

z/OS용 CICS Transaction Server
버전 5 릴리스 3



어플리케이션 프로그래밍 안내서

z/OS용 CICS Transaction Server
버전 5 릴리스 3



어플리케이션 프로그래밍 안내서

참고

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 1027 페이지의 『주의사항』의 정보를 읽으십시오.

이 개정판은 새 개정판에 별도로 명시하지 않는 한, z/OS용 IBM CICS Transaction Server 버전 5 릴리스 3(제품 번호 5655-Y04) 및 모든 후속 릴리스와 수정에 적용됩니다.

© Copyright IBM Corporation 1989, 2015.

목차

이 설명서의 내용	xi
이 매뉴얼의 사용자	xi
이 설명서를 이해하기 위해 알아야 하는 내용	xi
이 매뉴얼을 사용하는 방법	xi
Knowledge Center에서 주제의 위치	xi
용어에 대한 참고	xii
이 매뉴얼에 설명되지 않는 내용	xii

z/OS용 CICS Transaction Server, 버전 5 릴리

스 3 의 변경사항	xiii

제 1 부 CICS 애플리케이션 작성 1

제 1 장 CICS 애플리케이션 쓰기	3
CICS 애플리케이션의 개념	3
CICS 프로그램, 트랜잭션 및 태스크	3
CICS 프로그래밍	4
CICS 프로그래밍 명령	5
실행 인터페이스 블록(EIB)	6
변환	6
CICS 테스트	7
CICS 프로그래밍 로드맵	7

제 2 부 프로그래밍 언어 및 Language Environment 9

제 2 장 프로그래밍 언어 및 Language Environment	11
Language Environment 호출 가능 서비스	12
Language Environment 이상 종료 및 조건 처리	13
Language Environment 저장영역	15
Language Environment에서 언어 혼합	16
동적 링크 라이브러리(DLL)	18
Language Environment 런타임 옵션 정의	19
CEEEXITA 및 CEECSTX 사용자 엑시트	21
CICSVAR: CICS 환경 변수	22
Language Environment에 대한 CEEBINT 엑시트	24

제 3 장 COBOL 애플리케이션 개발	25
COBOL 프로그래밍 제한사항 및 요구사항	26
Language Environment CBLPSHPOP 옵션	29
DL/I CALL 인터페이스 사용	30
VS COBOL II 프로그램	31

COBOL에서 기본 주소 지정 사용	33
COBOL 프로그램에서 서브프로그램 호출	34
프로그램 및 서브프로그램 간 제어 플로우	35
서브프로그램 호출에 대한 규칙	37
COBOL2 및 COBOL3 변환기 옵션	41
COBOL 프로그램을 위한 CICS 변환기 조치	42
COBOL 프로그램의 일괄처리 컴파일	44
중첩된 COBOL 프로그램	47

제 4 장 C 및 C++로 프로그래밍	53
C 및 C++ 프로그래밍 제한사항과 요구사항	54
C 및 C++의 인수 전달	58
C 및 C++에서 EIB 액세스	60
C 및 C++에 대한 로케일 지원	61
XPLink와 C 및 C++ 프로그래밍	61
XPLink의 X8 및 X9 모드 TCB 사용	62
XPLink 및 비XPLink 오브젝트 간 제어 전달	62
글로벌 사용자 엑시트 및 XPLink	63

제 5 장 PL/I 애플리케이션 개발	65
PL/I 프로그래밍 제한사항 및 요구사항	66
PL/I 애플리케이션에 대한 Language Environment 코드화 요구사항	67
폐지된 PL/I 루틴	69

제 6 장 어셈블리 언어 애플리케이션 개발	71
어셈블리 언어 프로그래밍 제한사항 및 요구사항	71
EXEC CICS 어셈블리 인터페이스 코딩	76
어셈블리 언어 애플리케이션에 대한 Language Environment 코드화 요구사항	81
어셈블리 언어 프로그램 호출	83
동적 저장영역으로의 확장	86
AMODE(64) 어셈블리 언어 프로그램 개발	88

제 3 부 애플리케이션 변환, 컴파일, 설치, 테스트 91

제 7 장 변환 및 컴파일	93
통합 CICS 변환기	93
통합 CICS 변환기 사용	94
CICS 변환기 옵션 지정하기	95
변환 프로세스	96
CICS 제공 변환기	99

개별 변환기 동적 호출	100
CICS 변환기 사용	111
변환기 옵션 정의	113
변환기 옵션 표.	114
COPY문 사용	115
CICS 제공 인터페이스 모듈	116
AMODE(24) 및 AMODE(31) 애플리케이션에	
EXEC 인터페이스 모듈 사용	117
LEASM이 있는 예제 어셈블러 언어 프로그램	120
AMODE(64) 애플리케이션에 EXEC 인터페이스	
모듈 사용	128
 제 8 장 플랫폼에 애플리케이션 전개	131
플랫폼에 애플리케이션 설치	132
플랫폼과 애플리케이션의 클라우드 인에이블먼트 시	
작하기.	134
 제 9 장 애플리케이션 설치.	143
프로그램 설치 단계	144
동적 프로그램 LIBRARY 자원 사용	145
동적 LIBRARY 자원 사용 예제.	146
MVS 거주와 주소 지정 모드 정의	156
프로그램의 주소 지정 모드 설정	156
CICS 주소 공간 고려사항	157
프로그램을 영구 상주로 만들기	157
링크 팩 영역에서 애플리케이션 실행	158
읽기 전용 DSA에서 애플리케이션 실행.	159
어셈블러	159
C 및 C++	161
COBOL	162
PL/I	162
애플리케이션에서 BMS 맵핑 세트 사용	163
CICS 제공 프로시저를 사용하여 애플리케이션 설치	164
로드 라이브러리 2차 익스텐트에 프로그램 설치	165
CICS 제공 인터페이스 모듈 포함	166
어셈블러 언어 애플리케이션 변환, 어셈블 및 링크	
편집	167
COBOL 애플리케이션 설치	169
COBOL 애플리케이션을 설치하기 위한 샘플	
JCL	170
PL/I 애플리케이션 설치.	174
C 애플리케이션 설치.	174
C 애플리케이션을 설치하기 위한 샘플 JCL	176
자체 작업 스트림 사용	179
 제 10 장 맵핑 세트 및 파티션 세트 설치.	183
맵핑 세트 설치.	185

맵핑 세트의 유형	185
물리적 맵핑 세트 설치	187
기호 설명 맵핑 세트 설치	189
물리적 및 기호 설명 맵을 함께 설치	191
파티션 세트 설치	194
CICS에 프로그램, 맵핑 세트 및 파티션 세트 정의	196
 제 11 장 애플리케이션 테스트	197
애플리케이션의 테스트 준비	198
테스트에 대한 시스템 준비.	199
 제 12 장 진단 기능 실행(EDF)	201
EDF 사용 시 제한사항	202
EDF에서 표시하는 내용.	205
헤더	205
본문	206
EDF를 사용하여 프로그램 테스트	212
프로그램 실행 인터럽트	212
단일 화면 모드에서 EDF 사용	214
이중 화면 모드에서 EDF 사용	216
EDF 및 원격 트랜잭션	216
EDF 및 비터미널 트랜잭션.	217
EDF 및 DTP 프로그램	217
EDF 중지	218
EDF를 사용하여 정보 변경.	219
EDF 응답	221
EDF 메뉴 기능 사용.	221
 제 13 장 임시 저장영역 찾아보기(CEBR)	231
CEBR 트랜잭션 사용	231
CEBR 트랜잭션이 표시하는 내용	233
CEBR 기능 키 사용.	234
CEBR 명령 사용	236
트랜지언트 데이터에 CEBR 트랜잭션 사용	238
 제 14 장 명령 레벨 해독기(CECI)	241
CECI가 표시하는 내용	241
명령행.	241
상태 표시줄.	242
화면 본문	245
메시지 행	246
기능 키의 CECI 옵션	246
CECI 사용	247
변수 정의	248
명령 저장	250
CECI의 실행 방식	251
공유 저장영역: LENGTH 옵션 없는 ENQ 명령	252

제 15 장 CICS 애플리케이션에서 디버거 사용 준비	253
디버깅 프로파일	255
디버깅할 프로그램을 선택하기 위해 디버깅 프로파일 사용	256
디버깅 프로파일에서 일반 매개변수 사용	258
제 16 장 워크스테이션에서 CICS 애플리케이션 디버깅	261
워크스테이션에서 애플리케이션 디버깅 준비	261
제 17 장 CICS 애플리케이션에서 디버깅 도구 사용	263
디버깅 도구 정보	263
디버깅 도구를 사용한 애플리케이션 디버깅 준비	264
제 4 부 CICS 애플리케이션 프로그래밍 기술	265
제 18 장 애플리케이션 디자인	267
의사 대화식 및 대화식 디자인	267
터미널 중단 기능	269
태스크 시작 방법	270
트랜잭션 선택	271
비즈니스 및 표시 논리 분리	274
멀티스레딩: 재진입 가능, 유사 재진입 가능 및 스레드 세이프 프로그램	275
유사 재진입 가능 애플리케이션	277
스레드 세이프 프로그램	278
애플리케이션 스레드 세이프화	281
CONCURRENCY(REQUIRED) 프로그램	286
OPENAPI 프로그램	287
FORCEQR 시스템 초기화 매개변수 사용	291
재진입 불가능 프로그램	291
트랜잭션에서 데이터 저장	292
TWA(Transaction Work Area)	292
사용자 저장영역	293
LINK 및 XCTL 명령의 COMMAREA	294
LINK 및 XCTL 명령의 채널	295
프로그램 저장영역	295
임시 저장영역 큐	295
리전 내부 트랜지언트 데이터	298
GETMAIN SHARED 명령	299
사용자 고유의 데이터 세트	299
CICS에 전달된 영역의 길이	299
오류 최소화	301
CICS를 애플리케이션 오류로부터 보호하기	301

애플리케이션 테스트	301
비터미널 트랜잭션 보안	303
성능 향상을 위한 디자인	303
프로그램 크기	304
가상 저장영역	305
자원 독점 제어	309
운영 제어	310
운영 체제 대기	311
NOSUSPEND 옵션	311
효율적 순차 데이터 세트 액세스	313
효율적 로깅	314
트랜잭션 사이에 데이터 공유	314
CWA(Common Work Area) 사용	315
TCTTE 사용자 영역(TCTUA) 사용	319
RETURN 명령에서 COMMAREA 사용	320
RETURN 명령에서 채널 사용	320
표시장치 화면을 사용하여 데이터 공유	321
채널을 사용하는 프로그램 간 데이터 통신	321
채널: 빠른 시작	322
채널 사용: 일부 일반 시나리오	327
채널의 이점	330
채널 작성	332
현재 채널	333
채널의 범위	338
프로그램에 전달된 컨테이너 발견	343
링크에서 리턴된 컨테이너 발견	343
채널과 컨테이너 삭제 및 해당 저장영역의 사용 가능화	344
채널 설계: 우수 사례	346
채널 구성 및 사용: 예제	348
CICS 읽기 전용 컨테이너	350
JCICS에서 채널 사용	350
채널 및 BTS 활동	350
채널을 사용한 동적 라우팅	353
데이터 변환	353
COMMAREA에서 채널로 마이그레이션	360
프로그램 제어	364
프로그램 링크	366
다른 프로그램으로 데이터 전달	367
혼합 주소 지정 모드 사용	372
LINK를 사용한 데이터 전달	373
RETURN을 사용한 데이터 전달	376
연관관계	379
연관관계 유형	380
프로그래밍 기술 및 연관관계	382
연관관계를 피하는 프로그래밍 기술	384

연관관계를 작성하는 프로그래밍 기술	388
연관관계를 작성할 수 있는 프로그래밍 기술	396
트랜잭션간 연관관계 발견	408
트랜잭션간 연관관계의 지속 기간 및 범위	409
복구 디자인	418
저널링	418
동기점 지정	421
예외 조건 처리	423
기본 CICS 예외 처리	423
인라인 코드로 예외 조건 처리	424
기본 CICS 예외 처리 수정	428
HANDLE CONDITION 명령 사용	429
HANDLE CONDITION ERROR 명령 사용	432
IGNORE CONDITION 명령 사용	433
HANDLE ABEND 명령 사용	434
PUSH HANDLE 및 POP HANDLE 명령 사 용	435
SOAP 결합 메시지 구문 분석	436
I 배수 효과 방지	441
비정상 종료 복구	443
프로그램 레벨 이상 종료 프로그램 또는 루틴 작 성	445
조작 재시도	446
추적	447
모니터링 애플리케이션 성능	449
덤프	449
QUERY SECURITY 명령	451
QUERY SECURITY 명령 사용	451
CICS 상호 통신	452
트랜잭션 라우팅	454
Function shipping	454
DPL(Distributed Program Link)	455
비동기 처리	468
DTP(Distributed Transaction Processing)	469
공통 프로그래밍 인터페이스 통신(CPI 통신)	469
외부 CICS 인터페이스(EXCI)	470

제 5 부 데이터 맵핑 473

제 19 장 애플리케이션 데이터와 XML 맵핑 및 변 환	475
CICS XML 지원	476
DFHLS2SC: 고급 언어 대 XML 스키마 변환	477
DFHSC2LS: XML 스키마 대 고급 언어 변환	484
CICS 지원의 맵핑 레벨	495
고급 언어 및 XML 스키마 맵핑	501
변동 요소 배열	538

XML 속성에 대한 지원	543
<xsd:any> 및 xsd:anyType에 대한 지원	546
<xsd:choice>에 대한 지원	549
<xsd:sequence>에 대한 지원	551
대체 그룹에 대한 지원	552
추상 요소 및 추상 데이터 유형에 대한 지원	553
COBOL의 가변적 반복 내용 처리 방법	555
가변 길이 값과 공백에 대한 지원	559
애플리케이션 데이터의 UTF-16에 대한 지원	561
언어 구조에서 맵핑 생성	563
XML 스키마에서 맵핑 생성	566
애플리케이션 데이터를 XML로 변환	568
XML을 애플리케이션 데이터로 변환	569
애플리케이션에서 XML 조회	571
데이터 유형으로 XML 처리	572
<xsd:any> 데이터 유형 처리	573
XML 변환 유효성 검사	574

제 20 장 애플리케이션 데이터와 JSON 맵핑 및 변환	577
CICS JSON 지원	578
DFHLS2JS: 링크 가능한 인터페이스의 고급 언 어 대 JSON 스키마 변환	579
DFHJS2LS: 링크 가능한 인터페이스의 JSON 스키마 대 고급 언어 변환	586
CICS JSON 지원의 맵핑 레벨	596
고급 언어 및 JSON 스키마 맵핑	601
DFHJS2LS의 변동 요소 배열	634
애플리케이션 데이터의 UTF-16에 대한 지원	639
언어 구조에서 맵핑 생성	641
JSON 스키마에서 맵핑 생성	644
애플리케이션 데이터를 JSON으로 변환	645
JSON을 애플리케이션 데이터로 변환	647
JSON 웹 서비스 클라이언트 애플리케이션 작성	648

제 6 부 비즈니스 서비스 및 번들 651

제 21 장 CICS 애플리케이션에서 비즈니스 서비스 작성	653
SCA(Service Component Architecture)	653
SCA 컴포지트 및 와이어링	655
컴포지트 작성 및 전개 우수 사례	657
채널 기반 서비스 작성	658
XML 기반 서비스 작성	660
CICS 서비스 처리	662
SCA 문제점 해결	663

제 22 장 전개를 위한 애플리케이션 패키징	665
애플리케이션을 플랫폼으로 바인드하는 방법	667

제 7 부 파일 제어 671

제 23 장 파일 제어 이해	673
VSAM 데이터 세트: KSDS, ESDS, RRDS	673
RLS 모드에서 파일에 액세스	676
VSAM 레코드 식별	677
키	677
상대 레코드 번호(RRN), 상대 바이트 주소 (RBA) 및 확장 상대 바이트 주소(XRBA)	678
ESDS에 대한 확장된 주소 지정으로 업그레이드	679
복구 가능한 파일에서 VSAM 레코드 잠금	681
RLS 레코드 레벨 잠금	682
독점 잠금 및 공유 잠금	683
잠금 기간	684
잠금에 대한 활성 및 보유 상태	685
BDAM 데이터 세트	686
BDAM 레코드 식별	687
CICS 공유 데이터 테이블	689
결합 기능 데이터 테이블	690
데이터 공유 기술	693
트랜잭션 교착상태	695
VSAM 발견 교착(RLS만 해당)	697
교착을 피하기 위한 규칙	697
파일 제어 조작	698
CICS 명령을 사용하여 레코드 읽기	698
CICS 명령을 사용하여 레코드 갱신	708
CICS 명령을 사용한 레코드 삭제	711
찾아보기에서 레코드 갱신 및 삭제(VSAM RLS만)	712
CICS 명령을 사용한 레코드 추가	713
유효 데이터 세트 운영	716

제 8 부 터미널 제어 721

제 24 장 터미널 액세스 방법 지원	723
제 25 장 터미널 제어 명령	725
전송/수신 모드	725
터미널에 대한 결합	726
RETURN IMMEDIATE	726
상황에 벗어나서 말하기	727
인터럽트	728
터미널 대기	728

제 26 장 데이터 전송 명령 사용	731
RECEIVE에서 가져오는 사항	731

제 27 장 디바이스 제어 명령 733

제 28 장 터미널 디바이스 지원 735

제 29 장 터미널 정보 찾기	739
터미널 제어 운영에 대한 EIB 피드백	740

제 30 장 SNA 사용	743
입력 데이터 채이닝	743
출력 데이터 채이닝	743
논리 레코드 처리	744
응답 프로토콜	744
기능 관리 헤더 사용	745
인바운드 FMH	745
아웃바운드 FMH	745
브래킷 프로토콜로 인터럽트 방지	746

제 31 장 순차 터미널 지원 사용	747
순차 터미널에 대한 코딩 고려사항	748
인쇄 형식화	748
GOODNIGHT 규칙	748

제 32 장 일괄처리 데이터 상호 교환 사용 751

제 33 장 터미널 제어: 성능 향상을 위한 디자인 755

제 34 장 터미널의 3270 제품군	757
3270 버퍼	757
출력 데이터 스트림	758
3270 표시장치 필드	759
표시장치 특성	760
3270 필드 속성	761
확장 속성	763
데이터 스트림에서의 순서	763
시작 필드 순서	764
수정 필드 순서	765
설정 버퍼 주소 순서	766
설정 속성 순서	767
아웃바운드 데이터 스트림 예제	767
3270 터미널에서의 입력	769
3270 터미널에서 읽기	771
인바운드 필드 형식	772
입력 데이터 스트림 예제	772
형식화되지 않은 모드	773

제 9 부 간격 제어 및 태스크 제어 775

제 35 장 간격 제어	777
만기 시간	778

제 36 장 태스크 제어	781
자원에 대한 액세스의 순서 제어	783

제 10 부 저장영역 보호 및 트랜잭션 분리 785

제 37 장 CICS 저장영역 보호 및 트랜잭션 분리	787
저장영역 제어	787
저장영역 보호	788
저장영역 카테고리	789
트랜잭션 분리	790
애플리케이션에 대한 저장영역 키 정의	792
시스템 전체 저장영역	792
태스크 수명 저장영역	792
특히 엑시트와 PLT 프로그램을 위한 프로그램	
작업 중인 저장영역	793
COMMAREA를 사용하여 데이터 전달	793
GETMAIN 명령	793
실행 및 저장 키 선택	795
사용자 키 애플리케이션	796
CICS 키 애플리케이션	797
트랜잭션 분리로 보호	800
MVS 서브공간	802

제 11 부 트랜지언트 데이터 및 임시 저장영역 805

제 38 장 트랜지언트 데이터 제어	807
리전 내부 트랜지언트 데이터 큐	807
리전 외부 큐	808
간접 큐	809
ATI(Automatic Transaction Initiation)	809

제 39 장 임시 저장영역 제어	813

제 12 부 CICS 문서 815

제 40 장 CICS 문서 및 문서 템플릿	817
기호 및 기호 목록	818
문서 템플릿의 캐싱 및 새로 고치기	821
문서에 대한 코드 페이지 변환	823

제 41 장 문서 템플릿 설정	825
파티션된 데이터 세트의 템플릿	825
z/OS UNIX System Services 파일의 템플릿	826

CICS 파일, 임시 저장영역 또는 트랜지언트 데이터	
의 템플릿	826
CICS 프로그램의 템플릿	827
DFHDHTL - 프로그램 템플릿 프로로그 및	
에필로그 매크로	829
엑시트 프로그램의 템플릿	830
엑시트 프로그램에서 템플릿을 위한 통신 영역	830
문서 템플릿에서 기호 사용	832
임베디드 템플릿 명령	833

제 42 장 문서 및 문서 템플릿을 사용하여 프로그래밍	835
문서 작성	835
기호 값 정의	837
기호 및 기호 목록을 지정하기 위한 규칙	839
문서에 추가 데이터 추가	842
문서에서 데이터 바꾸기	844
문서 검색, 저장 및 재사용	845
문서 삭제	848

제 13 부 이름 지정된 카운터 서버 849

제 43 장 이름 지정된 카운터 서버	851
이름 지정된 카운터 필드	852
이름 지정된 카운터 풀	852
이름 지정된 카운터 옵션 테이블	853
이름 지정된 카운터 EXEC 인터페이스 사용	854
이름 지정된 카운터 CALL 인터페이스 사용	856
애플리케이션 고려사항	856
구문	858
리턴 코드	867
이름 지정된 카운터 복구	871

제 14 부 인쇄 및 스폴 파일 875

제 44 장 인쇄 및 스폴 파일	877
CICS 프린터의 형식	877
인쇄된 출력에 대한 요청	878
CICS 3270 프린터	879
CICS 3270 프린터 옵션	880
PRINT 옵션과 인쇄 제어 비트	880
ERASE 옵션	881
행 너비 옵션: L40, L64, L80, HONEOM	881
NLEOM 옵션	882
FORMFEED	883
PRINTERCOMP 옵션	883
비3270 CICS 프린터	884

SCS 입력	885
CICS에서 프린터 사용	885
CICS 프린터의 특성 결정하기.	885
CICS 프린터 사용	887
비CICS 프린터 사용.	890
표시장치 화면 인쇄	894
JES에 대한 CICS 인터페이스.	895
JES에 대한 CICS 인터페이스 사용.	898
출력 스푼 파일 작성	899
입력 스푼 파일 읽기.	900
스푼 파일 식별.	902
SPOOL 명령의 예제.	905

제 15 부 BMS(Basic Mapping Support) 909

제 45 장 BMS(Basic Mapping Support)	911
BMS 지원 레벨	912
BMS 출력 예제	913
맵 작성	916
맵 필드 정의: DFHMDF	917
맵 정의: DFHMDI	919
맵핑 세트 정의: DFHMSD.	920
BMS 매크로 쓰기.	921
맵 어셈블링.	924
복합 필드 사용.	927
블록 데이터.	930
비3270 터미널에 대한 지원	931
디바이스 종속 맵	933
BMS 맵핑된 출력 전송.	937
맵의 저장영역 획득 및 정의	937
출력 맵 초기화.	939
변수 데이터를 맵으로 이동.	940
표시장치 특성 설정	940
SEND MAP 명령 사용.	943
SEND MAP 제어 옵션.	943
기호 및 물리적 맵핑 병합	945
출력 화면 빌드.	946
커서 위치 지정.	949
올바르지 않은 데이터 및 기타 오류 전송	950
출력 배치 옵션: TERMINAL, SET, PAGING	950
맵핑된 데이터 수신	953
입출력(I/O) 예제	953
맵핑된 입력 프로그래밍	956
RECEIVE MAP 명령 사용	957
맵핑된 입력에 대한 저장영역 가져오기.	958

형식화된 화면 입력	958
어텐션 ID 사용	960
커서 찾기	962
맵핑된 입력 처리	962
입력 오류 처리.	963
맵핑된 입력 후 맵핑된 출력 전송.	965
MAPFAIL 및 기타 예외 조건	966
다른 입력 형식화	967
BMS 논리 메시지.	968
논리 메시지 빌드	968
SEND PAGE 명령	969
RETAIN 및 RELEASE	970
AUTOPAGE 옵션	971
터미널 운영자 페이징: CSPG 트랜잭션.	972
논리 메시지 복구	973
누적 출력 — ACCUM 옵션	973
플로팅 맵: BMS가 ACCUM을 사용하여 맵을 위치시키는 방법	974
페이지 중단: BMS 오버플로우 처리.	974
맵 배치 규칙	976
컴포지트 화면의 입력.	979
성능 가이드.	979
텍스트 출력.	981
SEND TEXT 명령	981
텍스트 페이지	982
텍스트 행	984
헤더 및 트레일러 형식	984
SEND TEXT MAPPED 및 SEND TEXT NOEDIT.	986
메시지 라우팅	987
메시지 대상.	988
라우팅 목록 형식	991
메시지 전달.	993
복구 가능 메시지	994
라우팅에 대한 프로그래밍 고려사항	995
MAPPINGDEV 기능	997
샘플 어셈블러 MAPPINGDEV 애플리케이션	999
파티션 지원	1001
파티션된 화면에 대한 사용	1002
파티션 정의	1003
파티셔닝 설정.	1005
BMS SEND 명령에 대한 파티션 옵션	1006
BMS RECEIVE 명령에 대한 파티션 옵션	1007
파티션 및 논리 메시지.	1007
어텐션 ID 및 예외 상태	1008
터미널 공유	1009

특수 하드웨어에 대한 지원	1010
논리 디바이스 구성요소	1010
10/63 자기 슬롯 리더	1012
필드 선택사항 기능	1013
커서 및 펜 감지 가능 필드	1014
아웃보드 형식화	1016
BMS: 성능 향상을 위한 디자인.	1017
페이지 빌드 및 라우팅 작업	1020

제 16 부 부록 1025

주의사항	1027
상표	1029

참고 문헌	1031
z/OS용 CICS Transaction Server의 CICS 서적	1031
z/OS용 CICS Transaction Server에 대한	
CICSplex SM 서적	1032
기타 CICS 서적	1033
기타 IBM 서적	1033
내게 필요한 옵션	1035
색인	1037

이 설명서의 내용

이 매뉴얼은 고객이 IBM® CICS® Transaction Server 버전 5 릴리스 3의 서비스를 확보하기 위해 프로그램을 쓸 수 있는 의도한 프로그래밍 인터페이스를 문서화합니다.

이 설명서는 CICS 서비스와 자원에 액세스하기 위해 CICS EXEC 애플리케이션 프로그래밍 인터페이스를 사용하는 절차 애플리케이션의 개발에 대한 **안내**를 제공합니다. 이는 *CICS 애플리케이션 프로그래밍 참조 설명서*에서 **참조** 정보를 보충합니다. 이러한 CICS 애플리케이션 디버깅에 대한 안내 정보는 *CICS 문제점 판별 안내서*의 내용을 참조하십시오. Java™ 언어를 사용하여 애플리케이션 개발에 대한 안내는 CICS에서 *Java 애플리케이션*을 참조하고, CICS OO 클래스 사용에 대한 안내는 *CICS C++ OO 클래스 라이브러리*를 참조하십시오.

이 매뉴얼의 사용자

이 매뉴얼은 기본적으로 경험이 풍부한 애플리케이션을 위한 것입니다. CICS에 상대적으로 초보인 사용자는 이를 이해할 수 있어야 합니다. 시스템 프로그래머이거나 시스템 분석가라면 이것이 유용함을 알고 있어야 합니다.

이 설명서를 이해하기 위해 알아야 하는 내용

COBOL, C, C++, PL/I 또는 어셈블러 언어로 프로그래밍할 수 있어야 하며 *CICS 애플리케이션 디자인 및 프로그래밍 레벨*에서 CICS 애플리케이션의 기본 지식이 있어야 합니다.

이 매뉴얼을 사용하는 방법

알아야 하는 내용을 다루는 파트를 읽으십시오. (각 파트에는 원하는 것을 찾는 데 도움을 주는 전체 목차가 있습니다.) 매뉴얼은 참조 매뉴얼이 아니라 안내서입니다. 처음 읽을 때 대략 한 파트를 처음부터 끝까지 읽어보는 것이 좋습니다.

Knowledge Center에서 주제의 위치

이 서적의 주제는 z/OS용 CICS Transaction Server Knowledge Center에서도 찾을 수 있습니다. Knowledge Center는 정보가 표시되는 방식을 구조화시키기 위해 콘텐츠 유형을 사용합니다.

Knowledge Center 콘텐츠 유형은 일반적으로 업그레이드, 구성, 설치와 같은 태스크 중심적입니다. 다른 콘텐츠 유형으로는 참조, 개요, 시나리오 또는 학습서 기반 정보가 있습니다. 다음 맵핑은 이 서적의 주제와 외부 Knowledge Center에 대한 링크가 있는 Knowledge Center 콘텐츠 유형 간의 관계를 보여줍니다.

표 1. PDF 주제 대 Knowledge Center 콘텐츠 유형 맵핑. 이 표는 Knowledge Center의 콘텐츠 유형 주제와 PDF의 주제 간 관계를 나열합니다.

이 서적의 주제 세트	Knowledge Center의 위치
3 페이지의 제 1 장 『CICS 애플리케이션 쓰기』 및 9 페이지의 제 2 부 『프로그래밍 언어 및 Language Environment』 91 페이지의 제 3 부 『애플리케이션 변환, 컴파일, 설치, 테스트』 - 다른 모든 파트	- 애플리케이션 개발 - 디버깅 - 전개

용어에 대한 참고

API 별도로 언급되지 않는 한 CICS 명령 레벨 애플리케이션 프로그래밍 인터페이스를 참조하십시오.

ASM 종종 어셈블러 언어의 약어로 사용됩니다.

MVS™

z/OS®, OS/390® 또는 MVS/Enterprise System Architecture System Product(MVS/ESA SP)의 요소일 수 있는 운영 체제를 가리킵니다.

VTAM®

ACF/VTAM을 가리킵니다.

이 책에 설명된 샘플 프로그램에서 달러 기호(\$)는 자국의 통화 기호로 사용되고 EBCDIC 코드 포인트 X'5B'가 지정될 것으로 가정됩니다. 몇몇 국가에서는 다른 통화 기호, 예를 들어, 파운드 기호(£) 또는 엔 기호(¥)는 동일 EBCDIC 코드 포인트가 지정됩니다. 이러한 국가에서는 달러 기호 대신에 적합한 통화 기호가 사용되어야 합니다.

이 매뉴얼에 설명되지 않는 내용

CICS FEPI(Front End Programming Interface)의 사용 안내는 이 매뉴얼에서는 논의되지 않습니다. FEPI 설계 고려사항 및 해당 API에 대한 프로그래밍 정보는 *CICS FEPI(Front End Programming Interface) 사용자 안내서*의 내용을 참조하십시오.

EXEC CICS WEB 명령의 사용 안내는 이 매뉴얼에서는 논의되지 않습니다. 자세한 정보는 *CICS 인터넷 안내서*를 참조하십시오.

객체 지향 프로그래밍 언어와 기술의 사용 안내는 이 매뉴얼에는 포함되지 않습니다. Java 언어를 사용하는 애플리케이션 개발에 대한 안내는 *CICS의 Java 애플리케이션* 및 *CICS OO 클래스 사용*에 대한 안내는 *CICS C++ OO 클래스 라이브러리*를 참조하십시오.

z/OS용 CICS Transaction Server, 버전 5 릴리스 3 의 변경사항

이번 릴리스의 변경사항에 대한 정보는 Knowledge Center의 새로운 기능, 또는 다음 서적을 참조하십시오.

- *z/OS용 CICS Transaction Server* 새로운 기능
- *CICS TS* 버전 5.3으로 *z/OS용 CICS Transaction Server* 업그레이드

릴리스 후 텍스트의 기술 변경사항은 각각의 변경된 또는 새 정보 행의 왼쪽에 새로 막대(I)로 표시됩니다.

제 1 부 CICS 애플리케이션 작성

제 1 장 CICS 애플리케이션 쓰기

CICS 애플리케이션과 CICS에서 실행되는 프로그램 작성 프로시저를 소개합니다.

CICS API(Application Programming Interface)에 대한 참조 정보는 참조 -> 애플리케이션 개발에서 CICS API 명령의 내용을 참조하십시오.

CICS 애플리케이션의 개념

애플리케이션은 제품 주문 처리 또는 회사 급여 준비와 같은 비즈니스 조작을 함께 수행하는 관련 프로그램 컬렉션입니다. CICS 애플리케이션은 CICS 제어 하에 실행되며 CICS 서비스 및 인터페이스를 사용하여 프로그램과 파일에 액세스합니다.

CICS는 트랜잭션 처리 서브시스템입니다. 많은 다른 사용자가 동일한 파일과 프로그램을 사용하여 동일한 애플리케이션을 실행하도록 요청을 제출하면 동시에 요청을 통해 온라인으로 애플리케이션을 실행하도록 서비스를 제공합니다. CICS는 빠른 응답을 통해 자원 공유, 데이터 무결성, 실행 우선순위 지정을 관리합니다.

CICS 애플리케이션은 일반적으로 트랜잭션 요청을 제출하여 실행됩니다. 트랜잭션 실행은 필수 기능을 구현하는 하나 이상의 애플리케이션 실행으로 이루어집니다. CICS 문서에서는 CICS 애플리케이션을 때때로 프로그램이라고 하며 트랜잭션이란 용어는 애플리케이션이 수행하는 처리를 나타내는 데 사용됩니다.

트랜잭션이란 용어는 이제 IT 업계에서 복구 단위(CICS에서는 작업 단위라고 함)를 설명하는 데 광범위하게 사용됩니다. 이는 일반적으로 복구 가능한 전체 조작입니다. 프로그래밍된 명령 또는 시스템 장애의 결과로 전체로 확약 또는 백아웃할 수 있습니다. 많은 경우에 CICS 트랜잭션의 범위는 단일 작업 단위이기도 하지만 비CICS 문서를 사용하는 경우 의미 차이를 알아 두십시오.

CICS 프로그램, 트랜잭션 및 태스크

CICS 애플리케이션을 개발하고 실행하려면 CICS 프로그램, 트랜잭션, 태스크 간의 관계를 이해해야 합니다.

CICS 문서 전반 및 다수의 명령에서 사용되는 용어는 다음과 같습니다.

프로그램

프로그램은 CICS가 전체 트랜잭션 또는 트랜잭션의 일부를 처리하는 데 사용합니다. 프로그램은 프로그램 라이브러리에 저장됩니다. CICS 번들에서 사용되는 프로그램 자원 및 프로그램에 대한 자세한 정보는 참조 -> 시스템 정의에서 PROGRAM 자원의 내용을 참조하십시오.

트랜잭션

트랜잭션은 단일 요청에 의해 시작되는 처리 항목입니다. 이 요청은 일반적으로 터미널에서 일반 사용자로부터 비롯되지만 웹 페이지, 원격 워크스테이션 프로그램 또는 다른 CICS 시스템의 애플리케이션에서 수행되거나 사전 정의된 시간에 자동으로 트리거될 수도 있습니다. CICS 트랜잭션을 실행하는 여러 가지 방법에 대한 설명은 제품 개요의 인터넷, TCP/IP 및 HTTP 개념 및 제품 개요에서 CICS 외부 인터페이스의 개요의 내용을 참조하십시오.

단일 트랜잭션은 실행 시 필요한 처리를 수행하는 하나 이상의 애플리케이션으로 구성됩니다.

그러나 트랜잭션이란 용어는 CICS에서 단일 이벤트와 동일 유형의 다른 모든 트랜잭션을 둘 다 의미하는 데 사용됩니다. 각 트랜잭션 유형을 TRANSACTION 자원 정의와 함께 CICS에 설명합니다. 이 정의는 트랜잭션 유형에 이름(트랜잭션 ID 또는 TRANSID)을 제공하고 수행할 작업에 대한 정보를 CICS에 제공합니다(예: 먼저 호출할 프로그램, 트랜잭션 실행 전반에서 필요한 인증 유형).

CICS에 TRANSID를 제출하여 트랜잭션을 실행합니다. CICS는 TRANSACTION 정의에 기록된 정보를 사용하여 올바른 실행 환경을 확립하고 첫 번째 프로그램을 시작합니다.

트랜잭션이란 용어는 이제 IT 업계에서 복구 단위(CICS에서는 작업 단위라고 함)를 설명하는 데 광범위하게 사용됩니다. 이는 일반적으로 복구 가능한 전체 조작입니다. 프로그래밍된 명령 또는 시스템 장애의 결과로 전체로 확약 또는 백아웃할 수 있습니다. 많은 경우에 CICS 트랜잭션의 범위는 단일 작업 단위이기도 하지만 비CICS 문서를 사용하는 경우 의미 차이를 알아 두십시오.

태스크 태스크란 용어는 CICS 및 CICS 문서에서 특정 의미를 갖습니다. CICS는 트랜잭션 실행 요청을 수신하면 트랜잭션 유형 실행의 이 한 인스턴스와 연관된 새 태스크를 시작합니다(즉, 일반적으로 특정 터미널에서 특정 사용자를 대신 하여 특정 데이터 세트와 함께 하나의 트랜잭션 실행). 태스크를 스레드와 유사한 것으로 간주할 수도 있습니다. 트랜잭션이 완료되면 태스크가 종료됩니다.

CICS 프로그래밍

다른 프로그램과 비슷한 방법으로 CICS 프로그램을 작성합니다. COBOL, C, C++, Java, PL/I 또는 어셈블러 언어를 사용하여 CICS 애플리케이션을 작성할 수 있습니다. 대부분의 처리 논리는 표준 언어 명령문으로 표현되지만 CICS 명령 또는 Java 및 C++ 클래스 라이브러리를 사용하여 CICS 서비스를 요청할 수 있습니다.

이 정보는 COBOL, C, C++, PL/I 또는 어셈블리 언어 프로그램에서 사용될 수 있는 CICS 명령 레벨 프로그래밍 인터페이스인 **EXEC CICS** 사용을 설명합니다. 이러한 명령은 애플리케이션 개발 참조서에서 자세히 정의되어 있습니다. 다음 프로그래밍 정보도 사용할 수 있습니다.

- JCICS 클래스 라이브러리를 사용한 Java 프로그래밍은 애플리케이션 개발에서 JCICS를 사용하는 Java 프로그래밍에 설명되어 있습니다.
- CICS C++ 클래스를 사용한 C++ 프로그래밍은 애플리케이션 개발에서 CICS C++ OO 기반 클래스 사용에 설명되어 있습니다.
- HTTP 요청 및 응답 처리를 위한 웹 애플리케이션 작성에 대한 정보는 애플리케이션 개발에서 HTTP 애플리케이션 개발의 내용을 참조하십시오.

CICS에서 언어 사용에 대한 추가 지침은 11 페이지의 제 2 장 『프로그래밍 언어 및 Language Environment』의 내용을 참조하십시오.

CICS 명령은 물론 SQL문, DLI 요청, CPI문, CICS FEPI(Front End Programming Interface) 명령을 프로그램에서 사용할 수 있습니다. 관련 세부사항은 다음 정보를 참조하십시오.

- SQL을 사용하려면 z/OS용 Information Management Software 제품 문서의 *z/OS용 DB2 reference: DB2 SQL* 및 *z/OS용 DB2 Application Programming and SQL* 을 참조하십시오.
- DL/I를 사용하려면 z/OS용 Information Management Software 제품 문서의 *IMS: Application programming for EXEC DLI* 및 *IMS: Application programming design* 을 참조하십시오.
- CPI를 사용하려면 z/VM Application Programming의 내용을 참조하십시오.
- FEPI를 사용하려면 애플리케이션 개발에서 FEPI 애플리케이션 개발의 내용을 참조하십시오.
- WebSphere MQ를 사용하려면 WebSphere MQ Library의 내용을 참조하십시오.

관련 정보:

 참조 > 시스템 프로그래밍의 시스템 명령

CICS 프로그래밍 명령

CICS 명령의 일반 형식은 필수 명령 이름과 하나 이상의 옵션이 뒤에 오는 EXECUTE CICS(또는 EXEC CICS)입니다.

CICS 제어 블록과 저장영역의 필드에 대한 지식 또는 참조 없이 CICS 명령 레벨을 사용하여 많은 애플리케이션을 작성할 수 있습니다. 그러나 애플리케이션의 로컬 환경 외부의 올바른 정보를 가져와야 합니다. ADDRESS 및 ASSIGN 명령을 사용하여 이러한 정보에 액세스할 수 있습니다.

ADDRESS 및 ASSIGN 명령을 사용할 때 다양한 필드를 읽을 수 있지만 다른 방법으로 설정하거나 사용해서는 안됩니다. EXEC 인터페이스 모듈이 CICS 필드를 변경할 수 있으므로 이러한 필드를 CICS 명령의 인수로 사용하지 마십시오.

애플리케이션에 INQUIRE, SET 및 PERFORM 명령을 사용하여 CICS 자원에 대한 정보에 액세스할 수 있습니다. 이러한 명령은 시스템 프로그래밍 명령으로 알려져 있습니다. 애플리케이션은 CICS 데이터 세트, 터미널, 시스템 항목, 모드 이름, 시스템 속성, 프로그램, 트랜잭션에 대한 정보를 검색하고 수정할 수 있습니다. 이러한 명령과 JES(Job Entry Subsystem)에 대한 CICS 인터페이스의 스푼 명령은 주로 시스템 프로그래머가 사용합니다.

CICS 프로그래밍 명령에 대한 참조 정보는 참조 -> 애플리케이션 개발에서 CICS API 명령의 내용을 참조하십시오.

실행 인터페이스 블록(EIB)

일반적인 CICS 제어 블록 이외에 명령 레벨 환경의 각 태스크에는 이와 연관된 EXEC 인터페이스 블록(EIB)으로 알려진 제어 블록이 있습니다.

애플리케이션은 이름으로 EIB의 모든 필드에 액세스할 수 있습니다. EIB는 애플리케이션 실행 중 유용한 정보(예: 트랜잭션 ID, 시간 및 날짜(처음에는 태스크가 시작될 때 표시되고 나중에는 ASKTIME을 사용하여 애플리케이션이 갱신된 경우 표시됨), 표시 장치의 커서 위치)를 포함합니다. 또한 EIB는 덤프를 사용하여 프로그램을 디버그할 때 도움이 되는 정보를 포함합니다. EIB 필드에 대한 프로그래밍 정보는 CICS 애플리케이션 프로그래밍 참조서의 내용을 참조하십시오.

변환

일부 기존 컴파일러(및 어셈블러)는 CICS 명령을 직접 처리할 수 없습니다. 프로그램을 실행 코드로 변환하려면 추가 단계가 필요합니다. 이 단계를 변환이라고 하며, 컴파일러(또는 어셈블러)가 이해할 수 있도록 CICS 명령을 나머지 프로그램이 코드화된 언어로 변환하는 것으로 이루어집니다.

대부분의 컴파일러는 CICS 변환기 접근방식을 사용하는데, 여기서 컴파일러는 컴파일 시 CICS와 상호작용하여 CICS 명령을 해석하고 CICS 서비스 루틴에 대한 호출로 자동 변환합니다. 통합 CICS 변환기 접근방식을 사용하는 경우 다수의 변환 태스크는 컴파일 시 수행되며 추가 변환 단계를 완료하지 않아도 됩니다. 변환 단계의 태스크에 대한 자세한 정보는 96 페이지의 『변환 프로세스』의 내용을 참조하십시오.

CICS 테스트

프로그램에는 CICS 환경에서 실행 중인지 여부를 판별하기 위한 두 가지 방법이 있는데, C 언어 iscics() 함수를 사용하거나 DFH3QSS 프로그램을 호출하는 것입니다.

이 태스크 정보

iscics

기존 C 언어 프로그램을 조정하거나 CICS에서는 물론 CICS 외부에서도 실행되도록 설계된 새 프로그램을 작성하는 경우, C 언어 iscics() 함수는 유용성을 입증할 수 있습니다. 프로그램이 CICS에서 현재 실행 중인 경우 0이 아닌 값을 리턴하고 그렇지 않으면 0을 리턴합니다. 이 함수는 C 라이브러리의 확장입니다.

DFH3QSS

프로그램은 DFH3QSS 프로그램을 호출하여 CICS 환경 및 API 기능을 조회할 수 있습니다. DFH3QSS를 자체 애플리케이션에 정적으로 링크하십시오. 리턴 시 레지스터 15는 자신을 포함하는 반자 길이 다음에 예약 반자(현재 0)가 오고 그 뒤에 비트 문자열이 오는 결과 구조를 주소 지정합니다.

비트 0 1로 설정된 경우, 이는 호출자가 CICS 관리 TCB 또는 해당 하위 중 하나의 CICS 환경에서 실행 중임을 의미합니다.

비트 1 1로 설정된 경우, 이는 호출자가 현재 PSW 키, ASC 모드, AMODE 및 교차 메모리 환경에서 CICS API를 사용할 수 있음을 의미합니다.

요청이 실행된 TCB가 종료되지 않았고 DFH3QSS 자체가 가상 저장영역에 여전히 존재하는 한 출력 구조에 계속 액세스할 수 있습니다. 실행 상태(예: PSW 키, ASC 모드, AMODE 또는 교차 메모리 환경) 변경은 CICS API 가용성에 영향을 줄 수 있습니다. 레지스터는 보존됩니다.

Environment

문제점 상태에서 이 기능을 호출해야 합니다.

CICS 프로그래밍 로드맵

이 로드맵에 있는 단계에 따라 EXEC CICS 명령 레벨 프로그래밍 인터페이스를 사용하는 CICS 애플리케이션을 개발합니다.

프로시저

1. 애플리케이션을 디자인하고 사용할 CICS 자원과 서비스를 식별하십시오. CICS 애플리케이션 디자인에 관한 지침은 267 페이지의 제 18 장 『애플리케이션 디자인』 및 303 페이지의 『성능 향상을 위한 디자인』의 내용을 참조하십시오.
2. CICS 서비스를 요청하는 **EXEC CICS** 명령을 포함하여 선택한 언어로 프로그램을 작성하십시오. CICS 명령 목록은 *CICS 애플리케이션 프로그래밍 참조서*의 내용을 참조하십시오.

3. 프로그램을 변환하고 컴파일하십시오. 93 페이지의 『통합 CICS 변환기』와 통합하는 컴파일러를 사용 중인 경우, 143 페이지의 제 9 장 『애플리케이션 설치』에 설명된 프로세스를 사용하여 프로그램을 컴파일한 후 CICS에 설치하면 됩니다. 그렇지 않으면, 111 페이지의 『CICS 변환기 사용』에 설명된 프로세스를 사용하여 프로그램의 변환기 옵션을 정의한 후 프로그램을 변환 및 컴파일하고 143 페이지의 제 9 장 『애플리케이션 설치』에 설명된 프로세스를 사용하여 CICS에 설치해야 합니다.
4. CICS에 프로그램 및 관련 트랜잭션을 정의하십시오. *CICS* 자원 정의 안내서에 설명된 대로 PROGRAM 자원 정의 및 TRANSACTION 자원 정의를 사용하십시오.
5. 프로그램이 사용하는 CICS 자원(예: 파일, 큐 또는 터미널)을 정의하고 설치하십시오.
6. 프로그램을 실행하십시오. CICS 터미널에서 트랜잭션 ID를 입력하거나 *CICS* 외부 인터페이스 안내서 또는 기타 시스템에 CICS 연결에 설명된 방법을 사용하십시오.

제 2 부 프로그래밍 언어 및 Language Environment

Language Environment®에 대한 프로그래머용 정보와 COBOL, C 및 C++, PL/I, 어셈블러 언어로 프로그래밍하기 위한 언어 특정 지침입니다.

제 2 장 프로그래밍 언어 및 Language Environment

z/OS의 요소로 제공되는 Language Environment는 공통 런타임 라이브러리 세트를 제공합니다. 대부분의 시스템 종속성은 제거되었으므로 프로그래밍 언어 또는 시스템 자원 요구에 관계없이 Language Environment를 사용하면 애플리케이션에 하나의 런타임 환경만 사용할 수 있습니다.

Language Environment가 도입되기 전에, 각 고급 언어(HLL)는 별도의 런타임 환경을 제공했습니다. Language Environment가 제공하는 런타임 라이브러리는 VS COBOL II, OS PL/I, C/370™과 같은 기존 컴파일러와 함께 제공된 런타임 라이브러리를 대체합니다. 공통 환경은 두 가지 중요한 장점을 제공합니다.

- CICS가 지원하는 모든 언어를 단일 프로그램에서 혼합할 수 있습니다.
- 동일한 Language Environment 호출 가능 서비스는 모든 프로그램에서 사용할 수 있습니다. 예를 들어, 다음과 같습니다.
 - PL/I 프로그램에서 Language Environment 호출 가능 서비스를 사용하여 확보한 저장영역으로 작성된 링크된 목록은 나중에 처리할 수 있으며 COBOL 루틴에서 호출 가능 서비스를 사용하여 저장영역을 비울 수 있습니다.
 - 보고서 자체가 COBOL 프로그램을 통해 생성되더라도 일련의 보고서에서 사용되는 통화 기호를 어셈블러 루틴에 설정할 수 있습니다.
 - 다른 언어로 작성된 프로그램의 시스템 메시지는 동일한 출력 대상으로 모두 전송됩니다.

자세한 정보는 *z/OS Language Environment Concepts Guide*를 참조하십시오. 이러한 장점때문에 CICS에서의 고급 언어 지원은 Language Environment에 따라 달라집니다.

CICS는 다양한 컴파일러를 사용하여 컴파일된 애플리케이션을 지원합니다. 이 z/OS용 CICS Transaction Server 릴리스에서 지원되는 컴파일러 목록은 고급 언어 지원을 참조하십시오.

Language Environment는 AMODE(64) 어셈블러 언어 프로그램을 지원하지 않습니다.

CICS 및 Language Environment에서 지원하는 대부분의 컴파일러는 Language Environment 준수 컴파일러이며, 이는 이러한 컴파일러가 컴파일한 프로그램은 CICS 리전에서 사용 가능한 모든 Language Environment 기능의 장점을 이용할 수 있음을 의미합니다. CICS 및 Language Environment는 Language Environment를 준수하지 않는 일부 Language Environment 이전 컴파일러가 컴파일한 프로그램도 지원합니다. 그러나 CICS는 Language Environment가 지원하는 Language Environment 이전 모든 컴파일러를 지원하지 않습니다.

