과 P3은 선행 호출 스택 항목이 OPM 프로그램 A와 B이므로 잠재 제어 경계입니다.

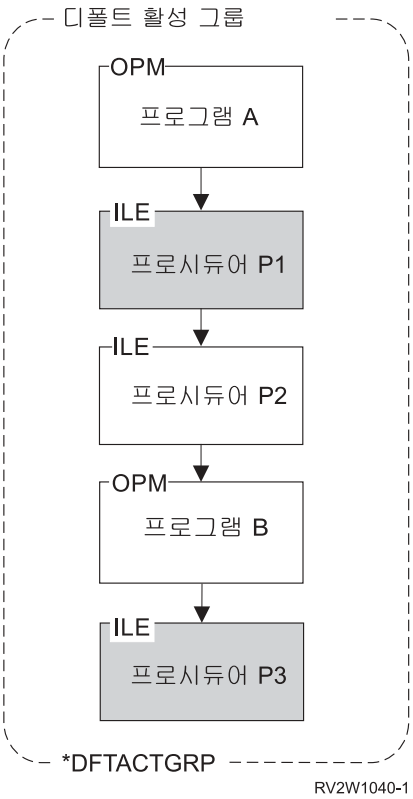

그림 23. 디폴트 활성 그룹내의 제어 경계. 명암 처리된 프로시듀어가 제어 경계입니다.

제어 경계 사용

ILE HLL 종료 동사를 사용할 때 ILE는 호출 스택에 있는 가장 최신 제어 경계를 사용하여 제어를 전송할 곳을 판별합니다. 제어 경계 바로 앞에 있는 호출 스택 항목은 ILE가 모든 종료 처리를 완료한 후 제어를 수신합니다.

제어 경계는 미처리 기능 검사가 ILE 프로시듀어내에서 발생할 때 사용됩니다. 제어 경계는 미처리 기능 검사가 일반적인 ILE 실패 조건으로 **승격되는** 호출 스택상의 위치를 정의합니다. 추가 정보는 45 페이지의 『오류 처리』 부분을 참조하십시오.

가장 가까운 제어 경계가 ILE 활성 그룹에서 가장 오래된 호출 스택 항목인 경우 HLL 종료 동사나 미처리 기능 검사는 활성 그룹이 삭제되도록 합니다. 가장 가까운 제어 경계가 ILE 활성 그룹에서 가장 오래된 호출 스택 항목이 아닌 경우 제어는 제어 경계 바로 앞의 호출 스택 항목으로 리턴합니다. 활성 그룹은 보다 앞선 호출 스택 항목이 같은 활성 그룹내에 있기 때문에 삭제되지 않습니다.

43 페이지의 그림 22에서는 활성 그룹에서 가장 오래된 호출 스택 항목으로서 프로시듀어 P2와 P3을 보여줍니다. 프로시듀어 P2, P3, P4 또는 P5(P6 또는 P7이 아닌)내의 HLL 종료 동사를 사용하면 활성 그룹 A2를 삭제하게 됩니다.

오류 처리

여기에서는 OPM과 ILE 프로그램의 향상된 오류 처리 기능을 설명합니다. 이들 기능이 예외 메세지 구조와 어떻게 연결되는지를 이해하려면 그림 24를 참조하십시오. 특정 참조 정보와 추가적인 개념은 135 페이지의 제 9 장『예외 및 조건 관리』부분을 참조하십시오. 그림 24는 오류 처리의 개요를 보여줍니다. 본문에서 설명하고 있는 내용은 이 그림의 맨 아래 계층에서 시작하여 맨 위 계층까지 계속 연결됩니다. 맨 위 계층은 OPM 또는 ILE 프로그램에서 오류를 처리하는 데 사용할 수 있는 기능을 나타냅니다.

그림 24. ILE 및 OPM 오류 처리

작업 메세지 대기행렬

메세지 대기행렬은 각 OS/400 작업내 모든 호출 스택에 대해 존재합니다. 이 메세지 대기행렬은 호출 스택에서 실행 중인 프로그램과 프로시듀어간에 정보용 메세지와 예외 메세지의 송수신을 용이하게 합니다. 메세지 대기행렬은 **호출 메세지 대기행렬**을 말합니다.

호출 메세지 대기행렬은 호출 스택에 있는 OPM 프로그램이나 ILE 프로시듀어의 이름에 의해 식별됩니다. 프로시듀어명이나 프로그램명은 송신하는 메세지의 목표 호출 스택 항목을 지정하기 위해 사용될 수 있습니다. ILE 프로시듀어명은 고유하지 않으며 ILE 모듈명과 ILE 프로그램 또는 서비스 프로그램명은 선택적으로 지정될 수 있습니다. 같은 프로그램이나 프로시듀어에 여러 개의 호출 스택 항목이 있을 경우에는 가장 가까운 호출 메세지 대기행렬이 사용됩니다.

호출 메세지 대기행렬 외에도 각 OS/400 작업마다 하나의 **외부 메세지 대기행렬**이 있습니다. 작업내에서 실행 중인 모든 프로그램과 프로시듀어는 이 대기행렬을 사용하여 대화식 작업과 워크스테이션 사용자간에 메세지를 주고받을 수 있습니다.

IBM에는 메세지 처리 API를 사용하여 예외 메세지를 수신하는 방법에 관한 온라인 정보가 있습니다. iSeries Information Center에 대한 **프로그래밍** 범주의 *API* 절을 참조하십시오.

예외 메세지와 송신 방법

여기에서는 기타 예외 메세지 유형과 예외 메세지가 송신되는 방법을 설명합니다.

ILE와 OPM의 오류 처리는 예외 메세지 유형에 기초하고 있습니다. 달리 규정하지 않는 한, **예외 메세지**라는 용어는 아래 메세지 유형을 나타냅니다.

이탈(*ESCAPE)

프로그램이 작업을 완료하지 못하고 비정상적으로 종료되도록 하는 오류를 나타냅니다. 이탈 예외 메세지를 송신한 후 제어를 수신하지 못합니다.

상태(*STATUS)

프로그램에 의해 수행 중인 작업의 상태를 설명합니다. 이 메세지 유형을 송신한 후 제어를 수신할 수 있습니다. 제어의 수신 여부는 수신 프로그램이 상태 메시지를 처리하는 방식에 달려 있습니다.

통지(*NOTIFY)

정정 조치나 호출하는 프로그램에서의 응답을 요구하는 조건을 설명합니다. 이 메세지 유형을 송신한 후 제어를 수신할 수 있습니다. 제어의 수신 여부는 수신 프로그램이 통지 메시지를 처리하는 방식에 달려 있습니다.

기능 검사

프로그램이 예상하지 않은 종료 조건을 설명합니다. ILE 기능 검사, CEE9901은 시스템에 의해서만 송신되는 특수 메세지 유형입니다. OPM 기능 검사는 메세지 ID CPF9999가 있는 이탈 메세지 유형입니다.

IBM에는 이와 같은 메세지 유형과 다른 OS/400 메세지 유형에 관한 온라인 정보가 있습니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

예외 메세지는 다음과 같은 방식으로 송신됩니다.

- 시스템에 의해 생성됨

  OS/400(HLL 포함)에서 프로그래밍 오류나 상태 정보를 나타내는 예외 메세지를 생성합니다.

- 메세지 핸들러 API

  프로그램 메세지 송신(QMHSNDPM) API는 특정 호출 메세지 대기행렬에 예외 메세지를 송신하는 데 사용될 수 있습니다.

- ILE API

  조건 신호(CEESGL) 바인드 가능 API는 ILE 조건 제기에 사용될 수 있습니다. 이 조건은 결과적으로 이탈 예외 메세지나 상태 예외 메세지를 만듭니다.

- 언어별 고유 동사

  ILE C 및 ILE C++의 경우 raise() 함수가 C 신호를 생성합니다. ILE RPG 또는 ILE COBOL 중 어느 것도 유사한 함수를 갖지 않습니다.

예외 메세지 처리 방법

사용자나 시스템이 예외 메세지를 송신할 때 예외 처리가 시작됩니다. 이 처리는 예외가 처리되고 예외 메세지가 수정되어 처리가 완료되었음을 나타낼 때까지 계속됩니다.

시스템은 예외 메세지를 수정하여 OPM 호출 메세지 대기행렬에 대한 예외 핸들러를 호출할 때 예외 메세지가 처리되었음을 나타냅니다. 사용자의 ILE HLL은 예외 핸들러가 ILE 호출 메세지 대기행렬을 위해 호출되기 전에 예외 메세지를 수정합니다. 결과적으로, HLL 특정 오류 처리는 사용자의 핸들러가 호출될 때 예외 메세지가 처리되는 것으로 간주합니다. HLL 특정 오류 핸들러를 사용하지 않을 경우 사용자의 ILE HLL이 예외 메세지를 처리하거나 예외 처리가 계속되도록 할 수 있습니다. 미처리 예외 메세지에 대한 사용자의 HLL 디폴트 조치를 판별하려면 ILE HLL 참조 설명서를 참조하십시오.

ILE에 정의된 추가 기능을 통해 언어별로 오류 핸들러를 바이패스할 수 있습니다. 이와 같은 기능에는 직접 모니터 핸들러와 ILE 조건 핸들러가 포함됩니다. 이러한 기능을 사용할 경우 예외 메세지가 처리되었음을 나타내기 위해 사용자가 예외 메세지를 변경시켜야 합니다. 사용자가 예외 메세지를 변경하지 않으면 시스템은 다른 예외 핸들러를 찾아 예외 처리를 계속합니다. 49 페이지의 『예외 핸들러의 유형』에는 직접 모니터 핸들러와 ILE 조건 핸들러에 대한 자세한 설명이 포함되어 있습니다. IBM에서는 예외 메세지의 변경 방법을 설명하는 온라인 정보를 제공합니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절에 있는 예외 메세지 변경(QMHCHGEM) API를 참조하십시오.

예외 회복

사용자는 예외가 송신된 후에도 처리를 계속할 수 있습니다. 오류에서 회복하는 것은 오류를 처리하는 어플리케이션을 전달할 수 있게 하는 유용한 어플리케이션 툴이 될 수 있습니다. ILE 및 OPM 프로그램의 경우 시스템이 **재개점**의 개념을 정의합니다. 재개점은 초기에 예외 발생에 이어 계속되는 명령어에 설정됩니다. 사용자는 예외를 처리한 후 재개점에서 처리를 계속할 수 있습니다. 재개점을 사용하고 수정하는 방법에 관해서는 135 페이지의 제 9 장 『예외 및 조건 관리』 부분을 참조하십시오.

미처리 예외에 대한 디폴트 조치

HLL에서 예외 메세지를 처리하지 않으면 시스템은 미처리 예외에 대해 디폴트 조치를 취합니다.

45 페이지의 그림 24에서는 예외가 OPM 또는 ILE 프로그램으로 송신되었는지에 기초하여 미처리 예외를 위한 디폴트 조치를 보여줍니다. OPM과 ILE에 대한 기타 디폴트 조치는 오류 처리 기능의 근본적인 차이를 나타냅니다.

OPM의 경우 미처리 예외는 기능 검사 메세지라고 하는 특수 이탈 메세지를 생성합니다. 이 메세지에는 특수 메세지 ID의 CPF9999가 부여됩니다. 이 메세지는 원래 예외 메세지를 초래한 호출 스택 항목의 호출 메세지 대기행렬에 송신됩니다. 기능 검사 메세지가 처리되지 않으면 시스템은 해당 호출 스택 항목을 제거합니다. 그런 다음, 시스템은 기능 검사 메세지를 이전 호출 스택 항목으로 송신합니다. 이 처리는 기능 검사 메세지가 처리될 때까지 계속됩니다. 기능 검사 메세지가 처리되지 않는 경우 작업이 종료됩니다.

ILE의 경우 미처리 예외 메세지는 이전 호출 스택 항목 메세지 대기행렬에서 여과됩니다. 예외 메세지가 이전 호출 메세지 대기행렬로 이동될 때 **여과(percolation)**가 발생합니다. 이것은 같은 예외 메세지를 이전 호출 메세지 대기행렬로 송신하는 결과를 만듭니다. 이러한 상황이 발생하면 예외 처리는 이전 호출 스택 항목에서 계속됩니다.

49 페이지의 그림 25에서는 ILE내의 미처리 예외 메세지를 보여줍니다. 이 예에서는 프로시듀어 P1이 제어 경계입니다. 프로시듀어 P1은 또한 활성 그룹에서 가장 오래된 호출 스택 항목이기도 합니다. 프로시듀어 P4는 미처리된 예외 메세지를 초래합니다. 미처리 예외 메세지의 여과는 제어 경계에 도달하거나 예외 메세지가 처리될 때까지 계속됩니다. 미처리 예외는 제어 경계로 여과될 때 기능 검사로 변환됩니다. 예외가 이탈인 경우 기능 검사가 생성됩니다. 예외가 통지 예외이면 디폴트 응답이 송신되고, 예외가 처리되며, 통지 송신자에게는 처리를 계속하도록 허용됩니다. 그것이 상태 예외이면 예외가 처리되고 상태 송신자에게는 처리를 계속하도록 허용됩니다. 재개점(프로시듀어 P3에 나타나는)은 기능 검사의 예외 처리가 계속되어야 하는 호출 스택 항목을 정의하는 데 사용됩니다. ILE의 경우 다음 처리 단계는 이 호출 스택 항목에 특수 기능 검사 예외 메세지를 송신하는 것입니다. 이 예에서 이것은 프로시듀어 P3입니다.

이제, 기능 검사 예외 메세지가 처리되거나 제어 경계로 여과될 수 있습니다. 기능 검사 예외 메세지가 처리되면 정상 처리가 계속되고 예외 처리는 종료됩니다. 기능 검사 메세지가 제어 경계로 여과되면 ILE는 어플리케이션이 예기치 않은 오류로 인해 종료되었다고 간주합니다. 일반적인 실패 예외 메세지는 모든 언어에 대한 ILE에 의해 정의됩니다. 이 메세지는 CEE9901이며 ILE에 의해 제어 경계의 호출자로 송신됩니다.

ILE에 정의된 미처리 예외 메세지에 대한 디폴트 조치를 통해 사용자는 혼합 언어 어플리케이션내에서 발생하는 오류 상태에서 회복할 수 있습니다. 예기치 않은 오류의 경

우 ILE는 모든 언어에 대해 일관된 실패 메세지를 강제합니다. 이것은 다른 소스에서 어플리케이션을 통합하기 위한 능력을 향상시킵니다.

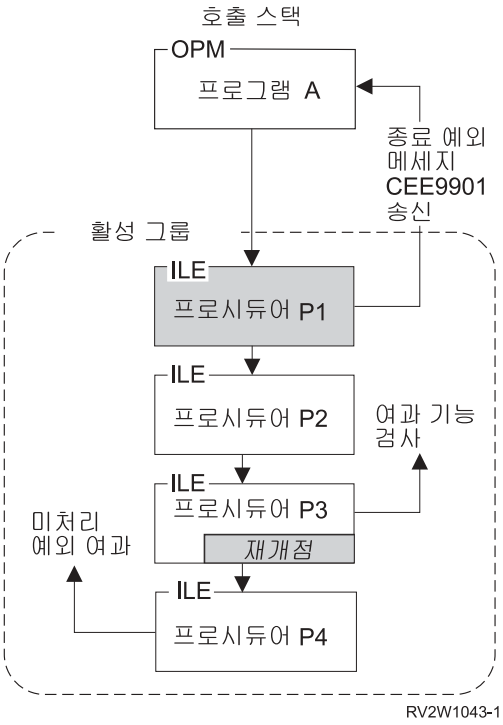

그림 25. 미처리 예외 디폴트 조치

예외 핸들러의 유형

여기에서는 OPM 및 ILE 프로그램 둘다에 제공되는 예외 핸들러 유형의 개요를 제공합니다. 45 페이지의 그림 24에 나오는 것처럼 이것은 예외 메세지 구조의 맨 위 계층입니다. ILE는 OPM과 비교해 볼 때 추가 예외 처리 기능을 제공합니다.

OPM 프로그램의 경우 HLL 특정 오류 처리는 각 호출 스택 항목에 대해 하나 이상의 처리 루틴을 제공합니다. 해당 루틴은 예외가 OPM 프로그램에 송신될 때 시스템에 의해 호출됩니다.

ILE에 있어서 HLL 특정 오류 처리는 같은 기능을 제공합니다. 그러나 ILE에는 추가 유형의 예외 핸들러가 있습니다. 이러한 유형의 핸들러는 사용자에게 예외 메세지 구조의 직접적인 제어를 제공하며 HLL 특정 오류 처리를 바이패스할 수 있게 합니다. ILE를 위한 추가 핸들러 유형은 다음과 같습니다.

직접 모니터 핸들러
ILE 조건 핸들러

이러한 유형의 핸들러가 HLL에 의해 지원되는지를 판별하려면 ILE HLL 프로그래머 안내서를 참조하십시오.

직접 모니터 핸들러는 제한된 HLL 소스 명령문 주위에 예외 모니터를 직접 선언할 수 있게 합니다. ILE C의 경우 이 기능은 #pragma 지시문을 통해 가능합니다. ILE COBOL은 ILE C가 수행하는 것과 같은 맥락에서 제한된 HLL 소스 명령문의 주변에는 예외 모니터를 직접 선언하지 않습니다. ILE COBOL 프로그램은 임의의 소스 코드 주변에 핸들러를 작동가능하게 만들거나 작동불가능하게 만드는 코드를 직접 코딩할 수 없습니다. 그러나

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

와 같은 명령문은 내부적으로 같은 메커니즘을 사용하도록 맵핑됩니다. 그러므로 핸들러가 먼저 제어를 얻는 우선순위의 견지에서 명령문으로 범위가 정해지는 그러한 조건 명령문은 ILE 조건 핸들러(CEEHDLR을 통해 등록된)보다 먼저 제어를 얻습니다. 그런 다음, 제어는 COBOL에서 USE 선언문 부분으로 진행됩니다.

ILE 조건 핸들러는 실행시 예외 핸들러를 **등록**할 수 있게 합니다. ILE 조건 핸들러는 특정 호출 스택 항목을 위해 등록됩니다. ILE 조건 핸들러를 등록하려면 사용자 작성 조건 핸들러 등록(CEEHDLR) 바인드 가능 API를 사용하십시오. 이 API를 사용하면 예외가 발생할 때 제어를 부여해야 하는 프로시듀어를 실행시에 식별할 수 있습니다. CEEHDLR API는 언어내에 프로시듀어 포인터를 선언하고 설정하는 능력이 필요합니다. CEEHDLR은 내장 함수로 구현됩니다. 그러므로 주소는 지정될 수 없으며, 프로시듀어 포인터를 통해 호출할 수 없습니다. ILE 조건 핸들러는 사용자 작성 조건 핸들러 등록 해제(CEEHDLU) 바인드 가능 API를 호출하여 **등록 해제**시킬 수 있습니다.

OPM과 ILE는 HLL 특정 핸들러를 지원합니다. **HLL 특정 핸들러**는 오류 처리를 위해 정의된 언어 피처입니다. 예를 들어, ILE C 신호 기능을 예외 메세지 처리에 사용할 수 있습니다. RPG에서 HLL 특정 오류 처리에는 다닐 명령문(E 확장기), 명령문 그룹(MONITOR) 또는 전체 프로시듀어(*PSSR 및 INFSR 서브루틴)에 대한 예외를 처리하는 능력이 포함됩니다. COBOL에서 HLL 특수 오류 처리에는 I/O 오류 처리를 위한 USE 선언문과 ON SIZE ERROR 및 AT INVALID KEY와 같은 명령문 범위 조건의 지시문이 포함됩니다.

예외 핸들러 우선순위는 HLL 특정 오류 처리와 추가 ILE 예외 핸들러 유형을 둘다 사용하는 경우 중요합니다.

52 페이지의 그림 26에서는 프로시듀어 P2를 위한 호출 스택 항목을 보여줍니다. 이 예에서 세 가지 유형 핸들러는 모두 단일 호출 스택 항목을 위해 정의되었습니다. 이것이 전형적인 예는 아니더라도 세 가지 유형을 모두 정의하는 것은 가능합니다. 세 가지 유형이 모두 정의되기 때문에 예외 핸들러 우선순위가 정의됩니다. 이 그림에서는 우선순위를 보여줍니다. 예외 메세지가 송신될 때 예외 핸들러는 다음 순서로 호출됩니다.

1. 직접 모니터 핸들러

   먼저 호출이 선택된 후 해당 호출에서 핸들러의 상대적인 순서가 선택됩니다. 호출 내에서 모든 직접 모니터 핸들러와 COBOL 명령문으로 범위가 정해진 조건 명령

문은 ILE 조건 핸들러보다 먼저 제어를 얻습니다. 이와 유사한 방식으로 ILE 조건 핸들러는 다른 HLL 특정 핸들러보다 먼저 제어를 얻습니다.

직접 모니터 핸들러가 예외를 초래한 명령문 주위에서 선언된 경우 이러한 핸들러는 HLL 특정 핸들러보다 먼저 호출됩니다. 예를 들어, 52 페이지의 그림 26에서 프로시듀어 P2에 HLL 특정 핸들러가 있고 프로시듀어 P1에 직접 모니터 핸들러가 있는 경우 P2 핸들러는 P1의 직접 모니터 핸들러 이전에 고려됩니다.

직접 모니터는 어휘적으로(lexically) 내포될 수 있습니다. 가장 깊이 내포된 직접 모니터에서 지정된 핸들러는 같은 우선순위 번호를 지정하는 내포된 복수 모니터내에서 제일 먼저 선택됩니다.

2. ILE 조건 핸들러

   ILE 조건 핸들러가 호출 스택 항목을 위해 등록된 경우 이 핸들러는 두 번째로 호출됩니다. 여러 개의 ILE 조건 핸들러가 등록될 수 있습니다. 이 예에서 프로시듀어 P5와 프로시듀어 P6은 ILE 조건 핸들러입니다. 여러 개의 ILE 조건 핸들러가 같은 호출 스택 항목을 위해 등록될 때 시스템은 이러한 핸들러를 후입선출(LIFO) 순서로 호출합니다. COBOL 명령문으로 범위가 정해진 조건 명령문을 HLL 특정 핸들러로 분류하는 경우 명령문은 ILE 조건 핸들러보다 우선순위가 높습니다. 보통은 HLL 고유 핸들러에 가장 낮은 우선순위가 지정되며 그 다음이 직접 모니터 핸들러와 조건 핸들러입니다. 한 가지 예외는 COBOL문 범위 조건 명령문 즉 HLL 고유 핸들러로서 직접 모니터 핸들러와 같은 우선순위를 가집니다.

3. HLL 특정 핸들러

   HLL 특정 핸들러는 마지막으로 호출됩니다.

시스템은 예외 메세지가 수정되어 처리되었음을 보여줄 때 예외 처리를 종료합니다. 직접 모니터 핸들러나 ILE 조건 핸들러를 사용 중인 경우 예외 메세지를 수정하는 것은 사용자의 책임입니다. 사용자는 몇 가지의 제어 조치를 사용할 수 있습니다. 예를 들어, 제어 조치로서 핸들러를 지정할 수 있습니다. 예외 메세지가 미처리 상태로 남아 있는 한 시스템은 이전에 정의된 우선순위를 사용하여 계속해서 예외 핸들러를 탐색합니다. 예외가 현재 호출 스택 항목내에서 처리되지 않으면 이전 호출 스택 항목으로의 여과가 발생합니다. HLL 특정 오류 처리를 사용하지 않으면 사용자의 ILE HLL은 예외 처리를 허용하여 이전 호출 스택 항목에서 계속하도록 선택될 수 있습니다.

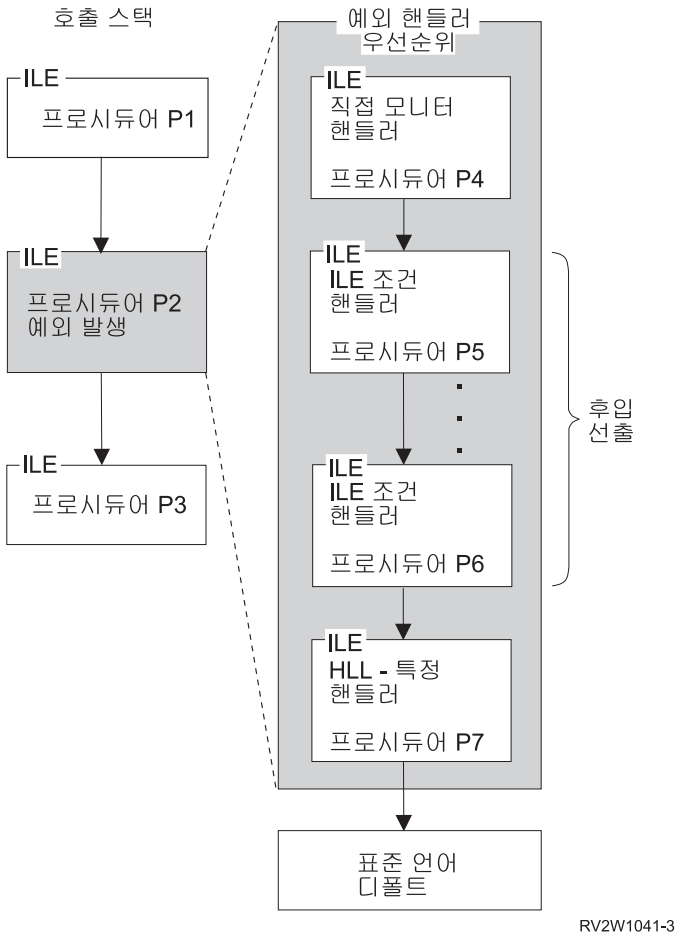

그림 26. 예외 핸들러 우선순위

ILE 조건

보다 큰 시스템간에 일관성이 유지되도록 ILE는 사용자가 오류 조건에 대해 작업할 수 있는 피처를 정의합니다. ILE **조건**은 HLL내에서 시스템과는 무관한 오류 조건의 표현입니다. OS/400의 경우 각 ILE 조건은 해당 예외 메세지를 갖습니다. ILE 조건은 조건 토큰에 의해 표현됩니다. **조건 토큰**은 여러 참여 시스템간에서 일관된 12바이트의 자료 구조로 이루어집니다. 이 자료 구조에는 사용자가 조건과 기본이 되는 예외 메세지를 연관시킬 수 있도록 하는 정보가 들어 있습니다.

시스템에 걸쳐 일관된 프로그램을 작성하려면 ILE 조건 핸들러와 ILE 조건 토큰을 사용해야 합니다. ILE 조건에 관한 자세한 정보는 135 페이지의 제 9 장『예외 및 조건 관리』부분을 참조하십시오.

자료 관리 범위지정 규칙

자료 관리 범위지정 규칙은 자료 관리 자원의 사용을 제어합니다. 이러한 자원은 프로그램이 자료 관리에 대해 작업할 수 있게 하는 임시 오브젝트입니다. 예를 들어, 프로그램이 파일을 열 때 열린 자료 경로(ODP)라고 하는 오브젝트가 작성되어 프로그램을 파일에 연결합니다. 프로그램이 대체 속성을 작성하여 파일이 처리되어야 하는 방법을 변경할 때는 시스템이 대체 오브젝트를 작성합니다.

자료 관리 범위지정 규칙은 자원이 호출 스택에서 실행 중인 여러 프로그램이나 프로시듀어에 의해 공유될 수 있는 시점을 판별합니다. 예를 들어, SHARE(*YES) 매개변수 값으로 작성된 열린 파일이나 확약 정의 오브젝트는 동시에 많은 프로그램에 의해 사용될 수 있습니다. 자료 관리 자원을 공유하는 능력은 자료 관리 자원에 대한 범위지정 레벨에 달려 있습니다.

또한 자료 관리 범위지정 규칙은 자원의 존재를 판별합니다. 시스템은 범위지정 규칙에 따라 작업내에서 사용되지 않는 자원을 자동으로 삭제합니다. 이 자동 지우기(cleanup) 작업의 결과로서 작업은 기억장치를 덜 사용하며 작업 성능은 향상됩니다.

ILE는 다음과 같은 범위지정 레벨내에 OPM과 ILE 프로그램 둘다에 대한 자료 관리 범위지정 규칙을 형성합니다.

호출
활성 그룹
작업

사용 중인 자료 관리 자원에 따라 하나 이상의 범위지정 레벨이 명시적으로 지정될 수 있습니다. 범위지정 레벨을 선택하지 않은 경우 시스템은 레벨 중 하나를 디폴트로 선택합니다.

각 자료 관리 자원이 범위지정 레벨을 지원하는 방법에 관해서는 151 페이지의 제 11 장『자료 관리 범위지정』부분을 참조하십시오.

호출 레벨 범위지정

호출 레벨 범위지정은 자료 관리 자원이 자원을 작성한 호출 스택 항목에 연결될 때 발생합니다. 54 페이지의 그림 27에서는 예를 보여줍니다. 호출 레벨 범위지정은 보통 디폴트 활성 그룹에서 실행되는 프로그램을 위한 디폴트 범위지정 레벨입니다. 이 그림에서 OPM 프로그램 A, OPM 프로그램 B 또는 ILE 프로시듀어 P2는 각각의 파일인 F1, F2 또는 F3을 닫지 않고 리턴하도록 선택할 수 있습니다. 자료 관리는 각 파일의 ODP를 파일을 연 호출 레벨 번호와 연관시킵니다. RCLRSC 명령은 그 명령에 전달된 특정 호출 레벨 번호에 기초하여 파일을 닫는 데 사용될 수 있습니다.

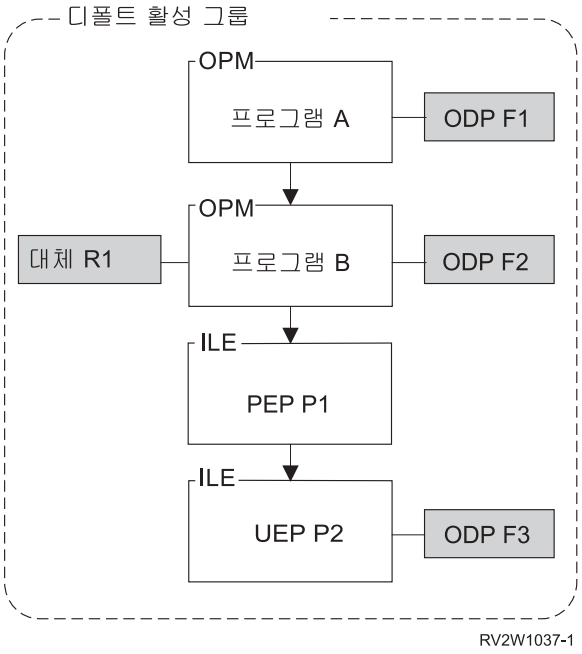

그림 27. 호출 레벨 범위지정. ODP와 대체 속성은 호출 레벨로 범위가 지정될 수 있습니다.

특정 호출 레벨로 범위가 지정되는 대체 속성은 대응 호출 스택 항목이 리턴할 때 삭제됩니다. 대체는 대체를 작성한 호출 레벨 아래에 있는 모든 호출 스택 항목에 의해 공유될 수 있습니다.

활성 그룹 레벨 범위지정

활성 그룹 레벨 범위지정은 자료 관리 자원이 자원을 작성한 ILE 프로그램의 활성 그룹이나 ILE 서비스 프로그램에 연결될 때 발생합니다. 활성 그룹이 삭제될 때 자료 관리는 활성 그룹에서 실행 중인 프로그램에 의해 열린 상태로 남아 있는 활성 그룹과 연관된 모든 자원을 닫습니다. 55 페이지의 그림 28에서는 활성 그룹 레벨 범위지정의 예를 보여줍니다. 활성 그룹 레벨 범위지정은 디폴트 활성 그룹에서 실행되지 않는 ILE 프로시듀어에 의해 사용되는 거의 모든 유형의 자료 관리 자원에 대한 디폴트 범위지정 레벨입니다. 예를 들어, 이 그림에서는 F1, F2 및 F3 파일의 ODP와 활성 그룹에 범위지정된 대체 R1을 보여줍니다.

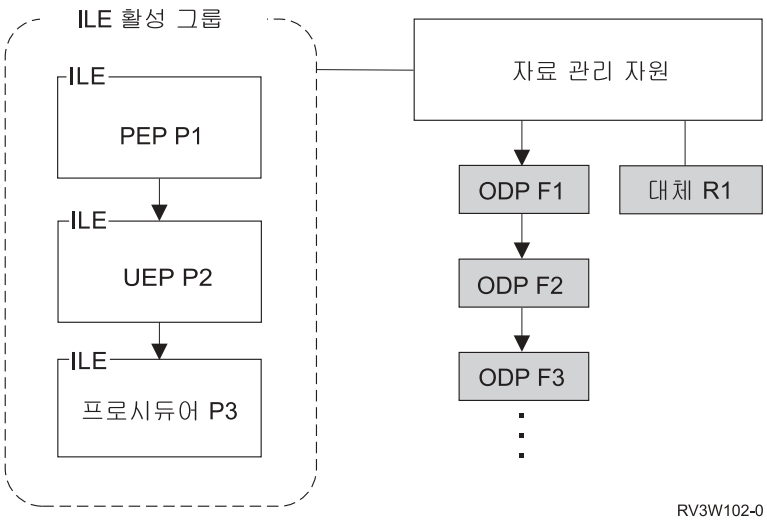

그림 28. 활성 그룹 레벨 범위지정. ODP와 대체 속성은 활성 그룹으로 범위지정될 수 있습니다.

활성 그룹으로 범위지정된 자료 관리 자원을 공유하는 능력은 해당 활성 그룹에서 실행 중인 프로그램으로 제한됩니다. 이것은 어플리케이션 분리와 보호를 제공합니다. 예를 들어, 그림에서 파일 F1이 SHARE(*YES) 매개변수 값으로 열린 것으로 가정하십시오. 파일 F1은 같은 활성 그룹에서 실행 중인 모든 ILE 프로시듀어에 의해 사용될 수 있습니다. 다른 활성 그룹에서의 파일 F1의 또다른 열기 조작은 해당 파일에 대해 제2의 ODP가 작성되는 결과를 초래합니다.

작업 레벨 범위지정

작업 레벨 범위지정은 자료 관리 자원이 작업에 연결될 때 발생합니다. 작업 레벨 범위지정은 OPM 및 ILE 프로그램 둘다에 대해 사용가능합니다. 작업 레벨 범위지정은 다른 활성 그룹에서 실행 중인 프로그램간에 자료 관리 자원의 공유를 허용합니다. 앞에서 설명한 것처럼 활성 그룹에 대한 범위지정 자원은 해당 자원의 공유를 해당 활성 그룹에서 실행 중인 프로그램으로 한정합니다. 작업 레벨 범위지정은 작업에서 실행 중인 모든 ILE 및 OPM 프로그램간에 자료 관리 자원의 공유를 허용합니다.

56 페이지의 그림 29에서는 작업 레벨 범위지정의 예를 보여줍니다. 프로그램 A는 작업 레벨 범위지정으로 파일 F1을 열 수 있습니다. 이 파일의 ODP는 작업에 연결됩니다. 파일은 작업이 종료되지 않는 한, 시스템에 의해 닫히지 않습니다. ODP가 SHARE(YES) 매개변수 값으로 작성된 경우에는 모든 OPM 프로그램이나 ILE 프로시듀어가 파일을 공유할 가능성이 있습니다.

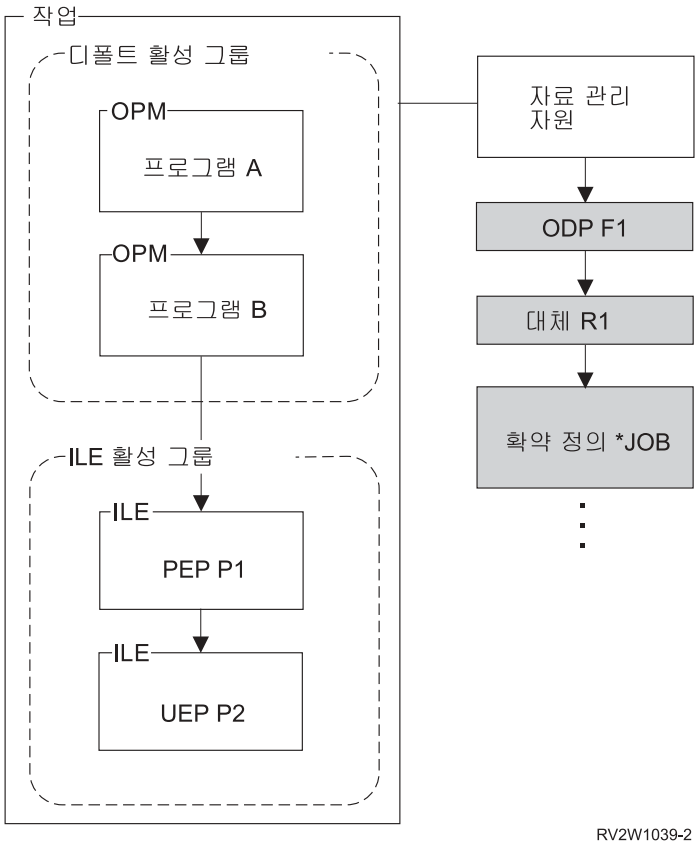

그림 29. 작업 레벨 범위지정. ODP, 대체 속성 및 확약 정의는 작업 레벨로 범위지정될 수 있습니다.

작업 레벨로 범위지정된 대체 속성은 작업에 있는 모든 열린 파일 조작에 영향을 줍니다. 이 예에서 대체 속성 R1은 프로시듀어 P2에 의해 작성되었을 수 있습니다. 작업 레벨 대체 속성은 명시적으로 삭제되거나 작업이 종료될 때까지 활동 상태로 남아 있습니다. 작업 레벨 대체 속성은 병합이 발생할 때 가장 높은 우선순위 대체 속성을 갖습니다. 이것은 호출 레벨 대체가 여러 대체 속성이 호출 스택에 있을 때 함께 병합되기 때문입니다.

자료 관리 범위지정 레벨은 대체 명령에 범위지정 매개변수를 사용하거나 확약 제어 명령 및 다양한 API를 통해 명시적으로 지정될 수 있습니다. 범위지정 규칙을 사용하는 자료 관리 자원의 전체 리스트는 151 페이지의 제 11 장『자료 관리 범위지정』부분을 참조하십시오.

제 4 장 테라스페이스 및 단일 레벨 저장

ILE 프로그램을 작성할 때 iSeries 서버에서 2가지 유형의 기억장치(테라스페이스 및 단일 레벨 저장)에서 선택할 수 있습니다. 이 장에서는 더 새로운 테라스페이스 옵션에 초점을 둡니다. ILE 프로그램은 디폴트로 단일 레벨 저장을 사용합니다.

테라스페이스 특성

테라스페이스는 작업에 로컬인 큰 임시 공간입니다. 테라스페이스는 연속 주소 공간을 제공하지만 사이에 할당되지 않은 영역을 사용하여 개별적으로 할당된 여러 영역으로 구성될 수 있습니다. 테라스페이스는 작업 시작 및 작업 끝 사이의 시간 이상으로 존재하지 않습니다.

테라스페이스는 영역 오브젝트가 아닙니다. 이것은 시스템 오브젝트가 아니며 시스템 포인터를 사용하여 이를 참조할 수 없음을 의미합니다. 그러나 테라스페이스는 공백 포인터를 사용하여 주소지정할 수 있습니다.

다음 표에서는 테라스페이스를 단일 레벨 저장과 비교하는 방식을 보여줍니다.

표 2. 테라스페이스 및 단일 레벨 저장의 비교

속성	테라스페이스	단일 레벨 저장
위치	프로세스 로컬: 일반적으로 소유 작업만 액세스할 수 있습니다.	글로벌: 포인터를 갖는 작업을 액세스할 수 있습니다.
크기	총 1TB	많은 16MB 단위
메모리 맵핑 지원?	예	아니오
8바이트 포인터로 주소지정?	예	아니오
작업간의 공유 지원?	공유 메모리 API를 사용하여 수행되어야 합니다(예를 들어, shmat 또는 mmap).	다른 작업으로 포인터를 전달하거나 공유 메모리 API를 사용하여 수행될 수 있습니다.

테라스페이스의 사용자 프로그램 작동

ILE 프로그램은 디폴트로 단일 레벨 저장을 사용합니다. 테라스페이스 주소를 처리하려면 프로그램이 *테라스페이스를 사용*할 수 있어야 합니다. 테라스페이스 사용 프로그램은 다양한 문맥에서 테라스페이스 주소를 처리할 수 있습니다. 예를 들어, 다음과 같습니다.

- 테라스페이스 힙(heap) 기억장치를 할당하기 위해 요청에서 리턴될 때
- 테라스페이스 공유 메모리를 할당하기 위해 요청에서 리턴될 때
- 다른 프로그램에서 전달될 때

다음 컴파일러는 테라스페이스 사용 코드를 생성합니다.

- 모듈 및 프로그램을 작성할 때 ILE C(TERASPACE(*YES) 선택)
- 모듈 및 프로그램을 작성할 때 ILE C++(TERASPACE(*YES) 선택)
- ILE RPG(V4R4부터 테라스페이스 사용이 디폴트입니다)
- ILE COBOL(V4R4부터 테라스페이스 사용이 디폴트입니다)
- ILE CL(V5R1부터 테라스페이스 사용이 디폴트입니다)

ILE C 및 C++ 컴파일러는 TERASPACE(*YES *TSIFC) 작성 명령 옵션을 제공하여 소스 코드 변경사항 없이 기억장치 인터페이스의 테라스페이스 버전의 사용을 허용합니다. 예를 들어, `malloc()`는 `_C_TS_malloc()`로 맵핑됩니다.

이 컴파일러 옵션에 관한 자세한 설명은 WebSphere Development Studio ILE C/C++ Programmer's Guide 를 참조하십시오.

CHGPGM 명령을 사용하여 OPM 프로그램이 테라스페이스를 사용하게 만들 수 있습니다.

프로그램 기억장치 모델 선택

사용자 모듈 및 프로그램을 작성하여 테라스페이스 사용 단계를 넘어 *테라스페이스 기억장치 모델*을 사용할 수 있습니다. 테라스페이스 기억장치 모델 프로그램은 자동, 정적 및 상수 기억장치용 테라스페이스를 사용합니다. 테라스페이스 기억장치 모델을 선택할 때 일부 이러한 유형의 기억장치에 대해 더 큰 영역을 사용할 수 있습니다. 또한 테라스페이스 기억장치 모델 프로그램은 8바이트 포인터를 사용하여 이 기억장치 영역을 주소지정할 수 있습니다. 테라스페이스 기억장치 모델에 대한 자세한 정보는 65 페이지의 『테라스페이스 기억장치 모델 사용』을 참조하십시오.

사용자 모듈 및 프로그램에 대해 두가지 기억장치 모델 중 하나를 지정하는 옵션을 갖습니다. 단일 레벨 저장(*SNGLVL) 및 테라스페이스(*TERASPACE). 또한 서비스 프로그램이 실행하는 활성 그룹의 기억장치 모델을 상속하도록 허용하도록 선택할 수 있습니다. 이 주제는 테라스페이스 기억장치 모델을 설명합니다.

테라스페이스 기억장치 모델 지정

테라스페이스 기억장치 모델을 선택하려면 사용자 코드를 컴파일할 때 다음 옵션을 지정하십시오.

1. 사용자 모델이 테라스페이스를 사용할 수 있는 것인지 확인하십시오. 사용자 모델을 작성할 때 TERASPACE 매개변수에서 *YES를 지정하십시오.

DSPMOD, DSPPGM, DSPSRVPGM 명령을 사용하여 모듈, 프로그램, 서비스 프로그램 각각의 테라스페이스 속성을 표시할 수 있습니다. 각 모듈의 세부사항을 표시하기 위해 옵션 5를 사용할 경우 DETAIL(*MODULE)로부터 테라스페이스 속성을 볼 수 있습니다.

2. 사용자 ILE 프로그래밍 언어에 대해 모듈 작성 명령의 기억장치 모델(STGMDL) 매개변수에 대해 *TERASPACE 또는 *INHERIT를 지정하십시오.
3. CRTPGM 또는 CRTSRVPGM 명령의 STGMDL 매개변수에서 *TERASPACE를 지정하십시오. 이 선택사항은 프로그램과 바인드하는 모듈의 기억장치 모델과 호환되어야 합니다. 자세한 내용은 61 페이지의 『바인딩 모듈의 규칙』 부분을 참조하십시오.

   또한 하나의 모듈만 들어 있는 바인드 프로그램을 한 단계에서 작성하는 CRTBNDC 및 CRTBNDCPP 명령의 STGMDL 매개변수에서 *TERASPACE를 지정할 수 있습니다.

   CRTSRVPGM 명령에서 STGMDL 매개변수에 *INHERIT를 지정할 수 있습니다. 이것은 서비스 프로그램이 활성화되는 활성 그룹에서 사용하는 기억장치의 유형에 따라 단일 레벨 저장 또는 테라스페이스 중 하나를 사용할 수 있는 방식으로 서비스 프로그램이 작성되게 합니다.

   *INHERIT 속성의 사용은 가장 큰 유연성을 제공하지만 ACTGRP 매개변수에 *CALLER를 지정해야 합니다. 이러한 경우에 서비스 프로그램은 단일 레벨 저장이나 테라스페이스 중 하나로 활성화될 수 있음을 기억하고, 양쪽 상황을 효과적으로 처리할 수 있음을 주의해야 합니다. 예를 들어, 모든 정적 변수의 총 크기는 단일 레벨 저장용으로 부과된 더 작은 한계보다 더 크지 않아야 합니다.

표 3. 특정 유형의 프로그램에 대해 허용된 기억장치 모델.

프로그램 기억장치 모델	프로그램 유형		
	OPM *PGM	**ILE *PGM**	**ILE *SRVPGM**
*TERASPACE	아니오	예	예
*INHERIT	아니오	아니오	예, ACTGRP (*CALLER) 경우에만
*SNGLVL	예	예	예

호환가능한 활성 그룹 선택

활성 그룹은 작성되는 활성 그룹을 야기한 루트 프로그램의 기억장치 모델을 반영합니다. 기억장치 모델은 프로그램에 제공된 자동, 정적 및 상수 기억장치의 유형을 판별합니다.

단일 레벨 저장 기억장치 모델 프로그램은 단일 레벨 저장 자동, 정적 및 상수 기억장치를 수신합니다. 디폴트로 이 프로그램은 힙(heap) 기억장치에 대해 단일 레벨 저장을 사용합니다.

테라스페이스 기억장치 모델 프로그램은 자동, 정적 및 상수 기억장치를 수신합니다. 디폴트로 이 프로그램은 힙(heap) 기억장치에 대해 테라스페이스를 사용합니다.

테라스페이스 기억장치 모델을 사용하는 프로그램은 루트 프로그램이 단일 레벨 저장 기억장치 모델을 사용하는 활성 그룹으로 활성화될 수 없습니다. 단일 레벨 저장 기억장치 모델을 사용하는 프로그램은 루트 프로그램이 테라스페이스 기억장치 모델을 사용하는 활성 그룹으로 활성화될 수 없습니다.

다음 표는 기억장치 모델 및 활성 그룹 유형간의 관계를 요약합니다.

표 4. 기억장치 모델과 활성 그룹의 관계

프로그램 기억장치 모델	활성 그룹 속성			
	***CALLER**	***DFTACTGRP**	***NEW**	**Named**
*TERASPACE	예. 클라이언트 프로그램도 테라스페이스 기억장치 모델로 작성했는지 확인하십시오.	허용되지 않음. 디폴트 활성 그룹은 *SNGLVL 뿐입니다.	예	예
*INHERIT	예	허용되지 않음.	허용되지 않음.	허용되지 않음.
*SNGLVL	예	예	예	예

사용자 프로그램 또는 서비스 프로그램이 실행하는 활성 그룹을 선택할 때 다음 지침을 고려하십시오.

- 서비스 프로그램이 STGMDL(*INHERIT)을 지정하는 경우 ACTGRP(*CALLER)를 지정해야 합니다.
- 프로그램이 STGMDL(*TERASPACE)을 지정하는 경우
  - ACTGRP(*NEW) 또는 명명된 활성 그룹을 지정하십시오.
  - 사용자 프로그램을 호출하는 모든 프로그램이 테라스페이스 기억장치 모델을 사용함을 보장할 수 있는 경우에만 ACTGRP(*CALLER)를 지정하십시오.

기억장치 모델이 상호작용하는 방식

기억장치 모델을 사용하는 모듈 및 프로그램간에 일관성이 요구됩니다. 여기에는 프로그램이 적절하게 상호작용함을 보증하는 규칙이 있습니다.

- 61 페이지의『바인딩 모듈의 규칙』
- 61 페이지의『서비스 프로그램에 대한 바인딩 규칙』
- 61 페이지의『활성화 프로그램 및 서비스 프로그램 활성화를 위한 규칙』
- 62 페이지의『프로그램 및 프로시듀어 호출의 규칙』

바인딩 모듈의 규칙

다음 표는 바인딩 모듈에 대한 규칙을 보여줍니다.

바인딩 규칙: 지정된 기억장치 모듈을 사용하여 프로그램으로 모듈 M 바인딩.		작성 중인 프로그램의 기억장치 모델.		
		테라스페이스	상속(서비스 프로그램만)	단일 레벨 저장
M	테라스페이스	테라스페이스	오류	오류
	상속	테라스페이스	상속	단일 레벨 저장
	단일 레벨 저장	오류	오류	단일 레벨 저장

서비스 프로그램에 대한 바인딩 규칙

다음 표는 목표 서비스 프로그램에 프로그램을 바인딩하는 규칙을 보여줍니다.

서비스 프로그램 바인딩 규칙: 호출 프로그램 또는 서비스 프로그램을 목표 서비스 프로그램으로 바인드할 수 있습니까?		목표 서비스 프로그램 기억장치 모델		
		테라스페이스	상속	단일 레벨 저장
호출 프로그램 또는 서비스 프로그램의 기억장치 모델	테라스페이스	예	예	예[2]
	상속[1]	예[3]	예	예[3]
	단일 레벨 저장	예[2]	예	예

주:

1. 서비스 프로그램만 기억장치 모델을 상속하도록 지정될 수 있습니다.
2. 목표 서비스 프로그램은 고유한 활성 그룹에서 실행되어야 합니다. 예를 들어, 목표 서비스 프로그램은 ACTGRP(*CALLER) 속성을 가질 수 없습니다. 단일 활성 그룹내에서 기억장치 모델을 혼합하는 것은 가능하지 않습니다.
3. 상속 기억장치 모델을 사용하는 서비스 프로그램을 단일 레벨 저장이나 테라스페이스 서비스 프로그램에 바인딩하는 것은 허용되지만 조작의 궁극적인 성공은 활성 시간까지 알려질 수 없습니다. 목표 서비스 프로그램이 ACTGRP(*CALLER) 속성을 갖는 경우 호출 서비스 프로그램(기억장치 모델을 상속하도록 지정됨)은 목표 서비스 프로그램과 호환가능한 활성 그룹으로 활성화되어야 합니다. 목표 서비스 프로그램은 ACTGRP(*CALLER)를 지정할 수 없거나 호출 프로그램이나 서비스 프로그램과 동일하게 명명된 활성 그룹을 지정할 수 없습니다.

활성화 프로그램 및 서비스 프로그램 활성화를 위한 규칙

상속 기억장치 모델을 지정하는 테라스페이스 사용 서비스 프로그램은 단일 레벨 저장이나 테라스페이스 기억장치 모델을 사용하는 프로그램을 실행하는 활성 그룹으로 활성화할 수 있습니다. 그렇지 않으면 서비스 프로그램의 기억장치 모델이 활성 그룹에서 실행하는 다른 프로그램의 기억장치 모델과 일치해야 합니다.

프로그램 및 프로시듀어 호출의 규칙

서로 다른 기억장치 모델을 사용하는 프로그램과 서비스 프로그램들간에 상호조작이 가능합니다. 이 장에서 설명하는 규칙 및 제한사항과 일치한다면 자료를 함께 바인드시키고 공유할 수 있습니다.

테라스페이스를 사용할 수 없는 코드는 테라스페이스 주소를 처리할 수 없습니다. 그러한 시도는 보통 MCH0607(지원되지 않는 공간 사용) 예외를 야기합니다.

기억장치 모델을 상속하도록 서비스 프로그램 변환

기억장치 모델을 상속하도록 서비스 프로그램을 변환하여(STGMDL 매개변수에서 *INHERIT 지정), 테러 영역이나 단일 레벨 기억장치 환경 중 하나에서 서비스 프로그램을 사용할 수 있게 합니다. 테라스페이스 기억장치 모델에 대해 기존 서비스 프로그램을 작동할 수 있게 하려면 다음 단계를 따르십시오.

1. 상속 기억장치 모델로 모든 사용자 모듈을 작성하십시오. 모듈 중 하나가 단일 레벨 저장 또는 테라스페이스 기억장치 모델로 작성된 경우 상속 기억장치 모델로 서비스 프로그램을 작성할 수 없습니다.
2. 사용자 코드가 테라스페이스 및 단일 레벨 저장 기억장치간의 포인터를 예상하고 효과적으로 관리하는지 확인하십시오. 자세한 정보는 65 페이지의 『테라스페이스 사용: 최적 실행』을 참조하십시오.
3. 상속 기억장치 모델로 서비스 프로그램을 작성하십시오. 마찬가지로 ACTGRP 매개변수에서 *CALLER를 지정하십시오.

사용자 프로그램 변경 및 갱신: 테라스페이스 고려사항

특정 환경에서 테라스페이스를 사용할 수 있도록 사용자 프로그램을 변경하거나 갱신할 수 있습니다. 사용자 프로그램을 갱신하기 위해 사용되는 모듈의 기억장치 모델에 대한 제한사항이 있습니다.

사용자 프로그램 변경:

비테라스페이스 사용 프로그램을 테라스페이스 사용 프로그램으로 변환하기 위해 CHGPGM 및 CHGSRVPGM 명령을 사용할 수 있습니다. 모든 바인드 모듈과 함께 V4R4M0 이상의 목표 릴리스를 가진 ILE 프로그램에 대해 이를 수행할 수 있습니다. 또한 V4R4M0 이상의 목표 릴리스를 가진 OPM 프로그램에 대해 이를 수행할 수 있습니다.

사용자 프로그램 갱신:

동일한 기억장치 모델을 사용하는 한 프로그램내에서 모듈을 추가하고 대체할 수 있습니다. 그러나 바인드 모듈이나 프로그램의 기억장치 모델을 변경하기 위해 갱신 명령을 사용할 수 없습니다.

C 및 C++ 코드에서 8바이트 포인터 이용

8바이트 포인터는 테라스페이스만 포인트할 수 있습니다. 8바이트 프로시듀어 포인터는 테라스페이스를 통해 활동 프로시듀어를 나타냅니다. 유일한 8바이트 포인터 유형은 공간 및 프로시듀어 포인터입니다.

대조적으로, 여러 가지 유형의 16바이트 포인터가 있습니다. 다음 표는 8바이트 및 16바이트 포인터를 비교한 것입니다.

표 5. 포인터 비교

등록 정보	**8**바이트 포인터	**16**바이트 포인터
길이(필요한 메모리)	8바이트	16바이트
태그화	아니오	예
정렬	바이트 정렬이 허용(즉, 팩 구조). 성능을 위해서는 ″자연적″ 정렬(8바이트)이 좋습니다.	항상 16바이트.
아토믹(Atomicity)	8바이트 정렬시 아토믹(atomic) 로드 및 저장 조작. 집합 복사 조작에는 적용 안됨.	아토믹(atomic) 로드 및 저장 조작. 집합의 일부에서 아토믹(atomic) 복사.
주소지정 가능 범위	테라스페이스 기억장치	테라스페이스 기억장치 + 단일 레벨 기억장치
포인터 내용	테라스페이스로의 오프셋을 나타내는 64비트 값. 유효 주소 포함 안함.	16바이트 포인터 유형 비트 및 64비트 유효 주소.
참조 위치	로컬 기억장치 참조 처리(8바이트 포인터는 기억장치 참조가 발생하는 작업의 테라스페이스를 참조만 할 수 있습니다).	프로세스 로컬 또는 단일 레벨 저장 기억장치 참조 (16바이트 포인터는 로컬로 다른 작업이 소유하는 기억장치를 참조할 수 있습니다).
허용 조작	공백 포인터 및 프로시듀어 포인터에 허용되는 포인터 고유의 조작, 비포인터 보기를 사용하여 포인터를 무효화시키지 않고 2진 자료에 적절한 모든 산술 및 논리 조작을 사용할 수 있습니다.	포인터 고유의 조작만.
가장 빠른 기억장치 참조	아니오	예
가장 빠른 로드, 저장, 공백 포인터 산술	예, EAO 오버헤드 회피 포함	아니오
포인터에 캐스트할 때 보유했던 2진 값의 크기	8바이트	4바이트
예외 핸들러 또는 취소 핸들러인 프로시듀어가 매개변수로 허용	아니오	예

C 및 C++ 컴파일러에서 포인터 지원

IBM C 또는 C++ 컴파일러로 사용자 코드를 컴파일할 때 8바이트 포인터를 완전히 이용하려면 STGMDL(*TERASPACE) 및 DTAMDL(*LLP64)을 지정하십시오.

또한 C 및 C++ 컴파일러는 다음과 같은 포인터 지원을 제공합니다.

- 8 또는 16바이트 포인터를 명시적으로 선언하는 구문:
  - 8바이트를 `char * __ptr64`로 선언
  - 16바이트 포인터를 `char * __ptr128`로 선언
- C 및 C++ 프로그래밍 환경에 고유한 *자료 모델*을 지정하는 컴파일러 옵션 및 프라그마(pragma). 자료 모델은 명시적 규정자 중 하나가 없을 때 포인터의 디폴트 *크기*에 영향을 미칩니다. 자료 모델에는 2가지 선택사항이 있습니다.
  - `P128`, 4-4-16이라고도 합니다.[1]
  - `LLP64`, 4-4-8이라고도 합니다.[2]

포인터 변환

IBM C 및 C++ 컴파일러는 필요하면 함수 및 변수 선언에 근거하여 `__ptr128`을 `__ptr64` 등으로 변환합니다. 다음에 특히 유의하십시오.

- 단일 레벨 저장 기억장치를 지시하는 `A_ptr128`은 임시 `__ptr64` 값으로 변환됩니다
- 테라스페이스를 사용할 수 없는 코드는 테라스페이스에 액세스할 수 없습니다
- 포인터간 매개변수에 대한 인터페이스는 특수한 처리를 필요로 합니다.

컴파일러는 포인터 길이를 일치시키기 위해 자동으로 포인터 변환을 삽입합니다. 예를 들어, 함수에 대한 포인터 인수가 함수의 원형에서 포인터 매개변수의 길이와 일치하지 않을 때 변환이 삽입됩니다. 아니면, 다른 길이의 포인터가 비교되는 경우 컴파일러는 내재적으로 8바이트 포인터를 비교할 16바이트 포인터로 변환합니다. 또한 컴파일러는 캐스트와 같이 명시적 변환이 지정되도록 허용합니다. 포인터 캐스트를 추가하는 경우 이 포인트를 보존하십시오.

- 16바이트 포인터에 테라스페이스 주소나 널 포인터 값이 들어 있는 경우에만 16바이트 포인터에서 8바이트 포인터로의 변환이 작동합니다. 그렇지 않으면 MCH0609 예외가 신호되거나 임의 테라스페이스 오프셋 값이 리턴됩니다.
- 16바이트 포인터는 하나에서 다른 것으로 변환된 유형을 가질 수 없지만 16바이트 OPEN 포인터는 임의의 포인터 유형을 포함할 수 있습니다. 대조적으로, 8바이트 OPEN 포인터는 존재하지 않지만 8바이트 포인터는 공백 포인터와 프로시듀어 포인터간에 논리적으로 변환될 수 있습니다. 그렇다고 할지라도, 8바이트 포인터 변환은 포인터 유형의 보기이므로, 공백 포인터가 프로시듀어에 대한 포인트로서 설정된 경우를 제외하고는 공백 포인터를 프로시듀어 포인터로서 실제 사용되도록 허용하지 않습니다.

---

1. 4-4-16 = sizeof(int) - sizeof(long) - sizeof(pointer)

2. 4-4-8 = sizeof(int) - sizeof(long) - sizeof(pointer)

포인터와 2진 값간의 명시적 캐스트를 추가할 때 8바이트 및 16바이트 포인터는 다르게 작동함을 기억하십시오. 8바이트 포인터는 완전한 8바이트 2진 값을 보유할 수 있지만 16바이트 포인터는 4바이트 2진 값만 보유할 수 있습니다. 2진 값을 보유하지만 포인터용으로 정의된 유일한 조작은 2진 필드로 다시 변환하는 것입니다. 포인터로서 사용, 다른 포인터 길이로 변환 및 포인터 비교를 포함하여 다른 모든 조작은 정의되지 않습니다. 그러므로, 예를 들어 동일한 정수 값이 8바이트 포인터 및 16바이트 포인터에 지정된 경우 8바이트 포인터는 16바이트 포인터로 변환되며 16바이트 비교가 수행되며, 비교 결과는 정의되지 않으며 동일한 결과를 생성하지 않기 쉽습니다.

혼합 길이 포인터 비교는 16바이트 포인터가 테라스페이스 주소를 보유하고 8바이트 포인터도 테라스페이스 주소를 보유할 때에만(즉 8바이트 포인터에는 2진 값이 들어 있지 않습니다) 정의됩니다. 그러면 8바이트 포인터를 16바이트 포인터로 변환하고 2개의 16바이트 포인터를 비교하는 것이 유효합니다. 다른 모든 경우에 비교 결과는 정의되지 않습니다. 그러므로, 예를 들어, 16바이트 포인터가 8바이트 포인터로 변환된 다음 8바이트 포인터로 비교된 경우 결과는 정의되지 않습니다.

테라스페이스 기억장치 모델 사용

이상적인 테라스페이스 환경에서 모든 사용자 모듈 프로그램 및 서비스 프로그램은 테라스페이스 기억장치 모델을 사용합니다. 그러나 실제 레벨에서 모듈, 프로그램, 그리고 양쪽 기억장치 모델을 사용하는 서비스 프로그램을 조합하는 환경을 관리하는 것이 필요합니다.

이 절에서는 이상적인 테라스페이스 환경을 구현할 수 있는 실제 상황을 설명합니다. 또한 이 절에서는 단일 레벨 저장 및 테라스페이스를 사용하는 프로그램을 혼합함으로써 발생할 수 있는 잠재적 문제점을 최소화하는 방법에 관해 설명합니다.

테라스페이스 사용: 최적 실행

- *테라스페이스 기억장치 모델 모듈만 사용*

  테라스페이스를 사용하고 기억장치 모델을 상속하는 모듈을 작성하십시오. 이 단일 레벨 저장 모듈을 사용자 프로그램으로 바인드할 수 없으므로 단일 레벨 저장 모듈은 테라스페이스 환경에 적합하지 않습니다. 절대적으로 단일 레벨 저장 모듈을 사용해야 하는 경우(예를 들어, 모듈의 소스 코드에 대한 액세스를 갖지 않은 경우), 69 페이지의 『테라스페이스 사용 추가 정보』에서 시나리오 9를 참조하십시오.

- *테라스페이스 또는 상속 기억장치 모델을 사용하는 서비스 프로그램에만 바인드*

  사용자 테라스페이스 기억장치 모델 프로그램은 거의 모든 종류의 서비스 프로그램에 바인드할 수 있습니다. 그러나 일반적으로 상속 또는 테라스페이스 기억장치 모델 서비스 프로그램에만 바인드합니다. 서비스 프로그램을 제어하는 경우 바인드하는 프로그램의 기억장치 모델을 상속할 수 있는 모든 서비스 프로그램을 작성해야 합니다. 일반적으로, IBM 서비스 프로그램은 이러한 방식으로 작성됩니다. 타사 프로그

래머에게 서비스 프로그램을 제공하려는 경우 특히 동일하게 수행할 필요가 있을 수 있습니다. 절대적으로 단일 레벨 저장 서비스 프로그램에 바인드해야 하는 경우 69 페이지의 『테라스페이스 사용 추가 정보』에서 시나리오 10을 참조하십시오.

- *테라스페이스 사용 프로그램만 호출*

  사용자 프로그램은 외부 프로그램 호출을 작성할 수 있습니다. 테라스페이스를 사용할 수 없는 프로그램을 호출하고 사용자 매개변수가 테라스페이스에 있는 경우 호출된 프로그램이 실패할 수 있습니다. 이러한 고려사항은 사용자 나감 프로그램에도 적용합니다.

  그리고, 사용자가 호출한 테라스페이스 사용 프로그램이 테라스페이스 사용 불가 서비스 프로그램이나 프로그램에 테라스페이스 주소를 전달하지 않도록 해야 합니다. 그렇지 않으면 이 주제에 요약되어 있는 최상의 방법을 따라 작업할 수 있습니다. 테라스페이스를 사용하지 않는 프로그램을 호출해야 하거나 호출 중인 프로그램이 테라스페이스를 사용하는지를 판별할 수 없으면 69 페이지의 『테라스페이스 사용 추가 정보』의 시나리오 9에 있는 단계에 따라 호출할 수 있습니다.

- *테라스페이스 사용 프로시듀어에만 포인터 호출 작성*

  사용자 코드는 프로시듀어 포인터를 얻어 이를 사용하여 프로시듀어를 호출할 수 있었습니다. 호출 중인 프로시듀어가 테라스페이스 사용 프로그램이나 서비스 프로그램에 있는지 확인하십시오. 또한 테라스페이스를 사용할 수 없는 프로그램에는 테라스페이스 주소가 전달되지 않게 하십시오. 이와 같이 하거나 테라스페이스 주소를 전달하는지 알 수 없으면 69 페이지의 『테라스페이스 사용 추가 정보』의 시나리오 9에 나오는 지침에 따르십시오.

  프로시듀어가 들어 있는 프로그램이나 서비스 프로그램에는 테라스페이스를 사용하는 모든 모듈이 있어야 하며 그렇지 않으면 프로시듀어 프린터 호출이 실행시 MCH4443으로 실패합니다. 프로그램에 있는 모든 모듈이 테라스페이스를 사용하면 호출된 프로시듀어가 테라스페이스를 사용할 수 있습니다.

이 주제에서 설명한 지침을 따른 경우 사용자 프로그램에서 테라스페이스를 사용할 수 있습니다. 그러나 테라스페이스 사용시 단일 레벨 저장이 디폴트로 사용되므로 코딩시 주의해야 합니다. 다음 주제는 테라스페이스에서 수행할 수 없는 것들과 사용자가 수행할 수 없는 것들을 설명합니다. 어떤 경우에 시스템이 사용자가 특정 조치를 수행하는 것을 막지만 그와 같은 경우가 아니라면 사용자 자신이 테라스페이스 및 단일 레벨 저장 상호작용을 관리해야 합니다.

- 67 페이지의 『테라스페이스 프로그램이 작성될 때의 시스템 제어』
- 67 페이지의 『테라스페이스 프로그램이 활성화될 때의 시스템 제어』
- 67 페이지의 『테라스페이스 프로그램이 실행될 때의 시스템 제어』

주: 상속 기억장치 모델을 사용하는 서비스 프로그램은 테라스페이스를 사용하도록 활성화될 수 있으므로 이러한 실행을 따라야 합니다.

테라스페이스 프로그램이 작성될 때의 시스템 제어

대부분의 경우 시스템은 다음과 같은 조치 중 하나를 사용자가 수행하지 못하게 합니다.

- 단일 레벨 저장 및 테라스페이스 기억장치 모델 모듈을 동일한 프로그램이나 서비스 프로그램으로 결합
- 디폴트 활성 그룹(ACTGRP(*DFTACTGRP))을 지정하는 테라스페이스 기억장치 모델 프로그램 또는 서비스 프로그램 작성
- 단일 레벨 저장 프로그램을 *CALLER의 활성 그룹을 지정하는 테라스페이스 기억장치 모델 서비스로 바인딩

테라스페이스 프로그램이 활성화될 때의 시스템 제어

활성화 시간의 대부분의 경우에 시스템은 단일 레벨 저장 및 테라스페이스 기억장치 모델 프로그램 또는 서비스 프로그램 모두 동일한 활성 그룹으로 활성화하려고 시도하는 방식으로 사용자 프로그램 및 서비스 프로그램을 작성했음을 판별합니다. 그러면 시스템은 활성화 액세스 위반 예외를 송신하며 활성화가 실패합니다.

테라스페이스 프로그램이 실행될 때의 시스템 제어

시스템은 실행 시간까지 다음과 같은 문제점을 감지할 수 없습니다.

- 테라스페이스 기억장치 모델 코드에서 테라스페이스를 사용할 수 없는 단일 레벨 저장 기억장치 모델 코드 호출
- 테라스페이스를 사용할 수 없는 프로그램에서 테라스페이스로 포인터를 사용하려는 시도. 사용자 프로그램이 테라스페이스를 사용할 수 있어야 하며 호출 중인 프로시듀어가 들어 있는 모듈의 경우에만 사용할 수 없어야 합니다.

OS/400 인터페이스 및 테라스페이스

일반적으로, OS/400은 테라스페이스를 사용할 수 있게 작성됩니다.

포인터 매개변수를 갖는 OS/400 인터페이스는 일반적으로 태그화된 16바이트(__ptr128) 포인터를 예상합니다.

- 컴파일러는 필요시 포인터를 변환하므로 8바이트(__ptr64) 포인터를 사용하여 직접 포인터의 단일 레벨(예를 들어, `void f(char*p);`)만으로 인터페이스를 호출할 수 있습니다. 시스템 헤더 파일을 사용하십시오.
- 포인터의 다중 레벨을 갖는 인터페이스(예를 들어, `voidg(char**p);`)는 보통 두 번째 레벨에 대해 16바이트 포인터를 명시적으로 선언하도록 요구합니다. 그러나 8바이트 포인터를 채택하는 버전이 이러한 유형의 대부분의 시스템 인터페이스에 제공되어 8바이트 포인터만 사용하는 코드에서 직접 호출되게 합니다. 사용자가 datamodel(LLP64) 옵션을 선택할 때 표준 헤더 파일을 통해 이 인터페이스가 작동됩니다.

테라스페이스 사용을 위한 바인드 가능 API

IBM에서는 테라스페이스의 할당 및 삭제를 위한 바인드 가능 API를 제공합니다.[3]

`malloc()`, `free()`, `calloc()` 및 `realloc()`는 TERASPACE(*YES *TSIFC) 컴파일러 옵션으로 컴파일된 경우를 제외하고는 호출 프로그램의 기억장치 모델에 따라 단일 레벨 기억장치 또는 테라스페이스 기억장치를 할당하거나 할당해제합니다.

POSIX 공유 메모리 및 메모리 맵핑 파일 인터페이스는 테라스페이스를 사용할 수 있습니다. 프로세스간 통신 API 및 `shmget()` 인터페이스에 대한 자세한 정보는 iSeries Information Center(**Programming** 범주 및 **API** 부속 범주 아래)에서 *UNIX 유형 API* 주제를 참조하십시오.

테라스페이스 사용시 일어날 수 있는 잠재적인 문제점

사용자 프로그램에서 테라스페이스 사용시 일어날 수 있는 잠재적인 문제점을 인식해야 합니다.

- *테라스페이스 주소는 테라스페이스를 사용할 수 없는 프로그램이나 프로시듀어에는 전달되지 않습니다.* 테라스페이스를 사용할 수 없는 코드를 호출하면 프로그램 호출에서의 매개변수는 테라스페이스에 상주할 수 없으며 프로그램이나 프로시듀어 호출 중 하나에 대한 매개변수로서 전달된 포인터는 테라스페이스 주소를 포함할 수 없습니다. 상황에 따라서는 그러한 시도가 MCH0607, MCH3601 또는 MCH3602 예외를 야기합니다.
- *일부 MI 명령어는 테라스페이스 주소를 처리할 수 없습니다.* 이 명령어에서 테라스페이스 주소를 사용하려는 시도는 MCH0607 예외를 야기합니다.
  - CIPHER (일부 옵션만 제한됩니다)
  - MATBPGM
  - MATPG
  - SCANX (일부 옵션만 제한됩니다)
  - SETDP
  - SETDPADR

---

3. STGMDL(*SNGLVL)이 지정되었을 때 테라스페이스 버전과 자동으로 malloc(), free(), calloc(), realloc()을 맵핑하기 위해 ILE C 및 C 컴파일러에서 나온 TERASPACE(*YES *TSIFC)를 사용할 수 있습니다.
   - _C_TS_malloc()는 테라스페이스내에 기억장치를 할당합니다.
   - _C_TS_free()는 테라스페이스에서 하나의 이전 할당을 해제합니다.
   - _C_TS_realloc()는 이전 테라스페이스 할당 크기를 변경합니다.
   - _C_TS_calloc()는 테라스페이스내에 기억장치를 할당하고 0으로 설정합니다.

- *작업간 액세스는 예측될 수 없습니다.* 일부 환경에서 다른 작업으로 전달되는 테라스페이스에 대한 포인터는 테라스페이스가 하나의 작업에 로컬로서 정의될지라도 사용이 가능합니다. 테라스페이스에 대한 포인터를 다른 작업으로 전달되지 않도록 하여 작업간 액세스가 작동하지 않을 때 발생할 수 있는 어플리케이션 실패를 방지하십시오.
- *유효 주소 넘침(EAO)은 성능을 개선시킬 수 있습니다.* 이 상황은 16바이트 포인터에서의 주소 연산이 시작 주소와는 다른 16MB 영역에서 결과 주소를 생성할 때 발생합니다. 하드웨어 인터럽트는 생성되어 시스템 소프트웨어에 의해 처리됩니다. 그러한 여러 인터럽트는 성능에 영향을 미칠 수 있습니다. 16바이트 포인터 연산 사용시 테라스페이스내에서 16MB 경계를 스팬하는 빈번한 주소 연산을 피하십시오.

테라스페이스 사용 추가 정보

테라스페이스 기억장치 모델로 작업할 때 다음과 같은 시나리오를 만날 수 있습니다. 권장되는 솔루션이 제공됩니다.

- *시나리오 1: 단일 할당에서 16MB보다 많은 동적 기억장치를 필요로 합니다*

  malloc를 사용하기 전에 컴파일러 작성 명령에서 _C_TS_malloc를 사용하거나 TERASPACE(*YES *TSIFC)를 지정하십시오. 이것은 테라스페이스 사용 프로그램에 힙(heap) 기억장치를 제공합니다.
- *시나리오 2: 16MB보다 많은 공유 메모리를 필요로 합니다*

  테라스페이스 옵션으로 공유 메모리(shmget)를 사용하십시오.
- *시나리오 3: 효과적으로 큰바이트 문자열 파일을 액세스할 필요가 있습니다*

  메모리 맵핑 파일 사용(mmap).

  테라스페이스 사용 프로그램에서 메모리 맵핑 파일을 액세스할 수 있지만 최적 성능을 위해서는 테라스페이스 기억장치 모델 및 8바이트 프린터 자료 모델을 사용하십시오.
- *시나리오 4: 연속적인 자동 또는 정적 기억장치가 16MB 보다 많이 필요합니다*

  테라스페이스 기억장치 모델을 사용하십시오. 8바이트 또는 16바이트 포인터 중 하나를 사용하여 테라스페이스를 사용할 수 있지만 최적 성능을 위해서는 8바이트 포인터 자료 모델을 선택하십시오.
- *시나리오 5: 사용자 어플리케이션은 공백 포인터를 비중있게 사용합니다*

  테라스페이스 기억장치 모델 및 8바이트 포인터 자료 모델을 사용하여 메모리 자국을 줄이고 포인터 조작 속도를 증가시키십시오.
- *시나리오 6: 다른 시스템에서 코드를 포트하고 16바이트 포인터 사용에 고유한 문제를 피하려고 합니다*

  테라스페이스 기억장치 모델 및 8바이트 포인터 자료 모델을 사용하십시오.

- *시나리오 7: 사용자 테라스페이스 프로그램에서 단일 레벨 저장 기억장치를 사용할 필요가 있습니다*

  때때로 유일한 선택은 테라스페이스 기억장치 모델 프로그램에서 단일 레벨 저장 기억장치를 사용하는 것입니다. 예를 들어, 테라스페이스를 사용할 수 없는 호출 프로그램이나 서비스 프로그램에 매개변수를 저장할 필요가 있을 수 있습니다. 아니면, 프로세스간 통신을 위해 사용자 자료를 저장할 필요가 있을 수 있습니다. 다음 소스 중 하나에서 단일 레벨 저장 기억장치를 구할 수 있습니다.

  – CRTS MI 지침에서 획득된 사용자 공간에 있는 기억장치
  – `malloc`의 단일 레벨 저장 버전
  – 사용자 프로그램으로 전달된 단일 레벨 저장 참조
  – ALCHS MI 명령어에서 얻은 단일 레벨 저장 기억장치 힙(heap) 공간

- *시나리오 8: 사용자 코드에서 8바이트 포인터 이용*

  STGMDL(*TERASPACE)을 사용하여 사용자 모듈 및 프로그램을 작성하십시오. DTAMDL(*LLP64) 또는 명시적 선언(`__ptr64`)을 사용하여 8바이트 포인터를 얻어 테라스페이스를 참조하십시오(테라스페이스를 지시하는 16바이트 포인터와 반대). 그러면 63 페이지의 『C 및 C++ 코드에서 8바이트 포인터 이용』에 나오는 장점을 이용할 수 있습니다.

- *시나리오 9: 단일 레벨 저장 기억장치 모델 모듈 결합*

  테라스페이스 기억장치 모델 모듈로 단일 레벨 저장 모듈을 바인드할 수 없습니다. 이를 수행할 필요가 있는 경우 우선 테라스페이스 기억장치 모델을 사용하는(또는 상속하는) 모듈 버전을 구한 후 65 페이지의 『테라스페이스 사용: 최적 실행』에 나오는 설명처럼 사용하십시오. 아니면 2가지 옵션을 사용할 수 있습니다.

  – 모듈을 별도의 서비스 프로그램으로 패키지하십시오. 서비스 프로그램은 단일 레벨 저장 기억장치 모델을 사용하므로, 아래의 시나리오 10에서 제공하는 접근방식을 사용하여 서비스 프로그램을 호출하십시오.
  – 모듈을 별도의 프로그램으로 패키지하십시오. 이 프로그램은 단일 레벨 저장 기억장치 모델을 사용합니다. 아래의 시나리오 11에서 설명된 접근방식을 사용하여 모듈을 호출하십시오.

- *시나리오 10: 단일 레벨 저장 기억장치 모델 서비스 프로그램에 바인딩*

  2개의 서비스 프로그램이 별도의 활성 그룹으로 활성화하는 경우 단일 레벨 저장을 사용하는 서비스 프로그램에 테라스페이스 프로그램을 바인드할 수 있습니다. 단일 레벨 저장 서비스 프로그램이 ACTGRP(*CALLER) 옵션을 지정하는 경우 이를 수행할 수 없습니다.

  또한 단일 레벨 저장 서비스 프로그램이 테라스페이스 사용할 수 없는 경우에도 테라스페이스 사용 버전을 구하십시오. 테라스페이스 사용 버전을 구할 수 없으면 아래 시나리오 11을 참조하십시오.

- *시나리오 11: 테라스페이스를 사용할 수 없는 프로그램이나 서비스 프로그램 호출*

가능하면 테라스페이스를 사용하지 않는 프로그램은 호출하지 않는 방식으로 프로그램을 코드화하십시오. 그러나 단일 레벨 저장 기억장치에 저장된 매개변수만 전달하려는 경우 이를 수행할 수 있습니다.

이를 수행하려면 테라영역에서 단일 레벨 저장 기억장치로 자료를 복사하고, 프로그램으로 전달한 다음 리턴할 때 임의의 결과나 변경된 기억장치를 다시 테라스페이스로 복사하십시오.

테라스페이스 모델 호출자로부터 테라스페이스를 사용할 수 없는 프로그램의 프로시듀어로 프로시듀어 포인터 호출을 작성할 수 없습니다.

- *시나리오 12: 포인터간(pointer-to-pointer) 매개변수가 있는 함수 호출*

  포인터간 매개변수가 있는 일부 함수의 호출에는 DTMDL(*LLP64 옵션)으로 컴파일한 모듀로부터 특별한 처리가 필요합니다. 8바이트와 16바이트 포인터 사이의 내재적 변환이 포인터 매개변수에 적용됩니다. 포인터 대상 또한 포인터이더라도 포인터 매개변수가 가리키는 자료 오브젝트에 적용되지 않습니다. 예를 들어 일반적으로 사용되는 P128 자료 모델을 나타내는 헤더 파일에 **char**** 인터페이스 사용을 선언할 경우 자료 모델 LLP64로 작성된 모듈에 몇 가지 코드가 필요합니다. 이 경우 반드시 16바이트 포인터 주소를 전달해야 합니다. 다음은 이와 관련된 몇 가지 예입니다.

  – 이 예에서는 CRTCMOD와 같은 작성 명령에 STGMDL (*TERASPACE)DTAMDL(*LLP64) 옵션과 함께 8바이트 포인터를 사용하여 테라스페이스 기억장치 모델을 작성합니다. 이제 포인터에 대한 포인터를 테라스페이스 기억장치 모델 프로그램에서 P128 **char****로, 배열 가운데 한 문자에 전달할 것입니다. 이와 같이 하기 위해서는 반드시 명시적으로 16바이트 포인터를 선언해야 합니다.

```
#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 */
```

  – 일반적으로 포인터간(pointer to pointer) 매개변수와 함께 어플리케이션 프로그래밍 인터페이스(API)에 사용되는 것은 iconv입니다. 이것은 16바이트 포인터만 기대합니다. 다음은 **iconv**에 대한 헤더 파일 부분입니다.

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

```
                size_t   *outbytesleft);
...
#pragma datamodel(pop)
...
```

다음 코드는 DTAMDL(*LLP64) 옵션으로 컴파일된 프로그램에서 **iconv**를 호출합니다.

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

또한 QUSPTRUS(사용자 공간에 대한 포인터 검색) 인터페이스의 헤더 파일이 실제로 포인터에 대한 포인터를 기대하는 곳에서 void* 매개변수를 지정하는지 확인하십시오. 두 번째 피연산자에 대한 16바이트 포인터의 주소가 반드시 전달되어야 합니다.

- *시나리오 13: 명령 처리, 유효성 검사, 프롬트 대체 프로그램을 위한 매개변수 액세스*

  명령 처리, 유효성 검사, 테라스페이스 기억장치 모델을 사용하여 작성된 프롬트 대체 프로그램은 단일 레벨 기억장치에 각 매개변수를 수신합니다. 그러한 프로그램들은 명령행에서 CALL을 사용하여 호출될 경우 테라스페이스 기억장치에 각 매개변수를 수신합니다.

  이 프로그램들은 매개변수들이 테라스페이스에 처음으로 복사된 것이 아니면 8바이트를 사용하여 각 매개변수를 액세스할 수 없습니다. 어플리케이션의 나머지 부분이 테라스페이스 함수를 최대로 활용하도록 만드는 한 가지 방법은 TERASPACE(*YES *TSIFC)와 DTAMDL(*P128) 옵션을 사용하여 프롬트 대체 프로그램을 작성하는 것입니다. 이 옵션을 사용하면 사용자의 프로그램이 테라스페이스를 사용하고 malloc 처리 시 테라스페이스를 확보하고 16바이트 포인터를 사용할 수 있습니다. 8바이트 포인터로 액세스되는 매개변수들은 malloc로 할당된 테라스페이스 안에 맨 먼저 복사시킬 수 있습니다.

  명령 처리 프로그램에 이와 같은 코드를 포함시킴으로서 명령에서 나온 매개변수를 테라스페이스 기억장치 모델과 8바이트 포인터를 사용하는 어플리케이션의 나머지 부분으로 전달할 수 있습니다.

```
#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 */
    ...
}
```

- *시나리오 14: 함수 재선언*

  함수 재선언 방지는 IBM이 제공하는 헤더 파일에 이미 선언되어 있습니다. 로컬 선언에는 올바른 포인터 길이가 지정되어 있지 않을 수 있습니다. 일반적으로 사용되는 그러한 인터페이스 중 하나가 **errno**이며, 이것은 OS/400에 하나의 함수로 구현되어 있습니다.

- *시나리오 15: 포인터를 리턴하는 프로그램 및 함수와 함께 자료 모델 *LLP64 사용*

  자료 모델 *LLP64를 사용할 경우 포인터를 리턴하는 프로그램과 함수를 잘 살펴보십시오. 포인터가 단일 레벨 기억장치를 가리킬 경우 그 값이 8바이트로 올바르게 지정될 수 없기 때문에 이 인터페이스의 클라이언트들이 16바이트 포인터로 리턴된 값을 반드시 유지시켜야 합니다. 그와 같은 API로 QUSPTRUS가 있습니다. 사용자 공간은 단일 레벨 기억장치에 상주합니다.

  포인터를 리턴하는 함수의 예로는 Java Native Interface (JNI) 함수 GetStringChars 그리고 GetByteArrayElements가 있습니다. 처음에는 단일 레벨 기억장치에 상주하는 Unicode 문자열로 포인터를 리턴하며 그 다음에는 역시 단일 레벨 기억장치에 상주하는 기본 배열(primitive array)이나 그 사본으로 포인터를 리턴합니다.

- *시나리오 16: JNI 함수 사용 시 문제 발생 방지*

  테라스페이스 기억장치를 사용하며 JNI 함수를 호출할 경우에는 PTF MF26929를 설치하십시오.

- *시나리오 17: 사용권 내부 코드 옵션 DetectConvertTo8BytePointerError를 사용하여 초기 디버깅 수행*

  STGMDL(*TERASPACE)로 작성된 테라스페이스 기억장치 모델 프로그램의 초기 디버깅의 경우 사용권 내부 코드 옵션 DetectConvertTo8BytePointerError를 사용할 것을 고려해보십시오. 모듈과 프로그램을 작성할 때 이 옵션을 사용하면 단일 레벨 기억장치 주소를 8바이트 포인터로 변환하는 시도를 할 때 MCH0609를 발생하는 코드가 생성될 수 있습니다.

- *시나리오 18: 사용권 내부 코드 옵션 MinimizeTeraspaceFalseEAOs 사용*

  16MB보다 큰 테라스페이스 기억장치 할당을 위한 주소지정에서 16바이트 포인터를 사용하는 프로그램의 경우 고급 최적화 기술에 설명되어 있는 사용권 내부 코드 옵션 MinimizeTeraspaceFalseEAOs를 사용할 것을 고려해 보십시오. 16바이트 포인터 대신 8바이트 포인터를 사용함으로써 EAO(유효 주소 넘침)을 줄일 수 있습니다.

필요하지 않을 때 이 옵션을 사용할 경우 15% 정도의 성능 저하가 발생할 수 있다는 점을 기억하십시오. 그러나 이 옵션을 올바로 사용한다면 60% 이상의 성능 개선을 가져올 수 있습니다.

제 5 장 프로그램 작성 개념

ILE 프로그램이나 서비스 프로그램 작성을 위한 처리는 사용자에게 어플리케이션을 설계하고 유지보수하는 데 보다 많은 융통성과 제어를 제공합니다. 이 처리에는 다음과 같은 두 단계가 포함됩니다.

1. 소스 코드를 모듈로 컴파일
2. 모듈을 ILE 프로그램이나 서비스 프로그램으로 바인딩. 프로그램 작성(CRTPGM) 또는 서비스 프로그램 작성(CRTSRVPGM) 명령이 실행될 때 바인딩이 발생합니다.

이 장에서는 바인더와 연관된 개념과 ILE 프로그램 또는 서비스 프로그램 작성 처리와 연관된 개념을 설명합니다. 이 장을 읽기 전에 13 페이지의 제 2 장『ILE 기본 개념』에 나오는 바인딩 개념을 이해하도록 하십시오.

프로그램 작성 및 서비스 프로그램 작성 명령

프로그램 작성(CRTPGM)과 서비스 프로그램 작성(CRTSRVPGM) 명령은 유사하며 같은 매개변수를 공유하고 있습니다. 두 명령에 사용되는 매개변수를 비교하면 각 명령의 사용 방법을 보다 분명하게 알 수 있습니다.

표 6에서는 제공되는 디폴트 값을 사용하고 있는 명령과 해당 매개변수를 보여줍니다.

표 6. CRTPGM 및 CRTSRVPGM 명령의 매개변수

매개변수 그룹	**CRTPGM 명령**	**CRTSRVPGM 명령**
ID	PGM(*CURLIB/WORK) MODULE(*PGM)	SRVPGM(*CURLIB/UTILITY) MODULE(*SRVPGM)
프로그램 액세스	ENTMOD(*FIRST)	EXPORT(*SRCFILE) SRCFILE(*LIBL/QSRVSRC) SRCMBR(*SRVPGM)
바인딩	BNDSRVPGM(*NONE) BNDDIR(*NONE)	BNDSRVPGM(*NONE) BNDDIR(*NONE)
실행시	ACTGRP(*ENTMOD)	ACTGRP(*CALLER)
최적화	IPA(*NO) IPACTLFILE(*NONE)	IPA(*NO) IPACTLFILE(*NONE)

표 6. *CRTPGM 및 CRTSRVPGM 명령의 매개변수 (계속)*

매개변수 그룹	**CRTPGM 명령**	**CRTSRVPGM 명령**
기타	OPTION(*GEN *NODUPPROC *NODUPVAR *WARN *RSLVREF)	OPTION(*GEN *NODUPPROC *NODUPVAR *WARN *RSLVREF)
	DETAIL(*NONE)	DETAIL(*NONE)
	ALWUPD(*YES)	ALWUPD(*YES)
	ALWRINZ(*NO)	ALWRINZ(*NO)
	REPLACE(*YES)	REPLACE(*YES)
	AUT(*LIBCRTAUT)	AUT(*LIBCRTAUT)
	TEXT(*ENTMODTXT)	TEXT(*ENTMODTXT)
	TGTRLS(*CURRENT)	TGTRLS(*CURRENT)
	USRPRF(*USER)	USRPRF(*USER)
	STGMDL(*SNGLVL)	STGMDL(*SNGLVL)

두 명령에 대한 ID 매개변수는 작성될 오브젝트와 복사될 모듈을 명명합니다. 두 매개변수의 유일한 차이점은 오브젝트 작성시 사용할 디폴트 모듈명에 있습니다. CRTPGM의 경우 프로그램(*PGM) 매개변수에서 지정된 것과 같은 모듈명을 사용하십시오. CRTSRVPGM의 경우 서비스 프로그램(*SRVPGM) 매개변수에서 지정된 것과 같은 모듈명을 사용하십시오. 그렇지 않으면 이들 매개변수는 동일하게 보이고 동일하게 활동합니다.

이 두 명령에서 가장 중요한 유사성은 바인더가 가져오기 및 내보내기 사이에 기호를 해결하는 방법입니다. 두 경우 모두 모듈(MODULE), 바인드 서비스 프로그램(BNDSRVPGM) 및 바인딩 디렉토리(BNDDIR) 매개변수에서의 입력을 바인더가 처리합니다.

명령에서 가장 중요한 차이점은 프로그램 액세스 매개변수입니다(85 페이지의 『프로그램 액세스』 참조). CRTPGM 명령의 경우 바인더에 식별되어야 하는 것은 어느 모듈에 프로그램 입력 프로시듀어가 있는가 하는 것 뿐입니다. 일단 프로그램이 작성되고 이 프로그램에 동적 프로그램 호출이 수행되면 처리는 프로그램 입력 프로시듀어가 있는 모듈로 시작됩니다. CRTSRVPGM 명령에는 더 많은 프로그램 액세스 정보가 필요합니다. 이것은 다른 프로그램이나 서비스 프로그램에 대해 몇 개의 액세스점 인터페이스를 제공할 수 있기 때문입니다.

허용된 권한 사용(QUSEADPAUT)

QUESADPAUT 시스템 값은 허용된 권한 사용(USEADPAUT(*YES)) 속성으로 프로그램을 작성할 수 있는 사용자가 누구인지를 정의합니다. QUSEADPAUT 시스템 값으로 권한을 부여 받은 모든 사용자의 경우 필요한 권한이 있으면 허용된 권한을 사용하는 프로그램과 서비스 프로그램을 작성하거나 변경할 수 있습니다. 필요한 권한에 대해 알아보려면 iSeries 보안 참조서를 참조하십시오.

시스템 값에는 권한 부여 리스트명이 있습니다. 사용자의 권한은 이 리스트에 대해 검사됩니다. 명명된 권한 부여 리스트에 대해 최소한 *USE 권한이 사용자에게 있으면 USEADPAUT(*YES) 속성으로 프로그램이나 서비스 프로그램을 작성, 변경, 갱신할 수 있습니다. 권한 부여 리스트에 대한 권한은 허용된 권한에서는 나올 수 없습니다.

권한 부여 리스트가 시스템 값에 명명되어 있고, 권한 부여 리스트가 누락된 경우 시도 중인 기능이 완료되지 않습니다. 메세지가 송신됩니다. 그러나 프로그램이 QPRCRTPG API로 작성되어 있고 *NOADPAUT 값이 옵션 템플리트에 지정된 경우 권한 부여 리스트가 없다고 할지라도 프로그램이 성공적으로 작성됩니다. 명령이나 API에 여러 기능이 요구되어 있고 권한 부여 리스트가 누락된 경우 기능이 수행되지 않습니다.

표 7. QUSEADPAUT에 가능한 값

값	설명
권한 부여 리스트명	다음 모든 사항에 해당되면 프로그램이 USEADPAUT(*NO)로 작성되었음을 나타내기 위해 진단 메세지가 표시됩니다. • 권한 부여 리스트가 QUSEADPAUT 시스템 값에 대해 지정되었습니다. • 사용자에게 위에서 언급한 권한 부여 리스트에 대한 권한이 없습니다. • 프로그램이나 서비스 프로그램이 작성될 때 오류가 발생하지 않았습니다. 사용자에게 권한 부여 리스트에 대한 권한이 있으면 프로그램이나 서비스 프로그램이 USEADPAUT(*YES)로 작성됩니다.
***NONE**	QUSEADPAUT 시스템 값에 의해 허가된 모든 사용자는 필요한 권한이 있는 사용자의 경우 허용된 권한을 사용하기 위한 프로그램과 서비스 프로그램을 작성하거나 변경할 수 있습니다. 필요한 권한에 대해 알아보려면 iSeries 보안 참조서를 참조하십시오.

QUSEADPAUT 시스템 값에 대한 자세한 정보는 보안 - 참조서를 참조하십시오.

최적화 매개변수 사용

ILE 바인드 프로그램 또는 서비스 프로그램을 더 최적화하려면 최적화 매개변수를 지정하십시오. 바인드 시간 최적화에 대한 자세한 정보는 171 페이지의 『프로시듀어간 분석(IPA)』을 참조하십시오.

기호 해결

기호 해결이란 바인더가 다음을 일치시키기 위해 수행하는 처리입니다.

- 복사에 의해 바인드될 모듈 세트에서의 가져오기 요구
- 지정된 모듈과 서비스 프로그램에 의해 제공되는 내보내기 세트

기호 해결 중에 사용될 내보내기 세트는 순서화(순차적으로 번호가 매겨진) 리스트로 간주될 수 있습니다. 내보내기 순서는 다음에 의해 결정됩니다.

- CRTPGM 또는 CRTSRVPGM 명령의 MODULE, BNDSRVPGM 및 BNDDIR 매개변수에서 오브젝트가 지정되는 순서

- 지정된 모듈의 언어 실행시 루틴에서의 내보내기

해결 및 미해결 가져오기

가져오기와 내보내기는 각각 프로시듀어 또는 자료 유형과 이름으로 이루어져 있습니다. **미해결 가져오기**는 유형 및 이름이 내보내기의 유형 및 이름과 일치하지 않습니다. **해결 가져오기**는 유형 및 이름이 내보내기의 유형 및 이름과 일치합니다.

복사에 의해 바인드된 모듈에서의 가져오기만 미해결된 가져오기 리스트에 들어갑니다. 기호 해결이 진행되는 중에는 그 다음의 미해결 가져오기가 순서화된 내보내기 리스트에서 일치사항을 탐색하는 데 사용됩니다. 순서화된 내보내기 세트를 검사한 후에 미해결 가져오기가 존재하면 일반적으로 프로그램 오브젝트나 서비스 프로그램이 작성되지 않습니다. 그러나 옵션 매개변수에 *UNRSLVREF가 지정되면 미해결 가져오기가 있는 프로그램 오브젝트나 서비스 프로그램이 작성될 수 있습니다. 그러한 프로그램 오브젝트나 서비스 프로그램이 실행시 미해결 가져오기의 사용을 시도하면 다음이 발생합니다.

- 프로그램 오브젝트나 서비스 프로그램이 버전 2 릴리스 3 시스템에 대해 작성되거나 갱신되면 오류 메세지 MCH3203이 발행됩니다. 이 메세지는 『기계 지시어의 기능 오류』를 말합니다.
- 프로그램 오브젝트나 서비스 프로그램이 버전 3 릴리스 1 이상 시스템에 대해 작성되거나 갱신되면 오류 메세지 MCH4439가 발행됩니다. 이 메세지 내용은 『해결되지 않은 가져오기 사용 시도』입니다.

복사에 의한 바인딩

MODULE 매개변수에 지정된 모듈은 항상 복사에 의해 바인드됩니다. BNDDIR 매개변수에 의해 지정된 바인딩 디렉토리에서 명명된 모듈은 그것이 필요할 경우 복사에 의해 바인드됩니다. 바인딩 디렉토리에서 명명된 모듈은 다음과 같은 경우에 필요합니다.

- 모듈이 미해결 가져오기에 대한 내보내기를 제공하는 경우
- 모듈이 서비스 프로그램을 작성하는 데 사용될 바인더 언어 소스 파일의 현재 내보내기 블록에서 명명된 내보내기를 제공하는 경우

바인더 언어에 있는 내보내기가 모듈 오브젝트에서 온 경우 그것이 명령 행에 명시적으로 제공된 것인지 아니면 바인딩 디렉토리에서 온 것인지와 관계 없이 해당 모듈은 항상 복사에 의해 바인드됩니다. 예를 들면, 다음과 같습니다.

```
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)
```

서비스 프로그램 SRV1에는 세 개의 모듈(M1, M2 및 M3)이 있습니다. P3이 현재 내보내기 블록에 있으므로 M3이 복사됩니다.

참조에 의한 바인딩

BNDSRVPGM 매개변수에서 지정된 서비스 프로그램은 참조에 의해 바인드됩니다. 바인딩 디렉토리에서 명명된 서비스 프로그램이 미해결 가져오기에 대한 내보내기를 제공할 경우 해당 서비스 프로그램은 참조에 의해 바인드됩니다. 이러한 방법으로 바인드된 서비스 프로그램은 새로운 가져오기를 추가하지 않습니다.

주: 어떤 것을 프로그램에 바인드시킬 것인지를 더 잘 제어하려면 총칭 서비스 프로그램명 또는 특정 라이브러리를 지정하십시오. 값 *LIBL은 어떤 것이 프로그램에 바인드될 것인지를 사용자가 정확히 알고 있는 사용자 제어 환경에서만 지정해야 합니다. OPTION(*DUPPROC *DUPVAR)과 함께 BNDSRVPGM(*LIBL/*ALL)을 지정하지 마십시오. *ALL과 함께 *LIBL을 지정하면 프로그램 실행시에 예기치 못한 결과가 발생할 수도 있습니다.

많은 수의 모듈을 바인딩

CRTPGM 및 CRTSRVPGM 명령의 모듈(MODULE) 매개변수에는 사용자가 지정할 수 있는 모듈의 수에 한계가 있습니다. 바인드하려는 모듈 수가 한계를 초과한 경우에는 다음 중 한 가지 방법을 사용할 수 있습니다.

1. 바인딩 디렉토리를 사용하여, 다른 모듈에서 필요한 내보내기를 제공하는 많은 수의 모듈을 바인드합니다.
2. CRTPGM 및 CRTSRVPGM 명령의 MODULE 매개변수에 총칭 모듈명이 지정될 수 있도록 하는 모듈 명명 규칙을 사용합니다. 예를 들어, `CRTPGM PGM(mylib/payroll) MODULE(mylib/pay*)`과 같이 사용할 수 있습니다. pay로 시작하는 모든 모듈명이 무조건적으로 프로그램 mylib/payroll에 포함됩니다. 그러므로, CRTPGM 또는 CRTSRVPGM 명령에 지정된 총칭명이 사용자가 원하지 않는 모듈을 바인드하지 않도록 주의하여 명명 규칙을 선택해야 합니다.
3. 값 *ALL이 MODULE 매개변수상의 특정 라이브러리명과 함께 사용될 수 있도록 모듈을 분리 라이브러리로 그룹화합니다. 예를 들어, `CRTPGM PGM(mylib/payroll) MODULE(payroll/*ALL)`과 같이 사용할 수 있습니다. 이 경우 라이브러리 payroll의 모든 모듈이 무조건적으로 프로그램 mylib/payroll에 포함됩니다.
4. 방법 2와 3에서 설명한 총칭명과 특정 라이브러리를 조합하여 사용합니다.
5. 서비스 프로그램의 경우 바인딩 소스 언어를 사용합니다. 모듈이 내보내기를 충족시킨다면, 바인딩 소스 언어에 지정된 내보내기로 인해 모듈이 바인드될 수 있습니다. RTVBNDSRC 명령을 사용하면 바인딩 소스 언어를 작성하는 데 도움이 됩니

다. RTVBNDSRC 명령의 MODULE 매개변수가 MODULE 매개변수에 명시적으로 지정될 수 있는 모듈 수를 제한하더라도, 총칭 모듈명 및 *ALL(특정 라이브러리명과 함께)을 사용할 수 있습니다. 또한 동일 소스 파일로 출력을 지시하여 RTVBNDSRC 명령을 여러 번 사용할 수 있습니다. 그러나 이 경우에는 바인딩 소스 언어를 편집해야 할 수도 있습니다.

내보내기 순서의 중요성

명령을 약간 변경하는 것만으로도 같지는 않지만 잠재적으로는 동등한 유효한 프로그램을 작성할 수 있습니다. MODULE, BNDSRVPGM 및 BNDDIR 매개변수에 오브젝트가 지정되는 순서는 일반적으로 다음 사항이 모두 참인 경우에만 중요합니다.

- 여러 모듈 또는 서비스 프로그램이 중복 기호명을 내보내기하고 있을 경우
- 다른 모듈이 기호명을 가져오기할 필요가 있을 경우

대부분의 어플리케이션에는 중복 기호가 없으므로, 프로그래머는 오브젝트가 지정되는 순서에 대해 걱정할 필요가 없습니다. 중복 기호를 내보내기시키며 또한 가져오기되는 어플리케이션의 경우에는 CRTPGM 또는 CRTSRVPGM 명령에 오브젝트가 나열되는 순서를 고려하십시오.

다음 예에서는 기호 해결이 이루어지는 방법을 보여줍니다. 81 페이지의 그림 30내의 모듈, 서비스 프로그램, 바인딩 디렉토리가 82 페이지의 그림 31 및 84 페이지의 그림 32에서 CRTPGM 요구에 사용됩니다. 식별된 내보내기 및 가져오기 모두를 프로시듀어로 가정하십시오.

또한 이 예에서는 프로그램 작성 처리에서의 바인딩 디렉토리의 역할도 보여줍니다. 라이브러리 MYLIB가 CRTPGM 명령과 CRTSRVPGM 명령의 라이브러리 리스트에 있는 것으로 가정하십시오. 다음의 명령이 라이브러리 MYLIB에 바인딩 디렉토리 L을 작성합니다.

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

다음 명령은 모듈 M1과 M2 그리고 서비스 프로그램 S와 T의 이름을 바인딩 디렉토리 L에 추가합니다.

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

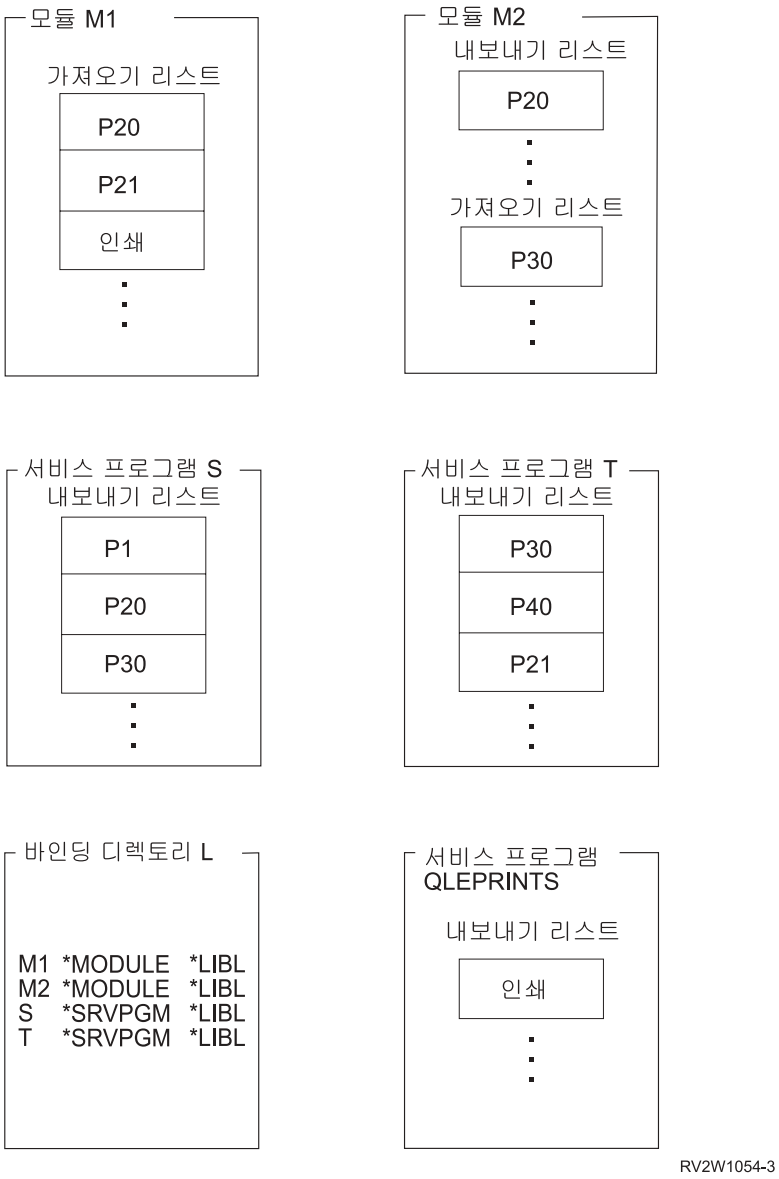

그림 30. 모듈, 서비스 프로그램, 바인딩 디렉토리

프로그램 작성 예 1

다음 명령이 82 페이지의 그림 31에 있는 프로그램 A를 작성하는 데 사용되는 것으로 가정하십시오.

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

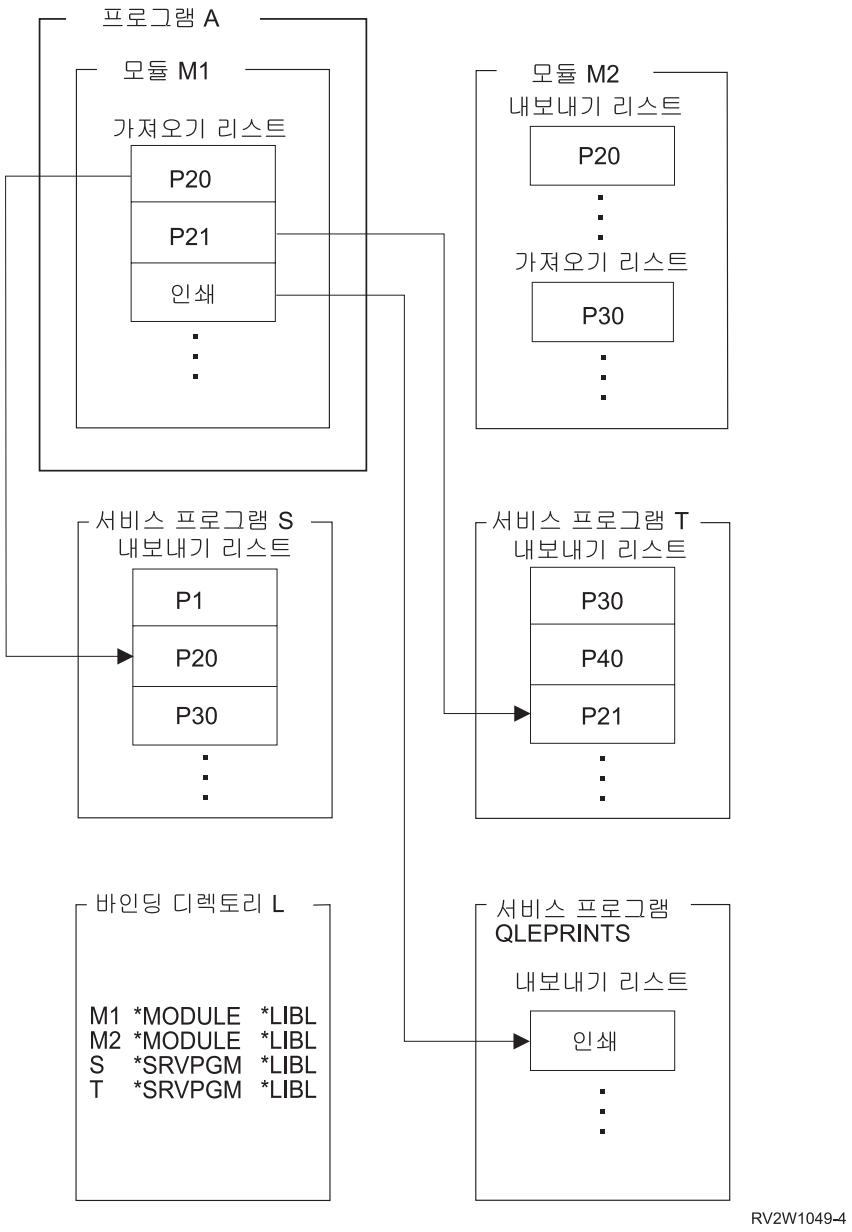

그림 31. 기호 해결 및 프로그램 작성: 예 1

프로그램 A를 작성하기 위해 바인더가 CRTPGM 명령 매개변수에 지정된 오브젝트를 지정된 순서로 처리합니다.

1. 첫 번째 매개변수(PGM)에 지정된 값은 A로, 작성될 프로그램의 이름입니다.
2. 두 번째 매개변수(모듈)에 지정된 값은 M1입니다. 바인더는 그곳에서 시작됩니다. 모듈 M1에는 해결되어야 할 세 개의 가져오기(P20, P21 및 인쇄)가 들어 있습니다.
3. 세 번째 매개변수(BNDSRVPGM)에 지정된 값은 S입니다. 바인더는 미해결 가져오기 요구를 해결하는 프로시듀어에 대해 서비스 프로그램 S의 내보내기 리스트를 스캔합니다. 내보내기 리스트에 프로시듀어 P20이 들어 있으므로, 해당 가져오기 요구가 해결됩니다.

4. 네 번째 매개변수(BNDDIR)에 지정된 값은 L입니다. 바인더가 다음으로 바인딩 디렉토리 L을 스캔합니다.
   a. 바인딩 디렉토리에 지정된 첫 번째 오브젝트는 모듈 M1입니다. 모듈 M1은 모듈 매개변수에서 지정되었기 때문에 현재 알려져 있으나 내보내기를 제공하지는 않습니다.
   b. 바인딩 디렉토리에 지정된 두 번째 오브젝트는 모듈 M2입니다. 모듈 M2는 내보내기를 제공하지만 그 중 어떠한 것도 현재 해결되지 않은 가져오기 요구(P21 및 인쇄)와 일치하지 않습니다.
   c. 바인딩 디렉토리에 지정된 세 번째 오브젝트는 서비스 프로그램 S입니다. 서비스 프로그램 S는 이미 82 페이지의 3단계에서 처리되었고, 추가의 내보내기를 제공하지 않습니다.
   d. 바인딩 디렉토리에 지정된 네 번째 오브젝트는 서비스 프로그램 T입니다. 바인더가 서비스 프로그램 T의 내보내기 리스트를 스캔합니다. 프로시듀어 P21이 발견되고, 해당 가져오기 요구를 해결합니다.
5. 해결되어야 할 최종 가져오기(인쇄)는 매개변수에 지정되지 않습니다. 그럼에도 바인더는 서비스 프로그램 QLEPRINTS의 내보내기 리스트에서 인쇄 프로시듀어를 발견하게 되는데, 이는 이 예에서 컴파일러에 의해 제공되는 공통 실행시 루틴입니다. 모듈 컴파일시, 컴파일러는 컴파일러가 소유하는 실행시 서비스 프로그램과 ILE 실행시 서비스 프로그램이 들어 있는 바인딩 디렉토리를 디폴트로 지정합니다. 이것이 바로 바인더가 컴파일러에 의해 제공된 실행시 서비스 프로그램에서 나머지 미해결 참조를 찾아야 하는 방법입니다. 바인더가 실행시 서비스 프로그램을 탐색한 후 해결할 수 없는 참조가 있으면 바인드는 대개 실패합니다. 그러나 작성 명령에서 OPTION(*UNRSLVREF)을 지정하면 프로그램이 작성됩니다.

프로그램 작성 예 2

84 페이지의 그림 32에서는 BNDSRVPGM 매개변수의 서비스 프로그램이 제거되었음을 제외하면 CRTPGM 요구와 유사한 결과를 보여줍니다.