Language Environment 이전 컴파일러로 컴파일 및 링크된 애플리케이션은 Language Environment에서 제공하는 런타임 지원을 사용하여 실행할 수 있습니다. 이러한 애플리케이션에는 재컴파일링 또는 재링크 편집이 필요하지 않습니다. 일부 환경에서는 애플리케이션을 올바르게 실행할 수 있도록 Language Environment 런타임 옵션을 조정해야 합니다. 사용 중인 언어, 추가 정보는 *z/OS Language Environment Run-Time Application Migration Guide* 및 컴파일러 및 런타임 마이그레이션 안내서를 참조하십시오.

Language Environment 이전 컴파일러와 함께 제공된 런타임 라이브러리는 지원되지 않습니다. CICS 시동 JCL에 Language Environment 라이브러리 이외의 다른 언어 라이브러리를 포함시키지 마십시오.

기존 애플리케이션을 수정하거나 새 프로그램을 작성할 경우 Language Environment에서 지원하는 컴파일러를 사용해야 합니다. 애플리케이션은 Language Environment SCEELKED 라이브러리를 사용하여 링크 편집되어야 하며, 이는 결과 애플리케이션 로드 모듈을 Language Environment에서만 실행할 수 있음을 의미합니다.

CICS에서 Language Environment를 준수하는 어셈블러 MAIN 프로그램도 작성할 수 있습니다. 어셈블러 프로그램에 대한 자세한 정보는 71 페이지의 제 6 장 『어셈블러 언어 애플리케이션 개발』의 내용을 참조하십시오.

Language Environment 호출 가능 서비스

Language Environment는 CICS에서 실행 중인 프로그램이 액세스할 수 있는 호출 가능 서비스를 제공합니다.

Language Environment가 제공하는 호출 가능 서비스는 다음 범주로 분류됩니다.

저장영역 서비스

Language Environment 힙에서 저장영역을 할당하고 비울 수 있게 합니다.

오류 처리 서비스

오류를 처리할 수 있도록 정보를 얻는 일반적인 방법을 제공합니다.

메시지 서비스

메시지를 처리하고 발행하는 일반적인 방법을 제공합니다.

날짜 및 시간

날짜 및 시간을 나타내는 값 읽기, 계산, 쓰기를 가능하게 합니다. Language Environment는 입력 레코드에 포함되거나 운영 체제 서비스가 생성한 날짜 및 시간 형식을 거의 처리할 수 있게 하는 고유 패턴 일치 기능을 제공합니다.

자국어 지원

주어진 국가에 맞게 Language Environment 출력(예: 메시지, RPTOPTS 보고서, RPTSTG 보고서, 덤프)을 사용자 정의할 수 있게 합니다.

로케일

로케일 이름을 지정하여 주어진 자국어, 국가, 코드 세트에 맞게 문화적으로 민감한 출력을 사용자 정의할 수 있게 합니다.

일반

특정 Language Environment 기능(예: 덤프)과 직접적으로 관련되지 않은 호출 가능 서비스 세트입니다.

수학

표준 수학적 계산을 수행할 수 있게 합니다.

이러한 서비스는 일반적으로 Language Environment 준수 컴파일러를 사용하여 컴파일된 프로그램에만 사용할 수 있습니다. 예외로, VS COBOL II 프로그램은 날짜 및 시간 호출 가능 서비스를 동적으로 호출할 수 있지만 Language Environment 호출 가능 서비스에 대한 다른 동적 호출이나 정적 호출을 수행할 수 없습니다.

이러한 서비스의 세부사항에 대한 추가 정보는 *z/OS Language Environment Programming Guide*를 참조하십시오. 서비스 호출에 필요한 구문에 대한 정보는 *z/OS Language Environment Programming Reference*를 참조하십시오.

메시지 및 덤프 서비스

Language Environment 서비스 CEEMOUT(메시지 디스패치) 및 CEE3DMP(덤프 생성)가 CICS에서 실행되는 경우, 메시지와 덤프는 둘 다 CESE라는 트랜지언트 데이터 큐로 전송되며 일반적인 대상으로 전송되지 않습니다.

Language Environment 메시지 및 덤프의 일반적인 대상은 메시지의 경우 MSGFILE 런타임 옵션에 지정된 *ddname*이고 덤프의 경우 CEE3DMP 서비스의 *fname* 인수에 지정된 *ddname*입니다. CICS는 이 *ddname*을 둘 다 무시합니다.

Language Environment 이상 종료 및 조건 처리

Language Environment 이상 종료 처리는 CICS HANDLE ABEND 사용에 따라 다릅니다. CICS HANDLE ABEND가 활성화되지 않은 경우 사용자 작성 조건 핸들러를 사용할 수 있습니다. Language Environment는 CICS 정의 예외 조건 처리 또는 어텐션 ID(AID) 발견에 관련되지 않습니다.

이상 종료 처리

CICS 애플리케이션이 Language Environment에서 실행 중일 때 비정상 종료에 대해 태스크가 스케줄된 경우 수행되는 조치는 CICS HANDLE ABEND 활성화 여부에 따라 다릅니다.

- HANDLE ABEND가 활성화된 경우 CICS HANDLE ABEND에 정의된 조치가 발생합니다. Language Environment 조건 처리는 이상 종료 또는 프로그램 인터럽트에 대한 제어를 획득하지 않으며 CEEHDLR이 확립한 사용자 작성 조건 핸들러는 무시합니다.
- CICS HANDLE ABEND가 활성화되지 않은 경우 런타임 옵션 TRAP(ON)이 지정되어 있으면 Language Environment 조건 처리는 이상 종료 및 프로그램 인터럽트에 대한 제어를 획득합니다. 그러면 정상 Language Environment 조건 처리가 수행됩니다. TRAP(OFF)가 지정된 경우 오류 처리가 발생하지 않고 이상 종료가 진행됩니다. 정상 Language Environment 조건 처리에 대한 세부사항은 *z/OS Language Environment Programming Guide*를 참조하십시오.

사용자 작성 Language Environment 조건 핸들러

Language Environment 런타임 옵션 USRHDLR을 사용하여 최상위 레벨에서 사용자 작성 조건 핸들러를 등록할 수 있습니다. 하위 레벨에서는(예: 서브루틴 CALL 이후) CEEHDLR 서비스를 사용하여 해당 레벨에 대해 조건 핸들러를 등록할 수 있습니다. 이 하위 레벨 핸들러는 하위 레벨에서 리턴 시 자동으로 등록 해제됩니다. CEEHDLU 서비스를 사용하여 명시적으로 등록 해제할 수 있습니다. 스택 레벨에 대한 설명, USRHDLR 런타임 옵션과 CEEHDLR 및 CEEHDLU 서비스에 대한 세부사항은 *z/OS Language Environment Programming Guide*를 참조하십시오.

COBOL로 작성되지 않은 사용자 작성 Language Environment 조건 핸들러를 작성하는 경우, 대부분의 CICS 명령(NOHANDLE, RESP 또는 RESP2 옵션을 사용하여 코드화된 경우)을 사용하여 조건 핸들러 실행 중에 추가 조건이 발생하는 것을 방지할 수 있습니다. 사용할 수 없는 유일한 명령은 다음과 같습니다(조건 핸들러 또는 핸들러가 호출하는 프로그램에 나타나지 않아야 함).

- ABEND
- HANDLE ABEND
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION
- POP HANDLE
- PUSH HANDLE

NOLINKAGE 변환기 옵션을 사용하는 경우가 아니면 CICS 변환기를 사용하여 CEEHDLR 서비스를 사용하는 루틴에 대해 등록한 COBOL 사용자 작성 조건 핸들러를 변환하지 마십시오. 그 이유는 CICS 변환기가 COBOL 프로그램의 PROCEDURE DIVISION 헤더에 두 개의 추가 인수 즉, 실행 인터페이스 블록(EIB)과 COMMAREA를 추가하기 때문입니다. 이러한 인수는 Language Environment가 전달하는 인수와 일치하지 않습니다. 따라서 COBOL 조건 핸들러는 CICS 명령을 포함할 수 없습니다.

그러나 사용자 작성 조건 핸들러는 서브루틴을 호출하여 CICS 명령을 수행할 수 있으며 이는 COBOL 루틴이 될 수 있습니다. 이 서브루틴에 인수를 전달해야 하는 경우 호출자에서 앞에 두 개의 더미 인수를 배치하십시오. 호출된 서브루틴은 다른 CICS 명령을 실행하기 전에 EXEC CICS ADDRESS EIB(DFHEIPTR)을 실행해야 합니다.

애플리케이션이 사용자 작성 Language Environment 조건 핸들러를 사용하려면 런타임 시 조건 핸들러를 사용할 수 있어야 합니다(예: STEPLIB 연결 또는 LPA 사용). 프로그램 자동 설치를 사용하지 말고 CICS 리전에 대해 CICS 시스템 정의 데이터 세트(CSD)에 이러한 조건 핸들러를 정의하십시오. 여기에는 샘플 사용자 정의 조건 핸들러 CEEWUCHA가 포함됩니다.

Language Environment 조건 처리 루틴의 필수 인터페이스에 대한 상세 정보는 *z/OS Language Environment Programming Guide*를 참조하십시오.

CICS 조건 및 어텐션 ID(AID) 처리

Language Environment 조건 처리는 CICS HANDLE CONDITION 또는 HANDLE AID 명령을 사용하는 애플리케이션의 작동을 변경하지 않습니다. Language Environment는 CICS에 의해서만 발생되고 처리되는 CICS 정의 예외 조건 처리에 관련되지 않습니다. 마찬가지로 AID 발견은 Language Environment가 영향을 주지 않는 CICS 기능입니다.

Language Environment 저장영역

Language Environment는 각 실행 단위에 CICS에서 얻은 저장영역을 사용합니다. 각 프로그램을 처음 사용할 때 Language Environment는 실행 단위 작업 영역(RUWA)이 필요로 하는 저장영역을 CICS에 알려줍니다. 저장영역 할당은 CICS 시스템 초기화 매개변수 RUWAPool의 설정에 따라 다릅니다.

RUWAPool=NO를 지정하면, 각 CICS 링크 레벨 시작 시 CICS가 이 저장영역에 대해 GETMAIN을 실행하고 제어 블록과 저장영역(예: STACK, LIBSTACK, HEAP)에 사용하도록 Language Environment에 전달합니다. 트랜잭션에 지정된 기본 키에서 저장영역을 획득합니다. 프로그램이 종료되면 FREEMAIN을 사용하여 저장영역을 비웁니다.

RUWAPool=YES를 지정하면, 트랜잭션 최초 실행은 RUWAPool=NO의 경우와 같지만 CICS가 트랜잭션을 실행하도록 요청된 총 저장영역 히스토리를 RUWA에 대해 보존합니다. 이는 트랜잭션이 다시 실행될 때 CICS가 총 저장영역에 대해 단일 GETMAIN(및 태스크 종료 시 FREEMAIN)을 실행하고 RUWAPool을 작성함을 의미합니다. 트랜잭션이 동일한 경로를 따를 경우 CICS는 RUWAPool에서 저장영역을 할당하며, 추가 GETMAIN을 실행하지 않아도 됩니다. 다른 또는 추가 CICS 링크 때문에 RUWA에 저장영역이 더 필요하면 CICS가 GETMAIN을 실행하고 히스토리

를 갱신하므로 다음 번 단일 GETMAIN(및 FREEMAIN)은 양이 많은 경우입니다. CICS LINK 명령을 아주 많이 실행하는 트랜잭션의 경우 성능 향상을 고려할 수 있습니다.

CICS 시스템 초기화 매개변수 AUTODST=YES를 지정하면, CICS가 동적 저장영역 튜닝을 지원할 수 있음을 Language Environment에 표시합니다.

프로그램이 ALL31(OFF) 런타임 옵션을 지정하고 Language Environment가 16MB 라인 이하의 저장영역을 사용해야 하는 경우 두 개의 저장영역이 할당되는데, 하나는 16MB 라인 이하이고 하나는 16MB 라인 이상입니다.

필요하면 애플리케이션이 CICS GETMAIN 명령을 사용하여 CICS DATAKEY 또는 USER DATAKEY 저장영역을 얻을 수 있습니다. 그러나 EXEC KEY가 USER인 프로그램은 CICS DATAKEY 저장영역을 사용할 수 없습니다.

Language Environment에서 언어 혼합

Language Environment를 사용하여 여러 고급 소스 언어와 어셈블러 언어로 작성된 프로그램으로 구성된 애플리케이션을 빌드할 수 있습니다.

고급 언어(HLL) 프로그램에서 호출된 어셈블러 언어 서브루틴은 상당히 간단하고 공통적입니다. 하나의 HLL에서 호출되지만 다른 언어로 작성된 서브루틴에는 보다 세심한 고려가 필요하며 언어간 통신(ILC)을 수반합니다. Language Environment는 ILC 애플리케이션을 두 개 이상의 HLL과 선택적으로 어셈블러 언어에서 빌드된 애플리케이션으로 정의합니다. 자세한 정보는 z/OS Language Environment Writing Interlanguage Communication Applications의 내용을 참조하십시오.

Language Environment에서는 CICS의 실행 단위에 ILC가 있으면 각 컴파일 단위를 Language Environment 준수 컴파일러를 사용하여 컴파일해야 합니다. CICS가 지원하는 HLL은 다음과 같습니다.

- C/C++
- COBOL
- PL/I

다음 절에서는 각 HLL 쌍에 대한 조건을 설명합니다. 애플리케이션에 두 개의 HLL만 포함된 경우에는 해당 절을 참조하십시오. 애플리케이션에 세 개의 HLL이 모두 포함된 경우에는 애플리케이션의 각 인터페이스에 해당하는 절을 참조하십시오.

C/C++ 및 COBOL

Language Environment가 C/C++ 및 COBOL로 작성된 루틴 간에 ILC를 지원하는 조건은 다음 요인에 따라 다릅니다.

- 언어가 C인지 C++인지 여부.

- 사용되는 COBOL 컴파일러와 DLL이 컴파일러 옵션으로 지정되었는지 여부.
- 호출이 정적인지 동적인지 여부.
- 호출된 함수가 모듈에 있는지 DLL에서 내보냈는지 여부.
- 프로그램 재진입 가능 여부.
- C 프로그램에 있는 #pragma 연계 명령문(있는 경우).
- C 프로그램이 함수를 내보내는지 변수를 내보내는지 여부.
- C++ 프로그램에 있는 extern문(있는 경우).

이러한 요인의 영향에 대한 정보는 z/OS Language Environment Writing Interlanguage Communication Applications의 내용을 참조하십시오.

C/C++ 및 PL/I

CICS에서는 C/C++ 및 PL/I 애플리케이션의 모든 구성요소가 재진입 가능하면 다음과 같이 Language Environment가 C/C++ 및 PL/I로 컴파일된 루틴 간에 ILC를 지원합니다.

- C/C++ 루틴은 PL/I 루틴을 정적으로 호출할 수 있고 PL/I 루틴은 C/C++ 루틴을 정적으로 호출할 수 있습니다.
- C/C++ 루틴은 OPTIONS(FETCHABLE)이 지정된 PL/I 루틴을 fetch()할 수 있습니다. 호출된 루틴에 CICS 명령이 포함된 경우 C/C++는 EIB 및 COMMAREA를 호출문의 처음 두 매개변수로 전달해야 합니다.
- PL/I 루틴은 CICS 변환기가 처리하지 않은 해당 C/C++ 루틴만 FETCH할 수 있습니다. 그 이유는 동적 호출 동안 변환기가 작성한 특정 정적 필드를 올바르게 설정할 수 없기 때문입니다.

COBOL 및 PL/I

CICS에서 Language Environment는 다음과 같이 Language Environment 지원 COBOL 및 PL/I 컴파일러를 사용하여 컴파일된 루틴 간에 ILC를 지원합니다.

- COBOL 루틴은 PL/I 루틴을 정적으로 호출할 수 있고 PL/I 루틴은 COBOL 루틴을 정적으로 호출할 수 있습니다.
- COBOL 프로그램은 OPTIONS(FETCHABLE)이 지정된 PL/I 루틴을 동적으로 호출할 수 있고 PL/I 루틴은 COBOL 프로그램을 FETCH할 수 있습니다.

호출된 루틴에 CICS 명령이 포함된 경우 호출 루틴은 EIB 및 COMMAREA를 CALL 문의 처음 두 매개변수로 전달해야 합니다.

어셈블러 언어

- Language Environment 준수 HLL 프로그램에서 Language Environment 준수 어셈블러 언어 서브루틴으로 정적 또는 동적 호출을 수행할 수 있습니다. 반대로,

Language Environment 준수 어셈블러 언어 루틴은 Language Environment 준수 루틴을 정적으로 호출할 수 있으며 Language Environment 매크로 CEEFETCH 또는 CEELoad를 사용하여 다른 루틴(어셈블러 언어 또는 HLL)을 동적으로 로드할 수 있습니다.

- CEELoad를 사용하여 ILC 모듈을 삭제(해제)할 수 없습니다.
- CEERELES 매크로를 사용하여 CEEFETCH를 사용하여 폐치한 ILC 모듈을 해제할 수 없습니다.
- 어셈블러 언어 루틴을 삭제하려면 루틴을 폐치한 언어를 사용하십시오. 해제할 모듈에 PL/I와의 ILC가 없는 경우에만 C/C++, COBOL, PL/I에서 이를 수행할 수 있습니다.

또한 Language Environment 준수 HLL 프로그램 또는 어셈블러 언어 서브루틴에서 비준수 어셈블러 언어 서브루틴으로 정적 호출을 수행할 수 있습니다. 그러나 비준수 어셈블러 언어 루틴은 Language Environment 준수 루틴을 정적으로 호출할 수 없으며, Language Environment 매크로를 사용할 수 없기 때문에 준수 루틴을 폐치하거나 로드할 수 없습니다.

어셈블러 언어가 C 또는 C++를 호출하려면 다음 명령문을 포함시켜야 합니다.

C #pragma linkage(,0S)

C++ extern "0S"

DL/I

CICS의 ILC 애플리케이션에서 DL/I를 사용하는 경우, 기본 프로그램과 동일한 언어로 작성된 프로그램에서만 EXEC DLI문 또는 CALL xxxTDLI를 사용하여 DL/I를 호출할 수 있습니다.

Language Environment는 CICS에서 CALL CEETDLI를 지원하지 않습니다.

동적 링크 라이브러리(DLL)

z/OS 동적 링크 라이브러리(DLL) 기능은 프로그램과 데이터를 별도의 다른 로드 모듈에서 액세스할 수 있는 로드 모듈(DLL)로 패키징하는 메커니즘을 제공합니다.

DLL은 DLL 외부에서 호출할 수 있는 루틴을 나타내는 기호를 내보낼 수 있으며, 대상 루틴을 참조 루틴과 동일한 로드 모듈에 링크할 필요가 없도록 다른 DLL에서 루틴이나 데이터 또는 둘 다를 나타내는 기호를 가져올 수 있습니다. 애플리케이션이 별도의 DLL을 처음 참조할 때 시스템이 메모리에 DLL을 자동으로 로드합니다.

모든 잠재적 DLL 실행 가능 모듈을 CICS에 PROGRAM 자원으로 정의해야 합니다.

*z/OS Language Environment Programming Guide*에 나열된 컴파일러를 사용하여 코드가 컴파일된 CICS 애플리케이션에 DLL 지원을 사용할 수 있습니다. DLL 빌드 및 사용에 대한 자세한 정보는 해당 매뉴얼을 참조하십시오.

Language Environment 런타임 옵션 정의

Language Environment는 프로그램 처리를 제어하기 위한 런타임 옵션을 제공합니다. CICS에서는 특정 프로그램 실행에 적용되는 옵션이 프로그램뿐만 아니라 프로그램 실행 방법에 따라서도 다릅니다.

Java 프로그램과 웹에서 시작된 프로그램은 Language Environment 초기화 전 모듈인 CEEPIPI를 사용합니다. 여기에는 자체 버전의 CEEDOPT CSECT가 있으며 이러한 프로그램은 이 CSECT에서 런타임 옵션을 가져옵니다.

정상 CICS 태스크(예: 터미널에서 시작된 태스크)의 경우, 나열된 다음 방법 중 하나를 사용하여 Language Environment 런타임 옵션을 설정하십시오. Language Environment 런타임 옵션의 전체 우선순위에 대한 자세한 정보는 *z/OS V1R11.0 Language Environment 프로그래밍 안내서 SA22-7561-10*을 참조하십시오. 방법은 처리 순서로 표시되어 있습니다. 다음 설정으로 각 설정을 덮어쓸 수 있습니다. 이는 우선순위 역순으로 적용됩니다.

1. CEECCICS에 빌드된 CEEDOPT CSECT는 IBM Language Environment 기본 런타임 옵션을 포함합니다. SCEESAMP에 있는 CEEWCOPT 샘플 작업을 사용하여 이러한 기본 런타임 옵션을 변경할 수 있습니다. 이 옵션이 지원되긴 하지만 CEEPRMxx parmlib 멤버를 사용하여 런타임 옵션을 지정하는 것을 선호하며 가장 쉬운 방법입니다.
2. CEEPRMxx parmlib 멤버는 CICS에 대해 Language Environment 런타임 옵션을 설정하기 위한 선호 방법인 CEECOPT 옵션 그룹에 대한 지원을 제공합니다.
3. 리전 전체 기본 옵션이 있는 CEEROPT CSECT에서. 이 CSECT는 동일 이름의 로드 모듈에 링크 편집되며 CICS 작업에 대한 DFHRPL 라이브러리 연결에서 데이터 세트에 배치됩니다.
4. 사용자 대체 가능 프로그램 DFHAPXPO(XPLINK 프로그램에만 적용됨).
5. 사용자 제공 애플리케이션 레벨 런타임 옵션이 있는 CEEUOPT CSECT에서. 이 CSECT는 애플리케이션 자체에 링크됩니다.
6. 다음과 같이 프로그래밍 언어 옵션 명령문을 사용하는 애플리케이션 소스 코드에서.
 - 프로그램 소스의 #pragma runopts문을 통해 C 프로그램에서. 예를 들어, 다음과 같습니다.


```
#pragma runopts(rptstg(on))
```
 - 프로그램 내 PLIXOPT 선언문을 통해 PL/I 프로그램에서. 예를 들어, 다음과 같습니다.

```
DECLARE PLIXOPT CHARACTER(18) VARYING STATIC EXTERNAL
INIT('RPTOPTS(ON) NOSTAE');
```

참고: COBOL 프로그램 내에서 또는 C++ 프로그램 내에서 런타임 옵션 설정을 허용하는 소스 코드 메커니즘은 없습니다.

7. 디버깅 프로파일에 지정된 Language Environment 옵션에서, 자세한 정보는 255 페이지의 『디버깅 프로파일』의 내용을 참조하십시오.

대부분의 설치에서 애플리케이션 프로그래머는 이전 목록의 첫 번째 방법을 사용할 수 없으며 두 번째는 종종 사용할 수 없습니다. 그러나 애플리케이션 프로그래머는 방법 4 또는 방법 5를 사용할 수 있습니다. 하나의 방법만 선택하십시오. 방법 4와 방법 5를 둘 다 사용하려고 시도하지 마십시오. CEEUOPT CSECT를 생성하여 애플리케이션에 링크하는 방법에 대한 정보는 *z/OS Language Environment Customization*를 참조하십시오.

이후 스펙이 덮어쓸 수 없도록 CEEDOPT와 CEEROPT 둘 다 옵션을 설정할 수 있습니다.

Language Environment 런타임 옵션 지정 방법 및 해당 의미에 대한 자세한 정보는 *z/OS Language Environment Programming Reference*를 참조하십시오.

CICS에서 무시되는 런타임 옵션

CICS에서는 상당수의 Language Environment 런타임 옵션이 무시됩니다. 여기에는 모든 Fortran 전용 옵션과 다음이 포함됩니다.

- ABPERC
- AIXBLD
- CBLOPTS
- CBLQDA
- DEBUG
- EXECOPS
- INTERRUPT
- LIBRARY
- MSGFILE
- NONIPTSTACK
- PLITASKCOUNT
- POSIX(XPLINK 또는 Java 프로그램이 아닌 경우)
- RTEREUS
- RTLS
- SIMVRD

- THREADHEAP
- VERSION

사용된 런타임 옵션 판별

프로그램이 실행될 때 적용된 Language Environment 런타임 옵션을 알려면 RPTOPTS(ON) 옵션을 지정하십시오. 프로그램이 종료되면 사용된 모든 런타임 옵션 목록이 생성됩니다. 목록은 CESE TD 큐에 기록됩니다. 목록은 실제 옵션 설정은 물론 기점 즉, 설치 기본값인지 리전 기본값인지 여부 또는 프로그래머가 설정했는지 엑시트 중 하나에 설정되었는지 여부도 포함합니다.

참고: 프로덕션 환경에서는 RPTOPTS(ON)을 사용하지 마십시오. 상당한 오버헤드가 발생하며 이로 인해 많은 양의 데이터가 CESE 큐에 기록될 수 있습니다.

하위 엔클레이브의 런타임 옵션: 성능 가이드

CICS에서 CICS LINK 명령을 실행하면 Language Environment가 하위 엔클레이브를 호출합니다. 새 환경이 초기화되고 하위 엔클레이브는 런타임 옵션을 가져옵니다. 이러한 런타임 옵션은 엔클레이브 작성 시 있던 해당 옵션과 관계가 없습니다.

빈번한 EXEC CICS LINK 실행과 많은 런타임 옵션의 개별 설정은 성능에 영향을 줄 수 있습니다. 정적 또는 동적 호출은 이러한 오버헤드를 초래하지 않습니다. CEEUOPT를 사용하여 옵션을 지정해야 하는 경우, 기본값과 다른 해당 옵션만 지정하면 성능이 향상됩니다.

CICS XCTL 명령이 실행될 때 비슷한 상황이 발생합니다. 이 경우, 하위 엔클레이브를 가져오지 않지만 기존 엔클레이브가 종료된 후 새 프로그램에 대해 판별된 런타임 옵션을 사용하여 다시 초기화됩니다. 동일한 성능 가이드가 적용됩니다.

CEEBXITA 및 CEECSTX 사용자 엑시트

이 두 Language Environment 사용자 엑시트는 일부 Language Environment 런타임 옵션을 변경할 수 있습니다.

- CEEBXITA 사용자 엑시트의 CEEAUE_A_OPTION 리턴 매개변수를 설정하면 LIBRARY, RTLS, STACK 및 VERSION 옵션을 제외한 옵션을 변경할 수 있습니다.
- 저장영역 튜닝 사용자 엑시트 CEECSTX에서 STACK, LIBSTACK, HEAP, ANYHEAP 및 BELOWHEAP 옵션을 설정할 수 있습니다.

엑시트는 위에 나열된 순서로 호출됩니다.

저장영역 튜닝 엑시트 CEECSTX는 CEEROPT CSECT와 마찬가지로 리전 전체에 적용되지만 CEEBXITA는 모든 프로그램에 링크됩니다.

Language Environment는 CEEBXITA 어셈블러 엑시트를 호출하는데 그 이유는 CEECSTX와 마찬가지로 환경이 완전히 확립되기 전에 호출되기 때문입니다. 따라서 어셈블러 언어로 코드화해야 합니다.

Language Environment는 SCEESAMP 라이브러리에 샘플 소스 버전의 CEEBXITA를 제공합니다(어떤 이유로 호출되었건 호출자에게 리턴). 이를 있는 그대로 사용하거나 설치 기본 버전으로 사용하도록 수정할 수 있습니다. 그러나 특별히 조정된 버전의 CEEBXITA를 애플리케이션에 링크 편집할 수 있으며 그러면 이 버전이 설치 기본 버전 대신 사용됩니다. CEEBXITA는 프로그램 실행 과정에서 최대 다섯 가지의 서로 다른 이유로 호출되기 때문에 이 방법을 선택하는 경우 세심한 주의를 기울여야 하며 애플리케이션 특정 버전의 CEEBXITA는 이러한 호출을 모두 처리할 수 있어야 합니다.

자체 버전의 CEEBXITA를 작성하는 경우 어셈블러 언어로 작성할 수 있습니다. NOHANDLE, RESP 또는 RESP2 옵션을 지정하면 여기 나열된 명령을 제외한 모든 CICS 명령을 사용하여 엑시트 실행 중에 조건이 발생하지 않게 할 수 있습니다. CEEBXITA에서 사용할 수 없는 명령 또는 CEEBXITA가 호출한 루틴은 다음과 같습니다.

- ABEND
- HANDLE ABEND
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION
- POP HANDLE
- PUSH HANDLE

CEEBXITA 및 CEECSTX 둘 다에 대한 세부사항은 *z/OS Language Environment Customization*를 참조하십시오.

CICSVAR: CICS 환경 변수

CICS는 CONCURRENCY 및 API 프로그램 속성이 애플리케이션 자체와 밀접하게 연관될 수 있도록 CICSVAR이라는 환경 변수를 제공합니다. Language Environment 런타임 옵션 ENVAR을 사용하여 이 환경 변수를 지정할 수 있습니다.

CICSVAR을 CEEDOPT CSECT에서 사용하여 설치 기본값을 설정할 수 있지만 개별 프로그램에 링크 편집된 CEEUOPT CSECT에서 설정하거나 C 또는 C++ 프로그램의 소스에 #pragma문을 사용하여 설정하거나 PL/I 프로그램에서 PLIXOPT문을 사용하여 설정하는 경우 가장 유용합니다. 예를 들어, 프로그램이 스레드 세이프 표준으로 코드화된 경우 PROGRAM 자원 정의를 변경하지 않고 그와 같이 정의하거나 프로그램 자동 설치 엑시트가 올바른 속성으로 설치할 수 있도록 설치 정의 이름 지정 표준을 준수할 수 있습니다.

Language Environment 준수 컴파일러를 사용하여 프로그램이 컴파일된 경우 CICSVAR을 Language Environment 준수 어셈블러 언어, PL/I, COBOL, C 및 C++ 프로그램(XPLINK 옵션을 사용하여 컴파일된 경우와 사용하지 않고 컴파일된 경우 둘 다)에 사용할 수 있습니다. CICSVAR를 Language Environment 비준수 어셈블러 언어 프로그램 또는 Java 프로그램에 사용할 수는 없습니다.

CICSVAR를 사용하면 표준 RDO 인터페이스를 통해 또는 프로그램 자동 설치를 통해 설치된 PROGRAM 자원 정의의 설정을 덮어씁니다. 프로그램이 처음 실행되기 전에 INQUIRE PROGRAM 명령은 프로그램 정의의 키워드 설정을 표시합니다. 애플리케이션이 한 번 실행된 경우 INQUIRE PROGRAM 명령은 CICSVAR 덮어쓰기가 적용된 설정을 표시합니다.

올바른 CICSVAR 값은 QUASIRENT, THREADSAFE, REQUIRED, OPENAPI입니다.

CICSVAR=QUASIRENT

CONCURRENCY(QUASIRENT) 및 APIST(CICSAPI) 속성이 있는 프로그램이 발생합니다.

CICSVAR=THREADSAFE

RCONCURRENCY(THREADSAFE) 및 APIST(CICSAPI) 속성이 있는 프로그램이 발생합니다.

CICSVAR=REQUIRED

CONCURRENCY(REQUIRED) 및 APIST(CICSAPI) 속성이 있는 프로그램이 발생합니다.

CICSVAR=OPENAPI

CONCURRENCY(REQUIRED) 및 APIST(OPENAPI) 속성이 있는 프로그램이 발생합니다.

다음 예제는 CEEUOPT CSECT에 코드화된 Language Environment 런타임 옵션 ENVAR를 표시합니다.

```
CEEUOPT CSECT
CEEUOPT AMODE ANY
CEEUOPT RMODE ANY
      CEEXOPT ENVAR=('CICSVAR=THREADSAFE')
      END
```

이 코드를 어셈블하고 로드 모듈에 링크 편집할 수 있으며, Language Environment가 지원하는 언어 프로그램과 함께 CEEUOPT 로드 모듈을 함께 링크 편집할 수 있습니다.

또는 C 및 C++ 프로그램의 경우 프로그램 소스 시작 부분에 다른 C 명령문 앞에 다음 명령문을 추가하십시오.

```
#pragma runopts(ENVAR(CICSVAR=THREADSAFE))
```

PL/I 프로그램의 경우 PL/I MAIN 프로시저 명령문 뒤에 다음 명령문을 추가하십시오.

```
DCL PLIXOPT CHAR(25) VAR STATIC EXTERNAL INIT('ENVAR(CICSVAR=THREADSAFE)');
```

Language Environment에 대한 CEEBINT 엑시트

Language Environment에서 실행 중인 모든 프로그램은 CEEBXITA 및 CEECSTX 엑시트 호출 직후 프로그램 초기화 시 CEEBINT라는 서브루틴을 호출합니다. 런타임 환경은 이 시점에서 완전히 작동합니다. Language Environment는 이 프로그램을 고급 언어(HLL) 사용자 엑시트라고 합니다.

Language Environment는 SCEELKED 라이브러리에 이 프로그램이 포함된 모듈을 제공하며(호출자에게 리턴), 따라서 이는 설치 기본 버전입니다. 그러나 조정된 버전을 작성하고 프로그램에 링크 편집하여 기본값을 대체할 수 있습니다.

일반적인 Language Environment 코딩 규칙이 CEEBINT에 적용되며, C, C++, PL/I 또는 Language Environment 준수 어셈블러 언어로 이를 작성할 수 있습니다. CEEBINT는 다른 것과 마찬가지로 COBOL 프로그램에 적용되지만 COBOL로 작성하거나 COBOL 프로그램을 호출할 수는 없습니다. CEEBINT가 프로그램에 두 번째 HLL을 도입하는 경우 16 페이지의 『Language Environment에서 언어 혼합』에 설명된 HLL 혼합 규칙이 적용됩니다.

CEEBINT에 대한 자세한 정보는 *z/OS Language Environment 프로그래밍 안내서*를 참조하십시오.

제 3 장 COBOL 애플리케이션 개발

이 정보를 사용하면 CICS 애플리케이션으로 사용할 COBOL 프로그램을 코드화하고 변환하고 컴파일하는 데 도움이 될 수 있습니다.

에서는 z/OS용 CICS Transaction Server, 버전 5 릴리스 3 에 의해 지원되는 COBOL 컴파일러 및 z/OS에서의 서비스 상태를 나열합니다.

z/OS용 CICS Transaction Server, 버전 5 릴리스 3 문서에서 COBOL에 대한 모든 참조는 지원되는 Language Environment(z/OS용 Enterprise COBOL과 같이 이 Language Environment에 부합하는 컴파일러)의 사용을 의미합니다. z/OS용 CICS Transaction Server, 버전 5 릴리스 3 의 런타임 지원이 있고 Language Environment 에는 부합되지 않는 유일한 COBOL 컴파일러는 VS COBOL II 컴파일러입니다.

COBOL 컴파일러 간 마이그레이션에 대한 정보는 Enterprise COBOL for z/OS Compiler and Runtime Migration Guide의 내용을 참조하십시오.

VS COBOL II에 대한 지원

z/OS용 CICS Transaction Server, 버전 5 릴리스 3 에서 VS COBOL II 컴파일러로 컴파일된 애플리케이션은 Language Environment 런타임 라이브러리 루틴을 사용하여 실행됩니다. VS COBOL II에 제공된 런타임 라이브러리는 지원되지 않습니다.

31 페이지의 『VS COBOL II 프로그램』에서는 VS COBOL II 컴파일러로 컴파일된 프로그램과 연관된 일부 제한사항 및 고려사항을 나열합니다.

VS COBOL II 컴파일러는 이러한 애플리케이션이 올바르게 실행될 수 있도록 Language Environment 런타임 옵션을 조정할 수 있습니다. 자세한 정보는 Enterprise COBOL for z/OS Compiler and Runtime Migration Guide의 내용을 참조하십시오.

OO COBOL에 대한 지원

z/OS용 CICS Transaction Server, 버전 5 릴리스 3에서는 COBOL 클래스 정의 및 방법(객체 지향 COBOL)을 사용할 수 없습니다. 이 제한사항에는 Java 클래스 및 COBOL 클래스 둘 다 포함됩니다.

OOCOBOL 변환기 옵션이 있는 이전 CICS 릴리스에서 컴파일된 모듈은 z/OS용 CICS Transaction Server, 버전 5 릴리스 3 에서 실행될 수 없습니다. OOCOBOL 변환기 옵션은 이전 SOM 기반(System Object Manager 기반) OO COBOL에서 사용되었고

이 양식의 OO COBOL에 대한 런타임 지원은 z/OS V1.2에서 철회되었습니다. Enterprise COBOL에서 사용되는 새 Java 기반 OO COBOL은 CICS 변환기에서 지원되지 않습니다.

기존 SOM 기반 OO COBOL 프로그램이 있는 경우 Enterprise COBOL 컴파일러를 사용하려면 OO COBOL을 프로시저상 (non-OO) COBOL로 재작성하십시오. Java 기반 OO COBOL은 SOM 기반 OO COBOL 프로그램과 호환되지 않으며 SOM 기반 OO COBOL 프로그램에 대한 마이그레이션 경로로 계획되지 않습니다.

작업 중인 저장영역

컴파일러 옵션 DATA(24)에서 작업 중인 저장영역이 16MB 라인 아래로 할당됩니다. 컴파일러 옵션 DATA(31)에서 작업 중인 저장영역이 16MB 라인 위로 할당됩니다.

COBOL 프로그래밍 제한사항 및 요구사항

CICS 애플리케이션으로 사용되는 COBOL 프로그램에는 일부 제한사항 및 요구사항이 적용됩니다.

기본적으로 CICS 변환기 및 COBOL 컴파일러는 다음 제한사항의 영향을 받는 COBOL 단어의 사용을 감지하지 않습니다. CICS 환경에서 제한된 단어의 사용은 실행 시 장애의 원인이 될 수 있습니다. 그러나 COBOL은 CICS 애플리케이션에 대해 예약어 테이블 IGYCCICS를 제공합니다. 컴파일러 옵션 WORD(CICS)를 지정하는 경우 컴파일러는 IGYCCICS 테이블을 사용하고 CICS에서 지원되지 않는 COBOL 단어에 오류 메시지와 함께 플래그를 지정합니다. 기본 IBM 제공 예약어 테이블에 의해 일반적으로 제한되는 COBOL 단어에도 플래그가 지정됩니다. 제한되는 단어 및 IGYCCICS 테이블에 있는 단어의 현재 목록은 Enterprise COBOL for z/OS Programming Guide의 내용을 참조하십시오.

사용할 수 없는 함수 및 명령문

- CICS의 COBOL에서는 시작점을 사용할 수 없습니다.
- 대부분의 입력 및 출력 처리에 CICS 명령을 사용해야 합니다. 따라서 파일을 설명하거나 OPEN, CLOSE, READ, START, REWRITE, WRITE 또는 DELETE문을 코드화하지 마십시오. 대신 CICS 명령을 사용하여 데이터를 검색하고 갱신하고 삽입 및 삭제하십시오.
- CICS 프로그램에서는 형식-1 ACCEPT문을 사용하지 마십시오. 형식-2 ACCEPT문은 Language Environment가 사용 설정된 컴파일러에서 지원됩니다.
- DISPLAY . . . UPON CONSOLE 및 DISPLAY . . . UPON SYSPUNCH를 사용하지 마십시오. 시스템 논리 출력 디바이스(SYSOUT, SYSLIST, SYSLST)에 대한 DISPLAY는 지원되지 않습니다.
- STOP 『리터럴』을 사용하지 마십시오.

- SORT문의 사용에는 제한사항이 적용됩니다. Enterprise COBOL for z/OS Programming Guide의 내용을 참조하십시오. MERGE는 사용하지 마십시오.
- 다음을 사용하지 마십시오.
 - USE 선언문
 - CICS가 데이터 관리를 처리하므로 데이터 관리와 연관된 ENVIRONMENT DIVISION 및 FILE SECTION 항목. 이전에 나열된 제한된 SORT 기능과 연관된 경우 이러한 항목을 사용할 수 있습니다.
 - 기본 프로그램에 대한 사용자 지정 매개변수

코드화 요구사항

- 디버깅 행이 주석으로 사용될 경우 불일치 따옴표를 포함해서는 안됩니다.
- 변수 길이 영역을 생성하는 명령문(예: OCCURS DEPENDING ON)은 WORKING-STORAGE SECTION 내에서 주의하여 사용해야 합니다.
- 선언 섹션에서는 EXEC CICS 명령을 사용하지 마십시오.
- 영역 B(12 - 71열)에서 EXEC CICS 및 END-EXEC문을 둘 다 시작하십시오. 통합 CICS 변환기는 EXEC CICS문이 영역 A(8 - 11열)에서 시작되는 경우에 오류 메시지를 생성하지 않음에 유의하십시오. 키워드 EXEC가 영역 A에서 SQL문과 같은 다른 명령문의 일부로 사용될 수 있기 때문입니다.
- IDENTIFICATION DIVISION이 없는 경우 CICS 명령만 확장됩니다. IDENTIFICATION DIVISION만 있는 경우 DFHEIVAR, DFHEIBLK 및 DFHCOMMAREA만 생성됩니다.
- Language Environment 런타임이 있는 VS COBOL II 프로그램에서는 WORKING-STORAGE 길이에 다음과 같은 한계가 적용됩니다.
 - 컴파일러 옵션 DATA(24)가 사용되는 경우 한계는 16MB 라인 아래의 사용 가능한 공간입니다.
 - 컴파일러 옵션 DATA(31)가 사용되는 경우 한계는 128MB입니다.

저장영역 계정 및 저장 영역의 경우 80바이트가 필요합니다. 이를 한계에 포함시켜야 합니다.
- DLI 옵션이 지정되고 기존 프로그램의 PROCEDURE DIVISION 헤더 바로 뒤에 ENTRY문이 있을 경우 PROGRAM-ID 이름을 ENTRY문 리터럴로 변경하고 CICS에서 프로그램을 호출하기 전에 ENTRY문을 삭제하십시오.
- HANDLE CONDITION 또는 HANDLE AID를 사용하는 경우 SET(ADDRESS OF A-DATA) 또는 SET(A-POINTER)를 사용하여 주소 지정 문제점을 피할 수 있습니다. 여기서 A-DATA는 LINKAGE SECTION에 있는 구조이고 A-POINTER는 USAGE IS POINTER 절로 정의됩니다.

Language Environment 코드화 요구사항

Language Environment에서 처음으로 COBOL로 작성된 CICS 애플리케이션을 실행하는 경우 설치 시 사용 중인 Language Environment 런타임 옵션에 대한 검토가 필요할 수 있습니다. 특히 WORKING-STORAGE SECTION이 올바르게 초기화되도록 애플리케이션이 코드화되지 않은 경우(예: 맵을 전송하기 전에 2진 0으로 지워짐) STORAGE 런타임 옵션을 사용해야 합니다. Language Environment 런타임 옵션의 사용자 정의에 대한 정보는 z/OS Language Environment Programming Reference의 내용을 참조하십시오.

31비트 주소 지정

16MB 라인 위에서 실행 중인 COBOL 프로그램의 경우 31비트 주소 지정에 다음 제한사항이 적용됩니다.

- 수신 프로그램이 AMODE(31)로 링크 편집되는 경우 이 프로그램에 전달되는 주소는 31비트 길이(또는 제일 왼쪽 바이트가 0으로 설정된 24비트 길이)여야 합니다.
- 수신 프로그램이 AMODE(24)로 링크 편집되는 경우 이 프로그램에 전달되는 주소는 24비트 길이여야 합니다.

31비트 주소 지정 모드로 실행 중이고 데이터 인수를 24비트 주소 지정 모드 프로그램에 전달하는 프로그램에 DATA(24) 컴파일러 옵션을 지정하십시오. 이렇게 하면 데이터가 호출된 프로그램에 의해 주소 지정 가능해집니다.

64비트 주소 지정

64비트 주소 지정 모드는 COBOL 프로그램에서 지원되지 않습니다.

컴파일러 옵션

- 다음 컴파일러 옵션을 사용하지 마십시오.
 - DYNAM(프로그램이 변환되어야 하는 경우)
 - NOLIB(프로그램이 변환되어야 하는 경우)
 - NORENT
- DLL 컴파일러 옵션을 사용할 수도 있습니다.
- 다음 컴파일러 옵션은 CICS 환경에서 적용되지 않습니다.
 - ADV
 - AWO
 - EXPORTALL
 - FASTSRT
 - NAME
 - OOCOBOL

- OUTDD
- THREAD

- TEST(SYM,NOSEPARATE) 컴파일러 옵션을 사용하면 프로그램 크기가 크게 늘어납니다. 따라서 이 옵션을 사용하면 저장영역 부족 문제점이 발생할 수 있습니다. 프로그램 크기의 증가 없이 같은 기능을 제공하는 TEST(SYM,SEPARATE)를 사용할 수 있습니다. TEST 컴파일러 옵션에 대한 자세한 정보는 Enterprise COBOL for z/OS Programming Guide의 내용을 참조하십시오.
- PICTURE 정의와 일치하는 2진 데이터 항목 처리에 TRUNC(OPT)를 사용하십시오. 그렇지 않으면 TRUNC(OPT)를 컴파일러 옵션으로 사용하고 USAGE COMP-5를 2진 값이 PICTURE 절이 허용하는 값보다 클 수 있는 항목에 사용하십시오. TRUNC(BIN)는 런타임 성능을 제한합니다. 따라서 이 옵션은 2진 데이터 항목(예: 코드 생성기에서 작성된 2진 데이터 항목)을 제어할 수 없는 경우에만 사용하십시오 (TRUNC(STD)가 기본값임).

애플리케이션이 EIB의 필드를 사용하는 경우 DFHEIBLK 카피북은 EIBCALEN과 같은 필드를 PICTURE S9(4) USAGE COMPUTATIONAL로 정의함에 유의하십시오. DFHEIBLK 카피북에서 TRUNC(OPT) 컴파일러 옵션을 사용하면 2진 필드의 값이 9999보다 클 경우 해당 값이 잘립니다. 잘림 문제점을 피하기 위해서 갱신된 버전의 DFHEIBLK 카피북을 사용하는 통합 변환기를 사용하도록 권장됩니다. 통합 변환기에서 사용되는 DFHEIBLK의 버전은 TRUNC(OPT) 또는 TRUNC(BIN) 컴파일 옵션의 영향을 받는 모든 필드를 USAGE COMP-5로 정의합니다.

TRUNC 옵션에 대한 자세한 정보는 Enterprise COBOL for z/OS Customization Guide의 내용을 참조하십시오.

- RMODE(24) 컴파일러 옵션의 사용은 프로그램이 항상 16MB 라인 아래에 머무른다는 의미이므로 권장되지 않습니다. 대신 RMODE(ANY) 또는 RMODE(AUTO)를 사용해야 합니다. RMODE 컴파일러 옵션에 대한 자세한 정보는 Enterprise COBOL for z/OS Programming Guide의 내용을 참조하십시오.

WITH DEBUGGING MODE

『D』가 COBOL EXEC CICS 명령의 첫 번째 행 7열에 배치되는 경우 이 『D』는 변환된 CALL 문에도 있습니다. 이 변환된 명령은 WITH DEBUGGING MODE가 지정되는 경우에만 실행됩니다. EXEC CICS문의 첫 번째 행 외의 행에 배치된 『D』는 필수가 아니며 변환기에서 무시됩니다.

Language Environment CBLPSHPOP 옵션

CBLPSHPOP 런타임 옵션은 COBOL 서브루틴이 호출될 때마다 Language Environment가 초기화 중에는 EXEC CICS PUSH HANDLE 명령을 자동 실행하고 종료 중에는 EXEC CICS POP HANDLE 명령을 자동 실행하는지 여부를 제어합니다.

CICS에서 애플리케이션이 다수의 COBOL 서브루틴을 호출할 경우 성능은 CBLPSHPOP(ON)일 때보다 CBLPSHPOP(OFF)일 때 더 좋습니다. 19 페이지의 『Language Environment 런타임 옵션 정의』에서 설명한 대로 CEEUOPT를 사용하여 개별 트랜잭션 기준으로 CBLPSHPOP를 설정할 수 있습니다.

그러나 조건 처리가 스택되지 않았기 때문에 다음에 유의해야 합니다.

- 호출된 루틴에서 CICS가 호출 루틴의 조건 핸들러에게 제어 전달을 시도하게 하는 조건이 발생할 경우 이는 오류이고 트랜잭션은 비정상적으로 종료됩니다.
- 호출된 루틴 내에서 PUSHable CICS 명령(HANDLE ABEND, HANDLE AID, HANDLE CONDITION 또는 IGNORE CONDITION) 중 어느 하나를 사용하는 경우 호출자의 설정을 변경하게 되고 이로 인해 나중에 오류가 발생할 수 있습니다.
- 어셈블러 루틴을 호출하고 현재 핸들을 일시중단한 후 복귀시켜야 하는 경우 어셈블러 루틴은 푸시 및 팝 핸들을 요청해야 합니다. Language Environment는 COBOL 프로그램이 어셈블러 루틴을 호출할 때 이를 자동으로 수행하지 않습니다.

DL/I CALL 인터페이스 사용

CALL DL/I를 사용하는 COBOL 프로그램이 있고 아직 다음과 같이 변경하지 않은 경우 지금 변경해야 합니다.

- LINKAGE SECTION에 사용자 인터페이스 블록(DLIUIB) 선언 및 최소 한 개 이상의 프로그램 제어 블록(PCB) 선언을 보유합니다.
- 다음과 같이 UIB를 직접 지정하도록 PCB 호출을 변경하십시오.

```
CALL 'CBLTDLI' USING PCB-CALL
                        PSB-NAME
                        ADDRESS OF DLIUIB.
```

- UIB의 주소 목록에서 필요한 PCB의 주소를 얻으십시오.

31 페이지의 그림 1에서는 이 프로세스 전체를 설명합니다. 그림의 예제에서는 PSB에 세 개의 PCB가 정의되어 있고 데이터베이스 CALL에서 두 번째 PCB를 사용하려고 하는 것으로 가정합니다. 따라서 LINKAGE SECTION 그룹 항목 PCB의 ADDRESS 특수 레지스터를 설정할 때 프로그램은 작업 중인 저장영역 테이블 PCB-ADDRESS-LIST를 색인화하는 데 2를 사용합니다. n번째 PCB를 사용하려면 PCB-ADDRESS-LIST를 색인화하는 데 숫자 n을 사용합니다.

```

WORKING-STORAGE SECTION.
    77 PCB-CALL          PIC X(4) VALUE 'PCB '.
    77 GET-HOLD-UNIQUE   PIC X(4) VALUE 'GHU '.
    77 PSB-NAME          PIC X(8) VALUE 'CBLPSB'.
    77 SSA1              PIC X(40) VALUE SPACES.
    01 DLI-IO-AREA.
        02 DLI-IO-AREA1 PIC X(99).

*
LINKAGE SECTION.
    COPY DLIUIB.
    01 OVERLAY-DLIUIB REDEFINES DLIUIB.
        02 PCBADDR      USAGE IS POINTER.
        02 FILLER        PIC XX.
    01 PCB-ADDR-LIST.
        02 PCB-ADDRESS-LIST USAGE IS POINTER
                                OCCURS 10 TIMES.

    01 PCB.
        02 PCB-DBD-NAME   PIC X(8).
        02 PCB-SEG-LEVEL  PIC XX.
        02 PCB-STATUS-CODE PIC XX.

*
PROCEDURE DIVISION.
*SCHEDULE THE PSB AND ADDRESS THE UIB
    CALL 'CBLTDLI' USING PCB-CALL PSB-NAME ADDRESS OF DLIUIB.

*
*MOVE VALUE OF UIBPCBAL, ADDRESS OF PCB ADDRESS LIST (HELD IN UIB)
*(REDEFINED AS PCBADDR, A POINTER VARIABLE), TO
*ADDRESS SPECIAL REGISTER OF PCB-ADDR-LIST TO PCBADDR.
    SET ADDRESS OF PCB-ADDR-LIST TO PCBADDR.
*MOVE VALUE OF SECOND ITEM IN PCB-ADDRESS-LIST TO ADDRESS SPECIAL
*REGISTER OF PCB, DEFINED IN LINKAGE SECTION.
    SET ADDRESS OF PCB TO PCB-ADDRESS-LIST(2).
*PERFORM DATABASE CALLS .....
    .....
    MOVE ..... TO SSA1.
    CALL 'CBLTDLI' USING GET-HOLD-UNIQUE PCB DLI-IO-AREA SSA1.
*CHECK SUCCESS OF CALLS .....
    IF UIBFCTR IS NOT EQUAL LOW-VALUES THEN
                                ..... error diagnostic code

    .....
    IF PCB-STATUS-CODE IS NOT EQUAL SPACES THEN
                                ..... error diagnostic code

    .....

```

그림 1. DLI CALL 인터페이스 사용

VS COBOL II 프로그램

Language Environment는 VS COBOL II 컴파일러로 컴파일된 프로그램의 실행에 대한 지원을 제공합니다. 이 컴파일러에 대한 고유 런타임 라이브러리는 지원되지 않습니다. 그러나 이 컴파일러는 Language Environment를 준수하지 않기 때문에(Language Environment 이전 컴파일러임) 그 사용에 연관된 일부 제한사항 및 고려사항이 있습니다.

VS COBOL II 프로그램을 Language Environment 지원으로 업그레이드하는 것에 대한 자세한 정보는 *Enterprise COBOL for z/OS: Compiler and Run-Time Migration Guide*의 내용을 참조하십시오.

Language Environment 호출 가능 서비스

Language Environment 준수 COBOL 컴파일러에 의해 컴파일되는 프로그램은 모든 Language Environment 호출 가능 서비스를 동적으로 또는 정적으로 사용할 수 있습니다. 그러나 CICS 애플리케이션의 경우 CEEMOUT(메시지 디스패치) 및 CEE3DMP(덤프 생성) 서비스는 메시지와 덤프가 MSGFILE 런타임 옵션에서 지정된 ddname이 아닌 CESE 트랜지언트 데이터 큐에 전송된다는 점에서 다릅니다.

VS COBOL II 프로그램은 날짜 및 시간 호출 가능 서비스를 호출할 수 있지만 Language Environment 호출 가능 서비스에 대한 그 외 정적 또는 동적 호출은 VS COBOL II 프로그램에서 지원되지 않습니다.

VS COBOL II 프로그램 재링크

Language Environment에서 제공하는 런타임 지원을 사용하기 위해 기존 VS COBOL II 프로그램에 재링크하는 데 오브젝트 모듈을 사용할 수 없는 경우 태스크를 수행하기 위한 동일한 작업 스트림이 SCEESAMP 샘플 라이브러리의 IGZWRLKA 멤버에서 제공됩니다.

CICS 스텝

이전 CICS 스텝인 DFHECI에 링크된 COBOL 프로그램이 Language Environment에서 실행되더라도 DFHELII 스텝 사용을 권장하며 혼합 Language Environment에서 DFHELII 스텝을 사용하는 것은 매우 중요합니다. DFHECI는 애플리케이션의 맨 위에서 링크 편집해야 하지만 DFHELII는 애플리케이션의 모든 위치에서 링크될 수 있습니다.

CEEWUCHA 사용

Language Environment에서 제공하는 런타임 지원을 사용하기 위해 VS COBOL II 프로그램을 조정하는 경우 Language Environment에 의해 SCEESAMP 라이브러리에서 제공되는 샘플 사용자 조건 핸들러인 CEEWUCHA를 유리하게 사용할 수 있습니다. 이 핸들러는 다음과 같이 작동합니다.

- 런타임 중 감지된 오류가 발생하면 EXEC CICS HANDLE ABEND LABEL문이 제어할 수 있도록 하여 CICS에서 실행되는 기존 VS COBOL II 애플리케이션과의 호환성을 제공합니다.
- 런타임 중 감지된 모든 미처리 오류를 VS COBOL II에서 실행된 해당 사용자 1xxx 이상 종료로 변환합니다.

- IGZETUN 또는 IGZEOPT가 VS COBOL II 애플리케이션에서 링크 편집될 때 생성되는 모든 IGZ0014W 메시지를 비활성화합니다(프로그램이 IGZETUN 또는 IGZEOPT로 링크 편집되는 경우 성능은 더 우수함).

샘플 사용자 조건 핸들러인 CEEWUCHA를 런타임 시 사용할 수 있는지 확인하십시오(예: STEPLIB 연결 또는 LPA 사용). 프로그램 자동 설치를 사용하기 보다는 CICS 시스템 정의 데이터 세트(CSD)에서 CICS 리전에 대한 조건 핸들러를 정의하십시오.

COBOL에서 기본 주소 지정 사용

COBOL은 포인터 변수 및 ADDRESS 특수 레지스터를 사용하여 LINKAGE SECTION에서 정의된 CICS 데이터 영역에 대한 주소 지정성을 얻는 단순한 방법을 제공합니다.

CICS 애플리케이션은 데이터가 CICS 내부 영역에 있고 그 주소만 프로그램에 전달될 때 데이터에 동적으로 액세스해야 합니다. 예는 다음과 같습니다.

- ADDRESS 명령을 사용하여 액세스하는 CWA, TWA 및 TCTTE 사용자 영역 (TCTUA) 등의 CICS 영역
- SET 옵션을 포함한 READ 및 RECEIVE 등의 EXEC CICS 명령을 통해 얻을 수 있는 입력 데이터

ADDRESS 특수 레지스터는 LINKAGE SECTION의 레벨 01 또는 77에서 정의된 레코드의 주소를 보유합니다. 이 레지스터는 ADDRESS 모드에서 임의 명령의 SET 옵션에서 사용할 수 있습니다. 이러한 명령에는 GETMAIN, LOAD, READ 및 READQ가 있습니다.

34 페이지의 그림 2에서는 COBOL에서 ADDRESS 특수 레지스터의 사용을 보여줍니다. READ 또는 REWRITE 명령의 레코드 길이가 고정된 경우 LENGTH 옵션은 필요하지 않습니다. 이 예제에서는 변수 길이 레코드로 가정합니다. READ 뒤 LENGTH 옵션(여기에서는 LRECL-REC1)에서 이름 지정되는 필드의 레코드 길이를 알 수 있습니다. REWRITE 명령에서 갱신된 레코드를 다른 길이의 레코드로 바꾸려는 경우 LENGTH 옵션을 코드화해야 합니다.

```

WORKING-STORAGE SECTION.
77 LRECL-REC1    PIC S9(4) COMP.
LINKAGE SECTION.
01 REC-1.
   02 FLAG1      PIC X.
   02 MAIN-DATA  PIC X(5000).
   02 OPTL-DATA  PIC X(1000).
01 REC-2.
   02 ...
PROCEDURE DIVISION.
   EXEC CICS READ UPDATE...
       SET(ADDRESS OF REC-1)
       LENGTH(LRECL-REC1)
       END-EXEC.
   IF FLAG1 EQUAL X'Y'
       MOVE OPTL-DATA TO ...

   EXEC CICS REWRITE...
       FROM(REC-1)
       END-EXEC.

```

그림 2. 찾기 모드에서 CICS 데이터 영역의 주소 지정

COBOL 프로그램에서 서브프로그램 호출

CICS 시스템에서, 제어가 활성 프로그램에서 외부 프로그램으로 전송되지만 전송 프로그램이 활성인 상태로 유지되고 이 프로그램에 제어가 리턴될 수 있는 경우 제어가 전송되는 프로그램을 서브프로그램이라고 합니다. COBOL에는 제어를 서브프로그램에 전송하는 세 가지 방식이 있습니다.

EXEC CICS LINK

호출 프로그램에는 다음 양식 중 하나인 명령이 포함됩니다.

```

EXEC CICS LINK PROGRAM('subpgname')
EXEC CICS LINK PROGRAM(name)

```

첫 번째 양식에서 호출되는 서브프로그램은 영숫자 리터럴로 지정됩니다. 두 번째 양식에서 *name*은 서브프로그램의 이름에 필요한 길이와 같은 길이의 COBOL 데이터 영역을 나타냅니다.

정적 COBOL 호출

호출 프로그램은 다음과 같은 양식의 COBOL문을 포함합니다.

```
CALL 'subpgname'
```

호출되는 서브프로그램은 리터럴 문자열로 명시적으로 이름 지정됩니다.

동적 COBOL 호출

호출 프로그램은 다음과 같은 양식의 COBOL문을 포함합니다.

```
CALL identifier
```

이 ID는 호출된 서브프로그램의 이름을 포함해야 하는 COBOL 데이터 영역의 이름입니다.

서브프로그램을 호출하기 위해 이러한 방법 각각을 사용할 때의 성능 영향에 대한 정보는 *Enterprise COBOL for z/OS: Programming Guide* 및 Enterprise COBOL 버전 4 릴리스 2 성능 튜닝(<http://www.ibm.com/support/docview.wss?uid=swg27018287>)을 참조하십시오.

COBOL 프로그램은 정적으로 또는 동적으로 CICS가 지원하는 모든 언어로 프로그램을 호출할 수 있습니다. COMMAREA와 같은 CICS 기능을 사용하려는 경우가 아니면 LINK 또는 XCTL은 언어 간 통신에서 필요하지 않습니다. 언어 간 통신에 대한 자세한 정보는 16 페이지의 『Language Environment에서 언어 혼합』의 내용을 참조하십시오.

어셈블러 언어 서브프로그램이 더 낮은 레벨의 서브프로그램을 호출할 수 없다는 점을 제외하고 호출 또는 링크된 서브프로그램의 내용은 CICS가 지원하는 모든 함수일 수 있습니다(DB2[®] 및 DL/I와 같은 외부 데이터베이스에 대한 호출 포함).

프로그램 및 서브프로그램 간 제어 플로우

COBOL 기본 프로그램과 서브프로그램 사이에는 수많은 플로우가 있을 수 있습니다.

실행 단위는 COBOL 정적 또는 동적 CALL문을 통해 서로 통신하는 한 개 이상의 실행 중인 프로그램으로 이루어진 세트입니다. CICS 환경에서 실행 단위는 CICS 태스크 시작 시 입력되거나 LINK 또는 XCTL 명령으로 호출됩니다. 실행 단위는 PROGRAM 자원 정의로 정의되는 프로그램의 실행으로 정의될 수 있습니다. 동적 CALL의 경우에도 호출된 프로그램에 대한 후속 PROGRAM 정의가 필요합니다. XCTL 명령으로 제어가 전달되면 제어를 수신하는 프로그램은 RETURN 명령 또는 GOBACK 문을 통해 호출 프로그램에 제어를 리턴할 수 없고 따라서 이 프로그램은 서브프로그램이 아닙니다.

각 LINK 명령은 호출 프로그램보다 한 레벨 낮은 레벨의 호출되는 프로그램인 새 CICS 애플리케이션 논리 레벨을 작성합니다(CICS는 0 레벨에 있는 것으로 간주됨). 36 페이지의 그림 3에서는 논리 레벨과 링크 및 호출된 프로그램에서 RETURN 명령 및 CALL문의 영향을 보여줍니다.

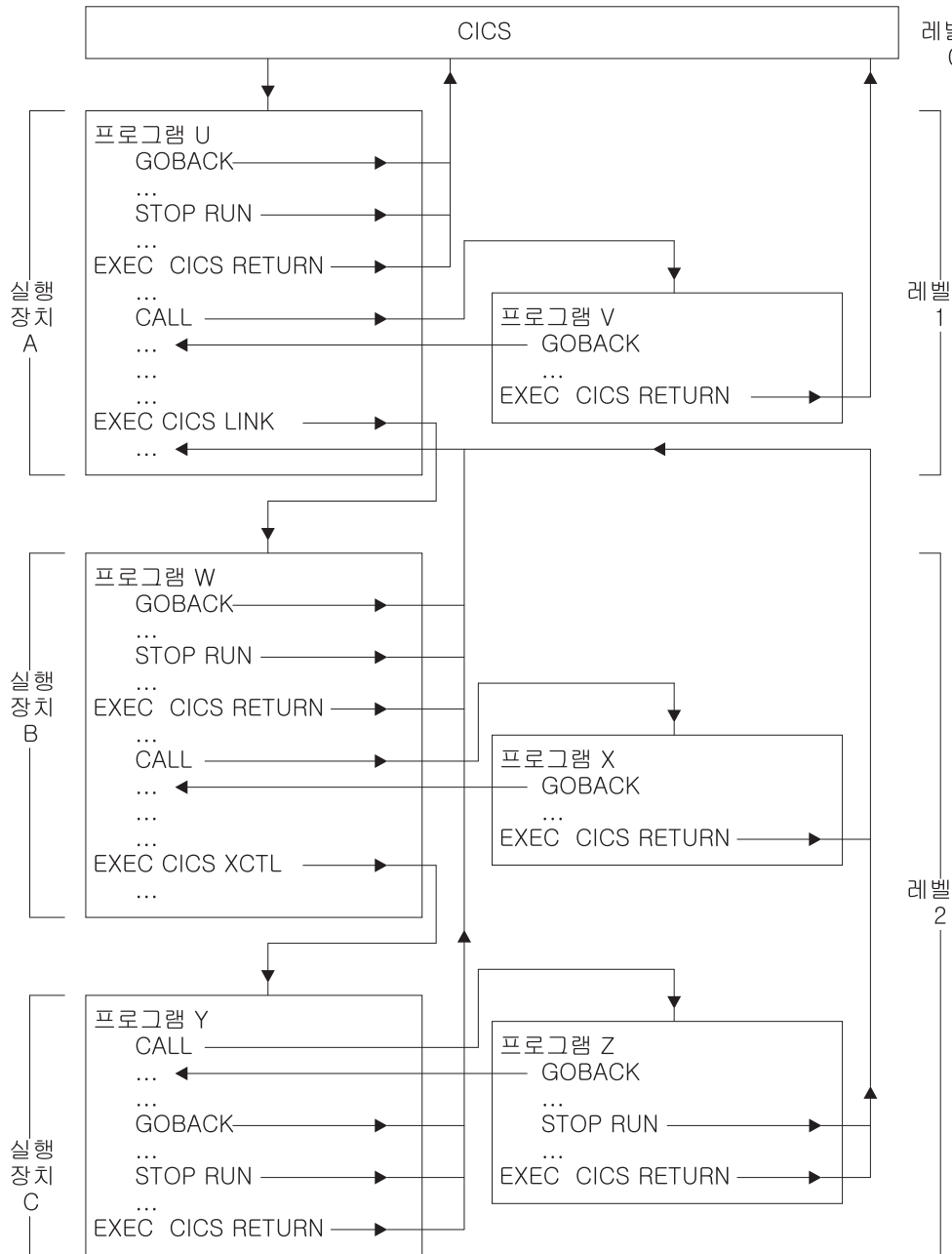


그림 3. COBOL 프로그램, 실행 단위 및 CICS 간 제어 플로우

기본 또는 레벨 1 프로그램은 COBOL GOBACK 또는 STOP RUN문 또는 CICS RETURN 명령을 사용하여 종료하고 CICS로 리턴할 수 있습니다. 이 프로그램은 COBOL CALL문을 사용하여 같은 논리 레벨(레벨 1)에 있는 서브프로그램을 호출하거나 CICS LINK 명령을 사용하여 낮은 논리 레벨에 있는 서브프로그램을 호출할 수 있습니다. 레벨 1에서 호출된 서브프로그램은 COBOL GOBACK문을 사용하여 호출자에게 리턴하거나 종료하고 EXEC CICS RETURN을 사용하여 CICS로 리턴할 수 있습니다.

레벨 2 서브프로그램 실행에서는 COBOL GOBACK 또는 STOP RUN문 또는 CICS RETURN 명령을 사용하여 종료하고 레벨 1 호출 프로그램으로 리턴할 수 있습니다. 이 프로그램은 COBOL CALL문 또는 CICS XCTL 명령을 사용하여 같은 레벨(레벨 2)에 있는 서브프로그램을 호출할 수 있습니다. 레벨 2에서 COBOL CALL을 사용하여 호출된 서브프로그램은 GOBACK문을 사용하여 호출자(레벨 2에 있음)에게 리턴하거나 EXEC CICS RETURN을 사용하여 레벨 1 호출 프로그램으로 리턴할 수 있습니다. 레벨 2에서 XCTL을 사용하여 호출된 서브프로그램은 GOBACK, STOP RUN 또는 EXEC CICS RETURN을 사용하여 레벨 1로만 리턴할 수 있습니다.

프로그램 논리 레벨에 대한 자세한 정보는 366 페이지의 『애플리케이션 논리 레벨』의 내용을 참조하십시오.

서브프로그램 호출에 대한 규칙

다음 규칙은 COBOL 프로그램에서 호출되거나 링크된 서브프로그램의 작동 및 요구사항을 설명합니다. 적용되는 규칙은 제어가 서브프로그램에 전송되는 방식(EXEC CICS LINK 명령, 정적 COBOL 호출 또는 동적 COBOL 호출 사용)에 따라 다릅니다.

서브프로그램의 위치

EXEC CICS LINK

서브프로그램은 원격일 수 있습니다.

정적 또는 동적 COBOL 호출

서브프로그램은 로컬이어야 합니다.

변환

통합 변환기가 있는 컴파일러를 사용하는 경우 변환이 필요하지 않습니다.

EXEC CICS LINK

링크된 서브프로그램 또는 이 프로그램에서 호출된 서브프로그램에 CICS 기능이 있는 경우 링크된 서브프로그램을 변환해야 합니다.

정적 또는 동적 COBOL 호출

CICS 명령 또는 EXEC 인터페이스 블록(DFHEIBLK)이나 CICS 통신 영역(DFHCOMMAREA)에 대한 참조를 포함할 경우 호출된 서브프로그램을 변환해야 합니다.

컴파일

CICS와 함께 실행되는 COBOL 프로그램이 동적 호출을 실행한다고 해도 이 프로그램을 컴파일할 때에는 항상 NODYNAM 컴파일러 옵션(기본값)을 사용해야 합니다.

링크 편집

EXEC CICS LINK

링크된 서브프로그램은 개별 프로그램으로 컴파일 및 링크 편집해야 합니다.

정적 COBOL 호출

호출된 서브프로그램은 단일 로드 모듈(그러나 프로그램은 개별적으로 컴파일할 수 있음)을 구성하도록 호출 프로그램으로 링크 편집해야 합니다. 이렇게 하면 대형 프로그램 모듈을 생성할 수 있고 같은 프로그램을 호출하는 두 프로그램이 해당 프로그램의 사본을 공유하는 것도 중지합니다.

동적 COBOL 호출

호출된 프로그램은 개별 로드 모듈로 컴파일 및 링크 편집해야 합니다. 이 프로그램은 링크 팩 영역 또는 동시에 다른 CICS 및 비CICS 리전과 공유하는 라이브러리에 상주할 수 있습니다.

프로그램 자동 설치가 없는 CICS CSD 항목

프로그램 자동 설치를 사용하는 경우 CSD의 항목이 필요하지 않습니다.

EXEC CICS LINK

링크된 서브프로그램은 RDO를 사용하여 정의해야 합니다. 링크된 서브프로그램이 알 수 없거나 사용 불가능한 경우 자동 설치가 활성화되더라도 LINK는 PGMIDERR 조건으로 실패합니다.

정적 COBOL 호출

호출 프로그램은 CSD에서 정의해야 합니다. 프로그램 A가 프로그램 B를 호출한 후 프로그램 B가 프로그램 A 호출을 시도하면 COBOL이 메시지 및 이상 종료를 발행합니다(1015). 서브프로그램은 호출 프로그램의 일부이므로 CSC 항목은 필요하지 않습니다.

동적 COBOL 호출

호출 프로그램은 CSD에서 정의해야 합니다. 프로그램 A가 프로그램 B를 호출한 후 프로그램 B가 프로그램 A 호출을 시도하면 COBOL이 메시지 및 이상 종료를 발행합니다(1015). 호출된 서브프로그램은 CSD에서 정의해야 합니다. 자동 설치가 활성화된 경우에도 호출된 서브프로그램을 로드할 수 없거나 사용할 수 없는 경우 COBOL은 메시지 및 이상 종료를 발행합니다(1029).

COBOL의 반복 호출

프로그램 A가 프로그램 B를 호출하고 프로그램 B는 프로그램 A 호출을 시도하면 Language Environment는 CEEMSG에 IGZ0064S 및 이상 종료(4038)를 발행합니다.

프로그램 A 및 프로그램 B의 PROGRAM-ID에 RECURSIVE 키워드가 있는 경우 반복 호출이 허용됩니다.

서브프로그램에 매개변수 전달

호출 또는 링크된 서브프로그램이 CICS 변환기에 의해 처리되는 경우 데이터는 표준 CICS 메소드(COMMAREA, TWA, TCTUA, TS 쿼)로 전달할 수 있습니다.

EXEC CICS LINK

COMMAREA를 사용하는 경우 그 주소를 LINK 명령으로 전달해야 합니다. 링크된 서브프로그램이 24비트 주소 지정을 사용하고 COMMAREA가 16MB 라인 위에 있는 경우 CICS는 이 주소를 16MB 라인 아래에 복사하고 리턴 시 다시 복사합니다.

정적 COBOL 호출

호출된 프로그램이 EXEC CICS 요청을 발행하거나 호출된 프로그램이 **EXEC CICS ADDRESS** 명령을 발행할 수 있는 경우 CALL문은 DFHEIBLK 및 DFHCOMMAREA를 처음 두 매개변수로 전달할 수 있습니다. COMMAREA는 선택적이지만 다른 매개변수가 전달되는 경우 더미 COMMAREA도 전달되어야 합니다. 중첩된 프로그램에 대한 규칙은 다를 수 있습니다.

동적 COBOL 호출

호출된 프로그램이 EXEC CICS 요청을 발행하거나 호출된 프로그램이 **EXEC CICS ADDRESS** 명령을 발행할 수 있는 경우 CALL문은 DFHEIBLK 및 DFHCOMMAREA를 처음 두 매개변수로 전달할 수 있습니다. COMMAREA는 선택적이지만 다른 매개변수가 전달되는 경우 더미 COMMAREA도 전달되어야 합니다. 호출된 서브프로그램이 24비트 주소 지정을 사용하고 매개변수가 16MB 라인 위에 있는 경우 COBOL은 메시지 및 이상 종료를 발행합니다(1033).

서브프로그램에서 리턴

EXEC CICS LINK

링크된 서브프로그램은 **EXEC CICS RETURN** 또는 고유 언어 리턴 명령(예: COBOL문 **GOBACK**)을 사용하여 리턴해야 합니다.

정적 또는 동적 COBOL 호출

호출된 서브프로그램은 고유 언어 리턴 명령문(예: COBOL문 **GOBACK** 또는 **EXIT PROGRAM**)을 사용하여 리턴해야 합니다. 호출된 서브프로그램이 **EXEC CICS RETURN**을 사용하면 호출 프로그램이 종료됩니다.

저장영역

EXEC CICS LINK

링크된 서브프로그램에 입력할 때마다 WORKING-STORAGE SECTION의 새 초기화된 사본이 제공되고 실행 단위가 재초기화됩니다(일부 환경에서는 이로 인해 성능 저하가 발생할 수 있음).

링크된 서브프로그램에 입력할 때마다 LOCAL-STORAGE 섹션의 새 초기화된 사본이 제공됩니다.

정적 또는 동적 COBOL 호출

CICS 논리 레벨 내에서 호출된 서브프로그램에 처음 입력 시 WORKING-STORAGE SECTION의 새 초기화된 사본이 제공됩니다. 같은 논리 레벨에서 호출된 서브프로그램에 후속 입력 시 같은 WORKING STORAGE가 최종 사용 상태로 제공됩니다. 즉, 저장영역이 사용 가능해지거나 획득되거나 초기화되지 않습니다. LINK 명령의 성능이 만족스럽지 못한 경우 COBOL 호출이 개선된 결과를 제공할 수 있습니다.

CICS 논리 레벨에서 호출된 서브프로그램에 입력할 때마다 해당 LOCAL-STORAGE SECTION의 새 초기화된 사본이 제공됩니다.

CICS 조건, AID 및 이상 종료 처리

EXEC CICS LINK

호출된 서브프로그램에 입력 시 이상 종료 또는 조건 처리가 활성이 아닙니다. 서브프로그램 내에서는 일반적인 CICS 규칙이 적용됩니다. 서브프로그램의 지속 기간 동안 존재하는 이상 종료 또는 조건 처리 환경을 설정하려면 서브프로그램에 입력 시 새 HANDLE 명령을 실행해야 합니다. 이렇게 작성된 환경은 추가 HANDLE 명령이 실행되거나 서브프로그램이 호출자에게 제어를 리턴할 때까지 유효한 상태로 유지됩니다.

정적 또는 동적 COBOL 호출

- Language Environment 및 CBLPSHPOP가 켜진 상태에서 동적으로 호출된 COBOL 프로그램이 이상 종료되는 경우 호출된 서브프로그램에 입력 시 이상 종료 또는 조건 처리가 활성이 아닙니다. 서브프로그램 내에서는 일반적인 CICS 규칙이 적용됩니다. 호출된 서브프로그램에 입력 시 COBOL은 PUSH HANDLE을 실행하여 호출 프로그램의 조건 또는 이상 종료 핸들러를 스택합니다. 서브프로그램 지속 기간 동안 존재하는 이상 종료 또는 조건 처리 환경을 설정하려면 서브프로그램에 입력 시 새 HANDLE 명령을 실행해야 합니다. 이렇게 작성된 환경은 추가 HANDLE 명령이 실행되거나 서브프로그램이 호출자에게 제어를 리턴할 때까지 유효한 상태로 유지됩니다. 서브프로그램에서 호출 프로그램으로 제어가 리턴되면 COBOL은 POP HANDLE을 사용하여 조건 및 이상 종료 핸들러를 스택 해제합니다.
- 동적으로 호출된 COBOL 프로그램이 CBLPSHPOP OFF 상태에서 이상 종료되고 조건, AID 또는 호출 프로그램에 대한 이상 종료 처리가 활성인 경우 이 프로그램은 이상 종료 코드 APC2로 종료됩니다.
- 정적으로 호출된 COBOL 프로그램의 경우 CBLPSHPOP의 설정과 관계없이 조건, AID 및 이상 종료 처리가 유효한 상태로 유지됩니다.

COBOL2 및 COBOL3 변환기 옵션

z/OS용 CICS Transaction Server, 버전 5 릴리스 3 에서 COBOL 프로그램에 대해 COBOL2와 COBOL3 CICS 변환기 옵션 중 하나를 선택할 수 있습니다.

ANSI85 변환기 옵션이 z/OS용 CICS Transaction Server, 버전 2 릴리스 2에서 더 이상 사용 가능하지 않습니다.

OOCOBOL 변환기 옵션으로 이전 CICS 릴리스에서 컴파일된 모듈은 z/OS용 CICS Transaction Server, 버전 5 릴리스 3 에서 실행할 수 없습니다. OOCOBOL 변환기 옵션은 이전 SOM 기반(System Object Manager 기반) OO COBOL에서 사용되었고 이 양식의 OO COBOL에 대한 런타임 지원은 z/OS V1.2에서 철회되었습니다. Enterprise COBOL에서 사용되는 새 Java 기반 OO COBOL은 CICS 변환기에서 지원되지 않습니다.

COBOL2 옵션은 기본값입니다. 이 옵션은 변환기에서는 z/OS용 CICS Transaction Server, 버전 2 릴리스 1 및 이전 릴리스에서와 같은 효과를 가지지 않습니다. COBOL2는 EXEC CICS 및 EXEC DLI 요청에 사용할 임시 변수 선언을 포함시키는 것 외에도 변환기에 COBOL3로 변환하도록 지시합니다.

임시 변수의 사용을 요구하는 방식으로 작성된 이전 프로그램을 재변환하는 경우 COBOL2 옵션을 선택하십시오. 특히 임시 변수를 사용하면 프로그램의 인수 값이 잘못 정의된 경우에 일반적으로 발생하는 오류를 피할 수 있다는 점에 유의하십시오. z/OS용 CICS Transaction Server, 버전 2 릴리스 1 및 이전 릴리스에서 COBOL2 옵션은 임시 변수의 선언을 제공했습니다. 이러한 기능으로 인해 원래 CICS 트랜잭션 서버의 이전 릴리스에서 COBOL2 옵션을 사용하여 변환된 프로그램에서 인수 값의 잘못된 정의가 발생할 수 있지만 런타임 시 이를 알 수 없습니다. COBOL3 옵션을 사용하여 이러한 프로그램을 변환하면 처음으로 이러한 오류를 밝힐 수 있습니다. CICS 새 릴리스로의 업그레이드를 지원하기 위해 오류를 정정하는 대신 프로그램에서 계속해서 오류를 피할 수 있는 새 COBOL2 옵션을 사용할 수 있습니다.

프로그램에 변환기의 임시 변수가 필요하지 않음을 확인하는 경우 작업 저장영역이 비교적 작은 COBOL3를 사용할 수 있습니다. COBOL3 옵션에는 이전 COBOL2 및 ANSI85 변환기 옵션의 모든 기능이 포함되지만 임시 변수의 선언은 제외됩니다.

참고: COBOL2 및 COBOL3는 상호 배타적입니다. 다른 방법으로 두 옵션을 모두 지정하는 경우 두 옵션이 지정된 위치에 관계없이 항상 COBOL3 옵션이 사용됩니다. 이러한 상황이 발생하면 변환기가 경고 메시지를 발행합니다.

z/OS용 CICS Transaction Server, 버전 2 릴리스 2 및 이후 버전과 릴리스의 CICS 변환기 지원은 이전 COBOL 컴파일러에서 사용 가능했던 CMPR2 컴파일러 옵션을 지

원하지 않습니다. 이러한 COBOL 프로그램을 NOCMR2 기능으로 업그레이드하는 것에 대한 정보는 Enterprise COBOL for z/OS Compiler and Runtime Migration Guide의 내용을 참조하십시오.

프로그램 변환 및 실행을 위한 준비에 대한 일반 정보는 93 페이지의 제 7 장 『변환 및 컴파일』의 내용을 참조하십시오.

COBOL 프로그램을 위한 CICS 변환기 조치

다음 노트는 COBOL3 옵션이 사용될 때 취하는 특정 변환기 조치를 설명합니다. COBOL2 옵션을 사용한 처리는 임시 변수의 선언을 제외한 모든 측면에서 동일합니다.

빈 행에 개입하는 리터럴

빈 행은 COBOL 소스 프로그램의 모든 위치에 있을 수 있습니다. 빈 행에는 7열과 72열 사이 포함된 공백 외에 아무 것도 포함되지 않습니다.

COBOL 소스 프로그램에서 리터럴 내에 빈 행이 발생하면 변환기는 변환된 출력에서 해당 행을 제거하지만 변환된 목록에는 포함시킵니다.

소문자

소문자는 사용자 정의 이름, 시스템 이름 및 예약어를 포함한 모든 COBOL 단어에서 발생할 수 있습니다. 변환기 목록 및 출력에서는 COBOL 텍스트의 대소문자 구분을 입력 시와 동일하게 유지합니다.

또한 변환기는 다음 항목에서 혼합 대소문자를 승인합니다.

- 변환기 옵션
- 키워드 및 키워드에 대한 인수 둘 다의 EXEC CICS 명령
- CBL 및 PROCESS문
- EJECT 및 SKIP1과 같은 컴파일러 지시문

변환기는 소문자 텍스트를 대문자로 변환하지 않습니다. COBOL 텍스트로 된 일부 이름(예: 파일 이름 및 트랜잭션 ID)은 외부에서 정의된 이름과 일치해야 합니다. 이러한 이름은 항상 외부 정의와 같은 대소문자로 입력해야 합니다.

LINKAGE 변환기 옵션을 지정하거나 이를 기본값으로 승인하면 EIB 구조의 혼합 대소문자 버전(DFHEIBLC)이 LINKAGE SECTION에 삽입됩니다.

문자를 포함하는 순서 번호

COBOL 소스 프로그램에서 순서 번호 필드는 컴퓨터의 문자 세트에 있는 모든 문자를 포함할 수 있습니다. 순서 번호 필드는 순서대로 있거나 고유하지 않아도 됩니다.

REPLACE문

COBOL 프로그램은 식별된 텍스트를 정의된 대체 텍스트로 대체할 수 있도록 하는 REPLACE문을 포함할 수 있습니다. 대체 및 삽입될 텍스트는 의사 텍스트, ID, 리터럴 또는 COBOL 단어일 수 있습니다. REPLACE문은 COPY문 후 처리됩니다.

CICS 제공 변환기에서 COBOL 소스 명령문을 처리하는 경우 변환기는 REPLACE문을 승인하지만 발생하는 위치에서 변환되는 CICS 기본 제공 함수(DFHRESP 및 DFHVALUE)를 제외하고 의사 텍스트 구분 기호 사이에서 텍스트를 변환하지 않습니다. CICS 명령은 의사 텍스트 구분 기호 사이에 배치해야 합니다.

통합 변환기를 사용하는 경우 이 변환기는 REPLACE문을 승인하고 의사 텍스트 구분 기호 사이에서 텍스트를 변환합니다. CICS 명령은 의사 텍스트 구분 기호 사이에 배치할 수 있습니다.

참조 수정

참조 수정은 데이터 항목에서 하위 문자열의 시작 위치(제일 왼쪽)를 지정하고 선택적으로 하위 문자열의 길이를 지정하여 문자 데이터 항목의 하위 문자열 참조 메소드를 지원합니다. 허용되는 형식은 다음과 같습니다.

```
data-name (leftmost-character-position:)
data-name (leftmost-character-position: length)
```

데이터 이름은 첨자화되거나 규정되거나 둘 다일 수 있습니다. *leftmost-character-position* 및 *length* 둘 다 산술 연산식일 수 있습니다. 참조 수정, 규정화 및 첨자화에 대한 자세한 정보는 *Enterprise COBOL for z/OS: Language Reference*의 내용을 참조하십시오.

변환기는 문자 변수의 이름이 COBOL 프로그램 또는 EXEC CICS 명령에서 허용될 때마다 참조 수정을 승인합니다.

참고: CICS 명령이 데이터 값 정의에서 참조 수정을 사용하는 경우 NOLENGTH 변환기 옵션을 사용하지 않는 한 LENGTH 옵션을 포함시켜 데이터 길이를 지정해야 합니다. 그렇지 않으면 변환기가 다음과 같은 양식의 LENGTH 레지스터 참조가 포함된 COBOL 호출을 생성합니다.

```
LENGTH OF (reference modification)
```

이는 컴파일러에서 거부됩니다.

글로벌 변수

GLOBAL 절은 지원됩니다. 최상위 레벨 프로그램에서 GLOBAL 절로 정의되는 변수 (47 페이지의 『중첩된 COBOL 프로그램』 참조)는 직접적으로 또는 간접적으로 포함되었는지에 관계없이 해당 중첩 프로그램에서 참조될 수 있습니다.

변환기는 GLOBAL 키워드를 허용합니다.

구분 기호로 사용하는 쉼표와 세미콜론

분리문자 쉼표는 공백이 뒤에 오는 쉼표입니다. 분리문자 세미콜론은 공백이 뒤에 오는 세미콜론입니다. 분리문자 쉼표 또는 분리문자 세미콜론은 공백 자체가 사용될 수 있는 모든 위치에서 분리문자로 사용할 수 있습니다.

변환기는 COBOL 명령문에서 공백을 사용할 수 있는 모든 위치에 분리문자 쉼표 또는 분리문자 세미콜론의 사용을 허용합니다. 예를 들어, 변환기는 다음 명령문을 승인합니다.

```
IDENTIFICATION; DIVISION
```

변환기는 EXEC CICS 명령의 구분 기호로 분리문자 쉼표 및 분리문자 세미콜론을 사용하는 것을 지원하지 않습니다. EXEC CICS 명령에서 유일하게 허용되는 단어 구분 기호는 계속해서 공백입니다.

기호 문자 정의

기호 문자는 ALPHABET 절 뒤에 오는 SPECIAL-NAMES 단락에서 정의할 수 있습니다. 기호 문자는 1문자 표의 상수를 나타내는 프로그램 정의 단어입니다.

변환기는 표준에 지정된 기호 문자의 사용을 허용합니다.

참고: 일반적으로 컴파일러는 CALL문의 인수로 표의 상수 및 기호 문자의 사용을 허용하지 않습니다. 따라서 변환기에서 CALL문으로 변환되는 EXEC CICS 명령에 표의 상수 또는 기호 상수를 사용하지 마십시오. 이 제한사항에는 다음과 같은 한 가지 예외가 있습니다. 표의 상수는 올바른 데이터 유형인 경우 **pass** 값에 대한 인수로 EXEC CICS 명령에서 허용 가능합니다. 예를 들어, LENGTH 옵션에서 숫자 표의 상수를 사용할 수 있습니다.

COBOL 프로그램의 일괄처리 컴파일

개별 COBOL 프로그램을 한 개의 입력 파일로 함께 컴파일할 수 있습니다. END PROGRAM 헤더 명령문은 각 프로그램을 종료하고 일괄처리 내 마지막 프로그램에 대해서 선택적입니다. 변환기는 개별 COBOL 프로그램을 단일 입력 파일로 승인하고 END PROGRAM 헤더 명령문을 해석합니다.

변환기를 호출할 때 매개변수로 지정되는 변환기 옵션은 전체 일괄처리 동안 유효하지만 단위를 시작하는 CBL 또는 PROCESS 카드에서 지정된 옵션이 컴파일 단위를 덮어쓸 수 있습니다.

컴파일 단위 옵션은 다음 우선순위에 따라 판별됩니다.

1. 사용자 수정 불가능 설치 옵션으로 고정된 옵션

2. 단위를 시작하는 CBL 또는 PROCESS 카드에서 지정된 옵션
3. 변환기를 호출할 때 지정된 옵션
4. 기본 옵션

컴파일에 대한 자세한 정보는 143 페이지의 제 9 장 『애플리케이션 설치』의 내용을 참조하십시오.

일괄처리 컴파일을 사용하는 경우 컴파일 및 연계 편집에 성공할 수 있도록 다음과 같은 추가 조치를 취해야 합니다.

- 컴파일러를 호출하는 JCL문 또는 각 최상위 레벨(비중첩) 프로그램에 대한 CBL문의 매개변수로 컴파일러 NAME 옵션을 포함시킵니다. 이렇게 하면 각 프로그램의 끝에 NAME문이 포함됩니다. 자세한 정보는 46 페이지의 그림 4의 내용을 참조하십시오.
- 각 오브젝트 모듈에 대한 CICS COBOL 스텝에 INCLUDE 및 ORDER문을 추가하도록 컴파일러 출력을 편집합니다. 이러한 명령문을 통해 연계 편집기는 스텝을 각 로드 모듈의 시작부에 포함시킵니다. 이러한 명령문은 규칙에서는 시작부에 있는 것으로 되어 있지만 모듈 임의의 위치에 있을 수 있습니다. 이러한 명령문을 각 NAME문 바로 앞 모듈 끝에 배치하는 것이 편리할 수도 있습니다. 47 페이지의 그림 5에서는 이러한 방식으로 편집 후 46 페이지의 그림 4의 출력을 보여줍니다.

배치 컴파일에서는 143 페이지의 제 9 장 『애플리케이션 설치』에서 설명된 프로시저를 변형해야 합니다. 다음은 제안 메소드입니다.

1. 제공된 카탈로그화된 프로시저 DFHYITVL을 두 개의 프로시저 PROC1(변환 및 컴파일 단계(TRN 및 COB 포함)) 및 PROC2(연계 편집기 단계 COPYLINK 및 LKED 포함)로 분할합니다.
2. PROC1에서 컴파일러의 EXEC문의 매개변수에 NAME 옵션을 추가하십시오. 그러면 다음과 같이 표시됩니다.

```
//COB      EXEC PGM=IGYCRCTL,REGION=...,
//          PARM='....,NAME,....',
```

3. PROC1에서는 컴파일러 출력 데이터 세트 LOADSET의 이름 및 배치를 변경하십시오. 최소한 데이터 세트 이름에서 이니셜을 제거하고 배치를 CATLG로 변경하십시오. 그러면 SYSLIN문이 다음과 같아야 합니다.

```
//SYSLIN DD DSN=LOADSET,DISP=(NEW,CATLG),
//          UNIT=&WORK,SPACE=(80,(250,100))
```

4. PROC1을 실행하십시오.

```

.....
....program a....
.....
NAME PROGA(R)
.....
.....
....program b....
.....
NAME PROGB(R)
.....
....program c....
.....
NAME PROGC(R)

```

그림 4. 편집 전 컴파일러 출력

5. 47 페이지의 그림 5에 표시된 대로 데이터 세트 LOADSET에서 컴파일러 출력을 편집하여 INCLUDE 및 ORDER문을 포함시키십시오. 일괄처리에서 다수의 프로그램을 사용하는 경우 단순한 프로그램 또는 REXX EXEC를 작성하여 ORDER 및 INCLUDE문을 삽입해야 합니다.

6. PROC2에서 CICS 스텝을 포함하는 라이브러리에 대한 DD문을 추가하십시오. 이 라이브러리의 표준 이름은 CICSTS53.CICS.SDFHLOAD입니다. 이 스텝의 INCLUDE문은 DD 이름으로 이 라이브러리를 참조합니다. 47 페이지의 그림 5에 서는 DD 이름 SYSLIB를 사용한 것으로(또는 이 라이브러리를 SYSLIB에 연결 함) 가정합니다. 제안되는 명령문은 다음과 같습니다.

```
//SYSLIB DD DSN=CICSTS53.CICS.SDFHLOAD,
//      DISP=SHR
```

7. PROC2에서 SYSLIN 연결을 다음과 같이 단일 명령문으로 바꿉니다.

```
//SYSLIN DD DSN=LOADSET,
//      DISP=(OLD,DELETE)
```

이 명령문에서는 컴파일러 출력 데이터 세트 LOADSET의 이름을 바꾼 것으로 가정합니다.

8. Run PROC2.

```

        ....program a....
        .....
INCLUDE SYSLIB(DFHELII)
ORDER DFHELII
NAME PROGA(R)
        .....
        .....
        ....program b....
        .....
        .....
INCLUDE SYSLIB(DFHELII)
ORDER DFHELII
NAME PROGB(R)
        .....
        ....program c....
        .....
INCLUDE SYSLIB(DFHELII)
ORDER DFHELII
NAME PROGC(R)

```

그림 5. 연계 편집기 입력

참고: DFHELII 스텝을 사용하도록 권장되지만 DFHECI는 계속 제공되어 사용할 수 있습니다.

중첩된 COBOL 프로그램

COBOL 프로그램에 COBOL 프로그램이 포함될 수 있습니다. 포함되는 프로그램은 포함하는 프로그램의 END PROGRAM문 바로 앞에 포함됩니다. 포함되는 프로그램도 포함하는 프로그램이 될 수 있습니다. 즉, 포함되는 프로그램 자체도 다른 프로그램을 포함할 수 있다는 의미입니다. 각 포함되는 프로그램 또는 포함하는 프로그램은 END PROGRAM문으로 종료됩니다.

중첩된 프로그램에 대한 올바른 호출 및 중첩된 프로그램의 COMMON 속성에 대한 설명은 *Enterprise COBOL for z/OS: Customization Guide*의 내용을 참조하십시오.

CICS 변환기는 최상위 레벨 프로그램과 중첩된 프로그램을 다르게 처리합니다.

이 변환기는 최상위 레벨 프로그램(다른 프로그램에 포함되지 않은 프로그램)을 한 가지 추가사항과 함께 일반적인 방식으로 변환합니다. 변환기는 WORKING-STORAGE SECTION에 있는 모든 변환기 생성 변수에 대해 GLOBAL 속성을 지정합니다.

변환기는 중첩되거나 포함되는 프로그램을 다음과 같이 특수한 방식으로 변환합니다.

- 아직 없는 경우에는 DATA DIVISION 및 LINKAGE SECTION이 추가됩니다.
- DFHEIBLK(실행 인터페이스 블록) 및 DFH통신 영역이 LINKAGE SECTION에 삽입됩니다.
- EXEC CICS 명령 및 CICS 기본 제공 함수가 변환됩니다.
- PROCEDURE DIVISION 헤더는 수정되지 않습니다.

- 미리 호출 지정에 사용되는 변환기 생성 임시 변수는 WORKING-STORAGE SECTION에 삽입되지 않습니다.

변환기는 첫 번째 비주석 레코드가 다음에 해당되는 경우 입력 소스가 최상위 레벨에서 시작하는 것으로 해석합니다.

- IDENTIFICATION DIVISION문
- CBL 카드
- PROCESS 카드

첫 번째 레코드가 위의 항목에 해당되지 않으면 변환기가 입력을 중첩된 프로그램의 PROCEDURE DIVISION의 일부로 처리합니다. 첫 번째 CBL 또는 PROCESS 카드는 최상위 레벨 프로그램 및 새 컴파일 단위의 시작을 표시합니다. 첫 번째 최상위 레벨 프로그램 앞에 있는 IDENTIFICATION DIVISION문은 새 중첩된 프로그램이 시작을 나타냅니다.

이러한 규칙의 실질적인 효과는 중첩된 프로그램을 개별 파일에 보관하여 별도로 변환할 수 없다는 것입니다. 최상위 레벨 프로그램 및 그에 직간접적으로 포함된 모든 프로그램은 단일 컴파일 단위를 구성하고 변환기에 함께 제출되어야 합니다.

중첩된 프로그램의 주석

변환기는 END PROGRAM문 뒤에 오는 주석을 입력 소스에서 다음 프로그램에 속하는 것으로 처리합니다. IDENTIFICATION DIVISION문 앞에 있는 주석은 목록에서 IDENTIFICATION DIVISION문 뒤에 표시됩니다.

혼동을 피하기 위해 항상 다음과 같이 주석을 배치합니다.

- 주석이 참조하는 프로그램을 시작하는 IDENTIFICATION DIVISION문 뒤
- 주석이 참조하는 프로그램을 종료하는 END PROGRAM문 앞

별도의 변환기를 사용하는 경우

EXEC CICS 명령을 포함하는 중첩된 프로그램에 대해 통합 CICS 변환기를 사용하지 않고 별도의 변환기를 사용하는 경우 이 섹션에서 설명한 대로 CALL 및 PROCEDURE DIVISION의 USING 구문에서 EIB 및 COMMAREA를 명시적으로 코드화해야 합니다.

통합 CICS 변환기를 사용하는 경우 이 조치는 EXEC CICS 명령을 포함하는 중첩된 프로그램에서 필요하지 않습니다. 컴파일러는 실제로 최상위 레벨 프로그램에서 DFHEIBLK 및 DFHCOMMAREA를 글로벌로 선언합니다. 이는 명시적 코드화가 필요하지 않음을 의미합니다.

별도의 변환기를 사용하는 경우 다음을 수행하십시오.

1. EXEC CICS 명령을 포함하는 각 중첩 프로그램에서 CICS 기본 제공 함수나 EIB 또는 COMMAREA에 대한 참조는 다음과 같이 DFHEIBLK 및 DFHCOMMAREA를 PROCEDURE DIVISION 헤더의 처음 두 매개변수로 코드화합니다.

```
PROCEDURE DIVISION USING DFHEIBLK
                        DFHCOMMAREA PARM1 PARM2 ...
```

2. EXEC CICS 명령을 포함하는 중첩 프로그램에 대한 모든 호출에서 CICS 기본 제공 함수나 EIB 또는 COMMAREA에 대한 참조는 다음과 같이 DFHEIBLK 및 DFHCOMMAREA를 CALL문의 처음 두 매개변수로 코드화합니다.

```
CALL 'PROGA' USING DFHEIBLK
                        DFHCOMMAREA PARM1 PARM2 ...
```

3. 최상위 레벨 프로그램과 EXEC CICS 명령을 포함하는 중첩 프로그램 간 제어 계층 구조의 일부를 구성하는 모든 호출에 대해 CICS 기본 제공 함수나 EIB 또는 COMMAREA에 대한 참조는 DFHEIBLK 및 DFHCOMMAREA를 CALL문의 처음 두 매개변수로 코드화합니다. 호출된 프로그램의 PROCEDURE DIVISION에서 DFHEIBLK 및 DFHCOMMAREA를 코드화하십시오. 이는 EIB 및 COMMAREA에 대한 주소 지정성이 최상위 레벨 프로그램에 직접적으로 포함되지 않은 프로그램에 전달될 수 있도록 하는 데 필요합니다.
4. 위에 나열된 이유로 중첩된 프로그램의 PROCEDURE DIVISION에서 DFHEIBLK 및 DFHCOMMAREA를 삽입할 필요가 없는 경우에 해당 프로그램에 대한 호출은 CALL문의 매개변수 목록에 DFHEIBLK 및 COMMAREA를 포함시키지 않아야 합니다.