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

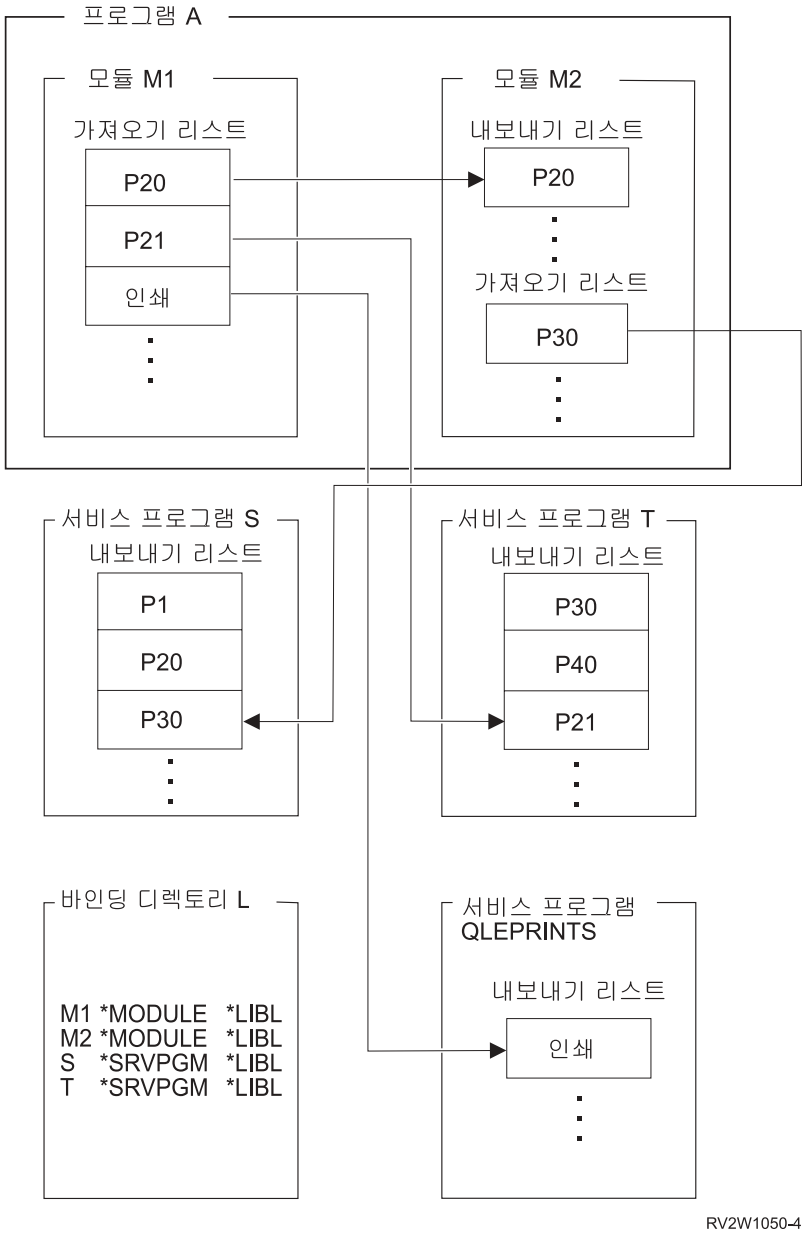

그림 32. 기호 해결 및 프로그램 작성: 예 2

처리될 오브젝트 순서를 변경하는 것이 내보내기 순서를 변경시킵니다. 또한 예 1에서 작성된 프로그램과 다른 프로그램이 작성되도록 합니다. 서비스 프로그램 S가 CRTPGM 명령의 BNDSRVPGM 매개변수에 지정되지 않았기 때문에 바인딩 디렉토리가 처리됩니다. 모듈 M2가 프로시듀어 P20을 내보내기하고, 서비스 프로그램 S 앞의 바인딩 디렉토리에 지정됩니다. 따라서, 모듈 M2는 이 예에서 결과 프로그램 오브젝트로 복사됩니다. 82 페이지의 그림 31과 그림 32를 비교해보면 다음을 알 수 있습니다.

- 예 1의 프로그램 A에는 모듈 M1만 있고 서비스 프로그램 S, T 및 QLEPRINTS의 프로시듀어를 사용합니다.
- 예 2의 프로그램 A에서는 두 개의 모듈 M1과 M2가 서비스 프로그램 T와 QLEPRINTS를 사용합니다.

예 2의 프로그램은 다음과 같이 작성됩니다.

1. 첫 번째 매개변수(PGM)는 작성될 프로그램명을 지정합니다.
2. 두 번째 매개변수(MODULE)에 지정된 값은 M1이고 바인더가 다시 여기에서 시작됩니다. 모듈 M1에는 해결되어야 할 동일한 세 개의 가져오기(P20, P21 및 인쇄)가 있습니다.
3. 이때 세 번째 지정된 매개변수는 BNDSRVPGM이 아니라 BNDDIR입니다. 그러므로, 바인더는 우선 지정된 바인딩 디렉토리(L)를 스캔합니다.
   a. 바인딩 디렉토리에 지정된 첫 번째 항목은 모듈 M1입니다. 이 라이브러리의 모듈 M1은 이미 모듈 매개변수에 의해 처리됩니다.
   b. 바인딩 디렉토리에 지정된 두 번째 항목은 모듈 M2에 대한 것입니다. 바인더는 모듈 M2의 내보내기 리스트를 스캔합니다. 해당 내보내기 리스트에 P20이 들어 있으므로 가져오기 요구가 해결됩니다. 모듈 M2는 복사에 의해 바인드되고 가져오기는 처리를 위해 미해결 가져오기 요구 리스트에 추가되어야 합니다. 현재 미해결 가져오기 요구는 P21, 인쇄 및 P30입니다.
   c. 바인딩 디렉토리, 'S' 서비스 프로그램에 지정된 다음 오브젝트로 처리가 계속됩니다. 이 경우 서비스 프로그램 S에서 현재 해결되지 않은 P21의 가져오기와 인쇄 요구로 P30 내보내기를 제공합니다. 바인딩 디렉토리에 나열된 다음 오브젝트, 서비스 프로그램 T로 처리가 계속됩니다.
   d. 서비스 프로그램 T는 미해결 가져오기에 대해 P21 내보내기를 제공합니다.
4. 예 1에서와 같이, 가져오기 요구 인쇄는 지정되지 않습니다. 그러나 모듈 M1이 작성된 언어에서 제공되는 실행시 루틴 안에서 프로시듀어가 발견됩니다.

기호 해결은 내보내기의 강도(strength)에 의해서도 영향을 받습니다. 약 반출 및 강 반출에 관한 정보는 88 페이지의 『가져오기 및 내보내기 개념』에서 내보내기 부분을 참조하십시오.

프로그램 액세스

ILE 프로그램 오브젝트나 서비스 프로그램 오브젝트를 작성할 경우 다른 프로그램이 해당 프로그램에 액세스할 수 있는 방법을 지정해야 합니다. CRTPGM 명령에서 모듈 입력(ENTMOD) 매개변수를 사용하여 지정합니다. CRTSRVPGM 명령에서는 내보내기(EXPORT) 매개변수로 이와 같이 수행할 수 있습니다(75 페이지의 표 6 참조).

CRTPGM 명령의 프로그램 입력 프로시듀어 모듈 매개변수

프로그램 입력 프로시듀어 모듈(ENTMOD) 매개변수는 바인더에 다음을 찾을 수 있는 모듈의 이름을 알려줍니다.

PEP(프로그램 입력 프로시듀어)
UEP(사용자 입력 프로시듀어)

이 정보는 작성된 프로그램에 동적 호출이 이루어질 때 제어권을 받는 PEP가 있는 모듈을 식별합니다.

ENTMOD 매개변수에 대한 디폴트 값은 *FIRST입니다. 이 값은 바인더가 PEP가 들어 있는 모듈 매개변수에 지정된 모듈 리스트에서 첫 번째로 발견되는 모듈을 입력 모듈로 사용하도록 지정합니다.

다음 조건이 존재하는 경우
ENTMOD 매개변수에 *FIRST가 지정됨
PEP가 있는 두 번째 모듈이 발견됨

바인더는 이 두 번째 모듈을 프로그램 오브젝트로 복사하고 바인딩 처리를 계속합니다. 바인더는 추가 PEP를 무시합니다.

ENTMOD 매개변수에 *ONLY가 지정되면 프로그램에서 하나의 모듈에만 PEP를 가집니다. *ONLY가 지정되었는데 PEP가 있는 두 번째 모듈이 발견될 경우 프로그램이 작성되지 않습니다.

명시적인 제어를 위해 PEP가 들어 있는 모듈명을 지정할 수 있습니다. 다른 PEP는 무시됩니다. 명시적으로 지정된 모듈에 PEP가 들어 있지 않으면 CRTPGM 요구는 실패합니다.

모듈에 프로그램 입력 프로시듀어가 들어 있는지 알아보려면 모듈 표시(DSPMOD) 명령을 사용할 수 있습니다. 모듈 정보 화면 표시의 *프로그램 입력 프로시듀어명* 필드에 정보가 나옵니다. 필드에 *NONE이 지정되어 있으면 이 모듈에는 PEP가 없음을 나타냅니다. 필드에 이름이 지정되면 이 모듈은 PEP를 가집니다.

CRTSRVPGM 명령의 내보내기 매개변수

내보내기(EXPORT), 소스 파일(SRCFILE) 및 소스 멤버(SRCMBR) 매개변수는 작성되어 있는 서비스 프로그램의 공용 인터페이스를 식별합니다. 매개변수는 서비스 프로그램이 다른 ILE 프로그램이나 서비스 프로그램에 의해 사용될 수 있도록 만드는 내보내기(프로시듀어와 자료)를 지정합니다.

내보내기 매개변수의 디폴트 값은 *SRCFILE입니다. 해당 값은 서비스 프로그램 내보내기에 관한 정보를 참조하기 위해 SRCFILE 매개변수로 바인더를 보냅니다. 추가 정보는 그 안에 바인더 언어 소스가 있는 소스 파일입니다(90 페이지의 『바인더 언어』 참조). 바인더는 바인더 언어 소스를 찾아 지정된 이름에서 내보내기되도록 하나 이상의 서명을 생성합니다. 바인더 언어를 사용하면 바인더가 서명을 생성하는 대신 사용자가 선택한 서명을 지정할 수도 있습니다.

바인더 소스 검색(RTVBNDSRC) 명령은 바인더 언어 소스가 들어 있는 소스 파일을 작성하는 데 사용될 수 있습니다. 소스는 기존 서비스 프로그램 또는 모듈 세트 중 하나를 기반으로 할 수 있습니다. 서비스 프로그램을 기반으로 하는 경우 소스는 해당 서

비스 프로그램 재작성 또는 갱신에 적합합니다. 모듈 세트를 기반으로 하는 경우 소스는 모듈에서 내보낼 수 있는 모든 기호를 포함합니다. 어느 경우에든, 사용자는 이 파일을 편집하여 내보내기할 기호만 포함시킨 후 CRTSRVPGM 또는 UPDSRVPGM 명령의 SRCFILE 매개변수에 이 파일을 지정할 수 있습니다.

내보내기 매개변수로 가능한 다른 값은 *ALL입니다. EXPORT(*ALL)가 지정되면 복사된 모듈에서 내보내기된 모든 기호가 서비스 프로그램에서 내보내기됩니다. 생성되는 서명은 다음에 의해 결정됩니다.

- 내보내기된 기호의 수
- 내보내기된 기호의 영문 순서

EXPORT(*ALL)가 지정되면 서비스 프로그램에서의 내보내기를 정의하는 데 바인더 언어가 필요하지 않습니다. 바인더 언어 소스를 생성하지 않아도 되므로, 이 값을 사용하는 것이 가장 쉽습니다. 그러나 EXPORT(*ALL)가 지정된 서비스 프로그램은 일단 다른 프로그램에 의해 내보내기가 사용되면 갱신하거나 정정하기가 어렵습니다. 서비스 프로그램이 변경되면 내보내기 순서 또는 수가 변경될 수 있습니다. 따라서, 해당 서비스 프로그램의 서명도 변경될 수 있습니다. 서명이 변경되면 변경된 서비스 프로그램을 사용하는 모든 프로그램이나 서비스 프로그램을 재작성해야 합니다.

EXPORT(*ALL)는 서비스 프로그램에서 사용된 모듈에서 내보내기된 모든 기호가 서비스 프로그램에서 내보내기된 것임을 나타냅니다. ILE C는 글로벌 또는 정적으로서 내보내기를 정의할 수 있습니다. 글로벌로서 ILE C에 선언된 외부 변수만
EXPORT(*ALL)로 사용할 수 있습니다. ILE RPG에서 다음은 EXPORT(*ALL)로 사용할 수 있습니다.

- RPG 기본 프로시듀어명
- 모든 내보낸 서브프로시듀어의 이름
- 키워드 EXPORT를 사용하여 정의된 변수

ILE COBOL에서 다음 언어 요소는 모듈 내보내기입니다.

- 컴파일 단위의 사전적으로 가장 외부에 있는 COBOL 프로그램(*PGM 오브젝트와 혼동하지 말 것)의 PROGRAM-ID 단락에 있는 이름. 이것은 강 프로시듀어 내보내기에 맵핑됩니다.
- 프로그램에 INITIAL 속성이 없는 경우 앞 문장에서 불릿한 PROGRAM-ID 단락에 있는 이름에서 파생된 COBOL 컴파일러 생성명. 이것은 강 프로시듀어 내보내기에 맵핑됩니다. 약 반출 및 강 반출에 관한 정보는 88 페이지의 『가져오기 및 내보내기 개념』에서 내보내기 부분을 참조하십시오.
- EXTERNAL로 선언된 자료 항목 또는 파일 항목. 이것은 약 반출에 맵핑됩니다.

소스 파일과 소스 멤버 매개변수에 사용된 내보내기 매개변수

내보내기 매개변수에서의 디폴트 값은 *SRCFILE입니다. 내보내기 매개변수에 *SRCFILE이 지정되면 바인더는 SRCFILE 및 SRCMBR 매개변수를 사용하여 바인더 언어 소스를 찾으십시오.

다음 예의 명령은 디폴트를 사용하여 바인더 언어 소스를 찾아 UTILITY로 명명된 서비스 프로그램을 바인드합니다.

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

이 명령이 서비스 프로그램을 작성할 경우 소스 파일 QSRVSRC에 UTILITY로 명명된 멤버가 있어야 합니다. 그리고 이 멤버에는 바인더가 서명과 내보내기 ID 세트로 변환하는 바인더 언어 소스가 있어야 합니다. 디폴트는 서비스 프로그램명 UTILITY와 이름이 같은 멤버에서 바인더 언어 소스를 확보하는 것입니다. 이들 매개변수에서 제공된 값이 있는 파일, 멤버 또는 바인더 언어 소스가 없을 경우 서비스 프로그램은 작성되지 않습니다.

SRCFILE 매개변수를 위한 파일의 최대 폭

V3R7 이후의 릴리스에서 CRTSRVPGM 또는 UPDSRVPGM 명령의 소스 파일(SRCFILE) 매개변수에 대한 최대 파일 폭은 240자입니다. 파일이 최대 폭보다 크면, CPF5D07 메세지가 표시됩니다. V3R2의 경우 최대 폭은 80자입니다. V3R6, V3R1, V2R3의 경우에는 최대 폭에 대한 제한이 없습니다.

가져오기 및 내보내기 개념

ILE 언어는 다음과 같은 유형의 가져오기와 내보내기를 지원합니다.

- 약 자료 내보내기
- 약 자료 가져오기
- 강 자료 내보내기
- 강 자료 가져오기
- 강 프로시듀어 내보내기
- 약 프로시듀어 내보내기
- 프로시듀어 가져오기

ILE 모듈 오브젝트는 내보내기 프로시듀어나 자료 항목을 다른 모듈로 내보내기할 수 있습니다. 그리고 ILE 모듈 오브젝트는 다른 모듈에서 프로시듀어나 자료 항목을 가져오기(참조)할 수 있습니다. 서비스 프로그램을 작성하기 위해 CRTSRVPGM 명령에 모듈 오브젝트를 사용할 때 해당 내보내기가 서비스 프로그램에서 선택적으로 내보내기됩

니다(86 페이지의 『CRTSRVPGM 명령의 내보내기 매개변수』 참조). 내보내기의 강도(강약)는 프로그래밍 언어에 따라 다릅니다. 강도는 자료 항목의 크기와 같은 특성을 설정하는 데 있어서 내보내기에 관해 충분히 알게 되는 시기를 결정합니다. 강 반출의 특성은 바인드시 설정됩니다. 내보내기의 강도가 기호 해결에 영향을 줍니다.

- 하나 이상의 약 반출에 같은 이름이 있을 경우 바인더가 강 반출의 특성을 사용합니다.
- 약 반출이 강 반출과 다른 이름이 있을 경우 활성화될 때까지 특성을 설정할 수 없습니다. 활성화 시점에 동일한 이름의 여러 약 반출이 있으면 프로그램이 가장 큰 내보내기를 사용합니다. 이것은 동일한 이름으로 이미 활성화된 약 반출이 그 특성을 이미 설정하지 않은 경우에 해당됩니다.
- 바인드시 바인딩 디렉토리가 사용되고 약반입에 일치하는 약 반출이 발견될 경우 바인드가 이루어집니다. 이것은 미해결 가져오기가 해결될 경우의 바인딩 검색된 디렉토리에만 해당됩니다. 일단 모든 가져오기가 해결되면 바인딩 디렉토리 항목을 통한 탐색이 중단됩니다. 중복 약 반출은 중복 변수나 프로시듀어로 플래그되지 않습니다. 바인딩 디렉토리 안에서 항목의 순서는 매우 중요합니다.

활성화시 해결을 위해 약 반출을 프로그램 오브젝트나 서비스 프로그램 외부로 내보내기할 수 있습니다. 이것은 단지 서비스 프로그램 외부로 바인드시에만 내보내기될 수 있는 강 반출과 반대되는 점입니다.

그러나 프로그램 오브젝트 외부로는 강 반출을 내보내기할 수 없습니다. 바인드시 다음 중 하나를 충족시키기 위해서는 서비스 프로그램 외부로 강 프로시듀어를 내보내기할 수 있습니다.

- 서비스 프로그램을 참조에 의해 바인드하는 프로그램으로 가져오기
- 해당 프로그램에 대한 참조에 의해 바인드되는 다른 서비스 프로그램으로 가져오기

서비스 프로그램은 바인딩 소스 언어를 통해 해당 공용 인터페이스를 정의합니다.

바인딩 소스 언어를 통해 서비스 프로그램을 위한 공용 인터페이스의 약 프로시듀어 내보내기 부분을 작성할 수 있습니다. 그러나 바인딩 소스 언어를 통해 서비스 프로그램에서 약 프로시듀어 내보내기를 하는 것은 더이상 약 반출로 표시되지 않습니다. 이것은 강 프로시듀어 내보내기로 처리됩니다.

약 자료는 활성 그룹으로 내보내기만 할 수 있습니다. 그것은 바인더 언어 소스를 통해 서비스 프로그램에서 내보내기되는 공용 인터페이스의 일부가 될 수 없습니다. 바인더 소스 언어에 약 자료를 지정하는 것은 바인드 실패의 원인입니다.

90 페이지의 표 8에는 일부 ILE 언어에 의해 지원되는 가져오기 및 내보내기를 보여줍니다.

표 8. ILE 언어에 의해 지원되는 가져오기 및 내보내기

ILE 언어	약 자료 내보내기	약 자료 가져오기	강 자료 내보내기	강 자료 가져오기	강 프로시듀어 내보내기	약 프로시듀어 내보내기	프로시듀어 가져오기
RPG IV	아니오	아니오	예	예	예	아니오	예
COBOL[2]	예[3]	예[3]	아니오	아니오	예[1]	아니오	예
CL	아니오	아니오	아니오	아니오	예[1]	아니오	예
C	아니오	아니오	예	예	예	아니오	예
C++	아니오	아니오	예	예	예	예	예

주:

1. COBOL 및 CL은 하나의 프로시듀어만 모듈에서 내보내기될 수 있도록 합니다.
2. COBOL은 약 자료 모델을 사용합니다. 외부 자료로 선언된 자료 항목은 해당 모듈에 대해 약 반출 및 약반입 모두가 될 수 있습니다.
3. COBOL은 대소문자 NOMONOPRC 옵션을 사용합니다. 이 옵션이 없으면 소문자가 자동으로 대문자로 변환됩니다.

특정 언어에 대해 어떤 선언이 가져오기 및 내보내기되는지 알려면 다음 중 하나를 참조하십시오.

- 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

바인더 언어

바인더 언어는 서비스 프로그램에 대한 내보내기를 정의하는 실행불가능한 명령 세트입니다. 바인더 언어는 BND 소스 유형이 지정될 때 입력을 프롬트하고 유효성 검사를 할 수 있도록 소스 입력 유틸리티(SEU) 구문 검사 프로그램을 작동가능하게 합니다.

주: 와일드카드가 포함되어 있는 바인더 소스 파일에 대해서는 SEU 구문 검사 유형인 BND를 사용할 수 없습니다. 또한 254자보다 긴 이름의 바인더 소스 파일에 대해서도 사용할 수 없습니다.

바인더 언어는 다음과 같은 명령의 리스트로 구성되어 있습니다.

1. 서비스 프로그램에서 내보내기 리스트의 시작을 식별하는 프로그램 내보내기 시작(STRPGMEXP) 명령
2. 서비스 프로그램에서 내보내기될 기호명을 식별하는 각각의 기호 내보내기(EXPORT) 명령
3. 서비스 프로그램에서 내보내기 리스트의 끝을 식별하는 프로그램 내보내기 종료(ENDPGMEXP) 명령

그림 33은 소스 파일에 있는 바인더 언어의 샘플입니다.

```
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
```

그림 33. 소스 파일에 있는 바인더 언어 예

바인더 소스 검색(RTVBNDSRC) 명령은 하나 이상의 모듈에서 내보내기된 것에 기초하여 바인더 언어 소스 생성을 돕는 데 사용될 수 있습니다.

서명

한 쌍의 STRPGMEXP PGMLVL(*CURRENT)과 ENDPGMEXP 사이에서 식별된 기호는 서비스 프로그램에 대한 공용 인터페이스를 정의합니다. 해당 공용 인터페이스는 **서명**으로 표시됩니다. 서명은 서비스 프로그램에 의해 지원되는 인터페이스를 식별하는 값입니다.

주: 이 주제에서 설명한 서명을 *디지털 오브젝트 서명*과 혼동하지 마십시오. OS/400 오브젝트에 있는 디지털 서명은 소프트웨어 및 자료의 무결성을 보장합니다. 또한 자료 변경, 바이러스 도입 또는 오브젝트에 대한 권한이 없는 수정을 방지하는 역할을 합니다. 또한 서명은 자료 기점의 양의 식별을 제공합니다. 디지털 오브젝트 서명에 대한 자세한 정보는 iSeries Information Center에서 정보의 **보안** 범주를 참조하십시오.

명시적 서명을 지정하지 않으면 바인더는 내보내기될 프로시듀어 및 자료 항목명의 리스트에서 그리고 지정된 순서에서 서명을 생성합니다. 그러므로, 서명은 서비스 프로그램에 대한 공용 인터페이스를 인증하는 데 쉽고도 편리한 방법을 제공합니다. 서명은 서비스 프로그램내에 있는 특정 프로시듀어에 대한 인터페이스를 인증하지 않습니다.

주: 서비스 프로그램에 대한 비호환 변경을 피하기 위해 기존 프로시듀어와 자료 항목 명을 바인더 언어 소스에서 제거시키거나 재배열시켜서는 안됩니다. 추가 내보내기 블록에는 기존 내보내기 블록과 같은 순서의 동일한 기호를 반드시 포함시켜야 합니다. 추가 기호는 리스트의 끝에만 추가시켜야 합니다.

프로그램이나 해당 서비스 프로그램에 바인드된 서비스 프로그램에서 내보내기가 필요할 수 있으므로, 기존 프로그램 및 서비스 프로그램과 호환되는 방식으로 서비스 프로그램 내보내기를 제거할 수 없습니다.

서비스 프로그램에 비호환이 변경된 경우 서비스 프로그램에 바인드된 상태로 남아 있는 기존 프로그램이 더이상 올바로 작동하지 못할 수 있습니다. 서비스 프로그램에 대한 비호환 변경은 해당 서비스 프로그램에 바인드된 모든 프로그램 및 서비스 프로그램에 비호환이 변경된 후 **CRTPGM** 또는 **CRTSRVPGM**으로 재작성될 수 있음을 보장받는 경우에만 가능합니다.

프로그램 내보내기 시작 및 프로그램 내보내기 종료 명령

프로그램 내보내기 시작(STRPGMEXP) 명령은 서비스 프로그램에서 내보내기 리스트의 시작을 식별합니다. 프로그램 내보내기 종료(ENDPGMEXP) 명령은 서비스 프로그램에서 내보내기 리스트의 끝을 식별합니다.

소스 파일내에 여러 쌍의 STRPGMEXP와 ENDPGMEXP가 지정되면 다수의 서명이 작성됩니다. STRPGMEXP와 ENDPGMEXP의 쌍이 발견되는 순서는 중요하지 않습니다.

STRPGMEXP 명령의 프로그램 레벨 매개변수

하나의 STRPGMEXP 명령만 PGMLVL(*CURRENT)을 지정할 수 있으나 그것이 첫 번째 STRPGMEXP 명령일 필요는 없습니다. 소스 파일내에 있는 기타 모든 STRPGMEXP 명령은 PGMLVL(*PRV)을 지정해야 합니다. 현재 서명은 어느 STRPGMEXP 명령이 PGMLVL(*CURRENT)을 지정했는지를 나타냅니다.

STRPGMEXP 명령의 레벨 검사 매개변수

STRPGMEXP 명령에서 레벨 검사(LVLCHK) 매개변수는 바인더가 서비스 프로그램에 대한 공용 인터페이스를 자동으로 검사해야 하는지 여부를 지정합니다.
LVLCHK(*YES)를 지정하거나 디폴트 값을 LVLCHK(*YES)로 하면 바인더가 서명 매개변수를 조사합니다. 서명 매개변수는 바인더가 명시적인 서명 값을 사용하는지 아니면 0 이외의 서명 값을 생성하는지 여부를 결정합니다. 바인더가 서명 값을 생성하는 경우 시스템이 해당 값과 서비스 프로그램 클라이언트에 알려진 값과의 일치 여부를 확인합니다. 값이 일치하면 서비스 프로그램의 클라이언트는 재컴파일 없이 공용 인터페이스를 사용할 수 있습니다.

LVLCHK(*NO)를 지정하면 자동 서명 검사를 할 수 없게 됩니다. 다음 상황에서는 이 피처를 사용하도록 결정할 수 있습니다.

- 서비스 프로그램의 인터페이스에 대한 특정 변경사항이 호환불가를 초래하지 않는다는 것을 사용자가 아는 경우
- 바인더 언어 소스 파일의 갱신이나 클라이언트의 재컴파일을 원하지 않을 경우

LVLCHK(*NO) 값을 사용할 때는 주의하십시오. 이 값을 사용하게 되면 공용 인터페이스가 이전 레벨과 호환가능한지를 사용자가 수동으로 검증해야 하기 때문입니다. 서비스 프로그램의 어떤 프로시듀어를 호출할 것이며, 어떤 변수가 클라이언트에 의해 사용될 것인지를 사용자가 제어할 수 있을 때에만 LVLCHK(*NO)를 지정하십시오. 공용 인터페이스를 제어할 수 없을 경우 실행시 활성 오류가 발생합니다. 바인더 언어 사용시 발생할 수 있는 일반적인 오류에 관한 설명은 205 페이지의 『바인더 언어 오류』 부분을 참조하십시오.

STRPGMEXP 명령의 서명 매개변수

서명(SIGNATURE) 매개변수를 사용하여 서비스 프로그램에 대한 서명을 명시적으로 지정할 수 있습니다. 명시적 서명은 16진 스트링이나 문자 스트링이 될 수 있습니다. 다음과 같은 이유로 서명의 명시적 지정을 고려해 볼 수 있습니다.

- 바인더는 사용자가 원하지 않는 호환가능한 서명을 생성할 수 있습니다. 서명은 지정된 내보내기의 이름과 순서에 기초합니다. 그러므로, 두 개의 내보내기 블록에 동일한 순서의 동일한 내보내기가 있다면, 이들은 동일한 서명을 가집니다. 서비스 프로그램 제공자로서 사용자는 두 개의 인터페이스가 호환되지 않다는 것을(예를 들어, 해당 매개변수 리스트가 서로 다르기 때문에) 알 수 있습니다. 이 경우 바인더가 호환가능한 서명을 생성하도록 하는 대신 사용자가 새로운 서명을 명시적으로 지정할 수 있습니다. 새로운 서명을 지정할 때에는 서비스 프로그램에 호환불가를 작성하여, 일부 또는 모든 클라이언트가 재컴파일되도록 하십시오.
- 바인더는 사용자가 원하지 않는 호환불가능한 서명을 생성할 수 있습니다. 두 개의 내보내기 블록에 다른 내보내기나 다른 순서가 있는 경우 이들은 다른 서명을 갖습니다. 서비스 프로그램 제공자로서 사용자가 두 개의 인터페이스가 호환가능하다는 것을(예를 들어, 기능명이 변경되었으나 계속 동일한 기능으로 유지되는 것을 보고) 알고 있는 경우 바인더가 호환불가능한 서명을 생성하도록 하는 대신 사용자가 바인더에 의해 이전에 생성된 것과 동일한 서명을 명시적으로 지정할 수 있습니다. 동일한 서명을 지정하는 경우 서비스 프로그램의 호환가능성을 유지보수하여 사용자의 클라이언트가 사용자의 서비스 프로그램을 리바인드하지 않고도 사용할 수 있도록 하십시오.

서명 매개변수의 디폴트 값 *GEN은 바인더가 내보내기된 기호에서 서명을 생성하도록 합니다.

서비스 프로그램 표시(DSPSRVPGM) 명령을 사용하고 DETAIL(*SIGNATURE)을 지정하여 서비스 프로그램에 대한 서명 값을 결정할 수 있습니다.

기호 내보내기 명령

기호 내보내기(EXPORT) 명령은 서비스 프로그램에서 내보내기하는 데 사용될 수 있는 기호명을 식별합니다.

만일 내보내기된 기호에 소문자로 된 문자가 포함되어 있으면 91 페이지의 그림 33에서와 같이 기호명을 어포스트로피로 묶어야 합니다. 어포스트로피를 사용하지 않으면 기호명은 모두 대문자로 변환됩니다. 예를 들면, 바인더는 p1이 아니라 P1로 명명된 내보내기를 탐색합니다.

기호명 또한 와일드카드 문자(<<< 또는 >>>)를 사용하여 내보내기할 수 있습니다. 기호명이 있고 지정된 와일드카드와 일치하면 기호명이 내보내기됩니다. 다음 조건 중 하나라도 존재하면 오류가 표시되고 서비스 프로그램은 작성되지 않습니다.

- 어떠한 기호명도 지정된 와일드카드와 일치하지 않는 경우
- 둘 이상의 기호명이 지정된 와일드카드와 일치하는 경우
- 기호명이 지정된 와일드카드와 일치하지만 내보내기를 위해 사용할 수 없는 경우

와일드카드 스펙에서 서브스트링을 따옴표로 묶어야 합니다.

서명은 와일드카드 스펙의 문자에 의해 판별됩니다. 와일드카드 스펙을 변경하면 변경된 와일드카드 스펙이 같은 내보내기와 일치하더라도 서명을 변경합니다. 예를 들어, 두 개의 와일드카드 스펙 『r』>>> 및 『ra』>>> 모두 『rate』 기호를 내보내기할 수 있으나 서로 다른 두 개의 서명을 작성합니다. 그러므로, 가능한 한 기호 내보내기와 유사한 와일드카드 스펙을 사용하는 것이 좋습니다.

주: 와일드카드가 포함되어 있는 바인더 소스 파일에 대해서는 SEU 구문 검사 유형인 BND를 사용할 수 없습니다.

와일드카드 내보내기 기호 예

다음 예에서 가능한 내보내기 기호 리스트가 다음과 같이 구성되어 있는 것으로 가정하십시오.

```
interest_rate
international
prime_rate
```

다음 예는 어떤 내보내기가 선택되는지 또는 오류가 왜 발생하는지를 보여줍니다.

EXPORT SYMBOL (『interest』>>>)

『interest』로 시작하는 유일한 기호인 『interest_rate』를 내보내기합니다.

EXPORT SYMBOL (『i』>>>『rate』>>>)

『i』로 시작하고 뒤에 『rate』가 있는 유일한 기호인 『interest_rate』를 내보내기합니다.

EXPORT SYMBOL (<<<『i』>>>『rate』)

『와일드카드 스펙의 복수 일치 오류』가 발생합니다. 『prime_rate』와 『interest_rate』에는 둘다 『i』가 있고, 뒤에 『rate』로 끝납니다.

EXPORT SYMBOL (『inter』>>>『prime』)

『와일드카드 스펙과 일치하지 않음』 오류가 발생합니다. 『inter』로 시작하고 뒤에 『prime』으로 끝나는 기호가 없습니다.

EXPORT SYMBOL(<<<)

『와일드카드 스펙의 복수 일치 오류』가 발생합니다. 이 기호는 세 기호 모두와 일치하므로 유효하지 않습니다. 내보내기 명령문은 결과적으로 하나의 기호만 내보내기합니다.

바인더 언어 예

바인더 언어를 사용하는 하나의 예로서 다음 프로시듀어가 있는 간단한 재무관리 어플리케이션을 개발하는 것을 가정하십시오.

- Rate 프로시듀어

  Loan_Amount, Term_of_Payment 및 Payment_Amount 값이 있는 경우 Interest_Rate를 계산합니다.

- Amount 프로시듀어

  Interest_Rate, Term_of_Payment 및 Payment_Amount 값이 있는 경우 Loan_Amount를 계산합니다.

- Payment 프로시듀어

  Interest_Rate, Term_of_Payment 및 Loan_Amount 값이 있는 경우 Payment_Amount를 계산합니다.

- Term 프로시듀어

  Interest_Rate, Loan_Amount 및 Payment_Amount 값이 있는 경우 Term_of_Payment를 계산합니다.

이 어플리케이션을 위한 출력의 일부는 195 페이지의 부록 A 『CRTPGM, CRTSRVPGM, UPDPGM, UPDSRVPGM 명령의 출력 리스팅』에 있습니다.

이 바인더 언어의 예에는 각 모듈에 둘 이상의 프로시듀어가 들어 있습니다. 이 예는 단 하나의 프로시듀어만 포함한 모듈에도 적용됩니다.

바인더 언어 예 1

Rate, Amount, Payment 및 Term 프로시듀어의 바인더 언어는 다음과 같습니다.

```
FILE: MYLIB/QSRVSRC  MEMBER: FINANCIAL

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL('Term')
```

```
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
                    ENDPGMEXP
```

몇 가지 초기 설계 단계의 의사 결정이 이루어졌으며, 세 개의 모듈(MONEY, RATES, CALCS)이 필요한 프로시듀어를 제공합니다.

97 페이지의 그림 34에 나오는 것과 같은 서비스 프로그램을 작성하기 위해 다음 CRTSRVPGM 명령에 바인더 언어가 지정됩니다.

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

SRCFILE 매개변수에서 지정된 라이브러리 MYLIB의 소스 파일 QSRVSRC는 바인더 언어 소스가 들어 있는 파일이라는 것을 주의하십시오.

또한 서비스 프로그램을 작성하기 위해 필요한 모든 모듈이 MODULE 매개변수에서 지정되므로, 어떠한 바인딩 디렉토리도 필요하지 않다는 것을 주의하십시오.

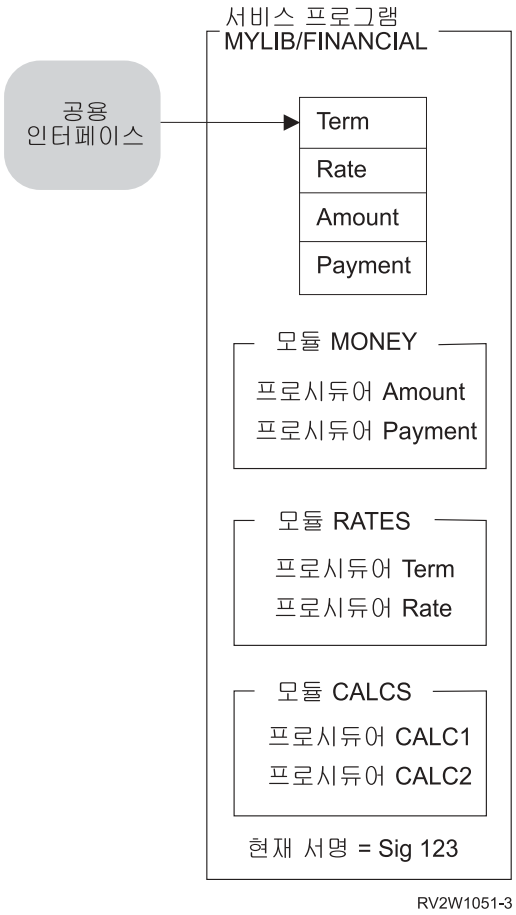

그림 34. 바인더 언어를 사용한 서비스 프로그램 작성

바인더 언어 예 2

어플리케이션의 개발이 진행됨에 따라 BANKER라는 프로그램이 작성됩니다. BANKER는 서비스 프로그램 FINANCIAL의 Payment 프로시듀어를 사용해야 합니다. 98 페이지의 그림 35에서는 BANKER 프로그램의 결과 어플리케이션을 보여줍니다.

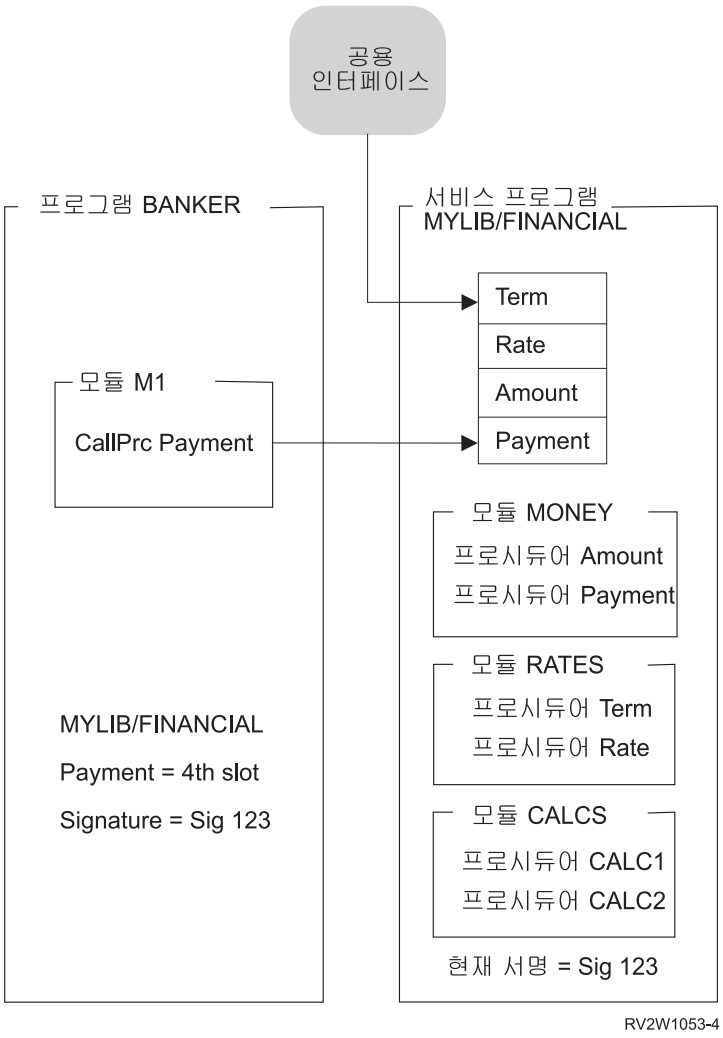

그림 35. 서비스 프로그램 FINANCIAL의 사용

BANKER 프로그램이 작성될 때 BNDSRVPGM 매개변수에 MYLIB/FINANCIAL 서비스 프로그램이 제공되었습니다. FINANCIAL 서비스 프로그램 공용 인터페이스의 네 번째 슬롯에서 기호 Payment가 내보내기되었습니다. MYLIB/FINANCIAL의 현재 서명이 Payment 인터페이스와 연관된 슬롯과 함께 BANKER 프로그램에 저장됩니다.

BANKER의 실행 준비를 하는 동안 활성화가 다음을 검증합니다.

- 라이브러리 MYLIB의 서비스 프로그램 FINANCIAL을 발견할 수 있는지
- 서비스 프로그램이 여전히 BANKER에 저장된 서명(SIG 123)을 지원하는지

이 서명 검사는 작성시 BANKER에 의해 사용된 공용 인터페이스가 실행시에도 여전히 유효한지를 검증합니다.

98 페이지의 그림 35에 나오는 것처럼 BANKER가 호출 받을 때 MYLIB/FINANCIAL이 BANKER에 의해 사용되는 공용 인터페이스를 계속해서 지원합니다. 활성화가 MYLIB/FINANCIAL이나 서비스 프로그램 MYLIB/FINANCIAL에서 일치하는 서명을 찾지 못할 경우 다음의 상황이 발생합니다.

BANKER가 활성화되지 않습니다.

오류 메세지가 발행됩니다.

바인더 언어 예 3

어플리케이션이 계속 진행됨에 따라, 재무관리 패키지를 완료하기 위해 두 개의 새로운 프로시듀어가 필요합니다. 두 개의 새로운 프로시듀어, OpenAccount 및 CloseAccount는 각각 Accounts를 열고 닫습니다. 프로그램 BANKER가 재작성될 필요 없이 MYLIB/FINANCIAL을 갱신하려면 다음 단계를 실행해야 합니다.

1. 프로시듀어 OpenAccount와 CloseAccount를 작성합니다.
2. 바인더 언어를 갱신하여 새로운 프로시듀어를 지정합니다.

갱신된 바인더 언어는 새로운 프로시듀어를 지원합니다. 또한 FINANCIAL 서비스 프로그램을 사용하는 기존의 ILE 프로그램이나 서비스 프로그램을 변경되지 않은 상태로 유지시킵니다. 바인더 언어는 다음과 같습니다.

```
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
```

다음 모두를 수행하기 위해 서비스 프로그램에 대한 갱신 조작이 필요한 경우:

- 새로운 프로시듀어나 자료 항목을 지원합니다.
- 변경된 서비스 프로그램을 사용하는 기존의 프로그램과 서비스 프로그램이 변경되지 않도록 합니다.

두 가지 대안 중 하나를 선택해야 합니다. 첫 번째 대안은 다음의 단계를 실행하는 것입니다.

1. PGMLVL(*CURRENT)이 들어 있는 STRPGMEXP, ENDPGMEXP 블록을 복사하십시오.

2. 복사된 PGMLVL(*CURRENT) 값을 PGMLVL(*PRV)로 변경하십시오.
3. PGMLVL(*CURRENT)이 들어 있는 STRPGMEXP 명령에서 리스트의 끝에 내보내기될 자료 항목이나 새로운 프로시듀어를 추가하십시오.
4. 소스 파일에 변경사항을 저장하십시오.
5. 새로운 모듈이나 변경된 모듈을 작성 또는 재작성하십시오.
6. 갱신된 바인더 언어를 사용하여, 새로운 모듈이나 변경된 모듈에서 서비스 프로그램을 작성하십시오.

두 번째 대안은 STRPGMEXP 명령의 서명 매개변수를 이용하여 내보내기 불록의 끝에 새로운 기호를 추가하는 것입니다.

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

101 페이지의 그림 36에서와 같이 향상된 서비스 프로그램을 작성하기 위해 99 페이지에 지정된 갱신 바인더 언어가 다음 CRTSRVPGM 명령에 사용됩니다.

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

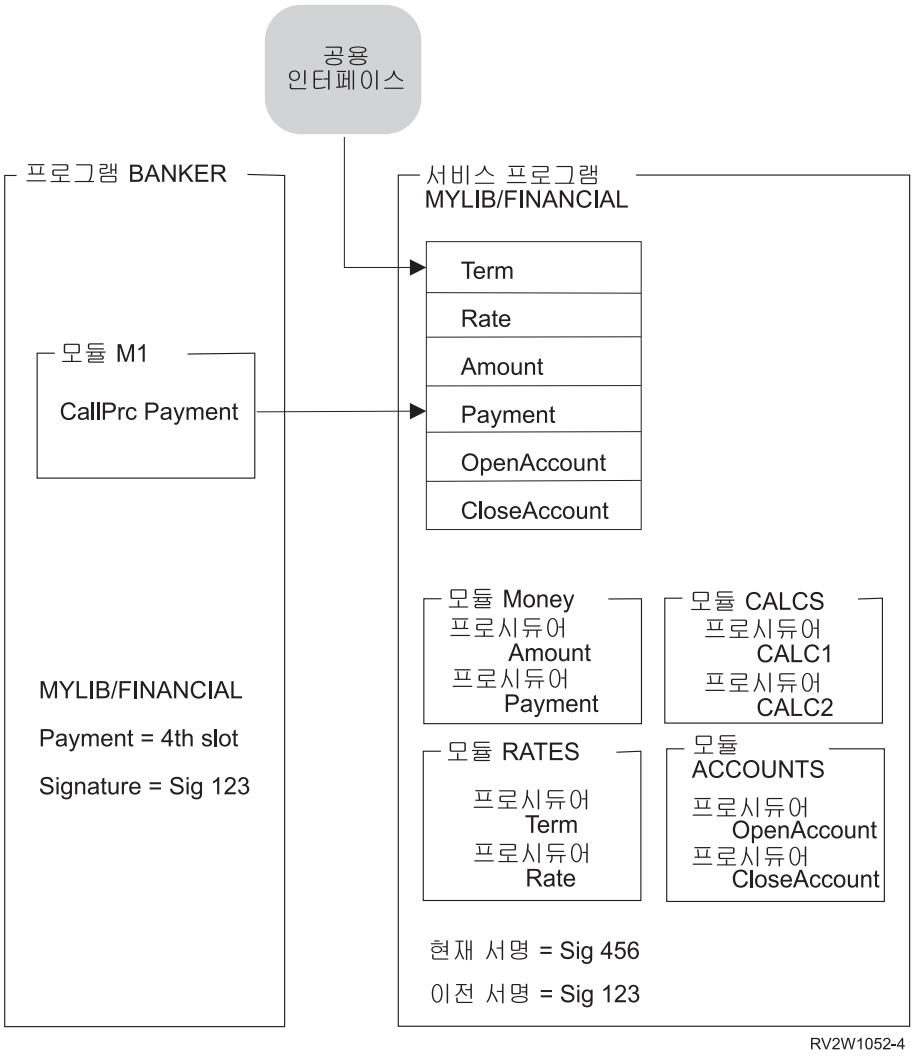

그림 36. 바인더 언어를 사용한 서비스 프로그램 갱신

이전의 서명이 여전히 지원되므로 BANKER 프로그램은 변경할 필요가 없습니다(서비스 프로그램 MYLIB/FINANCIAL의 이전 서명과 BANKER에 저장된 서명 참조). CRTPGM 명령으로 BANKER가 재작성되면 BANKER와 함께 저장되는 서명이 서비스 프로그램 FINANCIAL의 현재 서명이 됩니다. 프로그램 BANKER를 재작성해야 할 유일한 경우는 프로그램이 서비스 프로그램 FINANCIAL에 의해 제공된 새로운 프로시듀어 중 하나를 사용했을 때 뿐입니다. 바인더 언어를 사용하면 변경된 서비스 프로그램을 사용하는 프로그램이나 서비스 프로그램을 변경하지 않고 서비스 프로그램을 확장할 수 있습니다.

바인더 언어 예 4

갱신된 FINANCIAL 서비스 프로그램을 출하한 후 다음에 기초하여 이자율(interest rate)을 작성하라는 요구를 받은 것으로 가정하십시오.

Rate 프로시듀어의 현재 매개변수

신청자의 신용 내역(credit history)

5번째 매개변수 Credit_History가 호출시 Rate 프로시듀어에 추가되어야 합니다. Credit_History는 Rate 프로시듀어에서 리턴되는 Interest_Rate 매개변수를 갱신합니다. 또 하나의 요구사항은 FINANCIAL 서비스 프로그램을 사용하는 기존의 ILE 프로그램이나 서비스 프로그램이 변경되어서는 안 된다는 것입니다. 만일 언어가 일정하지 않은 수의 매개변수 전달을 지원하지 않는 경우 다음 중 어느 것도 수행하기 어렵습니다.

- 서비스 프로그램 갱신
- FINANCIAL 서비스 프로그램을 사용하는 다른 모든 오브젝트의 재작성 방지

그러나 다행히도 이를 수행할 수 있는 방법이 있습니다. 다음의 바인더 언어는 갱신된 Rate 프로시듀어를 지원합니다. 이것에 의해 FINANCIAL 서비스 프로그램을 사용하는 기존의 ILE 프로그램이나 서비스 프로그램이 변경되지 않고 그대로 유지될 수 있습니다.

```
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
```

원래 기호 Rate가 Old_Rate로 재명명되었으나 내보내기될 기호의 동일한 상대 위치에 그대로 남아 있습니다. 이것을 반드시 기억하십시오.

주석은 Old_Rate 기호와 연관되어 있습니다. /*와 */ 사이에 있는 모든 내용이 바로 주석입니다. 서비스 프로그램이 작성될 때 바인더는 바인더 언어 소스에 있는 주석을 무시합니다.

추가 매개변수 Credit_History를 지원하는 새로운 프로시듀어 Rate도 내보내기되어야 합니다. 이 갱신 프로시듀어는 내보내기 리스트의 끝에 추가됩니다.

다음의 두 가지 방법으로 원래의 Rate 프로시듀어를 처리할 수 있습니다.

- 네 개의 매개변수를 지원하는 원래의 Rate 프로시듀어를 Old_Rate로 재명명하십시오. Old_Rate 프로시듀어를 복사하십시오(이를 Rate라 함). 5번째 매개변수 Credit_History를 지원하도록 코드를 갱신하십시오.
- 5번째 매개변수 Credit_History를 지원하도록 초기 Rate 프로시듀어를 갱신하십시오. Old_Rate라는 새로운 프로시듀어를 작성하십시오. Old_Rate는 Rate의 원래 네 개의 매개변수를 지원합니다. 또한 그것은 형식적인 5번째 매개변수가 있는 새로 갱신된 Rate 프로시듀어를 호출합니다.

  이 방법은 보다 간단하게 유지보수할 수 있고 오브젝트의 크기도 더 작기 때문에 선호되는 방법입니다.

프로시듀어 Rate, Term 및 Old_Rate를 지원하는 새로운 RATES 모듈과 갱신된 바인더 언어를 사용하여 다음의 FINANCIAL 서비스 프로그램을 작성하십시오.

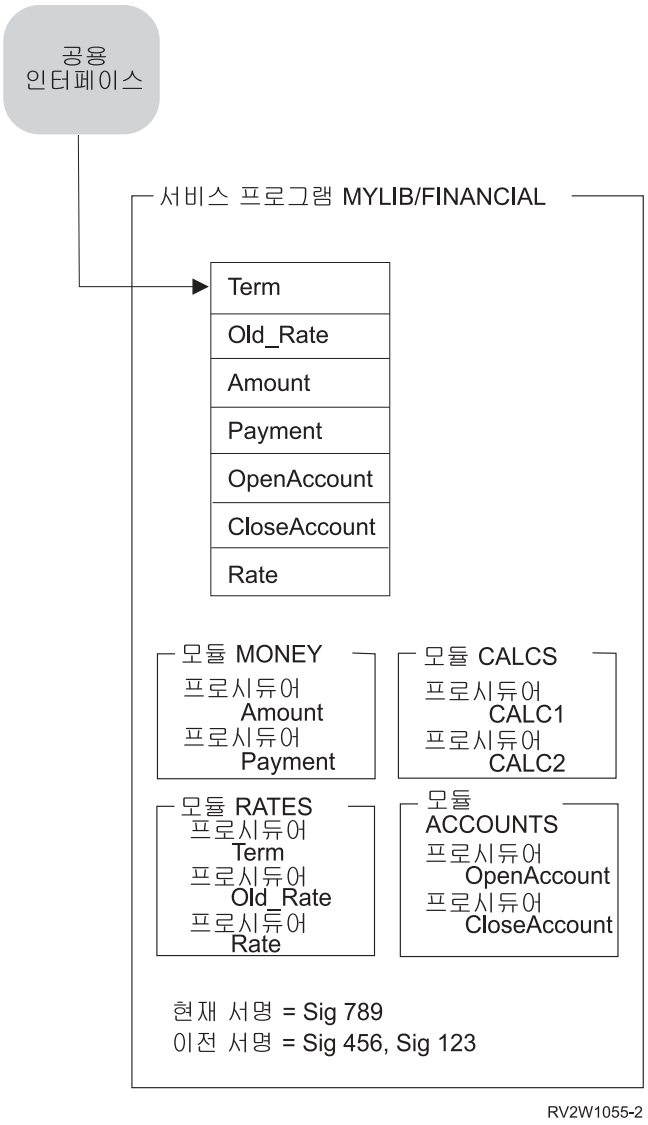

그림 37. 바인더 언어를 사용한 서비스 프로그램 갱신

FINANCIAL 서비스 프로그램의 원래 Rate 프로시듀어를 사용하는 ILE 프로그램과 서비스 프로그램이 슬롯 2로 이동합니다. 이것은 Old_Rate 프로시듀어로 호출을 보내며, Old_Rate는 원래 네 개의 매개변수를 처리하므로 편리합니다. 초기 Rate 프로시듀어를 사용한 모든 ILE 프로그램이나 서비스 프로그램이 재작성되어야 할 경우 다음 중 하나를 수행하십시오.

- 네 개의 매개변수가 있는 원래의 Rate 프로시듀어를 계속 사용하려면 Rate 프로시듀어 대신 Old_Rate 프로시듀어를 호출하십시오.
- 새로운 Rate 프로시듀어를 사용하려면 Rate 프로시듀어에 대한 각 호출마다 각각 5번째 매개변수 Credit_History를 추가하십시오.

서비스 프로그램에 대한 갱신이 다음 요구사항을 충족시켜야 할 경우:

- 처리할 수 있는 매개변수의 수를 변경시킨 프로시듀어를 지원합니다.

- 변경된 서비스 프로그램을 사용하는 기존의 프로그램과 서비스 프로그램이 변경되지 않도록 합니다.

다음 단계를 수행하십시오.

1. PGMLVL(*CURRENT)이 들어 있는 STRPGMEXP, ENDPGMEXP 블록을 복사하십시오.
2. 복사된 PGMLVL(*CURRENT) 값을 PGMLVL(*PRV)로 변경하십시오.
3. PGMLVL(*CURRENT)이 들어 있는 STRPGMEXP 명령에서 초기 프로시듀어 명을 재명명하되, 상대 위치는 그대로 유지하십시오.

   이 예에서 Rate는 Old_Rate로 변경되었으나 내보내기될 기호 리스트에서 동일한 상대 위치에 그대로 남아 있습니다.
4. PGMLVL(*CURRENT)이 있는 STRPGMEXP 명령에서 서로 다른 수의 매개변수를 지원하는 리스트의 끝에 원래 프로시듀어명을 배치하십시오.

   이 예에서 Rate는 내보내기된 기호의 리스트 끝에 추가되지만 이 Rate 프로시듀어는 추가 매개변수 Credit_History를 지원합니다.
5. 바인더 언어 소스 파일에 대한 변경사항을 저장하십시오.
6. 소스 코드가 들어 있는 파일에서 새로운 매개변수를 지원하도록 초기 프로시듀어를 확장하십시오.

   이 예에서는 5번째 매개변수 Credit_History를 지원하기 위해 기존의 Rate 프로시듀어를 변경한다는 것을 의미합니다.
7. 원래 매개변수를 입력으로 처리하고 더미(dummy) 추가 매개변수가 있는 새로운 프로시듀어를 호출하는 새로운 프로시듀어를 작성하십시오.

   이 예에서 이것은 원래 매개변수를 처리하는 Old_Rate 프로시듀어를 추가하고, 더미(dummy) 5번째 매개변수가 있는 새로운 Rate 프로시듀어를 호출하는 것을 의미합니다.
8. 바인더 언어 소스 코드 변경사항을 저장하십시오.
9. 새로운 프로시듀어 및 변경된 프로시듀어와 함께 모듈 오브젝트를 작성하십시오.
10. 갱신된 바인더 언어를 사용하여 새로운 모듈과 변경된 모듈에서 서비스 프로그램을 작성하십시오.

프로그램 변경

프로그램 변경(CHGPGM) 명령은 재컴파일 없이 프로그램의 속성을 변경합니다. 변경이 가능한 속성은 다음과 같습니다.

- 최적화 속성
- 사용자 프로파일 속성
- 허용된 권한 사용 속성
- 성능 콜렉션 속성

- 프로파일링 자료 속성
- 프로그램 텍스트
- 사용권 내부 코드 옵션
- 테라공간 속성

지정된 속성이 현재 속성과 동일한 경우에도 사용자가 프로그램 재작성을 강제할 수 있습니다. *YES 값의 프로그램 재작성 강제(FRCCRT) 매개변수를 사용하십시오.

*NO 및 *NOCRT 값으로 지정된 프로그램 재작성 강제(FRCCRT) 매개변수를 사용할 수도 있습니다. 이 값은 요구된 프로그램 속성이 해당 변경이 프로그램이 재작성되기를 요구할 때 실제로 변경되는지 여부를 판별합니다. 다음 프로그램 속성을 수정하면 프로그램이 재작성될 수 있습니다.

- 프로그램 프롬트 최적화(OPTIMIZE 매개변수)
- 허용된 권한 프롬트 사용(USEADPAUT 매개변수)
- 성능 콜렉션 프롬트 작동가능(ENBPFRCOL 매개변수)
- 프로파일링 자료 프롬트(PRFDTA 매개변수)
- 사용자 프로파일 프롬트(USRPRF 매개변수)
- 사용권 내부 코드 옵션 프롬트(LICOPT 매개변수)
- 테라공간 프롬트(TERASPACE 매개변수)

프로그램 재작성 강제(FRCCRT) 매개변수에 대한 값 *NO는 재작성이 강제 실행되지 않지만 재작성을 요구하는 프로그램 속성 중 하나가 변경되는 경우 프로그램이 재작성됨을 의미합니다. 이 옵션은 시스템이 변경이 필요한지 여부를 판별할 수 있게 합니다.

하나 이상의 작업이 프로그램을 사용 중인 동안 CHGPGM 또는 CHGSRVPGM으로 프로그램을 재작성하면『오브젝트가 파괴됨』예외가 발생하며, 이들 작업이 실패할 수 있습니다. 프로그램 재작성 강제(FRCCRT) 매개변수에 대한 명령 디폴트를 *NOCRT로 변경하여, 이 상황이 우발적으로 발생하지 않도록 막을 수 있습니다.