중첩 프로그램의 예제

컴파일 단위는 최상위 레벨 프로그램 W와 그에 직접 포함되는 세 개의 중첩 프로그램 X, Y 및 Z로 구성됩니다.

프로그램 W

초기화 및 종료 중에 초기 CICS 처리 및 비CICS 파일 액세스를 수행하기 위해 Y 및 Z를 호출합니다. 기본 처리를 수행하기 위해 X를 호출합니다.

프로그램 X

비CICS 파일 액세스를 위해 Z를 호출하고 CICS 처리를 위해 Y를 호출합니다.

프로그램 Y

CICS 명령을 실행합니다. 비CICS 파일 액세스를 위해 Z를 호출합니다.

프로그램 Z

일괄처리 모드로 파일에 액세스합니다.

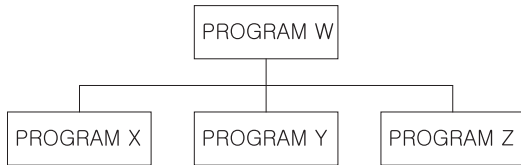


그림 6. 중첩된 프로그램 예제-중첩 구조

다음과 같이 규칙을 적용합니다.

- Y는 X로부터의 호출을 가능하게 하기 위해 COMMON이어야 합니다.
- Z는 X 및 Y로부터의 호출을 가능하게 하기 위해 COMMON이어야 합니다.
- Y는 CICS 명령을 실행하므로 별도의 변환기를 사용하는 경우:
 - Y에 대한 모든 호출에는 처음 두 매개변수로 DFHEIBLK 및 COMMAREA가 있어야 합니다.
 - Y의 PROCEDURE DIVISION 헤더에는 처음 두 매개변수로 DFHEIBLK 및 DFHCOMMAREA가 있어야 합니다.
- X는 EIB 또는 통신 영역에 액세스하지 않지만 CICS 명령을 실행하는 Y를 호출합니다. 따라서 별도의 변환기를 사용하는 경우 X에 대한 호출에는 처음 두 매개변수로 DFHEIBLK 및 COMMAREA가 있어야 하고 X의 PROCEDURE DIVISION 헤더에는 처음 두 매개변수로 DFHEIBLK 및 DFHCOMMAREA가 있어야 합니다.

51 페이지의 그림 7에서는 이러한 위치를 설명합니다.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. W.  
.  
.  
PROCEDURE DIVISION.  
.  
.  
    CALL Z.  
.  
.  
    CALL Y USING DFHEIBLK COMMAREA.  
.  
.  
    CALL X USING DFHEIBLK COMMAREA.  
.  
.  
IDENTIFICATION DIVISION.  
PROGRAM-ID. X.  
.  
.  
PROCEDURE DIVISION USING DFHEIBLK DFHCOMMAREA  
.  
.  
    CALL Z.  
.  
.  
    CALL Y USING DFHEIBLK COMMAREA.  
.  
.  
END PROGRAM X.  
IDENTIFICATION DIVISION.  
PROGRAM-ID. Y IS COMMON.  
.  
.  
PROCEDURE DIVISION USING DFHEIBLK DFHCOMMAREA.  
.  
.  
    CALL Z.  
.  
.  
    EXEC CICS...  
.  
.  
END PROGRAM Y.  
IDENTIFICATION DIVISION.  
PROGRAM-ID. Z IS COMMON.  
.  
.  
PROCEDURE DIVISION.  
.  
.  
END PROGRAM Z.  
END PROGRAM W.
```

그림 7. 중첩 프로그램 예제: 코딩

제 4 장 C 및 C++로 프로그래밍

이 정보를 사용하여 CICS 애플리케이션으로 사용하려는 C 및 C++ 프로그램을 코딩, 변환, 컴파일합니다.

고급 언어 지원은 z/OS용 CICS Transaction Server, 버전 5 릴리스 3 에서 지원되는 C 및 C++ 컴파일러와 z/OS의 컴파일러 서비스 상태를 나열합니다. z/OS용 CICS Transaction Server, 버전 5 릴리스 3 문서에서 C 및 C++에 대한 모든 참조는 다르게 명시되지 않았으면 지원되는 Language Environment 준수 컴파일러의 사용을 내포합니다. 구조화되지 않은 예외 처리에 관련된 명령을 제외하면 COBOL, PL/I, 어셈블러 언어 애플리케이션에서 사용 가능한 모든 **EXEC CICS** 명령도 C 및 C++ 애플리케이션에서 지원됩니다.

C++ 애플리케이션은 **EXEC CICS** 인터페이스 대신 CICS C++ OO 클래스를 사용하여 CICS 서비스에 액세스할 수도 있습니다. 이 인터페이스에 대한 자세한 정보는 애플리케이션 개발에서 CICS C++ OO 기반 클래스 사용의 내용을 참조하십시오. C++는 객체 지향 프로그래밍을 지원하며 C 언어를 사용한 동일한 방식으로 이 언어를 사용할 수 있습니다. 변환기가 CPP 옵션을 사용하여 C++를 변환함을 지정해야 합니다. C++ 프로그램도 LANGUAGE(LE370) 옵션으로 정의해야 합니다.

작업 중인 저장영역

C 및 C++에서 작업 중인 저장영역은 스택과 힙으로 이루어집니다. 16MB 라인에 대한 스택과 힙의 위치는 스택 및 힙 런타임 옵션의 ANYWHERE 및 BELOW 옵션으로 제어됩니다. 기본값은 스택과 힙이 모두 16MB 라인 위에 있는 것입니다.

샘플 프로그램

C 또는 C++ 언어로 쓴 프로그램에 **EXEC CICS** 명령을 사용할 수 있는 방식을 보여주는 샘플 애플리케이션 세트가 제공됩니다.

표 2. 샘플 프로그램

샘플 프로그램	맵핑 세트	맵 소스	트랜잭션 ID
DFH\$DMNU 운영자 명령어(3270)	DFH\$DGA	DFH\$DMA	DMNU
DFH\$DALL 갱신(3270)	DFH\$DGB	DFH\$DMB	DINQ, DADD, DUPD
DFH\$DBRW 찾아보기(3270)	DFH\$DGC	DFH\$DMC	DBRW
DFH\$DREN 주문 입력(3270)	DFH\$DGK	DFH\$DMK	DORD
DFH\$DCOM 주문 입력 큐 인쇄(3270)	DFH\$DGL	DFH\$DML	DORQ
DFH\$DREP 보고서(3270)	DFH\$DGD	DFH\$DMD	DREP

트랜잭션 및 프로그램 정의가 CSD의 DFH\$DFLA 그룹에 제공되며 다음 명령을 사용하여 설치할 수 있습니다.

CEDA INSTALL GROUP(DFH\$DFLA)

다음 레코드 설명 파일이 C 또는 C++ 언어 헤더 파일로 제공됩니다.

- DFH\$DFIL: FILEA 레코드 설명자
- DFH\$DL86: L860 레코드 설명자

FLOAT 컴파일러 옵션

z/OS V1.11 XL C(또는 C++) 이상의 경우 FLOAT(NOAFP) 컴파일러 옵션이나 FLOAT(AFP(VOLATILE)) 컴파일러 옵션을 지정하십시오.

- 프로그램이 부동 소수점을 거의 또는 전혀 사용하지 않는 경우에는 FLOAT(NOAFP) 옵션을 지정하십시오. 프로그램에 일반적인 네 개의 부동 소수점 레지스터만 사용되어 레지스터를 저장할 때 수행하는 작업이 감소합니다.
- 프로그램이 부동 소수점을 상당히 사용하는 경우에는 FLOAT(AFP) 옵션 또는 FLOAT(NOVOLATILE) 옵션을 지정하십시오. 프로그램에 16가지 모든 부동 소수점 레지스터를 사용할 수 있으며 CICS는 프로그램에 사용되는 부동 소수점 레지스터를 보존합니다.
- FLOAT(AFP(VOLATILE)) 옵션을 지정하는 경우에는 CICS, C, C++가 부동 소수점 레지스터를 보존합니다. 여분의 코드가 생성되므로 성능이 저하될 수 있습니다.

C 및 C++ 프로그래밍 제한사항과 요구사항

CICS 애플리케이션으로 사용되는 C 또는 C++ 프로그램에는 몇 가지 제한사항과 요구사항이 적용됩니다.

사용할 수 없는 함수 및 명령

구조화되지 않은 예외 처리에 관련된 다음 EXEC CICS 명령은 C 및 C++ 애플리케이션에 지원되지 않습니다.

- **HANDLE ABEND LABEL(label)**
- **HANDLE AID**
- **HANDLE CONDITION**
- **IGNORE CONDITION**
- **PUSH HANDLE**
- **POP HANDLE**

변환기는 이러한 명령의 사용을 진단합니다. **HANDLE ABEND PROGRAM** 명령은 허용됩니다.

CICS에서 `system()` 함수가 지원되지 않지만 두 개의 CICS 명령, **LINK** 및 **XCTL**이 동일한 기능을 제공합니다.

CICS는 확장 정밀도 부동 소수점을 지원하지 않습니다.

C++은 팩형 10진수 데이터를 지원하지 않습니다. 애플리케이션은 문자열 데이터 유형을 사용하여 팩형 10진수 데이터에 액세스합니다. 이 데이터에 대한 산술을 수행할 수 있는 C++ 표준 라이브러리 함수가 없지만 직접 함수를 쓸 수 있습니다. 시간을 지정하는 옵션이 있는 CICS 명령(예를 들어, **DELAY** 또는 **POST** 명령)을 사용할 때에는 **HOURS**, **MINUTES**, **SECONDS** 옵션을 사용할 것을 권장합니다. 팩형 10진수 데이터 유형인 **TIME** 또는 **INTERVAL** 옵션을 사용하여 시간을 정의할 수 있습니다(애플리케이션에서 이 유형을 처리할 함수를 제공하는 경우).

C 및 C++은 CICS 명령을 매크로로 사용하도록 지원하지 않습니다.

고유 C 또는 C++ 파일 조작은 `type=memory`가 지정된 채로 열린 파일에서만 작동합니다. CICS 지원 액세스 방법에 대한 I/O에 CICS API를 사용해야 합니다.

고유 C 및 C++ 함수는 소스 프로그램에서 허용되지만 다음 함수는 권장하지 않습니다. 일부 함수는 실행 가능하지 않으며 해당 함수가 실패했음을 나타내는 포인터나 리턴 코드를 유발합니다. 다른 일부는 작용하지만 CICS의 실행이나 성능에 영향을 미칠 수 있습니다.

- CDUMP
- CSNAP
- CTEST
- CTRACE
- CLOCK(`clock()` 함수는 `(time_t) -1` 값을 리턴합니다.)
- CTDLI
- SVC99
- SYSTEM
- SETLOCALE

코드화 요구사항

- 대문자만 사용해야 하는 `#pragma` 지시문의 CICS 키워드를 제외하면, 모든 CICS 키워드를 대소문자를 혼합하여 입력할 수 있습니다.
- CICS가 프로그램 이름, 맵 이름 또는 큐 이름과 같은 고정 길이 문자열을 예상하는 경우 리터럴이 예상보다 짧으면 리터럴을 필수 길이까지 공백으로 채워야 합니다. EXEC DLI 명령의 경우에는 리터럴이 전달되면 변환기가 **SEGMENT** 이름을 채웁니다.

- 어셈블리어에 허용 가능하지만 C 또는 C++ 컴파일러의 이상 종료를 유발하는 필드 이름은 사용하지 마십시오. \$, #, @가 포함된 이름이 그러한 예입니다.
- C++는 단일 행 주석에 '/'를 사용합니다. 이러한 주석을 EXEC CICS 명령의 가운데에 넣지 마십시오. 예를 들어, 다음 코드는 올바르지 않습니다.

```
EXEC CICS SEND TEXT FROM(errmsg)
          LENGTH(msglen) // Send error message to screen
          RESP(rcode)
          RESP2(rcode2);
```

다음 코드 예제는 올바릅니다.

```
EXEC CICS SEND TEXT FROM(errmsg)
          LENGTH(msglen)
          RESP(rcode)
          RESP2(rcode2);    //Send error message to screen

EXEC CICS SEND TEXT FROM(errmsg)
          LENGTH(msglen)    /* Send error message to screen */
          RESP(rcode)
          RESP2(rcode2);
```

조건 처리

C 또는 C++ 애플리케이션에서 모든 EXEC CICS 명령은 NOHANDLE 또는 RESP 옵션이 지정된 것처럼 처리됩니다. 따라서 발생했지만 처리되고 있지 않은 조건으로 인한 『시스템 조치』 트랜잭션 세트의 이상 종료가 가능하지 않습니다. 제어는 항상 다음 명령어로 플로우되며 애플리케이션이 정상 응답을 테스트해야 합니다.

COMMAREA

통신 영역의 주소는 C 또는 C++ 기본 함수에 인수로 전달되지 않습니다. 이는 C 및 C++ 함수가 ADDRESS COMMAREA를 사용하여 통신 영역의 주소를 얻어야 함을 의미합니다.

EIB

EXEC 인터페이스 블록(EIB)의 주소는 C 또는 C++ 기본 함수에 인수로서 전달되지 않습니다. 이는 C 및 C++ 함수가 ADDRESS EIB를 사용하여 EIB의 주소를 얻어야 함을 의미합니다. 자세한 정보는 60 페이지의 『C 및 C++에서 EIB 액세스』의 내용을 참조하십시오.

LENGTH

LENGTH를 지원하는 명령(예를 들어, **READ**, **READNEXT**, **READPREV**, **WRITE** 명령)에 LENGTH 옵션을 지정하지 않는 경우 변환기가 기본값을 제공하지 않습니다. 사실, NOLENGTH가 C 프로그램의 내재된 항목입니다.

OVERFLOW 조건

ACCUM 옵션이 지정된 **SEND MAP** 명령에서 리턴 시 OVERFLOW 조건을 RESP 필드에 표시하려면 NOFLUSH 옵션을 지정해야 합니다.

주소 지정 모드

CICS에서 실행하는 C 및 C++ 언어 프로그램은 AMODE(31), RMODE(ANY) 속성으로 링크 편집해야 합니다. 16MB 라인 위에 상주할 수 있습니다.

결과적으로 CSP(Cross-System Product) 대화식 애플리케이션 생성기로 생성한 프로그램에 매개변수를 전달할 때에는 다음 중 하나를 수행해야 합니다.

- 16MB 미만의 매개변수 전달
- CSP 로드 라이브러리를 AMODE(31)로 재링크

64비트 주소 지정 모드(AMODE(64))는 C 및 C++ 언어 프로그램에 지원되지 않습니다.

리턴값

C 또는 C++ 프로그램을 CICS **RETURN** 명령 대신에 리턴문이나 `exit()` 함수로 종료하는 경우 프로그램에서 리턴 시 `exit()` 함수를 통해 전달된 값은 EJB의 EIBRESP2 필드에 저장됩니다.

참고: 프로그램이 DPL을 사용하여 또 다른 CICS 리전의 프로그램에 링크하면 원격 리전의 EIBRESP2 값이 DPL을 수행하는 프로그램으로 리턴되지 않습니다.

데이터 선언

CICS는 C 및 C++에 대한 다음 데이터 선언을 제공합니다.

- 실행 인터페이스 블록 정의(EIB). EIB 선언은 `#ifndef` 및 `#endif` 행으로 에워싸여 있으며 모든 변환된 파일에 포함됩니다. C 또는 C++ 컴파일러는 중첩된 선언을 무시합니다. 삽입된 코드는 C 및 C++로 코딩된 EIB의 모든 필드 정의를 포함합니다.
- BMS 화면 속성 정의: DFHBMSCA, DFHMSRCA, DFHAID 파일의 C 및 C++ 버전은 CICS에서 제공되며 BMS를 사용할 때 애플리케이션 프로그래머가 이를 포함할 수 있습니다.
- DLI 지원: DFHDIB의 C 언어 버전은 변환기 옵션이 지정된 경우 DLI 변환기를 통해 포함됩니다. (CALL DLI 인터페이스가 사용되면 DLIUIB을 포함시켜야 합니다.)

페치 함수

Language Environment 준수 프로그램은 fetch() 및 release() 함수를 지원합니다. 페치할 모듈을 자동 설치를 통해 명시적 또는 내재적으로 CICS에 PROGRAM 자원으로 정의해야 합니다.

로케일 함수

CSD에 정의된 로케일에 대한 모든 로케일 함수가 지원됩니다. IBM 제공 로케일에 대한 CSD 정의는 SCEESAMP 라이브러리의 CEECCSD 멤버에 제공됩니다. 로케일이 정의되지 않은 경우 setlocale() 함수는 NULL을 리턴합니다.

디버깅 함수

덤프 함수 csnap(), cdump(), ctrace()가 지원됩니다. 출력은 CESE 트랜지언트 데이터 큐로 전송됩니다. 큐의 레코드 길이(LRECL)가 충분하지 않으면 덤프를 쓸 수 없습니다. 최소 161의 LRECL을 권장합니다.

iscics 함수

iscics() 함수는 CICS에서 그리고 CICS 외부에서 실행하도록 설계된 새 프로그램을 쓰거나 기존 프로그램을 채택하는 경우에 유용할 수 있습니다. 함수는 프로그램이 현재 CICS에서 실행 중인 경우 0이 아닌 값을 리턴하고 그렇지 않으면 0을 리턴합니다. 이 함수는 C 라이브러리의 확장입니다.

문자열 처리 함수

C 또는 C++ 표준 라이브러리의 문자열 처리 함수는 널(null) 문자를 문자열 끝 표시자로 사용합니다. CICS는 널을 문자열 끝 표시자로 인식하지 못하므로 CICS 데이터 영역에서 작동하기 위해 C 또는 C++ 함수(예를 들어, strcmp)를 사용할 때 주의해야 합니다.

argc 및 argv 인수

두 개의 argc 및 argv 인수는 일반적으로 C 또는 C++ 기본 함수에 전달됩니다. argc는 전달된 변수 수를 나타내며 argv는 0 종결 변수 문자열의 배열입니다. CICS에서 argc의 값은 1이고, argv[0]은 트랜잭션 ID이며, argv[1]은 NULL입니다.

C 및 C++의 인수 전달

C 및 C++ 언어는 런타임에 프로그램 스택에 복사되며 함수가 스택에서 이를 읽습니다. 이 인수는 자체적인 값이거나 전달 중인 데이터를 포함한 메모리 영역에 대한 포인터일 수 있습니다. 포인터 전달은 참조를 통한 값 전달이라고도 합니다.

COBOL 및 PL/I와 같은 기타 언어는 일반적으로 참조를 통해 인수를 전달하며 이는 컴파일러가 전달할 인수를 가리키는 주소 목록을 전달함을 의미합니다. 이는 CICS에서 지원되는 호출 인터페이스입니다. 참조를 통해 인수를 전달하려면 변수 이름에 **&** 접두부를 붙이십시오(배열이 전달되고 있을 때처럼 이미 포인터인 경우가 아니면).

빌드 프로세스의 일부로 컴파일러가 인수를 한 데이터 유형에서 다른 데이터 유형으로 변환할 수 있습니다. 예를 들어, **char** 유형의 인수가 **short** 유형이나 **long** 유형으로 변환될 수 있습니다.

C 또는 C++ 프로그램에서 CICS로 값을 전송할 때 변환기는 올바른 형식의 인수 목록이 CICS에 전달되도록 코드를 생성하는 데 필요한 조치를 수행합니다. 변환기는 이를 수행하는 데 사용할 충분한 정보가 항상 있는 것은 아니지만 일반적으로 인수가 단일 문자나 반자 변수이면 올바른 데이터 유형의 변수에 사전 호출 지정을 작성하고 이 임시 변수의 주소를 호출에 전달합니다.

CICS에서 데이터를 수신할 때 변환기는 수신한 변수 이름에 **&** 접두부를 붙여서 C 또는 C++ 컴파일러가 값을 통한 방식이 아닌 참조를 통해 값을 전달하게 합니다(변경되지 않는 문자열 이름은 예외). **&**를 추가하지 않을 경우 컴파일러는 수신 변수를 복사한 후 사본의 주소를 CICS에 전달합니다. 이 복사 중에 발생하는 승격으로 인해 CICS에서 리턴된 데이터가 유실될 수 있습니다.

표 3은 EXEC CICS 명령에 값을 인수로 전달할 때 적용되는 규칙을 보여줍니다.

표 3. EXEC CICS 명령의 인수로 값을 전달하기 위한 규칙

데이터 유형	사용법	인수 코딩
문자 리터럴	데이터 값(발신인)	사용자가 직접 문자 리터럴을 지정해야 합니다. 변환기는 필요한 간접 조치를 처리합니다.
문자 변수(char)	데이터 영역(수신인)	사용자가 아마도 변수 이름에 & 접두부를 붙여서 변수에 포인터를 지정해야 합니다.
문자 변수(char)	데이터 값(발신인)	사용자가 직접 문자 변수를 지정해야 합니다. 변환기는 필요한 간접 조치를 처리합니다.
문자열 문자	이름(발신인)	사용자가 문자열을 리터럴 문자열로 직접 코딩하거나 문자열의 첫 번째 문자를 가리키는 포인터를 사용할 수 있습니다.
문자열 변수	데이터 영역(수신인) 이름 (발신인)	수신 또는 전송과 무관하게, 인수는 문자열-배열의 첫 번째 요소의 주소를 포함한 문자 배열의 이름이어야 합니다.
정수 변수(short, long 또는 int)	데이터 영역(수신인)	사용자가 아마도 변수 이름에 & 접두부를 붙여서 변수에 포인터를 지정해야 합니다.
정수 변수(short, long 또는 int)	데이터 값(발신인)	사용자가 변수의 이름을 지정해야 합니다. 변환기는 필요한 간접 조치를 처리합니다.
정수 상수(short, long 또는 int)	데이터 값(발신인)	사용자가 직접 정수 상수를 지정해야 합니다. 변환기는 필요한 간접 조치를 처리합니다.
구조 또는 union	데이터 영역(발신인) 데이터 영역(수신인)	사용자가 아마도 이름에 & 접두부를 붙여서 구조나 union의 시작 주소를 코딩해야 합니다.

표 3. EXEC CICS 명령의 인수로 값을 전달하기 위한 규칙 (계속)

데이터 유형	사용법	인수 코딩
배열(어느 것이나)	데이터 영역(수신인) 데이터 값(발신인)	변환기가 아무 것도 수행하지 않습니다. 배열의 첫 번째 멤버의 주소를 코딩해야 합니다. 이는 일반적으로 컴파일러가 첫 번째 멤버의 주소로 해석하는 배열의 이름을 코딩해서 수행합니다.
포인터(어느 것이나)	Ptr-ref(수신인) 데이터 영역(발신인)	수신 또는 전송과 무관하게, 인수는 관심 있는 주소를 나타내는 변수의 이름이어야 합니다. 변환기는 CICS가 포인터를 갱신하도록 하는 데 필요한 추가 간접 레벨을 처리합니다.

참고: 수신인은 데이터가 CICS로부터 수신되는 위치이고 발신인은 데이터가 CICS로 전달되는 위치입니다.

C 및 C++에서 EIB 액세스

EXEC 인터페이스 블록(EIB)의 주소는 C 또는 C++ 기본 함수에 인수로서 전달되지 않습니다. 이는 C 및 C++ 함수가 ADDRESS EIB 명령을 사용하여 EIB의 주소를 얻어야 함을 의미합니다.

EIB에 액세스하려는 경우나 RESP 또는 RESP2 옵션을 포함한 명령을 사용 중인 경우 각 애플리케이션의 초반에 ADDRESS EIB문을 코딩해야 합니다.

주소 지정 가능성은 다음 명령을 사용하여

```
EXEC CICS ADDRESS EIB(dfheiptr);
```

또는 EIB 주소나 특정 필드를 외부 프로시저를 호출하는 CALL문의 인수로 전달해서 확보합니다.

EIB에 액세스해야 하는 경우에는 각 프로그램 초기에 ADDRESS EIB 명령이 필요합니다.

C 또는 C++ 애플리케이션 내에서 EIB의 필드는 소문자로 그리고 완전한 필드로 참조됩니다(예를 들어, “dfheiptr->eibtrnid”).

다음 데이터 유형 매핑이 사용됩니다.

- 반자 2진 정수는 “short int”로 정의됨
- 전자 2진 정수는 “long int”로 정의됨
- 단일 문자 필드는 “부호 없는 char”로 정의됨
- 문자열은 “부호 없는 char” 배열로 정의됨

C 및 C++에 대한 로케일 지원

CICS 변환기는 기본적으로 C 또는 C++ 언어로 쓴 프로그램은 EBCDIC Latin-1 코드 페이지 IBM-1047로 편집되었다고 가정합니다.

대체 코드 페이지를 사용한 경우 애플리케이션 초기의 pragma filetag 지시문에 이를 지정할 수 있습니다. pragma문은 프로그램의 첫 번째 비설명 문장이고 filetag 지시문은 pragma문의 다른 지시문 이전에 지정해야 합니다. CICS 변환기는 filetag 지시문의 존재 여부를 스캔합니다. CICS 변환기는 기본 코드 페이지 IBM-1047, 덴마크어 EBCDIC 코드 페이지 IBM-277, 독일어 EBCDIC 코드 페이지 IBM-273, 중국어 EBCDIC 코드 페이지 IBM-935 및 IBM-1388만 지원합니다.

예를 들어, 편집기로 독일어 EBCDIC 코드 페이지를 사용하여 프로그램을 준비했으면 프로그램은 다음 지시문으로 시작해야 합니다.

```
??=pragma filetag ("IBM-273")
```

애플리케이션이 여러 다른 코드 페이지를 혼합하여 사용하면(예를 들어, 보통 소스 파일에 사용된 것과 다른 코드 페이지로 편집된 헤더 파일을 포함하는 경우), 기본 코드 페이지 IBM-1047이더라도 모든 파일이 pragma filetag 지시문을 포함해야 합니다.

더 이상 서비스되지 않지만 CICS 변환기에는 여전히 사용될 수 있는 이전의 일부 IBM C 컴파일러는 pragma filetag 지시문을 지원하지 않을 수도 있습니다. 컴파일러가 이를 지원하는지 확실하지 않은 경우 컴파일러에 대한 문서를 확인하십시오.

XPLink와 C 및 C++ 프로그래밍

CICS는 XPLINK 옵션으로 컴파일한 C 및 C++ 프로그램에 대한 지원을 제공합니다. CICS XPLink 지원을 사용한 모든 프로그램은 재진입 가능하며 스레드세이프 방식이어야 합니다.

줄여서 XPLink라 부르는 Extra Performance Linkage는 고성능 서브루틴 호출과 리턴 메커니즘을 제공하는 z/OS 기능입니다. 이는 실행 경로 길이를 단축시키고 고도로 최적화시킵니다.

객체 지향 프로그래밍은 '메시지'를 오브젝트에 보내서 오브젝트가 어떤 조치를 수행하게 한다는 개념에 기반을 두고 있습니다. 메시지 전송 활동은 서브루틴 호출로 구현됩니다. C++ 용어로 멤버 함수라고 하는 서브루틴은 일반적으로 작은 코드 조각입니다. 일반 C++ 프로그램의 특징적 실행 플로우는 작은 코드 조각에 대한 많은 서브루틴 호출입니다. 이러한 성질의 프로그램은 XPLink 최적화 기술의 혜택을 받습니다.

MVS에는 초기 System/360 시기까지 거슬러 추적할 수 있는 표준 서브루틴 호출 규약이 있습니다. 이 규약은 서브루틴이 보다 복잡하고, 상대적으로 수가 적으며, 상대적으로 드물게 호출된 환경에 최적화되어 있습니다. 객체 지향 프로그래밍 규약은 이를 변

경했습니다. 서브루틴이 보다 단순하지만 수가 많아졌고 서브루틴 호출 빈도가 늘어난 수만큼 증가했습니다. 이러한 서브루틴의 크기, 수, 사용 패턴의 변화로 인해 관련된 시스템 오버헤드를 최적화시키는 것이 바람직해졌습니다. XPLink는 이 최적화의 결과입니다.

XPLink를 사용하려면 C 또는 C++ 애플리케이션 코드가 재진입가능 및 스레드세이프 방식이어야 합니다. 동일한 코드 인스턴스를 둘 이상의 MVS TCB에서 실행할 수 있으며 공유 자원을 보호하는 스레드세이프 메커니즘 없이는, 애플리케이션 코드의 실행 작동이 예측 불가능합니다. 이는 아무리 강조해도 지나치지 않은 사항입니다.

XPLINK 옵션으로 C 및 C++ 프로그램을 CICS 환경에 맞게 컴파일할 계획인 경우 애플리케이션 개발자는 CICS XPLink 지원을 이용해서 다음을 수행할 것이라 예상합니다.

- 스레드세이프 프로그래밍 원리와 기술을 엄격히 따르는 코드를 개발하십시오.
- XPLINK 옵션을 설정해서 C 또는 C++ 프로그램을 컴파일하십시오.
- PROGRAM 자원 정의에 프로그램이 스레드세이프 방식이라고 표시하십시오.
- CEEUOPT 또는 #pragma의 CICSVAR 사용을 고려하십시오. (세부사항은 19 페이지의 『Language Environment 런타임 옵션 정의』의 내용을 참조하십시오.)

CICS XPLink 지원을 사용한 모든 프로그램은 재진입 가능하며 스레드세이프 방식이어야 합니다. 애플리케이션 개발자만이 특정 애플리케이션에 대한 코드가 이 요구사항을 충족시킴을 보장할 수 있습니다.

XPLink의 X8 및 X9 모드 TCB 사용

CICS는 OTE(Open Transaction Environment) 기술의 다중 TCB 특성을 사용하여 XPLINK 옵션으로 컴파일한 C 및 C++ 프로그램에 대한 지원을 제공합니다. X8 및 X9 모드 TCB는 CICS 키와 USER 키의 XPLINK 태스크를 지원하도록 정의되어 있습니다. XPLink 프로그램의 각 인스턴스는 하나의 X8 또는 X9 TCB를 사용합니다.

XPLINK 옵션으로 컴파일한 프로그램에 대한 CICS 지원에 필요한 사항은 PROGRAM 자원 정의에 프로그램이 스레드세이프 방식이라고 표시하는 것 뿐입니다. 이 표시와 로드 모듈의 XPLink “서명”이 X8 또는 X9 TCB에 태스크를 두는 데 필요한 유일한 조건입니다.

특정 프로그램에 적합한 TCB 선택사항에서, XPLink는 PROGRAM 자원 정의의 API 속성에 대한 OPENAPI 값의 존재보다 우선합니다.

XPLink 및 비XPLink 오브젝트 간 제어 전달

XPLink 오브젝트에서 비XPLink 오브젝트로(또는 그 반대로) 제어가 전송될 때마다 QR TCB와 열려 있는 TCB(X8 또는 X9 TCB) 사이에 전환이 초래됩니다. 성능적인 면에서 TCB 전환은 비용이 많이 드는 작업이므로 이 성능 오버헤드를 고려해야 합니다.

XPLink 오브젝트는 EXEC CICS 인터페이스나 Language Environment 인터페이스를 사용하여 비XPLink 오브젝트를 호출할 수 있습니다.

비XPLink 오브젝트는 EXEC CICS 인터페이스만을 사용하여 XPLink 오브젝트를 호출할 수 있습니다. 이러한 호출에 대한 Language Environment 인터페이스의 사용은 지원되지 않습니다.

글로벌 사용자 엑시트 및 XPLink

XPCFTCH 및 XPCTA 엑시트는 XPLINK 옵션 사용의 영향을 받습니다. CICS는 XPCFTCH의 시작점을 수정하려는 시도와 XPCTA의 재개 주소를 정의하려는 시도를 무시합니다. 기타 글로벌 사용자 엑시트는 XPLink 지원의 영향을 받지 않습니다.

XPCFTCH

XPLink 옵션으로 컴파일한 C 또는 C++ 프로그램에 대해 XPCFTCH 엑시트가 호출되면 수정된 시작점 주소(엑시트를 통해 지정된 경우)를 무시함을 나타내는 플래그가 설정됩니다.

XPCTA

XPLink 옵션으로 컴파일한 C 또는 C++ 프로그램에 대해 XPCTA 엑시트가 호출되는 경우 재개 주소(엑시트를 통해 지정된 경우)를 무시함을 나타내는 플래그가 설정됩니다.

이 활동을 무시하는 이유는 XPLink 프로그램에 사용된 일괄처리 Language Environment 런타임이 프로그램이 이상 종료될 때 CICS에 제어를 주지 않고 자체 이상 종료 처리를 수행하기 때문입니다. 제어가 CICS에 도달할 때 Language Environment 인클레이브가 종료되었으므로 CICS는 시작점 주소나 재개 주소를 사용할 수 없습니다.

이러한 활동을 이행하는 애플리케이션이 있는 경우에는 이 요구사항을 관리할 다른 방안을 찾거나 프로그램이 XPLINK 최적화의 적합한 후보가 아니라고 결론내야 합니다. 가능한 한 가지 솔루션은 *z/OS Language Environment Customization* 매뉴얼의 “사용자 엑시트 사용자 정의”에 설명된 대로, Language Environment 비정상 종료 엑시트를 쓰는 방법입니다.

제 5 장 PL/I 애플리케이션 개발

CICS 애플리케이션을 사용할 PL/I 프로그램을 쉽게 코드화, 변환, 컴파일하려면 이 정보를 사용하십시오.

고급 언어 지원에서는 z/OS용 CICS Transaction Server, 버전 5 릴리스 3 에서 지원하는 PL/I 컴파일러와 z/OS에서의 해당 서비스 상태를 나열합니다.

z/OS용 CICS Transaction Server, 버전 5 릴리스 3 문서에서 PL/I에 대한 모든 참조는 별도로 언급하지 않는 한 지원되는 Language Environment 준수 컴파일러 사용을 나타냅니다.

OPTIONS(MAIN) 스펙

PL/I 애플리케이션에서 OPTIONS(MAIN) 옵션을 지정하면 해당 프로그램이 트랜잭션의 첫 번째 프로그램이 되거나 LINK 또는 XCTL 명령을 사용하여 제어를 전달할 수 있습니다.

OPTIONS(MAIN) 옵션이 지정되지 않은 PL/I 애플리케이션에서는 트랜잭션의 첫 번째 프로그램이 될 수도 없고 LINK 또는 XCTL 명령을 사용하여 제어를 전달할 수도 없지만 기본 프로그램에 링크 편집할 수 있습니다.

FLOAT 컴파일러 옵션

엔터프라이즈 PL/I의 경우, FLOAT 옵션을 지정하면 추가 부동 소수점 레지스터 사용을 제어합니다.

- 프로그램이 부동 소수점을 거의 또는 전혀 사용하지 않는 경우에는 FLOAT(NOAFP) 옵션을 지정하십시오. 프로그램은 기존의 4개의 부동 소수점 레지스터를 사용하며 레지스터 저장 시 수행할 작업이 적습니다.
- 프로그램이 부동 소수점을 상당히 사용하는 경우에는 FLOAT(AFP) 옵션 또는 FLOAT(NOVOLATILE) 옵션을 지정하십시오. 프로그램은 16개의 부동 소수점 레지스터를 모두 사용할 수 있으며 CICS는 프로그램에서 사용하는 부동 소수점 레지스터를 보존합니다.
- FLOAT(AFP(VOLATILE)) 옵션을 지정하면 CICS와 PL/I 둘 다 부동 소수점 레지스터를 보존합니다. 추가 코드가 생성되며 결과적으로 성능에 영향을 줄 수 있습니다.

PL/I 프로그래밍 제한사항 및 요구사항

몇 가지 제한사항 및 요구사항이 CICS 애플리케이션으로 사용되는 PL/I 프로그램에 적용됩니다.

사용할 수 없는 함수 및 명령문

- 다음 멀티태스킹 기본 제공 함수를 사용할 수 없습니다.

COMPLETION

PRIORITY

STATUS

- 다음 멀티태스킹 옵션을 사용할 수 없습니다.

EVENT

PRIORITY

TASK

- 다음 PL/I 명령문을 사용해서는 안됩니다.

CLOSE

DELAY

DELETE

DISPLAY

EXIT

GET

HALT

LOCATE

OPEN

PUT

READ

REWRITE

STOP

WRITE

UNLOCK

FETCH 및 RELEASE문이 지원됩니다. EXEC CICS 명령은 데이터 저장 및 검색, 터미널과의 통신을 위해 제공됩니다. 그러나 SYSPRINT에 CLOSE, PUT, OPEN을 사용할 수 있습니다.

- PL/I 정렬/병합을 사용할 수 없습니다.
- 정적 저장영역을 사용할 수 없습니다(읽기 전용 데이터의 경우 제외).

코드화 요구사항

- STATIC 속성 및 EXTERNAL 속성을 가진 변수를 선언하는 경우 INITIAL 속성도 포함시켜야 합니다. 포함시키지 않으면 이러한 선언은 CICS가 처리할 수 없는 공통 CSECT를 생성합니다.
- 변환기가 생성한 변수 이름과 동일한 변수 이름을 가진 변수 또는 구조를 정의하지 마십시오. 이는 DFH로 시작합니다. 이러한 변수 이름이 내재적으로 생성되지 않도록 LIKE 키워드를 주의하여 사용하십시오.
- 소문자일 수 있는 PROCEDURE 이름을 제외하고 모든 PROCEDURE문은 대문자여야 합니다.
- *PROCESS문의 XOPTS 옵션의 서브옵션은 대문자여야 합니다.
- EXEC CICS문에서 PL/I 48자 설정 옵션을 사용할 수 없습니다.
- CICS 명령이 데이터 값을 정의할 때 SUBSTR 기본 제공 함수를 사용하는 경우, 변환기 옵션 NOLENGTH가 지정되지 않으면 LENGTH 옵션을 포함하여 데이터 길이를 지정해야 합니다. 그렇지 않으면 변환기가 다음 양식의 CSTG 기본 제공 함수 호출을 포함하여 PL/I 호출을 생성합니다.

```
CSTG(SUBSTR(...,...))
```

이 호출은 컴파일러가 거부합니다.

64비트 주소 지정

64비트 주소 지정 모드는 PL/I 프로그램에 지원되지 않습니다.

PL/I 애플리케이션에 대한 Language Environment 코드화 요구사항

모든 PL/I 프로그램은 Language Environment가 제공한 런타임 지원 하에서 실행됩니다. Language Environment 이전 PL/I 프로그램과 비교하여 몇 가지 추가 코드화 요구사항이 있습니다.

필요하면 Language Environment 런타임 옵션을 **plixopt** 문자열에 지정할 수 있습니다. 런타임 옵션 사용자 정의에 대한 정보는 19 페이지의 『Language Environment 런타임 옵션 정의』 및 *z/OS Language Environment Programming Reference*를 참조하십시오.

비Language Environment 준수 컴파일러를 사용하여 이전에 컴파일된 PL/I 프로그램을 변환하는 경우 NOSTAE 및 NOSPIE가 **plixopt**에 지정되지 않았는지 확인해야 합니다. 이 중 하나는 Language Environment가 TRAP (OFF)를 설정하게 하기 때문입니다. 애플리케이션을 정상적으로 실행하려면 TRAP (ON)을 적용해야 합니다.

시작점

CEESTART는 Language Environment에서 실행되는 PL/I 애플리케이션의 유일한 시작점입니다. 이 시작점은 Language Environment 준수 컴파일러를 사용하여 컴파일된 프로그램에 대해 설정됩니다.

다음 연계 편집기 명령문을 사용하여 Language Environment에서 실행되도록 비 Language Environment 준수 컴파일러가 생성한 오브젝트 모듈을 재링크할 수 있습니다.

```
INCLUDE SYSLIB(CEESTART)
INCLUDE SYSLIB(CEESG010)
INCLUDE SYSLIB(DFHELII)
REPLACE PLISTART
CHANGE PLIMAIN(CEEMAIN)
INCLUDE mainprog
INCLUDE subprog1
.....
.....
ORDER CEESTART
ENTRY CEESTART
NAME progname(R)
```

오브젝트 모듈에 대한 INCLUDE문은 CHANGE문 바로 뒤에 와야 합니다. 기본 프로그램이 서브루틴 앞에 포함되어야 한다는 Language Environment 요구사항도 있습니다(이 요구사항은 비준수 컴파일러가 생성한 모듈에 대해서는 존재하지 않음).

OPTIONS(FETCHABLE)을 사용하여 컴파일된 엔터프라이즈 PL/I 프로그램의 경우, 바인더 ENTRY문이 PROCEDURE 이름이어야 합니다.

PL/I용 재링크 유틸리티

비준수 컴파일러로 컴파일된 CICS 프로그램에 대한 로드 모듈만 있는 경우, 특별히 CICS 프로그램용 연계 편집기 입력 파일인 IBMWRLKC가 있습니다(샘플 라이브러리 SCEESAMP에 있음). 이 입력 파일은 비준수 실행 프로그램의 OS PL/I 라이브러리 루틴을 Language Environment 루틴으로 대체합니다.

IBMWRLKC 사용에 대한 자세한 정보는 *PL/I MVS & VM 컴파일러 및 런타임 매이그레이션 안내서*를 참조하십시오.

준수 및 비준수 PL/I 루틴 간의 통신

Language Environment 준수 PL/I 프로그램은 FETCH 또는 RELEASE문에 나타나는 프로그램을 호출(CALL)할 수 있으며 나중에 해제(RELEASE)할 수 있습니다.

비Language Environment 준수 PL/I 서브루틴을 Language Environment 준수 기본 프로그램에 링크 편집할 수 있습니다.

정적 호출은 임의 버전의 PL/I에서 지원되지만 동적 호출은 Language Environment 준수 프로시저에서만 지원됩니다.

서브루틴에서 EIB 주소를 사용할 수 있는 경우 호출된 서브루틴은 CICS 명령을 실행할 수 있습니다. EIB 주소를 서브루틴에 전달하거나 다른 CICS 명령을 실행하기 전에 서브루틴에서 EXEC CICS ADDRESS EIB(DFHEIPTR)을 코드화하여 이를 수행할 수 있습니다.

이상 종료 처리

CICS PL/I 프로그램이 Language Environment에서 이상 종료되면 CICS 이상 종료 핸들러에는 PL/I 이상 종료 코드가 아닌 Language Environment 이상 종료 코드가 부여됩니다.

프로그램 변경을 방지하려면 Language Environment가 SCEESAMP 라이브러리에 제공하는 샘플 사용자 조건 핸들러인 CEEWUCHA를 수정할 수 있습니다. Language Environment 코드 대신 PL/I 이상 종료 코드를 리턴하도록 이 사용자 조건 핸들러를 작성할 수 있습니다. 이렇게 하려면 USRHDLR 런타임 옵션을 사용하여 등록하십시오. 이 옵션 세부사항은 *z/OS Language Environment Programming Guide*를 참조하십시오.

샘플 사용자 조건 핸들러인 CEEWUCHA를 런타임 시 사용할 수 있는지 확인하십시오(예: STEPLIB 연결 또는 LPA 사용). 프로그램 자동 설치를 사용하지 말고 CICS 리전에 대한 CICS 시스템 정의 데이터 세트(CSD)에 조건 핸들러를 정의하십시오.

폐치된 PL/I 루틴

PL/I 프로시저를 폐치할 수 있게 하려면 PROCEDURE문의 OPTIONS에서 FETCHABLE 옵션을 코드화하십시오.

FETCHABLE 옵션은 프로시저를 동적으로만 호출할 수 있음을 나타냅니다. OPTIONS(MAIN) 프로시저는 폐치할 수 없습니다. 즉, FETCHABLE과 MAIN은 상호 배타적 옵션입니다.

OPTIONS(FETCHABLE)을 사용하여 컴파일된 엔터프라이즈 PL/I 프로그램의 경우, 바인더 ENTRY문이 PROCEDURE 이름이어야 합니다.

FETCHABLE 프로시저를 정상 CICS 프로그램으로 간주하십시오. 즉, 필수 서브루틴에 링크 편집되고 CICS 애플리케이션 라이브러리에 배치되며, CSD에 또는 프로그램 자동 설치를 사용하여 프로그램으로 설치됩니다.

Language Environment 준수 PL/I 프로그램은 FETCH 또는 RELEASE문에 나타나는 프로그램을 호출(CALL)할 수 있으며 나중에 해제(RELEASE)할 수 있습니다.

페치된 프로시저에서 사용할 수 있는 MVS 및 VM 명령문의 경우 PL/I에 대한 몇 가지 제한사항이 있습니다. 이러한 제한사항은 *PL/I MVS & VM 언어 참조서*에 설명되어 있습니다. VisualAge® PL/I로 많은 제한사항이 제거되었습니다. *VisualAge PL/I 컴파일러 및 런타임 마이그레이션 안내서*를 참조하십시오.

페치하는 프로그램과 페치된 프로그램 둘 다에 동일한 속성이 있으면 FETCH 사용에 적용되는 특별한 고려사항이 없습니다. 그러나 Language Environment는 FETCH를 실행하는 프로그램과 다른 AMODE 속성을 가진 로드 모듈 페치도 지원합니다. 이 경우, Language Environment는 AMODE 전환을 수행하며 다음 제한조건이 적용됩니다.

- 페치된 모듈이 24비트 주소 지정 모드에서 실행될 경우, 페치하는 모듈에는 AMODE 속성과 관계없이 RMODE(24) 속성이 있어야 합니다.
- 페치된 프로시저의 AMODE에서 페치된 루틴에 전달된 변수의 주소를 지정할 수 있어야 합니다.

제 6 장 어셈블러 언어 애플리케이션 개발

이 정보를 사용하여 CICS 애플리케이션으로 사용하려는 어셈블러 언어 프로그램을 코딩합니다.

작업 중인 저장영역

어셈블러 언어 프로그램의 작업 중인 저장영역은 CSD에서 PROGRAM 정의의 **DATALOCATION** 매개변수의 값에 따라 16MB 라인 위나 아래로 할당됩니다.

샘플 프로그램

어셈블러 언어로 쓴 프로그램에 **EXEC CICS** 명령을 사용할 수 있는 방식을 보여주는 샘플 애플리케이션 세트가 제공됩니다. 이 프로그램은 AMODE(64)이며 DFH\$AREP를 제외하면 상대 주소 지정을 사용합니다. DFH\$AREP는 상대 주소 지정을 사용하지만 HANDLE CONDITION LABEL 명령 사용을 시연하므로 AMODE(31)입니다.

표 4. 샘플 프로그램

샘플 프로그램	맵핑 세트	맵 소스	트랜잭션 ID
DFH\$AMNU 운영자 명령어(3270)	DFH\$AGA	DFH\$AMA	AMNU
DFH\$AALL 갱신(3270)	DFH\$AGB	DFH\$AMB	AADD, AINQ, AUPD
DFH\$ABRW 찾아보기(3270)	DFH\$AGC	DFH\$AMC	ABRW
DFH\$AREN 주문 입력(3270)	DFH\$AGK	DFH\$AMK	AORD
DFH\$ACOM 주문 입력 큐 인쇄(3270)	DFH\$AGL	DFH\$AML	AORQ
DFH\$AREP 보고서(3270)	DFH\$AGD	DFH\$AMD	AREP

트랜잭션 및 프로그램 정의가 CSD의 DFH\$AFLA 그룹에 제공되며 다음 명령을 사용하여 설치할 수 있습니다.

```
CEDA INSTALL GROUP(DFH$AFLA)
```

다음 레코드 설명 파일이 제공됩니다.

- DFH\$AFIL: FILEA 레코드 설명자
- DFH\$AL86: L860 레코드 설명자

어셈블러 언어 프로그래밍 제한사항 및 요구사항

CICS 애플리케이션으로 사용되는 어셈블러 언어 프로그램에는 몇 가지 제한사항과 요구사항이 적용됩니다.

LEASM 옵션

다음 제한사항은 LEASM 옵션으로 변환되는 어셈블러 언어 프로그램에 적용됩니다.

- 레지스터 2는 코드 베이스 레지스터로 사용할 수 없습니다.
- 레지스터 12는 Language Environment(LE)에 Language Environment 공통 앵커 영역(CAA)을 가리키도록 예약되어 있으므로 적합하게 저장 및 복원하지 않고는 프로그램에 사용할 수 없습니다.
- 레지스터 13은 작업 중인 저장영역 기본 레지스터로만 사용해야 합니다.
- 프로그램이 글로벌 사용자 엑시트 프로그램(GLUE) 또는 태스크 관련 사용자 엑시트 프로그램(TRUE)이면 안됩니다.
- 프로그램이 AMODE(24) 코드를 사용하거나 이 코드에 의존해선 안됩니다.
- AMODE(64) 프로그램은 지원되지 않습니다.

EXEC CICS 명령의 LENGTH 옵션

LENGTH 옵션을 CICS 명령에 지정하는 경우 어셈블러 언어에 사용하는 방식이 올바른 반자 길이를 지정하는지 확인하십시오. 길이가 0이거나 CICS 변환기가 인식할 수 없는 변수는 지정하지 마십시오. 저장영역 위반 또는 프로그램 검사를 유발할 수 있습니다. 자세한 정보는 시스템 프로그램 개발의 어셈블러 언어 인수 값의 내용을 참조하십시오.

다음 LINK 명령 예제는 길이를 제대로 지정하며 변수 COMMAL의 주소를 명령 프로세서에 전달합니다.

```
EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMMA) LENGTH(COMMAL)
...
COMMA DC CL20'This is the COMMAREA'
COMMAL DC H'20'
```

다음 예제도 길이를 제대로 지정합니다.