프로그램 갱신

ILE 프로그램 오브젝트나 서비스 프로그램이 작성된 후 내부의 오류를 정정하거나 개선을 추가할 수 있습니다. 그러나 사용자가 오브젝트를 제공한 후 오브젝트가 너무 커서 고객에게 전체 오브젝트를 제공하는 것이 어려워지거나 비용이 많이 지출될 수 있습니다.

사용자는 프로그램 갱신(UPDPGM) 명령 또는 서비스 프로그램 갱신(UPDSRVPGM) 명령을 사용하여 출하 크기를 줄일 수 있습니다. 이들 명령은 지정된 모듈만 대체하므로, 변경되거나 추가된 모듈만 고객에게 배포해야 합니다.

PTF 처리를 사용할 경우 UPDPGM 또는 UPDSRVPGM 명령에 대한 하나 이상의 호출을 포함하고 있는 나감 프로그램이 갱신 기능에 사용될 수 있습니다. 같은 모듈을 여러 프로그램 오브젝트나 서비스 프로그램으로 바인드하기 위해서는 각 *PGM 및 *SRVPGM 오브젝트에 대해 UPDPGM 또는 UPDSRVPGM 명령을 수행해야 합니다.

예를 들어 그림 38을 참조하십시오.

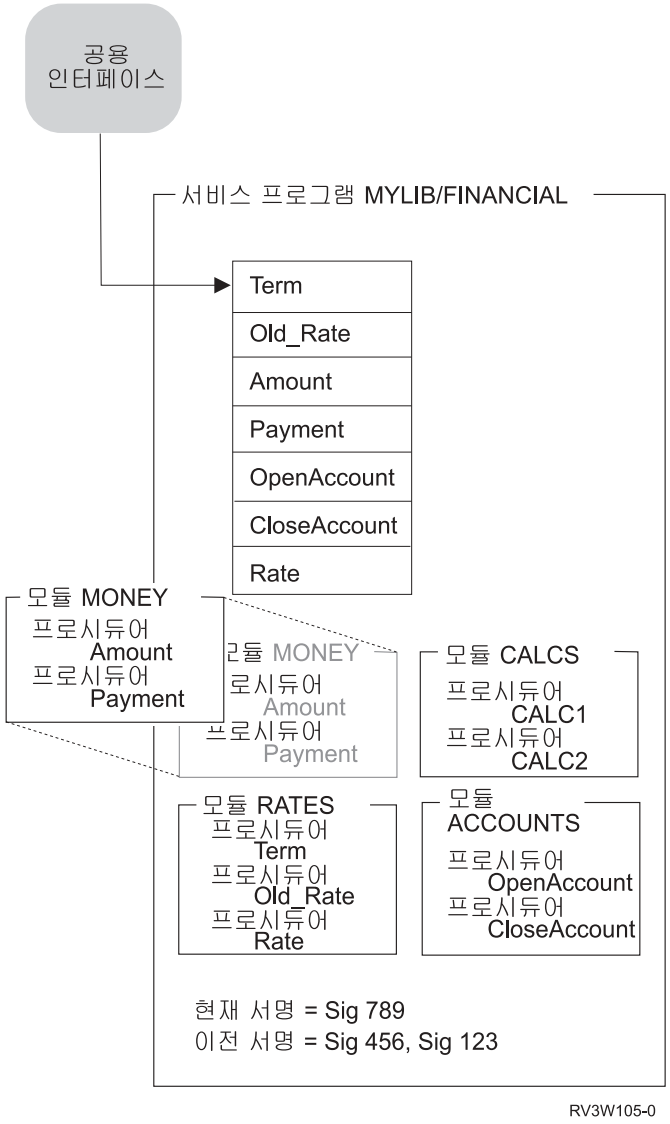

그림 38. 서비스 프로그램에서 모듈 대체

프로그램 또는 서비스 프로그램이 다른 작업에서 활성인 상태에 있는 중에 갱신되는 경우 작업은 프로그램 또는 서비스 프로그램의 이전 버전을 계속 사용합니다. 새 활성화가 프로그램 또는 서비스 프로그램의 갱신된 버전을 사용합니다.

CRTPGM 또는 CRTSRVPGM 명령의 갱신 허용(ALWUPD) 및 *SRVPGM 라이브러리 갱신 허용(ALWLIBUPD) 매개변수는 프로그램 오브젝트나 서비스 프로그램이 갱신될 수 있는지 여부를 결정합니다. ALWUPD(*NO)가 지정될 경우 프로그램 오브젝트나 서비스 프로그램의 모듈이 UPDPGM 또는 UPDSRVPGM 명령에 의해 대체될 수 없습니다. ALWUPD(*YES) 및 ALWLIBUPD(*YES)가 지정될 경우에는 이전에 지정되지 않은 라이브러리에서 서비스 프로그램을 사용하기 위해 프로그램을 갱신할 수 있습니다. ALWUPD(*YES) 및 ALWLIBUPD(*NO)가 지정될 경우에는 모듈을 갱신할 수 있으나 바인드된 서비스 프로그램 라이브러리는 갱신할 수 없습니다. 동시에 ALWUPD(*NO) 및 ALWLIBUPD(*YES)를 지정할 수는 없습니다.

UPDPGM 및 UPDSRVPGM 명령의 매개변수

모듈 매개변수에 지정된 각 모듈은 프로그램 오브젝트나 서비스 프로그램으로 바인드되는 같은 이름의 모듈을 대체합니다. 프로그램 오브젝트나 서비스 프로그램으로 바인드되는 둘 이상의 모듈에 같은 이름이 있을 경우 대체 라이브러리(RPLLIB) 매개변수가 사용됩니다. 이 매개변수는 대체될 모듈을 선택하기 위해 사용될 방법을 지정합니다. 이미 프로그램 오브젝트나 서비스 프로그램으로 바인드된 같은 이름의 모듈이 없는 경우 프로그램 오브젝트나 서비스 프로그램은 갱신되지 않습니다.

바인드된 서비스 프로그램(BNDSRVPGM) 매개변수는 프로그램 오브젝트나 서비스 프로그램이 이미 바인드된 서비스 프로그램 이외에 추가의 서비스 프로그램을 지정합니다. 대체될 모듈보다 대체하는 모듈에 더 많은 가져오기 또는 더 적은 내보내기가 있는 경우 해당 가져오기를 해결하기 위해 이들 서비스 프로그램이 필요할 수도 있습니다.

서비스 프로그램 라이브러리(SRVPGMLIB) 매개변수를 사용하면 바인드 서비스 프로그램을 저장하는 라이브러리를 지정할 수 있습니다. UPDPGM 또는 UPDSRVPGM 명령을 실행할 때마다, 지정된 라이브러리에서 바인드 서비스 프로그램이 사용됩니다. ALWLIBUPD(*YES)가 사용된 경우 UPDPGM 또는 UPDSRVPGM 명령을 이용하여 라이브러리를 변경할 수 있습니다.

바인딩 디렉토리(BNDDIR) 매개변수는 여분의 가져오기를 해결하는 데 필요할 수 있는 모듈이나 서비스 프로그램을 포함한 바인딩 디렉토리를 지정합니다.

활성 그룹(ACTGRP) 매개변수는 프로그램 또는 서비스 프로그램이 활성화되었을 때 사용될 활성 그룹명을 지정합니다. 또한 이 매개변수를 사용하여 명명된 활성화 그룹의 활성 그룹명을 변경할 수 있습니다.

더 적은 가져오기가 있는 모듈로 대체된 모듈

모듈이 보다 적은 가져오기가 있는 다른 모듈로 대체되는 경우 언제나 새로운 프로그램 오브젝트 또는 서비스 프로그램이 작성됩니다. 그러나 다음의 조건이 존재하면 갱신된 프로그램 오브젝트나 서비스 프로그램에 분리된 모듈이 포함됩니다.

- 현재 누락 가져오기 때문에, 프로그램 오브젝트나 서비스 프로그램으로 바인드된 모듈 중 하나가 더 이상 가져오기를 해결하지 않는 경우
- 해당 모듈이 원래 CRTPGM 또는 CRTSRVPGM 명령에서 사용된 바인딩 디렉토리에서 온 것일 경우

분리된 모듈이 있는 프로그램은 시간을 많이 초과하여 진행될 수도 있습니다. 더이상 가져오기를 해결하지 않는 원래 바인딩 디렉토리에서 온 모듈을 제거하기 위해 오브젝트 갱신시 OPTION(*TRIM)을 지정할 수 있습니다. 그러나 이 옵션을 사용하면 모듈에 있는 내보내기가 이후의 프로그램 갱신에 사용될 수 없습니다.

더 많은 가져오기가 있는 모듈로 대체된 모듈

모듈이 많은 가져오기가 있는 모듈로 대체될 때 다음과 같은 여분의 가져오기가 해결될 경우 프로그램 오브젝트나 서비스 프로그램이 갱신될 수 있습니다.

- 오브젝트로 바인드된 모듈의 기존 세트
- 오브젝트로 바인드된 서비스 프로그램
- 명령에 지정된 바인딩 디렉토리

  이들 바인딩 디렉토리에 있는 모듈 중 하나에 필수 내보내기가 들어 있으면 모듈은 프로그램이나 서비스 프로그램에 추가됩니다. 이들 바인딩 디렉토리 중 하나에 있는 서비스 프로그램에 필수 내보내기가 들어 있으면 프로그램이나 서비스 프로그램의 참조에 의해 서비스 프로그램이 바인드됩니다.
- 내재 바인딩 디렉토리. **내재 바인딩 디렉토리**는 모듈이 있는 프로그램을 작성하기 위해 필요한 내보내기가 들어 있는 바인딩 디렉토리입니다. 모든 ILE 컴파일러는 빌드하는 각 모듈에 내재 바인딩 디렉토리의 리스트를 작성합니다.

여분의 가져오기를 해결할 수 없는 경우 갱신 명령에 OPTION(*UNRSLVREF)이 지정되지 않는 한 갱신 조작은 실패합니다.

더 적은 내보내기가 있는 모듈로 대체된 모듈

어떤 모듈을 보다 적은 내보내기가 있는 다른 모듈로 대체하는 경우 다음 조건이면 갱신이 발생됩니다.

- 누락 내보내기가 바인딩될 필요가 없을 때
- UPDSRVPGM의 경우 누락 내보내기가 서비스 프로그램에서 내보내기되지 않을 때

서비스 프로그램이 지정된 EXPORT(*ALL)로 갱신되면 새 내보내기 리스트가 작성됩니다. 새 내보내기 리스트는 기존의 내보내기 리스트와는 다릅니다.

다음 조건에서는 갱신되지 않습니다.

- 일부 가져오기가 누락 내보내기로 인해 해결될 수 없을 때
- 이들 누락 내보내기를 명령에 지정된 바인딩 디렉토리와 여분의 서비스 프로그램에서 찾을 수 없을 때

- 바인더 언어가 기호 내보내기를 표시하지만 내보내기가 누락될 때

더 많은 내보내기가 있는 모듈로 대체된 모듈

모듈이 많은 내보내기가 있는 다른 모듈로 대체되면 모든 여분의 내보내기가 고유하게 명명되는 경우 갱신 조작이 발생합니다. EXPORT(*ALL)가 지정되는 경우에는 서비스 프로그램 내보내기가 다릅니다.

그러나 하나 이상 여분의 내보내기가 고유하게 명명되어 있지 않을 경우 중복된 이름이 문제점을 발생시킬 수 있습니다.

- OPTION(*NODUPPROC) 또는 OPTION(*NODUPVAR)이 갱신 명령에서 지정되는 경우 프로그램 오브젝트나 서비스 프로그램이 갱신되지 않습니다.
- OPTION(*DUPPROC) 또는 OPTION(*DUPVAR)이 지정되는 경우 갱신이 발생하지만 동일한 이름의 기존의 내보내기가 아니라 추가적인 내보내기가 사용될 수 있습니다.

모듈, 프로그램 및 서비스 프로그램을 작성하기 위한 추가 정보

모듈, ILE 프로그램 및 서비스 프로그램을 편리하게 작성하고 유지보수하려면 다음을 고려하십시오.

- 프로그램 또는 서비스 프로그램을 작성하려면 복사될 모듈의 명명 규칙을 따르십시오.

  공통 접두부가 있는 명명 방법을 사용하면 모듈 매개변수에 모듈을 총칭으로 지정할 때보다 쉽게 할 수 있습니다.
- 용이한 유지보수를 위해 각 모듈을 단 하나의 프로그램이나 서비스 프로그램에 포함시키십시오. 둘 이상의 프로그램이 하나의 모듈을 사용해야 하는 경우 서비스 프로그램에 모듈을 배치하십시오. 즉, 모듈을 재설계해야 하는 경우 한 곳에서만 재설계해야 합니다.
- 서명을 확인하려면 서비스 프로그램을 작성할 때마다 바인더 언어를 사용하십시오.

  바인더 언어를 사용하여, 사용 중인 프로그램이나 서비스 프로그램을 재작성하지 않고도 손쉽게 서비스 프로그램을 갱신할 수 있습니다.

  바인더 소스 검색(RTVBNDSRC) 명령은 하나 이상의 모듈 또는 서비스 프로그램에서 내보내기된 것에 기초하여 바인더 언어 소스 생성을 돕는 데 사용될 수 있습니다.

  다음 조건 중 하나가 존재할 경우:
  - 서비스 프로그램을 절대로 변경하지 않습니다.
  - 서비스 프로그램의 사용자가 서명을 변경할 때 프로그램이 변경되는 것을 무시합니다.

바인더 언어를 사용할 필요가 없습니다. 대부분의 어플리케이션에는 이러한 상황이 거의 없으므로, 모든 서비스 프로그램에 대해 바인더 언어를 사용하는 것이 좋습니다.

- CRTPGM, CRTSRVPGM 또는 UPDPGM과 같은 명령을 사용할 때 CPF5D04 메세지가 발생했으나 프로그램이나 서비스 프로그램이 작성된 경우에는 다음과 같은 두 가지의 가능성이 있습니다.

  1. OPTION(*UNRSLVREF)로 프로그램이 작성되지만 해결되지 않은 참조가 있습니다.
  2. *PUBLIC *EXCLUDE 권한을 제공받은 *BNDDIR QSYS/QUSAPIBD에서 *SRVPGM과 바인드를 시도하지만 권한이 없습니다. 오브젝트에 권한이 있는 사람을 알아보려면 DSPOBJAUT 명령을 사용하십시오. 시스템 *BNDDIR QUSAPIBD에는 시스템 API가 제공하는 *SRVPGM 이름이 있습니다. 이 API 중 일부는 보안에 민감한 것으로서 *SRVPGM이 *PUBLIC *EXCLUDE 권한으로 제공됩니다. 이 *SRVPGM은 QUSAPIBD 끝에 그룹으로 있습니다. 이 리스트에서 *PUBLIC *EXCLUDE 서비스 프로그램을 사용할 때 바인더가 먼저 다른 *PUBLIC *EXCLUDE *SRVPGM을 검사하여 CPF5D04를 생성합니다.

  CPF5D04 메세지를 받지 않으려면 다음 방법 중 하나를 사용하십시오.

  – 사용자의 프로그램이나 서비스 프로그램을 바인드시킬 *SRVPGM을 명시적으로 지정합니다. 사용자의 프로그램이나 서비스 프로그램을 바인드시킬 *SRVPGMS 리스트를 보려면 DSPPGM이나 DSPSRVPGM DETAIL(*SRVPGM)을 사용하십시오. 이 *SRVPGM을 CRTPGM이나 CRTSRVPGM BNDSRVPGM 매개변수에 지정할 수 있습니다. 또한 CRTBNDRPG, CRTRPGMOD, CRTBNDCBL, CRTPGM 또는 CRTSRVPGM BNDDIR 매개변수에 지정한 바인딩 디렉토리에 위치시키거나 RPG H 스펙에서 가져올 수 있습니다. 이러한 조치는 *BNDDIR QUSAPIBD에서 *PUBLIC *EXCLUDE *SRVPGM을 검사하기 이전에 모든 참조를 해결합니다.

  – *PUBLIC이나 개별 권한을 CPF5D04 메세지에 나오는 *SRVPGM에 부여합니다. 이것은 잠재적으로 보안에 민감한 인터페이스에 대해 사용자들에게 불필요하게 권한을 부여하는 단점이 있습니다.

  – OPTION(*UNRSLVREF)을 사용하며 프로그램에 해결되지 않은 참조가 있을 경우 모든 참조를 반드시 해결합니다.

- 사용자가 작성하는 프로그램 오브젝트나 서비스 프로그램을 다른 사용자가 사용하는 경우 작성시 OPTION(*RSLVREF)을 지정합니다. 어플리케이션을 개발할 때 미해결 가져오기가 있는 프로그램 오브젝트나 서비스 프로그램을 작성할 수도 있습니다. 그러나 작성시에 모든 가져오기가 해결되어야 합니다.

OPTION(*WARN)이 지정되면 CRTPGM 또는 CRTSRVPGM 요구가 포함된 작업 기록부에 미해결 참조사항이 나열됩니다. DETAIL 매개변수에 리스트를 지정하면 참조사항은 프로그램 리스트에 포함될 수도 있습니다. 사용자는 작업 기록부나 리스트를 보유해야 합니다.

- 새로운 어플리케이션을 설계하는 경우 하나 이상의 서비스 프로그램으로 갈 공통 프로시듀어가 식별될 수 있는지를 결정하십시오.

  새로운 어플리케이션에 대한 공통 프로시듀어를 식별하고 설계하는 것이 가장 쉽습니다. ILE를 사용하기 위해 기존 어플리케이션을 변환할 경우 서비스 프로그램에 대한 공통 프로시듀어를 결정하기는 더 어렵습니다. 그렇지만 어플리케이션에 의해 요구된 공통 프로시듀어를 식별한 후 공통 프로시듀어를 포함하는 서비스 프로그램을 작성하도록 하십시오.

- 기존 어플리케이션을 ILE로 변환하는 경우 소수의 큰 프로그램을 작성하도록 하십시오.

  최소한 변경으로, 기존 어플리케이션을 쉽게 변환하여 ILE 기능을 이용할 수 있습니다. 모듈 작성 후 이들을 몇 개의 큰 프로그램으로 조합하면 가장 손쉽게 최소의 비용으로 ILE로 변환할 수 있습니다.

  많은 수의 작은 프로그램보다는 소수의 큰 프로그램을 사용하면 기억장치 사용량을 줄일 수 있으므로 더욱 유리합니다.

- 어플리케이션이 사용하는 서비스 프로그램의 수를 제한하도록 하십시오.

  이것은 서비스 프로그램이 둘 이상의 모듈에서 작성되도록 요구할 수 있습니다. 활성화 시간이 보다 짧아지고 바인딩 처리가 더욱 빨라진다는 이점이 있습니다.

  어플리케이션이 사용해야 하는 서비스 프로그램의 수에 대한 정확한 해답은 거의 없다고 볼 수 있습니다. 프로그램이 수백 개의 서비스 프로그램을 사용하면 너무 많이 사용하는 것이고, 그렇다고 하나의 서비스 프로그램만 사용하는 것도 실용적이지 않습니다.

  예를 들어, OS/400이 제공하는 언어별 공통 실행시 루틴에는 약 10개의 서비스 프로그램이 제공됩니다. 그리고 70개가 넘는 모듈이 이들 10개의 서비스 프로그램을 작성하기 위해 사용되었습니다. 이 비율은 성능, 이해도 및 유지보수 면에서 적절히 균형을 이루고 있다고 할 수 있습니다.

제 6 장 활성 그룹 관리

이 장에서는 활성 그룹을 사용하는 어플리케이션 구성 방법의 예가 소개됩니다. 주제는 다음과 같습니다.

- 복수 어플리케이션 지원
- OPM 및 ILE 프로그램과 함께 자원 재생(RCLRSC) 명령 사용
- 활성 그룹 재요구(RCLACTGRP) 명령으로 활성 그룹 삭제
- 서비스 프로그램 및 활성 그룹

같은 작업에서 실행 중인 복수 어플리케이션

사용자 명명 활성 그룹을 사용하면 나중에 사용하기 위해 작업내에 활성 그룹을 남겨 둘 수 있습니다. 제어 경계를 지나 이루어지는 정상 리턴 조작이나 건너뜀 조작(예: ILE C에서의 longjmp())은 사용자의 활성 그룹을 삭제하지 않습니다.

이것은 사용자의 어플리케이션을 마지막 사용 상태로 남아 있도록 합니다. 정적 변수와 열린 파일은 어플리케이션으로의 호출 사이에서 변경되지 않은 채 있습니다. 이로 인해 처리 시간을 단축시킬 수 있고, 이것이 사용자가 제공하려는 기능을 수행하는 데 필수적일 수도 있습니다.

그러나 같은 작업내에서 실행 중인 여러 독립 클라이언트에서의 요구를 허용할 준비를 해야 합니다. 시스템은 ILE 서비스 프로그램에 바인드할 ILE 프로그램의 수를 제한하지 않습니다. 결과적으로, 사용자가 복수 클라이언트를 지원해야 합니다.

114 페이지의 그림 39에서는 사용자 명명 활성 그룹의 성능상의 이점을 활용하는 동안 공통 서비스 기능을 사용할 수 있도록 하는 방법을 보여줍니다.

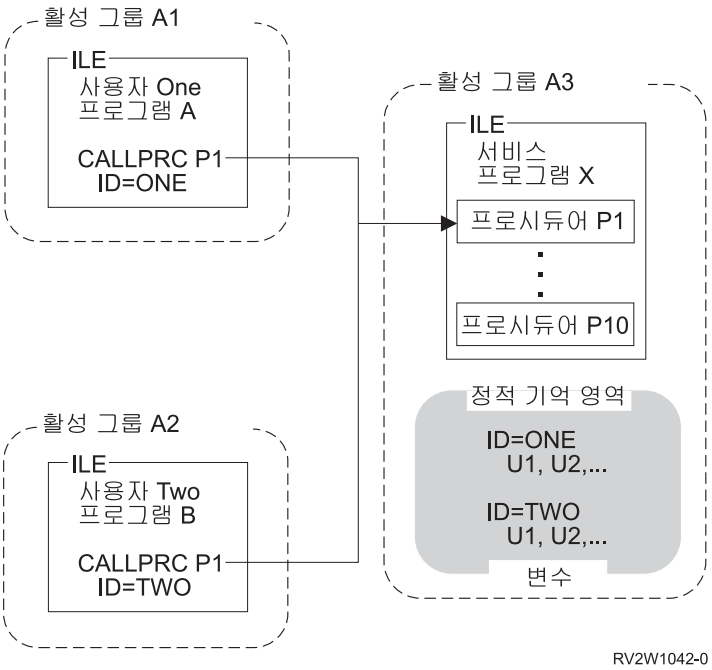

그림 39. 같은 작업에서 실행 중인 복수 어플리케이션

서비스 프로그램 X에서 프로시듀어에 대한 각 호출은 사용자 처리이 필요합니다. 필드 ID는 이 예에서 사용자 처리을 나타냅니다. 각 사용자에게는 이 처리를 제공할 책임이 있습니다. 각 사용자에 대한 고유 처리을 리턴하기 위한 초기화 루틴을 수행합니다.

서비스 프로그램 호출이 있는 경우 이 사용자와 관련이 있는 기억장치 변수를 찾기 위해 사용자 처리이 사용됩니다. 활성 그룹 작성 시간을 줄이면서, 동시에 복수 클라이언트를 지원할 수 있습니다.

자원 재생 명령

자원 재생(RCLRSC) 명령은 **레벨 번호**로 불리는 시스템 개념에 기초하고 있습니다. 레벨 번호는 작업내에서 사용자가 사용하는 특정 자원에 대해 시스템이 지정한 고유 값입니다. 세 개의 레벨 번호가 다음과 같이 정의됩니다.

호출 레벨 번호

각 호출 스택 항목에 고유 레벨 번호가 제공됩니다.

프로그램 활성화 레벨 번호

각 OPM 및 ILE 프로그램 활성화에 고유 레벨 번호가 제공됩니다.

활성 그룹 레벨 번호

각 활성 그룹에 고유 레벨 번호가 제공됩니다.

작업 실행 중, 시스템은 바로 위에서 설명된 자원이 새로 발견될 때마다 계속 고유 레벨 번호를 지정합니다. 레벨 번호는 오름차순으로 지정됩니다. 레벨 번호가 큰 자원은 번호가 작은 자원 다음에 작성됩니다.

그림 40에서는 OPM 및 ILE 프로그램에서의 RCLRSC 명령 사용의 예를 보여줍니다. 호출 레벨 범위 지정은 이 예에서 볼 수 있는 열린 파일에 대해 사용되어 왔습니다. 호출 레벨 범위 지정이 사용될 때 각 자료 관리 자원에 해당 자원을 작성한 호출 스택 항목으로 같은 레벨 번호가 제공됩니다.

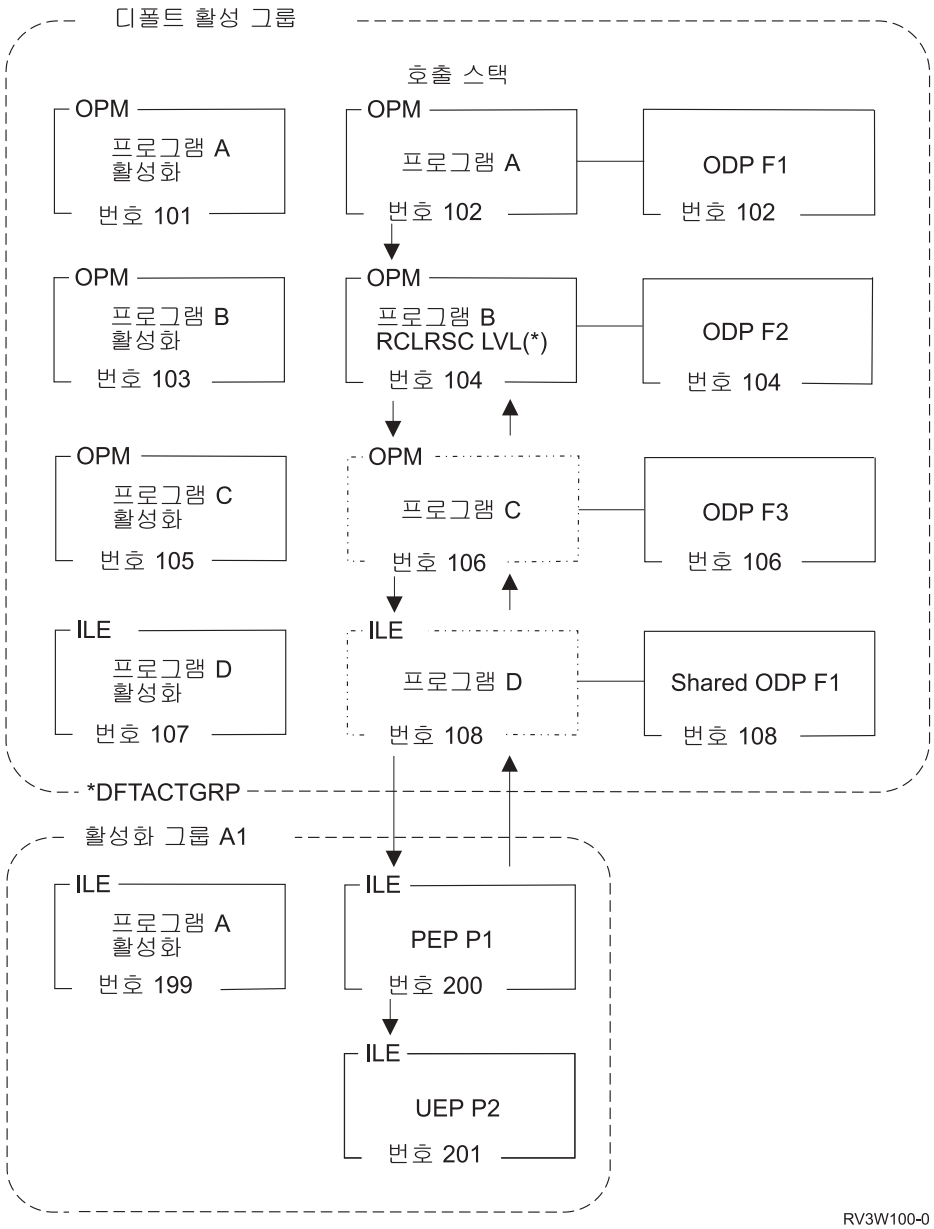

그림 40. 자원 재생

이 예에서 호출 순서는 프로그램 A, B, C, D순입니다. 프로그램 D와 C는 프로그램 B로 리턴합니다. 프로그램 B에서는 LVL(*) 옵션으로 RCLRSC 명령을 사용합니다. RCLRSC 명령은 LVL(레벨) 매개변수를 사용하여 자원을 지웁니다. 현재 호출 스택 항목의 호출 레벨 번호보다 큰 호출 레벨 번호가 있는 모든 자원이 지워집니다. 이 예에서는 호출 레벨 번호 104가 시작점으로 사용됩니다. 호출 레벨 번호 104보다 큰 모든

자원이 삭제됩니다. 호출 레벨 200과 201의 자원은 ILE 활성 그룹내에 있기 때문에 RCLRSC의 영향을 받지 않습니다. RCLRSC는 디폴트 활성 그룹내에서만 작업합니다.

또한 프로그램 C와 D의 기억장치와 파일 F3의 열린 자료 경로(ODP)가 닫혀집니다. 파일 F1은 프로그램 A에서 열린 ODP와 공유됩니다. 공유된 ODP는 닫히지만 파일 F1은 열린 상태로 유지됩니다.

OPM 프로그램을 위한 자원 재생 명령

자원 재생(RCLRSC) 명령은 종료하지 않고 리턴한 OPM 프로그램을 위해 열린 파일을 닫고 정적 기억장치를 해제하는 데 사용될 수 있습니다. 일부 OPM 언어(예: RPG)를 사용하면 프로그램을 종료하지 않고 리턴할 수 있습니다. 나중에 프로그램 파일을 닫고 기억장치를 해제하려는 경우 RCLRSC 명령을 사용할 수 있습니다.

ILE 프로그램을 위한 자원 재생 명령

DFTACTGRP(*YES)가 지정된 CRTBNDxxx 명령을 사용하여 작성된 ILE 프로그램의 경우 RCLRSC 명령은 OPM 프로그램에서와 같이 정적 기억장치를 해제합니다. DFTACTGRP(*YES)가 지정된 CRTBNDxxx 명령을 사용하여 작성되지 **않은** ILE 프로그램의 경우 RCLRSC 명령이 디폴트 활성 그룹내에서 작성된 활성화를 다시 초기화하되, 정적 기억장치를 해제하지 않습니다. 많은 양의 정적 기억장치를 사용하는 ILE 프로그램은 ILE 활성 그룹에서 활성화되어야 합니다. 활성 그룹을 삭제하면 이 기억장치는 시스템으로 리턴됩니다. RCLRSC 명령은 디폴트 활성 그룹내에서 실행 중인 서비스 프로그램이나 ILE 프로그램으로 열린 파일을 닫습니다. RCLRSC 명령은 서비스 프로그램의 정적 기억장치를 다시 초기설정하지 않으며 디폴트가 아닌 활성 그룹에 영향을 주지 않습니다.

ILE에서 직접 RCLRSC 명령을 사용하기 위해서는 QCAPCMD API 또는 ILE CL 프로시듀어를 사용할 수 있습니다. QCAPCMD API를 사용하면 CL 프로그램을 사용하지 않고 시스템 명령을 직접 호출할 수 있습니다. 115 페이지의 그림 40에서는 직접적으로 호출하는 시스템 명령이 중요하며 그 이유는 특정 ILE 프로시듀어의 호출 레벨 번호를 사용하고자 원하는 경우가 있기 때문입니다. ILE C와 같은 특정 언어의 경우 OS/400 명령을 직접 실행할 수 있는 시스템 기능을 제공합니다.

활성 그룹 재생 명령

활성 그룹 재생(RCLACTGRP) 명령은 사용되지 않는 비디폴트 활성 그룹의 삭제에 사용될 수 있습니다. 이 명령은 적당한 활성 그룹 모두를 삭제하거나 이름별로 삭제할 수 있는 활성화 그룹 옵션을 허용합니다.

서비스 프로그램 및 활성 그룹

ILE 서비스 프로그램을 작성하는 경우 *CALLER 옵션이나 ACTGRP 매개변수명의 지정 여부를 결정하십시오. 이 옵션은 서비스 프로그램이 호출자의 활성 그룹으로 활성화되는지 아니면 개별적으로 명명된 활성 그룹으로 활성화되는지를 결정합니다. 이 두 가지 선택에는 각각 장단점이 있습니다. 이 주제에서는 각 옵션이 제공하는 것에 대해 설명합니다.

ACTGRP(*CALLER) 옵션의 경우 서비스 프로그램 기능은 다음과 같습니다.

- 정적 프로시듀어 호출의 신속성

  정적 프로시듀어의 서비스 프로그램에 대한 호출은 동일 활성 그룹 내에서 실행 중일 때 최적화됩니다.

- 공유 외부 자료

  서비스 프로그램은 같은 활성 그룹내의 다른 프로그램 및 서비스 프로그램에서 사용될 자료를 내보내기할 수 있습니다.

- 공유 자료 관리 자원

  열린 파일 및 다른 자료 관리 자원은 활성 그룹내에서 서비스 프로그램과 다른 프로그램간에 공유될 수도 있습니다. 서비스 프로그램은 확약 조작이나 롤백 조작을 발행하여 활성 그룹내의 다른 프로그램에 영향을 미칠 수 있습니다.

- 제어 경계 없음

  서비스 프로그램내의 미처리 예외는 클라이언트 프로그램에 여과됩니다. 서비스 프로그램내에서 사용된 HLL 종료 동사는 클라이언트 프로그램의 활성 그룹을 삭제할 수 있습니다.

ACTGRP(이름) 옵션의 경우 서비스 프로그램은 다음과 같은 기능을 합니다.

- 변수의 분리 주소 공간

  클라이언트 프로그램은 작업 기억장치의 주소를 지정하기 위해 포인터를 조작할 수 없습니다. 서비스 프로그램이 허용된 권한으로 실행 중일 경우 중요합니다.

- 분리 자료 관리 자원

  사용자에게는 사용자의 열린 파일과 확약 정의가 있습니다. 열린 파일이 실수로 공유되는 것을 막습니다.

- 상태 정보 제어

  어플리케이션 기억장치가 삭제되는 시기를 제어합니다. HLL 종료 동사나 정상 언어 리턴 명령문을 사용하여 어플리케이션의 삭제 시기를 결정할 수 있습니다. 그러나 복수 클라이언트에 대한 상태 정보를 관리해야 합니다.

제 7 장 프로시듀어 및 프로그램 호출

ILE 호출 스택과 인수 전달 방법은 언어간 통신을 원활하게 하여 혼합 언어 어플리케이션을 보다 쉽게 작성할 수 있게 해줍니다. 여기에서는 25 페이지의 『프로그램 및 프로시듀어 호출』에서 소개한 동적 프로그램 호출과 정적 프로시듀어 호출의 서로 다른 예에 관해 논의합니다. 호출의 세 번째 유형으로 프로시듀어 포인터 호출이 소개됩니다.

또한 OPM 및 ILE 어플리케이션 프로그래밍 인터페이스(API)에 대한 기본 프로그램 모델(OPM)의 지원에 대해서도 설명합니다.

호출 스택

호출 스택은 **호출 스택 항목**의 후입선출(LIFO) 리스트입니다. 각 호출 스택 항목에는 프로시듀어나 프로그램을 위한 자동 변수와 조건 핸들러 및 취소 핸들러 등 호출 스택 항목에 이르는 기타 자원에 관한 정보가 들어 있습니다.

작업당 한 개의 호출 스택이 있습니다. 호출은 호출된 프로시듀어 또는 프로그램의 호출 스택에 새로운 항목을 추가하고 호출된 오브젝트에 제어를 전달합니다. 리턴은 스택 항목을 제거하고 다시 이전 스택 항목의 호출 프로시듀어 또는 프로그램으로 제어를 전달합니다.

호출 스택 예

120 페이지의 그림 41에는 OPM 프로그램(프로그램 A)과 ILE 프로그램(프로그램 B)의 두 프로그램과 함께 호출 스택의 세그먼트가 포함되어 있습니다. 프로그램 B에는 세 개의 프로시듀어, 즉 프로그램 입력 프로시듀어, 사용자 입력 프로시듀어 그리고 또다른 프로시듀어(P1)가 포함되어 있습니다. 프로그램 입력 프로시듀어(PEP)와 사용자 입력 프로시듀어(UEP)의 개념은 14 페이지의 『모듈 오브젝트』에 정의되어 있습니다. 호출 흐름은 다음과 같은 단계로 이루어집니다.

1. 프로그램 A에 대한 동적 프로그램을 호출합니다.
2. 프로그램 A가 프로그램 B를 호출하여 PEP로 제어를 전달합니다. 프로그램 B에 대한 이 호출은 동적 프로그램 호출입니다.
3. PEP가 UEP를 호출합니다. 이는 정적 프로시듀어 호출입니다.
4. UEP가 프로시듀어 P1을 호출합니다. 이는 정적 프로시듀어 호출입니다.

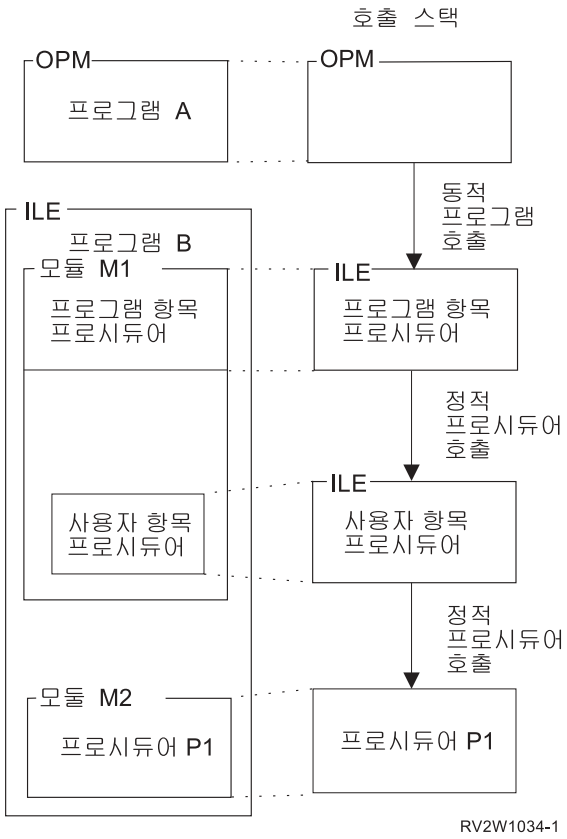

그림 41. 호출 스택에서의 동적 프로그램 호출 및 정적 프로시듀어 호출

그림 41에서는 이 예에 대한 호출 스택을 그림으로 설명하고 있습니다. 스택에서 가장 최근에 호출된 항목이 스택의 맨 아래에 표시되어 있습니다. 이것이 바로 현재 처리 중인 항목입니다. 현재 호출 스택 항목은 다음 중 하나를 수행할 수 있습니다.

- 다른 프로시듀어나 프로그램을 호출하여 스택의 맨 아래에 다른 항목을 추가합니다.
- 처리 완료 후 호출자에게 제어를 리턴시켜 스택에서 제거합니다.

프로시듀어 P1이 완료된 후 프로그램 B에서 더 이상의 처리가 필요하지 않은 것으로 가정하십시오. 프로시듀어 P1이 UEP에 제어를 리턴시키고 P1이 스택에서 제거됩니다. 그런 다음, UEP가 PEP로 제어를 리턴시키고 UEP가 스택에서 제거됩니다. 마지막으로 PEP가 프로그램 A로 제어를 리턴시키고 PEP가 스택에서 제거됩니다. 결국, 프로그램 A만 이 호출 스택의 세그먼트에 남게 됩니다. 프로그램 A는 프로그램 B에 대한 동적 프로그램 호출을 수행한 지점에서 처리를 계속합니다.

프로그램 및 프로시듀어 호출

ILE 실행시 세 가지 유형의 호출(동적 프로그램 호출, 정적 프로시듀어 호출, 프로시듀어 포인터 호출)이 수행될 수 있습니다.

ILE 프로그램이 활성화될 때 PEP를 제외한 모든 프로시듀어에 대해 정적 프로시듀어 호출과 프로시듀어 포인터 호출을 할 수 있습니다. 프로그램이 동적 프로그램 호출에 의

해 호출되고 활성화가 이미 이루어져 있지 않으면 프로그램 활성화가 발생합니다. 프로그램이 활성화될 때 이 프로그램으로 바인드된 모든 서비스 프로그램도 활성화됩니다. ILE 서비스 프로그램의 프로시듀어는 정적 프로시듀어 호출 또는 프로시듀어 포인터 호출(동적 프로그램 호출이 아니라)에 의해서만 액세스될 수 있습니다.

정적 프로시듀어 호출

ILE 프로시듀어 호출은 새로운 호출 스택 항목을 스택의 맨 아래에 추가하고 지정된 프로시듀어로 제어를 전달합니다. 다음의 모든 사항이 예에 해당됩니다.

1. 동일 모듈내의 프로시듀어로 호출
2. 동일 ILE 프로그램이나 서비스 프로그램내의 다른 모듈에 있는 프로시듀어로 호출
3. 동일 활성 그룹내의 ILE 서비스 프로그램에서 내보내기된 프로시듀어로 호출
4. 다른 활성 그룹내의 ILE 서비스 프로그램에서 내보내기된 프로시듀어로 호출

1, 2, 3에 있는 예에서 정적 프로시듀어 호출은 활성 그룹 범위를 넘지 않습니다. 성능에 영향을 미치는 호출 경로 길이는 동일합니다. 이 호출 경로는 ILE 또는 OPM 프로그램에 대한 동적 프로그램 호출의 경로보다 훨씬 짧습니다. 4 예에서는 호출이 활성 그룹 경계를 지나서 활성 그룹 자원 교환을 위해 추가 처리가 이루어집니다. 호출 경로 길이는 활성 그룹내에서만 이루어지는 정적 프로시듀어 호출의 경로 길이보다는 길지만 여전히 동적 프로그램 호출보다는 짧습니다.

정적 프로시듀어 호출의 경우 호출되는 프로시듀어는 반드시 바인드하는 동안 호출하는 프로시듀어로 바인드되어야 합니다. 호출은 항상 동일 프로시듀어를 액세스합니다. 이는 포인터를 통한 프로시듀어 호출과 대조되며, 여기서 각 호출에 따라 호출의 목표가 달라질 수 있습니다.

프로시듀어 포인터 호출

프로시듀어 포인터 호출에서는 프로시듀어를 동적으로 호출하기 위한 방법을 제공합니다. 예를 들면, 프로시듀어명이나 주소의 배열 또는 표를 조작하여, 다른 프로시듀어에 대한 프로시듀어 호출을 동적으로 라우트(route)할 수 있습니다.

프로시듀어 포인터 호출은 정적 프로시듀어 호출과 같은 방법으로 호출 스택에 항목을 추가합니다. 정적 프로시듀어 호출을 사용하여 호출될 수 있는 프로시듀어는 프로시듀어 포인터를 통해서도 역시 호출될 수 있습니다. 호출된 프로시듀어가 동일한 활성 그룹내에 있는 경우 프로시듀어 포인터 호출의 비용은 정적 프로시듀어 호출의 비용과 거의 동일합니다. 뿐만 아니라, 프로시듀어 포인터 호출은 활성화된 ILE 프로그램의 프로시듀어에도 액세스할 수 있습니다.

ILE 프로시듀어로 인수 전달

ILE 프로시듀어 호출에서 **인수**는 호출 프로시듀어가 호출에 지정된 프로시듀어로 전달하는 값을 나타내는 표현식입니다. ILE 언어는 다음의 세 가지 인수 전달 방법을 사용합니다.

값으로 직접
: 자료 오브젝트 값을 직접 인수 리스트에 위치시킵니다.

값으로 간접
: 자료 오브젝트 값이 임시 위치에 복사됩니다. 사본의 주소(포인터)를 인수 리스트에 위치시킵니다.

참조에 의해
: 자료 오브젝트의 포인터를 인수 리스트에 위치시킵니다. 호출된 프로시듀어에 의한 인수 변경사항이 호출하는 프로시듀어에 반영됩니다.

그림 42에서는 형식을 전달하는 인수를 설명합니다. 모든 ILE 언어가 값으로 직접 전달하는 형식을 지원하는 것은 아닙니다. 사용가능한 전달 형식은 ILE HLL 프로그래머 안내서에 설명되어 있습니다.

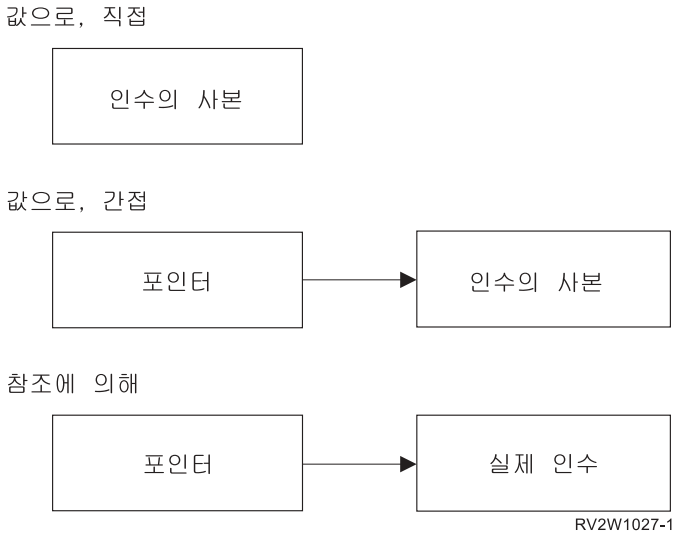

그림 42. ILE 프로시듀어로 인수를 전달하는 방법

HLL 의미는 일반적으로 자료가 값으로 전달될 시기와 참조에 의해 전달될 시기를 결정합니다. 예를 들어, ILE C에서는 값에 의해 직접 인수를 전달하고 허용하는 반면, ILE COBOL과 ILE RPG에서는 보통 참조에 의해 인수가 전달됩니다. 사용자는 호출하는 프로그램이나 프로시듀어가 호출된 프로시듀어에 의해 예측되는 방식으로 인수를 전달하는지를 확인해야 합니다. ILE HLL 프로그래머 안내서에 다른 언어로 인수를 전달하는 데 관한 자세한 정보가 나옵니다.

정적 프로시듀어 호출에서는 최대 400개의 인수가 허용됩니다. 각 ILE 언어는 인수의 최대수를 더 제한할 수도 있습니다. ILE 언어는 다음의 인수 전달 형식을 지원합니다.

- ILE C 및 C++는 기본적으로 정수와 부동 소수점 값을 확장하여 값을 사용하여 직접 인수를 전달하고 허용합니다. 인수는 #pragma 인수(이름, 확장되지 않음)를 지정하여 확장되지 않고 전달될 수 있습니다. 또한 인수는 호출된 기능에 대해 #pragma 인수를 지정함으로써 값을 사용하여 간접적으로 전달되기도 합니다.
- ILE COBOL은 값, 참조 또는 간접적인 값으로 인수를 전달하고 허용합니다.
- ILE RPG에서는 값 또는 참조로 인수를 전달하고 허용합니다. RPG는 디폴트로 정수 및 부동 소수점 값을 확장하지 않지만, 값에 의해, 코딩 EXTPROC(*CWIDEN)에 의해 전달되는 매개변수에 사용할 수 있습니다.
- ILE CL은 참조 및 값으로 인수를 전달하고 허용합니다.

기능 결과

기능의 정의(결과 인수를 리턴시키는 프로시듀어)를 허용하는 HLL을 지원하기 위해 모델은 그림 43에서처럼 특수 기능 결과 인수가 존재하는 것으로 가정합니다. ILE HLL 프로그래머 안내서의 설명에서처럼 기능 결과를 지원하는 ILE 언어는 기능 결과를 리턴시키기 위해 일반적인 형식을 사용합니다.

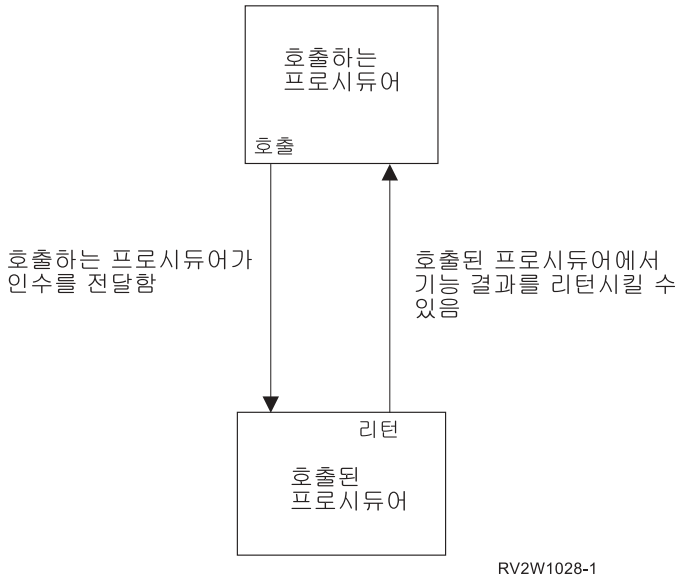

그림 43. 프로그램 호출 인수 용어

생략된 인수

모든 ILE 언어는 생략된 인수를 시뮬레이트할 수 있으므로, ILE 조건 핸들러 및 기타 실행시 프로시듀어에 대한 피드백 코드 매커니즘을 사용할 수 있습니다. 예를 들어, ILE C 프로시듀어 또는 ILE 바인드 가능 API가 참조를 사용하여 인수를 전달할 것으로 예상되면 그 위치로 널(null) 포인터를 전달하여 인수를 생략할 수도 있습니다. 특정 ILE

언어에서 생략된 인수를 지정하는 방법에 관한 정보는 해당 언어에 대한 프로그래머 안내서를 참조하십시오. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절은 각 API에 대해 어떤 인수를 생략할 수 있는지 지정합니다.

ILE 언어의 경우 호출된 프로시듀어에 인수가 생략되었는지 테스트하는 고유의 방법을 제공하지 않는 반면, 생략된 인수를 위한 테스트(CEETSTA) 바인드 가능 API를 사용할 수 있습니다.

동적 프로그램 호출

동적 프로그램 호출은 프로그램 오브젝트에 수행되는 호출입니다. 예를 들어, CL 명령 CALL을 사용하는 경우 이는 동적 프로그램 호출에 해당됩니다.

OPM 프로그램은 동적 프로그램 호출을 사용하여 호출됩니다. 뿐만 아니라, OPM 프로그램은 동적 프로그램 호출만 수행할 수 있도록 제한됩니다.

ILE 프로그램 또한 동적 프로그램 호출에 의해 호출됩니다. 활성화된 ILE 프로그램내의 프로시듀어는 정적 프로시듀어 호출이나 프로시듀어 포인터 호출을 사용하여 액세스될 수 있습니다. 아직 활성화하지 않은 ILE 프로그램은 반드시 동적 프로그램으로 호출해야 합니다.

컴파일시 바인드되는 정적 프로시듀어 호출과는 대조적으로, 동적 프로그램 호출의 기호는 호출이 실행될 때 주소로 결정됩니다. 결과적으로, 동적 프로그램 호출은 정적 프로시듀어 호출보다 더 많은 시스템 자원을 사용합니다. 동적 프로그램 호출의 예는 다음과 같습니다.

- ILE 프로그램 또는 OPM 프로그램에 대한 호출
- 바인드 불가 API에 대한 호출

ILE 프로그램에 대한 동적 프로그램 호출은 식별된 프로그램의 PEP로 제어를 전달한 다음, 다시 프로그램의 UEP로 제어를 전달합니다. 호출된 프로그램이 처리된 후 제어는 호출 프로그램 명령어 다음의 명령어로 다시 전달됩니다.

동적 프로그램 호출에서 인수 전달

ILE 또는 OPM 프로그램에 대한 호출은 대부분(ILE 프로시듀어 호출과는 대조적으로) 참조에 의해 인수를 전달합니다. 이는 호출된 프로그램이 인수의 주소를 수신하는 것을 의미합니다.

동적 프로그램 호출을 사용할 때 사용자는 호출된 프로그램에 의해 예상되는 인수 전달 방법 및 필요한 경우 인수의 시뮬레이트 방법까지 알아둘 필요가 있습니다. 동적 프로그램 호출에는 최대 255개까지의 인수가 허용됩니다. 각 ILE 언어에서는 인수의 최대수를 더 제한할 수도 있습니다. 일부 ILE 언어는 내장 함수 CALLPGMV를 지원하는데, 이것은 최대 16383개 인수를 허용합니다. 상이한 전달 메소드를 사용하는 방법에 대한 정보는 ILE HLL 프로그래머 안내서를 참조하십시오.

언어간 자료 호환성

ILE 호출을 사용하여 서로 다른 HLL로 작성된 프로시듀어 사이에서 인수를 전달할 수 있습니다. HLL간의 자료 공유를 돕기 위해 일부 ILE 언어에 자료 유형이 추가되었습니다. 예를 들어, ILE COBOL에는 새로운 자료 유형으로서 USAGE PROCEDURE-POINTER가 추가되었습니다.

HLL 사이에 인수를 전달하려면 각 HLL이 수신할 것으로 예상하는 인수의 형식을 알아야 합니다. 인수가 호출된 프로시듀어에 의해 예상된 *크기*와 유형인지를 확인하기 위해서는 호출 프로시듀어가 필요합니다. 예를 들어, 짧은 정수(2바이트)가 매개변수 리스트에 선언되었더라도 ILE C 함수가 4바이트 정수를 기대할 수 있습니다. 인수를 전달하기 위해 자료 유형 요구사항을 일치시키는 방법에 관한 정보는 ILE HLL 프로그래머 안내서를 참조하십시오.

혼합 언어 어플리케이션에서의 인수 전달을 위한 구문

일부 ILE 언어는 다른 ILE 언어로 된 프로시듀어에 인수를 전달하기 위한 구문을 제공합니다. 예를 들어, ILE C는 간접적으로 값을 사용하여 다른 ILE 프로시듀어에 값 인수를 전달하는 #pragma 인수를 제공합니다. RPG는 EXTPROC 프로토타입 키워드에 대한 특수 값을 가집니다.

조작 설명자

다른 HLL로 작성된 프로시듀어에서 인수를 수신할 수 있는 API나 프로시듀어를 작성하는 경우 조작 설명자가 유용하게 사용될 수 있습니다. **조작 설명자**는 호출된 프로시듀어가 인수의 양식(예: 다른 유형의 스트링)을 정확하게 예상할 수 없는 경우에 호출된 프로시듀어로 설명 정보를 제공합니다. 프로시듀어는 추가 정보를 사용하여 인수가 올바로 해석되도록 합니다.

인수가 값을 제공하고 조작 설명자가 인수의 *크기*와 유형에 관한 정보를 제공합니다. 예를 들면, 이 정보에는 문자 스트링의 길이와 스트링 유형이 포함될 수 있습니다.

조작 설명자를 사용하면 각 HLL이 서로 다른 바인딩을 갖도록 하는 데 바인드 가능 API와 같은 서비스가 요구되지 않으며 HLL은 호환불가능 자료 유형을 모방할 필요가 없습니다. 일부 ILE 바인드 가능 API는 HLL간의 공통 스트링 자료 유형의 부족을 조절하기 위해 조작 설명자를 사용합니다. 조작 설명자는 API 사용자에게 투명하게 존재합니다.

조작 설명자는 사용하지 않거나 예상했던 프로시듀어에서 볼 수 없는 동안에도 HLL 의미(semantics)를 지원합니다. 각 ILE 언어는 언어에 적합한 자료 유형을 사용할 수 있습니다. 각 ILE 언어 컴파일러는 조작 설명자 생성에 대한 방법을 최소한 하나는 제공합니다. 조작 설명자의 HLL 의미에 관한 자세한 정보는 ILE HLL 참조서를 참조하십시오.

조작 설명자는 사용자에게 익숙한 다른 자료 설명자와 구별됩니다. 예를 들면, 조작 설명자는 분산 자료나 파일에 연관된 설명자와는 관련이 없습니다.

조작 설명자의 요구사항

조작 설명자가 다른 ILE 언어로 작성되어 호출되는 프로시듀어에 의해 예상되는 경우와 ILE 바인드 가능 API에 의해 예상되는 경우 조작 설명자를 사용해야 합니다. 일반적으로, 바인드 가능 API에는 대부분의 스트링 인수에 대해 설명자가 필요합니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절에 나오는 바인드 가능 API 관련 정보는 주어진 바인드 가능 API가 조작 가능한 설명자를 요구하는지 여부를 지정합니다.

필요한 설명자의 부재

필요한 설명자가 누락되면 오류가 발생합니다. 프로시듀어에 특정 매개변수에 대한 설명자가 필요한 경우 이 요구사항으로 인해 해당 프로시듀어에 대한 인터페이스의 부분이 형성됩니다. 필요한 설명자가 제공되지 않으면 실행시 실패하게 됩니다.

불필요한 설명자의 존재

불필요한 설명자의 존재가 인수에 대해 호출된 프로시듀어의 액세스를 방해하지는 않습니다. 조작 설명자가 필요하지 않거나 예상되지 않은 경우 호출된 프로시듀어는 조작 설명자를 간단히 무시합니다.

주: 설명자가 필요하지 않은데도 생성된 경우 언어간 통신에 장애가 될 수 있습니다. 설명자가 호출 경로의 길이를 증가시켜 성능을 저하시킬 수 있습니다.