```
EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMMA) LENGTH(=AL2(COMMAL))
...
COMMA DC CL20'This is the COMMAREA'
COMMAL EQU *-COMMA
```

다음 예제는 CICS 변환기가 변수 COMMAL의 유형을 알 수 없으므로 올바르지 않으며, COMMAL의 값을 길이를 포함한 반자 필드의 주소로서 전달합니다. 이는 무작위 길이 값을 제공할 수 있습니다. 또는 프로그램이 이 주소의 저장영역이 사용 가능하지 않은지 확인합니다.

```
EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMMA) LENGTH(COMMAL)
...
COMMA DC CL20'This is the COMMAREA'
COMMAL EQU *-COMMA
```

31비트 주소 지정

다음 제한사항은 31비트 주소 지정 모드에서 실행하는 어셈블러 언어 애플리케이션에 적용됩니다.

- 혼합 주소 지정 모드 트랜잭션 환경에서는 COMMAREA 옵션이 제한됩니다. 제한 사항에 대한 논의는 372 페이지의 『혼합 주소 지정 모드 사용』의 내용을 참조하십시오.
- CICS는 DFHEIENT 및 DFHEIRET 매크로를 사용하지 않는 어셈블러 언어 프로그램에 HANDLE ABEND LABEL의 사용을 허용하지 않습니다. Language Environment 스텝 CEESTART를 사용하는 어셈블러 언어 프로그램은 HANDLE ABEND PROGRAM 또는 Language Environment 서비스(예를 들어, CEEHDLR)를 사용해야 합니다. CEEHDLR에 대한 정보는 13 페이지의 『Language Environment 이상 종료 및 조건 처리』의 내용을 참조하십시오.

다음 제한사항은 64비트 주소 지정 모드나 64비트 2진 오퍼레이션을 사용하기 위해 64비트 레지스터를 사용하는 AMODE(24) 또는 AMODE(31) 어셈블러 언어 애플리케이션에 적용됩니다.

- CICS는 64비트 레지스터의 상위 순서 단어를 항상 보존하지는 않습니다. CICS 서비스를 호출하기 전에 저장하고 64비트 레지스터를 다시 사용하기 전에 복원해야 합니다.

64비트 주소 지정

CICS는 비Language Environment AMODE(64) 어셈블러 언어 CICS 애플리케이션의 프로그램 실행을 지원합니다. 프로그램은 상대 주소 지정을 사용해야 합니다. 88 페이지의 『AMODE(64) 어셈블러 언어 프로그램 개발』의 내용을 참조하십시오. 이 섹션의 후반부에 나열된 명령을 제외한 모든 CICS API 명령이 지원됩니다. CICS API에 대한 정보는 참조 -> 애플리케이션 개발에서 CICS API 명령의 내용을 참조하십시오.

다음 API는 AMODE(64) 프로그램에 지원되지 않습니다.

- CICSplex[®] SM API(Application Programming Interface)
- APPC 기본 대화에 사용할 CICS API 명령(GDS 명령)
- FEPI(Front End Programming Interface)
- 공통 프로그래밍 인터페이스(CPI) 통신 API
- DL/I 요청

다음 인터페이스는 AMODE(64) 프로그램에 지원되지 않습니다.

- CICS DB2 인터페이스
- CICS-MQ 브릿지 외부 인터페이스

다음 CICS API 명령은 AMODE(64) 프로그램 용도로 제공됩니다.

- FREEMAIN64
- GETMAIN64
- GET64 CONTAINER
- PUT64 CONTAINER

구조화되지 않은 예외 처리에 관련된 다음 CICS API 명령은 AMODE(64) 어셈블러 언어 프로그램에 지원되지 않습니다.

- HANDLE ABEND LABEL(label)
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION
- POP HANDLE
- PUSH HANDLE

변환기는 이러한 명령의 사용을 발견합니다.

AMODE(64) 어셈블러 언어 프로그램은 COBOL, C, C++ 또는 PL/I CALL문이나 어셈블러 언어 CALL 매크로로 호출할 수 없습니다. 하지만 어셈블러 언어 애플리케이션은 LINK나 XCTL 명령을 사용하여 COBOL, C, C++, PL/I 또는 어셈블러 언어 애플리케이션으로 호출할 수 있습니다.

AMODE(64) 어셈블러 언어 프로그램의 경우 모든 EXEC CICS 명령에 NOHANDLE 조건의 사용이 내포됩니다. 이는 명령 실행으로 인한 조건에 대해서는 아무런 조치를 취하지 않음을 의미합니다. RESP 옵션을 명령과 함께 사용하여 해당 명령 실행 중 조건이 제기되었는지 여부를 테스트할 수 있습니다. 참조 -> 애플리케이션 개발에서 RESP 및 RESP2 옵션 및 참조 -> 애플리케이션 개발에서 NOHANDLE 옵션의 내용을 참조하십시오.

AMODE(64) 어셈블러 언어 프로그램의 경우 CICS 64비트 레지스터 전체를 사용합니다. 모든 레지스터가 올바른지 확인하십시오.

COMMAREA가 64비트 저장영역에 있으면 안됩니다. COMMAREA에 대한 자세한 정보는 367 페이지의 『COMMAREA』의 내용을 참조하십시오.

AMODE 64 태스크 관련 사용자 엑시트(TRUE)가 지원되지 않으며 AMODE(64) 애플리케이션은 TRUE를 호출할 수 없습니다.

64비트 주소 지정 모드 및 64비트 2진 오퍼레이션 사용에 대한 자세한 정보는 z/OS MVS Programming: Assembler Services Guide의 내용을 참조하십시오.

액세스 레지스터

다음 제한사항은 액세스 레지스터를 사용하여 z/Architecture® 시스템의 확장된 주소 지정 가능성을 사용하는 어셈블러 언어 애플리케이션에 적용됩니다.

- CICS 서비스를 호출할 때 기본 주소 지정 모드에 있어야 합니다. 기본 주소 공간은 홈 주소 공간이어야 합니다. CICS에 전달된 모든 매개변수는 기본 주소 공간에 상주해야 합니다.
- CICS는 액세스 레지스터를 항상 보존하지는 않습니다. CICS 서비스를 호출하기 전에 저장하고 액세스 레지스터를 다시 사용하기 전에 복원해야 합니다.

액세스 레지스터 사용에 대한 추가 안내 정보는 z/OS MVS Programming: Extended Addressability Guide의 내용을 참조하십시오.

BAKR 명령어(분기 및 스택)

BAKR 명령어(분기 및 스택)를 사용하여 어셈블러 언어 프로그램 간의 연계를 제공하는 경우 링크된 프로그램이 EXEC CICS 요청을 발행하지 않는지 확인하십시오. CICS가 제어를 수신하고 링크된 프로그램이 PR 명령어(프로그램 리턴)를 통해 태스크 전환을 수행하면 기타 태스크가 디스패치되어 추가 BAKR / PR 호출을 발행할 수도 있습니다. 이 호출은 연계 스택을 수정해서 원래 태스크가 PR 명령어를 실행할 때 잘못된 환경이 복원되도록 합니다.

사용할 수 없는 명령어

CICS 애플리케이션으로 사용되는 어셈블러 언어 프로그램에는 다음 명령어를 사용할 수 없습니다.

COM 공백의 공통 제어 섹션을 식별합니다.

ICTL Input Format Control.

OPSYN

Equate Operation Code.

설명

CICS 명령에 대해 설명을 추가하려는 경우 마지막 인수 다음에 마침표 또는 쉼표를 구분 기호로 사용하여 어셈블러 언어에서만 추가할 수 있습니다. 예를 들어, 다음과 같습니다.

```
EXEC CICS ADDRESS EIB(MYUEIB), @F1A
```

EXEC CICS 명령에 마침표나 쉼표가 사용되는 경우 다음 행이 2열과 16열 사이에서 시작하고 연속 문자는 72열에 있어야 합니다. 다음 행은 17열 뒤에서 시작할 수 없습니다. 쉼표나 마침표가 추가되지 않은 경우에는 다음 행이 2열 또는 2열 다음에서 시작하고 71열에서 끝나며 연속 문자는 72열에 있어야 합니다.

EXEC CICS 어셈블러 인터페이스 코딩

AMODE(24) 및 AMODE(31) 어셈블러 언어 프로그램의 경우 선택적으로 사용자 자신의 DFHEIENT 매크로를 지정할 수 있습니다. 어셈블러 언어 프로그램이 AMODE(64) 임을 선언하기 위해 z/OS 어셈블러 SYSSTATE 매크로를 사용합니다. AMODE(64) 어셈블러 언어 프로그램의 경우 DFHEIENT 매크로를 지정해야 하며 선택적으로 사용자 자신의 DFHEIRET 매크로를 지정할 수 있습니다.

AMODE(24) 및 AMODE(31) 어셈블러 언어 프로그램

DFHEIENT 매크로

AMODE(24) 및 AMODE(31) 프로그램의 경우, 변환기가 자동으로 삽입하는 DFHEIENT 매크로에 의해 제공되는 값이 4095바이트보다 큰 변환된 출력을 생성하는 애플리케이션에 충분하지 않을 수 있습니다. 이 상황에서는 사용자 자신의 DFHEIENT 매크로 버전을 제공할 수 있습니다. 변환기가 자동으로 DFHEIENT 매크로의 해당 버전을 삽입하지 못하게 하려면 NOPROLOG 변환기 옵션을 지정하십시오.

예를 들어, 기본적으로 변환기는 단 하나의 기본 레지스터(레지스터 3)를 설정 하는데, 이것은 프로그램의 주소 지정 가능성 문제점을 유발할 수 있습니다. 사용자 자신의 DFHEIENT 매크로를 CODEREG 피연산자와 함께 제공하여 둘 이상의 기본 레지스터를 지정할 수 있습니다. CSECT문과 동일한 레이블을 갖는 DFHEIENT 매크로의 사용자 고유 버전을 코딩하는 경우, 소스 프로그램의 CSECT문을 대체할 수 있습니다. DFHEIENT 매크로를 레이블 없이 코딩하는 경우 CSECT문 바로 뒤에 와야 합니다.

다음 피연산자를 사용하여 기본 레지스터를 지정할 수 있습니다.

- CODEREG - 기본 레지스터(레지스터 13,14,15, 0 및 1은 허용되지 않음)
- DATAREG - 동적 저장영역 레지스터
- EIBREG - EIB를 주소 지정하는 레지스터

예를 들어, 다음의 간단한 어셈블러 애플리케이션은 BMS 명령 SEND MAP을 사용하여 맵을 터미널로 보냅니다.

```
INSTRUCT CSECT
      EXEC CICS SEND MAP('DFH$AGA') MAPONLY ERASE
      END
```

다음 코드 예제는 이 프로그램에 대한 기본 및 데이터 레지스터의 수를 늘립니다.

```
INSTRUCT DFHEIENT CODEREG=(2,3,4),
      DATAREG=(13,5),
      EIBREG=6
```

```
EXEC CICS SEND
MAP('DFH$AGA')
MAPONLY ERASE
END
```

기호 레지스터 DFHEIPLR이 명시적으로 지정되거나 기본적으로 얻어지는 첫 번째 DATAREG에 동일시됩니다. DFHECALL 매크로는 레지스터 13이 DFHEISTG가 동적 저장영역에서 정의한 저장 영역을 주소 지정하도록 보장하므로, 레지스터 13을 첫 번째 데이터 동적 저장영역 레지스터로서 사용하는 것이 바람직합니다. 그렇지 않은 경우, DFHECALL에 의해 생성되는 코드가 레지스터 13을 조작하기 위한 추가 명령어를 추가합니다.

DFHEIPLR은 DFHEIENT에 의해 설정된 값을 포함하기 위해 CICS 명령의 확장에 의해 가정됩니다. 이 레지스터를 전용하거나 각 CICS 명령 전에 복원되는지 확인해야 합니다.

또한 DFHEIENT 매크로를 사용하여 사용자 프로그램에서 상대 주소 지정 명령어를 사용할 것을 지정할 수도 있습니다. 상대 주소 지정을 사용할 때, 프로그램 명령어를 주소 지정하기 위해 어떤 기본 레지스터도 사용할 필요가 없지만 프로그램의 정적 데이터를 주소 지정하기 위해 하나 이상의 레지스터를 사용해야 합니다. DFHEIENT 매크로에서 다음 피연산자를 지정하십시오.

- 프로그램 명령어를 주소 지정하기 위해 레지스터가 사용되지 않도록 지정하려면 CODEREG=0을 지정하십시오.
- 프로그램에서 정적 데이터 영역을 주소 지정하기 위해 하나 이상의 레지스터를 지정하려면 STATREG를 지정하십시오.
- 프로그램에서 정적 데이터의 시작 주소를 지정하려면 STATIC을 지정하십시오.

상대 주소 지정을 사용하는 경우 IEABRCX DEFINE 매크로(z/OS가 제공함)를 사용하여 상대 분기 명령어를 사용하기 위한 분기 명령어에 대한 어셈블러 니모닉을 재정의하십시오. 또한, 모든 LTORG 명령문과 EXECUTE 명령의 대상인 명령어가 STATIC 피연산자에서 지정되는 레이블 뒤에 나타나는지 확인하십시오. 예를 들어, 다음과 같습니다.

IEABRCX DEFINE		Define relative branch mnemonics
RELATIVE	DFHEIENT CODEREG=0,STATREG=(8,9),STATIC=MYSTATIC	
....		
EX	R2,VARMOVE	Execute instruction in static area
....		
MYSTATIC	DS 0D	Static data area
MYCONST	DC C'constant'	Static data value
VARMOVE	MVC WORKA(0),WORKB	Executed instruction
	LTORG ,	Literal pool

IEABRCX 매크로에 관한 자세한 정보는 z/OS MVS Programming: Assembler Services Reference, Volume 2의 IEABRCX - 상대 분기 매크로 확장의 내용을 참조하십시오.

DLI 옵션으로 변환되는 어셈블러 언어 프로그램은 각 CSECT문 뒤에 삽입된 DLI 초기화 호출을 갖습니다. 4095바이트보다 크고 다중 기본 레지스터를 설정하기 위해 DFHEIENT 매크로의 CODEREG 피연산자를 사용하지 않는 어셈블러 언어 프로그램은 LTORG문을 포함하여 DFHEIENT 또는 DLI 초기화 호출에 의해 생성되는 리터럴이 기본 레지스터의 범위에 들어가는지 확인해야 합니다.

일반적으로 LTORG문은 길이가 4095바이트를 초과하는 모든 CSECT에 필요합니다.

DFHEIRET 매크로

AMODE(24) 및 AMODE(31) 프로그램의 경우, DFHEIRET 매크로가 EPILOG 프로그램을 호출하여 애플리케이션의 작업 중인 저장영역을 해제합니다.

DFHEIRET 매크로에 대해 다음 매개변수를 지정할 수 있습니다.

RCREG

리턴 코드가 위치하는 레지스터를 지정하십시오. 값을 지정하지 않으면 리턴 코드가 프로그램의 호출자에게 다시 전달되지 않습니다. 기본값은 없습니다.

변환기는 END문 바로 앞에 어떤 매개변수도 지정되지 않은 DFHEIRET 매크로를 삽입합니다(이를 막기 위해 NOEPILOG 변환기 옵션을 지정하지 않는 한). END는 변환기가 인식하도록 대문자여야 합니다. DFHEIRET 매크로가 이 변환기가 삽입하는 DFHEIRET 매크로 앞에서 호출되는 경우, 변환기가 삽입한 매크로는 어떤 코드도 생성하지 않습니다.

AMODE(64) 어셈블러 언어 프로그램

CICS Transaction Server는 64비트 주소 지정 모드에서 실행하는 비Language Environment 어셈블러 언어 프로그램을 지원합니다.

SYSSTATE 매크로

애플리케이션이 AMODE(64)임을 선언하려면 z/OS 어셈블러 SYSSTATE(시스템 상태 식별) 매크로를 사용합니다.

1. 컴파일 JCL에 SYS1.MACLIB 매크로 라이브러리(SYSSTATE 매크로를 포함함)의 SYSLIB문을 포함하십시오.
2. 다음 중 하나를 확인하십시오.
 - 매개변수 AMODE64=YES를 갖는 SYSSTATE 매크로가 CICS 변환기 옵션 명령문 바로 뒤에 위치합니다. 예를 들어, 다음과 같습니다.

```
*ASM  XOPTS(NOPROLOG NOEPILOG)
      SYSSTATE AMODE64=YES
```

- 프로그램이 CICS 변환기 옵션을 지정하지 않는 경우, 매개변수 AMODE64=YES를 갖는 SYSSTATE 매크로가 첫 번째 명령문입니다.

CICS가 이 명령문에 대해 연속을 지원하지 않기 때문에 SYSSTATE문은 하나의 행에 있어야 합니다. SYSSTATE 매크로에 대한 자세한 정보는 z/OS MVS Programming: Assembler Services Reference, Volume 2의 SYSSTATE- 시스템 상태 식별의 내용을 참조하십시오.

애플리케이션이 이 방법으로 AMODE(64)로서 선언되는 경우 다음 CICS 제공 매크로가 AMODE(64) 코드를 생성합니다.

- DFHEIENT
- DFHEISTG
- DFHEIRET
- DFHECALL

SYSSTATE 매크로가 지정되지 않거나 이들 CICS 제공 매크로 중 하나가 처음 코드를 생성할 때 매개변수 AMODE64=YES 없이 지정되는 경우, 이들 매크로는 AMODE(24) 또는 AMODE(31) 코드를 생성합니다.

DFHEIENT 매크로

AMODE(64) 프로그램의 경우, DFHEIENT 매크로가 AMODE(64) PROLOG 프로그램을 호출하는데, 이것은 모든 사용자 변수를 보유하고 CICS가 사용하기 위한 작업 중인 저장영역을 할당합니다. PROLOG 프로그램이 이 저장영역의 CICS 부분을 설정합니다. PROLOG 프로그램이 리턴할 때, 이 코드는 DFHEIENT 매개변수에 의해 지정되는 레지스터를 설정합니다.

상대 주소 지정만 지원되기 때문에 프로그램이 상대 주소 지정 명령어를 사용하도록 지정하려면 DFHEIENT 매크로 매개변수를 지정해야 합니다. 상대 주소 지정의 경우, 프로그램 명령어를 주소 지정하기 위해 어떤 기본 레지스터도 필요없지만 프로그램의 정적 데이터를 주소 지정하기 위해 하나 이상의 레지스터를 사용해야 합니다. STATREG 및 STATIC 매개변수를 사용하여 하나 이상의 정적 레지스터를 설정하십시오.

DFHEIENT 매크로에 대한 다음 매개변수를 지정할 수 있습니다.

CODEREG

상대 주소 지정을 지정하려면 0(기본값)의 값을 지정하십시오.

DATAREG

애플리케이션에 대한 하나 이상의 작업 중인 저장영역을 지정하십시오. 기본값은 레지스터 13이며, 레지스터 13을 첫 번째 데이터 동적 저장영역 레지스터로 사용하는 것이 바람직합니다. 그렇지 않은 경우, DFHECALL 매크로에 의해 생성되는 코드가 레지스터 13을 조작하기 위한 추가 명령어를

추가합니다. DFHECALL 매크로는 레지스터 13이 DFHEISTG가 동적 저장영역에서 정의한 저장 영역을 주소 지정하도록 보장합니다.

EIBREG

실행 인터페이스 블록(EIB)을 주소 지정하기 위해 사용할 레지스터를 지정하십시오. 기본값은 레지스터 11입니다.

STATREG

애플리케이션이 사용할 하나 이상의 정적 레지스터를 지정하십시오. 기본값은 레지스터 3입니다.

STATIC

정적 영역 시작의 어셈블러 레이블을 지정하십시오. 사용자가 값을 지정해야 합니다. 이 매개변수에 대한 기본값이 없습니다.

NOPROLOG 변환기 옵션을 사용하고 상대 주소 지정에 적합한 매개변수를 갖는 DFHEIENT 매크로를 지정하십시오. DFHEIENT 매크로를 지정하지 않는 경우, 변환기는 필수 매개변수 없이 DFHEIENT 매크로를 삽입하고 다음 오류가 발생합니다.

12,DFHEIENT - AMODE 64 - STATIC REQUIRED

다음의 두 예제 DFHEIENT문은 동일한 코드를 생성합니다. 첫 번째 명령문에서 모든 매개변수가(기본값을 지정하여) 코딩됩니다. 두 번째 명령문에서, 기본값을 갖지 않는 매개변수가 코딩됩니다.

```
DFHEIENT CODEREG=0,DATAREG=13,EIBREG=11,STATREG=3,STATIC=STAT
```

```
DFHEIENT STATIC=STAT
```

DFHEIRET 매크로

AMODE(64) 프로그램의 경우, DFHEIRET 매크로는 AMODE(64) EPILOG 프로그램을 호출하여 애플리케이션의 작업 중인 저장영역을 해제합니다.

DFHEIRET 매크로에 대해 다음 매개변수를 지정할 수 있습니다.

RCREG

리턴 코드가 위치하는 레지스터를 지정하십시오. 값을 지정하지 않으면 리턴 코드가 프로그램의 호출자에게 다시 전달되지 않습니다. 기본값은 없습니다.

변환기는 END문 바로 앞에 어떤 매개변수도 지정되지 않은 DFHEIRET 매크로를 삽입합니다(이를 막기 위해 NOEPILOG 변환기 옵션을 지정하지 않는 한). END는 변환기가 인식하도록 대문자여야 합니다. DFHEIRET 매크로가 이 변환기가 삽입하는 DFHEIRET 매크로 앞에서 호출되는 경우, 변환기가 삽입한 매크로는 어떤 코드도 생성하지 않습니다.

DFHEISTG 저장영역에 대한 자세한 정보는 86 페이지의 『동적 저장영역으로의 확장』의 내용을 참조하십시오. DFHECALL 및 DFHEIRET 매크로에 대한 자세한 정보는 의 내용을 참조하십시오.

관련 정보

EXEC 인터페이스 디자인 개요

어셈블러 언어 애플리케이션에 대한 Language Environment 코드화 요구사항

어셈블러 언어 프로그램은 Language Environment에 관하여 준수 또는 비준수로 분류됩니다. 준수는 사용한 어셈블러가 아닌 관찰된 연계 및 레지스터 규칙에 따라 다릅니다.

Language Environment를 준수하는 어셈블러 언어 루틴은 CEEENTRY를 사용하여 코딩된 루틴으로 정의되며 Language Environment 매크로와 연관됩니다.

준수는 HLL 프로그램의 호출을 통해 어셈블러 프로그램의 사용을 제어합니다. 준수 및 비준수 어셈블러 언어 서브루틴 모두 C, C++, COBOL이나 PL/I에서 정적 또는 동적으로 호출할 수 있습니다. 하지만 두 가지 유형에 대한 기타 요구사항과 레지스터 규칙은 차이가 있습니다. 예를 들어, Language Environment 준수 어셈블러 언어 루틴과 제대로 통신하려면 어셈블러 언어 루틴의 입력(실행 중일 때) 및 어셈블러 언어 루틴의 종료에 대한 특정 레지스터 규칙을 준수해야 합니다.

어셈블러 언어를 포함한, 언어 혼합에 대한 규칙은 16 페이지의 『Language Environment에서 언어 혼합』에서 논의됩니다.

64비트 주소 지정 모드는 Language Environment 준수 어셈블러 언어 프로그램에 지원되지 않습니다.

이 섹션에 사용된 용어의 설명이나 보다 자세한 정보는 z/OS Language Environment Programming Guide의 내용을 참조하십시오.

MAIN 프로그램 준수

Language Environment 인터페이스에 맞추려는 새 어셈블러 언어 MAIN 프로그램을 코딩 중인 경우나 어셈블러 언어 루틴이 Language Environment 서비스를 호출하는 경우 다음을 준수하십시오.

- Language Environment에 제공된 매크로를 사용하십시오. 이 매크로 목록은 z/OS Language Environment Programming Guide의 내용을 참조하십시오.
- CEEENTRY 매크로에 MAIN=YES 옵션이 포함되었는지 확인하십시오(MAIN=YES가 기본값).
- *ASM XOPTS(LEASM) 또는 CICS 명령을 포함한 경우 *ASM XOPTS(LEASM NOPROLOG NOEPILOG)로 어셈블러 언어 루틴을 변환하십시오.

서브루틴 준수

Language Environment 인터페이스에 맞추려는 새 어셈블러 언어 서브루틴을 코딩 중인 경우나 어셈블러 언어 루틴이 Language Environment 서비스를 호출하는 경우 다음을 준수하십시오.

- Language Environment에 제공된 매크로를 사용하십시오. 이 매크로 목록은 z/OS Language Environment Programming Guide의 내용을 참조하십시오.
- CEEENTRY 매크로에 MAIN=NO 옵션이 포함되었는지 확인하십시오(MAIN=YES가 기본값).
- CICS 명령을 포함한 경우 *ASM XOPTS(NOPROLOG NOEPILOG)로 어셈블러 언어 루틴을 변환하십시오.
- 루틴이 VS COBOL II에서 정적 호출을 통해 호출되는 경우 CEEENTRY 매크로에 NAB=NO 옵션이 포함되었는지 확인하십시오. (NAB는 저장영역의 사용 가능한 다음 바이트입니다. NAB=NO는 이 필드가 사용 가능하지 않을 수 있음을 의미하므로 CEEENTRY 매크로는 사용 가능한 저장영역을 찾습니다.)

준수 루틴 입력을 위한 레지스터 규칙

Language Environment 준수 어셈블러 언어 서브루틴에 입력 시 다음 레지스터는 NAB=YES가 CEEENTRY 매크로에 지정되어 있을 때 다음 값을 포함해야 합니다.

- R0** 예약됨
- R1** 매개변수 목록의 주소 또는 0
- R12** 공통 앵커 영역(CAA) 주소
- R13** 호출자의 동적 저장영역(DSA)
- R14** 리턴 주소
- R15** 시작점 주소

Language Environment 준수 HLL은 이 레지스터 규칙을 따르는 코드를 생성하며 이를 사용하여 Language Environment 준수 어셈블러 언어 루틴을 쓸 때 제공된 매크로도 동일하게 수행합니다. 어셈블러 언어 루틴 입력 시 CEEENTRY는 호출자가 제공한 DSA에 호출자의 레지스터(R14 - R12)를 저장합니다. 새 DSA를 할당하고 NAB 필드를 새 DSA에 제대로 설정합니다. 새 DSA의 처음 반자는 2진 0으로 설정되고 두 번째 단어의 백체인은 호출자의 DSA를 가리키도록 설정됩니다.

준수 규칙이 실행 중일 때의 레지스터 규칙

R13은 Language Environment 준수 어셈블러 언어 루틴이 실행 중일 때 항상 루틴의 DSA를 가리켜야 합니다.

다른 프로그램을 호출하는 코드의 어느 지점에서 R12는 다음 경우를 제외하고, 공통 앵커 영역(CAA) 주소를 포함해야 합니다.

- COBOL 프로그램을 호출할 때
- Language Environment를 준수하지 않는 어셈블러 언어 루틴을 호출할 때
- CEEENTRY 매크로에 NAB=NO를 지정하는 Language Environment 준수 어셈블러 언어 루틴을 호출할 때

준수 루틴 종료를 위한 레지스터 규칙

Language Environment 준수 어셈블러 언어 루틴에서 종료할 때에는 R0, R1, R14, R15가 정의되지 않습니다. 다른 모든 레지스터에 입력 시 보유한 내용이 있어야 합니다.

CEEENTRY 모듈을 자동으로 AMODE (ANY) 및 RMODE (ANY)로 설정합니다. 기존 어셈블러 언어 루틴을 Language Environment 준수로 변환 중이며 루틴에 24비트 주소 지정 모드를 사용하여 코딩한 데이터 관리 매크로가 포함된 경우 매크로를 31비트 모드를 사용하도록 변경해야 합니다. 프로그램의 모든 모듈을 31비트 주소 지정 모드를 사용하도록 변경할 수 없는 경우 및 어느 모듈도 명시적으로 RMODE(24)를 설정하지 않는 경우에는 링크 편집 프로세스 중 프로그램을 RMODE(24)로 설정해야 합니다.

Language Environment에서 실행하는 비준수 어셈블러 언어 루틴

Language Environment에서 Language Environment 비준수 어셈블러 언어 루틴을 실행할 때에는 다음 규칙을 준수하십시오.

- R13은 실행 루틴의 레지스터 저장 영역 주소를 포함해야 합니다.
- 레지스터 저장 영역의 처음 두 개 바이트는 2진 0이어야 합니다.
- 레지스터 저장 영역 백체인을 올바른 31비트 주소로 설정해야 합니다(24비트 주소이면 최상위 바이트가 0이어야 함).

어셈블러 언어 루틴이 C, C++, COBOL 또는 PL/I 제어 블록에 종속되어 있으면(예를 들어, 제어 블록의 스위치나 플래그를 테스트하는 루틴) 이 제어 블록이 Language Environment에서 변경되지 않았는지 확인하십시오. 자세한 정보는 사용 중인 언어의 컴파일 및 런타임 마이그레이션 안내서를 참조하십시오.

비준수 어셈블러 언어 루틴은 Language Environment 호출 가능 서비스를 사용할 수 없습니다.

어셈블러 언어 프로그램 호출

어셈블러 언어 애플리케이션은 LINK나 XCTL 명령을 사용하여 COBOL, C, C++, PL/I 또는 어셈블러 언어 애플리케이션으로 호출할 수 있습니다.

64비트 주소 지정 모드

명령을 포함한 AMODE(64) 어셈블러 언어 애플리케이션에는 자체 RDO 프로그램 정의가 있을 수 있습니다. 이러한 프로그램은 LINK나 XCTL 명령을 사용하여 COBOL, C, C++, PL/I 또는 어셈블러 언어 애플리케이션으로 호출할 수 있습니다.

24비트 및 31비트 주소 지정 모드

명령을 포함한 AMODE(24) 및 AMODE(31) 어셈블러 언어 애플리케이션에는 자체 RDO 프로그램 정의가 있을 수 있습니다. 이러한 프로그램은 LINK나 XCTL 명령을 사용하여 COBOL, C, C++, PL/I 또는 어셈블러 언어 애플리케이션으로 호출할 수 있습니다. 하지만 명령을 포함한 AMODE(24) 및 AMODE(31) 프로그램은 시스템 표준 호출로 호출하기 때문에 COBOL, C, C++ 또는 PL/I CALL문으로 또는 어셈블러 언어 CALL 매크로로 호출할 수도 있습니다.

RDO 프로그램 정의에 정의된 단일 CICS 애플리케이션은 개별적으로 컴파일 또는 어셈블되었지만 함께 링크되지 않은 개별 CSECT로 이루어질 수 있습니다.

명령을 포함한 어셈블러 언어 애플리케이션은 기타 어셈블러 언어 프로그램과 또는 하나 이상의 고급 언어(COBOL, C, C++ 또는 PL/I)로 쓴 프로그램과 링크할 수 있습니다. 애플리케이션 로드 모듈의 언어 혼합에 대한 자세한 정보는 16 페이지의 『Language Environment에서 언어 혼합』의 내용을 참조하십시오. 여러 다른 주소 지정 모드의 프로그램에 대한 정보는 372 페이지의 『혼합 주소 지정 모드 사용』의 내용을 참조하십시오.

어셈블러 언어 프로그램(개별적으로 링크 편집된)이 명령 레벨 호출을 포함하며 고급 언어 프로그램으로부터 호출된 경우에는 자체 CICS 인터페이스 스텝이 필요합니다. 어셈블러 프로그램이 프로그램을 호출하는 고급 언어 프로그램과 링크 편집되면 어셈블러 프로그램에 스텝이 필요하지 않습니다. 스텝을 제공할 경우 MSGIEW024I 메시지가 발행되지만 무시할 수 있습니다.

명령을 포함한 어셈블러 언어 애플리케이션은 호출 시 항상 EIB 및 COMMAREA 매개변수가 전달되므로, CALL문이나 매크로가 이 두 매개변수를 선택적으로 기타 매개변수 이전에 전달해야 합니다.

예를 들어, PLITEST PLI 파일의 PL/I 프로그램은 ASMTEST ASSEMBLE 파일에 있는 어셈블러 언어 프로그램 ASMPROG를 호출합니다. PL/I 프로그램은 세 개의 매개변수를 어셈블러 언어 프로그램에 전달합니다(EIB, COMMAREA, 메시지 문자열).

```
PLIPROG:PROC OPTIONS(MAIN);
  DCL ASMPROG ENTRY EXTERNAL;
  DCL COMA CHAR(20), MSG CHAR(14) INIT('HELLO FROM PLI');
  CALL ASMPROG(DFHEIBLK,COMA,MSG);
  EXEC CICS RETURN;
END;
```

그림 8. *PLITEST PLI*

어셈블러 언어 프로그램은 PL/I 프로그램에서 전달된 메시지 문자열을 표시하는 EXEC CICS SEND TEXT 명령을 수행합니다.

```
DFHEISTG DSECT
MSG      DS    CL14
MYRESP   DS    F
ASMPROG   CSECT
          L      5,8(1)
          L      5,0(5)
          MVC     MSG,0(5)
          EXEC CICS SEND TEXT FROM(MSG) LENGTH(14) RESP(MYRESP)
          END
```

그림 9. *ASMTEST ASSEMBLE*

CICS에 제공된 JCL 프로시저를 사용하여 다음과 같이 애플리케이션을 컴파일 및 링크할 수 있습니다.

1. DFHEITAL 프로시저를 사용하여 ASMTEST를 어셈블 및 링크하십시오.

```
//ASMPROG EXEC DFHEITAL
//TRN.SYSIN DD *
.... program source ...
/*
//LKED.SYSIN DD *
NAME ASMTEST(R)
/*
```

2. DFHYITPL 프로시저를 사용하여 PLITEST를 컴파일 및 링크하고 DFHEITAL 프로시저로 작성된 ASMTEST 로드 모듈을 포함한 연계 편집기 제어문을 제공하십시오.

```
//PLIPROG EXEC DFHYITPL
//TRN.SYSIN DD *
.... program source ...
/*
//LKED.SYSIN DD *
INCLUDE SYSLIB(ASMTEST)
ENTRY CEESTART
NAME PLITEST(R)
/*
```

참고: 2단계는 DFHEITAL로 작성된 ASMTEST 로드 모듈이 SYSLIB 데이터 세트 연결에 포함된 라이브러리에 저장된다고 가정합니다.

DFHYITPL 프로시저로 작성된 로드 모듈은 DFHEAI 스텝(DFHEITAL로 포함) 및 DFHELII 스텝(DFHYITPL로 포함)을 모두 포함합니다. 연계 편집기 또는 바인더 프로그램은 두 스텝이 모두 DFHEII라는 시작점을 포함하기 때문에 경고 메시지를 발행합니다. 이 메시지는 무시할 수 있습니다.

DFHEAI 스텝이 링크 편집의 출력에 프로그램의 시작에 포함되어야 합니다. 이를 달성하기 위해, DFHEAI에 대한 ORDER 및 INCLUDE문이 JCL의 링크 편집 단계에 있어야 합니다. 어셈블러로 작성된 애플리케이션을 변환, 어셈블 및 링크 편집하기 위해 SDFHPROC 라이브러리의 CICS 제공 어셈블러 프로시저를 사용할 때, 이 프로시저의 COPYLINK 단계가 SDFHMAC(DFHEILIA)를 복사합니다. DFHEILIA는 포함되어야 하는 다음 명령문을 포함합니다.

```
ORDER DFHEAI
INCLUDE SYSLIB(DFHEAI)
```

명령문은 프로시저의 LKED 단계에 있는 어셈블된 애플리케이션 앞에서 연결되는 임시 파일에 넣어집니다.

JCL을 직접 쓰는 경우 모든 언어에 필요한 시작점이 포함되므로 DFHELII 스텝을 포함시켜야 합니다.

또 다른 프로그램으로 호출되는 어셈블러 언어 애플리케이션은 DFHEIENT 매크로로 시작하고 DFHEIRET 매크로로 끝납니다. CICS 변환기가 이를 삽입하므로 위의 예제에서처럼, 프로그램이 EXEC CICS 명령을 포함하고 변환기로 전달되는 경우에는 이 매크로를 코딩할 필요가 없습니다.

동적 저장영역으로의 확장

DFHEISTG라는 DSECT에서 사용자 소스 프로그램에서 사용자 변수를 정의하여 이들 변수를 위한 추가 저장영역을 제공하기 위해 동적 저장영역을 확장할 수 있습니다.

24비트 및 31비트 주소 지정 모드

AMODE(24) 및 AMODE(31) 프로그램의 경우, DFHEISTG DSECT를 사용하여 얻을 수 있는 동적 저장영역의 최대량은 65 264 바이트입니다. DFHEISTG는 예약된 이름입니다. 이 저장영역은 X'00'으로 초기화됩니다. 변환 시에, 변환기는 DFHEISTG DSECT 명령어 바로 뒤에 DFHEISTG 매크로를 삽입합니다. 이 방법으로 DSECT는 매개변수 목록, 명령 레벨 인터페이스 및 모든 사용자 변수에 필요한 동적 저장영역을 설명합니다. 링크 편집 시간에 CEEXOPT 매크로의 STORAGE 옵션을 사용하여 DFHEISTG 저장영역이 x'00'으로 초기화되는지 확인하십시오(예: CEEXOPT STORAGE=(, ,00)). 애플리케이션이 사용자 DFHEISTG 영역에서 정의되는 모든 상수를 전파하거나 초기화하는지 확인하십시오.

그림 10의 예제는 동적 저장영역에서 그런 변수를 사용하는 간단한 어셈블러 애플리케이션을 보여줍니다.

Source program	
DFHEISTG	DSECT
	COPY DFH\$AGA INPUT MAP DSECT
	COPY DFH\$AGB OUTPUT MAP DSECT
MESSAGE	DS CL39
INQUIRY	CSECT
	EXEC CICS RECEIVE MAP('DFH\$AGA')
	MVC NUMBO,KEYI
	MVC MESSAGE,=CL(L'MESSAGE)'THIS IS A MESSAGE'
	EXEC CICS SEND MAP('DFH\$AGB') ERASE
	END
This source program is translated to:	
	DFHEIGBL , INSERTED BY TRANSLATOR
DFHEISTG	DSECT
	DFHEISTG INSERTED BY TRANSLATOR
	COPY DFH\$AGA INPUT MAP DSECT
	COPY DFH\$AGB OUTPUT MAP DSECT
MESSAGE	DS CL39
INQUIRY	CSECT
	DFHEIENT INSERTED BY TRANSLATOR
*	EXEC CICS RECEIVE MAP('DFH\$AGA')
	DFHECALL =X'1802C0000800000000040900000020',
	(CHA7,=CL7'DFH\$AGA*'),(____RF,DFH\$AGAI)
	MVC NUMBO,KEYI
	MVC MESSAGE,=CL(L'MESSAGE)'THIS IS A MESSAGE'
*	EXEC CICS SEND MAP('DFH\$AGB') ERASE
	DFHECALL =X'1804C000080000000004E204000020',
	(CHA7,=CL7'DFH\$AGB*'),(____RF,DFH\$AGBO)
	DFHEIRET INSERTED BY TRANSLATOR
	DFHEISTG INSERTED BY TRANSLATOR
	DFHEIEND INSERTED BY TRANSLATOR
	END

그림 10. 사용자 변수에 대해 변환된 코드

64비트 주소 지정 모드

비Language Environment AMODE(64) 어셈블러 언어 프로그램의 경우, DFHEISTG 매크로는 AMODE(64) DSECT를 생성합니다. DFHEISTG 저장영역은 64비트 저장영역이 아니라 31비트 저장영역(16MB 위, 2GB 아래)으로부터 얻습니다. DFHEISTG DSECT를 사용하여 얻을 수 있는 동적 저장영역의 최대량은 65 264 바이트입니다. 이 저장영역은 X'00'으로 초기화됩니다.

변환 시에, 변환기는 DFHEISTG DSECT 명령어 바로 뒤에 DFHEISTG 매크로를 삽입합니다. 이 방법으로 DSECT는 매개변수 목록, 명령 레벨 인터페이스 및 모든 사용자 변수에 필요한 동적 저장영역을 설명합니다.

CICS는 DFHEISTG 저장영역의 앞쪽 부분을 정의합니다.

DFHEIENT 매크로가 제어를 애플리케이션으로 리턴하기 전에 EIB 포인터인 DFHEIBP와 COMMAREA 포인터인 DFHEICAP가 설정됩니다. 이들 포인터는 24비트 또는 31비트 저장영역에 있는 64비트 포인터입니다.

AMODE(64) 어셈블러 언어 프로그램 개발

CICS Transaction Server는 64비트 주소 지정 모드(AMODE(64))에서 실행하는 비 Language Environment(LE) 어셈블러 언어 프로그램을 지원합니다.

시작하기 전에

71 페이지의 『어셈블러 언어 프로그래밍 제한사항 및 요구사항』의 관련 섹션을 확인하십시오.

이 태스크 정보

다음 프로시저는 AMODE(64) 어셈블러 언어 프로그램 개발에 특히 관련된 정보를 요약합니다.

프로시저

1. z/OS 어셈블러 SYSSTATE 매크로를 사용하여 애플리케이션이 AMODE(64)임을 선언하십시오. 76 페이지의 『EXEC CICS 어셈블러 인터페이스 코딩』의 내용을 참조하십시오.
2. 상대 주소 지정을 위한 적합한 매개변수와 함께 DFHEIENT 매크로를 지정하십시오. 76 페이지의 『EXEC CICS 어셈블러 인터페이스 코딩』의 내용을 참조하십시오.

DFHEISTG 저장영역에 대한 정보는 86 페이지의 『동적 저장영역으로의 확장』의 내용을 참조하십시오.

3. EXEC CICS RETURN을 사용하지 않는 경우, DFHEIRET 매크로를 지정하거나 변환기 삽입 버전을 사용하십시오. 76 페이지의 『EXEC CICS 어셈블러 인터페이스 코딩』의 내용을 참조하십시오.
4. 프로그램을 변환, 어셈블, 링크 편집하십시오. 167 페이지의 『어셈블러 언어 애플리케이션 변환, 어셈블 및 링크 편집』의 내용을 참조하십시오.

DFHECALL 매크로에 대한 정보는 의 내용을 참조하십시오.

스텝 프로그램 링크 편집에 대한 정보는 128 페이지의 『AMODE(64) 애플리케이션에 EXEC 인터페이스 모듈 사용』의 내용을 참조하십시오.

관련 정보



z/OS MVS Programming: Assembler Services Guide

제 3 부 애플리케이션 변환, 컴파일, 설치, 테스트

CICS에서 애플리케이션을 변환, 컴파일 및 설치하는 방법과 애플리케이션 테스트를 위해 제공되는 기능.

제 7 장 변환 및 컴파일

일부 기존 컴파일러(및 어셈블러)는 CICS 명령을 직접 처리할 수 없습니다. 프로그램을 실행 코드로 변환하려면 추가 단계가 필요합니다. 이 단계를 **변환**이라고 하며, 컴파일러(또는 어셈블러)가 이해할 수 있도록 CICS 명령을 나머지 프로그램이 코드화된 언어로 변환하는 것으로 이루어집니다.

대부분의 컴파일러는 CICS 변환기 접근방식을 사용하는데, 여기서 컴파일러는 컴파일 시 CICS와 상호작용하여 CICS 명령을 해석하고 CICS 서비스 루틴에 대한 호출로 자동 변환합니다. 통합 CICS 변환기 접근방식을 사용하는 경우 다수의 변환 태스크는 컴파일 시 수행되며 추가 변환 단계를 완료하지 않아도 됩니다. 변환 단계의 태스크에 대한 자세한 정보는 96 페이지의 『변환 프로세스』의 내용을 참조하십시오.

이 섹션에서는 다음에 대해 설명합니다.

- 『통합 CICS 변환기』
- 96 페이지의 『변환 프로세스』
- 99 페이지의 『CICS 제공 변환기』
- 111 페이지의 『CICS 변환기 사용』
- 113 페이지의 『변환기 옵션 정의』
- 115 페이지의 『COPY문 사용』
- 116 페이지의 『CICS 제공 인터페이스 모듈』
- 117 페이지의 『AMODE(24) 및 AMODE(31) 애플리케이션에 EXEC 인터페이스 모듈 사용』

통합 CICS 변환기

통합 변환기를 사용하면 한 단계로 상위 레벨 소스 코드를 변환 및 컴파일할 수 있습니다. 통합 변환기를 지원하는 컴파일러는 애플리케이션 소스를 스캔하고 관련 위치에서 통합 변환기를 호출합니다. 통합 변환기는 **EXEC CICS** 명령을 주석으로 변환하고 언어에 적절한 CALL문을 생성합니다.

CICS 온라인 프로그램 컴파일 시 통합 변환기 버전이 있는 언어는 다음과 같습니다.

C
C++
COBOL
PL/I

또한 외부 CICS 인터페이스(EXCI) 명령 레벨 API를 사용하는 COBOL, C 및 C++ 일괄처리 프로그램을 컴파일할 때 통합 변환기를 사용할 수 있습니다.

통합 변환기를 사용하면 별도의 변환 단계가 없기 때문에 애플리케이션 개발이 빨라집니다. 하나의 목록만 있기 때문에 애플리케이션 개발도 훨씬 용이합니다. 즉, 원래 소스 문과 CICS 오류 메시지가 컴파일러 목록에 포함됩니다. CICS 제공 개별 변환기는 소스 프로그램의 행 번호를 변경합니다. 이는 변환기가 생성한 호출을 사용하면 애플리케이션 디버깅 시 사용해야 하는 중간 목록이 필요함을 의미합니다.

통합 변환기를 사용하면 포함된 멤버를 더 이상 개별적으로 컴파일할 필요가 없기 때문에 변환 및 컴파일 프로세스의 오류 발생 가능성도 줄어듭니다.

통합 변환기를 지원하는 Language Environment 준수 언어 컴파일러는 애플리케이션 소스를 스캔하고 관련 위치에서 통합 CICS 변환기를 호출합니다.

통합 변환기를 지원하는 언어 컴파일러 릴리스는 고급 언어 지원에 나열되어 있습니다. 어셈블러를 포함하여 다른 컴파일러를 사용하는 경우에는 컴파일하기 전에 프로그램을 변환해야 합니다.

통합 CICS 변환기 사용

언어 컴파일러는 통합 CICS 변환기와 함께 사용할 수 있는 다양한 프로시저를 제공합니다. 이들은 Programming Guides for Enterprise COBOL for z/OS, z/OS XL C/C++ 및 for Enterprise PL/I for z/OS에 문서화되어 있습니다.

이 태스크 정보

사용하는 프로시저는 CICSTS53.CICS.SDFHLOAD를 컴파일 단계를 위한 STEPLIB 접합에 추가해야 하고 링크 편집 단계가 단계의 초기에 인터페이스 모듈 DFHELII를 포함해야 합니다.

PL/I를 위한 통합된 CICS 변환기를 사용하려면 컴파일러 옵션 시스템(CICS)를 지정해야 합니다.

COBOL을 위한 통합된 CICS 변환기를 사용하려면 컴파일러 옵션 CICS, LIB, NODYNAM 및 RENT가 유효해야 합니다. NODYNAM은 통합 변환기에 고유한 제한사항이 아닙니다. DYNAM은 개별적으로 변환되고 컴파일되는 코드에는 지원되지 않습니다. 저장영역은 통합된 CICS 변환기 서비스를 위한 사용자 리전에 남아 있어야 하므로 SIZE(MAX)를 사용하지 마십시오. 대신 대부분의 프로그램에 작동하는 SIZE(4000K) 등과 같은 값을 사용하십시오.

C 및 C++을 위해 통합된 CICS 변환기를 사용하려면 CICS 옵션을 사용하십시오.

DB2 버전 7 이상을 실행 중이고 통합된 변환기와 함께 컴파일러를 사용하여 COBOL 프로그램을 준비 중인 경우에는 컴파일러는 또한 SQL 문 코프로세서를 제공하므로

(DBRM을 생성함), 개별 DB2 프리컴파일러를 사용할 필요가 없습니다. SQL 문 코프로세서 사용에 대한 자세한 정보는 *CICS DB2 안내서* 및 *DB2 for z/OS 애플리케이션 및 SQL 안내서*를 참조하십시오.

CICS 변환기 옵션 지정하기

PL/I, COBOL, XL C 또는 C++ 컴파일러를 사용할 때 CICS 변환기 옵션을 지정할 수 있습니다.

이 태스크 정보

모든 변환기 옵션에 대한 설명은 113 페이지의 『변환기 옵션 정의』의 내용을 참조하십시오.

통합된 CICS 변환기를 사용할 때(예를 들어, 변환기 목록과 연관된 변환기) 많은 변환기 옵션이 적용되지 않습니다. 이러한 옵션이 지정된 경우에는 이들은 무시됩니다. EXCI 옵션은 PL/I에는 지원되지 않지만 EXCI 명령 레벨 API를 사용하는 일괄처리 Cobol, C 및 C++ 프로그램에는 지원됩니다.

다음 변환기 옵션은 통합된 CICS 변환기와 함께 효율적으로 사용될 수 있습니다.

- APOST 또는 QUOTE
- CPSM 또는 NOCPSM
- CICS
- DBCS
- DEBUG 또는 NODEBUG
- DLI
- EDF 또는 NOEDF
- FEPI 또는 NOFEPI
- GRAPHIC
- LENGTH 또는 NOLENGTH
- LINKAGE 또는 NOLINKAGE
- NATLANG
- SP
- SYSEIB

프로시저

- PL/I 컴파일러를 사용할 때 CICS 변환기 옵션을 지정하려면 변환기 옵션이 어퍼스트로피와 안쪽 괄호로 둘러싸인 상태로 컴파일러 옵션, PP(CICS)를 지정하십시오. 예를 들어, 다음과 같습니다.

```
PP(CICS('opt1 opt2 optn ...'))
```

PL/I 컴파일러 옵션 지정에 대한 자세한 정보는 *Enterprise PL/I for z/OS Programming Guide*의 내용을 참조하십시오.

- COBOL 컴파일러를 사용할 때 CICS 변환기 옵션을 지정하려면 변환기 옵션이 어퍼스트로피와 안쪽 괄호로 둘러싸인 상태로 컴파일러 옵션, CICS를 지정하십시오. 예를 들어, 다음과 같습니다.