조작 설명자 액세스를 위한 바인드 가능 API

설명자는 일반적으로 프로시듀어가 작성된 HLL의 의미(semantics)에 따라 호출된 프로시듀어에 의해 직접 액세스됩니다. 일단 프로시듀어가 조작 설명자를 기대하도록 프로그래밍되면 일반적으로 프로그래머에 의한 처리는 더 이상 필요하지 않습니다. 그러나 호출된 프로시듀어는 때때로 액세스에 앞서 호출된 프로시듀에에 필요한 설명자가 존재하는지 여부를 판별해야 합니다. 이러한 이유로 다음의 바인드 가능 API가 제공됩니다.

- 조작 설명자 정보 검색(CEEDOD) 바인드 가능 API
- 스트링 정보 확보(CEESGI) 바인드 가능 API

OPM 및 ILE API를 위한 지원

ILE로 새로운 기능을 개발하거나 기존의 어플리케이션을 ILE로 변환하는 경우 사용자는 OPM의 호출 레벨 API를 계속 지원하고자 할 수 있습니다. 이 주제에서는 ILE로 사용자의 어플리케이션을 유지보수하면서, 이러한 이중 지원을 실현하기 위해 사용될 수 있는 한 가지 기법을 설명합니다.

ILE 서비스 프로그램은 사용자가 모든 ILE 언어에서 액세스할 수 있는 바인드 가능 API를 개발하고 전달하는 방법을 제공합니다. OPM 프로그램에 동일한 기능을 제공하려면 OPM 프로그램에서 ILE 서비스 프로그램을 직접 호출할 수 없다는 사실을 고려해야 합니다.

사용할 기법은 사용자가 지원하려는 각각의 바인드 가능 API에 대한 ILE 프로그램 스터브(stub)를 개발하는 것입니다. 사용자는 바인드 가능 API를 ILE 프로그램 스터브와 동일하게 명명하거나 다른 이름을 선택할 수 있습니다. 각 ILE 프로그램 스터브(stub)에는 실제 바인드 가능 API에 대한 정적 프로시듀어 호출이 포함됩니다.

그림 44에서는 이 기법의 예를 보여줍니다.

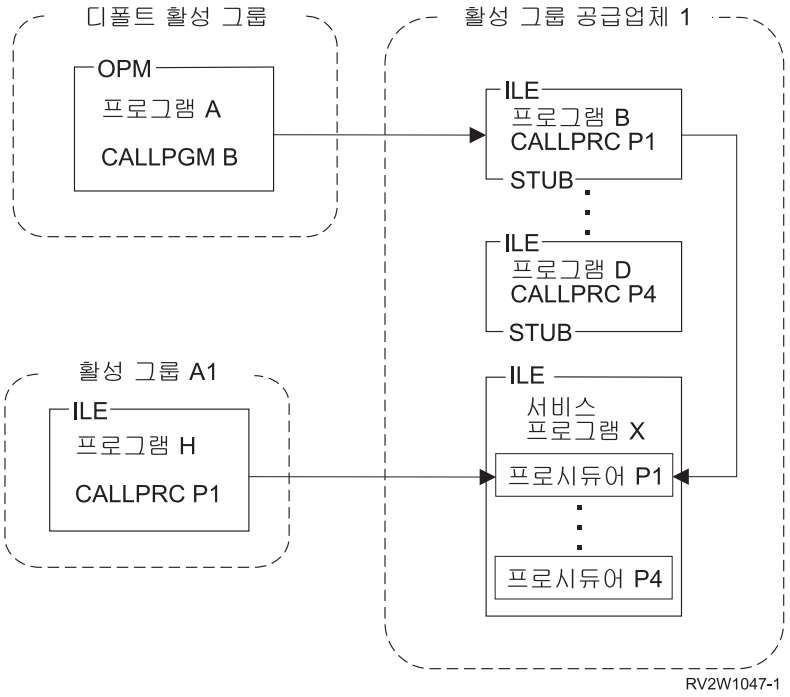

그림 44. OPM 및 ILE API 지원

프로그램 B-D는 ILE 프로그램 스터브(stub)입니다. 서비스 프로그램 X에는 각 바인드 가능 API의 실제 수행이 포함됩니다. 각 프로그램 스터브(stub)와 서비스 프로그램에는 동일한 활성 그룹명이 부여됩니다. 이 예에서는 활성 그룹명 VENDOR1이 선택됩니다.

활성 그룹 VENDOR1은 필요한 경우 시스템에 의해 작성됩니다. OPM 프로그램 A에서의 동적 프로그램 호출은 OPM 프로그램의 첫 번째 호출에서 활성 그룹을 작성합니다. ILE 프로그램 H에서의 정적 프로시듀어 호출은 ILE 프로그램 H가 활성화될 때 활성 그룹을 작성합니다. 일단 활성 그룹이 존재하면 프로그램 A나 프로그램 H 중 하나에서 사용될 수 있습니다.

사용자는 ILE 프로시듀어(이 예에서는 프로시듀어 P1)의 API 수행을 기록해야 합니다. 이 프로시듀어는 프로시듀어 호출을 통해 직접 또는 동적 프로그램 호출을 통해 간접적

으로 호출될 수 있습니다. 특정 호출 스택 구조에 의존하는 예외 메세지 송신과 같은 기능은 수행하면 안됩니다. 프로그램 스터브(stub)나 수행 프로시듀어에서의 정상 리턴은 다음 번 사용을 위해 활성 그룹을 작업내에 남겨둡니다. 각 호출에서 프로그램 스터브나 수행 프로시듀어에 제어 경계가 설정된 것을 알면, API 프로시듀어를 수행할 수 있습니다. HLL 종료 동사는 호출이 OPM 프로그램에서 시행된 것인지 또는 ILE 프로그램에서 시행된 것인지에 따라 활성 그룹을 삭제합니다.

제 8 장 기억장치 관리

오퍼레이팅 시스템은 ILE 고급 언어를 위한 기억장치를 지원합니다. 이 기억장치 지원으로 각 언어의 실행시 환경에 대한 고유 기억장치 관리 프로그램이 필요 없습니다. 이 지원은 고급 언어로 된 서로 다른 기억장치 관리 프로그램 및 메커니즘간의 호환을 가능하게 합니다.

오퍼레이팅 시스템은 실행시 프로그램 및 프로시듀어에 의해 사용되는 동적, 정적 및 자동 기억장치를 제공합니다. 자동 및 정적 기억장치는 오퍼레이팅 시스템에 의해 관리됩니다. 즉, 자동 및 정적 기억장치에 필요한 사항은 컴파일시 프로그램 변수 선언에서 알 수 있습니다. 동적 기억장치는 프로그램이나 프로시듀어에 의해 관리됩니다. 동적 기억장치에 대한 필요사항은 실행시에만 알 수 있습니다.

프로그램 활성화가 이루어지면, 프로그램 변수에 대한 정적 기억장치가 할당되고 초기 설정됩니다.

프로그램이나 프로시듀어가 실행되기 시작할 때 자동 기억장치가 할당됩니다. 자동 기억장치 스택은 호출 스택에 프로그램 또는 프로시듀어가 추가될 때 변수 사용을 위해 확장됩니다.

프로그램이나 프로시듀어가 실행될 때 동적 기억장치가 프로그램 제어하에서 할당됩니다. 이 기억장치는 추가 기억장치가 필요할 때 확장됩니다. 동적 기억장치를 제어할 권한이 사용자에게 있습니다. 이 장의 나머지 부분은 동적 기억장치와 제어 방법에 대한 내용입니다.

단일 레벨 저장 힙(heap)

힙은 동적 기억장치의 할당에 사용되는 기억장치의 한 부분입니다. 어플리케이션에 필요한 동적 기억장치의 양은 힙을 사용하는 프로그램 및 프로시듀어에 의해 처리될 자료에 따라 달라집니다. 오퍼레이팅 시스템은 동적으로 작성되고 삭제되는 복수의 단일 레벨 저장 힙 사용을 허용합니다. ALCHSS 지침은 항상 단일 레벨 저장을 사용합니다. 또한 일부 언어는 동적 기억장치에 대한 테라스페이스의 사용을 지원합니다.

힙의 특성

각각의 힙에는 다음과 같은 특성이 있습니다.

- 시스템은 활성 그룹내의 각각의 힙에 고유 ID를 할당합니다.

  디폴트 힙의 힙 ID는 항상 0입니다.

프로그램이나 프로시듀어에 의해 호출된 기억장치 관리 바인드 가능 API는 힙 ID를 사용하여, 활동하는 힙을 식별합니다. 바인드 가능 API는 힙을 소유하고 있는 활성 그룹내에서 실행되어야 합니다.

- 힙을 작성한 활성 그룹이 힙을 소유합니다.

  활성 그룹이 힙을 소유하므로 힙의 수명은 소유하고 있는 활성 그룹의 수명보다 길지 않습니다. 힙 ID는 힙 ID를 소유하고 있는 활성 그룹 내에서만 의미가 있으며 고유합니다.

- 힙의 *크기*는 할당 요구를 충족시키기 위해 동적으로 확장됩니다.

  힙의 최대 *크기*는 4GB에서 512K바이트를 뺀 값입니다. 총 할당 수(한 번에)가 128000을 초과하지 않을 경우 최대 힙 *크기*가 됩니다.

- 힙의 최대 단일 할당 *크기*는 16MB에서 64KB를 뺀 값까지로 제한됩니다.

디폴트 힙

단일 레벨 저장 기억장치를 사용 중인 활성 그룹내의 디폴트 힙에서 동적 기억장치를 위한 첫 번째 요구가 기억장치 할당 발생에서 디폴트 힙의 작성 결과를 가져옵니다. 동적 기억장치에 대한 후속 요구를 충족시키기 위한 충분한 기억장치가 없으면 힙이 확장되고 기억장치가 추가로 할당됩니다.

할당된 동적 기억장치는 명시적으로 해제되거나 힙이 삭제될 때까지 할당된 채로 남아 있습니다. 디폴트 힙은 소유하고 있는 활성 그룹이 종료될 경우에만 삭제됩니다.

해당 기억장치가 디폴트 힙에서 할당된 경우 같은 활성 그룹내의 프로그램은 자동으로 동적 기억장치를 공유하게 됩니다. 그러나 사용자는 활성 그룹내의 일부 프로그램과 프로시듀어가 사용하는 동적 기억장치를 분리할 수도 있습니다. 하나 이상의 힙을 작성하여 이를 수행하십시오.

사용자 작성 힙

ILE 바인드 가능 API를 사용하여 하나 이상의 힙을 명시적으로 작성하고 또 삭제할 수 있습니다. 이렇게 하여 사용자는 힙은 물론 힙에서 할당되는 동적 기억장치를 관리할 수 있습니다.

예를 들어, 활성 그룹내의 프로그램에 대해 사용자 작성 힙에 할당된 동적 기억장치는 시스템이 공유할 수도 있으며 공유하지 않을 수도 있습니다. 동적 기억장치의 공유는 프로그램에 의해 참조되는 힙 ID에 따라 결정됩니다. 사용자는 둘 이상의 힙을 사용하여 동적 기억장치의 자동적인 공유를 피할 수 있습니다. 이렇게 하면 자료의 논리 그룹을 분리시킬 수 있습니다. 다음은 하나 이상의 사용자 작성 힙을 사용하는 이유입니다.

- 한 번의 요구사항을 충족시킬 수 있도록 특정 기억장치 오브젝트를 함께 그룹화할 수 있습니다. 일단 요구사항이 충족되면 힙 삭제(CEEDSHP) 바인드 가능 API에 대한

단일 호출에 의해 할당된 동적 기억장치를 해제할 수 있습니다. 이 조작으로 동적 기억장치가 해제되고 힙이 삭제됩니다. 이 방법으로 동적 기억장치를 다른 요구를 충족시키기 위해서도 사용할 수 있습니다.

- 힙 표시(CEEMKHP) 및 힙 해제(CEERLHP) 바인드 가능 API를 사용하여 복수의 동적 기억장치를 즉시 해제시킬 수 있습니다. CEEMKHP 바인드 가능 API는 사용자가 힙을 표시할 수 있도록 해줍니다. 힙이 표시되어 할당 그룹을 해제할 준비가 되면 CEERLHP 바인드 가능 API를 사용하십시오. 표시 및 릴리스 기능을 사용하면 힙을 그대로 둔 상태에서 힙에서 할당된 동적 기억장치를 해제할 수 있습니다. 이러한 방법으로, 동적 기억장치 요구사항을 충족시키기 위해 기존의 힙을 재사용하여 힙 작성에 연관된 시스템 오버헤드를 막을 수 있습니다.
- 사용자의 기억장치 요구사항이 디폴트 힙을 정의하는 기억장치 속성과 일치하지 않을 수도 있습니다. 예를 들어, 디폴트 힙의 초기 *크기*는 4K바이트입니다. 사용자는 4K바이트를 초과하는 많은 동적 기억장치의 할당을 요구합니다. 4K바이트보다 큰 초기 *크기*의 힙을 작성할 수 있습니다. 이로써 힙의 *크기*를 내재적으로 확장시키고 지속적으로 힙 확장 부분에 액세스하는 두 경우 모두에서 발생할 수 있는 시스템 오버헤드를 감소시킬 수 있습니다. 유사한 방법으로 4K 바이트보다 큰 힙 확장 부분을 가질 수 있습니다. 힙 *크기* 정의에 관한 정보는 132 페이지의 『힙 할당 전략』 및 힙 속성에 관한 부분을 참조하십시오.

디폴트 힙보다 복수 힙을 사용하는 것에는 또 다른 이유가 있을 수 있습니다. 기억장치 관리 바인드 가능 API를 사용하여 사용자가 작성한 힙과 그와 같은 힙에 할당된 동적 기억장치를 관리할 수 있습니다. IBM에서는 기억장치 관리 바인드 가능 API를 설명하는 온라인 정보를 제공합니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

단일 힙 지원

기본적인 복수 힙 기억장치 지원이 없는 언어는 디폴트 단일 레벨 저장 힙을 사용합니다. 힙 삭제(CEEDSHP), 힙 표시(CEEMKHP) 또는 힙 해제(CEERLHP) 바인드 가능 API는 디폴트 힙과 함께 사용할 수 없습니다. 명시적 해제 조작을 사용하거나 힙을 소유한 활성 그룹을 종료하여 디폴트 힙에서 할당한 동적 기억장치를 해제할 수 있습니다.

이와 같은 제한은 할당된 동적 기억장치가 혼합 언어 어플리케이션에서 우연히 해제되는 것을 막아줍니다. 힙 해제 및 힙 삭제 조작은 다른 기억장치 지원에 의해 기존 코드가 재사용될 가능성이 있는 큰 어플리케이션에는 부적당합니다. 디폴트 힙에 유효한 힙 해제 조작은 사용하지 마십시오. 유효한 힙 해제 조작을 사용하면 개별적으로 사용될 때 표시 기능이 올바르게 사용되는 어플리케이션의 많은 부분이 함께 사용될 때 실패할 수 있습니다.

힙 할당 전략

디폴트 힙과 연관된 속성은 디폴트 할당 전략을 통해 시스템에 의해 정의됩니다. 이 할당 전략은 속성(예를 들면, 힙 작성 *크기*를 4K바이트로 하고, 확장 *크기*를 4K바이트가 되도록 함)을 정의합니다. 디폴트 할당 전략은 사용자가 변경할 수 없습니다.

그러나 힙 작성(CEECRHP) 바인드 가능 API를 통해 명시적으로 작성한 힙은 제어할 수 있습니다. 또한 힙 할당 전략 정의(CEE4DAS) 바인드 가능 API를 통해 명시적으로 작성된 힙에 대해서는 할당 전략도 정의할 수 있습니다. 그러면 힙을 명시적으로 작성할 때 사용자가 정의한 할당 전략에 의해 힙 속성이 제공됩니다. 이 방법으로, 사용자는 하나 이상의 명시적으로 작성된 힙에 대해 논리 할당 전략을 정의할 수 있습니다.

할당 전략을 정의하지 않고도 CEECRHP 바인드 가능 API를 사용할 수 있습니다. 이러한 경우 힙은 _CEE4ALC 할당 전략 유형의 속성에 의해 정의됩니다. _CEE4ALC 할당 전략 유형은 힙 작성 *크기*와 확장 *크기*를 4K바이트로 지정합니다. _CEE4ALC 할당 전략 유형에는 다음 속성이 포함됩니다.

```
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 */
```

여기에 나오는 속성은 _CEE4ALC 할당 전략 유형의 구조를 설명하기 위한 것입니다. IBM에서는 모든 _CEE4ALC 할당 전략 속성을 설명하는 온라인 정보를 제공합니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

단일 레벨 저장 힙 인터페이스

모든 힙 조작에는 바인드 가능 API가 제공됩니다. 바인드 가능 API나 언어 고유 함수 또는 둘다를 사용하여 어플리케이션을 작성할 수 있습니다.

바인드 가능 API는 다음과 같은 범주로 분류됩니다.

- 기본 힙 조작. 이들 조작은 디폴트 힙과 사용자 작성 힙에 사용될 수 있습니다.

  기억장치 해제(CEEFRST) 바인드 가능 API는 이전에 할당된 하나의 힙 기억장치를 해제합니다.

  힙 기억장치 확보(CEEGTST) 바인드 가능 API는 힙내에 기억장치를 할당합니다.

  기억장치 재할당(CEECZST) 바인드 가능 API는 이전에 할당된 기억장치의 크기를 변경합니다.
- 확장된 힙 조작. 이들 조작은 사용자 작성 힙에서만 사용될 수 있습니다.

  힙 작성(CEECRHP) 바인드 가능 API는 새로운 힙을 작성합니다.

힙 삭제(CEEDSHP) 바인드 가능 API는 기존의 힙을 삭제합니다.
힙 표시(CEERLHP) 바인드 가능 API는 CEERLHP 바인드 가능 API에 의해 해제시킬 힙 기억장치를 식별하는 데 사용될 수 있는 토큰을 리턴시킵니다.
힙 해제(CEERLHP) 바인드 가능 API는 표시가 지정되었기 때문에 힙에서 할당된 모든 기억장치를 해제합니다.

- 힙 할당 전략
  힙 할당 전략 정의(CEE4DAS) 바인드 가능 API는 CEECRHP 바인드 가능 API를 통해 작성된 힙의 속성을 결정하는 할당 전략을 정의할 수 있습니다.

IBM에서는 기억장치 관리 바인드 가능 API에 관한 특정 온라인 정보를 제공합니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

힙 지원

디폴트로 malloc, calloc, realloc 및 new가 제공하는 동적 기억장치는 활성 그룹에서 루트 프로그램의 기억장치 모델과 동일한 유형의 기억장치입니다. 그러나 단일 레벨 기억장치 모델이 사용 중이면 TERASPACE(*YES *TSIFC) 컴파일러 옵션이 지정된 경우 이 인터페이스가 테라스페이스 기억장치를 제공합니다. 유사하게, 단일 레벨 저장 기억장치 모델 프로그램은 _C_TS_malloc, _C_TS_free, _C_TS_realloc 및 _C_TS_calloc와 같은 테라스페이스와 작업하기 위해 명시적으로 바인드 가능한 API를 사용할 수 있습니다.

테라스페이스 기억장치를 사용할 수 있는 방법에 대한 세부사항은 57 페이지의 제 4 장 『테라스페이스 및 단일 레벨 저장』을 참조하십시오.

CEExxxx 기억장치 관리 바인드 가능 API 및 ILE C malloc(), calloc(), realloc() 및 free() 함수를 모두 사용하도록 선택한 경우 다음 규칙이 적용됩니다.

- C 함수 malloc(), calloc() 및 realloc()를 통해 할당된 동적 기억장치는 CEEFRST 및 CEECZST 바인드 가능 API로 해제하거나 재할당할 수 없습니다.
- CEEGTST 바인드 가능 API에 의해 할당된 동적 기억장치는 free() 함수로 해제할 수 있습니다.
- CEEGTST 바인드 가능 API에 의해 초기 할당된 동적 기억장치는 realloc() 함수로 재할당할 수 있습니다.

COBOL과 같은 다른 언어는 힙 기억장치 모델이 없습니다. 이들 언어는 동적 기억장치에 대한 바인드 가능 API를 통해 ILE 동적 기억장치 모델에 액세스할 수 있습니다.

RPG는 디폴트 힙 액세스를 위한 OP 코드 ALLOC, REALLOC 및 DEALLOC와 내장 함수 %ALLOC 및 %REALLOC를 갖습니다. RPG 지원은 CEEGTST, CEECZST 및 CEEFRST 바인드 가능 API를 사용합니다.

제 9 장 예외 및 조건 관리

이 장에서는 예외 처리 및 조건 처리에 대해 보다 자세한 내용을 다룹니다. 본문의 내용을 시작하기 전에 45 페이지의 『오류 처리』에서 설명하고 있는 개념을 먼저 읽으십시오.

OS/400의 예외 메세지 구조는 예외 처리 및 조건 처리 모두를 위해 사용됩니다. 예외 처리와 조건 처리는 서로 상호 작용하는 경우가 있습니다. 예를 들어, 사용자 작성 조건 핸들러 등록(CEEHDLR) 바인드 가능 API로 등록된 ILE 조건 핸들러는 프로그램 메세지 송신(QMHSNDPM) API로 송신된 예외 메세지를 처리하는 데 사용됩니다. 이들 상호 작용에 대한 내용이 이 장에서 설명됩니다. 본문 내용 중 **예외 핸들러**라는 용어는 OS/400 예외 핸들러 또는 ILE 조건 핸들러 중 하나를 의미합니다.

처리 커서 및 재개 커서

예외를 처리하기 위해 시스템은 처리 커서와 재개 커서라는 두 포인터를 사용합니다. 이들 포인터는 예외 처리의 과정을 기록합니다. 사용자는 확장 오류 처리 시나리오의 처리 커서와 재개 커서의 사용에 관해 알아야 합니다. 이들 개념은 뒤에 논의되는 주제에서 추가 오류 처리 피처를 설명하는 데 사용됩니다.

처리 커서는 현재 예외 핸들러를 추적하는 포인터입니다. 이는 시스템이 사용가능한 예외 핸들러를 탐색하는 동안 각 호출 스택 항목에 의해 정의된 예외 핸들러 리스트내의 다음 핸들러로 처리 커서를 이동시킵니다. 이 리스트에는 다음과 같은 내용이 들어 있습니다.

- 직접 모니터 핸들러
- ILE 조건 핸들러
- HLL 특정 핸들러

처리 커서는 예외가 처리될 때까지 우선순위가 낮은 핸들러로 예외 핸들러 리스트를 하향 이동시킵니다. 호출 스택 항목에 정의된 예외 핸들러에 의해 예외가 처리되지 않는 경우 처리 커서는 이전 호출 스택 항목에 대한 첫 번째(최우선 순위의) 핸들러로 이동합니다.

재개 커서는 예외 핸들러가 예외를 처리한 후 작업을 재개하는 현재 위치를 추적하기 위한 포인터입니다. 일반적으로, 시스템은 예외 발생 뒤의 다음 명령어에 재개 커서를 설정합니다. 예외를 발생시킨 프로시듀어 위의 호출 스택 항목에서는 현재 프로시듀어나 프로그램을 중단시킨 프로시듀어 또는 프로그램 호출 바로 다음이 재개점이 됩니다. 재개 커서를 이전 재개점으로 이동시키려면 재개 커서 이동(CEEMRCR) 바인드 가능 API를 사용하십시오.

그림 45에서는 처리 커서와 재개 커서의 예를 보여줍니다.

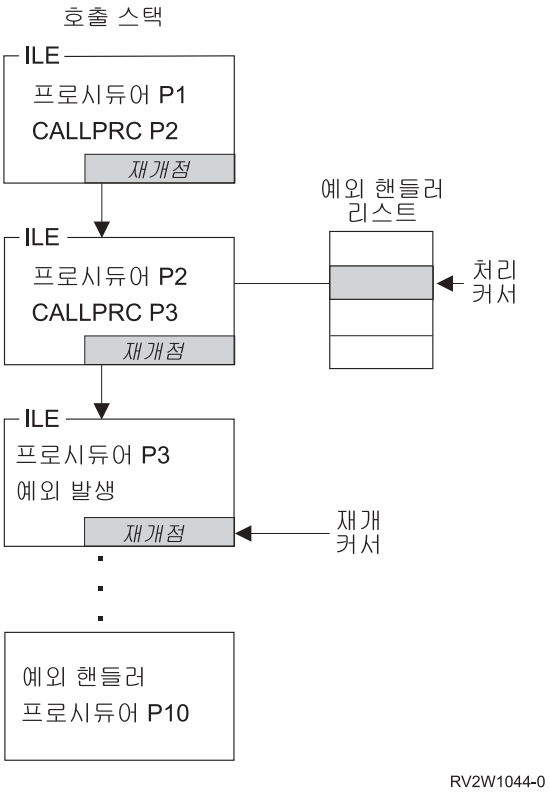

그림 45. 처리 커서와 재개 커서의 예

현재 처리 커서는 프로시듀어 P2에 대한 예외 핸들러 우선순위 리스트에 정의된 두 번째 예외 핸들러에 있습니다. 현재, 핸들러 프로시듀어 P10이 시스템에 의해 호출됩니다. 프로시듀어 P10이 예외를 처리하고 리턴하면 제어는 프로시듀어 P3에 정의된 현재의 재개 커서 위치로 이동합니다. 이 예는 프로시듀어 P3이 프로시듀어 P2로 예외를 여과시켰다고 가정한 것입니다.

예외 핸들러 프로시듀어 P10은 재개 커서 이동(CEEMRCR) 바인드 가능 API를 사용하여 재개 커서를 수정할 수 있습니다. 이 API와 함께 두 개의 옵션이 제공됩니다. 예외 핸들러는 재개 커서를 다음 중 하나로 수정할 수 있습니다.

- 처리 커서를 포함하는 호출 스택 항목
- 처리 커서 이전의 호출 스택 항목

그림 45에서 재개 커서를 프로시듀어 P2나 P1에서 수정했을 수 있습니다. 재개 커서가 수정되고 예외가 처리된 것으로 표시되면 예외 핸들러에서의 정상 리턴이 제어를 새로운 재개점으로 리턴시킵니다.

예외 핸들러 조치

시스템에 의해 예외 핸들러가 호출될 경우 예외를 처리하기 위한 몇 가지 조치를 취할 수 있습니다. 예를 들어, ILE C 부가 제품은 제어 조치, 분기점 핸들러, 메세지 ID별 모니터링을 지원합니다. 여기에 설명된 사용가능한 조치는 다음의 핸들러 유형과 관계가 있습니다.

- 직접 모니터 핸들러
- ILE 조건 핸들러
- HLL 특정 핸들러

처리 재개 방법

사용자가 처리를 계속하도록 결정하는 경우 현재 재개 커서 위치에서 재개할 수 있습니다. 처리를 재개하기 전에 먼저 예외 메세지가 처리 완료되었음을 표시하도록 예외 메세지가 변경되어야 합니다. 어떤 핸들러 유형은 메세지가 처리되었음을 나타내도록 사용자가 예외 메세지를 명시적으로 요구합니다. 그밖의 핸들러 유형의 경우 사용자의 핸들러가 호출되기 전에 시스템이 예외 메세지를 변경할 수 있습니다.

직접 모니터 핸들러의 경우 사용자는 예외 메세지에 취해질 조치를 지정할 수 있습니다. 해당 조치는 핸들러를 호출하거나 핸들러 호출 전에 예외를 처리하거나 예외를 처리하고 프로그램을 재개하는 것일 수 있습니다. 조치가 단지 핸들러 호출인 경우 예외 메세지 변경(QMHCHGEM) API나 바인드 가능 API 처리 조건(CEE4HC)을 사용하여 예외를 처리할 수 있습니다. 재개 커서 이동(CEEMRCR) 바인드 가능 API를 사용하여 직접 모니터 핸들러 안에 있는 재개점을 변경할 수 있습니다. 이러한 변경을 수행한 후 예외 핸들러에서 리턴하여 처리를 계속합니다.

ILE 조건 핸들러의 경우 리턴 코드 값을 설정하여 시스템으로 리턴시킴으로써 계속해서 처리됩니다. IBM에서는 사용자 작성 조건 핸들러 등록(CEEHDLR) 바인드 가능 API의 실제 리턴 코드 값을 설명하는 온라인 정보를 제공합니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

HLL 특정 핸들러의 경우 예외 메세지가 변경되어 핸들러가 호출되기 전에 예외 메세지가 처리되었음을 표시합니다. HLL 특정 핸들러에서 재개 커서를 수정할 수 있는지의 여부를 알려면 ILE HLL 프로그래머 안내서 책을 참조하십시오.

메세지 여과 방법

예외 메세지가 사용자의 핸들러에 의해 인지되지 않도록 결정한 경우 다음 번 사용가능한 핸들러로 예외 메세지를 여과할 수 있습니다. 여과가 발생한 경우 예외 메세지는 처리된 메세지로 간주되어서는 안됩니다. 동일한 또는 이전의 호출 스택 항목에 있는 다른 예외 핸들러에 예외 메세지를 처리할 수 있는 기회가 주어집니다. 예외 메세지의 여과 기법은 예외 핸들러의 유형에 따라 다릅니다.

직접 모니터 핸들러의 경우 예외 메세지가 처리되었다는 것을 나타내도록 예외 메세지를 변경하지 마십시오. 예외 핸들러에서의 정상 리턴은 시스템이 메세지를 여과하도록 합니다. 호출 스택 항목에 대한 예외 핸들러 리스트상의 다음 예외 핸들러로 메세지가 여과됩니다. 핸들러가 예외 핸들러 리스트의 끝에 있는 경우 메세지는 이전 호출 스택 항목의 첫 번째 예외 핸들러로 여과됩니다.

ILE 조건 핸들러의 경우 리턴 코드 값을 설정하고 시스템으로 리턴하여 여과 조치를 전달합니다. IBM에서는 사용자 작성 조건 핸들러 등록(CEEHDLR) 바인드 가능 API의 실제 리턴 코드 값을 설명하는 온라인 정보를 제공합니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

HLL 특정 핸들러의 경우 예외 메세지를 여과하는 것이 불가능할 수도 있습니다. 메세지의 여과 여부는 사용자의 HLL이 메세지를 핸들러가 호출되기 전에 처리된 것으로 표시하는지 여부에 따라 달라집니다. HLL 특정 핸들러를 선언하지 않은 경우 사용자의 HLL은 미처리 예외 메세지를 여과할 수 있습니다. HLL 특정 핸들러가 처리할 수 있는 예외 메세지를 알아보려면 ILE HLL 참조서를 참조하십시오.

메세지 승격 방법

제한된 특정 상황하에서 예외 메세지를 다른 메세지로 수정하도록 선택할 수 있습니다. 이 조치는 초기 예외 메세지를 처리된 것으로 표시한 후 새로운 예외 메세지로 예외 처리를 재시작합니다. 이 조치는 직접 모니터 핸들러와 ILE 조건 핸들러에서만 허용됩니다.

직접 모니터 핸들러의 경우 메세지 승격을 위해 메세지 승격(QMHPRMM) API를 사용할 수 있습니다. 시스템은 상태 및 이탈 메세지 유형만 승격시킬 수 있습니다. 이 API를 통해 예외 처리를 계속하기 위해 사용되는 처리 커서 배치에 대한 일부 제어를 제공받습니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

ILE 조건 핸들러의 경우 리턴 코드 값을 설정하고 시스템으로 리턴하여 승격 조치를 전달합니다. IBM에서는 사용자 작성 조건 핸들러 등록(CEEHDLR) 바인드 가능 API의 실제 리턴 코드 값을 설명하는 온라인 정보를 제공합니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

미처리 예외에 대한 디폴트 조치

예외 메세지가 제어 경계로 여과되는 경우 시스템이 디폴트 조치를 취합니다. 예외가 통지 메세지인 경우 시스템은 디폴트 응답을 송신하고 예외를 처리하여 통지 메세지의 송신자가 처리를 계속할 수 있도록 합니다. 예외가 상태 메세지인 경우 시스템은 예외를 처리하고 상태 메세지의 송신자가 처리를 계속할 수 있도록 합니다. 예외가 이탈 메세지인 경우 시스템은 이탈 메세지를 처리하고 재개 커서가 현재 위치한 지점으로 기능

검사 메세지를 다시 송신합니다. 미처리 예외가 기능 검사인 경우 제어 경계까지의 스택에 있는 모든 항목이 취소되며 CEE9901 이탈 메세지가 그 다음 우선순위의 스택 항목으로 송신됩니다.

표 9에는 예외가 제어 범위에서 처리되지 않을 때 시스템이 취하는 디폴트 응답이 들어 있습니다.

표 9. 미처리 예외에 대한 디폴트 응답

메세지 유형	조건의 심각도	조건 신호(CEESGL) 바인드 가능 API에 의해 발생하는 조건	다른 소스에서 시작되는 예외
상태	0(정보 메세지)	미처리 조건 리턴	메세지 기록 없이 재개
상태	1(경고)	미처리 조건 리턴	메세지 기록 없이 재개
통지	0(정보 메세지)	N/A	통지 메세지 기록 후 디폴트 응답 송신
통지	1(경고)	N/A	통지 메세지 기록 후 디폴트 응답 송신
이탈	2(오류)	미처리 조건 리턴	이탈 메세지 기록 후 현재 재개점의 호출 스택 항목으로 기능 검사 메세지 송신
이탈	3(심각한 오류)	미처리 조건 리턴	이탈 메세지 기록 후 현재 재개점의 호출 스택 항목으로 기능 검사 메세지 송신
이탈	4(심각한 ILE 오류)	이탈 메세지 기록 후 현재 재개점의 호출 스택 항목으로 기능 검사 메세지 송신	이탈 메세지 기록 후 현재 재개점의 호출 스택 항목으로 기능 검사 메세지 송신
기능 검사	4(심각한 ILE 오류)	N/A	어플리케이션 종료 후 제어 경계의 호출자로 CEE9901 메세지 송신

주: 어플리케이션이 미처리 기능 검사에 의해 종료될 때 제어 경계가 활성 그룹내에서 가장 오래된 호출 스택 항목인 경우에는 활성 그룹이 삭제됩니다.

내포된 예외

내포된 예외는 다른 예외가 처리되는 동안에 발생한 예외입니다. 이러한 예외가 발생하면 첫 번째 예외 처리가 일시중단됩니다. 시스템은 처리 커서 및 재개 커서의 위치와 같이 연관된 모든 정보를 저장합니다. 가장 최근에 생성된 예외에서 예외 처리가 다시 시작됩니다. 시스템에 의해 처리 커서와 재개 커서의 새로운 위치가 설정됩니다. 일단 새로운 예외가 적절하게 처리되면 원래 예외에 대한 처리 활동이 정상적으로 재개됩니다.

내포된 예외가 발생할 때 다음 두 가지 항목이 여전히 호출 스택에 남아 있게 됩니다.

- 원래의 예외와 연관된 호출 스택 항목
- 원래의 예외 핸들러와 연관된 호출 스택 항목

예외 처리 루프의 가능성을 줄이기 위해 시스템은 원래의 예외 핸들러 호출 스택 항목에서 내포된 예외의 여과를 중단합니다. 그런 다음, 시스템은 내포된 예외를 기능 검사 메세지로 승격시키고 같은 호출 스택 항목으로 기능 검사 메세지를 여과합니다. 내포된 예외나 기능 검사 메세지를 처리하지 못하는 경우 시스템이 이상 종료(CEE4ABN) 바인드 가능 API를 호출하여 어플리케이션을 종료시킵니다. 이 경우 메세지 CEE9901이 제어 경계의 호출자로 송신됩니다.

내포된 예외를 처리하면서 재개 커서를 이동시키는 경우 원래 예외를 내재적으로 수정할 수 있습니다. 이러한 결과를 발생시키려면 다음과 같이 하십시오.

1. 원래 예외를 발생시킨 호출 스택 항목 이전의 호출 스택 항목으로 재개 커서를 이동시키십시오.
2. 핸들러에서 리턴 후 처리를 재개하십시오.

조건 처리

ILE 조건은 시스템과 무관한 방식으로 표현된 OS/400 예외 메세지입니다. ILE 조건 토큰이 ILE 조건을 표시하는 데 사용됩니다. **조건 처리**는 언어 특정 오류 처리에서 오류를 별도로 처리할 수 있도록 하는 ILE 기능을 참조합니다. 다른 시스템들이 이들 기능을 수행했습니다. 조건 처리를 사용하여 조건 처리를 수행한 시스템간 어플리케이션의 이식성을 높일 수 있습니다.

ILE 조건 처리에는 다음 기능이 포함됩니다.

- ILE 조건 핸들러를 동적으로 등록할 수 있는 기능
- ILE 조건을 신호할 수 있는 기능
- 조건 토큰 구조
- 바인드 가능 ILE API에 대한 선택형 조건 토큰 피드백 코드

이 기능에 관해서는 다음의 주제에서 설명합니다.

조건 표시 방법

ILE **조건 토큰**은 조건의 특성을 나타내기 위한 구조화 필드를 포함하는 12바이트의 혼합 자료 유형입니다. 이러한 특성으로는 조건의 심각도, 조건의 연관 메세지 번호, 제공된 조건의 인스턴스에 고유한 정보 등이 포함될 수 있습니다. 조건 토큰은 시스템, 메세지 서비스, 바인드 가능 API, 프로시듀어의 조건에 관한 정보를 전달하는 데 사용됩니다. 예를 들어, 모든 ILE 바인드 가능 API의 선택형 fc 매개변수로 리턴되는 정보는 조건 토큰을 사용하여 전달합니다.

오퍼레이팅 시스템이나 하드웨어에 의해 예외가 탐색되는 경우 해당 조건 토큰이 시스템에 의해 자동으로 작성됩니다. 또한 조건 토큰 구성(CEENCOD) 바인드 가능 API를 사용하여 조건 토큰을 작성할 수 있습니다. 그런 후 조건 신호(CEESGL) 바인드 가능 API를 통해 토큰을 리턴시켜 시스템에 조건을 신호할 수 있습니다.

조건 토큰 배치

그림 46은 조건 토큰의 맵핑을 보여줍니다. 각 필드의 시작 비트 위치가 표시되어 있습니다.

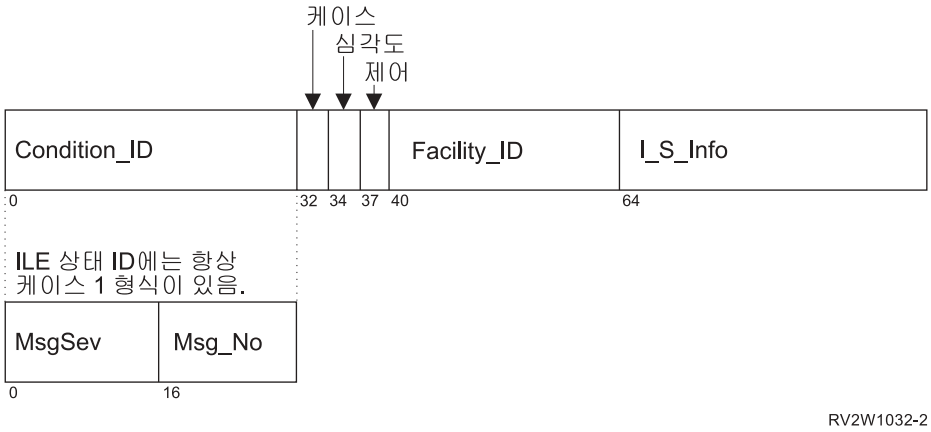

그림 46. ILE 조건 토큰 배치

모든 조건 토큰은 그림 46에 표시된 구성요소를 포함하고 있습니다.

Condition_ID

Facility_ID의 4바이트 ID는 토큰이 전달하는 조건을 설명합니다. ILE 바인드 가능 API 및 대다수의 어플리케이션에서는 케이스 1 조건을 만듭니다.

케이스 토큰의 Condition_ID 부분의 형식을 정의하는 2비트 필드. ILE 조건은 항상 케이스 1입니다.

심각도 조건의 심각도를 나타내는 3비트 2진 정수. 심각도와 MsgSev 필드에는 같은 정보가 들어 있습니다. ILE 조건 심각도의 리스트는 139 페이지의 표 9 부분을 참조하십시오. 해당 OS/400 메세지 심각도에 대해서는 143 페이지의 표 11 및 143 페이지의 표 12를 참조하십시오.

제어 조건 처리의 다양한 특성을 설명하거나 제어하는 플래그를 포함하는 3비트 필드. 세 번째 비트는 Facility_ID를 IBM이 지정한 것인지의 여부를 지정합니다.

Facility_ID

조건을 생성한 기능을 식별하는 세 문자의 영숫자 스트링. Facility_ID는 메세지가 시스템에 의해 생성되었는지 아니면 HLL 실행시 생성되었는지를 표시합니다. 142 페이지의 표 10에는 ILE에서 사용된 기능 ID가 나옵니다.

I_S_Info

주어진 조건의 인스턴스와 연관된 인스턴스 특정 정보를 식별하는 4바이트 필드. 이 필드에는 조건 토큰과 연관된 메세지의 인스턴스에 대한 참조 키가 들어 있습니다. 메세지 참조 키가 0인 경우 연관된 메세지가 없습니다.

MsgSev

조건의 심각도를 나타내는 2바이트 2진 정수. MsgSev와 심각도에는 같은 정

보가 들어 있습니다. ILE 조건 심각도의 리스트는 139 페이지의 표 9 부분을 참조하십시오. 해당 OS/400 메세지 심각도에 대해서는 143 페이지의 표 11 및 143 페이지의 표 12를 참조하십시오.

Msg_No

조건과 연관된 메세지를 식별하는 2바이트 2진수. Facility_ID 및 Msg_No의 조합은 고유하게 조건을 식별합니다.

표 10에는 ILE 조건 토큰 및 OS/400 메세지의 접두부에서 사용되는 기능 ID가 들어 있습니다.

표 10. 메세지와 ILE 조건 토큰에 사용된 기능 ID

기능 **ID**	기능
CEE	ILE 공통 라이브러리
CPF	OS/400 XPF 메세지
MCH	OS/400 기계 예외 메세지

조건 토큰 테스트

바인드 가능 API에서 리턴된 조건 토큰은 다음에 대해 테스트될 수 있습니다.

성공 성공 여부에 대해 테스트하려면 처음 4바이트가 0인지 판별하십시오. 처음 4바이트가 0이고, 나머지 조건 토큰이 0이면 이것은 바인드 가능 API에 대해 성공적으로 호출되었다는 것을 의미합니다.

상응 토큰

두 조건 토큰이 서로 상응한 것인지(같은 *유형*의 조건 토큰이지만 같은 *인스턴스*의 조건 토큰은 아닌) 알아보려면 각각의 처음 8바이트를 비교해보십시오. 이들바이트는 제공된 조건의 모든 인스턴스에 대해 동일합니다.

동등 토큰

두 조건 토큰이 서로 동등한 것인지(같은 인스턴스의 조건을 나타내는) 알아보려면 각각의 12바이트 모두를 비교해보십시오. 마지막 4바이트는 인스턴스에서 조건 인스턴스로 변경될 수 있습니다.

ILE 조건과 OS/400 메세지의 관계

메세지는 ILE에서 발생하는 모든 조건과 연관됩니다. 조건 토큰에는 메세지 파일에 대한 조건과 연관된 메세지를 기록하기 위해 ILE가 사용하는 고유 ID가 포함됩니다.

모든 실행시 메세지의 형식은 **FFFxxxx**입니다.

FFF ILE 및 ILE 언어하에서 생성된 모든 메세지가 사용하는 세 문자의 기능 ID. ID와 해당 기능 리스트는 표 10을 참조하십시오.

xxxx 오류 메세지 번호. 이것은 조건과 연관된 오류 메세지를 식별하는 16진수입니다.

표 11 및 표 12에서는 ILE 조건 심각도를 OS/400 메세지 심각도에 맵핑시키는 방법을 보여줍니다.

*표 11. OS/400 *ESCAPE 메세지 심각도와 ILE 조건 메세지의 맵핑*

OS/400 메세지 심각도에서	**ILE** 조건 심각도로	**OS/400** 메세지 심각도로
0-29	2	20
30-39	3	30
40-99	4	40

*표 12. OS/400 *STATUS 및 *NOTIFY 메세지 심각도와 ILE 조건 심각도의 맵핑*

OS/400 메세지 심각도에서	**ILE** 조건 심각도로	**OS/400** 메세지 심각도로
0	0	0
1-99	1	10

OS/400 메세지 및 바인드 가능 API 피드백 코드

바인드 가능 API에 대한 입력으로, 피드백 코드를 코딩하는 옵션과 프로시듀어에서 피드백 코드를 리턴(또는 피드백) 코드 검사로 사용하는 옵션이 있습니다. 피드백 코드는 다른 프로시듀어에 대한 호출에서 리턴 검사시의 융통성을 위해 제공되는 조건 토큰 값입니다. 따라서, 사용자는 조건 토큰에 대한 입력으로 피드백 코드를 사용할 수 있습니다. 바인드 가능 API 호출시 피드백 코드가 생략되고 하나의 조건이 발생하면 예외 메세지가 바인드 가능 API의 호출자에게 송신됩니다.

바인드 가능 API에서 피드백 정보를 수신하기 위해 사용자 어플리케이션에 피드백 코드 매개변수를 코드화하는 경우 조건이 발생할 때 다음 순서의 이벤트가 발생합니다.

1. 조건과 연관된 메세지를 전달하면서, API의 호출자에게 정보 메세지가 송신됩니다.
2. 조건이 발생한 바인드 가능 API가 조건에 대한 조건 토큰을 빌드합니다. 바인드 가능 API가 정보를 특정 인스턴스 정보 영역에 위치시킵니다. 조건 토큰의 인스턴스 특정 정보는 정보 메세지의 메세지 참조 키입니다. 이것은 시스템에 의해 조건에 반응하기 위해 사용됩니다.
3. 감지된 조건이 아주 심각한(심각도 4) 경우 시스템은 바인드 가능 API의 호출자에게 예외 메세지를 송신합니다.
4. 감지된 조건이 심각한 것이 아니면(심각도 4 미만), 조건 토큰이 바인드 가능 API를 호출한 루틴으로 리턴됩니다.
5. 조건 토큰이 사용자의 어플리케이션으로 리턴될 때 사용자에게 다음 옵션이 제공됩니다.
   - 무시하고 계속 처리합니다.
   - 조건 신호(CEESGL) 바인드 가능 API를 사용하여 조건을 신호합니다.
   - 메세지 확보, 형식, 디스패치(CEEMSG) 바인드 가능 API를 사용하여 표시하기 위해 메세지를 확보하고, 형식화하고, 디스패치합니다.

- 메세지 확보(CEEMGET) 바인드 가능 API를 사용하여 기억장치 영역에 메세지를 저장합니다.
- 사용자가 지정하는 목적지로 사용자 정의 메세지를 디스패치하기 위해 메세지 디스패치(CEEMOUT) 바인드 가능 API를 사용합니다.
- API의 호출자에게 제어가 다시 전달되면 정보 메세지는 제거되어 작업 기록부에 나오지 않습니다.

바인드 가능 API를 호출할 때 피드백 코드 매개변수를 생략하면 바인드 가능 API에서 예외 메세지를 바인드 가능 API의 호출자에게 송신합니다.

제 10 장 디버깅 고려사항

소스 디버거는 OPM, ILE, 서비스 프로그램을 디버그하는 데 사용됩니다. CL 명령은 기본 프로그램 모델(OPM) 프로그램을 디버그하는 데 여전히 사용될 수 있습니다.

이 장에서는 소스 디버거에 대한 몇 가지 고려사항을 다룹니다. 소스 디버거의 사용 방법에 대한 정보는 온라인 정보와 ILE 고급 언어(HLL)를 위한 프로그래머 참조서에서 찾아볼 수 있습니다. 특정 타스크(예 : 모듈 작성)에 대해 사용하는 명령에 대한 정보는 ILE HLL 프로그래머 참조서에서 다룹니다.

디버그 모드

소스 디버거를 사용하려면 세션이 디버그 모드에 있어야 합니다. **디버그 모드**는 일반적인 시스템 기능에 추가로 사용이 가능한 프로그램 디버그 기능의 특별한 환경입니다.

디버그 시작(STRDBG) 명령을 실행할 때 세션이 디버그 모드로 들어갑니다.

디버그 환경

프로그램은 다음의 두 환경 중 어디에서나 디버그될 수 있습니다.

- OPM 디버그 환경. 모든 OPM 프로그램은 OPM 프로그램이 ILE 디버그 환경에 명시적으로 추가되지 않는 한, 이 환경에서 디버그됩니다.
- ILE 디버그 환경. 모든 ILE 프로그램은 이 환경에서 디버그됩니다. 이 외에도 아래의 모든 기준을 충족시키는 경우 OPM 프로그램이 이 환경에서 디버그됩니다.
  - 프로그램이 CL, COBOL, RPG 프로그램인 경우
  - 프로그램이 OPM 소스 디버그 자료로 컴파일된 경우
  - STRDBG 명령의 OPMSRC 매개변수가 *YES로 설정된 경우

ILE 디버그 환경에서는 소스 레벨 디버그 지원을 제공합니다. 디버그 기능은 명령문, 소스, 코드의 리스트 뷰에서 나옵니다.

OPM 프로그램이 일단 ILE 디버그 환경에 놓이면 시스템은 같은 사용자 인터페이스를 통해 ILE 및 OPM 프로그램 모두를 무리 없이 디버깅합니다. ILE 디버그 환경에서 OPM 프로그램을 위한 소스 디버거의 사용법에 관한 정보는 CL, COBOL, RPG 등의 OPM 언어의 해당 ILE 고급 언어(HLL)에 대한 온라인 도움말이나 프로그래머 안내서를 참조하십시오.

디버그 모드로 프로그램 추가

프로그램을 디버그하려면 먼저 프로그램을 디버그 모드에 추가해야 합니다. OPM 프로그램, ILE 프로그램, ILE 서비스 프로그램이 동시에 디버그 모드에 있을 수 있습니다. 한번에 20개까지의 OPM 프로그램이 OPM 환경 디버그 모드에 있을 수 있습니다. 한번에 디버그 모드에 놓일 수 있는 ILE 디버그 환경의 ILE 프로그램, 서비스 프로그램, OPM 프로그램의 수에는 제한이 없습니다. 그러나 한번에 지원되는 디버그 자료의 최대량은 모듈당 16MB입니다.

디버그 모드에 프로그램이나 서비스 프로그램을 추가하려면 해당 프로그램에 *CHANGE 권한이 있어야 합니다. 프로그램이나 서비스 프로그램은 호출 스택에서 중단될 때 디버그 모드에 추가될 수 있습니다.

ILE 프로그램 및 서비스 프로그램은 소스 디버거에 의해 한번에 하나의 모듈씩 액세스됩니다. ILE 프로그램이나 서비스 프로그램 하나를 디버깅할 때 다른 프로그램이나 서비스 프로그램의 모듈을 디버그해야 하는 경우도 있습니다. 두 번째 프로그램을 디버그하려면 먼저 두 번째 프로그램이나 서비스 프로그램을 디버그 모드에 추가해야 합니다.

디버그 모드를 종료할 때 디버그 모드에서 모든 프로그램이 제거됩니다.

디버그에 영향을 주는 감시성(observability) 및 최적화

모듈의 감시성(observability)과 완전한 최적화 여부가 모듈을 디버그하는 기능에 영향을 줍니다.

모듈 **감시성(observability)**이란 다시 컴파일하지 않고도 변경할 수 있도록 모듈과 함께 저장할 수 있는 자료를 말합니다. **최적화**는 시스템이 동일한 출력을 생성하는 데 필요한 시스템 자원량을 줄이기 위해 신속한 처리 방법을 찾는 하나의 처리입니다.

감시성(observability)

모듈 감시성(observability)은 두 가지 유형의 자료로 구성됩니다.

디버그 자료

*DBGDTA 값으로 표시. 이 자료는 모듈을 디버그하는 데 필요합니다.

작성 자료

*CRTDTA 값으로 표시. 이 자료는 코드를 기계 명령어로 변환하는 데 필요합니다. 모듈 최적화 레벨을 변경하려면 모듈에 이 자료가 있어야 합니다.

모듈이 일단 컴파일되면 사용자는 단지 이 자료만 제거할 수 있습니다. 모듈 변경(CHGMOD) 명령을 사용하면 모듈에서 자료 유형 중 하나 또는 둘다를 제거할 수 있습니다. 모든 감시성(observability)을 제거하면 모듈이 최소 크기(압축에 의해)로 줄어

듭니다. 일단 이 자료가 제거되면 사용자가 모듈을 재컴파일하여 자료를 대체하지 않는 한 어떠한 방법으로도 모듈을 변경할 수 없습니다. 재컴파일하려면 소스 코드에 대한 권한이 있어야 합니다.

최적화 레벨

일반적으로 모듈에 작성 자료가 있는 경우 시스템에서의 실행을 위해 소스 코드가 최적화되는 레벨을 변경할 수 있습니다. 신속한 처리가 기계 코드로 변환되어 모듈에서 프로시듀어가 보다 효과적으로 실행될 수 있도록 합니다. 최적화 레벨이 높을수록 모듈의 프로시듀어는 더 효과적으로 실행됩니다.

그러나 보다 많은 최적화로 변수를 변경하지 않거나 디버깅중에 변수의 실제 값을 볼 수 있는 것은 아닙니다. 디버깅시에는 최적화 레벨을 10(*NONE)으로 설정하십시오. 이렇게 하면 모듈의 프로시듀어에는 최하위 레벨의 성능을 제공하지만 사용자는 변수를 정확하게 표시하고 변경할 수 있게 됩니다. 디버깅이 완료된 후 최적화 레벨을 30(*FULL) 또는 40으로 설정하십시오. 이렇게 하면 모듈의 프로시듀어에 최상위 레벨의 성능이 제공됩니다.

디버그 자료 작성 및 제거

디버그 자료는 각 모듈과 함께 저장되며, 모듈이 작성될 때 생성됩니다. 디버그 자료 없이 작성된 모듈에서 프로시듀어를 디버그하려면 디버그 자료로 모듈을 반드시 재작성한 후 모듈을 ILE 프로그램이나 서비스 프로그램으로 리바인드해야 합니다. 이미 자료를 디버그한 프로그램이나 서비스 프로그램에서는 다른 모듈을 모두 재컴파일할 필요가 없습니다.

모듈에서 디버그 자료를 제거하려면 디버그 자료 없이 모듈을 재작성하거나 모듈 변경(CHGMOD) 명령을 사용하십시오.

모듈 뷰

사용가능한 디버그 자료의 레벨은 ILE 프로그램이나 서비스 프로그램의 각 모듈에 따라 다를 수 있습니다. 모듈은 각각 따로 컴파일되며, 서로 다른 컴파일러 및 옵션을 사용하여 생성될 수 있습니다. 이와 같은 디버그 자료 레벨이 컴파일러에 의해 만들어지는 **뷰**와 소스 디버거에 의해 표시되는 뷰를 결정합니다. 사용가능한 값은 다음과 같습니다.

***NONE**

디버그 뷰가 생성되지 않습니다.

***STMT**

디버거에 의해 소스가 표시되지는 않으나 컴파일러 리스트에서 볼 수 있는 프로시듀어명과 명령문 번호를 사용하여 중단점이 추가될 수 있습니다. 이 뷰와 함께 저장되는 디버그 자료의 양은 디버깅에 필요한 자료의 최소량입니다.

***SOURCE**

모듈의 컴파일에 사용된 소스 파일이 아직 시스템에 남아 있는 경우 소스 디버거가 소스를 표시합니다.

***LIST**

리스트 뷰가 생성되어 모듈과 함께 저장됩니다. 따라서, 모듈 작성에 사용된 소스 파일이 시스템에 없더라도 소스 디버거가 소스를 표시할 수 있습니다. 프로그램이 변경될 경우 이 뷰는 백업 사본으로 유용할 수 있습니다. 그러나 특히 다른 파일이 리스트에 확장되는 경우 디버그 자료의 양이 아주 커질 수 있습니다. 모듈 작성시 사용된 컴파일러 옵션이 확장 여부를 결정합니다. 확장시킬 수 있는 파일로는 DDS 파일과 포함 파일(ILE C 포함 파일, ILE RPG /COPY 파일및 ILE COBOL COPY 파일 등)이 있습니다.

***ALL** 모든 디버그 뷰가 생성됩니다. 리스트 뷰의 경우와 마찬가지로, 디버그 자료의 양이 아주 커질 수 있습니다.

ILE RPG도 소스 뷰와 사본 뷰 모두를 생성하는 *COPY 디버그 옵션이 있습니다. 사본 뷰는 모든 /COPY 소스 멤버가 포함된 디버그 뷰입니다.

작업간 디버깅

사용자의 작업이나 일괄처리 작업에서 실행 중인 프로그램을 디버그하기 위해 분리 작업을 사용할 수 있습니다. 이는 디버거 패널의 방해 없이 프로그램의 기능을 감시하고자 하는 경우에 매우 유용합니다. 예를 들어, 어플리케이션이 표시하는 패널이나 창이 단계 이동(stepping)시 또는 중단점에서 디버거 패널에 의해 오버레이되거나 디버거 패널을 중복시킬 수 있습니다. 디버그중인 것과 다른 작업에서 서비스 작업과 디버거를 시작하여 이러한 문제점을 피할 수 있습니다. 자세한 정보는 CL 프로그래밍  책에서 테스트에 대한 부록을 참조하십시오.

OPM 및 ILE 디버거 지원

OPM 및 ILE 디버거 지원을 사용하여 ILE 디버거 API를 통한 OPM 프로그램의 소스 레벨 디버깅이 가능합니다. ILE 디버거 API에 대한 자세한 정보는 iSeries Information Center의 **프로그래밍** 범주 아래에 나오는 *API* 절을 참조하십시오. OPM 및 ILE 디버거 지원은 동일한 사용자 인터페이스를 통해 ILE 및 OPM 프로그램 모두의 무리 없는 디버깅을 제공합니다. 이 지원을 사용하기 위해서는 OPM 프로그램을 RPG, COBOL, CL 컴파일러로 반드시 컴파일시켜야 합니다. 컴파일을 위해서는 OPTION 매개변수인 *SRCDBG나 *LSTDBG를 반드시 설정해야 합니다.