```
CICS('opt1 opt2 optn ...')
```

참고: XOPTS 변환기 옵션은 CICS 컴파일러 옵션으로 변경되어야 합니다. XOPTS는 통합된 CICS 변환기를 사용할 때 승인되지 않습니다.

COBOL 컴파일러 옵션 지정에 대한 자세한 정보는 *Enterprise COBOL for z/OS: Programming Guide*의 내용을 참조하십시오.

- XL C 및 C++ 컴파일러를 사용할 때 CICS 변환기 옵션을 지정하려면 변환기 옵션이 어퍼스트로피와 안쪽 괄호로 둘러싸인 상태로 컴파일러 옵션, CICS를 지정하십시오. 예를 들어, 다음과 같습니다.

```
CICS(opt1,opt2,optn ...)
```

그렇지 않으면, XOPTS 또는 CICS 키워드 위의 프로그램 소스에서 #pragma 문에 변환기 옵션을 지정할 수 있습니다.

C와 C++ 컴파일러 옵션을 지정하는 데 대한 자세한 정보는 *z/OS XL C/C++ 사용자 안내서*의 내용을 참조하십시오.

변환 프로세스

통합 변환기가 없는 컴파일러의 경우, CICS는 EXEC CICS, EXEC CPSM 및 EXEC DLI문을 처리하기 위해 쓸 수 있는 각 언어에 변환기 프로그램을 제공합니다. 통합 변환기가 있는 컴파일러의 경우, 컴파일러가 CICS 변환기를 직접 호출하여 이러한 명령 변환을 처리합니다.

언어 변환기는 소스 프로그램을 읽고 새 프로그램을 작성합니다. 대부분의 일반적인 언어 명령문은 변경되지 않지만 CICS 명령은 코드화하고 있는 언어에 필요한 양식의 CALL문으로 변환됩니다. 호출은 CICS 제공 『EXEC』 인터페이스 모듈(나중에 로드 모듈에 링크 편집됨)을 호출하며 이는 실행 시 요청된 서비스를 다시 호출합니다.

세 개의 단계(변환, 컴파일(어셈블리), 링크 편집)가 있습니다. 97 페이지의 그림 11은 이 세 단계를 표시합니다.

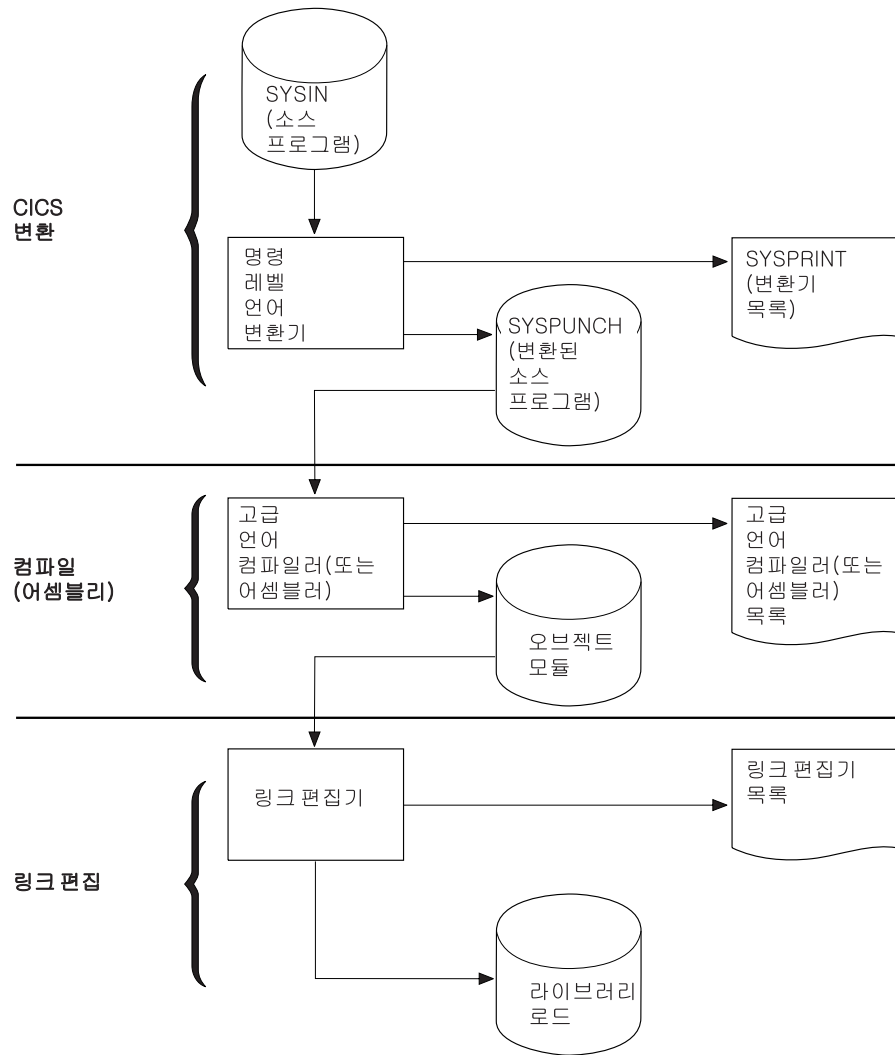


그림 11. 애플리케이션 준비

모든 언어의 경우 변환기는 하나의 입력 파일과 두 개의 출력 파일을 사용합니다.

SYSIN

(변환기 입력) 소스 프로그램을 포함하는 파일입니다.

SYSIN 파일이 블록화된 고정 데이터 세트로 정의된 경우, 데이터 세트가 소유할 수 있는 최대 레코드 길이는 80바이트입니다. 레코드 길이가 80바이트를 넘는 블록화된 고정 데이터 세트를 변환기로 전달하면 변환기 실행이 종료됩니다. SYSIN 파일이 블록화된 가변 데이터 세트로 정의된 경우, 데이터 세트가 소유할 수 있는 최대 레코드 길이는 100바이트입니다. 레코드 길이가 100바이트를 넘는 블록화된 가변 데이터 세트를 변환기로 전달하면 변환기가 중지되고 오류가 표시됩니다.

SYSPUNCH

(변환된 소스) 컴파일(어셈블) 단계의 입력이 되는 변환된 버전의 소스 코드입니다. 이 파일에서 소스는 다음과 같이 변경되었습니다.

- 실행 인터페이스 블록(EIB) 구조가 삽입되었습니다.
- EXEC CICS, EXEC CPSM 및 EXEC DLI 명령이 함수 호출문으로 변경되었습니다.
- CICS DFHRESP, EYUVALUE 및 DFHVALUE 기본 제공 함수가 처리되었습니다.
- 프로그램에 EXEC DLI문이 포함된 경우 데이터 교환 블록(DIB) 구조 및 초기화 호출이 삽입되었습니다.

변환된 CICS 명령은 소스에 여전히 나타나지만 주석으로만 나타납니다. 일반적으로 비CICS문은 변경되지 않습니다. 변환기의 출력은 항상 80바이트 고정 레코드 길이 데이터 세트로 이동합니다.

SYSPRINT

(변환기 목록) 변환기가 생성한 메시지 수, 메시지와 연관된 심각도가 가장 높은 코드를 표시합니다. NOOPTIONS 옵션을 사용하여 비활성화되지 않는 한 프로그램 변환에서 사용된 옵션도 나타냅니다.

COBOL, C, C++ 및 PL/I 프로그램의 경우, SYSPRINT는 메시지 자체도 포함합니다. 또한 변환기의 SOURCE 옵션을 지정하는 경우에는 SYSPRINT에 어노테이션이 있는 소스 목록도 표시됩니다. 이 목록은 후속 컴파일 목록과 거의 동일한 정보를 포함하므로 상당수의 설치에서는 이를 생략하도록 선택합니다(NOSOURCE 옵션). 그러나 이 목록에서 필요한 한 항목(컴파일 목록에는 없음)은 행 번호입니다(변환기가 행 번호를 지정하는 경우). 행 번호는 실행 진단 기능(EDF)을 사용하여 디버그할 때 코드의 위치를 표시하는 하나의 방법입니다. VBREF 옵션을 지정하면 행 번호로 교차 참조된 프로그램의 모든 명령 목록도 표시되며 이를 EDF 목적으로 소스 목록의 대안으로 사용할 수 있습니다.

어셈블러 언어 프로그램의 경우, SYSPRINT는 변환기 옵션, 메시지 수 및 최대 심각도 코드만 포함합니다. 메시지 자체는 관련 명령문 뒤에 주석으로 SYSPUNCH 파일에 삽입됩니다. 따라서 어셈블러는 이를 어셈블러 목록으로 복사하며, 여기서 확인할 수 있습니다. 어셈블러가 변환기에 발생한 문제점의 결과로 생성한 MNOTE도 볼 수 있습니다.

참고: EXEC SQL을 사용하는 경우, SQL문을 변환하고 바인드하려면 추가 단계가 필요합니다. 이러한 추가 단계에 대한 정보는 *애플리케이션 프로그래밍 및 SQL 안내서*를 참조하십시오.

CICS는 지원하는 각 언어에 대해 이러한 단계를 순서대로 실행하기 위한 프로시저를 제공합니다. 164 페이지의 『CICS 제공 프로시저를 사용하여 애플리케이션 설치』에서는 이러한 프로시저 사용 방법과 프로시저가 수행하는 작업을 설명합니다.

다수의 옵션을 지정하여 변환 프로세스를 제어할 수 있습니다. 예를 들어, 프로그램이 EXEC DLI 호출을 사용하는 경우 변환기에 알려야 합니다.

변환기는 오류 메시지를 생성할 수 있으며, 컴파일러와 링크 편집기가 생성한 메시지를 확인하는 것만큼 이러한 메시지를 확인하는 것이 중요합니다. 이러한 메시지의 위치는 『CICS 제공 변환기』의 내용을 참조하십시오.

EXEC 명령은 CICS 인터페이스 모듈을 호출하는 CALL문으로 변환됩니다. 이러한 모듈은 링크 편집 단계에서 오브젝트 모듈에 통합되며, 링크 편집 출력 목록에 표시됩니다. 116 페이지의 『CICS 제공 인터페이스 모듈』에서 이러한 모듈에 대해 자세히 알 수 있습니다.

관련 개념:

111 페이지의 『CICS 변환기 사용』

언어 변환기는 소스 프로그램을 읽고 새 프로그램을 작성합니다. 대부분의 일반적인 언어 명령문은 변경되지 않지만 CICS 명령은 코드화하고 있는 언어에 필요한 양식의 CALL문으로 변환됩니다.

관련 참조:

『CICS 제공 변환기』

CICS 제공 변환기는 CICSTS53.CICS.SDFHLOAD 라이브러리에 설치됩니다.

CICS 제공 변환기

CICS 제공 변환기는 CICSTS53.CICS.SDFHLOAD 라이브러리에 설치됩니다.

다음 표는 이 라이브러리의 변환기를 나열합니다.

표 5. 언어별 CICS 제공 변환기

언어	변환기
어셈블러	DFHEAPI\$
C	DFHEDPI\$
COBOL	DFHECP1\$
PL/I	DFHEPP1\$

관련 개념:

96 페이지의 『변환 프로세스』

통합 변환기가 없는 컴파일러의 경우, CICS는 EXEC CICS, EXEC CPSM 및 EXEC DLI문을 처리하기 위해 쓸 수 있는 각 언어에 변환기 프로그램을 제공합니다. 통합 변환기가 있는 컴파일러의 경우, 컴파일러가 CICS 변환기를 직접 호출하여 이러한 명령 변환을 처리합니다.

개별 변환기 동적 호출

ATTACH, CALL, LINK 또는 XCTL 매크로를 사용하는 일괄처리 어셈블러 언어 프로그램에서, 또는 CALL을 사용하는 C, PL/I 또는 COBOL 프로그램에서 명령 레벨 언어 변환기를 동적으로 호출할 수 있습니다.

이 태스크 정보

ATTACH, LINK 또는 XCTL을 사용하는 경우 해당 로드 모듈 DFHEXPI\$를 사용하십시오(여기서 어셈블러 언어의 경우 x=A, COBOL의 경우 x=C, C의 경우 x=D는 또는 PL/I의 경우 x=P임).

CALL을 사용하는 경우 PREPROC를 변환기 호출 시작점 이름으로 사용하십시오.

모든 경우에서 변환기에 다음 주소 매개변수를 전달하십시오.

- 변환기 옵션 목록의 주소
- 변환기가 사용하는 DD 이름 목록의 주소(이는 선택사항임)

이러한 주소는 전자 경계에 맞춘 인접 전자 형식이어야 합니다. 레지스터 1은 목록의 첫 번째 주소를 가리켜야 하며 마지막 주소의 최상위 비트를 1로 설정하여 목록의 끝을 나타내야 합니다. 이는 1주소 또는 2주소 둘 다에 해당합니다.

변환기 옵션

명령 레벨 언어 변환기에는 COBOL, C, C++, PL/I 및 어셈블러 프로그램을 변환하기 위해 선택할 수 있는 옵션 목록이 있습니다. 각 변환기 옵션을 설명하고 사용할 수 있는 언어를 나열합니다.

APOST

(COBOL 전용)

APOST는 리터럴이 어포스트로피 또는 따옴표를 사용하여 기술됨을 나타냅니다. QUOTE는 큰따옴표를 나타내는 대체 옵션입니다. 변환기 단계와 다음 컴파일 단계에 동일한 값을 지정해야 합니다.

제공된 COBOL 카피북은 이 옵션을 사용하여 생성되며 작은따옴표를 갖습니다. 애플리케이션에서 제공된 카피북을 사용하여 CICS 구성요소와 상호작용하는 경우, QUOTE 옵션이 아닌 APOST 옵션을 사용하십시오.

CBLCARD

(COBOL 전용) 약어: CBL

CBLCARD는 변환기가 CBL문을 생성하도록 지정합니다. CBLCARD는 기본값이며, 대체 옵션은 NOCBLCARD입니다.

CICS

CICS는 변환기가 EXEC CICS 명령을 처리하도록 지정합니다. 변환기의 기본 스펙입니다. CICS는 변환기 옵션 지정을 위한 XOPTS 키워드의 기존 이름이기도 하

며, 이는 XOPTS 목록에 포함시키거나 XOPTS 대신 사용하여 목록 이름을 지정함으로써 명시적으로 CICS 옵션을 지정할 수 있음을 의미합니다. CICS 명령이 없음을 나타내는 유일한 방법은 CICS 옵션 없이 XOPTS 키워드를 사용하는 것입니다. EXEC DLI 명령을 사용하여 일괄처리 DL/I 프로그램에서 이렇게 해야 합니다. 예를 들어, 어셈블러 언어로 작성된 일괄처리 DL/I 프로그램을 변환하려면 다음을 지정하십시오.

*ASM XOPTS(DLI)

EXEC API 명령이 포함된 COBOL로 작성된 일괄처리 프로그램을 변환하려면 다음을 지정하십시오.

CBL XOPTS(EXCI)

COBOL2

(COBOL 전용) 약어: CO2

COBOL2는 변환기가 변환된 EXEC문에서 사용할 임시 변수를 생성하도록 지정합니다. 다른 모든 면에서 프로그램은 COBOL3 옵션을 사용할 경우와 동일한 방법으로 변환됩니다. COBOL2와 COBOL3은 상호 배타적입니다. COBOL2는 COBOL의 경우 기본값입니다.

참고: COBOL2와 COBOL3을 여러 방법으로 지정하면 두 옵션이 지정된 위치에 관계 없이 항상 COBOL3 옵션이 사용됩니다. COBOL2와 COBOL3을 둘 다 지정하면 변환기가 경고 메시지를 발행합니다.

COBOL3

(COBOL 전용) 약어: CO3

COBOL3은 변환기가 Language Environment 준수 프로그램을 변환하도록 지정합니다. COBOL3 및 COBOL2 옵션은 상호 배타적입니다. 사용 가능한 Language Environment 준수 컴파일러에 대한 정보는 11 페이지의 제 2 장 『프로그래밍 언어 및 Language Environment』의 내용을 참조하십시오.

CPP

(C++ 전용)

CPP는 변환기가 지원되는 C++ 컴파일러가 컴파일할 C++ 프로그램을 변환하도록 지정합니다.

CPSM

CPSM은 변환기가 EXEC CPSM 명령을 처리하도록 지정합니다. 대체 옵션은 기본값인 NOCPSM입니다.

DBCS

(COBOL 전용)

DBCS는 소스 프로그램이 2바이트 문자를 포함함을 지정합니다. DBCS 옵션은 변환기가 16진 코드 X'0E' 및 X'0F'가 프로그램에 나타날 때마다 각각 시프트 아웃(SO) 및 시프트 인(SI) 코드로 간주하게 합니다.

DBCS를 사용하여 COBOL로 프로그래밍하는 방법에 대한 자세한 정보는 Enterprise COBOL for z/OS Language Reference에서 DBCS 문자열에 관한 절을 참조하십시오.

DEBUG

(COBOL, C, C++ 및 PL/I 전용)

DEBUG는 실행 진단 기능(EDF)이 사용할 수 있도록 CICS에 행 번호를 전달하는 코드를 생성하도록 변환기에 지시합니다. DEBUG는 기본값입니다. NODEBUG는 대체 옵션입니다.

DLI

DLI는 변환기가 EXEC DLI 명령을 처리하도록 지정합니다. XOPTS 옵션과 함께 지정해야 합니다(즉, XOPTS(DLI)).

EDF

EDF는 실행 진단 기능이 프로그램에 적용되도록 지정합니다. EDF는 기본값이며, 대체 옵션은 NOEDF입니다.

EPILOG

(어셈블러 언어 전용)

EPILOG는 변환기가 변환되는 프로그램 끝에 DFHEIRET 매크로를 삽입하도록 지정합니다. DFHEIRET는 실행 프로그램에서 이를 호출한 프로그램으로 제어를 리턴합니다. RETURN 명령의 옵션을 사용하려면 RETURN을 사용하고 NOEPILOG를 지정하십시오.

EPILOG는 기본값입니다. 대체 옵션 NOEPILOG는 변환기가 DFHEIRET 매크로를 삽입하지 못하게 합니다(DFHEIRET 매크로에 대한 프로그래밍 정보는 CICS 애플리케이션 프로그래밍 참조서 참조).

EXCI

EXCI는 변환기가 외부 CICS 인터페이스(EXCI)용 EXEC API 명령을 처리하도록 지정합니다. EXCI API 명령은 일괄처리 프로그램에서만 사용해야 하므로 EXCI 변환기 옵션은 CICS 변환기 옵션 또는 CICS 옵션을 나타내는 변환기 옵션과 상호 배타적입니다. CICS와 EXCI가 둘 다 지정되거나 CICS를 나타내는 변환기 옵션과 EXCI가 지정된 경우 오류 메시지가 생성됩니다.

EXCI 옵션은 DLI 옵션과도 상호 배타적입니다. EXEC DLI 명령을 사용하여 일괄처리 프로그램에서 외부 CICS 인터페이스용 EXEC API 명령을 코드화할 수 없습니다. EXCI 및 DLI 변환기 명령을 둘 다 지정하면 오류 메시지가 생성됩니다.

EXCI 변환기 옵션은 XOPTS를 통해 지정됩니다(즉, XOPTS(EXCI)).

FEPI

FEPI는 CICS FEPI(Front End Programming Interface)의 FEPI API 명령에 대한 액세스를 허용합니다. 대체 옵션은 NOFEPI입니다. FEPI에 대한 자세한 정보는 애플리케이션 개발에서 FEPI 애플리케이션 개발의 내용을 참조하십시오.

FLAG(I, W, E 또는 S)

(COBOL, C, C++ 및 PL/I 전용) 약어: F

FLAG는 변환에서 메시지를 나열해야 하는 최소 오류 심각도를 지정합니다.

I 모든 메시지.

W (기본값) 정보 메시지를 제외한 모든 메시지.

E 경고 및 정보 메시지를 제외한 모든 메시지.

S 심각 및 복구 불가능한 오류 메시지만.

GDS

(C, C++ 및 어셈블러 언어 전용)

GDS는 변환기가 CICS GDS(Generalized Data Stream) 명령을 처리하도록 지정합니다. 이러한 명령에 대한 프로그래밍 정보는 참조 -> 애플리케이션 개발에서 CICS API 명령의 내용을 참조하십시오.

GRAPHIC

(PL/I 전용)

GRAPHIC은 소스 프로그램이 2바이트 문자를 포함함을 지정합니다. GRAPHIC 옵션은 변환기가 16진 코드 X'0E' 및 X'0F'가 프로그램에 나타날 때마다 각각 시프트 아웃(SO) 및 시프트 인(SI) 코드로 간주하게 합니다.

또한 변환기가 16진 양식으로 시프트 아웃 및 시프트 인 값을 포함하는 매개변수 목록을 생성하지 못하게 합니다. 이러한 값이 정상적으로 나타날 때마다 변환기는 이를 2진 양식으로 표현하므로 컴파일러가 수신하는 데이터 스트림에는 의도하지 않은 DBCS 구분 기호가 없습니다.

사용 중인 컴파일러가 DBCS를 지원하는 경우, 2바이트 문자를 사용하지 않더라도 의도하지 않은 시프트 아웃 및 시프트 인 코드를 방지해야 합니다. 변환기에 GRAPHIC 옵션을 지정하여 의도하지 않은 시프트 아웃 및 시프트 인 코드를 방지하여 변환기가 이를 작성하지 않게 하거나 컴파일 단계에 NOGRAPHIC을 지정하여 컴파일러가 이를 DBCS 구분 기호로 해석하지 않게 하십시오.

DBCS를 사용하여 PL/I로 프로그래밍하는 방법에 대한 자세한 정보는 관련 언어 참조 매뉴얼을 참조하십시오.

LEASM

(어셈블러 언어 전용)

LEASM은 Language Environment 준수 어셈블러 MAIN 프로그램에 대한 코드를 생성하도록 변환기에 지시합니다.

LEASM 옵션이 지정된 경우, DFHEISTG, DFHEIENT, DFHEIRET 및 DFHEIEND 매크로가 다르게 확장되어 CICS 환경에서 어셈블러 서브루틴에 사용되는 매크로 확장 양식 대신 Language Environment 준수 어셈블러 MAIN 프로그램을 작성합니다. LEASM 옵션은 프로그램 소스를 변경하지 않고 Language Environment 지원을 이용하기 위해 NOPROLOG 및 NOEPILOG를 사용하고 자체 DFHEIENT 및 기타 매크로를 코드화한 고객 프로그램을 허용합니다. 예를 들어, 변환기는 다중 코드 베이스 레지스터를 지원하지 않기 때문에 두 개 이상의 코드 베이스 레지스터가 필요한 모든 프로그램은 이 범주에 속합니다.

LEASM 옵션을 사용하여 변환된 어셈블러 프로그램 예제는 120 페이지의 『LEASM 이 있는 예제 어셈블러 언어 프로그램』의 내용을 참조하십시오.

LENGTH

(COBOL, 어셈블러 언어 및 PL/I 전용)

LENGTH는 애플리케이션의 CICS 명령에서 LENGTH 옵션이 생략된 경우 기본 길이를 생성하도록 변환기에 지시합니다. 대체 옵션은 NOLENGTH입니다.

LINECOUNT(n)

약어: LC

LINECOUNT는 표제와 빈 행을 포함하여 변환기 목록의 각 페이지에 포함될 행 수를 지정합니다. n 값은 1 - 222 범위의 정수여야 합니다. n 이 5보다 작으면 목록의 표제와 한 행만 각 페이지에 포함됩니다. 기본값은 60입니다.

LINKAGE

(COBOL 전용) 약어: LIN

LINKAGE는 기존 규칙에 따라 최상위 레벨 프로그램의 LINKAGE SECTION 및 PROCEDURE DIVISION문을 수정하도록 변환기에 요청합니다.

이는 변환기가 PROCEDURE DIVISION에 USING DFHEIBLK DFHCOMMAREA문(아직 없는 경우)을 삽입함을 의미하며 LINKAGE SECTION(필요하면 작성)에 DFHEIBLK 및 DFHCOMMAREA에 대한 정의가 포함되게 합니다.

LINKAGE는 기본값이며, 대체 옵션은 NOLINKAGE입니다.

MARGINS(m,n[,c])

(C, C++ 및 PL/I 전용) 약어: MAR

MARGINS는 언어 또는 CICS 명령문을 포함하는 입력의 각 행 또는 레코드의 열을 지정합니다. 변환기는 소스 목록에 포함시키더라도 이러한 한계를 벗어난 데이터는 처리하지 않습니다.

MARGINS 옵션은 SOURCE 옵션이 지정된 경우 생성된 목록을 형식화하기 위해 미국 국가 표준 프린터 제어 문자의 위치도 지정합니다. 그렇지 않으면 사이에 빈 행 없이 입력 레코드가 나열됩니다. 여백 매개변수는 다음과 같습니다.

m 왼쪽 여백의 열 번호입니다.

n 오른쪽 여백의 열 번호입니다. m보다 커야 합니다.

참고: C 또는 C++ 컴파일러 옵션으로 사용되는 경우 MARGIN 옵션의 두 번째 인수에 별표(*)를 사용할 수 있습니다. 그러나 변환기의 경우, 1 - 100 사이의 숫자 값(두 값 포함)을 지정해야 합니다. 입력 데이터 세트에 고정 길이 레코드가 있으면 오른쪽 여백에 허용 가능한 최대값은 80입니다. 입력 데이터 세트에 고정 길이 레코드가 있으면 허용 가능한 최대값은 100입니다.

c 미국 국가 표준 프린터 제어 문자의 열 번호입니다. m과 n에 대해 지정한 값을 벗어나야 합니다. c에 0 값을 지정하면 프린터 제어 문자가 없음을 의미합니다. c가 0이 아니면 다음 프린터 제어 문자만 소스에 나타날 수 있습니다.

(공백) 인쇄 전 한 행 건너뛰기.

0 인쇄 전 두 행 건너뛰기.

- 인쇄 전 세 행 건너뛰기.

+ 인쇄 전 건너뛰기 없음.

1 새 페이지.

C 및 C++용 기본값은 고정 길이 레코드의 경우 MARGINS(1,72,0)이고 가변 길이 레코드의 경우 레코드 길이와 같습니다(1,레코드 길이,0). PL/I용 기본값은 고정 길이 레코드의 경우 MARGINS(2,72,0)이고 가변 길이 레코드의 경우 MARGINS(10,100,0)입니다.

NATLANG(EN 또는 KA)

NATLANG은 변환기 메시지 출력에 사용할 언어를 지정합니다.

CS 중국어.

EN (기본값) 영어.

KA 간지.

(이 옵션을 NATLANG API 옵션과 함께 혼동하지 않도록 주의하십시오.)

NOCBLCARD

(COBOL 전용)

NOCBLCARD는 변환기가 CBL문을 생성하지 않도록 지정합니다. CICS가 필요로 하는 컴파일러 옵션은 DFHYITVL 프로시저를 통해 지정됩니다. RENT, NODYNAM 및 LIB를 지정하십시오.

NOCPSM

NOCPSM은 변환기가 EXEC CPSM 명령을 처리하지 않도록 지정합니다. NOCPSM은 기본값이며, 대체 옵션은 CPSM입니다.

NODEBUG

(COBOL, C, C++ 및 PL/I 전용)

NODEBUG는 실행 진단 기능(EDF)이 사용할 수 있도록 CICS에 행 번호를 전달하는 코드를 생성하지 않도록 변환기에 지시합니다.

NOEDF

NOEDF는 실행 진단 기능이 프로그램에 적용되지 않도록 지정합니다. NOEDF 옵션 지정 시 성능상의 이점은 없지만 이 옵션은 잘 디버그된 서브프로그램의 명령이 EDF 표시장치에 나타나지 않게 하는 데 유용합니다.

NOEPILOG

(어셈블러 언어 전용)

NOEPILOG는 변환되는 프로그램 끝에 DFHEIRET 매크로를 삽입하지 않도록 변환기에 지시합니다. DFHEIRET는 실행 프로그램에서 이를 호출한 프로그램으로 제어를 리턴합니다. EXEC CICS RETURN 명령의 옵션을 사용하려면 EXEC CICS RETURN을 사용하고 NOEPILOG를 지정하십시오. NOEPILOG는 변환기가 DFHEIRET 매크로를 삽입하지 못하게 합니다. 대체 옵션은 기본값인 EPILOG입니다.

NOFEPI

NOFEPI는 CICS FEPI(Front End Programming Interface)의 FEPI API 명령에 대한 액세스를 허용하지 않습니다. NOFEPI는 기본값이며, 대체 옵션은 FEPI입니다.

NOLength

(COBOL, 어셈블러 언어 및 PL/I 전용)

NOLength는 애플리케이션의 CICS 명령에서 LENGTH 옵션이 생략된 경우 기본 길이를 생성하지 않도록 변환기에 지시합니다. 기본값은 LENGTH입니다.

NOLINKAGE

(COBOL 전용)

NOLINKAGE는 누락된 DFHEIBLK 및 DFHCOMMAREA문을 제공하거나 LINKAGE 섹션의 EIB 구조 정의를 삽입하기 위해 LINKAGE SECTION 및 PROCEDURE DIVISION문을 수정하지 않도록 변환기에 요청합니다.

NOLINKAGE 옵션은 COBOL 카피북을 제공하여 COMMAREA를 정의하고 EXEC CICS ADDRESS 명령을 사용할 수 있음을 의미합니다.

LINKAGE는 기본값입니다.

NONUM

(COBOL 전용)

NONUM은 프로그램 각 행의 1 - 6열에 나타나는 행 번호를 진단 메시지 및 교차 참조 목록의 행 번호로 사용하지 말고 자체 행 번호를 생성하도록 변환기에 지시합니다. NONUM은 기본값이며, 대체 옵션은 NUM입니다.

NOOPSEQUENCE

(C, C++ 및 PL/I 전용) 약어: NOS

NOOPSEQUENCE는 변환기 출력 레코드에 순서 필드 위치를 지정합니다. C 및 C++ 용 기본값은 고정 길이 레코드의 경우 OPSEQUENCE(73,80)이고 가변 길이 레코드의 경우 NOOPSEQUENCE입니다. PL/I의 경우, 두 유형의 레코드 모두 기본값은 OPSEQUENCE(73,80)입니다.

NOOPTIONS

약어: NOP

NOOPTIONS는 이 변환 중에 사용된 옵션 목록을 출력 목록에 포함하지 않도록 변환기에 지시합니다.

NOPROLOG

(어셈블러 언어 전용)

NOPROLOG는 어셈블되는 프로그램에 DFHEISTG, DFHEIEND 및 DFHEIENT 매크로를 삽입하지 않도록 변환기에 지시합니다. 이러한 매크로는 로컬 프로그램 저장영역을 정의하며 프로그램 입력 시 실행됩니다.

NOSEQ

(COBOL 전용)

NOSEQ는 1 - 6열에서 소스문의 순서 필드를 확인하지 않도록 변환기에 지시합니다. 대체 옵션은 기본값인 SEQ입니다. SEQ가 지정되고 명령문이 순서에 없으면 플래그가 지정됩니다.

NOSEQUENCE

(C, C++ 및 PL/I 전용) 약어: NSEQ

NOSEQUENCE는 변환기 입력의 명령문에 순서 번호가 매겨지지 않고 변환기가 자체 행 번호를 지정해야 함을 지정합니다.

고정 길이 레코드의 경우 기본값은 SEQUENCE(73,80)입니다. C 및 C++에서 가변 길이 레코드의 경우 기본값은 NOSEQUENCE이고 PL/I에서 가변 길이 레코드의 경우 기본값은 SEQUENCE(1,8)입니다.

NOSOURCE

NOSOURCE는 변환된 소스 프로그램 목록을 변환기 목록에 포함하지 않도록 변환기에 지시합니다.

NOSPIE

NOSPIE는 변환기가 복구 불가능한 오류를 트랩하지 않게 합니다. 대신, 덤프가 생성됩니다. IBM 지원 센터에서 그렇게 하도록 요청하는 경우에만 NOSPIE를 사용하십시오.

NOVBREF

(COBOL, C, C++ 및 PL/I 전용)

NOVBREF는 변환기 목록에 행 번호가 있는 명령의 교차 참조를 포함하지 않도록 변환기에 지시합니다(NOVBREF를 NOXREF라고 했으며, 호환성을 위해 NOXREF가 여전히 허용됨). NOVBREF는 기본값이며, 대체 옵션은 VBREF입니다.

NUM

(COBOL 전용)

NUM은 프로그램 각 행의 1 - 6열에 나타나는 행 번호를 진단 메시지 및 교차 참조 목록의 행 번호로 사용하도록 변환기에 지시합니다. 대체 옵션은 기본값인 NONUM입니다.

OPMARGINS(m,n[,c])

(C, C++ 및 PL/I 전용) 약어: OM

OPMARGINS는 변환기 출력 여백 즉, 다음 컴파일러에 대한 입력의 여백을 지정합니다. 일반적으로 이러한 여백은 변환기의 입력 여백과 같습니다. 입력 여백의 정의와 m , n , c 의 의미는 MARGINS를 참조하십시오. C 및 C++의 경우 기본값은 OPMARGINS(1,72,0)이고 PL/I의 경우 기본값은 OPMARGINS(2,72,0)입니다.

OPMARGINS 옵션에 허용 가능한 n 최대값은 80입니다. 변환기의 출력은 항상 고정 길이 레코드 형식입니다.

OPMARGINS 옵션이 변환기의 출력을 일정 형식으로 설정하는 데 사용되는 경우, 사용 중인 컴파일러의 입력 여백을 변경해야 합니다. OPMARGINS 값이 기본값으로 허용되는 경우에는 사용 중인 컴파일러의 입력 여백을 변경하지 않아도 됩니다.

OPSEQUENCE(m,n)

(C, C++ 및 PL/I 전용) 약어: OS

OPSEQUENCE는 변환기 출력 레코드에 순서 필드 위치를 지정합니다. m 및 n 의 의미는 SEQUENCE를 참조하십시오. C 및 C++용 기본값은 고정 길이 레코드의 경우 OPSEQUENCE(73,80)이고 가변 길이 레코드의 경우 NOOPSEQUENCE입니다. PL/I의 경우, 두 유형의 레코드 모두 기본값은 OPSEQUENCE(73,80)입니다.

OPTIONS

약어: OP

OPTIONS는 이 변환 중에 사용된 옵션 목록을 출력 목록에 포함하도록 변환기에 지시합니다.

PROLOG

(어셈블러 언어 전용)

PROLOG는 어셈블되는 프로그램에 DFHEISTG, DFHEIEND 및 DFHEIENT 매크로를 삽입하도록 변환기에 지시합니다. 이러한 매크로는 로컬 프로그램 저장영역을 정의하며 프로그램 입력 시 실행됩니다.

QUOTE

(COBOL 전용) 약어: Q

QUOTE는 리터럴이 큰따옴표(”)를 사용하여 기술됨을 나타냅니다. 변환기 단계와 다음 컴파일러 단계에 동일한 값을 지정해야 합니다.

제공된 COBOL 카피북은 작은따옴표(APOST)를 사용하여 생성됩니다. 애플리케이션에서 CICS 제공 카피북을 사용하여 CICS 구성요소와 상호작용하는 경우, QUOTE 옵션이 아닌 APOST 옵션이 적용되는지 확인하십시오.

SEQ

(COBOL 전용)

SEQ는 1 - 6열에서 소스문의 순서 필드를 확인하도록 변환기에 지시합니다. SEQ는 기본값이며, 대체 옵션은 NOSEQ입니다. 명령문이 순서에 없으면 플래그가 지정됩니다.

SEQUENCE(m,n)

(C, C++ 및 PL/I 전용) 약어: SEQ

SEQUENCE는 변환기 입력의 명령문에 순서 번호가 매겨지도록 지정하고 순서 필드를 포함하는 각 행 또는 레코드의 열을 지정합니다. 변환기는 이 번호를 오류 메시지 및 교차 참조 목록의 행 번호로 사용합니다. 입력 행 또는 레코드를 순서대로 정렬하려고 시도하지 않습니다. 순서 필드가 지정되지 않으면 변환기가 자체 행 번호를 지정합니다. SEQUENCE 매개변수는 다음과 같습니다.

m 맨 왼쪽 순서 번호 열.

n 맨 오른쪽 순서 번호 열.

시퀀스 번호 필드는 8자를 초과해서는 안되며 MARGINS 옵션에 지정된 대로 소스 프로그램을 겹치지 않아야 합니다.

고정 길이 레코드의 경우 기본값은 SEQUENCE(73,80)입니다. C 및 C++에서 가변 길이 레코드의 경우 기본값은 NOSEQUENCE이고 PL/I에서 가변 길이 레코드의 경우 기본값은 SEQUENCE(1,8)입니다.

SOURCE

약어: S

SOURCE는 변환된 소스 프로그램 목록을 변환기 목록에 포함하도록 변환기에 지시합니다. SOURCE는 기본값이며, 대체 옵션은 NOSOURCE입니다.

SP

특수(SP) CICS 명령을 포함하는 애플리케이션에는 SP를 지정해야 합니다. 그렇지 않으면 변환 시 거부됩니다. 이러한 명령은 ACQUIRE, COLLECT, CREATE, DISABLE, DISCARD, ENABLE, EXTRACT, INQUIRE, PERFORM, RESYNC, SET입니다. 시스템 프로그래머가 이러한 명령을 사용합니다. 이러한 명령에 대한 프로그래밍 정보는 참조 > 시스템 프로그래밍의 시스템 명령의 내용을 참조하십시오.

SPACE(1 또는 2 또는 3)

(COBOL 전용)

SPACE는 출력 목록에서 사용할 간격 유형을 표시합니다. SPACE(1)은 1행 간격, SPACE(2)는 2행 간격, SPACE(3)은 3행 간격을 지정합니다. SPACE(3)이 기본값입니다.

SPIE

SPIE은 변환기가 복구 불가능한 오류를 트랩하도록 지정합니다. SPIE는 기본값이며, 대체 옵션은 NOSPIE입니다.

SYSEIB

SYSEIB는 프로그램이 애플리케이션 EIB 대신 시스템 EIB를 사용함을 표시합니다. SYSEIB 옵션은 애플리케이션 EIB를 갱신하지 않고 프로그램이 CICS 명령을 실행할 수 있게 하여 실행의 해당 측면을 애플리케이션에 투명하게 합니다. 이 옵션을 사용하는 프로그램에 제한을 두므로 특수 상황에서만 이 옵션을 사용하십시오. SYSEIB 옵션을 사용하여 변환되는 프로그램은 다음을 수행해야 합니다.

- 시스템 EIB가 『TASKDATALOC(ANY)』 저장영역에 있다고 추정되므로 AMODE(31)에서 실행되어야 합니다.
- ADDRESS EIB 명령을 사용하는 시스템 EIB의 주소를 확보해야 합니다(프로그램이 SYSEIB 옵션을 사용하여 변환된 경우, 이 명령은 시스템 EIB의 주소를 자동으로 리턴함).
- SYSEIB 옵션 사용은 프로그램이 실행한 모든 CICS 명령에서 NOHANDLE 옵션 사용을 나타냅니다. 명령은 필요에 따라 RESP 옵션을 사용할 수 있습니다.

VBREF

(COBOL, C, C++ 및 PL/I 전용)

VBREF는 변환기가 변환기 목록에 행 번호가 있는 명령의 교차 참조를 포함할지 여부를 지정합니다(VBREF를 XREF라고 했으며 여전히 허용됨).

데이터 정의(DD 이름) 목록

DD 이름 목록은 반자 경계에서 시작되어야 합니다. 처음 2바이트에는 목록에 있는 바이트 수의 2진 계수를 포함합니다(계수 필드 제외). 목록에서 각 항목은 8바이트 필드를 차지해야 합니다.

항목의 순서는 다음과 같습니다.

항목	표준 DD 이름	항목	표준 DD 이름	항목	표준 DD 이름
1	해당사항 없음	3	해당사항 없음	5	SYSIN
2	해당사항 없음	4	해당사항 없음	6	SYSPRINT
				7	SYSPUNCH

적용 가능한 항목을 생략하면 변환기는 표준 DD 이름을 사용합니다. 8바이트 길이보다 작은 DD 이름을 사용하는 경우에는 필드 오른쪽에 공백으로 채워십시오. 첫 번째 바이트에 X'FF'를 배치하여 항목을 생략할 수 있습니다. 목록의 끝에 전체적으로 항목을 생략할 수 있습니다.

CICS 변환기 사용

언어 변환기는 소스 프로그램을 읽고 새 프로그램을 작성합니다. 대부분의 일반적인 언어 명령문은 변경되지 않지만 CICS 명령은 코드화하고 있는 언어에 필요한 양식의 CALL문으로 변환됩니다.

호출은 CICS 제공 『EXEC』 인터페이스 모듈(나중에 로드 모듈에 링크 편집됨)을 호출하며 이는 실행 시 요청된 서비스를 다시 호출합니다.

변환기 옵션을 지정하여 변환 프로세스를 제어할 수 있습니다. 113 페이지의 『변환기 옵션 정의』의 내용을 참조하십시오.

다음 방법으로 선택사항을 지정할 수 있습니다.

- 컴파일러(어셈블러)가 옵션 지정을 위해 제공하는 명령문에서 XOPTS 옵션의 서브 옵션으로 나열하십시오.

표 6. 각 언어에 제공된 명령문

언어	명령문
COBOL	CBL
COBOL	PROCESS
C	#pragma
C++	#pragma
PL/I	* PROCESS
어셈블러	*ASM 또는 *PROCESS ¹

- 변환 단계에 대한 EXEC 작업 제어문의 PARM 피연산자에 옵션을 나열하십시오.

대부분의 설치의 카탈로그화된 프로시저를 사용하여 CICS 프로그램을 변환, 컴파일 (어셈블) 및 링크하므로 프로시저를 호출하는 EXEC 작업 제어문에 이 PARM 필드를 지정합니다.

예를 들어, COBOL 프로그램의 프로시저 이름이 DFHYITVL이고 프로시저의 변환 단계 이름이 TRN이면 다음 예제 명령문은 COBOL 프로그램에 변환기 옵션을 설정합니다.

```
// EXEC DFHYITVL,PARM=TRN=(VBREF,QUOTE,SPACE(2),NOCBLCARD)
```

한 방법을 사용하여 옵션을 지정하고 다른 방법을 사용하여 동일한 옵션 또는 충돌하는 옵션을 지정하면 언어 명령문의 스펙이 EXEC문의 스펙을 덮어씁니다. 마찬가지로, 단일 옵션 또는 어느 한 명령문 유형에서 충돌하는 옵션에 다중 값을 지정하면 마지막 설정이 우선합니다. COBOL 프로그램을 제외하고 이러한 명령문은 각 소스 프로그램에 선행해야 합니다. 다른 언어로 다중 프로그램 처리를 일괄처리할 방법은 없습니다.

변환기 옵션은 하나 이상의 공백 또는 쉼표로 구분되어 임의의 순서로 나타날 수 있습니다. 옵션에 대한 언어 명령문에 변환기 옵션을 지정하는 경우, 다른 옵션은 변환기가 무시하고 컴파일러로 전달되기 때문에 XOPTS 매개변수 뒤에 괄호 안에 넣어 나타나야 합니다. 다음 COBOL 예제는 함께 전달되는 변환기 및 컴파일러 옵션 둘 다를 표시합니다.

```
CBL LIB XOPTS(QUOTE SPACE(2))
```

다음 예제는 단독으로 전달되는 변환기 옵션을 표시합니다.

```
#pragma XOPTS(FLAG(W) SOURCE);
* PROCESS XOPTS(FLAG(W) SOURCE);
*ASM XOPTS(NOPROLOG NOEPILOG)
```

EXEC 작업 제어문의 PARM 피연산자를 사용하여 옵션을 지정하는 경우, 여기서 허용되는 유일한 옵션이 변환기 옵션이므로 XOPTS 키워드는 불필요합니다. 그러나 연관된 괄호를 사용하거나 사용하지 않고 XOPTS를 사용할 수 있습니다. XOPTS에 괄호를 사용하는 경우 변환기 옵션을 모두 괄호 안에 넣으십시오. 예를 들어, 다음 양식은 올바릅니다.

```
PARM=(op1 op2 .. opn)
PARM=(XOPTS op1 op2 .. opn)
PARM=XOPTS(op1 op2 .. opn)
```

다음 양식은 올바르지 않습니다.