감시(watch) 지원

감시(watch) 지원에서는 지정된 기억장치 위치의 내용이 변경되면 프로그램 실행이 중단되도록 하는 기능을 제공합니다. 기억장치 위치는 프로그램 변수명에 의해 지정됩니다. 프로그램 변수는 기억장치 위치로 결정되며, 이 위치의 내용이 변경되었는지 감시됩

니다. 기억장치 위치의 내용이 변경되면 실행이 중단됩니다. 인터럽트된 프로그램 소스가 일시중단점에서 표시되고, 기억장치 위치를 변경한 명령문 다음에 강조표시된 소스 행이 수행됩니다.

모니터되지 않은 예외

모니터되지 않은 예외가 발견될 경우 실행 중인 프로그램은 기능 검사를 발행하고 작업 기록부로 메세지를 송신합니다. 사용자가 디버그 모드에 있고 프로그램 모듈이 디버그 자료와 함께 작성된 경우 소스 디버거가 '모듈 소스 표시' 화면을 보여줍니다. 필요한 경우 디버그 모드에 프로그램이 추가됩니다. 영향을 받은 행이 강조표시되면서 화면에 해당 모듈이 표시됩니다. 그런 다음, 프로그램을 디버그할 수 있습니다.

디버깅에 대한 글로벌화 제한

다음 중 하나가 존재하는 경우:

- 디버그 작업의 코드화 문자 세트 ID(CCSID)가 290, 930 또는 5026(일본 가다가나)입니다.
- 디버깅에 사용된 장치 설명의 코드 페이지가 290, 930 또는 5026(일본 가다가나)입니다.

디버그 명령, 기능 및 16진 리터럴은 대문자로 입력하십시오. 예를 들면, 다음과 같습니다.

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

일본 가다가나 코드 페이지에 대한 위의 제한은 디버그 명령에 ID명을 사용할 때(예: EVAL) 적용되지 않습니다. 그러나 ILE RPG, ILE COBOL 또는 ILE CL 모듈을 디버깅할 때 디버그 명령에 있는 식별자 이름은 소스 디버거에 의해 대문자로 변환되므로 다르게 표시될 수 있습니다.

제 11 장 자료 관리 범위지정

이 장에서는 ILE 프로그램이나 서비스 프로그램에 의해 사용되는 자료 관리 자원에 관한 정보를 다룹니다. 이 부분을 읽기 전에 53 페이지의 『자료 관리 범위지정 규칙』에서 설명하는 자료 관리 범위지정 개념을 이해하도록 하십시오.

각 자원 유형에 대한 자세한 내용은 각 ILE HLL 프로그래머 안내서를 참조하십시오.

공통 자료 관리 자원

여기에서는 자료 관리 범위지정 규칙을 따르는 모든 자료 관리 자원을 알아봅니다. 다음의 각 자원은 범위지정 방법에 대한 간략한 설명입니다. 각 자원에 대한 보다 자세한 내용은 언급된 관련 책에서 찾아볼 수 있습니다.

열린 파일 조작

열린 파일 조작 결과, 열린 자료 경로(ODP)가 호출한 임시 자원이 작성됩니다. HLL 열기 동사, 조회 파일 열기(OPNQRYF) 명령 또는 데이터베이스 파일 열기(OPNDBF) 명령을 사용하여 열기 기능을 시작할 수 있습니다. ODP는 파일을 연 프로그램의 활성 그룹으로 범위가 지정됩니다. 디폴트 활성 그룹내에서 실행되는 OPM 또는 ILE 프로그램의 경우 호출 레벨 번호에 이르기까지 ODP 범위가 지정됩니다. HLL 열기 동사의 범위를 변경하려는 경우 대체를 사용할 수 있습니다. 모든 대체 명령, OPNDBF 명령 및 OPNQRYF 명령에 열기 범위(OPNSCOPE) 매개변수를 사용하여 범위를 지정할 수 있습니다.

대체 대체는 호출 레벨, 활성 그룹 레벨 또는 작업 레벨에 이르기까지 범위가 지정됩니다. 대체 범위를 지정하려면 대체 명령에서 대체 범위(OVRSCOPE) 매개변수를 사용하십시오. 명시적으로 범위를 지정하지 않을 경우에는 시스템이 대체를 발행하는 위치에 의해 대체 범위가 결정됩니다. 시스템이 디폴트 활성 그룹으로부터 대체를 발행하는 경우 호출 레벨에 이르기까지 범위가 지정됩니다. 시스템이 다른 활성 그룹으로부터 대체를 발행하는 경우에는 활성 그룹 레벨에 이르기까지 범위가 지정됩니다.

확약 정의

확약 정의는 활성 그룹 레벨로의 범위지정과 작업 레벨로의 범위지정을 지원합니다. 범위지정 레벨은 확약 제어 시작(STRCMTCTL) 명령의 제어 범위(CTLSCOPE) 매개변수로 지정됩니다. 확약 제어에 대한 자세한 정보는 백업 및 회복 주제를 참조하십시오.

로컬 SQL 커서

ILE 컴파일러 제품용 SQL 프로그램을 작성할 수 있습니다. ILE 프로그램에

의해 사용되는 SQL 커서는 모듈이나 활성 그룹으로 범위가 지정될 수 있습니다. SQL 프로그램 작성 명령의 SQL 종료(ENDSQL) 매개변수를 통해 SQL 커서 범위를 지정할 수 있습니다.

리모트 SQL 연결

SQL 커서와 함께 사용되는 리모트 연결은 정상 SQL 처리의 일부로서 내재적 활성 그룹으로 범위가 지정됩니다. 이를 통해 하나의 소스 작업과 여러 목표 작업 또는 여러 시스템 사이에 여러 대화가 존재할 수 있습니다.

사용자 인터페이스 관리 프로그램

인쇄 어플리케이션 열기(QUIOPNPA) 및 표시 어플리케이션 열기 API는 어플리케이션 범위 매개변수를 지원합니다. 이들 API를 사용하여 사용자 인터페이스 관리 프로그램(UIM) 어플리케이션을 활성 그룹이나 작업으로 범위를 지정할 수 있습니다. 사용자 인터페이스 관리 프로그램에 대한 자세한 정보는 iSeries Information Center의 **프로그래밍** 범주 아래에서 *API* 절을 참조하십시오.

열린 자료 링크(열린 파일 관리)

링크 작동가능(QOLELINK) API는 자료 링크를 사용할 수 있도록 만듭니다. ILE 활성 그룹내에서 이 API를 사용하는 경우 자료 링크의 범위가 그 활성 그룹에 이르기까지 지정됩니다. 디폴트 활성 그룹 내에서 이 API를 사용하는 경우 자료 링크의 범위가 호출 레벨에 이르기까지 지정됩니다. 열린 자료 링크에 대한 자세한 정보는 iSeries Information Center의 **프로그래밍** 범주 아래에서 *API* 절을 참조하십시오.

공통 프로그래밍 인터페이스(CPI) 통신 대화

대화를 시작하는 활성 그룹이 해당 대화를 소유합니다. 링크 작동가능(QOLELINK) API를 통해 링크를 사용할 수 있도록 만든 활성 그룹이 그 링크를 소유합니다. IBM에는 공통 프로그래밍 인터페이스(CPI) 통신 대화에 관한 온라인 정보가 있습니다. iSeries Information Center의 **프로그래밍** 범주 아래에 나오는 *API* 절을 참조하십시오.

계층 파일 시스템

스트림 파일 열기(OHFOPNSF) API는 계층 파일 시스템(HFS) 파일을 관리합니다. 이 API의 정보 열기(OPENINFO) 매개변수를 사용하여 활성 그룹이나 작업 레벨에 이르기까지 범위지정을 제어할 수 있습니다. 계층 파일 시스템에 대한 자세한 정보는 iSeries Information Center의 **프로그래밍** 범주 아래에서 *API* 절을 참조하십시오.

확약 제어 범위지정

ILE는 확약 제어에 대해 두 가지 변경을 제시합니다.

- 작업당 복수의 독립된 확약 정의. 트랜잭션은 서로 독립적으로 확약되어 롤백될 수 있습니다. ILE 이전에는 작업당 하나의 확약 정의만 허용되었습니다.

- 활성 그룹이 정상적으로 종료할 때 변경사항이 보류 중인 경우 시스템이 내재적으로 변경사항을 확약합니다. ILE 이전에는 시스템이 변경사항을 확약하지 않았습니다.

확약 제어로 사용자는 데이터베이스 파일이나 표와 같은 자원에 대한 변경사항을 단일 트랜잭션으로 정의하고 처리합니다. **트랜잭션**은 사용자에게 단일 아토믹(atomic) 변경으로 표시되어야 하는 시스템상의 오브젝트에 대한 개별적인 변경 그룹입니다. 확약 제어는 다음 중 하나가 시스템에서 발생한다는 것을 보장합니다.

- 개별 변경의 전체 그룹이 발생합니다(**확약** 조작).
- 개별 변경이 발생하지 않습니다(**롤백** 조작).

OPM 프로그램과 ILE 프로그램을 모두 사용하여 다양한 자원을 확약 제어하에서 변경시킬 수 있습니다.

STRCMTCTL(확약 제어 시작) 명령은 작업내에서 실행되는 프로그램이 확약 제어하에서 변경을 수행할 수 있게 합니다. STRCMTCTL 명령으로 확약 제어가 시작할 때 시스템이 **확약 정의**를 작성합니다. 각각의 확약 정의는 STRCMTCTL 명령을 발행한 작업에만 알려집니다. 확약 정의에는 해당 작업내에서 확약 제어하에 변경 중인 자원에 관한 정보가 포함됩니다. 확약 정의의 확약 제어 정보는 확약 자원 변경시 시스템에 의해 유지보수됩니다. 확약 정의는 확약 제어 종료(ENDCMTCTL) 명령을 사용하여 종료됩니다. 확약 제어에 대한 자세한 정보는 백업 및 회복 주제를 참조하십시오.

확약 정의 및 활성 그룹

작업내에서 실행 중인 프로그램에 의해 복수 확약 정의가 시작되고 사용될 수 있습니다. 작업에 대한 각 확약 정의는 그에 관련된 자원이 있는 분리 트랜잭션을 나타냅니다. 이들 자원은 작업을 위해 시작되는 다른 모든 확약 정의에 대해 독립적으로 확약되거나 롤백될 수 있습니다.

주: ILE 프로그램만 디폴트 활성 그룹 이외의 활성 그룹에 대해 확약 제어를 시작할 수 있습니다. 그러므로, 하나의 작업은 하나 이상의 ILE 프로그램을 실행 중일 경우에만 복수의 확약 정의를 사용할 수 있습니다.

기본 프로그램 모델(OPM) 프로그램은 디폴트 활성 그룹내에서 실행됩니다. OPM 프로그램은 디폴트로 *DFTACTGRP 확약 정의를 사용합니다. OPM 프로그램의 경우 STRCMTCTL 명령에 CMTSCOPE(*JOB)를 지정하여 *JOB 확약 정의를 사용할 수 있습니다.

확약 제어 시작(STRCMTCTL) 명령을 사용하는 경우 확약 범위(CMTSCOPE) 매개변수에 확약 정의의 범위를 지정하십시오. 확약 정의를 위한 **범위**는 작업에서 실행되는 어떤 프로그램이 해당 확약 정의를 사용하는지를 나타냅니다. 확약 정의에 대한 디폴트 범위는 STRCMTCTL 명령을 발행하는 프로그램의 활성 그룹에 대한 것입니다. 해당 활성 그룹내에서 실행되는 프로그램만 확약 정의를 사용합니다. 활성 그룹으로 범위가 지정된 확약 정의를 **활성 그룹 레벨**의 확약 정의라고 합니다. OPM 디폴트 활성 그룹

에 대해 활성 그룹 레벨에서 시작되는 확약 정의를 디폴트 활성 그룹(*DFTACTGRP) 확약 정의라고 합니다. 많은 활성 그룹 레벨에 대한 확약 정의가 작업의 다양한 활성 그룹내에서 실행되는 프로그램에 의해 시작되고 사용됩니다.

또한 확약 정의는 작업에서 범위지정될 수도 있습니다. 이와 같은 범위 값의 확약 정의를 **작업 레벨** 또는 *JOB 확약 정의라고 합니다. 활성 그룹 레벨에서 시작되는 확약 정의가 없는 활성 그룹내에서 실행 중인 프로그램은 작업 레벨 확약 정의를 사용합니다. 작업 레벨 확약 정의가 이미 작업의 다른 프로그램에 의해 시작되었을 경우 이러한 상황이 발생합니다. 하나의 작업에 대해서는 단일 작업 레벨 확약 정의만 시작될 수 있습니다.

주어진 활성 그룹의 경우 단 하나의 확약 정의만 해당 활성 그룹 내에서 실행하는 프로그램에 의해 사용될 수 있습니다. 활성 그룹 내에서 실행되는 프로그램은 작업 레벨이나 활성 그룹 레벨 어디에서나 확약 정의를 사용할 수 있습니다. 그러나 두 가지 확약 정의 모두를 동시에 사용할 수는 없습니다.

프로그램은 확약 제어 조작을 실행할 때 요구에 사용할 확약 정의가 어느 것인지 직접 나타내지 않습니다. 대신, 요구하는 프로그램이 실행 중인 활성 그룹에 따라 시스템이 어떤 확약 정의를 사용하는지를 결정합니다. 이것은 어떤 시점에서나 하나의 활성 그룹 내에서 실행되는 프로그램은 단일 확약 정의만 사용할 수 있다는 사실 때문에 가능합니다.

확약 제어 종료

확약 제어는 확약 제어 종료(ENDCMTCTL) 명령을 사용하여 작업 레벨이나 활성 그룹 레벨의 확약 정의에 대해 종료될 수 있습니다. ENDCMTCTL 명령은 요구를 수행하는 프로그램의 활성 그룹에 대한 확약 정의가 종료될 것임을 시스템에 알려줍니다. ENDCMTCTL 명령은 작업에 대해 하나의 확약 정의만 종료합니다. 작업에 대한 다른 모든 확약 정의는 변경되지 않은 채 그대로 남게 됩니다.

활성 그룹 레벨의 확약 정의가 종료되면 활성 그룹내에서 실행되는 프로그램은 더 이상 확약 제어하에서 변경사항을 수행할 수 없습니다. 작업 레벨 확약 정의가 시작되거나 이미 존재하는 경우 확약 제어를 지정하는 새로운 열린 파일 조작은 작업 레벨 확약 정의를 사용합니다.

작업 레벨 확약 정의가 종료되면 작업 레벨 확약 정의를 사용한 작업내의 모든 실행 프로그램은 더 이상 확약 제어하에서 변경사항을 수행할 수 없습니다. 확약 제어가 STRCMTCTL 명령으로 다시 시작되는 경우 변경사항을 수행할 수 있습니다.

활성 그룹 종료시 확약 제어

다음 조건이 동시에 존재할 때:

- 활성 그룹이 종료합니다.

- 작업이 종료하지 않습니다.

시스템이 자동으로 활성 그룹 레벨의 확약 정의를 종료합니다. 다음 두 가지 조건이 모두 존재하는 경우:

- 활성 그룹 레벨의 확약 정의에 대해 확약되지 않은 변경사항이 존재합니다.
- 활성 그룹이 정상적으로 종료하고 있습니다.

시스템이 확약 정의를 종료하기 전에 우선 확약 정의에 대한 내재적 확약 조작을 수행합니다. 한편, 다음 중 어느 한 조건이 존재하는 경우:

- 활성 그룹이 비정상적으로 종료하고 있습니다.
- 활성 그룹으로 범위가 지정된 확약 제어하에서 열린 파일을 닫을 때 시스템이 오류를 발견했습니다.

종료되기 전에 활성 그룹 레벨의 확약 정의 대해 내재적 롤백 조작이 수행됩니다. 활성 그룹이 이상 종료되기 때문에, 시스템은 비정상적인 마지막 확약 조작으로 통지 오브젝트를 갱신합니다. 확약과 롤백은 보류 중인 변경사항에 기초합니다. 보류 중인 변경사항이 없는 경우, 롤백은 수행되지 않지만 통지 오브젝트는 여전히 갱신됩니다. 활성 그룹이 보류 중인 변경사항으로 이상 종료되면 시스템이 내재적으로 변경사항을 롤백합니다. 활성 그룹이 보류 중인 변경사항으로 정상 종료되는 경우, 시스템은 내재적으로 변경사항을 확약합니다.

내재 확약 조작이나 롤백 조작은 절대로 활성 그룹이 *JOB 또는 *DFTACTGRP 확약 정의를 위해 처리를 종료하는 중에는 실행되지 않습니다. *JOB 및 *DFTACTGRP 확약 정의는 절대로 활성 그룹 종료로 인해 종료되지 않기 때문입니다. 그 대신, 이들 확약 정의는 ENDCMTCTL 명령에 의해 명시적으로 종료되거나 아니면 작업 종료시 시스템에 의해 종료됩니다.

활성 그룹 종료시 시스템은 활성 그룹으로 범위가 지정된 파일을 자동으로 닫습니다. 여기에는 확약 제어하에서 열린 활성 그룹으로 범위가 지정된 모든 데이터베이스 파일이 포함됩니다. 활성 그룹 레벨의 확약 정의에 대해 내재적인 확약 조작이 수행되기 전에 이러한 파일에 대한 닫기 조작이 발생합니다. 따라서, 먼저 I/O 버퍼에 상주하는 레코드가 데이터베이스에 강제되어야 내재적인 확약 조작이 실행됩니다.

내재적 확약 조작이나 롤백 조작의 일부로, 시스템은 각 API 확약 자원에 대해 API 확약 및 롤백 나감 프로그램을 호출합니다. 각 API 확약 자원은 활성 그룹 레벨의 확약 정의에 연관되어 있어야 합니다. API 확약 및 롤백 나감 프로그램이 호출된 후 시스템이 API 확약 자원을 자동으로 제거합니다.

다음 조건이 존재하는 경우:

- 활성 그룹이 종료함으로 인해 종료 중인 확약 정의에 대해 내재적 롤백 조작이 이루어집니다.
- 확약 정의에 대한 통지 오브젝트가 정의되어 있습니다.

통지 오브젝트가 갱신됩니다.

제 12 장 ILE 바인드 가능 어플리케이션 프로그래밍 인터페이스

ILE 바인드 가능 어플리케이션 프로그래밍 인터페이스(바인드 가능 API)는 ILE의 중요한 부분입니다. 어떤 경우에는 이들이 특정 HLL에서 제공하는 것 이상의 추가 기능을 제공합니다. 예를 들어, 모든 HLL이 동적 기억장치를 조작하기 위한 고유 방법을 제공하는 것은 아닙니다. 이와 같은 경우 특정 바인드 가능 API를 사용하여 HLL 기능을 보충할 수 있습니다. 만일 사용 중인 HLL에서 특정 바인드 가능 API와 같은 기능을 제공하는 경우 HLL 특정 기능을 사용하십시오.

바인드 가능 API는 HLL과 무관합니다. 이것은 혼합 언어 어플리케이션에 유용하게 사용될 수 있습니다. 예를 들어, 혼합 언어 어플리케이션과 함께 조건 관리 바인드 가능 API만 사용하는 경우 해당 어플리케이션에 대해 공통 조건 처리 의미를 갖게 됩니다. 이것은 복수 HLL 특정 조건 핸들러를 사용할 때보다 조건 관리를 더 일관성 있게 만듭니다.

바인드 가능 API는 다음을 포함하여 광범위한 기능을 제공합니다.

- 활성 그룹 및 제어 흐름 관리
- 조건 관리
- 날짜 및 시간 조작
- 동적 화면 관리
- 산술 기능
- 메세지 처리
- 프로그램 또는 프로시듀어 호출 관리 및 선택적 설명자 액세스
- 소스 디버거
- 기억장치 관리

ILE 바인드 가능 API에 관한 참조 정보는 iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

사용할 수 있는 ILE 바인드 가능 API

대부분의 바인드 가능 API는 ILE가 지원하는 HLL에 대해 사용될 수 있습니다. ILE는 다음과 같은 바인드 가능 API를 제공합니다.

활성 그룹 및 제어 흐름 바인드 가능 API

- 이상 종료(CEE4ABN)
- 제어 경계 찾기(CEE4FCB)
- 활성 그룹 나감 프로시듀어 등록(CEE4RAGE)
- 호출 스택 항목 종료 사용자 나감 프로시듀어 등록(CEERTX)
- 종료 임박 조건 신호(CEETREC)

호출 스택 항목 종료 사용자 나감 프로시듀어 등록 해제(CEEUTX)

조건 관리 바인드 가능 API

조건 토큰 구성(CEENCOD)
조건 토큰 구성 해제(CEEDCOD)
조건 처리(CEE4HC)
재개 커서를 리턴점으로 이동(CEEMRCR)
사용자 작성 조건 핸들러 등록(CEEHDLR)
ILE 버전 및 플랫폼 ID(CEEGPID) 검색
상대 호출 번호 리턴(CEE4RIN)
조건 신호(CEESGL)
사용자 조건 핸들러 등록 해제(CEEHDLU)

날짜 및 시간 바인드 가능 API

릴리언(lilian) 날짜에서 주간 통산일 계산(CEEDYWK)
날짜를 릴리언 형식으로 변환(CEEDAYS)
정수를 초로 변환(CEEISEC)
릴리언 날짜를 문자 형식으로 변환(CEEDATE)
초를 문자 시간소인으로 변환(CEEDATM)
초를 정수로 변환(CEESECI)
시간소인을 초의 수로 변환(CEESECS)
현재 그리니치 표준시 확보(CEEGMT)
현재 현지 시간 확보(CEELOCT)
세계시에서 현지 시간으로 조절되는 오프셋 확보(CEEUTCO)
조절된 세계시 확보(CEEUTC)
세기 조회(CEEQCEN)
국가 또는 영역에 대한 디폴트 날짜 및 시간 스트링 리턴(CEEFMDT)
국가 또는 영역에 대한 디폴트 날짜 스트링 리턴(CEEFMDA)
국가 또는 영역에 대한 디폴트 시간 스트링 리턴(CEEFMTM)
세기 설정(CEESCEN)

산술 바인드 가능 API

각 산술 바인드 가능 API의 이름에서 x는 다음 자료 유형 중 하나를 말합니다.

I 32비트 2진 정수

S 32비트 단일 부동 소수점 수

D 64비트 이중 부동 소수점 수

T 32비트 단일 부동 복소수(실수부와 허수부 모두 32비트 길이임)

E 64비트 이중 부동 복소수(실수부와 허수부 모두 64비트 길이임)

절대 함수(CEESxABS)

아크코사인(CEESxACS)
아크사인(CEESxASN)
아크탄젠트(CEESxATN)
아크탄젠트2(CEESxAT2)
복소수 쌍(Conjugate of Complex)(CEESxCJG)
코사인(CEESxCOS)
코탄젠트(CEESxCTN)
오류 기능과 그 여각(CEESxERx)
지수 기수(CEESxEXP)
지수 표기(CEESxXPx)
팩토리얼(CEE4SIFAC)
부동 복소수 나눗셈(CEESxDVD)
부동 복소수 곱셈(CEESxMLT)
감마 함수(CEESxGMA)
쌍곡선 아크탄젠트(CEESxATH)
쌍곡선 코사인(CEESxCSH)
쌍곡선 사인(CEESxSNH)
쌍곡선 탄젠트(CEESxTNH)
복소수의 허수부(CEESxIMG)
로그 감마 함수(CEESxLGM)
로그 지수 10(CEESxLG1)
로그 지수 2(CEESxLG2)
로그 지수 e(CEESxLOG)
모듈라 산술(CEESxMOD)
근점 정수(CEESxNIN)
근점 자연수(CEESxNWN)
양수 차이(CEESxDIM)
사인(CEESxSIN)
제곱근(CEESxSQT)
탄젠트(CEESxTAN)
부호의 변환(CEESxSGN)
끝수 버림(CEESxINT)

다음은 추가적인 산술 바인드 가능 API입니다.

기본 난수 생성(CEERAN0)

메세지 처리 바인드 가능 API

메세지 디스패치(CEEMOUT)
메세지 확보(CEEMGET)
메세지 확보, 형식 및 디스패치(CEEMSG)

프로그램 또는 프로시듀어 호출 바인드 가능 API

스트링 정보 확보(CEEGSI)
조작 설명자 정보 검색(CEEDOD)
생략 인수 테스트(CEETSTA)

소스 디버거 바인드 가능 API

프로그램에 의한 디버그 명령문 발행 허용(QteSubmitDebugCommand)
세션에 소스 디버거 사용 허용(QteStartSourceDebug)
한 뷰에서 다른 뷰로 위치 대응(QteMapViewPosition)
모듈의 뷰 등록(QteRegisterDebugView)
모듈의 뷰 제거(QteRemoveDebugView)
소스 디버그 세션의 속성 검색(QteRetrieveDebugAttribute)
프로그램에 대해 모듈 및 뷰 리스트 검색(QteRetrieveModuleViews)
프로그램 중단 위치 검색(QteRetrieveStoppedPosition)
지정 뷰에서 소스 텍스트 검색(QteRetrieveViewText)
소스 디버그 세션의 속성 설정(QteSetDebugAttribute)
작업을 디버그 모드에서 가져오기(QteEndSourceDebug)

기억장치 관리 바인드 가능 API

힙 작성(CEECRHP)
힙 할당 전략 정의(CEE4DAS)
힙 삭제(CEEDSHP)
기억장치 해제(CEEFRST)
힙 기억장치 확보(CEEGTST)
힙 마크(CEEMKHP)
기억장치 재할당(CEECZST)
힙 해제(CEERLHP)

동적 화면 관리 프로그램 바인드 가능 API

동적 화면 관리 프로그램(DSM) 바인드 가능 API는 ILE 고급 언어의 화면을 작성하고 관리하기 위한 동적 방법을 제공하는 일련의 화면 I/O 인터페이스입니다.

DSM API는 다음과 같은 기능 그룹으로 구분할 수 있습니다.

- **하위 레벨 서비스**

  하위 레벨 서비스 API는 5250 자료 스트림 명령에 대한 직접 인터페이스를 제공합니다. API는 화면의 상태를 조회 및 조작하거나 화면과 대화하는 입력 및 명령 버퍼를 작성, 조회 및 조작하거나 화면에 필드를 정의하고 자료를 기록하는 데 사용됩니다.

- **창 서비스**

창 서비스 API는 창을 작성, 삭제, 이동, 조절하고 세션 중 동시에 여러 개의 창을 관리하는 데 사용됩니다.

- **세션 서비스**

  세션 서비스 API는 세션을 작성, 조회 및 조작하거나 세션에 대한 입출력 작업을 수행할 수 있는 일반 페이징 인터페이스를 제공합니다.

IBM에서 DSM 바인드 가능 API에 관한 온라인 정보를 제공합니다. iSeries Information Center의 **프로그래밍** 범주에서 *API* 절을 참조하십시오.

제 13 장 고급 최적화 기법

이 장에서는 ILE 프로그램 및 서비스 프로그램을 최적화하기 위해 사용할 수 있는 다음과 같은 기술을 설명합니다.

- 『프로그램 프로파일링』
- 171 페이지의 『프로시듀어간 분석(IPA)』
- 179 페이지의 『사용권 내부 코드 옵션』

프로그램 프로파일링

프로그램 프로파일링은 프로그램 수행 중에 수집된 통계 자료에 따라 프로시듀어, ILE 프로그램 및 서비스 프로그램내에 프로시듀어나 코드를 재순서화하기 위한 고급 최적화 기법입니다. 이 재순서화를 통해 명령어 캐시 이용을 향상시키고 프로그램에 필요한 페이징을 줄일 수 있으며, 그 결과로 성능을 향상시킬 수 있습니다. 프로그램의 의미론적인 특성은 프로그램 프로파일링에 의해 영향을 받지 않습니다.

프로그램 프로파일링에 의한 성능 향상은 어플리케이션의 유형에 따라 다릅니다. 일반적으로 실행시 또는 입출력 처리를 실행하는 데 시간을 소모하기보다는 어플리케이션 코드 그 자체에 대부분의 시간을 소모하는 프로그램이 더 많이 향상될 수 있습니다. 프로파일 자료를 적용할 때 발생되는 프로그램 코드의 성능은 일반적인 사용 중에 프로그램의 가장 중요한 부분을 올바르게 식별하는 최적화 변환 프로그램에 따라 달라집니다. 그러므로 일반 사용자가 수행할 타스크를 수행하고 프로그램이 실행될 환경에서 예상되는 것과 비슷한 입력 자료를 사용하는 경우 프로파일 자료를 수집하는 것이 중요합니다.

프로그램 프로파일링은 다음 조건에 맞는 ILE 프로그램과 서비스 프로그램에 대해서만 사용할 수 있습니다.

- 특별히 V4R2M0 이상의 릴리스용으로 작성된 프로그램
- 프로그램이 V5R2M0 이전의 릴리스용으로 작성된 경우, 프로그램의 대상 릴리스는 현재 시스템의 릴리스와 같아야 합니다.
- *FULL(30) 이상의 최적화 레벨을 사용하여 컴파일된 프로그램. V5R2M0 이상 시스템에서, 최적화 레벨 30 미만을 갖는 바인드시킨 모듈이 허용되지만 어플리케이션 프로파일링에는 참여하지 않습니다.

주: 최적화 요구사항으로 인해 프로그램 프로파일링을 사용하기 위해서는 먼저 프로그램을 완전하게 디버그해야 합니다.

프로파일링 유형

다음과 같은 두 가지 방법으로 프로그램을 프로파일할 수 있습니다.

- 블록순
- 프로시듀어순 및 블록순

블록순 프로파일링은 조건부 분기의 각 측면에서 수행한 횟수를 기록합니다. 블록순 프로파일링 자료가 프로그램에 적용될 때, 다양한 프로파일 기반 최적화가 최적화 변환 프로그램에 의해 프로시듀어 내에서 수행됩니다. 이들 최적화 중 하나에 대해, 코드는 프로시듀어의 가장 자주 실행되는 코드 경로가 프로그램 오브젝트 내에서 연속적이도록 정렬됩니다. 이 재순서화는 명령어 캐시 및 명령어 프리페치 장치 같은 프로세서 구성요소를 보다 효율적으로 활용하여 성능을 향상시킵니다.

프로시듀어순 프로파일링은 각 프로시듀어가 프로그램내에서 다른 프로시듀어를 호출하는 횟수를 기록합니다. 이로 인해 보다 자주 호출되는 프로시듀어가 함께 패키징되도록 프로그램내에서 프로시듀어의 재순서화가 이루어집니다. 이 재순서화는 메모리 페이징을 줄여서 성능을 향상시킵니다.

프로그램에 블록순 프로파일링만 적용하도록 선택할 수 있더라도, 가장 큰 성능 이득을 위해 두 가지 유형 모두를 적용하는 것이 바람직합니다.

프로그램의 프로파일 방법

프로그램의 프로파일링은 5가지 단계로 이루어집니다.

1. 프로그램이 프로파일링 자료를 수집할 수 있도록 하십시오.
2. 프로그램 프로파일링 시작(STRPGMPRF) 명령으로 시스템에서 프로그램 프로파일링 수집을 시작하십시오.
3. 자주 사용하는 코드 경로를 통해 프로그램을 실행하여 프로파일링 자료를 수집하십시오. 프로그램 프로파일링에서 최적화 수행을 위한 프로그램을 실행하는 동안 수집된 통계 자료를 사용하기 때문에 어플리케이션의 일반적인 사용을 통해 이 자료가 수집되도록 하는 것이 중요합니다.
4. 프로그램 프로파일링 종료(ENDPGMPRF) 명령으로 시스템에서 프로그램 프로파일링 수집을 종료하십시오.
5. 수집된 프로파일링 자료에 기초하여 최적화 성능을 위해 코드를 재순서화할 것인지를 요구함으로써 수집된 프로파일링 자료를 프로그램에 적용시키십시오.

프로그램이 프로파일링 자료를 수집할 수 있도록 함

프로그램으로 바인드된 모듈 중 적어도 하나가 프로파일링 자료 콜렉션을 위해 작동가능한 경우 프로그램이 프로파일링 자료 콜렉션을 위해 작동됩니다. 프로파일링 자료를 수집하기 위해 프로그램을 작동하는 것은, 프로파일링 자료를 수집하기 위해 하나 이상의 *MODULE 오브젝트를 변경한 후 이후 모듈로 프로그램을 작성 또는 갱신하거나

프로파일링 자료를 수집하기 위해 프로그램을 작성한 후 이를 변경함으로써 이루어질 수 있습니다. 두 가지 방법 모두 프로파일링 자료를 수집하기 위해 작동된 모듈이 프로그램에 바인드되는 결과를 가져옵니다.

사용하는 ILE 언어에 따라서는 컴파일러 명령 안에 프로파일링 자료 콜렉션을 위해 작동된 모듈을 작성하기 위한 옵션이 있을 수 있습니다. ILE 언어가 최소한 *FULL(30)의 최적화 레벨을 지원하는 경우 프로파일링 자료 콜렉션을 위해 임의의 ILE 모듈을 변경하도록 프로파일링 자료(PRFDTA) 매개변수에 *COL을 지정하여 모듈 변경(CHGMOD) 명령을 사용할 수 있습니다.

프로그램이 프로그램 변경(CHGPGM) 또는 서비스 프로그램 변경(CHGSRVPGM) 명령을 통한 작성 후에 프로파일링 자료를 수집할 수 있게 하려면, 감시가능 프로그램에 대해 다음을 수행하십시오.

- 프로파일링 자료(PRFDTA) 매개변수에 *COL을 지정하십시오. 이 스펙은 다음과 같은 프로그램에 바인드된 모든 모듈에 영향을 줍니다.
  - V4R2M0 이상의 릴리스용으로 작성된 프로그램. V5R2M0 이전의 시스템을 사용 중인 경우, 프로파일링 자료 콜렉션을 작동할 수 있기 위해서는 프로그램이 작성된 동일한 릴리스 레벨에 있는 시스템에 프로그램이 있어야 합니다. 바인드된 모듈에 대해 동일한 제한이 적용됩니다.
  - 최적화 레벨이 30 이상이어야 합니다. V5R2M0 이상 릴리스에서 최적화 레벨 30 미만을 갖는 모든 바인드된 모듈이 허용되지만 어플리케이션 프로파일링에는 참여하지 않습니다.

주: V5R2M0 이전의 릴리스를 갖는 시스템에서 어플리케이션 프로파일링 자료를 수집할 수 있는 프로그램은 해당 자료가 V5R2M0 이상 시스템에 적용되게 할 수 있지만 그 결과는 최적이 아닐 수 있습니다. V5R2M0 이상 시스템에서 프로파일링 자료를 적용하거나 결과 프로그램을 사용하려는 경우, V5R2M0 이상 시스템의 프로그램에 대해 프로파일링 자료 콜렉션을 작동 가능 또는 재작동 가능하게 해야 합니다.

프로파일링 자료를 수집하기 위해 모듈 또는 프로그램을 작동하려면 오브젝트가 재작성되어야 합니다. 그러므로, 프로파일링 자료를 수집하기 위해 모듈 또는 프로그램을 작동하는 데 필요한 시간과 오브젝트를 강제로 재작성(FRCCRT 매개변수)하는 데 소모되는 시간과의 비교가 가능합니다. 또한 최적화 변환 프로그램에 의해 생성되는 추가 기계 명령어로 인해 오브젝트의 크기가 보다 커지게 됩니다.

프로그램 또는 모듈이 프로파일링 자료를 수집하기 위해 일단 작동되면 다음 중 하나가 발생될 때까지 자료 작성 감시성(observability)을 제거시킬 수 없습니다.

- 수집된 프로파일링 자료가 프로그램에 적용됩니다.
- 프로파일링 자료를 수집할 수 없도록 프로그램 또는 모듈이 변경됩니다.

모듈 또는 프로그램이 프로파일링 자료를 수집하기 위해 작동되는지 알아보려면 모듈 표시(DSPMOD), 프로그램 표시(DSPPGM) 또는 서비스 프로그램 표시(DSPSRVPGM) 명령에 DETAIL(*BASIC)을 지정하여 사용하십시오. 프로그램 또는 서비스 프로그램의 경우에는 어느 바인드 모듈(들)이 프로파일링 자료를 수집하기 위해 작동되는지 알아보기 위해 DETAIL(*MODULE)에서 옵션 5(설명 표시)를 사용할 수 있습니다. 자세한 내용은 170 페이지의 『프로그램이나 모듈이 콜렉션용으로 프로파일 또는 작동되는지 알아보는 방법』 주제를 참조하십시오.

주: 프로그램이 이미 프로파일링 자료를 수집했으면(프로그램이 실행되는 동안 수집된 통계 자료), 이 자료는 프로그램이 프로파일링 자료를 수집하기 위해 재작동될 때 지워집니다. 자세한 내용은 168 페이지의 『프로파일링 자료를 수집하기 위해 작동된 프로그램 관리』 부분을 참조하십시오.

프로파일링 자료 콜렉션

프로그램 프로파일링은 프로파일링 자료를 수집하기 위해 작동된 프로그램이 수행될 기계상에서 시작되어야 하며 이는 해당 프로그램이 프로파일링 자료 계수를 갱신할 수 있도록 하기 위한 것입니다. 이것은 프로파일링 자료를 수집하기 전에 장시간 수행되는 큰 어플리케이션을 시작하여 안정 상태에 도달하게 합니다. 이것은 사용자에게 자료 수집 발생 시점에 대한 제어권을 부여합니다.

프로그램 프로파일링 시작(STRPGMPRF) 명령을 사용하여 기계에서 프로그램 프로파일링 수집을 시작하십시오. 기계상에서 프로그램 프로파일링을 종료하려면 프로그램 프로파일링 종료(ENDPGMPRF) 명령을 사용하십시오. IBM에서는 *EXCLUDE 공용 권한으로 두 가지 명령을 모두 제공합니다. 프로그램 프로파일링은 기계가 IPL될 때 내재적으로 종료됩니다.

프로그램 프로파일링이 일단 시작되면 프로파일링 자료를 수집하기 위해 또한 작동되어 실행되는 프로그램 또는 서비스 프로그램이 프로파일링 자료 계수를 갱신합니다. 이것은 STRPGMPRF 명령이 발행되기 전에 프로그램이 활성화되었는지 여부와는 관계 없이 발생합니다.

사용자가 프로파일링 자료를 수집할 프로그램이 기계상의 다중 작업에 의해 호출되는 경우 프로파일링 자료 계수가 이들 작업 모두에 의해 갱신됩니다. 이와 같은 상황을 원하지 않는다면, 프로그램의 복제 사본을 분리 라이브러리에 만들어 해당 사본이 대신 사용되도록 하십시오.

주:

1. 프로그램 프로파일링이 기계상에서 시작되면 프로파일링 자료를 수집하기 위해 작동된 프로그램이 실행되는 동안 프로파일링 자료 계수가 증가합니다. 그러므로, 프로그램이 이 계수를 연속적으로 지우지 않고 이전에 실행된 경우 ″실효된″(stale) 프로파일링 자료 계수를 추가하는 것이 가능합니다. 프로파일링 자료 계수는 여러 가

지 방법을 사용하여 강제로 지울 수 있습니다. 자세한 내용은 168 페이지의 『프로파일링 자료를 수집하기 위해 작동된 프로그램 관리』 부분을 참조하십시오.

2. 프로파일링 자료 계수는 증가될 때마다 DASD에 기록되지 않으므로 프로그램 실행 시 상당한 성능 저하를 발생시킵니다. 프로파일링 자료 계수는 프로그램이 자연적으로 페이지 아웃될 때만 DASD에 기록됩니다. 프로파일링 자료 계수가 DASD에 기록되는지 확인하려면 풀 지우기(CLRPOOL) 명령을 사용하여 프로그램이 실행될 기억장치 풀을 지우십시오.

수집된 프로파일링 자료 적용

수집된 프로파일링 자료의 적용에서는 다음을 수행합니다.

1. 최적의 성능을 위해 프로그램에 있는 프로시듀어(프로시듀어순 프로파일링 자료) 재순서화에 수집된 프로파일링 자료를 사용하도록 기계에 지시합니다.
2. 최적의 성능을 위해 프로그램에 있는 프로시듀어 내의 코드를 재순서화하기 위해 수집된 프로파일링 자료(기본 블록 프로파일링 자료)를 사용하도록 기계에 지시합니다.
3. 프로그램이 프로파일링 자료를 수집하기 위해 작동될 때 이전에 추가된 프로그램에서 기계 명령어를 제거합니다. 그러면 프로그램이 프로파일 자료를 더 이상 수집할 수 없습니다.
4. 수집된 프로파일링 자료를 다음과 같은 관측가능한 자료로 프로그램에 저장합니다.
   - *BLKORD(기본 블록 프로파일링 감시성(observability))
   - *PRCORD (프로시듀어순 프로파일링 감시성(observability))

수집된 자료가 일단 프로그램에 적용된 후에는 다시 적용될 수 없습니다. 프로파일링 자료를 다시 적용하기 위해서는 164 페이지의 『프로그램의 프로파일 방법』에 요약된 단계를 수행해야 합니다. **이전에 적용된 프로파일링 자료는 프로그램이 프로파일링 자료를 수집하기 위해 작동될 때 삭제됩니다.**

이미 수집한 자료를 다시 적용하려면 프로파일링 자료를 적용하기 전에 프로그램 사본을 만드십시오. 이것은 사용자가 프로파일링의 각 유형별(블록순, 프로시듀어순 또는 둘 다)로 어떤 이점이 있는지를 경험한 경우 바람직한 것일 수 있습니다.

프로파일링 자료를 적용하려면 프로그램 변경(CHGPGM) 또는 서비스 프로그램 변경(CHGSRVPGM) 명령을 사용하십시오. 프로파일링 자료(PRFDTA) 매개변수의 경우 다음을 지정하십시오.

- 블록순 프로파일링 자료(*APYBLKORD)
- 두 가지 블록순 및 프로파일링 자료(*APYALL) 또는 (*APYPRCORD)

IBM에서는 *APYALL을 사용할 것을 권장합니다.

프로파일링 자료를 프로그램에 적용하며 프로그램에 두 가지 추가 양식의 감시성(observability)을 작성 및 저장합니다. 추가 감시성(observability)은 프로그램 변경(CHGPGM) 및 서비스 프로그램 변경(CHGSRVPGM) 명령으로 제거할 수 있습니다.

- *BLKORD 감시성(observability)은 블록순 프로파일링 자료가 프로그램에 적용될 때 내재적으로 추가됩니다. 이는 프로그램이 재작성되는 경우에 프로그램에 대해 적용된 블록순 프로파일링 자료를 기계가 보유할 수 있도록 합니다.
- 프로시듀어순 프로파일링 자료를 프로그램에 적용하면 내재적으로 *PRCORD 및 *BLKORD 감시성(observability)을 추가합니다. 이는 프로그램이 재작성되거나 갱신되는 경우에 프로그램에 대해 적용된 프로시듀어순 프로파일링 자료를 기계가 보유할 수 있도록 합니다.

예를 들어, 사용자가 프로그램에 블록순 프로파일링 자료를 적용한 후 연속적으로 *BLKORD 감시성(observability)을 제거할 수 있습니다. 이 경우 프로그램은 여전히 블록순으로 프로파일됩니다. 그러나 프로그램을 재작형시키는 변경이 발생되면 이는 더 이상 블록순으로 프로파일되지 않습니다.

주: *CRTDTA 감시성(observability)을 제거하면 *BLKORD 감시성(observability)도 내재적으로 제거됩니다. 이는 프로그램이 재작성될 때 *BLKORD 감시성(observability)만 필요하기 때문입니다. *CRTDTA 감시성(observability)이 제거되면 프로그램이 재작성될 수 없기 때문에, *BLKORD가 더 이상 필요하지 않게 되어 제거됩니다. 그러나 *PRCORD 감시성(observability)은 제거되지 않습니다.

프로파일링 자료를 수집하기 위해 작동된 프로그램 관리

프로파일링 자료를 수집하기 위해 작동된 프로그램을 프로그램 변경(CHGPGM) 또는 서비스 프로그램 변경(CHGSRVPGM) 명령을 사용하여 변경하면 변경으로 인해 프로그램이 재작성될 필요가 있는 경우에 프로파일링 자료 계수가 내재적으로 제로(0)로 됩니다. 예를 들어, 프로파일링 자료를 수집하기 위해 작동된 프로그램을 최적화 레벨 *FULL에서 최적화 레벨 40으로 변경하면 수집된 프로파일링 자료가 내재적으로 지워집니다. 이는 프로파일링 자료를 수집하기 위해 작동된 프로그램이 복원되고, FRCOBJCVN(*YES *ALL)이 오브젝트 복원(RSTOBJ) 명령에 지정되는 경우에 항상 해당됩니다.

마찬가지로, 프로그램 갱신(UPDPGM) 또는 서비스 프로그램 갱신(UPDSRVPGM) 명령을 사용하여 프로파일링 자료를 수집하기 위해 작동된 프로그램을 갱신하면 결과 프로그램이 프로파일링 자료를 수집하기 위해 계속 작동된 경우에 프로파일링 자료 계수가 내재적으로 지워집니다. 예를 들어, 프로그램 P1에는 모듈 M1과 M2가 들어 있습니다. P1에 바인드된 모듈 M1은 프로파일링 자료를 수집하기 위해 작동되지만 모듈 M2는 작동되지 않습니다. 모듈 중 하나가 작동되는 동안에는 모듈 M1 또는 M2가 있는 프로그램 P1을 갱신하면 프로파일링 자료를 수집하기 위해 프로그램이 계속 작동됩니다. 그리고 모든 프로파일링 자료 계수가 지워집니다. 그러나 모듈 변경(CHGMOD) 명령의 프로파일링 자료(PRFDTA) 매개변수에 *NOCOL을 지정하여 모듈 M1이 프로파일링 자료를 수집하기 위해 더 이상 작동되지 않도록 변경되는 경우 M1이 있는 프로그램 P1을 갱신하면 프로그램 P1이 프로파일링 자료를 수집하기 위해 더 이상 작동되지 않습니다.

프로그램 변경(CHGPGM) 또는 서비스 프로그램 변경(CHGSRVPGM) 명령의 프로파일링 자료(PRFDTA) 매개변수에 *CLR 옵션을 지정하면 프로그램에서 프로파일링 계수를 명시적으로 지울 수 있습니다. 프로그램이 *CLR 옵션을 사용하기 위해서는 활성화 상태이면 안됩니다.

프로그램이 더 이상 프로파일링 자료를 수집하지 않게 하려면 다음 중 하나와 같이 하십시오.

- 프로그램 변경(CHGPGM) 명령의 프로파일링 자료(PRFDTA) 매개변수에 *NOCOL을 지정하십시오.
- 서비스 프로그램 변경(CHGSRVPGM) 명령의 프로파일링 자료(PRFDTA) 매개변수에 *NOCOL을 지정하십시오.

이는 프로그램이 프로파일링 자료를 수집하기 위해 작동되기 전의 상태로 프로그램을 다시 변경합니다. CHGMOD 명령을 사용하거나 모듈을 재컴파일링하여 모듈의 PRFDTA 값을 *NOCOL로 변경하고, 모듈을 프로그램에 리바인드할 수 있습니다.

프로그램에 적용된 프로파일링 자료가 있는 프로그램 관리

프로파일링 자료가 적용된 프로그램을 프로그램 변경(CHGPGM) 또는 서비스 프로그램 변경(CHGSRVPGM) 명령을 사용하여 변경하는 경우 다음의 두 가지 조건 모두가 충족되면 적용된 프로파일링 자료를 유실합니다.

- 변경으로 프로그램을 재작성해야 합니다.

  주: 프로파일링 자료를 적용시킨 프로그램의 최적화 레벨은 최적화 레벨 30보다 작게 변경시킬 수 없습니다. 이것은 프로파일링 자료가 최적화 레벨에 종속되기 때문입니다.

- 필요한 프로파일링 감시성(observability)이 제거되었습니다.

또한 프로파일링 감시성(observability)의 제거 여부와 상관 없이, 변경으로 인해 프로그램이 프로파일링 자료를 수집해야 할 경우에는 적용된 모든 프로파일링 자료가 유실됩니다. 이와 같은 요구는 프로그램이 프로파일링 자료를 수집할 수 있는 결과를 가져옵니다.

다음은 이와 관련된 몇 가지 예입니다.

- 프로그램 A에 프로시듀어순 및 블록순 프로파일링 자료를 적용했습니다. *BLKORD 감시성(observability)은 프로그램에서 제거되었지만 *PRCORD 감시성(observability)은 제거되지 않았습니다. CHGPGM 명령은 프로그램 A의 성능 콜렉션 속성을 변경하기 위해 실행되는 것으로서 프로그램 재작성을 요구합니다. 이러한 변경 요구는 프로그램 A가 더 이상 블록순으로 프로파일되지 않도록 합니다. 그러나 프로시듀어순 프로파일링 자료는 계속 적용됩니다.
- 프로그램 A에 프로시듀어순 및 블록순 프로파일링 자료를 적용했습니다. *BLKORD 및 *PRCORD 감시성(observability)이 프로그램에서 제거되었습니다. CHGPGM 명

령은 프로그램 A의 사용자 프로파일 속성을 변경하기 위해 실행되는 것으로서 프로그램 재작성을 요구합니다. 이러한 변경 요구는 프로그램 A가 더 이상 블록순 또는 프로시듀어순으로 프로파일되지 않도록 합니다. 프로그램 A는 프로파일링 자료가 적용되기 이전의 상태로 다시 되돌아갑니다.

- 프로그램 A에 블록순 프로파일링 자료를 적용했습니다. *BLKORD 감시성(observability)이 프로그램에서 제거되었습니다. CHGPGM 명령은 프로그램의 텍스트를 변경하기 위해 실행되는 것으로서 프로그램 재작성은 필요 **없습니다**. 이 변경 후에 프로그램 A는 여전히 블록순 프로파일링됩니다.
- 프로그램 A에 프로시듀어순 및 블록순 프로파일링 자료를 적용했습니다. 이것은 프로그램에서 *PRCORD 및 *BLKORD 감시성(observability)을 제거시키지 않습니다. CHGPGM 명령을 실행하여 프로파일링 자료를 수집하기 위해 프로그램이 작동되도록 합니다(이로써 프로그램이 재작성됩니다). 프로그램 A가 더 이상 블록순 또는 프로시듀어순으로 프로파일되지 않습니다. 프로그램이 프로파일링 자료가 적용되지 않은 것과 같은 상태가 됩니다. 모든 프로파일링 자료 계수가 지워진 프로파일링 자료를 수집하기 위해 프로그램을 작동시킬 수 있습니다.

프로파일링 자료가 적용된 프로그램(*APYALL, *APYBLKORD 또는 *APYPRCORD)은 CHGPGM 또는 CHGSRVPGM 명령에 PRFDTA(*NOCOL)를 지정하여 프로파일되지 않은 프로그램으로 즉시 변경될 수 없습니다. 이 피처는 프로파일링 자료의 우발적인 손실을 피하기 위한 안전망입니다. 따라서, 프로그램은 먼저 PRFDTA(*COL)로 변경되어 기존 프로파일링 자료를 효과적으로 제거한 후 PRFDTA(*NOCOL)로 변경되어야 합니다.

프로그램이나 모듈이 콜렉션용으로 프로파일 또는 작동되는지 알아보는 방법

프로그램의 프로그램 프로파일링 자료 속성을 알아보려면 프로그램 표시(DSPPGM) 또는 서비스 프로그램 표시(DSPSRVPGM) 명령에 DETAIL(*BASIC)을 지정하여 사용하십시오. "프로파일링 자료"의 값은 다음 값 중 하나가 됩니다.

- *NOCOL - 프로파일링 자료를 수집하기 위해 프로그램이 작동되지 않습니다.
- *COL - 프로그램에 있는 하나 이상의 모듈이 프로파일링 자료를 수집하기 위해 작동됩니다. 이 값은 프로파일링 자료가 실제로 수집되었는지 여부를 나타내는 것은 아닙니다.
- *APYALL - 블록순 및 프로시듀어순 프로파일링 자료가 이 프로그램에 적용됩니다. 프로파일링 자료의 콜렉션이 더 이상 작동되지 않습니다.
- *APYBLKORD - 블록순 프로파일링 자료가 이 프로그램에 있는 하나 이상의 바인드된 모듈의 프로시듀어에 적용됩니다. 이는 프로파일링 자료를 수집하기 위해 이전에 작동된 바인드 모듈에만 적용됩니다. 프로파일링 자료의 콜렉션이 더 이상 작동되지 않습니다.
- *APYPRCORD - 프로시듀어순 프로그램 프로파일링 자료가 이 프로그램에 적용됩니다. 프로파일링 자료의 콜렉션이 더 이상 작동되지 않습니다.

프로시듀어순 프로파일링만 적용하려면 프로그램이 다음과 같은 항목을 충족시켜야 합니다.

- 먼저 *APYALL 또는 *APYPRCORD(*APYALL과 같음)를 지정하여 프로파일합니다.
- 그런 다음 *BLKORD 감시성(observability)을 제거하고 프로그램을 재작성하십시오.

프로그램내에 바인드된 모듈의 프로그램 프로파일링 자료 속성을 표시하려면 DSPPGM 또는 DSPSRVPGM DETAIL(*MODULE)을 사용하고 모듈 레벨에서 이 매개변수의 값을 보려면 프로그램으로 바인드된 모듈에 옵션 5를 지정하십시오. "프로파일링 자료"의 값은 다음 값 중 하나가 됩니다.

- *NOCOL - 프로파일링 자료를 수집하기 위해 바인드된 모듈이 작동되지 않습니다.
- *COL - 프로파일링 자료를 수집하기 위해 바인드된 모듈이 작동됩니다. 이 값은 프로파일링 자료가 실제로 수집되었는지 여부를 나타내는 것은 아닙니다.
- *APYBLKORD - 블록순 프로파일링 자료가 바인드된 모듈 중 하나 이상의 프로시듀어에 적용됩니다. 프로파일링 자료의 콜렉션이 더 이상 작동되지 않습니다.

또한 DETAIL(*MODULE)은 프로그램 프로파일링 자료 속성에 의해 영향받는 프로시듀어 수를 지정하기 위해 다음의 필드를 표시합니다.

- 프로시듀어 수 - 모듈에 있는 총 프로시듀어 수
- 재순서화된 프로시듀어 블록 수 - 이 모듈의 프로시듀어 수는 재순서화된 기본 블록의 수입니다.
- 측정된 블록순 프로시듀어 수 - 블록순 프로파일링 자료가 적용될 때 블록순 프로파일링 자료가 수집된 바인드 모듈 안의 프로시듀어 수. 벤치마크가 실행될 때 프로시듀어가 벤치마크에서 실행되지 않았기 때문에 특정 프로시듀어에 대해 자료가 수집되지 않을 경우가 있습니다. 그러므로, 이 계수는 벤치마크로 실행된 실제 프로시듀어의 수를 반영합니다.

DSPMOD 명령을 사용하여 모듈의 프로파일링 속성을 알아보십시오. "프로파일링 자료"의 값은 다음 값 중 하나가 됩니다. 이는 기본 블록 자료가 독립형 모듈이 아닌, 프로그램에 바인드된 모듈에만 적용될 수 있기 때문에 절대로 *APYBLKORD를 표시하지 않습니다.

- *NOCOL - 프로파일링 자료를 수집하기 위해 모듈이 작동되지 않습니다.
- *COL - 프로파일링 자료를 수집하기 위해 모듈이 작동됩니다.

프로시듀어간 분석(IPA)

이 주제는 CRTPGM 및 CRTSRVPGM 명령에서 IPA 옵션을 통해 사용할 수 있는 프로시듀어간 분석(IPA) 처리의 개요를 제공합니다.

컴파일시 최적화 변환 프로그램이 프로시듀어내에서 그리고 프로시듀어간에서 분석 모두를 수행합니다. 프로시듀어내 분석은 해당 함수 및 컴파일 단위에 사용할 수 있는 정보만 사용하여, 컴파일 단위내에서 각 함수에 대한 최적화를 수행하는 메카니즘입니다. 프로시듀어간 분석은 함수 경계 상호간에 최적화를 수행하는 메카니즘입니다. 최적화 변환 프로그램은 프로시듀어간 분석을 수행하지만 컴파일 단위내에서만 수행합니다. IPA 컴파일러 옵션이 수행하는 프로시듀어간 분석은 위에서 설명한 제한된 프로시듀어간 분석을 개선시켜줍니다. IPA 옵션을 통해 프로시듀어간 분석을 실행할 때 IPA가 전체 프로그램에 걸쳐 최적화를 수행합니다. 또한 그렇지 않으면 최적화 변환 프로그램을 사용하여 컴파일 시간에 사용할 수 없는 최적화도 수행합니다. 최적화 변환 프로그램 또는 IPA 옵션은 다음과 같은 유형의 최적화를 수행합니다.

- *컴파일 단위에서의 인라인*. 인라인은 함수의 실제 코드로 특정 함수 호출을 대체합니다. 인라인은 호출의 오버헤드를 줄일 뿐만 아니라, 호출자에게 전체 함수를 노출시키므로 컴파일러가 사용자 코드를 더 나은 최적화를 수행할 수 있게 합니다.
- *프로그램 파티션*. 프로그램 파티션은 참조 위치를 이용하기 위해 함수를 재순서화하여 성능을 개선시킵니다. 파티션은 메모리에서 자주 근접하여 서로 호출하는 함수를 위치시킵니다. 프로그램 파티션에 대한 자세한 정보는 178 페이지의 『IPA가 작성한 파티션』을 참조하십시오.
- *글로벌 변수의 병합*. 컴파일러는 글로벌 변수를 하나 이상의 구조에 넣고 구조의 처음부터 오프셋을 계산하여 변수에 액세스합니다. 이것은 변수 액세스 비용을 낮추고 자료 집약성(locality)을 이용합니다.
- *코드 정리*. 코드 정리는 사용자 프로그램의 흐름을 합리화합니다.
- *도달할 수 없는 코드 제거*. 도달할 수 없는 코드 제거는 함수내에서 도달할 수 없는 코드를 제거합니다.
- *도달할 수 없는 함수의 호출 그래프 제거*. 도달할 수 없는 함수의 호출 그래프 제거는 100% 인라인되거나 절대로 참조할 수 없는 코드를 제거합니다.
- *프로시듀어내 상수 전파 및 세트 전파*. IPA는 부동 소수점 및 정수 상수를 사용하도록 전파하고 컴파일시 상수 표현식을 계산합니다. 또한 여러 상수 중 하나로 알려진 변수를 사용할 경우 조건 및 스위치 접기를 초래할 수 있습니다.
- *프로시듀어내 포인터 별명 분석*. IPA가 사용에 따른 포인터 정의를 추적하여 포인터 참조 해제가 사용하거나 정의할 수 있는 메모리 위치의 상세 정보를 추적옵니다. 이것은 컴파일러의 다른 부분이 그러한 참조 해제 주위의 코드의 최적화를 더 잘 수행하게 합니다. IPA는 자료 및 함수 포인터 정의를 추적합니다. 포인터가 단일 메모리 위치나 함수를 참조만 할 수 있을 때 IPA는 그것을 메모리 위치나 함수에 대한 명시적 참조가 되게 다시 씁니다.
- *프로시듀어간 복사 전파*. IPA는 표현식을 전파하며 변수 사용을 위한 일부 변수를 정의합니다. 이것은 상수 표현식의 접기를 위한 추가 기회를 작성합니다. 또한 불필요한 변수 사본을 제거합니다.

- *프로시듀어간 도달할 수 없는 코드 및 저장 제거*. IPA는 정의를 제공하는 연산과 함께, 도달할 수 없는 변수의 정의를 제거합니다.
- *참조 (주소) 변환 인수와 값 인수*. IPA는 형식 매개변수가 호출된 프로시듀어에서 쓰여지지 않을 때 참조 (주소) 인수를 값 인수로 변환합니다.
- *정적 변수를 자동 (스택) 변수로 변환*. IPA는 사용이 단일 프로시듀어 호출로 제한되었을 때 정적 변수를 자동 (스택) 변수로 변환합니다.

IPA를 사용하여 최적화시킨 코드의 실행 시간은 컴파일 시간에만 최적화된 코드보다 보통 더 빠릅니다. 모든 어플리케이션이 IPA 최적화에 적합하지는 않지만 IPA를 사용하여 얻는 성능 획득은 다양합니다. 특정 어플리케이션의 경우 프로시듀어간 분석 사용시 어플리케이션의 성능은 개선시킬 수 없습니다. 흔한 경우는 아니지만 프로시듀어간 분석 사용시 어플리케이션 성능이 실제로 저하될 수도 있습니다. 이런 경우 프로시듀어간 분석을 사용하지 않는 것이 좋습니다. 프로시듀어간 분석에서 얻는 성능 개선사항은 어플리케이션의 유형에 좌우됩니다. 성능상의 이점을 극대화할 수 있는 어플리케이션은 다음 특성의 어플리케이션입니다.

- 많은 수의 함수를 포함하고 있습니다.
- 많은 수의 컴파일 단위를 포함하고 있습니다.
- 호출자와 같은 컴파일 단위에 없는 많은 함수를 포함하고 있습니다.
- 많은 수의 입력 및 출력 조작을 수행하지 않습니다.

프로시듀어간 최적화는 다음 조건에 맞는 ILE 프로그램과 서비스 프로그램에 대해서만 사용가능합니다.

- 특히 V4R4M0 이상 릴리스에 대한 프로그램 또는 서비스 프로그램으로 바인드시킨 모듈을 작성했습니다.
- 최적화 레벨 20 (*BASIC) 이상을 사용하여 프로그램 또는 서비스 프로그램으로 바인드시킨 모듈을 컴파일했습니다.
- 프로그램 또는 서비스 프로그램으로 바인드시킨 모듈은 연관된 IL 자료를 갖습니다. 모듈 작성 옵션 MODCRTOPT(*KEEPILDTA)를 사용하여 모듈과 함께 중간 언어 (IL) 자료를 보유하십시오.

주: 최적화 요구사항으로 인해, 프로시듀어간 분석을 사용하기 전에 먼저 프로그램을 완전하게 디버그해야 합니다.

IPA를 사용하여 사용자 프로그램을 최적화하는 방법

IPA를 사용하여 사용자 프로그램 또는 서비스 프로그램 오브젝트를 최적화하려면 다음 단계를 수행하십시오.

1. MODCRTOPT(*KEEPILDTA) 및 20 이상(40 정도)으로 프로그램이나 서비스 프로그램에 필요한 모든 모듈을 컴파일하십시오. DETAIL(*BASIC) 매개변수로 DSPMOD 명령을 사용하여 단일 모듈이 올바른 옵션으로 컴파일되었는지 확인할

수 있습니다. 자료가 표시된 경우 **중간 언어 자료** 필드는 *YES 값을 갖습니다. **최적화 레벨** 필드는 모듈의 최적화 레벨을 나타냅니다.

2. CRTPGM 또는 CRTSRVPGM 명령에서 IPA(*YES)를 지정하십시오. 바인드의 IPA 부분이 실행될 때 시스템이 상태 메세지를 표시하여 IPA 진행을 나타냅니다.

다음과 같은 매개변수를 사용하여 IPA가 사용자 프로그램을 최적화하는 방식을 더 자세히 정의할 수 있습니다.

- 추가 IPA 정보를 제공하려면 IPACTLFILE(*IPA-control-file*)을 지정하십시오. 제어 파일에서 지정할 수 있는 옵션의 리스트는 『IPA 제어 파일 구문』을 참조하십시오.

CRTPGM 명령에 IPA(*YES)를 지정하면 프로그램에 대한 갱신을 허용할 수 없습니다(즉, ALWUPD(*YES)를 지정할 수 없습니다). 또한 이것은 CRTSRVPGM 명령에서 ALWLIBUPD 매개변수에 대해 참입니다. IPA(*YES)와 함께 지정된 경우 매개변수는 ALWLIBUPD(*NO)이어야 합니다.

IPA 제어 파일 구문

IPA 제어 파일은 추가 IPA 처리 지시문이 들어 있는 스트림 파일입니다. 제어 파일은 파일의 멤버일 수 있으며 QSYS.LIB 명명 규칙(예를 들어, `/qsys.lib/mylib.lib/xx.file/yy.mbr`)을 사용합니다. IPACTLFILE 매개변수가 이 파일의 경로 이름을 식별합니다.

제어 파일 지시문에 유효하지 않은 구문이 있으면 IPA가 오류 메세지를 발행합니다.

제어 파일에 다음과 같은 지시문을 지정할 수 있습니다.

exits=name[,name]

항상 프로그램을 종료하는 함수 리스트를 지정합니다. 호출이 프로그램으로 리턴하지 않으므로 이 함수에 대한 호출을 최적화할 수 있습니다(예를 들어, 저장 및 복원 순서를 제거하여). 이 함수가 연관 IL 자료를 가진 프로그램의 다른 부분을 호출해서는 안됩니다.

inline=attribute

컴파일러가 인라인으로 처리하는 기능을 식별하는 방식을 지정합니다. 이 지시문에 대해 다음 속성을 지정할 수 있습니다.

auto 인라인 한계 및 인라인 임계값을 기준으로 인라인 여부를 판별하는 것을 지정합니다. 어떤 인라인 지시문도 자동 인라인을 대체하지 않습니다. 이것이 디폴트입니다.

noauto

IPA가 인라인 지시문을 사용하여 이름으로 지정한 함수만 인라인에 고려하는 것을 지정합니다.

name[,name]

인라인하려는 함수 리스트를 지정합니다. 함수가 인라인될 수도 있고 인라인되지 않을 수도 있습니다.

name[,name] from name[,name]

특정 함수나 함수 리스트가 함수를 호출하는 경우 인라인에 권장되는 후보 함수 리스트를 지정합니다. 함수가 인라인될 수도 있고 인라인되지 않을 수도 있습니다.

inline-limit=num

인라인을 중지하기 전에 함수를 확장시킬 수 있는 최대 상대 *크기*(추상 코드 단위)를 지정합니다. 추상 코드 단위는 함수에서 실행 코드의 *크기*와 비례합니다. 이 숫자에 대한 값이 클수록 더 큰 서브프로그램, 더 많은 서브프로그램 또는 둘다 인라인하도록 허용합니다. 이 지시문은 inline=auto가 on인 경우에만 해당합니다. 디폴트 값은 8192입니다.

inline-threshhold=size

자동 인라인 후보가 될 수 있는 함수의 최대 *크기*(추상 코드 단위)를 지정합니다. 이 지시문은 inline=auto가 on인 경우에만 해당합니다. 디폴트 *크기*는 1024입니다.

isolated=name[,name]

″isolated″ 함수의 리스트를 지정합니다. isolated 함수는 가시적 함수에 액세스 가능한 글로벌 변수를 직접(또는 관련된 호출 체인내의 다른 함수를 통해 간접으로) 참조하거나 변경하지 않는 함수입니다. IPA는 서비스 프로그램에서 바인드시킨 함수가 분리되는 것으로 가정합니다.

lowfreq=name[,name]

자주 호출되지 않을 것으로 예상되는 함수의 이름을 지정합니다. 이것은 보통 오류 처리 함수이거나 추적 함수입니다. IPA는 이 함수에 대한 호출에 있어서 최적화를 덜 수행함으로써 프로그램의 다른 부분을 더 빠르게 작성할 수 있습니다.

missing=attribute

missing 함수의 프로시듀어간 작동 방식을 지정합니다. 누락 함수는 연관된 IL 자료가 없는 함수이며 unknown, safe, isolated, pure 지시문에서 명시적으로 명명되지 않은 함수입니다. 이 지시문은 연관 IL 자료가 없는 라이브러리 루틴에 대한 호출에 있어서 IPA가 안전하게 수행할 수 있는 최적화 정도를 지정합니다.

IPA에는 이 함수내 코드에 대한 가시성이 없습니다. 모든 사용자 참조가 사용자 라이브러리 또는 실행시 라이브러리로 해결되는지 확인해야 합니다.

이 지시문의 디폴트 설정은 unknown입니다. *Unknown*은 그와 같이 누락된 함수에 대한 호출을 통해 사용 및 변경시킬 수 있는 자료에 관한 비관적인 가정을 IPA에 지시합니다. 이 지시문에 대해 다음 속성을 지정할 수 있습니다.

unknown
누락된 함수가 ″unknown″ 함수임을 지정합니다. 아래에서 알 수 없는 지시문에 대한 설명을 참조하십시오. 이것은 디폴트 속성입니다.

safe 누락된 함수가 ″safe″ 함수임을 지정합니다. 아래에서 안전한 지시문에 대한 설명을 참조하십시오.

isolated
누락된 함수가 ″isolated″ 함수임을 지정합니다. 위에서 isolated 지시문에 대한 설명을 참조하십시오.

pure 누락된 함수가 ″pure″ 함수임을 지정합니다. 아래에서 pure 지시문에 대한 설명을 참조하십시오.

noinline=name[,name]
컴파일러가 인라인하지 않는 함수 리스트를 지정합니다.

noinline=name[,name] from name[,name]
특정 함수나 함수 리스트가 함수를 호출하는 경우 컴파일러가 인라인하지 않는 함수 리스트를 지정합니다.

partition=small| medium|large|unsigned-integer
IPA가 작성하는 각 프로그램 파티션의 *크기*를 지정합니다. 파티션의 *크기*는 링크에 필요한 시간 및 생성된 코드의 품질에 직접 비례합니다. 파티션 *크기*가 클수록 링크에 필요한 시간은 더 길어지지만 생성된 코드의 품질은 일반적으로 더 낫습니다.

이 지시문의 디폴트는 medium입니다.

좀 더 자세히 제어하기 위해 부호가 없는 정수 값을 사용하여 파티션 *크기*를 지정할 수 있습니다. 정수는 추상 코드 단위이며 릴리스간에 그 의미가 변경될 수 있습니다. 단시간에 걸쳐 조정하거나 파티션 수를 일정하게 유지해야 할 경우에만 이 정수를 사용하십시오.

pure=name[,name]
pure 함수 리스트를 지정합니다. 이것은 safe 및 isolated 함수입니다. pure 함수는 감시할만한 내부 상태가 없습니다. 이것은 함수 호출에 대해 리턴된 값이 함수의 이전 호출이나 미래 호출과 무관하다는 것을 의미합니다.

safe=name[,name]
safe 함수 리스트를 지정합니다. 이것은 연관된 IL 자료를 가진 임의의 함수를 직간접으로 호출하지 않는 함수입니다. safe 함수는 글로벌 변수를 참조할 수도 있고 변경할 수도 있습니다.

unknown=name[,name]

unknown 함수 리스트를 지정합니다. 이것은 safe, isolated, pure가 아닌 함수입니다.

IPA 사용 주의사항

- IPA를 사용함으로써 바인드 시간이 증가할 수 있습니다. 어플리케이션 크기 및 프로세서 속도에 따라 바인드 시간이 현저히 증가할 수 있습니다.
- IPA는 기존 바인딩 보다 훨씬 더 큰 바인드 프로그램 및 서비스 프로그램을 생성할 수 있습니다.
- IPA의 프로시듀어간 최적화는 프로그램 성능을 현저히 개선시키지만 실패를 발생시킬 오류가 있는 프로그램을 작동시킬 수 있습니다.
- IPA는 인라인으로 함수를 컴파일하므로 상대 스택 프레임 오프셋을 허용하는 API를 사용할 경우 주의하십시오(예를 들어, QMHRCVPM).
- 인라인 함수를 컴파일하기 위해 IPA는 백엔드 인라이너가 아닌 자체 인라이너를 사용합니다. 컴파일 명령에 INLINE 옵션을 사용하는 것과 같이 백엔드 라이너에 제공되는 매개변수들은 무시됩니다. IPA 인라이너를 위한 매개변수들은 IPA 제어 파일에서 제공합니다.

IPA 제한 및 한계

- IPA가 최적화한 바인드 프로그램이나 서비스 프로그램에서는 UPDPGM 또는 UPDSRVPGM 중 하나를 사용할 수 없습니다.
- IPA가 정상적인 소스 디버그 기능으로 최적화된 임의의 프로그램이나 서비스 프로그램을 디버그할 수 없습니다. 이것은 IL 자료내에서 디버그 정보를 유지보수하지 못하며 출력 파티션을 생성할 때 사실상 임의의 디버그 정보를 유실하기 때문입니다. 따라서 소스 디버거가 IPA 프로그램이나 서비스 프로그램을 처리하지 못합니다.
- 10,000개의 출력 파티션으로 제한됩니다. 이 한계에 도달하면 바인드에 실패하며 시스템이 메세지를 송신합니다. 따라서 CRTPGM 또는 CRTSRVPGM 명령을 다시 실행하여 더 큰 파티션 크기를 지정해야 합니다. 174 페이지의 『IPA 제어 파일 구문』에서 파티션 지시문을 참조하십시오.
- 해당 프로그램에 SQL 자료가 들어 있는 경우 사용자 프로그램에 적용할 수 있는 특정 IPA 제한사항이 있습니다. 사용하는 컴파일러가 IL 자료를 유지하는 옵션을 허용하는 경우 이 제한은 적용되지 않습니다. 사용하는 컴파일러가 IL 자료를 유지하는 옵션을 허용하지 않는 경우 아래 나오는 단계를 수행하여 SQL 자료가 들어 있는 프로그램에서 IPA를 사용하십시오. 예를 들어, 삽입된 SQL문이 있는 C 프로그램을 고려하십시오. 일반적으로 CRTSQLCI 명령으로 이 소스를 컴파일하지만 이 명령에는 MODCRTOPT(*KEEPILDTA) 옵션이 없습니다.

  다음 단계를 수행하여 삽입된 SQL 자료 및 IL 자료 모두가 들어 있는 *MODULE을 작성하십시오.

1. CRTSQLCI 명령으로 SQL C 소스 파일을 컴파일하십시오. OPTION (*NOGEN)과 TOSRCFILE(QTEMP/QSQLTEMP) 컴파일러 옵션을 지정하십시오. 이 단계는 SQL문을 사전컴파일하며 SQL 사전컴파일러를 원래 소스 파일의 연관된 공간으로 위치시킵니다. 또한 임시 소스 실제 파일 QTEMP/QSQLTEMP에 있는 같은 이름의 멤버 안으로 C 소스를 위치시킵니다.
2. QTEMP/QSQLTEMP에 있는 C 소스 파일을 컴파일러 명령에서 MODCRTOPT (*KEEPILDTA) 옵션으로 컴파일하십시오. 이 조치는 SQL C *MODULE 오브젝트를 작성하며 소스 원본 파일의 연관된 위치에서 모듈 오브젝트로 프리프로세서 자료를 전파합니다. 또한 이 *MODULE 오브젝트에는 IL 자료가 들어 있습니다. 이제 IPA(*YES) 매개변수를 사용하여 CRTPGM 또는 CRTSRVPGM 명령에서 *MODULE 오브젝트를 지정할 수 있습니다.

- IPA는 최적화 레벨 10(*NONE)에서 컴파일하는 모듈을 지원할 수 없습니다. IPA는 상위 최적화 레벨에서만 사용할 수 있는 IL 자료내의 정보를 요구합니다.
- IPA는 IL 자료가 포함되지 않은 모듈을 최적화할 수 없습니다. 이 때문에, IPA는 MODCRTOPT(*KEEPILDTA) 옵션을 제공하는 컴파일러로 작성하는 모듈만 최적화할 수 있습니다. 현재, 이것은 C 및 C++ 컴파일러를 포함합니다.
- 프로그램 입력점을 포함하고 있는 모듈로서 보통 이것이 기본 기능인 프로그램의 경우에는 위에 언급한 올바른 속성이 반드시 필요하며 그러한 속성이 없으면 IPA가 실패합니다. 서비스 프로그램의 경우에는 내보낸 기능들을 포함하고 있는 모듈 중에서 최소한 하나에 위에 언급한 올바른 속성이 반드시 필요하며 그러한 속성이 없으면 IPA가 실패합니다. 프로그램이나 서비스 프로그램 안의 다른 모듈 또한 올바른 속성이 있는 것이 좋지만 반드시 필요한 것은 아닙니다. 올바른 속성이 없는 모듈일지라도 IPA가 받아들이지만 최적화시키지 않습니다.

IPA가 작성한 파티션

IPA가 작성한 최종 프로그램이나 서비스 프로그램은 파티션으로 구성됩니다. IPA는 각 파티션에 대해 *MODULE을 작성합니다. 파티션에는 두 가지 목적이 있습니다.

- 같은 기억장치 영역에 관련 코드를 집중시킴으로써 프로그램에서 참조 위치를 개선시킵니다.
- 해당 파티션에 대한 오브젝트 코드 생성 중 메모리 요구사항을 줄입니다.

세가지 유형의 파티션이 있습니다.

- 초기화 파티션. 초기화 코드 및 자료가 들어 있습니다.
- 1차 파티션. 프로그램에 대한 1차 입력점 정보가 들어 있습니다.
- 2차 또는 기타 파티션.

IPA는 다음 방식으로 각 유형의 파티션 수를 판별합니다.

- IPACTLFILE 매개변수가 지정한 제어 파일내의 'partition' 지시문. 이 지시문은 각 파티션을 작성할 때 필요한 크기를 나타냅니다.

- 프로그램 호출 그래프내의 연결성. 연결성이 프로그램에 있는 함수간의 호출 볼륨을 나타냅니다.
- 다른 컴파일 단위에 지정된 컴파일러 옵션간 충돌 분석. IPA는 모든 컴파일 단위간에 공통 옵션을 적용시킴으로써 충돌을 해결하려고 시도합니다. 충돌을 해결할 수 없으면 원래 옵션의 효과가 별도의 파티션으로 유지보수되도록 컴파일 단위를 실행합니다.

  한 예로 *사용권 내부 코드 옵션*(LICOPT)이 있습니다. 두 개의 컴파일 단위에 서로 충돌하는 LICOPT가 있으면 IPA가 이 컴파일 단위의 함수를 동일한 출력 파티션으로 병합할 수 없습니다. 파티션 맵 리스팅 섹션의 예는 203 페이지의 『파티션 맵』을 참조하십시오. IPA는 임시 라이브러리에서 파티션을 작성하며 최종 프로그램이나 서비스 프로그램을 작성하기 위해 연관된 *MODULE을 함께 바인드합니다. IPA는 임의 접두부(예를 들어 QD0068xxxx, 여기서 xxxx는 0000 - 9999의 범위)를 사용하여 파티션 *MODULE 이름을 작성합니다.

  이로 인해 DSPPGM 또는 DSPSRVPGM내의 일부 필드가 예상과 다를 수 있습니다. '프로그램 항목 프로시듀어 모듈'은 *MODULE 파티션 이름을 표시하며 원래 *MODULE 이름을 표시하지 않습니다. 해당 모듈의 'Library' 필드는 원래 라이브러리명이 아닌 임시 라이브러리명을 표시합니다. 또한 모듈의 이름이 프로그램 안으로 바인드되거나 서비스 프로그램이 생성된 파티션 이름이 됩니다. IPA가 최적화시킨 임의의 프로그램이나 서비스 프로그램의 경우 '프로그램 속성' 필드는 IPA로서 그 프로그램이나 서비스 프로그램의 모든 바인드 모듈의 속성 필드가 됩니다.

주: IPA가 파티션 처리를 실행할 때 IPA가 함수나 자료명에 @nnn@이나 XXXX@nnn@ 접두부를 사용하는데 XXXX는 파티션 이름, nnn은 소스 파일 번호입니다. 이것은 정적 함수명과 정적 자료명을 고유하게 유지시켜 줍니다.

사용권 내부 코드 옵션

사용권 내부 코드 옵션(LICOPT)은 코드를 생성하거나 패키지하는 방법에 영향을 주기 위해 사용권 내부 코드에 전달되는 컴파일러 옵션입니다. 전달된 컴파일러 옵션이 모듈, ILE 프로그램 오브젝트 또는 컴파일된 Java 프로그램을 위해 생성되는 코드에 영향을 줍니다. 사용자 코드의 최적화를 조정하기 위해 몇가지 옵션을 조정할 수 있습니다. 일부 옵션은 프로그램의 디버깅에 도움을 줍니다. 이 절에서는 ILE와 관련이 있는 사용권 내부 코드 옵션에 관해서만 설명합니다. Java와 관련된 정보에 대해서는 CRTJVAPGM 명령의 LICOPT 매개변수에 대한 온라인 도움말 정보를 참조하십시오. 기타 정보 소스로는 Information Center에 나오는 IBM Developer Kit for Java 주제가 있습니다.

현재 정의된 옵션

현재 ILE를 위해 정의된 사용권 내부 코드 옵션은 다음과 같습니다.

[No]AlwaysTryToFoliate

최적화 레벨 40으로 컴파일할 때 **call foliation**이라고 하는 최적화를 더 적극적으로 시도하도록 최적화 변환 프로그램에 지시하는 것으로서 실행시 호출 스택에서 유지보수되는 스택 프레임의 수를 줄이려고 시도합니다. 장점은 경우에 따라서 더 적은 수의 스택 프레임이 필요하기 때문에 참조 집약성(locality)을 향상시키고 실행 시 스택 넘침 가능성을 최소화합니다. 단점은 프로그램 실패 이벤트에서 디버그할 때 참조를 위해 사용할 수 있는 단서가 더 적을 수 있다는 것입니다. 이 옵션은 디폴트로 Off입니다.

[No]CallTracingAtHighOpt

최적 레벨 40의 경우에도 스택을 요구하는 프로시듀어 각각에 대해 프로시듀어 프롤로그와 에필로그로 호출 및 리턴 트랩이 삽입되도록 요청하려면 이 옵션을 사용하십시오. 디폴트는 최적화 레벨 40에서 호출 및 리턴 트랩이 임의의 프로시듀어로 삽입되지 않는 것입니다. 호출 및 리턴 트랩을 삽입할 때의 장점은 작업 추적(TRCJOB)을 사용할 수 있다는 것이며, 단점은 실행 시 성능의 저하가 발생할 수 있다는 것입니다. 이 옵션은 디폴트로 Off입니다.

[No]Compact

실행 속도가 늦더라도 가능한 한 코드 크기를 줄이려면 이 옵션을 사용하십시오. 이것은 코드를 인라인으로 복제하거나 확장하는 최적화를 금지함으로써 이루어집니다. 이 옵션은 디폴트로 Off입니다.

[No]CreateSuperblocks

이 옵션은 수퍼블록의 형성을 제어하는데, 수퍼블록은 수퍼블록 헤더의 경우를 제외하고 제어 흐름 항목이 없는 큰 확장 기본 블록입니다. 또한 추적 전개 및 추적 표시하기 같이 수퍼블록에서 수행되는 특정 최적화를 제어합니다. 수퍼블록 형성 및 최적화는 많은 양의 코드 복제를 유발할 수 있습니다. 이 LICOPT를 사용하여 이들 최적화를 작동 불가능하게 만들 수 있습니다. 이 LICOPT는 프로파일 자료가 적용될 때만 유효합니다. 이 옵션은 디폴트로 On입니다.

[No]DetectConvertTo8BytePointerError

이 옵션은 테라스페이스 기억장치 모델 프로그램에만 적용됩니다(예: 컴파일 중인 소스 언어에 적합한 CRT 명령에 `STGMDL(*TERASPACE)`가 사용된 경우). 이 옵션이 사용되면 16바이트 포인터가 SLS(single level store) 주소를 포함하는 런타임 상황에서 16바이트 포인터로부터 8바이트 포인터로 변환할 때마다 여분의 코드가 생성됩니다. SLS 주소는 8바이트 포인터로 저장되지 않는데 그 이유는 8바이트 포인터가 테라스페이스만 가리킬 수 있기 때문입니다. 16바이트 포인터에서 8바이트 포인터로의 변환에 있어서 SLS 감지가 작동하지 않으며 16바이트 포인터에 SLS 주소가 포함된 경우에는 8바이트 포인터의 후속 사용 시 테라스페이스 안의 임의의 위치가 참조되거나 MCH0601 예외를 발생시킬 수 있습니다. 이와 반대로 감지가 작동할 경우에는 문제점을 명확히 나타내주는 MCH0609 예외가 발생합니다. 이 예외는 SLS 전체에서 디폴트로 작동하며 기억장치 모델 프로그램을 계승합니다. 테

라스페이스 기억장치 모델 프로그램에서는 이러한 감지가 프로그램 호출의 일부로 시작되는 프로그램 입력 프로시듀어에서만 디폴트로 작동하며 다른 프로시듀어에서는 적용되지 않습니다.

하나의 특정 포인터 변환 조작에 대한 감지는 이와는 다른 방법으로 포인터 변환을 수행하기 위한 언어 내장 기능 즉 기계 인터페이스 명령어 RETTSADR(테라스페이스 주소 검색)을 사용하여 처리될 수 있습니다.

이 옵션은 디폴트로 Off입니다.

[No]EnableInlining

이 옵션은 변환 프로그램을 최적화하여 프로시듀어 인라인닝을 제어합니다. 프로시듀어 인라인닝은 프로시듀어에 대한 호출이 프로시듀어 코드의 인라인 사본에 의해 대체됨을 의미합니다. 이 옵션은 디폴트로 On입니다.

[No]FoldFloat

컴파일시 시스템이 상수 부동 소수점 표현식을 평가할 수 있도록 지정합니다. 이 LICOPT가 'Fold 부동 상수' 모듈 생성 옵션을 대체합니다. LICOPT를 지정하지 않으면 모듈 작성 옵션이 사용됩니다.

LoopUnrolling=<option>

최적화 변환 프로그램에 의해 수행되는 루프 전개의 양을 제어하려면 LoopUnrolling 옵션을 사용하십시오. 유효한 값은 루프 전개를 작동 불가능하게 하려면 0, 코드 복제 줄이기를 강조하면서 작은 루프를 전개하려면 1, 적극적으로 루프를 전개하려면 2입니다. 옵션 2를 사용하면 생성되는 코드의 크기가 대체로 증가할 수 있습니다. 디폴트 값은 1입니다.

[No]Maf

부동 소수점 배가(multiply-add) 명령어의 생성을 허용합니다. 이 명령어는 중간 반올림 연산 없이 곱셈과 덧셈을 결합시킵니다. 실행 성능은 향상되지만 계산 결과에 영향을 줍니다. 이 LICOPT가 '배가(multiply-add) 사용' 모듈 작성 옵션을 대체합니다. LICOPT를 지정하지 않으면 모듈 작성 옵션이 사용됩니다.

[No]MinimizeTeraspaceFalseEAOs

현재 하드웨어 로드 및 저장 명령어가 **유효 주소 넘침**(EAO)으로서 알려진 16MB 경계 교차를 감지합니다. EAO 검사는 주소 산술 연산의 일부로서 수행됩니다. 동일하게 생성된 코드는 테라스페이스 및 단일 레벨 저장(SLS) 주소 모두를 처리해야 하므로 유효한 테라스페이스 사용이 거짓 EAO를 발생시킬 수 있습니다. 이 EAO 조건이 문제를 발생하지는 않지만 이 조건들을 처리할 때 상당한 처리 오버헤드를 추가시킵니다. MinimizeTeraspaceFalseEAOs LICOPT는 프로그램에 대해 생성된 하드웨어 명령어 순서에서 차이를 발생시킵니다. 첫째로, 보통의 경우보다 약간 더 느리지만 대부분의 EAQ 발생을 제거한 서로 다른 주소 산술 명령어가 생성됩니다. 또한 보통의 경우에서 더 빠른 코드를 생성하지만 거짓 EAO의 빈도를 증가시킴으로써 일정 수준의 최적화가 금지됩니다. 이 LICOPT를 사용해야 하는 경우로

는 모듈에서 수행되는 대부분의 주소 산술이 일반적인 기본 주소에 보통 16MB보다 더 큰 오프셋 값을 더한 테라스페이스 주소를 계산할 때 입니다. 이 옵션은 디폴트로 Off입니다.

[No]OrderedPtrComp

부호 없는 정수값으로 포인터를 비교하고, 항상 순서화 결과(같음, 보다 작음 또는 보다 큼)를 생성하려면 이 옵션을 사용하십시오. 이 옵션을 사용할 때 다른 공간을 참조하는 포인터가 순서화되지 않은 결과는 비교하지 않습니다. 이 옵션은 디폴트로 Off입니다.

[No]PredictBranchesInAbsenceOfProfiling

프로파일 자료가 제공되지 않을 경우 코드 최적화로 안내하는 정적 분기 예측을 수행할 때 이 옵션을 사용하십시오. 프로파일 자료가 제공되면 이 옵션에 관계 없이 분기 가능성을 예측하기 위해 프로파일 자료가 사용됩니다. 이 옵션은 디폴트로 Off입니다.

[No]PtrDisjoint

이 옵션은 최적화 변환 프로그램이 실행 시 성능을 향상시키기 위해 많은 양의 불필요한 로드를 제거시킬 수 있도록 정형화된(typed-based) 별명 처리를 가능하게 해 줍니다. 비포인터 유형을 통해 포인터의 내용을 액세스하지 않을 경우 어플리케이션이 이 옵션을 안전하게 사용할 수 있습니다. C로 작성된 다음 표현식은 안전하지 않은 방식으로 포인터 값을 액세스하는 것에 관한 설명입니다.

```
void* spp;
... = ((long long*) &spp) [1]; // Access low order 8 bytes of 16-byte pointer
```

Default: NoPtrDisjoint

TargetProcessorModel=<option>

`targetProcessorModel` 옵션은 변환 프로그램이 지정 프로세서 모델에 대해 최적화를 수행하도록 지시합니다. 이 옵션으로 작성된 프로그램은 지원되는 모든 하드웨어 모델에서 샐행되지만 일반적으로 특정 프로세서 모델에서 더 빨리 실행됩니다. Star 프로세서에 유효한 값은 0, POWER4™ 프로세서에 유효한 값은 2입니다. 디폴트는 프로그램 오브젝트와 연관된 목표 릴리스가 결정합니다. V5R2M0부터는 디폴트 값이 2입니다. 그러나 이전 릴리스의 경우에는 디폴트 값이 0입니다.

이 옵션에는 양과 음의 변화가 있으며, 음의 변화는 접두어 'no'로 시작합니다. 음의 변화는 옵션이 적용되지 않는 것을 의미합니다. 사용자가 명시적으로 옵션을 켜거나 끌 수 있도록 하기 위해 부울(boolean) 옵션에도 항상 두 개의 변화가 있습니다. 이와 같은 기능은 디폴트 옵션이 on인 옵션을 off로 만들 때 필요합니다. 옵션 디폴트는 릴리스마다 다를 수 있습니다.

어플리케이션

사용자 프로그램을 컴파일할 때 컴파일러 옵션으로서 사용권 내부 코드 옵션을 지정할 수 있습니다.

또한 기존 오브젝트에 적용하는 순서로 CHGMOD(모듈 변경), CHGPGM(프로그램 변경) 및 CHGSRVPGM(서비스 프로그램 변경) 명령에서 옵션을 지정할 수 있습니다. 명령의 매개변수명은 LICOPT입니다. 다음은 모듈에 사용권 내부 코드 옵션을 적용하는 예입니다.

```
> CHGMOD MODULE(TEST) LICOPT('maf')
```

CHGPGM이나 CHGSRVPGM에 사용될 경우에는 시스템이 지정 사용권 내부 코드 옵션을 ILE 프로그램 오브젝트 안에 포함되는 모든 모듈에 적용합니다. 다음은 ILE 프로그램 오브젝트에 사용권 내부 코드 옵션을 적용하는 예입니다.

```
> CHGPGM PGM(TEST) LICOPT('nomaf')
```

다음은 서비스 프로그램에 사용권 내부 코드 옵션을 적용하는 예입니다.

```
> CHGSRVPGM SRVPGM(TEST) LICOPT('maf')
```

제한사항

사용권 내부 코드 옵션을 적용할 수 있는 프로그램과 모듈의 유형에는 여러 가지 제한이 따릅니다.

- OPM 프로그램에는 사용권 내부 코드 옵션을 적용할 수 없습니다.
- 모듈, ILE 프로그램, 서비스 프로그램 오브젝트는 기본적으로 V4R5M0 릴리스 이상을 위해 작성된 것이어야 합니다.
- V4R5 이상의 프로그램이나 서비스 프로그램 안의 V4R5 이전 바인드 모듈에는 사용권 내부 코드 옵션을 적용할 수 없습니다. 이것은 LICOPT를 적용시킬 수 있는 프로그램 안의 다른 바인드 모듈에는 해당되지 않습니다.

구문

CHGMOD, CHGPGM, CHGSRVPGM 명령에서는 LICOPT 매개변수 값의 대소문자가 의미가 없습니다. 예를 들면, 다음 두 가지 명령 호출이 같은 결과를 나타냅니다.

```
> CHGMOD MODULE(TEST) LICOPT('nomaf')
> CHGMOD MODULE(TEST) LICOPT('NoMaf')
```

여러 사용권 내부 코드 옵션을 함께 지정할 때에는 쉼표로 옵션을 분리시켜야 합니다. 또한 시스템이 옵션의 앞뒤에 있는 모든 공간을 무시합니다. 다음은 이와 관련된 몇 가지 예입니다.

```
> CHGMOD MODULE(TEST) LICOPT('Maf,NoFoldFloat')
> CHGMOD MODULE(TEST) LICOPT('Maf,   NoFoldFloat')
> CHGMOD MODULE(TEST) LICOPT('   Maf   ,   NoFoldFloat   ')
```

부울(boolean) 옵션의 경우에는 시스템이 동시에 두 개의 반대되는 변화를 지정하지 못하게 합니다. 예를 들어, 다음은 시스템이 허용하지 않는 명령입니다.

```
> CHGMOD MODULE(TEST) LICOPT('Maf,NoMaf')   <- NOT ALLOWED!
```

그러나 같은 옵션은 여러 번 지정할 수 있습니다. 다음은 유효한 명령의 예입니다.

```
> CHGMOD MODULE(TEST) LICOPT('Maf, NoFoldFloat, Maf')
```

릴리스 호환성

V4R5M0 이전의 릴리스에 사용권 내부 코드 옵션을 적용시킨 모듈, ILE 프로그램, 서비스 프로그램은 사용자가 이동시킬 수 없습니다. 실제로는 사용자가 오브젝트를 매체나 저장 파일에 저장하려 할 때 시스템이 이전 목표 릴리스를 지정하지 못하게 합니다.

OS/400은 향후 릴리스(또는 PTF를 통해 제공되는 릴리스 안에서)에 새로운 사용권 내부 코드 옵션을 정의할 수 있습니다. 새로운 옵션을 지원하는 첫 릴리스나 그 이상의 릴리스가 있는 시스템에서 옵션을 사용할 수 있습니다. 새로운 옵션을 적용시킨 모듈, ILE 프로그램, 서비스 프로그램은 옵션을 지원하지 않는 릴리스로 이동시킬 수 있습니다. 그러나 반드시 V4R5M0 이상의 릴리스여야 합니다. 명령의 LICOPT 매개변수에 옵션을 지정하지 않으면 시스템이 재변환 오브젝트에 지원되지 않는 사용권 내부 코드 옵션을 무시하고 더 이상 적용하지 않습니다. 이와 같은 유형의 재작성은 시스템이 CHGMOD, CHGPGM, CHGSRVPGM 명령에 LICOPT(*SAME)를 사용하여 오브젝트를 재작성할 때 발생할 수 있습니다. 또한 이와 같은 재작성 유형은 시스템이 자동으로 오브젝트를 변환할 경우에도 발생할 수 있습니다. 이것이 재작성을 방해하지 않습니다. 다른 한편으로는 CHGMOD, CHGPGM, CHGSRVPGM 명령의 LICOPT 매개변수에 지원되지 않는 같은 옵션을 지정하려는 시도가 실패합니다.

모듈 및 ILE 프로그램 사용권 내부 코드 옵션 표시

DSPMOD, DSPPGM 및 DSPSRVPGM 명령은 적용시킨 사용권 내부 코드 옵션을 표시합니다. DSPMOD가 모듈 정보 섹션에 옵션을 표시합니다. 예를 들면, 다음과 같습니다.

```
사용권 내부 코드 옵션  . . . . . :   maf
```

DSPPGM과 DSPSRVPGM은 프로그램 안의 각 개별 모듈에 적용되는 사용권 내부 코드 옵션을 각 모듈을 위한 모듈 속성 섹션에 표시합니다.

같은 사용권 내부 코드 옵션을 여러 번 지정하면 마지막 옵션을 제외한 모든 발생 옵션의 앞에 '+' 기호가 나옵니다. 예를 들어, 모듈 오브젝트에 사용권 내부 코드 옵션을 적용하기 위해 다음 명령을 사용하는 것으로 가정하십시오.

```
> CHGMOD MODULE(TEST) LICOPT('maf, maf,   Maf')
```

이 경우 DSPMOD가 다음과 같이 표시합니다.

```
사용권 내부 코드 옵션  . . . . . :   +maf,+maf,Maf
```

'+'는 사용자가 같은 옵션의 중복 발생 옵션을 지정했음을 의미합니다.

사용권 내부 코드 옵션의 앞에 '*' 기호가 있으면 그 옵션이 모듈이나 ILE 프로그램에 더 이상 적용되지 않습니다. 이것은 오브젝트의 마지막 재작성을 수행한 시스템이 그 옵

션을 지원하지 않기 때문입니다. 자세한 정보는 184 페이지의 『릴리스 호환성』 절을 참조하십시오. 예를 들어, 다음 명령을 사용하여 기본적으로 새로운 옵션을 릴리스 N+1 시스템에 적용시킨 것으로 가정하십시오.

```
> CHGMOD MODULE(TEST) LICOPT('NewOption')
```

모듈이 그 옵션을 지원하지 않는 릴리스 N 시스템으로 되돌아가면 모듈 오브젝트가 다음 명령을 사용하여 재작성됩니다.

```
> CHGMOD MODULE(TEST) FRCCRT(*YES) LICOPT(*SAME)
```

DSPMOD가 표시하는 사용권 내부 코드 옵션은 다음과 같습니다.

```
사용권 내부 코드 옵션  . . . . . :   *NewOption
```

'*'는 옵션이 모듈에 더 이상 적용되지 않는 것을 의미합니다.

제 14 장 공유 기억장치 동기화

공유 기억장치는 동시에 두 개 이상 실행되는 스레드간의 통신을 위한 효과적인 수단을 제공합니다. 이 장에서는 공유 기억장치와 관련된 다양한 문제점을 설명합니다. 주로 공유 기억장치에 액세스할 때 발생할 수 있는 자료 동기화 문제점과 이를 해결하는 방법에 관해 설명합니다.

ILE에 고유한 것인 아니지만 공유 기억장치와 연관된 프로그래밍 문제점은 원래 MI 언어가 아닌 ILE 언어에서 발생하기 쉽습니다. ILE에서 다중 프로그래밍 어플리케이션 프로그래밍 인터페이스를 보다 광범위하게 지원하기 때문입니다.

공유 기억장치

이 설명에 관련하여 *공유 기억장치*란 용어는 둘 이상의 스레드에서 액세스하는 모든 영역 자료를 말합니다. 이 정의는 개별바이트로 직접 액세스할 수 있는 모든 기억장치는 물론 다음과 같은 기억장치 클래스를 포함하고 있습니다.

- MI 영역 오브젝트
- 다른 MI 오브젝트의 1차 연관 영역
- POSIX 공유 메모리 세그먼트
- 내재적 처리 영역: 자동 기억장치, 정적 기억장치 및 활성화 기반 힙 기억장치
- 테라스페이스

동시에 처리할 수 있는 복수 스레드에서 시스템에 액세스할 때 시스템은 이 영역을 상주 기간에 관계 없이 공유 기억장치로 간주합니다.

공유 기억장치 함정

공유 기억장치를 이용하는 어플리케이션을 작성할 경우 *경쟁 조건* 및 *기억장치 액세스 순서지정 문제점*과 같이 예상하지 못한 문제가 발생하지 않도록 방지하십시오.

- 경쟁 조건은 주로 여러 복합 스레드의 상대 시점으로 인해 서로 다른 프로그램 결과가 발생할 수 있을 때 존재합니다.

  경쟁하는 스레드 처리를 예상이 가능한 올바른 방식으로 동기화함으로써 경쟁 조건을 방지할 수 있습니다. 본문에서는 주로 기억장치 동기화에 관해 설명하고 있으나 스레드 실행 동기화 및 기억장치 동기화는 많은 부분에서 그 내용이 중복됩니다. 따라서, 이 장의 뒷부분에 나오는 문제점의 예를 통해 경쟁 조건에 관해 간단히 설명합니다.

- 기억장치 액세스 순서지정 문제점을 기억장치 동기화 또는 메모리 공존 문제점이라고도 합니다. 둘 이상의 복합 스레드가 공유 기억장치 갱신에 있어서 특정 순서에 의존하고 기억장치 액세스에 있어서 각 액세스가 동기화되지 않을 때 이러한 문제가 발생할 수 있습니다. 예를 들어, 하나의 스레드가 두 개의 공유 변수에 값을 저장할 수 있으면 또 다른 스레드가 특정 순서로 그와 같은 갱신 처리를 감시할 때 내재적 종속성을 갖습니다.

  공유 기억장치로부터의 읽기와 공유 기억장치로 쓰기를 실행하는 스레드를 사용하여 시스템이 기억장치 동기화 조치를 수행하도록 함으로써 공유 기억장치 액세스 순서지정 문제점을 방지할 수 있습니다. 이와 같은 일부 조치에 관해서는 다음 주제에서 설명합니다.

공유 기억장치 액세스 순서지정

스레드가 공유 기억장치를 공유할 때 하나의 스레드에서 수행하는 공유 기억장치 액세스(읽기 및 쓰기)를 다른 스레드가 해당하는 특정 순서로 감시한다는 보장은 없습니다. 공유 기억장치로 읽기 또는 쓰기를 실행 중인 스레드에서 수행하는 몇 가지 명시적 기억장치 동기화 양식을 사용하여 이를 방지할 수 있습니다.

둘 이상의 스레드가 공유 기억장치에 대한 동시 액세스를 시도 중이고, 스레드 논리의 구문이 공유 기억장치 액세스에 관한 일부 순서지정을 요구하는 경우 기억장치 동기화가 필요합니다. 공유 기억장치 갱신을 감시하는 순서가 중요하지 않을 경우에는 기억장치 동기화가 필요하지 않습니다. 제공된 스레드는 항상 자체 기억장치 갱신(공유 또는 비공유 기억장치에 대한)을 순서대로 감시합니다. 중복된 공유 기억장치 위치에 액세스 중인 모든 스레드는 같은 순서로 이들 액세스를 감시합니다.

경쟁 조건 및 기억장치 액세스 순서지정 문제점이 어떻게 예상하지 않은 결과를 발생시키는지에 관해 설명하는 다음 예를 참조하십시오.

```
휘발성 정수(int) X = 0;
휘발성 정수(int) Y = 0;
   스레드 A                          스레드 B
 -------------                    -------------
    Y = 1;                           print(X);
    X = 1;                           print(Y);
```

아래의 표는 B에서 인쇄할 수 있는 결과를 요약한 것입니다.

X	Y	문제점 유형	설명
0	0	경쟁 조건	스레드 B가 스레드 A를 수정하기 전에 변수를 읽습니다.
0	1	경쟁 조건	스레드 B가 스레드 A의 갱신을 감시하기 전에 Y에 대한 갱신을 감시하지만 X를 인쇄합니다.
1	1	경쟁 조건	스레드 B가 스레드 A를 갱신한 후 변수를 둘다 읽습니다.

X	Y	문제점 유형	설명
1	0	기억장치 액세스 순서지정	스레드 B가 X에 대한 갱신을 감시하였지만 Y에 대한 스레드 A의 갱신을 감시하지 못했습니다. 명시적 자료 동기화 조치를 사용하지 않고, 이러한 유형의 순서에 무관한 기억장치 액세스가 발생할 수 있습니다.

문제점 1 예: 하나의 writer, 복수의 reader

일반적으로 순서에 무관한 공유 기억장치 액세스의 경우 다중 스레드 프로그램 논리 정확성에 영향을 미치지 않을 수 있습니다. 그러나 스레드가 다른 스레드의 기억장치 갱신을 보는 순서가 프로그램 정확도에 매우 중요한 경우도 있습니다.

일부 명시적 자료 동기화 양식을 요구하는 일반적인 시나리오를 고려하십시오. 이것은 하나의 공유 기억장치의 위치를 사용하여 다른(비중복) 공유 기억장치 위치에 대한 액세스를 제어하는 경우입니다. 예를 들어, 한 스레드가 일부 공유 자료(DATA)를 초기설정하는 것으로 가정하십시오. 또한 그리고 나서 스레드가 공유 자료를 초기설정했음을 다른 모든 스레드에 표시하기 위해 공유 플래그(FLAG)를 설정하는 것으로 가정하십시오.

```
초기설정 중인 스레드              다른 모든 스레드
-------------------              ----------------------------
     DATA = 10                   FLAG의 값이 1이 될 때까지 루프
     FLAG = 1                    DATA 사용
```

이 경우 공유 스레드는 공유 기억장치 액세스에 관한 순서를 지정해야 합니다. 그렇지 않으면 다른 스레드가 초기설정중인 스레드의 공유 기억장치 갱신을 순서에 상관 없이 열람할 수도 있습니다. 이로 인하여 일부 또는 모든 다른 스레드가 DATA에서 초기설정되지 않은 값을 읽게 될 수 있습니다.

예 1 솔루션

위의 예에서 문제점을 해결하기 위한 선호하는 방법은 자료와 플래그 값 사이에 종속 관계를 방지하는 것입니다. 보다 강력한 스레드 동기화 구조를 사용하여 이를 수행할 수 있습니다. 많은 스레드 동기화 기술을 사용할 수 있지만 이러한 문제점에 적합한 것이 세마포어입니다(세마포어 지원은 AS/400 버전 3, 릴리스 2 이후부터 제공되고 있습니다).

다음에 나오는 논리가 적용되도록 하기 위해서는 다음을 가정하십시오.

- 프로그램이 복합 스레드를 시작하기 전에 세마포어를 작성합니다.
- 프로그램이 세마포어를 계수 1로 초기설정합니다.

```
초기설정 중인 스레드              다른 모든 스레드
-------------------              ----------------------------
    DATA = 10                    세마포어 계수가 0이 되도록 대기
    세마포어 감소                DATA 사용
```

기억장치 동기화 조치

공유 기억장치 액세스 순서지정이 필요한 경우 순서지정을 실행하도록 요구하는 모든 스레드가 공유 기억장치 액세스를 동기화하기 위한 명시적 조치를 취해야 합니다. 이러한 조치를 *기억장치 동기화 조치*라고 합니다.

스레드에서 취하는 동기화 조치는 스레드 논리 흐름에서 동기화 조치 앞에 표시된 공유 기억장치 액세스가 동기화 조치 이후 코드의 논리 흐름에 해당 액세스가 표시되기 전에 완료되도록 보장합니다. 이것은 동기화 조치에서 다른 스레드의 관점입니다. 다시 말해서, 스레드가 두 개의 공유 위치에 두 개의 쓰기를 수행하고 동기화 조치가 이들 쓰기를 분리한 경우 시스템이 다음을 구현합니다. 먼저 다음 동기화 조치가 발생하기 전, 두 번째 쓰기를 사용할 수 있게 되는 지점 전에 첫 번째 쓰기를 사용할 수 있도록 보장합니다.

기억장치 동기화 조치가 두 개의 공유 위치에서 두 개의 읽기를 분리하면 두 번째 읽기가 첫 번째 읽기 이후에 값을 읽습니다. 이것은 공유 기억장치에 쓰기를 수행할 때 다른 스레드가 순서지정을 시행하는 경우에만 해당합니다.

다음과 같은 스레드 동기화 조치도 기억장치 동기화 조치입니다.

메커니즘	동기화 조치	**VRM**에 첫 번째로 제공됨
오브젝트 잠금	잠금, 풀기	모두
영역 위치 잠금	잠금, 풀기	모두
Mutex	잠금, 풀기	V3R1M0
세마포어	통지, 대기	V3R2M0
Pthread 조건	대기, 신호, 브로드캐스트	V4R2M0
자료 대기행렬	대기행렬에 넣기, 대기행렬에서 제거하기	모두
메세지 대기행렬	대기행렬에 넣기, 대기행렬에서 제거하기	V3R2M0
비교 및 스왑	목표에 저장	V3R1M0
잠금 값 검사(CHKLKVAL)	목표에 저장	V5R3M0
잠금 값 지우기(CLRLKVAL)	항상	V5R3M0

그리고, 다음 MI 지침은 기억장치 동기화 조치를 구성하지만 동기화 스레드에는 사용할 수 없습니다.

메커니즘	동기화 조치	**VRM**에 첫 번째로 제공됨
SYNCSTG MI 명령어	항상	V4R5M0

둘 이상의 스레드간에 공유 기억장치 액세스 순서지정을 완벽하게 실행하기 위해서는 액세스 순서지정에 종속된 모든 스레드가 적절한 동기화 조치를 사용해야 합니다. 이것은

공유 자료의 reaser 및 writer 모두에 해당됩니다. reader와 writer 사이의 이와 같은 일치는 액세스 순서가 기초를 이루는 기계에서 사용하는 최적화에 의해 변경되지 않도록 보장합니다.

문제점 2 예: 두 개의 경합 writer 또는 reader

추가의 동기화를 요구하는 또 다른 일반적인 문제점은 아래 예에서와 같이 둘 이상의 스레드가 비공식 잠금 프로토콜을 시행하려고 시도하는 것입니다. 이 예에서는 두 개의 스레드가 공유 기억장치에 있는 자료를 처리합니다. 두 스레드가 모두 액세스 일련화를 시도할 때 공유 플래그를 사용하여, 두 개의 공유 자료 항목에 대하여 반복적으로 읽기 및 쓰기를 시도합니다.

```
            스레드 A                                    스레드 B
----------------------------------------    ----------------------------------------
/* Do some work on the shared data */          /* Do some work on the shared data */
for (int i=0; i<10; ++i) {                  for (int i=0; i<10; ++i) {
/* Wait until the locked flag is clear */     /* Wait until the locked flag is clear */
 while (locked == 1) {                        while (locked == 1) {
  sleep(1);                                        sleep(1);
 }                                            }

 locked = 1;  /* Set the lock */              locked = 1;     /* Set the lock */

/* Update the shared data */                 /* Update the shared data */
data1 += 5;                                     data1 += 4;
data2 += 10;                                    data2 += 6;

 locked = 0;  /* Clear the lock */            locked = 0;     /* Clear the lock */
 }                                            }
```

이 예는 공유 메모리 함정을 둘다 설명합니다.

경쟁 조건

여기서 사용되는 잠금 프로토콜이 자료 경쟁 조건을 우회하지 않습니다. 두 작업은 모두 잠겨진 플래그가 지워진 것을 볼 수 있으며, 둘다 자료를 갱신하는 논리를 시작합니다. 이때 읽거나 증가시키거나 기록할 자료 값에 대한 보장은 없습니다(많은 가능한 출력이 있을 수 있습니다).

기억장치 액세스 순서지정 문제점

잠시, 위에서 언급한 경쟁 조건을 무시하십시오. 잠금 및 공유 자료를 갱신하기 위해 두 작업이 사용하는 논리에 내재적 필드 갱신 순서지정에 관한 가정이 들어 있음을 유의하십시오. 특히, 다른 스레드가 자료 변경을 감시하기 전에 잠겨진 플래그가 1로 설정되었음을 감시하는 각각의 스레드 부분에 관한 가정이 있습니다. 추가로, 각각의 스레드가 잠겨진 플래그 값이 0인 것을 감시하기 전에 자료 변경을 감시하는 것으로 가정합니다. 이 설명의 앞부분에서 명시한 것처럼, 이러한 가정은 유효하지 않습니다.

예 2 솔루션

경쟁 조건을 방지하고 기억장치 순서지정을 시행하기 위해 위에 열거된 동기화 메커니즘 중 하나를 사용하여 공유 자료에 대한 액세스를 일련화해야 합니다. 이 예는(복수의

스레드가 공유 자원에 대해 경합하는), 일부 잠금 양식에 적합합니다. 영역 위치 잠금을 채용하는 솔루션 다음에 비교 및 스왑 메커니즘을 채용하는 대체 솔루션을 설명합니다.

스레드 A

```
for (i=0; i<10; ++i) {
  /* Get an exclusive lock on the shared
     data.  We go into a wait state until
     the lock is granted. */
  locksl( LOCK_LOC, _LENR_LOCK );

  /* Update the shared data */
  data1 += 5;
  data2 += 10;

  /* Unlock the shared data */
  unlocksl( LOCK_LOC, _LENR_LOCK );
}
```

스레드 B

```
  for (i=0; i<10; ++i) {
 /* Get an exclusive lock on the shared
    data.  We go into a wait state until
    the lock is granted. */
 locksl( LOCK_LOC, _LENR_LOCK );

    /* Update the shared data */
    data1 += 4;
    data2 += 6;

 /* Unlock the shared data */
 unlocksl( LOCK_LOC, _LENR_LOCK );
}
```

잠금을 사용하여 공유 자료에 대한 액세스를 제한하면 하나의 스레드만 특정 시점에서 자료를 액세스할 수 있도록 보장합니다. 이것은 경쟁 조건을 해결합니다. 이 솔루션은 또한 두 개의 공유 기억장치 위치간에 순서지정 종속 관계가 없으므로 기억장치 액세스 순서지정 문제점을 해결합니다.

대체 솔루션: 잠금 값 검사/잠금 값 지우기 사용

첫 번째 솔루션에서 사용된 것과 같은 공간 위치 잠금은 이러한 간단한 예에서는 필요하지 않습니다. 예를 들어, 공간 위치 잠금은 일정 기간 동안 잠금을 확보할 수 없는 경우 처리를 재개할 수 있도록 해주는 시간종료 값을 지원합니다. 공간 위치 잠금 또한 공유 잠금의 여러 조합을 지원합니다. 이것이 중요한 피처이기는 하지만 일정한 성능 오버헤드를 감수해야 합니다.

대안은 *잠금 값 검사* 및 *잠금 값 지우기*를 사용하는 것입니다. 이 두 MI 명령어는 함께 특히 잠금에 많은 경합이 있지 않은 경우 매우 단순하고 빠른 잠금 프로토콜을 구현하는 방법을 제공합니다.

이 솔루션에서 시스템은 CHKLKVAL을 사용하여 잠금 확보를 시도합니다. 이러한 시도가 실패하는 경우(시스템이 잠금이 이미 사용된 것을 발견하였으므로) 스레드는 잠시 대기한 후 잠금을 확보할 때까지 다시 시도합니다. 공유 자료를 업그레이드한 후 시스템은 CLRLKVAL을 사용하여 잠금을 해제합니다. 이 예에서는 공유 자료 항목에 추가하여 스레드가 또한 8바이트 위치의 주소를 공유하는 것으로 가정합니다. 이 코드는 해당 위치를 변수 LOCK으로서 참조합니다. 추가로, 정적 초기화 또는 이전에 동기화된 초기화를 통해 잠금이 0으로 초기화된 것으로 가정하십시오.