```
PARM=(XOPTS(op1 op2) opn)
```

이전 릴리스와의 호환성을 위해 일괄처리 EXEC DLI 프로그램을 변환하는 경우를 제외하고 CICS 키워드를 XOPTS의 대안으로 사용할 수 있습니다. PARM 피연산자를

사용하여 C 또는 C++ #pragma 카드 처리의 기본 여백을 변경하는 경우 순서 여백도 변경해야 한다는 점을 기억하십시오. NOSEQUENCE 옵션을 사용하여 이를 수행할 수 있습니다.

참고:

1. 어셈블러 언어 프로그램의 경우, *ASM문은 변환기 옵션만 포함합니다. 어셈블러가 이를 주석으로 간주합니다. *PROCESS문은 상위 레벨 어셈블러(HLASM)에 대한 변환기 또는 어셈블러 옵션을 포함할 수 있습니다.
2. 변환기 및 어셈블러 옵션은 동일한 *PROCESS문에 공존하지 않아야 합니다.
3. *PROCESS 및 *ASM문은 입력 시작 부분에 있어야 하며 어셈블러 명령문이 그 앞에 나타나지 않아야 합니다. 여기에는 주석과 명령문(『PRINT ON』 및 『EJECT』)이 포함됩니다. *PROCESS 및 *ASM문 둘 다 임의 순서로 포함될 수 있습니다.
4. 변환기 옵션만 포함된 *PROCESS문은 변환기에 대한 정보만 포함하며 어셈블러로 전달되지 않습니다.
5. 어셈블러 옵션이 포함된 *PROCESS문은 변환된 프로그램에 배치됩니다.

관련 개념:

96 페이지의 『변환 프로세스』

통합 변환기가 없는 컴파일러의 경우, CICS는 EXEC CICS, EXEC CPSM 및 EXEC DLI문을 처리하기 위해 쓸 수 있는 각 언어에 변환기 프로그램을 제공합니다. 통합 변환기가 있는 컴파일러의 경우, 컴파일러가 CICS 변환기를 직접 호출하여 이러한 명령 변환을 처리합니다.

변환기 옵션 정의

별도로 언급된 경우를 제외하고 모든 언어에 적용되는 변환기 옵션을 지정할 수 있습니다.

114 페이지의 표 7은 모든 변환기 옵션, 적용되는 프로그램 언어, 올바른 약어를 나열합니다.

설치에서 분배된 양식의 CICS 제공 프로시저를 사용하는 경우 기본 옵션이 사용됩니다. 이는 다음 옵션 설명에 명시적으로 설명되어 있습니다. 변환 단계의 출력을 나열하는 SYSPRINT 변환기를 살펴보면 설치에서 기본적으로 사용되는 옵션을 알 수 있습니다(99 페이지의 『CICS 제공 변환기』 참조). 기본값이 아닌 옵션을 원할 경우 111 페이지의 『CICS 변환기 사용』에 설명된 대로 옵션을 지정해야 합니다.

변환기 옵션 표

변환기 옵션이 적용되는 프로그램 언어와 함께 표 양식으로 표시되어 있습니다.

표 7. 프로그래밍 언어에 적용 가능한 변환기 옵션

변환기 옵션	COBOL	C	C++	PL/I	어셈블러
APOST 또는 QUOTE	X				
CBLCARD 또는 NOCBLCARD	X				
CICS	X	X	X	X	X
COBOL2	X				
COBOL3	X				
CPP			X		
CPSM 또는 NOCPSPM	X	X	X	X	X
DBCS	X				
DEBUG 또는 NODEBUG	X	X	X	X	
DLI	X	X	X	X	X
EDF 또는 NOEDF	X	X	X	X	X
EPILOG 또는 NOEPILOG					X
EXCI	X	X	X	X	X
FEPI 또는 NOFEPI	X	X	X	X	X
FLAG(I 또는 W 또는 E 또는 S)	X	X	X	X	
GDS		X	X		X
GRAPHIC				X	
LEASM					X
LENGTH 또는 NOLENGTH	X			X	X
LINECOUNT(n)	X	X	X	X	X
LINKAGE 또는 NOLINKAGE	X				
MARGINS(m,n)		X	X	X	

표 7. 프로그래밍 언어에 적용 가능한 변환기 옵션 (계속)

변환기 옵션	COBOL	C	C++	PL/I	어셈블러
NATLANG	X	X	X	X	X
NUM 또는 NONUM	X				
OPMARGINS(m,n[,c])		X	X	X	
OPSEQUENCE(m,n) 또는 NOOPSEQUENCE		X	X	X	
OPTIONS 또는 NOOPTIONS	X	X	X	X	X
PROLOG 또는 NOPROLOG					X
QUOTE 또는 APOST	X				
SEQ 또는 NOSEQ	X				
SEQUENCE(m,n) 또는 NOSEQUENCE		X	X	X	
SOURCE 또는 NOSOURCE		X	X	X	
SP	X	X	X	X	X
SPACE(1 또는 2 또는 3)	X				
SPIE 또는 NOSPIE	X	X	X	X	X
SYSEIB	X	X	X	X	X
VBREF 또는 NOVBREF	X	X	X	X	

COPY문 사용

컴파일러(또는 어셈블러)는 원래 소스가 아닌 변환된 버전의 프로그램을 입력으로 읽습니다. 이는 컴파일러(어셈블러) 목록에 표시된 내용에 영향을 줍니다. 변환기가 변환하려면 너무 늦기 때문에 소스 코드의 COPY문이 변환되지 않은 CICS 명령을 가져오지 않아야 함도 의미합니다.

이 태스크 정보

개별 변환기를 사용하고 카피북 내의 소스가 CICS 명령을 포함하는 경우, 포함될 프로그램을 변환 및 컴파일하기 전에 개별적으로 변환해야 합니다. 통합 CICS 변환기를 사용하고 카피북 내의 소스가 CICS 명령을 포함하는 경우, 포함될 프로그램을 컴파일하기 전에 개별적으로 변환하지 않아도 됩니다.

모든 CICS 명령이 포함된 카피북에 있더라도 외부 프로그램은 항상 CICS 변환기를 통과하거나 통합 CICS 변환기가 있는 컴파일러를 사용하여 컴파일해야 합니다.

CICS 제공 인터페이스 모듈

CICS에서 실행될 각 애플리케이션에는 CICS 기능을 사용하려면 하나 이상의 인터페이스 모듈(스텝으로도 알려짐)이 필요합니다.

다음 기능을 사용하려면 애플리케이션에 스텝이 필요합니다.

- EXEC 인터페이스
- CPI 통신 기능
- SAA 자원 복구 기능
- CICSplex SM API(Application Programming Interface)(CICSplex SM 스텝에 대한 정보는 *CICSplex SM 애플리케이션 프로그래밍 안내서* 참조)

EXEC 인터페이스 모듈

각 CICS 애플리케이션은 CICS에 대한 인터페이스를 포함해야 합니다. 이는 CICS 상위 레벨 프로그래밍 인터페이스에서 사용되는 EXEC 인터페이스 모듈 양식을 사용합니다. CICSTS53.CICS.SDFHLOAD 라이브러리에 설치된 모듈을 사용자의 코드와 EXEC 인터페이스 프로그램 DFHEIP 간의 통신을 제공하기 위해 애플리케이션에 링크 편집해야 합니다.

CPI 통신 인터페이스 모듈

통신용 공통 프로그래밍 인터페이스(CPI 통신)를 사용하는 각 CICS 애플리케이션은 CPI 통신에 대한 인터페이스를 포함해야 합니다. 이는 CICS 상위 레벨 프로그래밍 인터페이스에서 사용되는, 모든 프로그래밍 언어에 공통된 인터페이스 모듈 양식을 사용합니다. CICSTS53.CICS.SDFHLOAD 라이브러리에 설치된 모듈(DFHCPLC)을 CPI 통신을 사용하는 각 애플리케이션에 링크 편집해야 합니다.

SAA 자원 복구 인터페이스 모듈

SAA 자원 복구를 사용하는 각 CICS 애플리케이션은 SAA 자원 복구에 대한 인터페이스를 포함해야 합니다. 이는 CICS 상위 레벨 프로그래밍 인터페이스에서 사용되는, 모든 프로그래밍 언어에 공통된 인터페이스 모듈 양식을 사용합니다. CICSTS53.CICS.SDFHLOAD 라이브러리에 설치된 모듈(DFHCPLRR)을 SAA 자원 복구 기능을 사용하는 각 애플리케이션에 링크 편집해야 합니다.

AMODE(24) 및 AMODE(31) 애플리케이션에 EXEC 인터페이스 모듈 사용

AMODE(24) 및 AMODE(31) 애플리케이션의 경우, 언어 변환기가 생성하는 CALL 문은 사용자의 코드와 CICS EXEC 인터페이스 프로그램인 DFHEIP 간의 통신을 제공하는 EXEC 인터페이스 모듈을 호출합니다.

언어 변환기는 소스 프로그램을 읽고 새 프로그램을 작성합니다. 일반적인 언어 명령문은 변경되지 않지만 CICS 명령은 코드화하고 있는 언어에 필요한 양식의 CALL문으로 변환됩니다. 호출은 CICS 제공 『EXEC』 인터페이스 모듈 또는 스텝(즉, CICS 상위 레벨 프로그래밍 인터페이스에서 사용되는 코드의 함수 종속 섹션)을 호출합니다. SDFHLOAD 라이브러리에 제공되는 스텝을 애플리케이션에 링크 편집해야 합니다. 이러한 스텝은 EXEC CICS 및 EXEC DLI 명령 실행 중에 호출됩니다. 스텝은 각 프로그래밍 언어에 제공됩니다.

표 8. 인터페이스 모듈(스텝)

언어	인터페이스 모듈 이름
어셈블러	DFHELII 및 DFHEAI0
LEASM 옵션을 사용하는 모든 HLL 언어 및 어셈블러 MAIN 프로그램	DFHELII

CICS 제공 스텝 루틴은 호환 불가능한 방법으로는 변경되지 않는 내부 프로그래밍 인터페이스인 CICS 명령 레벨 인터페이스와 작동합니다. 따라서 이러한 스텝 모듈은 이전 및 이후 버전과 호환 가능하며, 이후 레벨의 스텝을 포함하도록 CICS 애플리케이션 모듈을 다시 링크하지 않아도 됩니다.

DFHEAI0를 제외하고 이러한 스텝은 모두 동일한 기능을 제공하는데, EXEC CICS 명령에서 필수 CICS 서비스로의 연계를 제공하는 것입니다. 이를 수행하기 위해 스텝은 변환된 EXEC CICS 명령에서 호출되는 다양한 시작점을 제공한 후 CICS의 EXEC 인터페이스 함수로 제어를 전달하는 명령어 순서를 실행합니다.

DFHELII는 대부분 기존 버전의 CICS PL/I 변환기에 호환성을 제공하는 다중 시작점을 포함합니다. DFHEXEC(C 및 C++ 애플리케이션의 경우), DFHEI1(COBOL 및 어셈블러의 경우), DFHEI01(PL/I의 경우) 항목을 포함합니다.

각 스텝은 DFHY $xnnn$ 양식의 8바이트 아이캐처로 시작합니다. 여기서 x 는 스텝이 지원하는 언어를 나타내며(A는 어셈블러를 나타내고 I는 스텝이 언어 독립적임을 나타냄) nnn 은 스텝이 포함된 CICS 릴리스를 나타냅니다. 아이캐처의 문자 Y는 스텝이 읽기 전용임을 나타냅니다. CICS 초기 릴리스와 함께 제공되었던 스텝에는 DFHE $xxxx$ 양식의 아이캐처가 포함되었습니다(문자 E는 스텝이 읽기 전용이 아님을 나타냄). z/OS용 CICS Transaction Server, 버전 5 릴리스 3 에서 DFHELII의 아이캐처는 DFHYI700입니다.

아이캐처는 CICS 애플리케이션 로드 모듈이 가장 최근에 링크된 CICS 릴리스를 판별하는 데 유용합니다.

COBOL

각 EXEC 명령은 DFHEI1 항목을 참조하는 COBOL CALL문으로 변환됩니다.

다음 예제는 단순 **EXEC CICS RETURN** 명령 처리 시 변환기가 생성한 출력을 표시합니다.

```
*EXEC CICS RETURN END-EXEC
      Call 'DFHEI1' using by content x'0e0800000600001000'
end-call.
```

DFHEI1에 대한 참조는 CICS 제공 프로시저(예: DFHYITVL 또는 DFHZITCL)의 연계 편집기 단계에 DFHELII 스텝 루틴을 포함시켜 해결합니다.

PL/I

PL/I 프로그램을 변환할 때 각 EXEC 명령은 시작점 DFHEI01에 대한 호출을 생성합니다. 이는 DFHEI01 항목과 연관된 변수 시작점 DFHEI0을 사용하여 수행됩니다. 변환기는 변환된 각 프로그램의 시작 부분 근처에 다음 명령문을 삽입하여 이를 가능하게 합니다.

```
DCL DFHEI0 ENTRY VARIABLE INIT(DFHEI01) AUTO;
DCL DFHEI01 ENTRY OPTIONS(INTER ASSEMBLER);
```

변환기는 성공적으로 변환된 각 EXEC 명령에 대해 DFHEI0을 기반으로 고유 항목 이름을 작성합니다. 다음 예제는 단순 **EXEC CICS RETURN** 명령 처리 시 변환기가 생성한 출력을 표시합니다.

```
/* EXEC CICS RETURN TRANSID(NEXT) */
DO;
DCL DFHENTRY_B62D3C38_296F2687 BASED(ADDR(DFHEI0)) OPTIONS(INTER ASSEMBLER) ENTRY(*,CHAR(4));
CALL DFHENTRY_B62D3C38_296F2687('xxxxxxxxxxxxxxxx' /* '0E 08 80 00 03
00 00 10 00 F0 F0 F0 F0 F0 F1 F0 'X */ , NEXT);
END;
```

이전 예제에서 DFHENTRY_B62D3C38_296F2687은 실제 항목 DFHEI01과 연관된 항목 변수 DFHEI0을 기반으로 합니다. 이 기술을 통해 변환기는 각 변수 항목 이름에 대한 PL/I 데이터 설명자 목록을 작성할 수 있습니다. 그러면 PL/I 컴파일러가 EXEC 명령에 참조된 변수 이름이 데이터 설명자 목록에 변환기가 정의한 속성과 일치하는 속성을 사용하여 정의되었는지 확인할 수 있습니다. 이 예제에서 ENTRY(*,CHAR(4))는 TRANSID 옵션과 연관된 변수(NEXT)가 길이가 4인 문자열이어야 함을 지정합니다.

DFHEI01에 대한 참조는 CICS 제공 프로시저 중 하나(예: DFHYITPL)의 연계 편집기 단계에 DFHELII 스텝 루틴을 포함시켜 해결합니다.

C 및 C++

C 및 C++ 프로그램에서 각 EXEC CICS 명령은 명령 변환기에 의해 DFHEXEC 함수 호출로 변환됩니다.

변환기는 변환된 각 프로그램의 시작 부분 근처에 다음 명령문을 삽입하여 이를 가능하게 합니다.

```
#pragma linkage(DFHEXEC,OS)    /* force OS linkage */
void DFHEXEC();                /* function to call CICS */
```

다음 예제는 단순 **EXEC CICS RETURN** 명령 처리 시 변환기가 생성한 출력을 표시합니다.

```
/* EXEC CICS RETURN */
{
DFHEXEC( "\x0E\x08\x00\x2F\x00\x00\x10\x00\xF0\xF0\xF0\xF1\xF8\xF0\xF0");
}
```

DFHEXEC에 대한 참조는 CICS 제공 프로시저 중 하나(예: DFHYITDL, DFHZITDL, DFHZITEL, DFHZITFL 또는 DFHZITGL)의 연계 편집기 단계에 DFHELII 스텝 루틴을 포함시켜 해결합니다.

어셈블러 언어

각 EXEC 명령은 DFHECALL 매크로 호출로 변환됩니다.

다음 예제는 단순 **EXEC CICS RETURN** 명령 처리 시 변환기가 생성한 출력을 표시합니다.

```
*          EXEC CICS RETURN
          DFHECALL =X'0E080000080001000'
```

표시된 대로 DFHECALL 매크로 호출의 어셈블리는 레지스터 1이 주소 지정한 매개 변수 목록을 빌드하고 레지스터 15에 DFHEI1 항목의 주소를 로드하며, BALR 명령어를 실행하여 스텝 루틴을 호출하는 코드를 생성합니다.

```
DS    0H
LA     1,DFHEITPL
LA     14,=x'0E080000080001000'
ST     14,0(,1)
OI     0(1),x'80'
L      15,=V(DFHEI1)
BALR   14,15
```

DFHEI1에 대한 참조는 CICS 제공 프로시저 중 하나(예: DFHEITAL)의 연계 편집기 단계에 DFHEAI 스텝 루틴을 포함시켜 해결합니다. z/OS용 CICS Transaction Server, 버전 5 릴리스 3 에서 DFHEAI의 아이캐처는 DFHYA700입니다(릴리스 번호는 이 스텝이 z/OS용 CICS Transaction Server, 버전 5 릴리스 3 과 함께 제공됨을 나타냄).

어셈블러 애플리케이션의 DFHEAI0 스텝은 연계 편집기 또는 바인더 유틸리티의 자동 호출 기능을 통해 포함됩니다. 각각 어셈블러 애플리케이션의 동적 저장영역을 얻고 비우기 위해 DFHEIENT 및 DFHEIRET 매크로가 생성한 코드를 통해 호출됩니다. 이 스텝은 어셈블러 애플리케이션에서만 필요합니다. 고급 언어로 작성된 프로그램에 동등한 기능을 제공하기 위해 필요하거나 제공된 스텝은 없습니다.

DFHEAI 스텝이 링크 편집의 출력에 프로그램의 시작에 포함되어야 합니다. 이를 달성하기 위해, DFHEAI에 대한 ORDER 및 INCLUDE문이 JCL의 링크 편집 단계에 있어야 합니다. 어셈블러로 작성된 애플리케이션을 변환, 어셈블 및 링크 편집하기 위해 SDFHPROC 라이브러리의 CICS 제공 어셈블러 프로시저를 사용할 때, 이 프로시저의 COPYLINK 단계가 SDFHMAC(DFHEILIA)를 복사합니다. DFHEILIA는 포함되어야 하는 다음 명령문을 포함합니다.

```
ORDER DFHEAI
INCLUDE SYSLIB(DFHEAI)
```

명령문은 프로시저의 LKED 단계에 있는 어셈블된 애플리케이션 앞에서 연결되는 임시 파일에 넣어집니다.

LEASM이 있는 예제 어셈블러 언어 프로그램

예제 CICS 어셈블러 프로그램 및 변환된 후 해당하는 코드는 LEASM 옵션을 사용하여 어셈블러 프로그램을 변환하는 결과를 보여줍니다. 이 변환기 옵션은 Language Environment 준수 어셈블러 MAIN 프로그램을 위한 코드를 생성합니다.

그림 12에서는 단순 CICS 어셈블러 언어 프로그램을 표시합니다.

```
*ASM XOPTS(LEASM)
DFHEISTG DSECT
OUTAREA DS CL200 DATA OUTPUT AREA
*
EIASM CSECT ,
      MVC OUTAREA(40),MSG1
      MVC OUTAREA(4),EIBTRMID
      EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(43) FREEKB ERASE
      EXEC CICS RECEIVE
      MVC OUTAREA(13),MSG2
      EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(13) FREEKB ERASE
      EXEC CICS RETURN
*
MSG1 DC C'xxxx: ASM program invoked. ENTER TO END.'
MSG2 DC C'PROGRAM ENDED'
END
```

그림 12. 단순 CICS 어셈블러 언어 프로그램.

『LEASM이 있는 예제 어셈블러 언어 프로그램』에서는 프로그램이 변환되고 어셈블된 후에 코드를 보여줍니다.

```

ASM XOPTS(LEASM)
      DFHEIGBL ,,,LE          INSERTED BY TRANSLATOR
*,&DFHEIDL; SETB 0 1 MEANS EXEC DLI IN PROGRAM          01-DFHEI
*,&DFHEIDB; SETB 0 1 MEANS BATCH PROGRAM                01-DFHEI
*,&DFHEIRS; SETB 0 1 MEANS RSECT                        01-DFHEI
*,&DFHEILE; SETB 1 1 MEANS LE MAIN                      01-DFHEI
DFHEISTG DSECT
      DFHEISTG          INSERTED BY TRANSLATOR
*****
*          EXEC INTERFACE DYNAMIC STORAGE          *
*****
DFHEISTG DSECT          EXEC INTERFACE STORAGE          @BBAC81A 01-DFHEI
      USING *,DFHEIPLR      ESTABLISH ADDRESSABILITY      @BBAC81A 01-DFHEI
*
*****
*  D Y N A M I C  S T O R A G E  A R E A  ( D S A )  *
*****
*
CEEDSA  DS      0D          Just keep the same label for formulae      02-CEEDS
*
CEEDSAFLAGS DS XL2          DSA flags          02-CEEDS
CEEDSALNGC EQU X'1000'      C library DSA          02-CEEDS
CEEDSALNGP EQU X'0800'      PL/I library DSA        02-CEEDS
CEEDSAEXIT EQU X'0008'      An Exit DSA            02-CEEDS
CEEDSAMEMD DS XL2          Member defined          02-CEEDS
CEEDSABKC DS A             Addr of DSA of caller    02-CEEDS
CEEDSAFWC DS A             Addr of DSA of last called rtn 02-CEEDS
CEEDSAR14 DS F             Save area for register 14 02-CEEDS
CEEDSAR15 DS F             Save area for register 15 02-CEEDS
CEEDSAR0 DS F             Save area for register 0  02-CEEDS
CEEDSAR1 DS F             Save area for register 1  02-CEEDS
CEEDSAR2 DS F             Save area for register 2  02-CEEDS
CEEDSAR3 DS F             Save area for register 3  02-CEEDS
CEEDSAR4 DS F             Save area for register 4  02-CEEDS
CEEDSAR5 DS F             Save area for register 5  02-CEEDS
CEEDSAR6 DS F             Save area for register 6  02-CEEDS
CEEDSAR7 DS F             Save area for register 7  02-CEEDS
CEEDSAR8 DS F             Save area for register 8  02-CEEDS
CEEDSAR9 DS F             Save area for register 9  02-CEEDS
CEEDSAR10 DS F            Save area for register 10 02-CEEDS
CEEDSAR11 DS F            Save area for register 11 02-CEEDS
CEEDSAR12 DS F            Save area for register 12 02-CEEDS
CEEDSALWS DS A             Addr of PL/I Language Working Space 02-CEEDS
CEEDSANAB DS A             Addr of next available byte 02-CEEDS
CEEDSAPNAB DS A            Addr of end-of-prolog NAB 02-CEEDS
      DS 4F              02-CEEDS
CEEDSATRAN DS 0A           HPL TxArea or          02-CEEDS
CEEDSARENT DS A           Program reentry address-IPAT 02-CEEDS
CEEDSACILC DS A           C to Fortran ILC save area 02-CEEDS
CEEDSAMODE DS A           Return address of module that 02-CEEDS
*                          caused the last mode switch
      DS 2F              02-CEEDS
CEEDSARMR DS A            Addr of language specific    02-CEEDS
*                          exception handler
*
      DS F              Reserved          02-CEEDS
CEEDSAAUTO DS 0D          Automatic storage starts here 02-CEEDS
CEEDSAEND DS 0D           End of DSA          02-CEEDS

```

그림 13. 변환 및 어셈블리 후의 어셈블리 언어 프로그램

```

CEEDSASZ EQU    CEEDSAEND-CEEDSA      Size of DSA                      02-CEEDS
CEEDSA_STDCEEDSA EQU    X'0000'      flag values of standard CEE DSA    02-CEEDS
*
*
*
DFHEISA DS      18F                      SAVE AREA R14-R12 AT 12 OFF @BBAC81A 01-DFHEI
DFHEILWS DS      F                      RESERVED                      @BBAC81A 01-DFHEI
DFHEINAB DS      F                      RESERVED                      @BBAC81A 01-DFHEI
DFHEIRS0 DS      F                      RESERVED                      @BBAC81A 01-DFHEI
DFHEIR13 DS      F                      REGISTER 13                  @BBAC81A 01-DFHEI
DFHEIRS1 DS      F                      RESERVED                      @BBAC81A 01-DFHEI
DFHEIBP DS      F                      EIB POINTER (NOT USED IF BATCH) 01-DFHEI
DFHEICAP DS      F                      COMMAREA POINTER (NOT USED IF BATCH) 01-DFHEI
DFHEIV00 DS      H                      HALFWORD TEMP USED BY DFHECALL 01-DFHEI
DFHEIRS2 DS      H                      RESERVED                      @BBAC81A 01-DFHEI
DFHEIPL DS      13F                      PARAMETER LIST                      @05C 01-DFHEI
          DS      51F                      ALLOW 64 PARAMETERS FOR DLI          @L2A 01-DFHEI
*
*                                AND IN XA2 ON, FOR EXEC CICS ALSO
DFHEIRS3 DS      F                      RESERVED                      @L2A 01-DFHEI
DFHEIRS4 DS      F                      RESERVED                      @L2A 01-DFHEI
DFHEITP1 DS      F                      TEMPORARY POINTER 1          @L2A 01-DFHEI
DFHEITP2 DS      F                      TEMPORARY POINTER 2          @L2A 01-DFHEI
DFHEITP3 DS      F                      TEMPORARY POINTER 3          @L2A 01-DFHEI
DFHEITP4 DS      F                      TEMPORARY POINTER 4          @L2A 01-DFHEI
*****
*                                START DEFINITION OF USER DYNAMIC STORAGE                                *
*****
DFHEIUSR DS      0D                      ALIGN USER DYNAMIC STORAGE @BBAC81A 01-DFHEI
*
OUTAREA DS      CL200                      DATA OUTPUT AREA
*
TESTLE  CSECT ,
          DFHEIENT                      INSERTED BY TRANSLATOR
*****
*
* CONTROL BLOCK NAME = DFHEIBLK
*
* NAME OF MATCHING PL/AS CONTROL BLOCK = None
*
* DESCRIPTIVE NAME = %PRODUCT EXEC Interface Block.
*
* @BANNER_START                      02
* Licensed Materials - Property of IBM
*
* "Restricted Materials of IBM"
*
* 5697-E93
*
* (C) Copyright IBM Corp. 1990, 1993
*
*
* @BANNER_END
*
* STATUS = %XA20
*

```

```

* FUNCTION = EXEC Interface Block.
*
* The exec interface block contains information on the
* transaction identifier, the time and date, and the cursor
* position on a display device. Some of the other fields are
* set indicating the next action that a program should take
* in certain circumstances.
* DFHEIBLK also contains information that will be helpful
* when a dump is being used to debug a program.
* This control block is included automatically by an
* application program using the command-level interface.
* EISEIBA in the EIS addresses the EIB.
*
*
*
* NOTES :
* DEPENDENCIES = S/370
* MODULE TYPE = Control block definition
* PROCESSOR = Assembler
*
*-----*
*
* CHANGE ACTIVITY :
* $SEG(DFHEIBLK),COMP(COMMAND),PROD(%PRODUCT) :
*
* PN= REASON REL YMMDD HDXXIII : REMARKS
* $L1= 550 %OG 900515 HDFSPC : Add an EIB length equate
* $D1= I05119 %B1 930226 HDDHDMA : Correct comments for date field
* $P1= M60581 %B0 900116 HDAEGB : Change for PLXMAP to data areas
*
*****
* EXEC INTERFACE BLOCK
*****
DFHEIBLK DSECT EXEC INTERFACE BLOCK @BBAC81A 01-DFHEI
        USING *,DFHEIBR @BBAC81A 01-DFHEI
EIBTIME DS PL4 TIME IN 0HHMMSS FORMAT @BBAC81A 01-DFHEI
EIBDATE DS PL4 DATE IN 0CYYDD+ FORMAT, @D1C 01-DFHEI
* where C is the century @D1A
* indicator (0=1900, 1=2000), @D1A
* YY is the year, DDD is the @D1A
* day number and '+' is the @D1A
* sign byte (positive) @D1A
EIBTRNID DS CL4 TRANSACTION IDENTIFIER @BBAC81A 01-DFHEI
EIBTASKN DS PL4 TASK NUMBER @BBAC81A 01-DFHEI
EIBTRMID DS CL4 TERMINAL IDENTIFIER @BBAC81A 01-DFHEI
EIBRSVD1 DS H RESERVED @BBAC81A 01-DFHEI
EIBCPOSN DS H CURSOR POSITION @BBAC81A 01-DFHEI
EIBCALEN DS H COMMAREA LENGTH @BBAC81A 01-DFHEI
EIBAID DS CL1 ATTENTION IDENTIFIER @BBAC81A 01-DFHEI
EIBFN DS CL2 FUNCTION CODE @BBAC81A 01-DFHEI
EIBRCODE DS CL6 RESPONSE CODE @BBAC81A 01-DFHEI
EIBDS DS CL8 DATASET NAME @BBAC81A 01-DFHEI
EIBREQID DS CL8 REQUEST IDENTIFIER @BBAC81A 01-DFHEI
EIBRSRCE DS CL8 RESOURCE NAME @BBDIA0U 01-DFHEI
EIBSYNC DS C X'FF' SYNCPOINT REQUESTED @BBDIA0U 01-DFHEI
EIBFREE DS C X'FF' FREE REQUESTED @BBDIA0U 01-DFHEI
EIBRECV DS C X'FF' RECEIVE REQUIRED @BBDIA0U 01-DFHEI

```



```

EIBSEND DS C RESERVED @BM13417 01-DFHEI
EIBATT DS C X'FF' ATTACH RECEIVED @BBDIA0U 01-DFHEI
EIBEOC DS C X'FF' EOC RECEIVED @BBDIA0U 01-DFHEI
EIBFMH DS C X'FF' FMHS RECEIVED @BBDIA0U 01-DFHEI
EIBCOMPL DS C X'FF' DATA COMPLETE 01-DFHEI
EIBSIG DS C X'FF' SIGNAL RECEIVED 01-DFHEI
EIBCONF DS C X'FF' CONFIRM REQUESTED 01-DFHEI
EIBERR DS C X'FF' ERROR RECEIVED 01-DFHEI
EIBERRCD DS CL4 ERROR CODE RECEIVED 01-DFHEI
EIBSYNRB DS C X'FF' SYNC ROLLBACK REQ'D 01-DFHEI
EIBNODAT DS C X'FF' NO APPL DATA RECEIVED 01-DFHEI
EIBRESP DS F INTERNAL CONDITION NUMBER 01-DFHEI
EIBRESP2 DS F MORE DETAILS ON SOME RESPONSES 01-DFHEI
EIBRLDBK DS CL1 ROLLED BACK 01-DFHEI
*
EIBLENG EQU *-EIBTIME Length of EIB @L1A 01-DFHEI
*****
* END OF EXEC INTERFACE BLOCK *
*****
DFHEIBR EQU 11 EIB REGISTER @BA02936 01-DFHEI
*****
* PROLOG CODE FOR EXEC INTERFACE *
*****
*DFHEICS; CEEENTRY PPA=DFHPPA,MAIN=YES,PLIST=OS,
* BASE=&CODEREG;
* AUTO=(DFHEIEND-DFHEISTG)
TESTLE CSECT , 02-CEEEN
TESTLE RMODE ANY 02-CEEEN
TESTLE AMODE ANY 02-CEEEN
ENTRY TESTLE 02-CEEEN
PUSH USING 02-CEEEN
DROP , @02A 02-CEEEN
USING *,15 02-CEEEN
B CEEZ0007 02-CEEEN
DC X'00C3C5C5' 02-CEEEN
CEEY0007 DC A((((DFHEIEND-DFHEISTG)+7)/8)*8) X02-CEEEN
. Size of automatic storage.
DC A(DFHPPA-TESTLE) . Address of PPA for this program 02-CEEEN
B 1(,15) 02-CEEEN
CEEZ0007 EQU * 02-CEEEN
STM 14,12,CEEDSAR14-CEEDSA(13) 02-CEEEN
L 2,CEEINPL0007 5@01D @01C 02-CEEEN
L 15,CEEINT0007 @01C 02-CEEEN
DROP 15 @01A 02-CEEEN
BALR 14,15 02-CEEEN
LR 2,1 02-CEEEN
L 14,752(,12) 02-CEEEN
OI 8(14),X'80' 02-CEEEN
BALR 3,0 @01A 02-CEEEN
USING *,3 @01A 02-CEEEN
L 3,CEE0EPV0007 @01A 02-CEEEN
POP USING @01A 02-CEEEN
USING TESTLE,3 @01A 02-CEEEN
L 1,CEEDSANAB-CEEDSA(,13) Get the current NAB 02-CEEEN
L 0,CEEY0007 02-CEEEN
ALR 0,1 Compute new value. 02-CEEEN
CL 0,CEECAAEOS-CEECAA(,12) Compare with EOS. 02-CEEEN

```

```

BNH    CEEX0007                                02-CEEEN
L      15,CEECAAGETS-CEECAA(,12) Get address overflow routine 02-CEEEN
BALR   14,15                                Get another stack segment. 02-CEEEN
LR     1,15                                02-CEEEN
B      CEEX0007                                Branch around statics @01A 02-CEEEN
CEEINPL0007 DC A(CEEINPL)                    @01A 02-CEEEN
CEEINT0007 DC V(CEEINT)                      @01A 02-CEEEN
CEEOPV0007 DC A(TESTLE)                     @01A 02-CEEEN
CEEEX0007 EQU *                                02-CEEEN
ST     13,CEEDSABKC-CEEDSA(,1) Set back chain. 02-CEEEN
ST     0,CEEDSANAB-CEEDSA(,1) Set new NAB value 02-CEEEN
XC     CEEDSAFLAGS-CEEDSA(,1),CEEDSAFLAGS-CEEDSA(1) . Clear 02-CEEEN
ST     1,CEEDSAFWC-CEEDSA(,13) Set forward chain. 02-CEEEN
LR     13,1                                Set save area address 02-CEEEN
USING  CEEDSA,13                            Addressability to SF V1R2M0 02-CEEEN
MVC    CEEDSALWS,CEECAALWS-CEECAA(12) Get LWS addr V1R2M0 02-CEEEN
LR     1,2                                02-CEEEN
BAL    1,*,+8                                @L2A 01-DFHEI
* The following gives an assembler message if DFHEISTG is too big @P7A
DS     0S((DFHEISTG+65264-DFHEIEND-4096)/4096) @04C 01-DFHEI
DC     AL2(DFHEIEND-DFHEISTG) LENGTH OF STORAGE @L2A 01-DFHEI
DC     H'0' Parameter list version number @P6C 01-DFHEI
*****
* ESTABLISH DATA ADDRESSIBILITY *
*****
DFHEIPLR EQU 13 PARAMETER LIST REGISTER @BBAC81A 01-DFHEI
LR     DFHEIPLR,15 @BBAC81A 01-DFHEI
USING  DFHEISTG,13 @BBAC81A 01-DFHEI
MVC    DFHEIBP(L'DFHEIBP+L'DFHEICAP),0(1) @D3AX01-DFHEI
COPY EIB AND CA PTRS @D3A
*****
* ESTABLISH EIB ADDRESSIBILITY *
*****
L      DFHEIBR,DFHEIBP @BBAC81A 01-DFHEI
USING  DFHEIBLK,DFHEIBR @BBAC81A 01-DFHEI
*****
* END OF PROLOG CODE FOR EXEC INTERFACE *
*****
MVC    OUTAREA(40),MSG1
MVC    OUTAREA(4),EIBTRMID
* EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(43) FREEKB ERASE
DFHECALL =X'180660000800C20000082204000020',,(____RF,OUTAREA*
), (FB_2,=Y(43))
*****
DS     0H                                01-DFHEC
LA     1,DFHEIPL                                01-DFHEC
LA     14,=X'180660000800C20000082204000020' 01-DFHEC
SR     15,15                                01-DFHEC
LA     0,OUTAREA                                01-DFHEC
STM    14,0,0(1)                                01-DFHEC
LA     14,=Y(43)                                01-DFHEC
ST     14,12(,1)                                01-DFHEC
OI     12(1),X'80' LAST ARGUMENT 01-DFHEC
L      15,=V(DFHEI1)                            01-DFHEC
BALR   14,15 INVOKE EXEC INTERFACE 01-DFHEC
*****
* EXEC CICS RECEIVE

```

```

DFHECALL =X'040200000800000014000040000000'
*****
DS      0H                                01-DFHEC
LA      1,DFHEIPL                        01-DFHEC
LA      14,=X'040200000800000014000040000000' 01-DFHEC
ST      14,0(,1)                        01-DFHEC
OI      0(1),X'80'                      01-DFHEC
L       15,=V(DFHEI1)                   01-DFHEC
BALR    14,15                          01-DFHEC
*****
MVC     OUTAREA(13),MSG2
*       EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(13) FREEKB ERASE
DFHECALL =X'180660000800C20000082204000020',,(____RF,OUTAREA*
), (FB_2,=Y(13))
*****
DS      0H                                01-DFHEC
LA      1,DFHEIPL                        01-DFHEC
LA      14,=X'180660000800C20000082204000020' 01-DFHEC
SR      15,15                          01-DFHEC
LA      0,OUTAREA                       01-DFHEC
STM     14,0,0(1)                      01-DFHEC
LA      14,=Y(13)                      01-DFHEC
ST      14,12(,1)                      01-DFHEC
OI      12(1),X'80'                    01-DFHEC
L       15,=V(DFHEI1)                   01-DFHEC
BALR    14,15                          01-DFHEC
*****
*       EXEC CICS RETURN
DFHECALL =X'0E0800000800001000'
*****
DS      0H                                01-DFHEC
LA      1,DFHEIPL                        01-DFHEC
LA      14,=X'0E0800000800001000'        01-DFHEC
ST      14,0(,1)                      01-DFHEC
OI      0(1),X'80'                    01-DFHEC
L       15,=V(DFHEI1)                   01-DFHEC
BALR    14,15                          01-DFHEC
*****
*
MSG1    DC      C'xxxx: ASM program invoked. ENTER TO END.'
MSG2    DC      C'PROGRAM ENDED'
DFHEIRET                                INSERTED BY TRANSLATOR
*****
*       EPILOG CODE FOR EXEC INTERFACE                                     *
*****
DS      0H                                @BBAC81A 01-DFHEI
LA      1,CEET0014                      Get address of termination list 02-CEETE
L       15,=V(CEETREC)                  Get address of termination rtn 02-CEETE
BALR    14,15                          Call termination routine.      02-CEETE

CEET0014 DC      A(*+8)                  Parm 1                        02-CEETE
DC      A(*+8+X'80000000')              Parm 2                        02-CEETE
DC      A(0)                            Enc_Modifier                  02-CEETE
DC      A(0)                            Return code.                  02-CEETE
CEEMAIN CSECT                          02-CEETE
CEEMAIN RMODE ANY                      02-CEETE
CEEMAIN AMODE ANY                      02-CEETE

```

```

      DC      A(TESTLE)                                @04A 02-CEETE
      DC      F'0'                                      02-CEETE
TESTLE  CSECT                                          02-CEETE
*****
*          END OF EPILOG CODE FOR EXEC INTERFACE      *
*****
      LTORG ,                                          @BBAC81A 01-DFHEI
      =V(DFHEI1)
      =V(CEETREC)
      =Y(43)
      =Y(13)
      =X'180660000800C20000082204000020'
      =X'040200000800000014000040000000'
      =X'0E0800000800001000'
      DS      0H                                      @F8E1S @L1C 01-DFHEI
      DFHEISTG          INSERTED BY TRANSLATOR
      DFHEIEND          INSERTED BY TRANSLATOR
*
*****
*          P R O G R A M   P R O L O G   A R E A 1   ( P P A 1 )      *
*****
*
PPA10018  DS      0F                                      02-CEEPP
DFHPPA   DS      0F                                      02-CEEPP
      DC      AL1(PPANL0018-*) Offset to the entry name length 02-CEEPP
      DC      X'CE'          LE/370 Indicator.                02-CEEPP
      DC      B'10100000'          . PPA flags                02-CEEPP
*
*          Bit 0      0 = Internal Procedure
*                      1 = External Procedure
*
*          Bit 1      0 = Primary Entry Point
*                      1 = Secondary Entry Point
*
*          Bit 2      0 = Block doesn't have a DSA
*                      1 = Block has a DSA
*
*          Bit 3      0 = compiled object
*                      1 = library object
*
*          Bit 4      0 = sampling interrupts to library
*                      1 = sampling interrupts to code
*
*          Bit 5      0 = not an exit DSA
*                      1 = Exit DSA
*
*          Bit 6      0 = own exception model
*                      1 = inherited (callers) exception model
*
*          Bit 7      Reserved
*
      DC      X'00'          Member flags                    02-CEEPP
      DC      A(PPA20018)    Addr of Compile Unit Block (PPA2) 02-CEEPP
      DC      A(0)           02-CEEPP
      DC      A(0)           Data Descriptors for this entry point 02-CEEPP
      DS      0H            02-CEEPP
PPANL0018 DC      AL2(6)      . Length of Entry Point Name    02-CEEPP
      DC      CL6'TESTLE'    . Entry Point Name                02-CEEPP
CEEINPL  DS      0D            02-CEEPP
      DC      A(PPA2M0018)    02-CEEPP
      DC      A(CEEINPLSTST-CEEINPL) 02-CEEPP
CEEINPLSTST DS 0F            02-CEEPP
      DC      X'00'          Control Level                    @01A 02-CEEPP
      DC      X'00'          ENCLAVE=NO                        @01A 02-CEEPP
      DC      X'00'          @01A 02-CEEPP
      DC      X'07'          Number of items.                  @01C 02-CEEPP

```

```

DC    A(PPA2M0018)    . A of A(first entry point in comp unit)  02-CEEPP
DC    V(CEESTART)     . A(Address of CEESTART)                   02-CEEPP
DC    V(CEEBETBL)     .                                     02-CEEPP
DC    A(15)           . Memeber id                               02-CEEPP
DC    A(0)             .                                     02-CEEPP
DC    XL4'00070000'    . EXECOPS(ON), PLIST                      02-CEEPP
DS    0H                                                       02-CEEPP

*
*****
*      P R O G R A M   P R O L O G   A R E A   2   ( P P A 2 )      *
*****
*
      EXTRN CEESTART                                           02-CEEPP
PPA20018 DS    0F                                           02-CEEPP
      DC    AL1(15)      Member ID                             02-CEEPP
      DC    AL1(0)       Sub ID                                02-CEEPP
      DC    AL1(0)       Member defined                        02-CEEPP
      DC    AL1(1)       Level of PPAX control blocks          02-CEEPP
PPA2S0018 DC    A(CEESTART)  A(CEESTART for this load module)  02-CEEPP
      DC    A(0)          A(Compile Debug Information (CDI) )  02-CEEPP
      DC    A(CEETIMES-PPA20018) A(Offset to time stamp)      02-CEEPP
PPA2M0018 DC    A(TESTLE)   . A(first entry point in comp. unit) 02-CEEPP
*
*****
*      T I M E   S T A M P                                         *
*****
*
      Time Stamp
*,Time Stamp = 2004/06/17 08:51:00                               02-CEEPP
*,Version 1 Release 1 Modification 0                               02-CEEPP
CEETIMES DS    0F                                           02-CEEPP
      DC    CL4'2004'      Year                                02-CEEPP
      DC    CL2'06'        Month                               02-CEEPP
      DC    CL2'17'        Day                                 02-CEEPP
      DC    CL2'08'        Hours                               02-CEEPP
      DC    CL2'51'        Minutes                             02-CEEPP
      DC    CL2'00'        Seconds                             02-CEEPP
      DC    CL2'1'         Version                             02-CEEPP
      DC    CL2'1'         Release                             02-CEEPP
      DC    CL2'0'         Modification                        02-CEEPP
*****
* C O M M O N   A N C H O R   A R E A   ( C A A )                  *
*****
*****
LEPTRLEN EQU    4                                           03-CEEDN
*
CEECAA      DSECT ,                      CAA mapping          02-CEECA

      (Definition of LE CAA removed)

*      TERMINATE DEFINITION OF DYNAMIC STORAGE                  *
DFHEISTG DSECT                                @BBAC81A 01-DFHEI
      ORG                                01-DFHEI
DFHEIEND DS    0X                      END OF DYNAMIC STORAGE @BBAC81A 01-DFHEI
      END

```

AMODE(64) 애플리케이션에 EXEC 인터페이스 모듈 사용

비Language Environment AMODE(64) 어셈블러 언어 프로그램의 경우, 언어 변환기가 생성하는 CALL문은 사용자의 코드와 CICS EXEC 인터페이스 프로그램 DFHEIG 간의 통신을 제공하는 EXEC 인터페이스 모듈을 호출합니다.

언어 변환기는 소스 프로그램을 읽고 새 프로그램을 작성합니다. 일반적인 언어 명령문은 변경되지 않지만 CICS 명령은 코드화하고 있는 언어에 필요한 양식의 CALL문으로 변환됩니다. 호출은 CICS 제공 『EXEC』 인터페이스 모듈 또는 스텝(즉, CICS 상위 레벨 프로그래밍 인터페이스에서 사용되는 코드의 함수 종속 섹션)을 호출합니다. SDFHLOAD 라이브러리에 제공되는 이러한 스텝을 애플리케이션에 링크 편집해야 합니다. 이러한 스텝은 EXEC CICS 명령 실행 중에 호출됩니다.

AMODE(64) 애플리케이션에서 CICS로 제어를 전송하기 위해 다음 스텝 프로그램이 제공됩니다. 이러한 스텝은 CICS 내부 제어 블록을 사용하여 필수 CICS 코드를 찾습니다.

- DFHEAG0(prolog 및 epilog 스텝).

DFHEIENT 매크로가 이 스텝을 호출하면 제어를 AMODE(64) PROLOG 프로그램으로 제어를 전송합니다.

DFHEIRET 매크로가 이 스텝을 호출하면 AMODE(64) EPILOG 프로그램으로 제어를 전송합니다.

- DFHEAG(명령 스텝).

DFHECALL 매크로가 이 스텝을 호출하면 AMODE(64) 초기 명령 프로세서로 제어를 전송합니다. DFHEG1은 DFHEAG의 별명이며 DFHECALL 매크로는 실제로 별명 DFHEG1을 호출합니다.

DFHEAG 및 DFHEAG0 스텝을 AMODE(64) 애플리케이션에 링크 편집해야 합니다. 다음 바인더 명령문을 사용하십시오.