스레드 A

```
/* Do some work on the shared data */
for (i=0; i<10; ++i) {

/* Attempt to acquire the lock using
   CHKLKVAL.  By convention, use value
   1 to indicate locked, 0 to indicate
   unlocked. */
 while (_CHKLKVAL(&LOCK, 0, 1) == 1) {
```

스레드 B

```
  /* Do some work on the shared data */
for (i=0; i<10; ++i) {

   /* Attempt to acquire the lock using
      CHKLKVAL.  By convention, use value
      1 to indicate locked, 0 to indicate
      unlocked. */
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);
   }
```

여기서, 스레드는 잠금 값 검사를 사용하여 경쟁이 없는 테스트와 잠금 변수의 갱신을 수행하고, 잠금 값 지우기를 사용하여 잠금 변수를 잠금 해제 상태로 재설정합니다. 이것은 원래 문제점 단편에서 경험한 경쟁 조건을 해결합니다. 또한 기억장치 액세스 순서지정 문제점을 해결합니다. 앞에서 언급한 것처럼, 이 방식으로 사용될 때 잠금 값 검사와 잠금 값 지우기는 동기화 조치입니다. 잠금 값 검사를 사용하여 공유 자료를 읽기 전에 잠금을 설정하면 스레드가 최근에 갱신된 자료를 읽을 수 있도록 보장합니다. 잠금 값 지우기를 사용하여 공유 자료를 갱신한 후 잠금을 지우면 그 다음 동기화 조치 후에 갱신이 임의의 스레드에 의한 후속 읽기에 사용 가능하게 됩니다.

부록 A. CRTPGM, CRTSRVPGM, UPDPGM, UPDSRVPGM 명령의 출력 리스팅

이 부록에서는 바인더 리스팅의 몇 가지 예를 소개하며 바인더 언어 사용의 결과로 발생할 수 있는 오류에 대해 설명합니다.

바인더 리스팅

프로그램 작성(CRTPGM), 서비스 프로그램 작성(CRTSRVPGM), 프로그램 갱신(UPDPGM), 서비스 프로그램 갱신(UPDSRVPGM) 명령에 대한 바인더 리스팅은 거의 동일합니다. 여기에서는 95 페이지의 『바인더 언어 예』에 나오는 FINANCIAL 서비스 프로그램을 작성하는 데 사용된 CRTSRVPGM 명령의 바인딩 리스팅을 보여줍니다.

CRTPGM, CRTSRVPGM, UPDPGM 또는 UPDSRVPGM 명령의 상세(DETAIL) 매개변수에는 다음의 세 가지 리스팅 유형이 지정될 수 있습니다.

*BASIC
*EXTENDED
*FULL

기본 리스팅

CRTPGM, CRTSRVPGM, UPDPGM 또는 UPDSRVPGM 명령에 DETAIL(*BASIC)을 지정하는 경우 리스팅은 다음으로 구성됩니다.

- CRTPGM, CRTSRVPGM, UPDPGM 또는 UPDSRVPGM 명령에 지정되는 값
- 간단한 요약 표
- 일부 바인딩 처리를 완료하는 데 소요되는 시간을 보여주는 자료

그림 47, 그림 48, 197 페이지의 그림 49에서 이 정보를 볼 수 있습니다.

```
                                    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
```

그림 47. CRTSRVPGM 명령에 지정된 값

```
                                    Create Service Program                      Page     3

간단한 요약 표

 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  * * * * *
```

그림 48. 간단한 요약 표

```
                                   Create Service Program                       Page    23
바인딩 통계

 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  * * * * *
```

그림 49. 바인딩 통계

확장 리스팅

CRTPGM, CRTSRVPGM, UPDPGM 또는 UPDSRVPGM 명령에 DETAIL(*EXTENDED)을 지정하면 리스팅에는 DETAIL(*BASIC)에 의해 제공된 모든 정보에 추가하여 확장 요약 표가 포함됩니다. 확장 요약 표에는 해결된 가져오기(참조)의 수와 처리된 내보내기(정의)의 수가 표시됩니다. 또한 CRTSRVPGM 또는 UPDSRVPGM 명령의 경우에는 리스팅에 사용된 바인더 언어, 생성된 서명 그리고 어떤 가져오기(참조)가 어떤 내보내기(정의)에 대응되었는지도 보여줍니다. 그림 50, 198 페이지의 그림 51, 199 페이지의 그림 52에서는 추가 자료의 예를 보여줍니다.

```
                                   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  * * * * *
```

그림 50. 확장 요약 리스팅

바인더 정보 리스팅

```
 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
```

그림 51. 바인더 정보 리스팅 (1/2)

```
 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
```

그림 51. 바인더 정보 리스팅 (2/2)

```
                                        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: 00000000000000000ADC89D09E0C6E7.

           *****  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  *****
```

그림 52. 바인더 언어 리스팅

전체 리스팅

CRTPGM, CRTSRVPGM, UPDPGM 또는 UPDSRVPGM 명령에 DETAIL(*FULL)을 지정하면 리스팅에는 DETAIL(*EXTENDED)에서 제공하는 모든 정보는 물론 상호 참조 리스팅이 포함됩니다. 200 페이지의 그림 53에서는 제공되는 추가 자료의 부분적인 예를 보여줍니다.

상호 참조 리스팅

Identifier	Defs	Ref	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
CEERLHP	000001A3			*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
CEE4HC	00000197			*SRVPGM	*LIBL	QLEAWI
CEE4RAGE	0000018B			*SRVPGM	*LIBL	QLEAWI
CEE4RIN	00000196			*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
Q LE leDefaultEh	00000180	00000007	0000000E	*SRVPGM	*LIBL	QLEAWI
	00000015		0000001C			
Q LE mhConversionEh	00000181	00000008	0000000F	*SRVPGM	*LIBL	QLEAWI
	00000016		0000001D			
Q LE AG_prod_rc	0000017F	00000004	0000000C	*SRVPGM	*LIBL	QLEAWI
	00000013	0000001A				
Q LE AG_user_rc	0000017E	00000005	0000000D	*SRVPGM	*LIBL	QLEAWI
		00000014	0000001B			
Q LE HdlrRouterEh	0000018F			*SRVPGM	*LIBL	QLEAWI
Q LE RtxRouterCh	0000018E			*SRVPGM	*LIBL	QLEAWI
Rate	0000000B			*MODULE	MYLIB	RATES
Term	0000000A			*MODULE	MYLIB	RATES

그림 53. 상호 참조 리스팅

IPA 리스팅 구성요소

다음 섹션에서는 리스팅의 IPA 구성요소를 설명합니다.

- 오브젝트 파일 맵
- 컴파일러 옵션 맵
- 인라인 보고서
- 글로벌 기호 맵
- 파티션 맵
- 소스 파일 맵
- 메세지
- 메세지 요약

IPA(*YES) 및 DETAIL(*BASIC 또는 *EXTENDED)을 지정하는 경우 CRTPGM 또는 CRTSRVPGM 명령은 인라인 보고서를 제외하고, 이 모든 섹션을 생성합니다. CRTPGM 또는 CRTSRVPGM 명령은 사용자가 IPA(*YES) 및 DETAIL(*FULL)을 지정하는 경우에만 인라인 보고서를 생성합니다.

오브젝트 파일 맵

오브젝트 파일 맵 리스팅 섹션은 IPA에 대한 입력으로서 사용된 오브젝트 파일의 이름을 표시합니다. 소스 파일 맵과 같은 다른 리스팅 섹션은 이 리스팅 섹션에 표시되는 FILE ID 번호를 사용합니다.

컴파일러 옵션 맵

컴파일러 옵션 맵 리스팅 섹션은 처리된 각 컴파일 단위의 IL 자료내에 지정된 컴파일러 옵션을 식별합니다. 각 컴파일 단위의 경우 IPA 처리와 관련된 옵션을 표시합니다. 컴파일러 옵션, #pragma 지시문을 통해, 또는 디폴트 값으로서 이 옵션들을 지정할 수 있습니다.

인라인 보고서

인라인 보고서 리스팅 섹션은 IPA 인라이너에 의해 수행된 조치를 기술합니다. 이 보고서에서 용어 'subprogram'은 C/C++ 기능 또는 C++ 메소드와 같습니다. 요약에는 다음과 같은 정보가 들어 있습니다.

- 정의된 각 서브프로그램의 이름. IPA는 영문자 순서로 서브프로그램 이름을 정렬합니다.
- 서브프로그램에서의 조치 이유:
  - 서브프로그램에 대해 #pragma 인라인을 지정하지 않았습니다.
  - 서브프로그램에 대해 #pragma 인라인을 지정했습니다.
  - IPA가 서브프로그램에서 자동 인라인을 수행했습니다.
  - 서브프로그램을 인라인할 이유가 없었습니다.

  - 파티션 충돌이 있었습니다.
  - IL 자료가 없어서 IPA는 서브프로그램을 인라인할 수 없었습니다.
- 서브프로그램에서의 조치:
  - IPA가 최소한 한번 서브프로그램을 인라인했습니다.
  - 초기 크기 제한사항 때문에 IPA가 서브프로그램을 인라인하지 않았습니다.
  - 크기 제한사항을 넘어선 확장 때문에 IPA가 서브프로그램을 인라인하지 않았습니다.
  - 서브프로그램이 인라인할 후보였지만 IPA가 인라인하지 않았습니다.
  - 서브프로그램이 인라인할 후보였지만 참조되지 않았습니다.
  - 서브프로그램이 직접 순환하거나 일부 호출에서 매개변수가 불일치했습니다.
- 인라인 후에 원래 서브프로그램의 상태:
  - 더 이상 참조되지 않고 정적 내부로서 정의되었으므로 IPA가 서브프로그램을 삭제했습니다.
  - IPA가 다양한 이유로 서브프로그램을 삭제하지 않았습니다.
    - 서브프로그램이 외부적입니다(컴파일 단위 외부에서 호출될 수 있습니다).
    - 이 서브프로그램에 대한 서브프로그램 호출이 남아 있습니다.
    - 서브프로그램이 획득한 주소를 갖습니다.
- 서브프로그램의 초기 상대 크기(추상 코드 단위).
- 인라인 후의 서브프로그램의 최종 상대 크기(추상 코드 단위).
- 서브프로그램내의 호출 수와 IPA가 서브프로그램으로 인라인된 이 호출의 수.
- 서브프로그램이 컴파일 단위에서 다른 서브프로그램에 의해 호출되는 횟수와 IPA가 서브프로그램을 인라인한 횟수.
- 선택된 모드와 지정된 임계값 및 한계값. 대체적으로 어플리케이션 내에서 이름이 고유하지 않을 수 있는 정적 기능은 `@nnn@`이나 `nnn` 접두부가 오는 이름을 사용하며 `XXXX`는 파티션 이름, `nnn`은 소스 파일 번호입니다.

자세한 호출 구조에는 다음과 같은 각 서브프로그램의 특정 정보가 들어 있습니다.

- 각 서브프로그램이 호출하는 서브프로그램.
- 각 서브프로그램을 호출한 서브프로그램.
- 각 서브프로그램이 인라인된 서브프로그램.

선택 모드에서 인라이너를 사용하려는 경우 정보는 더 나은 프로그램 분석을 허용할 수 있습니다. 이 보고서에 있는 계수는 비IPA에서 IPA 프로그램으로의 호출을 포함하지 않습니다.

글로벌 기호 맵

글로벌 기호 맵 나열 섹션은 최적화 프로세스를 병합시키는 글로벌 변수에 의해 글로벌 자료 구조의 멤버로 맵되는 방식을 보여줍니다. 기호 정보 및 파일 이름 정보를 포함합니다(파일 이름 정보는 대략적입니다). 그리고 행 번호 정보를 사용할 수 있습니다.

파티션 맵

파티션 맵 리스팅 섹션에서는 IPA가 작성한 각 오브젝트 코드 파티션을 설명합니다. 다음 정보를 제공합니다.

- 각 파티션을 생성하는 이유.
- 오브젝트 코드를 생성하는 데 사용된 옵션.
- 파티션에 포함된 기능 및 글로벌 자료.
- 파티션을 작성하기 위해 사용된 소스 파일.

소스 파일 맵

소스 파일 리스팅 섹션은 오브젝트 파일에 포함된 소스 파일을 식별합니다.

메세지

IPA가 오류나 오류 가능성을 감지한 경우 하나 이상의 진단 메세지를 발행하며 메세지 리스팅 섹션을 생성합니다. 이 리스팅 섹션에는 IPA 처리 중 발행된 메세지의 요약이 들어 있습니다. 메세지는 심각도에 의해 정렬됩니다. 메세지 리스팅 섹션은 각 메세지가 원래 표시된 리스팅 페이지 번호를 표시합니다. 또한 메세지 텍스트를 표시하며 선택적으로 파일 이름, 행(알려진 경우) 및 열(알려진 경우)에 관련된 정보를 표시합니다.

메세지 요약

메세지 요약 리스팅 섹션은 총 메세지 수와 각 심각도 레벨의 메세지 수를 표시합니다.

서비스 프로그램 예에 대한 리스팅

197 페이지의 그림 49, 198 페이지의 그림 51, 200 페이지의 그림 53은 101 페이지의 그림 36에서 FINANCIAL 서비스 프로그램을 작성하기 위해 DETAIL(*FULL)이 지정될 때 생성된 리스팅 자료의 일부를 표시한 것입니다. 이들 그림에서는 바인딩 통계, 바인더 정보 리스팅, 상호 참조 리스팅을 보여줍니다.

서비스 프로그램 예에 대한 바인더 정보 리스팅

바인더 정보 리스팅(198 페이지의 그림 51)에는 다음 자료와 열 표제가 들어 있습니다.

- 처리된 모듈이나 서비스 프로그램의 이름 및 라이브러리

  만일 *바인드* 필드 값이 모듈 오브젝트에 대해 *YES 값을 표시하면 모듈이 복사에 의해 바인드되도록 표시됩니다. *바인드* 필드 값이 서비스 프로그램에 대해 *YES 값을 표시하면 서비스 프로그램이 참조에 의해 바인드됩니다. 모듈 오브젝트나 서비스

프로그램에 대한 *바인드* 필드에 *NO 값이 표시되어 있으면 해당 오브젝트는 바인드에 포함되지 않습니다. 이것은 오브젝트가 미해결 가져오기를 충족시키는 내보내기를 제공하지 않기 때문입니다.

- Number

  처리된 각 모듈이나 서비스 프로그램의 경우 각각의 내보내기(정의)나 가져오기(참조)에는 연관된 고유 ID가 있습니다.

- Symbol

  이 열은 기호명을 내보내기(Def)나 가져오기(Ref)로 식별합니다.

- Ref

  이 열(Ref)에 지정된 번호는 가져오기 요구를 충족시키는 내보내기(Def)의 고유 ID입니다. 예를 들어, 198 페이지의 그림 51에서 가져오기 00000005를 위한 고유 ID는 내보내기 0000017E를 위한 고유 ID에 대응합니다.

- Identifier

  이것은 내보내기 또는 가져오기된 기호명입니다. 고유 ID 00000005의 가져오기 기호명은 Q LE AG_user_rc입니다. 또한 고유 ID 0000017E의 내보내기 기호명도 Q LE AG_user_rc입니다.

- Type

  기호명이 프로시듀어인 경우 기호는 Proc로 식별됩니다. 기호명이 자료 항목인 경우에는 Data로 식별됩니다.

- Scope

  모듈의 경우 이 열은 내보내기된 기호명이 모듈 레벨 또는 공용 인터페이스에서 서비스 프로그램으로 액세스되는지를 식별합니다. 프로그램이 작성중인 경우 내보내기된 기호명은 모듈 레벨에서만 액세스될 수 있습니다. 서비스 프로그램이 작성중인 경우 내보내기 기호명은 모듈 레벨 또는 서비스 프로그램(SrvPgm) 레벨에서 액세스될 수 있습니다. 내보내기된 기호가 공용 인터페이스의 일부이면 *범위* 열에 있는 값이 반드시 SrvPgm이어야 합니다.

- Export

  이 열은 모듈이나 서비스 프로그램에서 내보내기된 자료 항목의 강도를 식별합니다.

- Key

  이 열에는 약 반출에 대한 추가 정보가 포함됩니다. 일반적으로, 이 열은 공백입니다.

서비스 프로그램 예에 대한 상호 참조 리스팅

200 페이지의 그림 53에 있는 상호 참조 리스팅은 바인더 정보에 표시된 자료를 보는 다른 방법입니다. 상호 참조 리스팅에는 다음 열 표제가 포함됩니다.

- Identifier

  기호 해결시 처리된 내보내기의 이름

- Defs

  각 내보내기와 연관된 고유 ID

- Refs

  이 열의 번호는 해당 내보내기(정의)로 대응된 가져오기(참조)의 고유 ID를 나타냅니다.

- Type

  내보내기가 *MODULE 오브젝트 또는 *SRVPGM 오브젝트에서 왔는지를 식별합니다.

- Library

  명령이나 바인딩 디렉토리에서 지정된 라이브러리명

- Object

  내보내기(정의)를 제공한 오브젝트명

서비스 프로그램 예를 위한 바인딩 통계

197 페이지의 그림 49에는 서비스 프로그램 FINANCIAL 작성에 대한 통계 세트가 표시되어 있습니다. 이 통계는 바인더가 작성 요구를 처리하면서 어디에 시간을 소요했는지를 식별합니다. 이 섹션에서 표시된 자료에 대해 사용자는 간접 제어만 갖습니다. 처리 오버헤드 양을 측정할 수 없습니다. 따라서, *Total CPU time* 필드에 나오는 값은 앞 필드에 나오는 시간 합계보다 더 큽니다.

바인더 언어 오류

서비스 프로그램을 작성중에 시스템이 바인더 언어를 처리하는 동안 오류가 발생할 수 있습니다. 서비스 프로그램 작성(CRTSRVPGM) 명령에 DETAIL(*EXTENDED) 또는 DETAIL(*FULL)을 지정하면 스풀 파일에서 오류를 볼 수 있습니다.

다음과 같은 정보 메세지가 발생할 수 있습니다.

- 서명이 채워짐
- 서명이 잘림

다음과 같은 경고 오류가 발생할 수 있습니다.

- 현재 내보내기 블록이 인터페이스를 제한함
- 내보내기 블록이 중복됨
- 이전 내보내기시 기호가 중복됨
- 레벨 검사를 두 번 이상 작동 불가능하게 만들 수 없음, 무시됨
- 복수 현재 내보내기 블록은 허용되지 않음, 이전의 것이 가정됨

다음과 같은 심각한 오류가 발생할 수 있습니다.

- 현재 내보내기 블록이 비어 있음

- 내보내기 블록이 완료되지 않음, 파일 끝이 ENDPGMEXP 이전에 발견됨
- 내보내기 블록이 시작되지 않음, STRPGMEXP가 필요함
- 내보내기 블록이 내포될 수 없음, ENDPGMEXP가 누락됨
- 내보내기가 내보내기 블록내에 존재해야 함
- 서로 다른 내보내기 블록에 대한 동일한 서명. 내보내기를 변경해야 함
- 여러 개의 와일드카드가 일치함
- 현재 내보내기 블록이 없음
- 와일드카드가 일치하지 않음
- 이전 내보내기 블록이 비어 있음
- 서명에 변동 문자가 포함됨
- LVLCHK(*NO)에 SIGNATURE(*GEN)가 필요함
- 서명 구문이 유효하지 않음
- 기호명이 필요함
- 기호가 서비스 프로그램 내보내기에 허용되지 않음
- 기호가 정의되지 않음
- 구문이 유효하지 않음

서명이 채워짐

207 페이지의 그림 55에서는 이 메세지를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                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  * * * * *
```

그림 54. 제공된 서명이 16바이트보다 짧기 때문에 채워짐

이는 정보 메세지입니다.

제안되는 변경

변경이 필요 없습니다.

메세지를 받지 않으려면 제공되는 서명의 길이가 정확히 16바이트가 되도록 해야 합니다.

서명이 잘림

그림 55에서는 이 메세지를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                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  * * * * *
```

그림 55. 제공된 자료의 처음 16바이트만 서명으로 사용됨

이는 정보 메세지입니다.

제안되는 변경

변경이 필요 없습니다.

메세지를 받지 않으려면 제공되는 서명의 길이가 정확히 16바이트가 되도록 해야 합니다.

현재 내보내기 블록이 인터페이스를 제한함

그림 56에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                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  * * * * *
```

*그림 56. PGMLVL(*PRV)이 PGMLVL(*CURRENT)보다 많은 기호를 내보내기함*

이는 경고 오류입니다.

PGMLVL(*PRV) 내보내기 블록이 PGMLVL(*CURRENT) 내보내기 블록보다 많은 기호를 지정했습니다.

다른 오류가 발생하지 않으면 서비스 프로그램이 작성됩니다.

다음 두 가지 사항에 모두 해당하는 경우:

- PGMLVL(*PRV)이 C로 명명된 프로시듀어를 지원합니다.
- 새로운 서비스 프로그램하에서 프로시듀어 C가 더 이상 지원되지 않습니다.

해당 서비스 프로그램에서 프로시듀어 C를 호출한 ILE 프로그램이나 서비스 프로그램을 실행시킬 때 오류가 발생합니다.

제안되는 변경

1. PGMLVL(*CURRENT) 내보내기 블록에 PGMLVL(*PRV) 내보내기 블록보다 내보내기될 기호가 더 많이 있는지 확인하십시오.
2. CRTSRVPGM 명령을 다시 실행하십시오.

이 예에서는 EXPORT SYMBOL(C)이 PGMLVL(*CURRENT) 블록에 추가되는 대신 STRPGMEXP PGMLVL(*PRV) 블록에 잘못 추가되었습니다.

내보내기 블록이 중복됨

그림 57에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                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  *****
```

그림 57. STRPGMEXP/ENDPGMEXP 블록을 중복시킴

이는 경고 오류입니다.

둘 이상의 STRPGMEXP 및 ENDPGMEXP 블록이 동일한 모든 기호를 완전히 같은 순서로 내보내기했습니다.

다른 오류가 발생하지 않으면 서비스 프로그램이 작성됩니다. 중복된 서명은 작성되는 서비스 프로그램에 한 번만 포함됩니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - PGMLVL(*CURRENT) 내보내기 블록이 올바른지 확인하십시오. 이것을 적절하게 갱신하십시오.
   - 중복 내보내기 블록을 제거하십시오.
2. CRTSRVPGM 명령을 다시 실행하십시오.

이 예에서는 PGMLVL(*CURRENT)이 지정된 STRPGMEXP 명령이 EXPORT SYMBOL(B) 뒤에 다음의 소스 행을 추가해야 합니다.

```
EXPORT SYMBOL(C)
```

이전 내보내기시 기호가 중복됨

그림 58에서는 중복 기호 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                        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  * * * * *
```

그림 58. 내보내기된 기호를 중복시킴

이는 경고 오류입니다.

서비스 프로그램에서 내보내기된 기호가 STRPGMEXP 및 ENDPGMEXP 블록에서 두 번 이상 지정되었습니다.

다른 오류가 발생하지 않으면 서비스 프로그램이 작성됩니다. 첫 번째 중복 기호만 서비스 프로그램에서 내보내기됩니다. 중복 기호 모두가 생성되는 서명에 영향을 미칩니다.

제안되는 변경

1. 바인더 언어 소스 파일에서 중복 소스 행 중 하나를 제거하십시오.
2. CRTSRVPGM 명령을 다시 실행하십시오.

이 예에서는 두 번째 EXPORT SYMBOL(A)을 제거하십시오.

레벨 검사를 두 번 이상 작동 불가능하게 만들 수 없음, 무시됨

그림 59에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                Binder Language Listing

 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: 000000000000000000000000000000C1.

            * * * * *  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  * * * * *
```

*그림 59. 복수 STRPGMEXP 명령이 LVLCHK(*NO)를 지정함*

이는 경고 오류입니다.

둘 이상의 STRPGMEXP 블록이 LVLCHK(*NO)를 지정했습니다.

다른 오류가 발생하지 않으면 서비스 프로그램이 작성됩니다. 두 번째 LVLCHK(*NO)부터 LVLCHK(*YES)로 가정됩니다.

제안되는 변경

1. LVLCHK(*NO)가 하나의 STRPGMEXP 블록에만 지정되었는지 확인하십시오.
2. CRTSRVPGM 명령을 다시 실행하십시오.

이 예에서는 PGMLVL(*PRV) 내보내기 블록만 LVLCHK(*NO)를 지정한 유일한 내보내기 블록입니다. PGMLVL(*CURRENT) 내보내기 블록에서 LVLCHK(*NO) 값이 제거됩니다.

복수의 현재 내보내기 블록이 허용되지 않음, 이전의 것이 가정됨

211 페이지의 그림 60에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
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.

           *****  END OF BINDER LANGUAGE LISTING  *****
```

*그림 60. 둘 이상의 PGMLVL(*CURRENT) 값이 지정됨*

이는 경고 오류입니다.

둘 이상의 STRPGMEXP 명령에 PGMLVL(*CURRENT) 값이 지정되거나 PGMLVL(*CURRENT)를 디폴트로 취하도록 허용되었습니다. 값 PGMLVL (*CURRENT)의 값이 있는 두 번째 내보내기 블록부터 PGMLVL(*PRV)로 간주됩니다.

다른 오류가 발생하지 않으면 서비스 프로그램이 작성됩니다.

제안되는 변경

1. STRPGMEXP PGMLVL(*PRV)에 대한 적절한 소스 텍스트로 변경하십시오.
2. CRTSRVPGM 명령을 다시 실행하십시오.

이 예에서는 두 번째 STRPGMEXP가 변경됩니다.

현재 내보내기 블록이 비어 있음

212 페이지의 그림 61에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                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  * * * * *
```

*그림 61. STRPGMEXP PGMLVL(*CURRENT) 블록에서 내보내기시킬 기호가 없음*

이는 심각한 오류입니다.

*CURRENT 내보내기 블록에서 내보내기시키기 위해 식별된 기호가 없습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - 내보내기될 기호명을 추가하십시오.
   - 비어 있는 STRPGMEXP-ENDPGMEXP 블록을 제거하고, 다른 STRPGMEXP-ENDPGMEXP 블록을 PGMLVL(*CURRENT)로 만드십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이 예에서는 STRPGMEXP 및 ENDPGMEXP 명령 사이의 바인더 언어 소스 파일에 다음 소스 행이 추가됩니다.

```
EXPORT SYMBOL(A)
```

내보내기 블록이 완료되지 않음, 파일 끝이 ENDPGMEXP 전에 발견됨

그림 62에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                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  * * * * *
```

그림 62. ENDPGMEXP 명령을 찾지 못했으나 소스 파일 끝이 발견됨

이는 심각한 오류입니다.

파일 끝에 도달하기 전에 ENDPGMEXP를 발견하지 못했습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - 적절한 위치에 ENDPGMEXP 명령을 추가하십시오.
   - ENDPGMEXP 명령과 일치하지 않는 STRPGMEXP 명령을 제거하고, 내보내기될 기호명을 제거하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이 예에서는 STRPGMEXP 명령 다음에 다음 행이 추가됩니다.

```
   EXPORT SYMBOL(A)
ENDPGMEXP
```

내보내기 블록이 시작되지 않음, STRPGMEXP가 필요함

그림 63에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                              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  * * * * *
```

그림 63. STRPGMEXP 명령이 누락됨

이는 심각한 오류입니다.

ENDPGMEXP 명령 전에 STRPGMEXP 명령이 발견되지 않았습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - STRPGMEXP 명령을 추가하십시오.
   - 내보내기되는 기호와 ENDPGMEXP 명령을 제거하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이 예에서는 ENDPGMEXP 명령 이전의 바인더 언어 소스 파일에 다음 두 개의 소스 행이 추가됩니다.

```
STRPGMEXP
   EXPORT SYMBOL(A)
```

내보내기 블록이 내포될 수 없음, ENDPGMEXP가 누락됨

그림 64에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                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  * * * * *
```

그림 64. ENDPGMEXP 명령이 누락됨

이는 심각한 오류입니다.

다른 STRPGMEXP 명령 이전에 ENDPGMEXP 명령이 발견되지 않았습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - 다음 STRPGMEXP 명령 앞에 ENDPGMEXP 명령을 추가하십시오.
   - STRPGMEXP 명령과 내보내기될 기호명을 제거하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이 예에서는 두 번째 STRPGMEXP 명령 앞의 바인더 소스 파일에 ENDPGMEXP 명령이 추가됩니다.

내보내기가 내보내기 블록내에 존재해야 함

215 페이지의 그림 65에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
STRPGMEXP PGMLVL(*CURRENT)
   EXPORT SYMBOL(A)
   EXPORT SYMBOL(B)
ENDPGMEXP
 ******** Export signature: 00000000000000000000000000000CD2.
   EXPORT SYMBOL(A)
 ***ERROR Exports must exist inside export blocks.

           *****  END OF BINDER LANGUAGE LISTING  *****
```

그림 65. 내보내기될 기호명이 STRPGMEXP-ENDPGMEXP 블록 외부에 있음

이는 심각한 오류입니다.

내보내기될 기호가 STRPGMEXP-ENDPGMEXP 블록내에 정의되어 있지 않습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - 내보내기낼 기호를 이동시키십시오. STRPGMEXP-ENDPGMEXP 블록내에 기호를 위치시키십시오.
   - 기호를 제거하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이 예에서는 오류 상태의 소스 행이 바인더 언어 소스 파일에서 제거됩니다.

서로 다른 내보내기 블록에 대한 동일한 서명, 내보내기를 변경해야 함

이는 심각한 오류입니다.

서로 다른 기호를 내보내기한 STRPGMEXP-ENDPGMEXP 블록에서 동일한 서명이 생성되었습니다. 이러한 오류 상태는 거의 발생하지 않습니다. 내보내기될 기호가 매우 중요할 경우 이 오류는 매 3.4E28 시도마다 한 번씩만 발생해야 합니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - PGMLVL(*CURRENT) 블록에서 내보내기시킬 기호를 추가하십시오.

이미 내보내기시킨 기호를 지정하는 것이 좋은 방법입니다. 이렇게 하면 중복 기호에 대한 경고 오류가 발생할 수 있으나 서명을 고유하게 만드는 데 도움이 됩니다. 대체 방법으로는 아직 내보내기되지 않은 다른 기호가 내보내기되도록 추가하는 것입니다.

- 모듈에서 내보내기될 기호명을 변경하고, 바인더 언어 소스 파일에 상응하는 변경을 수행하십시오.
- 프로그램 내보내기 시작(STRPGMEXP) 명령의 SIGNATURE 매개변수를 사용하여 서명을 지정하십시오.

2. CRTSRVPGM 명령을 실행하십시오.

다수의 와일드카드가 일치함

그림 66에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                        Binder Language Listing

STRPGMEXP PGMLVL(*CURRENT)
EXPORT ("A"<<<)
***ERROR Multiple 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
```

그림 66. 와일드카드 스펙의 복수 일치

이는 심각한 오류입니다.

내보내기를 위해 지정된 와일드카드가 내보내기에 사용할 수 있는 둘 이상의 기호와 일치했습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 원하는 대응 내보내기가 유일한 대응 내보내기가 되도록 상세하게 와일드카드를 지정하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

현재 내보내기 블록이 없음

217 페이지의 그림 67에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                Binder Language Listing

 STRPGMEXP  PGMLVL(*PRV)
   EXPORT SYMBOL(A)
ENDPGMEXP
 ******** Export signature: 0000000000000000000000000000000C1.
 ***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  * * * * *
```

*그림 67. PGMLVL(*CURRENT) 내보내기 블록이 없음*

이는 심각한 오류입니다.

바인더 언어 소스 파일에 STRPGMEXP PGMLVL(*CURRENT)이 없습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - PGMLVL(*PRV)을 PGMLVL(*CURRENT)로 변경하십시오.
   - 올바른 *CURRENT 내보내기 블록인 STRPGMEXP-ENDPGMEXP 블록을 추가하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이 예에서는 PGMLVL(*PRV)이 PGMLVL(*CURRENT)로 변경됩니다.

와일드카드가 일치하지 않음

그림 68에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                Binder Language Listing

STRPGMEXP PGMLVL(*CURRENT)
EXPORT ("Z"<<<)
***ERROR No 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
```

그림 68. 와일드카드 스펙이 일치하지 않음

이는 심각한 오류입니다.

내보내기를 위해 지정된 와일드카드가 내보내기에 사용할 수 있는 어떤 기호와도 일치하지 않았습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 내보내기를 위해 원하는 기호와 일치하는 와일드카드를 지정하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이전 내보내기 블록이 비어 있음

그림 69에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                        Binder Language Listing

STRPGMEXP PGMLVL(*CURRENT)
   EXPORT SYMBOL(A)
   EXPORT SYMBOL(B)
ENDPGMEXP
 ******** Export signature: 0000000000000000000000000000CD2.
 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  * * * * *
```

*그림 69. PGMLVL(*CURRENT) 내보내기 블록이 없음*

이는 심각한 오류입니다.

STRPGMEXP PGMLVL(*PRV)이 있으나 기호가 지정되지 않았습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - 비어 있는 STRPGMEXP-ENDPGMEXP 블록에 기호를 추가하십시오.
   - 비어 있는 STRPGMEXP-ENDPGMEXP 블록을 제거하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이 예에서는 비어 있는 STRPGMEXP-ENDPGMEXP 블록이 바인더 언어 소스 파일에서 제거됩니다.

서명에 변동 문자가 포함됨

그림 70에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                Binder Language Listing

STRPGMEXP  SIGNATURE('\!cdefghijklmnop')
 ***ERROR Signature contains variant characters
   EXPORT     SYMBOL('Proc_2')
ENDPGMEXP

 ******** Export signature: E05A838485868788899192939495969 7.

            * * * * *  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  * * * * *
```

그림 70. 서명에 변동 문자가 포함됨

이는 심각한 오류입니다.

모든 코드화 문자 세트 ID(CCSID)에 없는 문자가 서명에 포함되어 있습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 변동 문자를 제거하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이와 같이 특별한 경우에는 \!를 제거해야 합니다.

SIGNATURE(*GEN)에 LVLCHK(*NO)가 필요함

그림 71에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                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  * * * * *
```

*그림 71. LVLCHK(*NO)가 지정되면 명시적 서명이 유효하지 않음*

이는 심각한 오류입니다.

LVLCHK(*NO)가 지정되는 경우에는 SIGNATURE(*GEN)가 필요합니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - SIGNATURE(*GEN)를 지정하십시오.
   - LVLCHK(*YES)를 지정하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

서명 구문이 유효하지 않음

그림 72에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                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  * * * * *
```

그림 72. 서명 값으로 지정된 것이 유효하지 않음

이는 심각한 오류입니다.

서명에는 유효하지 않은 문자가 들어 있습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 서명 값에서 유효하지 않은 문자를 제거하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이 경우 서명 필드에서 " 문자를 제거하십시오.

기호명이 필요함

221 페이지의 그림 73에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                Binder Language Listing

 STRPGMEXP PGMLVL(*CURRENT)
   EXPORT SYMBOL(A)
   EXPORT SYMBOL(')
 ***ERROR Symbol name required.
ENDPGMEXP
 ******** Export signature: 0000000000000000000000000000000C1.

           * * * * *  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  * * * * *
```

그림 73. 기호가 내보내기되지 않음

이는 심각한 오류입니다.

서비스 프로그램에서 내보내기할 기호명이 없습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - 오류가 있는 행을 바인더 언어 소스 파일에서 제거하십시오.
   - 서비스 프로그램에서 내보내기될 기호명을 추가하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이 예에서는 소스 행 EXPORT SYMBOL("")이 바인더 언어 소스 파일에서 제거됩니다.

기호가 서비스 프로그램 내보내기에 허용되지 않음

그림 74에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                Binder Language Listing

STRPGMEXP PGMLVL(*CURRENT)
   EXPORT SYMBOL(A)
 ***ERROR Symbol not allowed as service program export.
   EXPORT SYMBOL(D)
ENDPGMEXP
 ******** Export signature: 0000000000000000000000000000CD4.

           * * * * *  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  * * * * *
```

그림 74. 기호명이 서비스 프로그램에서의 내보내기에 유효하지 않음

이는 심각한 오류입니다.

서비스 프로그램에서 내보내기될 기호가 복사에 의해 바인드될 모듈 중 하나에서 내보내기되지 않았습니다. 일반적으로, 서비스 프로그램에서 내보내기되도록 지정되는 기호는 실제로 서비스 프로그램에 의해 가져오기되어야 하는 기호입니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - 바인더 언어 소스 파일에서 오류가 있는 기호를 제거하십시오.
   - CRTSRVPGM 명령의 MODULE 매개변수에 내보내기하려는 기호가 들어 있는 모듈을 지정하십시오.
   - 복사에 의해 바인드될 모듈 중 하나에 기호를 추가하고, 모듈 오브젝트를 재작성하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

이 예에서는 소스 행 EXPORT SYMBOL(A)이 바인더 언어 소스 파일에서 제거됩니다.

기호가 정의되지 않음

그림 75에서는 이 오류를 포함한 바인더 언어 리스팅을 보여줍니다.

```
                                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  *****
```

그림 75. 복사에 의해 바인드될 모듈에 기호가 없음

이는 심각한 오류입니다.

서비스 프로그램에서 내보내기될 기호가 복사에 의해 바인드될 모듈에 없습니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 다음 변경 중 하나를 수행하십시오.
   - 바인더 언어 소스 파일에서 정의되지 않은 기호를 제거하십시오.

- CRTSRVPGM 명령의 MODULE 매개변수에 내보내기하려는 기호가 들어 있는 모듈을 지정하십시오.
- 복사에 의해 바인드될 모듈 중 하나에 기호를 추가하고, 모듈 오브젝트를 재작성하십시오.

2. CRTSRVPGM 명령을 실행하십시오.

이 예에서는 소스 행 EXPORT SYMBOL(Q)이 바인더 언어 소스 파일에서 제거됩니다.

구문이 유효하지 않음

이는 심각한 오류입니다.

소스 멤버의 명령문이 유효한 바인더 언어 명령문이 아닙니다.

서비스 프로그램이 작성되지 않습니다.

제안되는 변경

1. 유효한 바인더 언어 명령문을 포함하도록 소스 멤버를 정정하십시오.
2. CRTSRVPGM 명령을 실행하십시오.

부록 B. 최적화 프로그램의 예외

드문 경우지만 최적화 레벨 30(*FULL) 또는 40으로 컴파일된 프로그램에서 MCH3601 예외 메세지가 발생할 수 있습니다. 이 부록에서는 이 메세지가 발생하는 한 가지 예에 대해 설명합니다. 최적화 레벨 10(*NONE) 또는 20(*BASIC)으로 컴파일되면 동일한 프로그램이라도 MCH3601 예외 메세지를 수신하지 않습니다. 이 예에서 메세지의 발생 여부는 사용자의 ILE HLL 컴파일러가 배열에 대해 기억장치를 할당하는 방법에 따라 달라집니다. 이 예는 사용자의 언어에 대해 이 경우가 절대로 발생하지 않습니다.

사용자가 최적화 레벨 30(*FULL) 또는 40을 요청하면 ILE는 루프 밖에 있는 배열 색인 참조를 계산하여 성능을 향상시키려고 합니다. 즉, 루프에서 배열을 참조할 때 모든 요소를 순서대로 액세스하게 됩니다. 이전 루프 반복에서 마지막 배열 요소의 주소를 저장하여 성능을 향상시킬 수 있습니다. 이와 같은 성능상의 향상을 얻기 위해 ILE는 루프의 밖에서 첫 번째 배열 요소 주소를 계산하고 값을 루프 내부에서 사용하기 위해 저장합니다.

다음 예를 참조하십시오.

```
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;
```

ARR[init_expression]에 대한 참조로 인해 틀린 배열 색인이 생성될 경우에는 다음의 예가 MCH3601 예외에 해당될 수 있습니다. 이것은 WHILE 루프를 입력하기 전에 ILE가 처음 배열 요소 주소를 계산하려고 시도했기 때문입니다.

최적화 레벨 30(*FULL) 또는 40에서 MCH3601 예외를 수신하게 되면 다음 상황을 살펴보십시오.

1. 변수를 배열 요소 색인으로 사용하기 전에 변수를 증분시키는 루프가 사용자에게 있습니다.
2. 루프에 들어갈 때 색인 변수의 초기 값이 음수입니다.
3. 변수의 초기 값을 사용하는 배열에 대한 참조가 유효하지 않습니다.

이들 조건이 존재하는 경우 다음과 같이 하여 최적화 레벨 30(*FULL) 또는 40이 계속 사용되도록 할 수 있습니다.

1. 변수 증분 프로그램의 부분을 루프의 맨 아래로 이동시키십시오.
2. 필요하면 변수에 대한 참조를 변경하십시오.

앞의 예가 다음과 같이 변경됩니다.

```
I = init_expression + 1;

WHILE ( I < limit_expression + 1 )

  ARR[I] = some_expression;

  I = I + 1;

END;
```

이러한 변경이 불가능하면 최적화 레벨을 30(*FULL)이나 40에서 20(*BASIC)이나 10(*NONE)으로 줄이십시오.

부록 C. ILE 오브젝트와 함께 사용되는 CL 명령

다음 표에서는 각 ILE 오브젝트와 함께 사용되는 CL 명령을 보여줍니다.

모듈과 함께 사용되는 CL 명령

표 13. 모듈과 함께 사용되는 CL 명령

명령	설명하는 이름
CHGMOD	모듈 변경
CRTCBLMOD	COBOL 모듈 작성
CRTCLMOD	CL 모듈 작성
CRTCMOD	C 모듈 작성
CRTCPPMOD	C++ 모듈 작성
CRTRPGMOD	RPG 모듈 작성
DLTMOD	모듈 삭제
DSPMOD	모듈 표시
RTVBNDSRC	바인더 소스 검색
WRKMOD	모듈에 대한 작업

프로그램 오브젝트와 함께 사용되는 CL 명령

표 14. 프로그램 오브젝트와 함께 사용되는 CL 명령

명령	설명하는 이름
CHGPGM	프로그램 변경
CRTBNDC	바인드 C 프로그램 작성
CRTBNDCBL	바인드 COBOL 프로그램 작성
CRTBNDCL	바인드 CL 프로그램 작성
CRTBNDCPP	바인드 C++ 프로그램 작성
CRTBNDRPG	바인드 RPG 프로그램 작성
CRTPGM	프로그램 작성
DLTPGM	프로그램 삭제
DSPPGM	프로그램 표시
DSPPGMREF	프로그램 참조 표시
UPDPGM	프로그램 갱신
WRKPGM	프로그램에 대한 작업

서비스 프로그램과 함께 사용되는 CL 명령

표 15. 서비스 프로그램과 함께 사용되는 CL 명령

명령	설명하는 이름
CHGSRVPGM	서비스 프로그램 변경
CRTSRVPGM	서비스 프로그램 작성
DLTSRVPGM	서비스 프로그램 삭제
DSPSRVPGM	서비스 프로그램 표시
RTVBNDSRC	바인더 소스 검색
UPDSRVPGM	서비스 프로그램 갱신
WRKSRVPGM	서비스 프로그램에 대한 작업

바인딩 디렉토리와 함께 사용되는 CL 명령

표 16. 바인딩 디렉토리와 함께 사용되는 CL 명령

명령	설명하는 이름
ADDBNDDIRE	바인딩 디렉토리 항목 추가
CRTBNDDIR	바인딩 디렉토리 작성
DLTBNDDIR	바인딩 디렉토리 삭제
DSPBNDDIR	바인딩 디렉토리 표시
RMVBNDDIRE	바인딩 디렉토리 항목 제거
WRKBNDDIR	바인딩 디렉토리에 대한 작업
WRKBNDDIRE	바인딩 디렉토리 항목에 대한 작업

SQL(구조화 조회 언어)와 함께 사용되는 CL 명령

표 17. SQL(구조화 조회 언어)와 함께 사용되는 CL 명령

명령	설명하는 이름
CRTSQLCI	SQL(구조화 조회 언어) ILE C 오브젝트 작성
CRTSQLCBLI	SQL(구조화 조회 언어) ILE COBOL 오브젝트 작성
CRTSQLRPGI	SQL(구조화 조회 언어) ILE RPG 오브젝트 작성

CICS®와 함께 사용되는 CL 명령

표 18. CICS와 함께 사용되는 CL 명령

명령	설명하는 이름
CRTCICSC	CICS ILE C 오브젝트 작성
CRTCICSCBL	CICS COBOL 프로그램 작성

소스 디버거와 함께 사용되는 CL 명령

표 19. 소스 디버거와 함께 사용되는 CL 명령

명령	설명하는 이름
DSPMODSRC	모듈 소스 표시
ENDDBG	디버그 종료
STRDBG	디버그 시작

바인더 언어 소스 파일 편집에 사용되는 CL 명령

표 20. 바인더 언어 소스 편집에 사용되는 CL 명령

명령	설명하는 이름
EDTF	파일 편집
STRPDM	프로그래밍 개발 관리자 시작
STRSEU	소스 입력 유틸리티 시작
주: 다음의 실행불능 명령이 바인더 언어 소스 파일에 입력될 수 있습니다.	
ENDPGMEXP	프로그램 내보내기 종료
EXPORT	Export

부록 D. 주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다.

IBM은 다른 국가에서는 이 자료에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급하는 것이 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 기능상 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수 있습니다. 그러나 비IBM 제품, 프로그램 및 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

135-270
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터
전화번호: 080-023-8080

2바이트(DBCS) 정보에 관한 라이센스 문의는 한국 IBM 고객만족센터에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다. IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 (단, 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증없이 이 책을 『현상태대로』 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 이 변경사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및(또는) 프로그램을 사전 통지없이 언제든지 개선 및(또는) 변경할 수 있습니다.

이 정보에서 비IBM의 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

(1) 독립적으로 생성된 프로그램이나 기타 프로그램(본 프로그램 포함)간의 정보 교환 및 (2) 교환된 정보의 상호 이용을 목적으로 정보를 원하는 프로그램 라이센스 사용자는 다음 주소로 문의하십시오.

135-270
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터

이러한 정보는 해당 조항 및 조건(예를 들면, 사용료 지불 포함)에 따라 사용할 수 있습니다.

이 정보에 기술된 라이센스가 있는 프로그램 및 이 프로그램에 대해 사용 가능한 모든 라이센스가 있는 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이들 예제에는 개념을 가능한 완벽하게 설명하기 위해 개인, 회사, 상표 및 제품의 이름이 사용될 수 있습니다. 이들 이름은 모두 가공의 것이며 실제 기업의 이름 및 주소와 유사하더라도 이는 전적으로 우연입니다.

저작권:

이 정보에는 여러 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원어로 된 샘플 응용프로그램이 들어 있습니다. 귀하는 이러한 샘플 프로그램의 작성 기준이 된 운영 플랫폼의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 추가 비용없이 이들 샘플 프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다. 이러한 샘플 프로그램은 모든 조건하에서 완전히 테스트된 것은 아닙니다. 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다. 귀하는 IBM의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 추가 비용없이 이러한 샘플 응용프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다.

이러한 샘플 프로그램 또는 파생 제품의 각 사본이나 그 일부에는 반드시 다음과 같은 저작권 표시가 포함되어야 합니다.

프로그래밍 인터페이스 정보

이 책은 통합 언어환경의 사용을 돕기 위한 것입니다. 이 책에는 OS/400에서 제공하는 범용 프로그래밍 인터페이스와 관련 지침 정보가 나옵니다.

범용 프로그래밍 인터페이스는 OS/400 서비스를 제공 받는 프로그램을 고객이 작성할 수 있게해 줍니다.

상표

다음 용어는 미국 또는 기타 국가에서 사용되는 IBM Corporation의 상표입니다.

400
AS/400
CICS
IBM
iSeries
OS/2
OS/400
POWER4
WebSphere

Windows 및 Windows 로고는 미국 또는 기타 국가에서 사용되는 Microsoft® Corporation의 상표입니다.

Java 및 모든 Java 기반 상표는 미국 또는 기타 국가에서 사용되는 Sun Microsystems, Inc.의 상표입니다.

UNIX는 미국 또는 기타 국가에서 사용되는 Open Group의 등록상표입니다.

기타 회사, 제품 및 서비스 이름은 해당 회사의 상표 또는 서비스 표입니다.

참고 서적

iSeries 서버에서의 ILE 환경 관련 주제에 관한 추가 정보는 다음 책을 참조하십시오.

- 백업 및 회복 Information Center 주제는 백업 및 회복 전략, 시스템 자료 저장 및 복원에 사용할 수 있는 서로 다른 매체 유형에 관한 정보와 저널링을 사용하여 데이터베이스 파일에 수행된 변경사항을 기록하는 방법 및 해당 정보를 시스템 회복에 사용하는 방법에 대한 설명을 제공합니다. 이 책은 다른 사용가능한 회복 주제와 함께 사용자 보조 기억장치 풀(ASP), 이중 복사(mirrored) 보호 및 체크섬의 계획 및 설치 방법을 설명합니다. 이 책은 또한 백업에서 시스템을 다시 설치하는 방법도 설명합니다.

- CL 프로그래밍 은 오브젝트와 라이브러리, CL 프로그래밍, 프로그램간 흐름 및 통신 제어, CL 프로그램에서 오브젝트에 대한 작업 및 CL 프로그램 작성에 대한 일반적인 설명을 포함하여 광범위한 프로그래밍 주제를 제공합니다. 기타 주제로는 디버그 모드, 중단점, 추적 및 표시 기능을 포함하여 사전정의 메세지, 즉시 메세지 및 메세지의 처리, 사용자 정의 명령 및 메뉴의 정의와 작성, 어플리케이션 테스트 등을 들 수 있습니다.

- Communications Management 는 통신 환경에서 작업 관리, 통신 상태, 통신 문제점 추적과 진단, 오류 처리와 회복, 성능 및 특정 회선 속도와 서브시스템 기억장치 정보를 제공합니다.

- ICF Programming 통신 및 OS/400 시스템간 통신 기능(OS/400-ICF)을 사용하는 어플리케이션 프로그램을 작성하는이 책에는 또한 자료 서술 스펙(DDS) 키워드, 시스템 제공 형식, 리턴 코드, 파일 전송 지원 및 프로그램 예에 관한 정보도 포함하고 있습니다.

- WebSphere Development Studio ILE C/C++ Programmer's Guide 에서는 ILE C 및 ILE C++ 프로그램 작성, 컴파일, 디버그 및 실행하는 방법에 대해 설명합니다. 이 안내서는 ILE 및 OS/400 프로그래밍 피처, iSeries 파일 시스템, 장치 및 피처, I/O 조작, 현지화, 프로그램 성능 및 C++ 런타임 유형 정보(RTTI)에 대한 정보를 제공합니다.

- WebSphere Development Studio C/C++ Language Reference 에서는 *프로그래밍 언어 - C*(ISO/IEC 9899:1990) 표준 및 *프로그래밍 언어 - C++*(ISO/IEC 14882:1998) 표준에 대한 언어 준수를 설명합니다.

- WebSphere Development Studio ILE C/C++ Compiler Reference 에는 프리프로세서문, 매크로, 프라그마(pragma), iSeries 및 QShell 환경에 대한 명령행 사용법 및 I/O 고려사항을 포함하여 ILE C/C++ 컴파일로에 대한 참조 정보가 들어 있습니다.

- ILE C/C++ Run-time Library Reference 는 ILE C/C++, 런타임 라이브러리 기능, 포함 파일 및 런타임 고려사항에 대한 참조 정보를 제공합니다.

- WebSphere Development Studio: ILE COBOL Programmer's Guide 는 AS/400 시스템에서 ILE COBOL 프로그램을 작성, 컴파일, 바인드, 실행, 디버그 및 유지보수하는 방법에 대해 설명합니다. 이 책은 다른 ILE COBOL과 비ILE COBOL 프로그램을 호출하는 방법, 다른 프로그램과의 자료 공유 방법, 포인터 사용 방법 및 예외 처리에 관한 프로그래밍 정보를 제공합니다. 이 책은 또한

외부 접속 장치, 데이터베이스 파일, 화면 파일 및 ICF 파일에서 입/출력 조작을 수행하는 방법도 설명합니다.

- WebSphere Development Studio: ILE COBOL Reference 는 ILE COBOL 프로그래밍 언어에 관해 설명합니다. 이 책은 ILE COBOL 프로그래밍 언어의 구조와 ILE COBOL 소스 프로그램에 대한 정보를 제공합니다. 이 책은 또한 모든 Identification Division 단락, Environment Division절, Data Division 단락, Procedure Division문 및 컴파일러 지정문을 설명합니다.
- WebSphere Development Studio: ILE RPG Programmer's Guide 는 iSeries 서버의 통합 언어 환경(ILE)에서 ILE RPG를 구현시킨 RPG IV 프로그래밍 언어를 사용하기 위한 안내서입니다. 이 책에는 프로시듀어 호출과 복수 언어(interlanguage)의 프로그래밍에 대한 고려사항과 함께 프로그램을 작성하고 실행하는 것에 관한 정보가 포함되어 있습니다. 이 책은 또한 예외 디버깅과 예외 처리를 다루고 있으며, RPG 프로그램에서 AS/400 파일과 장치를 사용하는 방법을 설명합니다. 부록에는 RPG IV로의 마이그레이션에 관한 정보와 샘플 컴파일러 리스팅이 포함되어 있습니다. 이 책은 자료 처리 개념과 RPG 언어에 대해 기초적인 지식이 있는 사용자용으로 고안된 것입니다.
- WebSphere Development Studio ILE RPG Reference 는 RPG IV 프로그래밍 언어를 사용하여 iSeries 서버용 프로그램을 작성하는 정보가 들어 있습니다. 이 책은 모든 RPG 스펙에 대해 유효한 항목을 위치별로 또는 키워드별로 설명하며 모든 조작 코드와 내장 기능에 대한 상세한 설명을 제공합니다. 이 책에는 또한 RPG 논리 주기, 배열 및 표, 편집 기능 및 인디케이터에 대한 정보가 들어 있습니다.
- Intrasystem Communications Programming 은 iSeries 서버에서 두 개의 어플리케이션 프로그램간의 인트라시스템 통신에 대한 정보가 들어 있습니다. 이 책은 다른 프로그램과 통신하기 위해 시스템간 통신 지원을 사용하는 프로그램에 코딩할 수 있는 통신 조작을 설명합니다. 이 책은 OS/400 시스템간 통신 기능(OS/400-ICF)을 사용하는 시스템간 통신 어플리케이션 프로그램의 개발에 관한 정보도 제공합니다.
- iSeries 보안 참조서 는 적절한 권한이 없는 사용자가 시스템 및 자료를 사용하지 않도록 보호하고, 자료를 고의로 또는 실수로 손상시키지 않도록 보호하며 보안 정보가 최신의 상태로 유지되도록 시스템에 보안을 설정하기 위한 시스템 보안 지원 사용법을 설명합니다.
- iSeries Information Center의 **시스템 관리** 범주 아래에 나오는 작업 관리는 작업 관리 환경을 작성하고 변경하는 방법에 관한 정보를 제공합니다. 기타 주제로는 시스템 조정(tune), 수집중인 자료의 레코드 형식과 내용에 관한 정보를 포함하여 성능 자료의 콜렉션, 시스템 전체 조작을 제어하거나 변경하기 위한 시스템 값에 대한 작업, 그리고 시스템의 현재 사용자와 사용 중인 자원을 판별하기 위해 자료를 모으는 방법에 관한 정보를 들 수 있습니다.

색인

[가]

[나]

[다]

[라]

[마]

[바]

[사]

[아]

[자]

[차]

[카]

[타]

[파]

[하]

A

C

D

E

H

I

IBM 한글 지원에 관한 설문

iSeries
ILE 개념
버전 5 릴리스 3

SA30-0240-07

FAX : (02) 3787-0123

보내 주시는 의견은 더 나은 고객 지원 체제를 위한 귀중한 자료가 됩니다. 독자 여러분의 좋은 의견을 기다립니다.

성 명		직위/담당업무	
회 사 명		부 서 명	
주 소			
전화번호		팩스번호	
전자우편 주소			
사용중인 시스템	o 중대형 서버 o UNIX 서버 o PC 및 PC 서버		

1. IBM에서 제공하는 한글 책자와 영문 책자 중 어느 것을 더 좋아하십니까?
   그 이유는 무엇입니까?
   ☐ 한글 책자 ☐ 영문 책자
   (이유:)

2. 본 책자와 해당 소프트웨어에서 사용된 한글 용어에 대한 귀하의 평가 점수는?
   ☐ 수 ☐ 우 ☐ 미 ☐ 양 ☐ 가

3. 본 책자와 해당 소프트웨어에서 번역 품질에 대한 귀하의 평가 점수는?
   ☐ 수 ☐ 우 ☐ 미 ☐ 양 ☐ 가

4. 본 책자의 인쇄 상태에 대한 귀하의 평가 점수는?
   ☐ 수 ☐ 우 ☐ 미 ☐ 양 ☐ 가

5. 한글 소프트웨어 및 책자가 지원되는 분야에 대해 귀하는 어떻게 생각하십니까?
   ☐ 한글 책자를 늘려야 함 ☐ 현재 수준으로 만족
   ☐ 그다지 필요성을 느끼지 않음

6. IBM은 인쇄물 형식(hardcopy)과 화면 형식(softcopy)의 두 종류로 책자를 제공합니다.
   어느 형식을 더 좋아하십니까?
   ☐ 인쇄물 형식(hardcopy) ☐ 화면 형식(softcopy) ☐ 둘 다

IBM 한글 지원 서비스에 대해 기타 제안사항이 있으시면 적어주십시오.

☺ 설문에 답해 주셔서 감사합니다.
귀하의 의견은 저희에게 매우 소중한 것이며, 고객 여러분들께 보다 좋은 제품을 제공해 드리기 위해 최선을 다하겠습니다.

SA30-0240-07