```
ORDER DFHEAG
INCLUDE SYSLIB(DFHEAG)
ENTRY program_name
NAME program_name(R)
```

*program_name*은 AMODE(64) 애플리케이션의 이름입니다.

이는 DFHEAI 및 DFHEAI0 스텝을 AMODE(24) 및 AMODE(31) 어셈블러 언어 애플리케이션에 링크 편집하는 방법과 비슷합니다.

관련 항목

DFHEAG

DFHEAG0

제 8 장 플랫폼에 애플리케이션 전개

다중 CICS 리전에서 CICS 애플리케이션을 전개 및 관리할 플랫폼을 설정할 수 있습니다. CICS Explorer® 또는 CICS 탐색기 SDK를 사용하여 플랫폼, 애플리케이션, 애플리케이션을 관리할 정책을 작성하십시오.

이 태스크 정보

이 태스크의 단계는 CICS Explorer 제품 문서의 애플리케이션 전개에 대한 워크플로우에 설명된 워크플로우를 요약합니다. 플랫폼에 전개할 새 애플리케이션을 개발 중이면 이 워크플로우를 따르십시오. 이 태스크의 각 단계에 대한 보다 자세한 지시사항은 CICS Explorer 제품 문서의 플랫폼, 애플리케이션, 정책에 대한 작업 및 CICS Explorer 제품 문서의 번들에 대한 작업의 내용을 참조하십시오.

기존 애플리케이션을 클라우드 인에이블먼트로 이동하려는 경우에는 클라우드 인에이블먼트 시작하기에 설명된 점진적 채택 로드맵을 따를 수 있습니다.

CICS Explorer를 사용하지 않는 자동화 애플리케이션 전개에 대한 정보는 DFHDPLOY 유틸리티를 사용하여 CICS 애플리케이션의 전개 및 전개 취소 자동화의 내용을 참조하십시오.

프로시저

1. 플랫폼을 작성하고 설치하십시오. 플랫폼 설정을 참조하십시오.
2. 플랫폼에 전개하려는 각 애플리케이션마다 애플리케이션의 구성요소를 정의하거나 가져오는 CICS 번들을 작성하고, 애플리케이션에 대한 액세스를 제공하는 자원에 대해 애플리케이션 시작점을 선언하십시오. 애플리케이션 번들과 애플리케이션 바인딩을 작성하고 내보내서 CICS 번들을 애플리케이션으로 패키징하십시오. 전개를 위한 애플리케이션 패키징을 참조하십시오.
3. 플랫폼에서 애플리케이션을 관리하기 위한 정책 규칙을 포함한 정책을 작성하십시오. 정책 규칙은 플랫폼에서 실행하는 하나 이상의 애플리케이션 태스크에 적용할 수 있는 제어 또는 조치를 설명합니다. 정책은 CICS 번들에 정의되고 전개됩니다. 정책을 참조하십시오.
4. 플랫폼에 전개하려는 각 애플리케이션을 설치하고 사용으로 설정한 후 사용 가능하게 하십시오. 플랫폼에 애플리케이션 설치를 참조하십시오.
5. 애플리케이션에 대해 생성된 애플리케이션 컨텍스트 데이터를 사용하여 각 애플리케이션이 얼마나 많은 자원을 사용 중인지 모니터링하고 측정하십시오. 자원 사용량이 임계값을 초과할 때 자동으로 응답하는 추가 정책 규칙을 추가할 수 있습니다. 애플리케이션의 자원 사용 측정을 참조하십시오.

6. 용량 요구사항이 늘어나거나 줄어드는대로 CICS 리전을 플랫폼에 추가하거나 플랫폼에서 이 리전을 제거하십시오. 플랫폼 및 애플리케이션 관리를 참조하십시오.
7. 필요에 따라 플랫폼에 애플리케이션의 새 버전을 갱신 및 설치하고 더 이상 필요하지 않을 때에는 사용 불가능하게 하거나, 사용 안함으로 설정하거나, 버리십시오. 플랫폼 및 애플리케이션 관리를 참조하십시오.

다음에 수행할 작업

IBM Workload Deployer를 사용하여 클라우드 환경에서 실행되는 가상 애플리케이션의 일부로서 전개된 CICS 애플리케이션을 통합할 수도 있습니다. 자세한 정보는 클라우드 환경에서 CICS 사용을 참조하십시오.

플랫폼에 애플리케이션 설치

패키징된 CICS 애플리케이션을 플랫폼에서 사용하려면 애플리케이션 정의를 설치하고 애플리케이션을 사용으로 설정한 후 애플리케이션을 사용 가능하게 합니다.

시작하기 전에

zFS의 플랫폼 홈 디렉토리로 애플리케이션을 내보낸 후 애플리케이션 정의를 작성하십시오. CICSplex의 데이터 저장소에 있는 APPLDEF 자원 정의인 애플리케이션 정의는 애플리케이션이 실행하는 대상 플랫폼 및 애플리케이션 번들의 위치를 식별합니다. 내보내기 프로세스 직후에 애플리케이션 정의를 작성하거나 나중에 작성할 수 있습니다. 665 페이지의 제 22 장 『전개를 위한 애플리케이션 패키징』의 내용을 참조하십시오.

이 태스크 정보

플랫폼에 다른 애플리케이션 버전이 설치되지 않은 경우 플랫폼의 첫 번째 특정 애플리케이션 설치에 이 절차를 사용하십시오. 애플리케이션의 기존 버전을 새 버전으로 바꿀 때에는 플랫폼에 전개된 애플리케이션 갱신의 전체 절차를 따르십시오.

CICS Explorer에서 CICS 클라우드 퍼스펙티브를 사용하여 CICS 애플리케이션을 플랫폼에 설치합니다. 다음 단계에 대한 자세한 도움말은 CICS Explorer 제품 문서의 애플리케이션에 대한 작업의 내용을 참조하십시오.

프로시저

1. CICS Explorer에 있는 CICS 클라우드 퍼스펙티브의 클라우드 탐색기 보기에서 애플리케이션 정의를 찾아서 설치하십시오. CICSplex SM은 CICSplex에 애플리케이션을 표현할 APPLCTN 자원을 작성하고 애플리케이션에 대한 번들의 복구 처리에 사용되는 애플리케이션에 대한 레코드를 데이터 저장소에 작성하기도 합니다.

CICSplex SM은 애플리케이션 번들 및 애플리케이션 바인딩의 정보를 사용하여 CICS 번들을 플랫폼의 CICS 리전에 설치합니다. 애플리케이션은 초기에 사용 안함 상태로 설치됩니다.

2. CICS Explorer에서 클라우드 탐색기 보거나 온라인 애플리케이션 편집기를 사용하여 설치된 애플리케이션을 사용 가능하게 설정하고 상태를 확인하십시오. 애플리케이션을 사용 가능하게 설정하면 CICSplex SM이 모든 CICS 리전에서 애플리케이션에 대해 설치된 CICS 번들을 사용 가능하게 설정하지만, 호출자는 아직 애플리케이션 시작점을 통해 애플리케이션을 사용할 수 없습니다. 애플리케이션 상태 확인에 대한 지시사항과 애플리케이션이 사용되지 않을 경우에 수행할 조치에 대해서는 애플리케이션의 상태 확인을 참조하십시오.
3. 플랫폼의 사용자에게 애플리케이션을 사용 가능하게 할 준비가 되면 CICS Explorer에서 클라우드 탐색기 보거나 온라인 애플리케이션 편집기를 사용하여 애플리케이션을 사용 가능하게 하십시오. 이제 호출자가 애플리케이션 시작점을 통해 애플리케이션을 사용할 수 있습니다.

결과

활성 플랫폼에 애플리케이션을 설치할 때 CICSplex SM은 플랫폼의 일부로 정의되어 있고 애플리케이션 정의를 설치할 때 실행 중인 모든 CICS 리전에 즉시 CICS 번들을 설치합니다. 애플리케이션 정의를 설치한 후에 리전을 시작 또는 재시작하는 경우에도 CICSplex SM은 플랫폼의 CICS 리전에 CICS 번들을 설치합니다. 이 CICS 번들의 자원은 리전 시작 시에 설치되지만 CICS로 제어가 리턴될 때까지는 완전히 사용되지 않을 수도 있습니다. 애플리케이션 정의를 설치한 후 플랫폼에 CICS 리전을 추가하면 CICSplex SM은 이 리전에도 CICS 번들을 추가합니다.

애플리케이션과 설치된 각 CICS 번들 간의 관계는 관리 파트에 저장됩니다. 관리 파트는 애플리케이션 설치 프로세스 중에 각 CICS 번들마다 자동으로 작성되는 MGMTPART 레코드입니다. 관리 파트는 번들이 설치된 CICS 리전을 기록하고 CICS 리전에서 번들의 상태를 추적합니다.

애플리케이션은 초기에 사용 안함 상태로 설치됩니다. 애플리케이션을 사용 가능하게 설정하면 CICSplex SM은 모든 CICS 리전에서 해당 애플리케이션에 대해 설치된 CICS 번들을 사용 가능하게 설정합니다. 애플리케이션을 사용 가능하게 설정한 후 플랫폼에서 CICS 리전을 시작 또는 다시 시작하면 CICSplex SM은 사용 상태의 이 리전에 번들을 설치합니다.

애플리케이션을 사용 가능하게 할 경우 CICS는 호출자가 애플리케이션 시작점으로 선언된 CICS 자원을 통해 애플리케이션에 액세스하도록 허용합니다. 호출자는 가용성이 가장 높은 애플리케이션 버전에 액세스하거나 EXEC CICS INVOKE APPLICATION 명령을 사용하여 사용 가능한 특정 애플리케이션 버전에 액세스할 수 있습니다. 애플리케이션을 사용 가능하게 설정한 후 플랫폼에서 CICS 리전을 시작 또는 다시 시작하는

경우 애플리케이션의 가용성 상태가 복원됩니다.

플랫폼과 애플리케이션의 클라우드 인에이블먼트 시작하기

CICS 애플리케이션의 클라우드 인에이블먼트를 진행하려는 경우 여기에 설명된 단계 접근법을 사용하여 플랫폼을 설정하고, 애플리케이션을 플랫폼에 전개하고, 정책 및 애플리케이션 컨텍스트 데이터를 사용하여 애플리케이션을 관리하고 모니터링한 후 라이프사이클이 끝나면 애플리케이션을 제거할 수 있습니다.

이 프로세스에 설명된 워크플로우는 기존 애플리케이션을 클라우드 인에이블먼트로 이동시키기 위해 플랫폼, 애플리케이션, 정책을 점진적으로 구현하는 채택 로드맵입니다. 플랫폼에 전개할 새 애플리케이션을 개발 중이면 CICS Explorer 제품 문서의 애플리케이션 전개에 대한 워크플로우에 설명된 워크플로우를 따르십시오. 이 절차는 CICS 클라우드 퍼스펙티브를 사용하여 플랫폼을 설정하고 적합한 애플리케이션과 정책을 작성하는 방법을 설명합니다.

CICS에 제공된 샘플 애플리케이션 중 하나를 사용하여 전체 프로세스를 먼저 시도해 보려면 시나리오: 플랫폼에 애플리케이션 전개를 따를 수 있습니다.

- 『단계 1: 플랫폼 작성』
- 136 페이지의 『단계 2: 애플리케이션 및 애플리케이션 시작점 작성』
- 138 페이지의 『단계 3: 정책 추가』
- 140 페이지의 『단계 4: 애플리케이션의 자원 추가』
- 142 페이지의 『단계 5: 플랫폼에서 애플리케이션 제거』

단계 1: 플랫폼 작성

다중 CICS 리전에서 CICS 애플리케이션을 전개 및 관리하려면 플랫폼을 사용합니다. 플랫폼은 애플리케이션에 서비스를 제공합니다. 클라우드 인에이블먼트의 이 단계에서는 플랫폼을 작성할 수 있습니다. 플랫폼은 기존 CICS 애플리케이션이 설치된 기존 CICS 리전을 채택할 수 있습니다. 플랫폼에 새 리전 유형을 작성해서 CICS 애플리케이션을 설치할 새 CICS 리전을 포함시킬 수도 있습니다. 플랫폼을 작성하려면 CICS 리전을 관리할 CICSplex SM을 사용하고 있어야 합니다(CICSplex SM 시작하기 참조).

플랫폼 작성은 플랫폼을 작성하고 CICSplex에서 전개하는 프로세스를 보여줍니다.

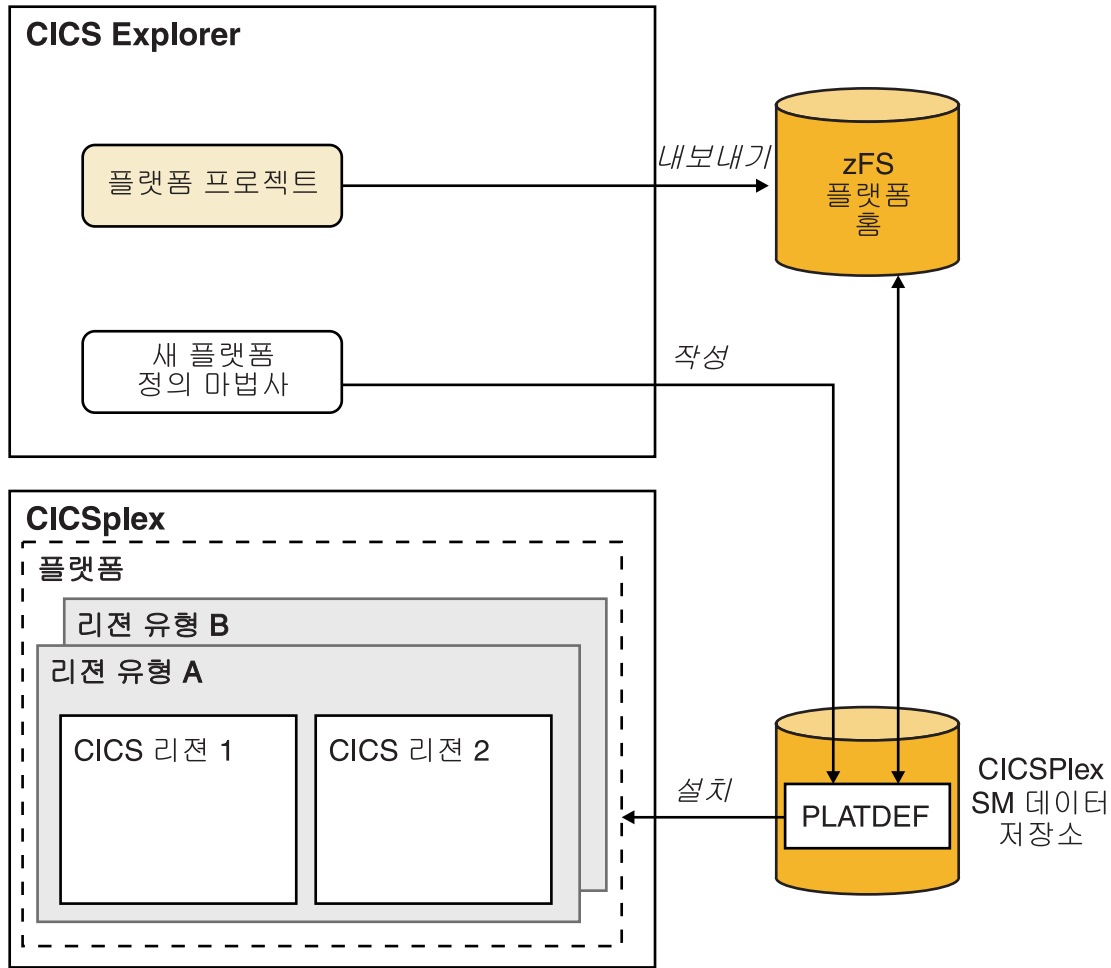


그림 14. 단계 1: 플랫폼 작성

CICS Explorer 또는 CICS Explorer SDK에서 CICS 클라우드 퍼스펙티브의 프로젝트 탐색기 보기를 사용하여 CICS 플랫폼 프로젝트를 작성한 후 z/OS UNIX 파일 시스템으로 내보내십시오. CICS 플랫폼 프로젝트는 플랫폼 및 리전 유형을 설명하는 관리 번들인 플랫폼 번들을 정의하며, 플랫폼 레벨에 전개된 CICS 번들을 참조합니다. 리전 유형을 정의하고 새 CICS 리전 정의를 설정하는 작성한 리전 유형 및 기존 CICS 리전과 함께 기존 CICS 시스템 그룹(CSYSGRP)을 리전 유형으로 채택하는 채택한 리전 유형을 혼합해서 플랫폼에 포함시킬 수 있습니다. CICS Explorer의 플랫폼 편집기를 사용하여 플랫폼 번들에 대해 작업하고 프로젝트를 내보내기 전에 리전 유형을 확인하십시오.

CICS 플랫폼 프로젝트를 내보낸 후 CICS Explorer를 사용하여 PLATDEF 자원 정의인 플랫폼 정의를 작성하고 설치하십시오. 플랫폼 정의를 설치할 때 CICS가 CICSplex에 플랫폼을 작성합니다. 설치된 활성 플랫폼에서 개별 CICS 리전을 리전 유형에 추가하거나 제거할 수 있습니다.

플랫폼 설정에 플랫폼 설계 및 작성 방법이 설명되어 있습니다. CICS 플랫폼 프로젝트 및 플랫폼 정의 작성에 대한 지시사항은 CICS Explorer 제품 문서의 플랫폼에 대한 작업의 내용을 참조하십시오.

단계 2: 애플리케이션 및 애플리케이션 시작점 작성

CICS 애플리케이션을 애플리케이션 번들에 패키징할 때 플랫폼의 다중 CICS 리전에 서 단일 엔티티로 애플리케이션을 전개, 관리, 모니터링할 수 있습니다. 애플리케이션 시작점은 애플리케이션에 대한 액세스 지점인 자원을 식별하고 플랫폼에 전개된 여러 버전의 애플리케이션에 대한 사용자의 액세스를 제어합니다. 클라우드 인에이블먼트의 이 단계에서는, 기존 CICS 애플리케이션을 플랫폼에 전개하도록 패키징하고 애플리케이션을 시작할 수 있는 각 방식에 대한 애플리케이션 시작점을 정의합니다.

그림 15는 애플리케이션 작성 프로세스를 보여줍니다.

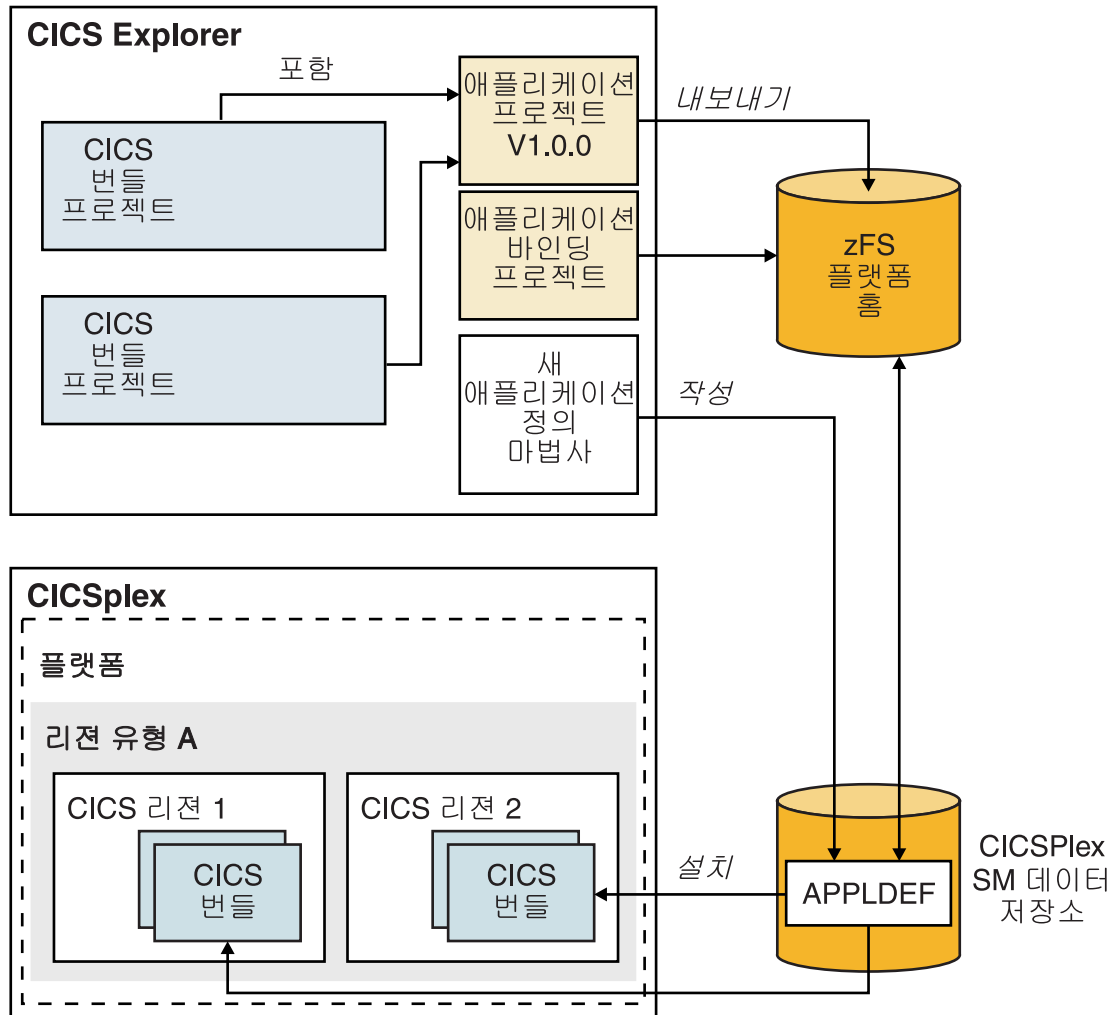


그림 15. 단계 2: 애플리케이션 작성

CICS Explorer를 사용하여 애플리케이션의 각 구성요소 자원을 포함할 하나 이상의 CICS 번들을 작성하십시오. CICS 번들은 실행할 애플리케이션의 CICS 리전에서 사용 가능해야 하는 가져오기라고도 하는, 번들 외부의 다른 자원에 대한 종속성을 선언하기도 합니다. 이 단계에서는, 기존 CICS 애플리케이션의 설치된 모든 자원을 가져오기로 지정할 수 있습니다. 단계 후반에, 번들이 설치될 때 자원이 작성되도록 번들을 수정할 수 있습니다.

CICS 번들에서 애플리케이션에 액세스하는 데 사용되는 PROGRAM 또는 TRANSACTION 자원 또는 애플리케이션에 액세스할 요청자에 대한 URI를 제공하는 URIMAP 자원의 애플리케이션 시작점을 추가하십시오. 애플리케이션 시작점으로 선언된 자원은 환경의 자원 유형에 고유 이름이 있어야 합니다. PROGRAM 자원의 경우 애플리케이션 시작점에 대한 자원이 애플리케이션 시작점 문장과 동일한 CICS 번들에 있지 않아도 됩니다.

URIMAP 또는 TRANSACTION 자원의 경우에는 애플리케이션 시작점을 통해 URIMAP 또는 TRANSACTION 자원이 제공하는 서비스에 대한 액세스를 제어하려면 URIMAP 또는 TRANSACTION 자원을 애플리케이션 시작점과 동일한 CICS 번들에 정의해야 합니다. 클라우드 인에이블먼트의 다음 단계에서 CICS 번들에 URIMAP 또는 TRANSACTION 자원을 정의함을 상기시킨다는 점에서 우선은, 애플리케이션 시작점과 동일한 CICS 번들에 URIMAP 또는 TRANSACTION 자원을 종속성으로 선언하십시오.

애플리케이션 구성요소의 CICS 번들 작성 및 애플리케이션 시작점 정의에 대한 지시 사항은 CICS Explorer 제품 문서의 번들에 대한 작업의 내용을 참조하십시오. CICS 번들 프로젝트는 애플리케이션 프로젝트에 포함할 버전 번호 및 ID가 있어야 합니다. CICS Transaction Server 버전 5 릴리스 1 이전의 릴리스(ID 및 버전 번호가 없음)에서 CICS 번들을 사용하려면 프로젝트의 번들 Manifest 파일을 갱신해서 이 정보를 추가하십시오.

적합한 CICS 번들을 작성하고 애플리케이션 시작점을 선언했다면 CICS Explorer 또는 CICS Explorer SDK에서 CICS 클라우드 퍼스펙티브의 프로젝트 탐색기 보기를 사용하여 CICS 애플리케이션 프로젝트를 작성하십시오. CICS 애플리케이션 프로젝트는 애플리케이션의 모든 CICS 번들을 참조하는 애플리케이션 번들을 정의합니다.

CICS 애플리케이션 바인딩 프로젝트에서 애플리케이션 바인딩을 작성하여 애플리케이션의 CICS 번들을 대상 플랫폼의 리전 유형으로 매핑하십시오. 애플리케이션의 구조에 따라 애플리케이션 바인딩을 사용하여 추가 CICS 번들을 전개해서 대상 플랫폼의 애플리케이션 작동을 사용자 정의할 수 있습니다. 또 다른 대안으로, CICS 번들을 설치된 플랫폼에 추가할 수 있습니다.

애플리케이션을 실행하려는 플랫폼에 대한 zFS의 플랫폼 홈 디렉토리에 애플리케이션 번들 및 애플리케이션 바인딩을 내보낸 후 애플리케이션 정의를 작성하십시오. APPLDEF

자원 정의는 대상 플랫폼에 대한 플랫폼 홈 디렉토리의 애플리케이션 바인딩 및 애플리케이션 번들을 가리킵니다. 애플리케이션 정의를 설치할 때 CICS는 CICS 번들을 플랫폼의 CICS 리전에 설치합니다. 애플리케이션이 설치를 검증할 수 있게 한 후 CICS 리전에서 사용 가능하게 하십시오. 애플리케이션을 사용 가능하게 하면 호출자가 애플리케이션 시작점 및 시작점이 애플리케이션에 대해 제어하는 자원을 사용할 수 있게 됩니다.

애플리케이션이 애플리케이션 시작점을 통해 액세스되면 CICS 리전은 애플리케이션에 관련된 태스크에 대한 애플리케이션 컨텍스트 데이터를 생성합니다. 애플리케이션 컨텍스트 데이터는 플랫폼, 애플리케이션, 애플리케이션 버전, 운영을 식별합니다. 이 정보를 사용하여 애플리케이션(또는 애플리케이션으로의 특정 경로)의 자원 이용을 측정하고, 애플리케이션의 정책 기반 관리를 사용하며, 문제점 진단을 지원할 특정 애플리케이션에 태스크를 관련시킬 수 있습니다. 애플리케이션 컨텍스트에 대한 자세한 정보는 제품 개요의 애플리케이션 컨텍스트의 내용을 참조하십시오.

전개를 위한 애플리케이션 패키징에 애플리케이션을 애플리케이션 번들로 패키징하는 방법이 설명되어 있습니다. 애플리케이션 번들, 애플리케이션 바인딩, 애플리케이션 정의 작성에 대한 지시사항은 CICS Explorer 제품 문서의 애플리케이션에 대한 작업의 내용을 참조하십시오.

단계 3: 정책 추가

정책은 하나 이상의 애플리케이션 태스크에 적용할 수 있는 제어나 조치를 설명하는 정책 규칙을 포함한 XML 정의입니다. 정책 규칙은 사용자 태스크의 작동을 관리할 임계값 조건을 정의합니다. 예를 들어, 사용자 태스크가 할당한 저장영역의 크기에 대한 임계값을 정의하는 규칙을 작성해서 임계값을 초과할 경우 CICS가 메시지를 발행하도록 할 수 있습니다. 클라우드 인에이블먼트의 이 단계에서는, 애플리케이션에 관련된 태스크를 관리하는 정책을 작성하고 전개할 수 있습니다.

139 페이지의 그림 16은 정책 추가 프로세스를 보여줍니다.

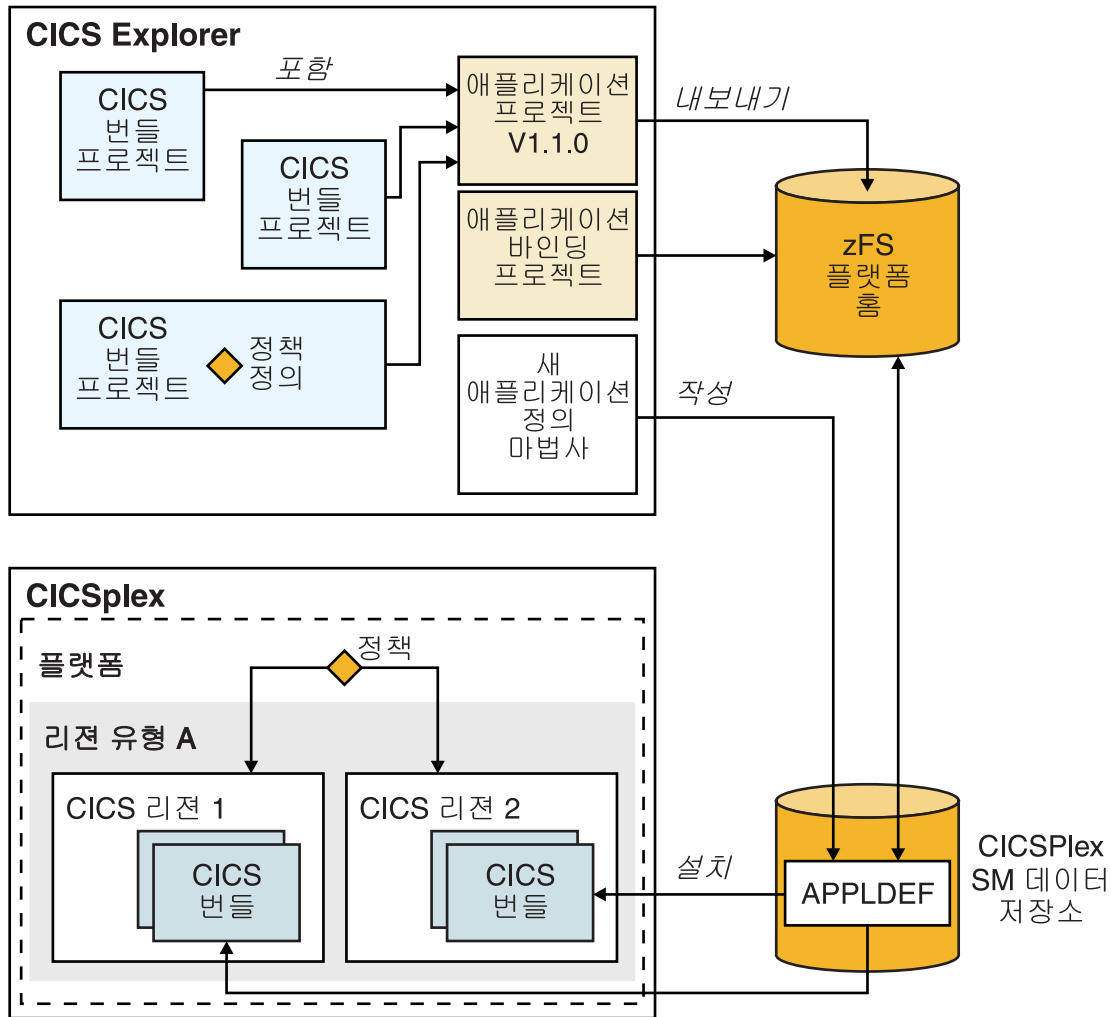


그림 16. 단계 3: 정책 추가

CICS 번들에서 기존 CICS 애플리케이션에 적합한 정책 규칙으로 정책을 정의하십시오. 정책을 포함한 CICS 번들을 애플리케이션 번들에 추가하거나, 프로덕션 CICS 리전과 같은 특정 환경을 정책의 대상으로 지정하려면 애플리케이션 바인딩에 대신 정책을 추가하십시오. 플랫폼의 모든 애플리케이션에 정책을 적용하려는 경우에는 플랫폼에 직접 정책을 추가하십시오.

애플리케이션 번들 또는 애플리케이션 바인딩과 함께 정책을 배치한 경우 애플리케이션과 애플리케이션 바인딩에 새 버전 번호를 갱신 및 적용하고, 애플리케이션의 새 버전을 플랫폼에 설치하십시오. 이제 애플리케이션에 대한 애플리케이션 컨텍스트가 있는 사용자 태스크 즉, 애플리케이션 시작점을 작성한 태스크에 정책이 적용됩니다.

정책 작성에 대한 지시사항은 CICS Explorer 제품 문서의 정책에 대한 작업의 내용을 참조하십시오.

단계 4: 애플리케이션의 자원 추가

클라우드 인에이블먼트의 이전 단계를 완료했다면 기존 CICS 애플리케이션이 설치된 CICS 리전이 플랫폼의 일부로 정의되었고, 기존 CICS 애플리케이션은 정책을 통해 관리되는 태스크와 함께 플랫폼의 애플리케이션으로 정의되었습니다. 하지만 애플리케이션의 자원은 이전과 동일한 방식으로 CICS 리전에서 정의되고 관리되었습니다. 이 단계에서는 자원 작성, 설치, 관리의 책임을 플랫폼으로 이전할 수 있습니다.

CICS 번들을 사용하여 CICS 프로그램이나 트랜잭션, CICS 라이브러리, Atom 피드, XML 기반 서비스, OSGi 번들로 패키징된 Java 애플리케이션 또는 애플리케이션이나 시스템 이벤트와 같은 아티팩트의 적합한 자원을 패키징할 수 있습니다. 사용자 정의 자원만 CICS 번들에 추가할 수 있고, CICS 제공 자원(DFH로 시작)은 추가할 수 없습니다. CICS 번들 정의에 CICS 번들에 정의할 수 있는 자원 유형 및 유형을 정의한 경우의 관리 방식에 대한 정보가 있습니다.

CICS 번들의 자원 정의는 자원의 관리 및 라이프사이클이 자원을 설치한 관리 번들과 CICS 번들에 위임되기 때문에 중요한 단계입니다. CICS 번들 및 애플리케이션에 대해 조작할 때 자원이 자동으로 추가, 갱신 또는 제거되기 때문에 더 이상 자원을 수정하거나 자원의 상태를 개별적으로 변경하지 않습니다. 따라서 애플리케이션 설계자는 애플리케이션의 어느 자원의 라이프사이클을 CICS 번들의 라이프사이클에 연관시킬지 주의해서 고려해야 합니다.

CICS에 정의할 자원 선택 및 배열 방법에 대해서는 번들된 자원의 특성의 정보를 검토하십시오. CICS 번들에 정의할 수 없는 자원이나 CICS 번들에 정의하지 않으려는 자원이 애플리케이션에 사용되면 계속해서 자원을 가져오기로 지정할 수 있습니다.

지원되는 자원 유형의 경우 애플리케이션의 일부로 패키징되고 설치된 CICS 번들에 자원이 정의되어 있으면 CICS 자원은 개인용입니다. 이 자원은 플랫폼에 설치된 다른 애플리케이션이나 버전 또는 CICS 리전의 다른 애플리케이션에 사용할 수 없으므로 자원 이름이 설치에서 고유할 필요가 없습니다. 개인용 자원의 관리 및 작동 변경을 이해하려면 애플리케이션 버전의 개인용 자원의 정보를 검토하십시오. 지원되는 자원 유형의 자원을 개인용으로 사용하지 않으려면 계속해서 이 자원을 가져오기로 지정하십시오.

정책 및 애플리케이션 시작점과 같은 다른 자원과 함께 개인용 자원으로 지원되는 CICS 자원을 사용하는 애플리케이션이 다중 버전화에 적합합니다. 애플리케이션의 여러 버전을 동시에 동일한 플랫폼 인스턴스에 설치하고 서비스 중단 없이 사용자가 새 버전을 사용하게 할 수 있습니다. 애플리케이션의 서로 다른 버전 사이의 자원 이름 충돌을 피하기 위해 자원을 적절하게 관리하는 경우 기타 자원이 다중 버전화된 애플리케이션에 포함될 수 있습니다. 다중 버전화에 적합한 애플리케이션 구조화의 이점을 보려면 플랫폼에 전개된 애플리케이션의 다중 버전화의 정보를 검토하십시오.

그림 17은 애플리케이션의 자원 추가 프로세스를 보여줍니다.

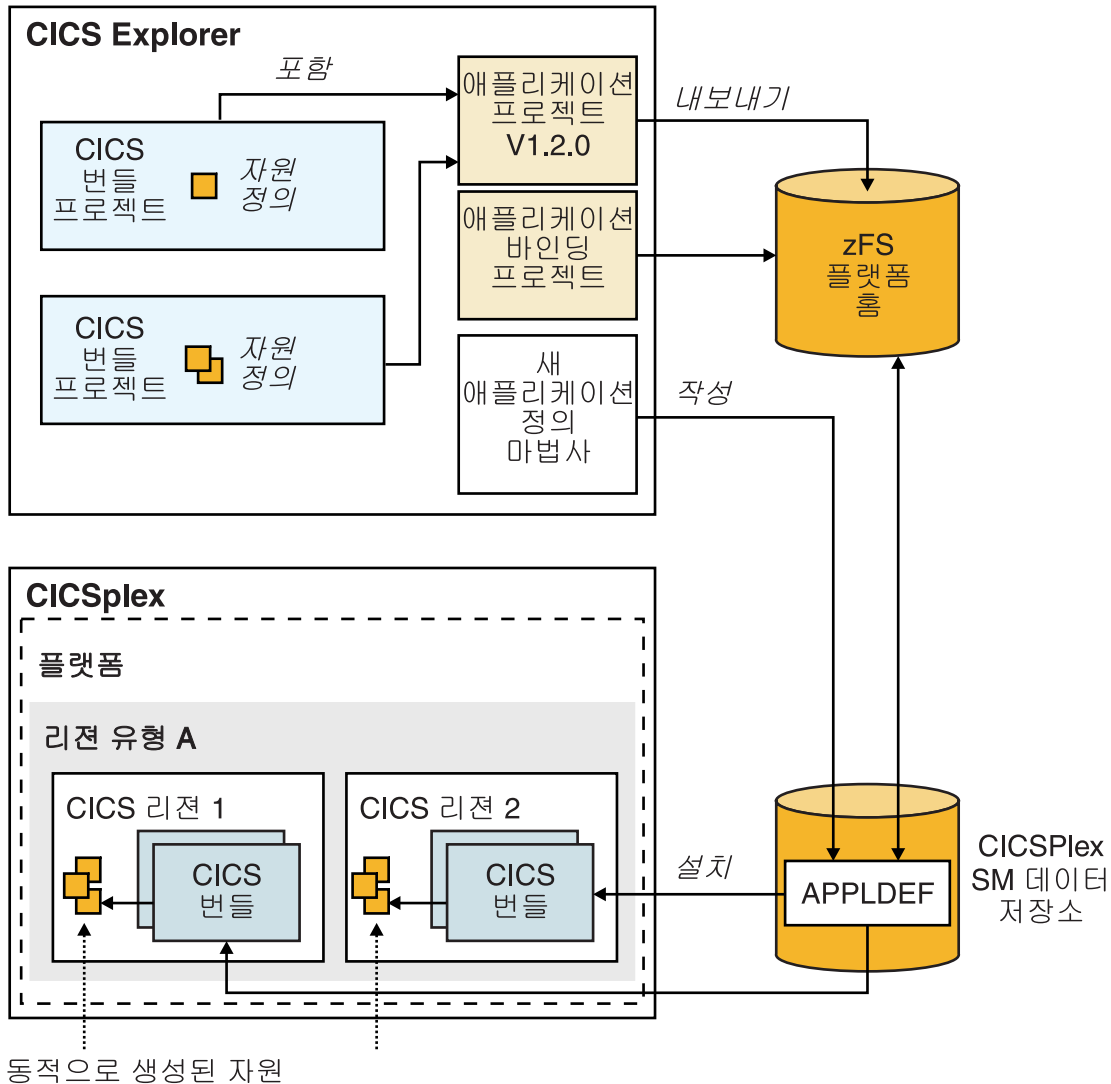


그림 17. 단계 4: 애플리케이션의 자원 추가

애플리케이션의 CICS 번들을 갱신해서 플랫폼에 애플리케이션을 설치할 때 CICS 리전에 동적으로 작성될 적합한 자원을 추가하십시오. URIMAP 자원의 경우에는 애플리케이션 시작점을 통해 URIMAP 자원이 제공하는 서비스에 대한 액세스를 제어하려면 URIMAP 자원을 관련된 애플리케이션 시작점과 동일한 CICS 번들에 정의해야 함을 기억하십시오.

CICS 번들의 새 버전으로 애플리케이션을 갱신하고, 애플리케이션 및 애플리케이션 바인딩에 새 버전 번호를 적용한 후 플랫폼에 애플리케이션의 새 버전을 설치하십시오. 애플리케이션을 설치할 때 CICS 번들에 정의한 자원이 CICS 리전에 동적으로 작성됩니다. 이 자원은 이제 CICS 번들을 통해 관리해야 합니다. 플랫폼에 있는 CICS 리전을

시작 또는 재시작할 때 CICS가 애플리케이션 전개를 자동으로 관리합니다. CICS 탐색기를 사용하여 설치된 애플리케이션과 개인용 자원을 볼 수 있습니다.

번들에 자원을 추가하는 방법에 대한 지시사항은 CICS Explorer 제품 문서의 번들에 대한 작업의 내용을 참조하십시오.

단계 5: 플랫폼에서 애플리케이션 제거

기존 CICS 애플리케이션이 더 이상 필요하지 않은 경우 플랫폼에서 애플리케이션을 제거해서 라이프사이클을 완료할 수 있습니다. 다시 필요할 것 같지 않은 애플리케이션의 개별 버전을 제거할 수도 있습니다. 애플리케이션의 버전을 사용 불가능하게 하고, 사용 안함으로 설정하고, 버리면 애플리케이션 버전의 일부로 전개된 CICS 번들도 바인드된 CICS 리전 유형에서 사용 불가능하게 되고, 사용 안함으로 설정되고, 버려집니다.

제거하려는 각 애플리케이션 버전마다 CICS Explorer 또는 CICS Explorer SDK에서 CICS 클라우드 퍼스펙티브의 클라우드 탐색기 보기를 사용하여 애플리케이션 버전을 사용 불가능하게 하십시오. 그런 다음 클라우드 탐색기 보기를 사용하여 애플리케이션 버전을 사용 안함으로 설정한 후 버리십시오. 또는 클라우드 탐색기 보기에서 애플리케이션 이름을 두 번 클릭하여 애플리케이션을 열고 사용 안함 및 버리기 옵션을 선택해서 애플리케이션 버전을 버리십시오.

애플리케이션 버전이 다시 필요하지 않으면 데이터 저장소에서 애플리케이션 정의를 제거할 수 있습니다. CICS Explorer에서 z/OS 퍼스펙티브를 사용하여 애플리케이션 번들을 포함한 버전화된 디렉토리를 zFS의 플랫폼 홈 디렉토리에서 제거할 수도 있습니다.

관리에서 플랫폼, 애플리케이션 및 정책 관리에 애플리케이션을 사용 불가능하게 하고, 사용 안함으로 설정하며, 플랫폼에서 버리는 방법에 대한 지시사항이 있습니다.

제 9 장 애플리케이션 설치

애플리케이션은 일반적으로 CICS 명령 레벨 애플리케이션 프로그래밍 인터페이스(API)를 사용하는 사용자 프로그램을 의미합니다. CICS에서 실행할 애플리케이션을 설치하려면 소스문을 변환 및 컴파일하고 결과 오브젝트 모듈을 CICS 라이브러리로 링크 편집하고 프로그램을 CICS에 자원으로 정의해야 합니다.

프로시저

1. 컴파일러가 CICS 명령을 변환하지 않는 경우 프로그램 소스 코드를 변환하여 CICS 명령을 컴파일러가 이해하는 호출로 바꾸어야 합니다.
 - a. 프로그램이 CICS 명령을 사용하지 않고 실행 중인 트랜잭션에 의해서만 호출되는 경우(CICS 태스크 시동에 의해 직접 호출되지 않음) 변환기 단계가 필요하지 않습니다.
 - b. 또한 DL/I CALL 또는 EXEC DLI 인터페이스를 통해 DL/I 서비스에 액세스하는 CICS 명령 레벨 프로그램도 변환해야 합니다. EXEC SQL 인터페이스를 사용하여 DB2 서비스에 액세스하는 애플리케이션에는 추가 사전 컴파일 단계가 필요합니다.
2. 프로그램 소스를 컴파일하여 오브젝트 코드를 생성하십시오.
3. 오브젝트 모듈을 링크 편집하여 로드 모듈을 생성하십시오. 로드 모듈은 DFHRPL 또는 동적 LIBRARY 연결에 있는 애플리케이션 로드 라이브러리에 저장됩니다. EXEC SQL 인터페이스를 사용하여 DB2 서비스에 액세스하는 애플리케이션에 대한 추가 INCLUDE문이 필요합니다.
4. 프로그램을 호출하는 트랜잭션에 대한 자원 정의를 작성하고 설치하십시오.
5. 다음 방법 중 하나를 사용하여 CICS에 프로그램을 자원으로 정의하십시오.
 - 단일 CICS 리전에 대한 자원 정의를 CSD에 작성하십시오. 자원 정의 개요의 내용을 참조하십시오.
 - 프로그램이 로드되면 CICS가 동적으로 단일 CICS 리전에서 자원 정의를 생성할 수 있도록 프로그램 자동 설치를 사용하십시오. 프로그램, 맵핑 세트 및 파티션 세트 자동 설치의 내용을 참조하십시오.
 - CICSplex SM BAS(Business Application Services)를 사용하여 자원 정의를 작성하십시오. 자원 정의는 여러 CICS 리전에 설치할 수 있습니다. BAS 관리의 내용을 참조하십시오.
 - z System z[®]용 Rational[®] Developer 또는 IBM CICS 탐색기를 사용하여 독립형 CICS 번들에서 자원 정의를 작성하십시오. CICS 번들은 여러 CICS 리전에 설치할 수 있습니다. CICS 번들 정의의 내용을 참조하십시오.

- CICS 번들 및 패키지에서 자원 정의를 작성하고 CICS 번들을 플랫폼에 전개되는 애플리케이션의 일부로 설치하십시오. 애플리케이션은 여러 CICS 리전에 설치할 수 있고 동시에 여러 버전의 애플리케이션을 설치할 수 있습니다. 애플리케이션을 플랫폼에 전개하는 내용을 참조하십시오.

프로그램 설치 단계

CICS에서 실행할 애플리케이션을 설치하기 위해 수행해야 하는 수많은 단계가 있습니다.

이 태스크 정보

단계는 다음과 같습니다.

프로시저

1. 컴파일러가 CICS 명령을 변환하지 않는 경우 프로그램 소스 코드를 변환하여 CICS 명령을 컴파일러가 이해하는 호출로 바꿔야 합니다.
 - a. 프로그램이 CICS 명령을 사용하지 않고 실행 중인 트랜잭션에 의해서만 호출되는 경우(CICS 태스크 시동에 의해 직접 호출되지 않음) 변환기 단계가 필요하지 않습니다.
 - b. 또한 DL/I CALL 또는 EXEC DLI 인터페이스를 통해 DL/I 서비스에 액세스하는 CICS 명령 레벨 프로그램도 변환해야 합니다. EXEC SQL 인터페이스를 사용하여 DB2 서비스에 액세스하는 애플리케이션에는 추가 사전 컴파일 단계가 필요합니다.
2. 프로그램 소스를 컴파일하여 오브젝트 코드를 생성하십시오.
3. 오브젝트 모듈을 링크 편집하여 로드 모듈을 생성하십시오. 로드 모듈은 DFHRPL 또는 동적 LIBRARY 연결에 있는 애플리케이션 로드 라이브러리에 저장됩니다. EXEC SQL 인터페이스를 사용하여 DB2 서비스에 액세스하는 애플리케이션에 대한 추가 INCLUDE문이 필요합니다.
4. 프로그램을 호출하는 트랜잭션에 대한 자원 정의를 작성하고 설치하십시오.
5. 다음 방법 중 하나를 사용하여 프로그램의 CSD에서 자원 정의 항목을 작성하십시오.
 - 프로그램 자동 설치 사용
 - RDO 사용

동적 프로그램 LIBRARY 자원 사용

애플리케이션을 실행하려면 로드 모듈이 CICS 로드 LIBRARY 연결의 데이터 세트에 상주해야 합니다.

CICS에는 다음 두 가지 유형의 로드 LIBRARY 연결이 있습니다.

- 정적 로드 LIBRARY 연결: DFHRPL
- 하나 이상의 동적으로 정의된 로드 LIBRARY 연결

정적 LIBRARY 연결 DFHRPL

시동 JCL은 CICS에 정적 로드 LIBRARY 연결, DFHRPL을 정의합니다. DFHRPL은 CICS를 시작하고 실행하는 데 사용할 수 있어야 하는 중요한 데이터 세트와 애플리케이션 엔티티를 포함합니다. CICS가 실행 중이면 CICS를 중지했다가 재시작해야 DFHRPL 데이터 세트 이름을 변경할 수 있습니다. 이러한 변경은 오늘날의 연속 가용성 환경에서는 일반적으로 옵션이 아닙니다.

DFHRPL 데이터 세트 이름은 MVS 데이터 이름 지정 규칙을 준수해야 합니다.

동적 프로그램 LIBRARY 연결의 데이터 세트는 확장 주소 블록(EAV) DASD 블록의 확장 주소 지정 공간(EAS)에 상주할 수 있습니다.

동적 프로그램 LIBRARY 연결

CICS에 프로그램 LIBRARY 연결을 동적으로 정의할 수 있습니다. 동적 프로그램 LIBRARY 연결 사용은 시스템 프로그래머와 조직에 다음과 같은 많은 장점을 제공합니다.

- 프로그램 아티팩트를 로드할 수 있는 하나 이상의 데이터 세트를 포함합니다.
- 연속 가용성에 영향을 주지 않고 언제든지 전개할 새 애플리케이션을 가져와 서비스할 수 있습니다.
- 연속 가용성에 영향을 주지 않고 동적 LIBRARY 연결의 기존 애플리케이션 서비스를 중단할 수 있습니다.
- 기존 LIBRARY보다 순위가 높은 LIBRARY 연결에 설치하여 연속 가용성에 영향을 주지 않고 기존 애플리케이션에 패치를 설치할 수 있습니다.
- 연속 가용성에 영향을 주지 않고 압축을 위해 동적 LIBRARY 연결의 데이터 세트를 오프라인 상태로 할 수 있습니다.

LIBRARY 데이터 세트 이름은 MVS 데이터 세트 이름 지정 규칙을 준수해야 하며 별명 데이터 세트를 사용할 수 있습니다. 동적 프로그램 LIBRARY 연결의 데이터 세트는 확장 주소 블록(EAV) DASD 블록의 확장 주소 지정 공간(EAS)에 상주할 수 있습니다.

동적 프로그램 LIBRARY 연결을 사용하지 않아도 됩니다. DFHRPL을 사용할 수 있습니다. 실제로 DFHRPL에 다음 데이터 세트를 정의해야 합니다.

- SDFHLOAD
- 단계 1 PLT 프로그램
- 비SMS 관리 대상 데이터 세트
- DISP가 SHR이 아닌 데이터 세트

동적 LIBRARY 연결을 사용 또는 사용 안함으로 설치하거나 작성할 수 있습니다.

사용 사용 상태가 사용인 LIBRARY를 설치하거나 작성하는 경우, CICS는 마지막으로 LIBRARY 연결을 열기 전에 데이터 세트를 할당한 후 연결하려고 시도합니다. 이러한 단계가 실패하면 이미 성공한 단계가 실행 취소되고 LIBRARY는 사용 안함으로 설치됩니다. 메시지는 실패한 단계를 표시합니다.

사용 안함

사용 상태가 사용 안함인 LIBRARY를 설치하거나 작성하는 경우, CICS는 데이터 세트를 할당하거나 연결하려고 시도하지 않습니다. 데이터 세트를 사용할 수 있고 LIBRARY를 사용할 준비가 되면 SET LIBRARY ENABLED 명령을 수행하여 데이터 세트를 할당하고 연결한 후 LIBRARY를 여십시오.

SET LIBRARY ENABLED 조작의 사용 단계가 실패하면 이미 성공한 단계가 실행 취소되고 LIBRARY는 여전히 사용 안함으로 설정됩니다. 메시지는 실패한 단계를 표시합니다.

CICS Explorer 또는 z Systems용 Rational Developer를 사용하여 다중 CICS 리전에서 자원 관리 및 전개를 위해 CICS 번들에 동적 LIBRARY 연결을 정의할 수 있습니다.

다음 예제는 동적 프로그램 LIBRARY 연결을 사용하도록 도와줍니다.

동적 LIBRARY 자원 사용 예제

동적 LIBRARY 자원을 채택할지 여부를 선택합니다. 이러한 자원을 테스트 환경 또는 프로덕션 환경 또는 둘 다에서 사용할 수도 있습니다. 일부 데이터 세트를 DFHRPL 외부로 이동한 다음 동적으로 정의하고 DFHRPL과 동적 프로그램 LIBRARY 자원의 조합을 사용하기로 결정할 수 있습니다.

동적 LIBRARY에서 정의하기에 적합한 후보는 벤더 패키지 또는 한 개 이상의 데이터 세트에서 제공되는 내부 애플리케이션입니다.

다음 예제는 동적 프로그램 LIBRARY 연결을 사용하여 프로그램을 관리할 수 있는 방법을 보여줍니다.

CICS 시스템에 응급 수정사항 적용

프로그램 수정사항이 포함된 임시 LIBRARY를 CICS 리전에 설치하십시오.

이 태스크 정보

- CICS 시스템이 사용 중인 애플리케이션 버전에 정정해야 하는 문제점이 있습니다.
- 갱신된 버전의 애플리케이션이 작성되었습니다.
- 지금 CICS를 재시작하여 수정사항을 적용할 수 없습니다.
- 이 조치를 수행하는 사용자에게 적절한 액세스 권한이 있습니다.

프로시저

1. PDS나 PDSE 데이터 세트 또는 데이터 세트의 세트에 수정사항을 제공하는 프로그램과 기타 아티팩트를 추가하십시오.
2. 수정사항이 포함된 데이터 세트를 포함하는 LIBRARY 자원을 정의하십시오. LIBRARY 자원은 검색 순서에서 프로그램의 실패 버전이 포함된 LIBRARY보다 우선하는 순위를 가져야 합니다. 이는 검색 순서에서 DFHRPL 앞에 LIBRARY를 배치함을 의미합니다. 다음 방법 중 하나를 사용하여 LIBRARY 자원을 정의하십시오.

- CICSplex SM의 BAS(Business Application Services) 구성요소
- CICS 자원 관리 트랜잭션 CEDA
- CICS 오프라인 CSD 유틸리티 프로그램 DFHCSDUP
- EXEC CICS CREATE LIBRARY 명령
- CICS Explorer

CICS Explorer를 사용하여 단일 CICS 리전이나 CICSplex용 데이터 저장소 또는 CICS 번들에 LIBRARY 자원을 정의할 수 있습니다. CICS 번들에 정의된 자원 관리에 대한 정보는 구성에서 번들된 자원의 특성의 내용을 참조하십시오.

3. EXEC CICS CREATE를 사용하여 LIBRARY를 정의하지 않은 경우 CEDA, CICSplex SM WUI 또는 CICS Explorer를 사용하여 새 LIBRARY 자원을 설치하십시오.
4. 영향받는 프로그램에 대해 EXEC CICS SET PROGRAM NEWCOPY나 EXEC CICS SET PROGRAM PHASEIN 명령 또는 CICSplex SM이나 CEMT의 해당 명령어를 실행하십시오.

결과

CICS 시스템이 계속 실행되며 정정된 버전의 애플리케이션이 이제 검색 순서에서 문제 버전 앞에 있으므로 수정된 버전이 대신 사용됩니다.

CICS 시스템에 새 애플리케이션 설치

하나 이상의 데이터 세트에 제공된 새 애플리케이션을 연속 가용성에 영향을 주지 않고 실행 중인 CICS 시스템에 도입합니다.

이 태스크 정보

- 애플리케이션이 하나 이상의 PDS 또는 PDSE 데이터 세트에 제공되었습니다. 애플리케이션은 하나 이상의 PDS 또는 PDSE 데이터 세트에 애플리케이션 아티팩트 세트에 제공된 써드 파티(벤더) 제품이거나 새 사내 애플리케이션입니다.
- 지금 CICS를 재시작하여 수정사항을 적용할 수 없습니다.
- 조치를 수행하는 사용자에게 적절한 액세스 권한이 있습니다.

프로시저

1. 새 애플리케이션이 포함된 데이터 세트를 포함하는 LIBRARY 자원을 정의하십시오. 일반적으로 애플리케이션은 기존 LIBRARY 자원과 교차하지 않으며 기본 순위 값을 사용할 수 있습니다. 다음 방법 중 하나를 사용하여 LIBRARY 자원을 정의하십시오.

- CICSplex SM의 BAS(Business Application Services) 구성요소
- CICS 자원 관리 트랜잭션 CEDA
- CICS 오프라인 CSD 유틸리티 프로그램 DFHCSDUP
- EXEC CICS CREATE LIBRARY 명령
- CICS Explorer

CICS Explorer를 사용하여 단일 CICS 리전이나 CICSplex용 데이터 저장소 또는 CICS 번들에 LIBRARY 자원을 정의할 수 있습니다. CICS 번들에 정의된 자원 관리에 대한 정보는 구성에서 번들된 자원의 특성의 내용을 참조하십시오.

2. EXEC CICS CREATE를 사용하여 LIBRARY를 정의하지 않은 경우 CEDA, CICSplex SM WUI 또는 CICS Explorer를 사용하여 새 LIBRARY 자원을 설치하십시오.
3. CICS에 이를 참조하는 애플리케이션 및 트랜잭션 정의를 구성하는 프로그램, 맵핑 세트를 정의하십시오.
4. 프로그램과 기타 정의를 설치하십시오.

결과

새 애플리케이션이 CICS 프로덕션 시스템에 설치되고 연속 가용성이 유지보수됩니다.

CICS 시스템 세트에 새 애플리케이션 설치

하나 이상의 데이터 세트에 제공된 새 애플리케이션을 CICSplex의 CICS 시스템 세트에 도입합니다. 이러한 시스템은 테스트 또는 개발 CICSplex에도 있을 수 있지만 프로덕션에 있을 가능성이 높습니다.

이 태스크 정보

- 애플리케이션이 하나 이상의 PDS 또는 PDSE 데이터 세트에 제공되었습니다.
- 애플리케이션이 다중 CICS 시스템에 동시에 도입됩니다.
- 지금 CICS 리전을 재시작하여 새 애플리케이션을 추가할 수 없거나 애플리케이션이 CICS 실행에 중요하지 않습니다.
- 이 조치를 수행하는 사용자에게 적절한 액세스 권한이 있습니다.

프로시저

1. CICSplex SM BAS를 사용하여 애플리케이션 데이터 세트가 포함된 CICSplex SM LIBRARY 정의(LIBDEF)를 정의하십시오.
2. CICS 리전에서 사용 중인 다른 LIBRARY 자원에 상대적인 순서 지정을 반영하는 순위를 LIBRARY에 지정하십시오. 일반적으로 애플리케이션은 기존 LIBRARY 자원과 교차하지 않으며 기본 순위 값을 사용할 수 있습니다.
3. 새 LIBDEF를 설치하고 CICS 시스템 세트를 포함하는 대상 범위를 지정하십시오.
4. CICS에 이를 참조하는 애플리케이션 및 트랜잭션 정의를 구성하는 프로그램, 맵핑 세트, 기타 아티팩트를 정의하십시오.
5. CICS 시스템 세트에 프로그램과 기타 정의를 설치하고 이를 사용하도록 시작하십시오.

결과

CICS 리전이 새 애플리케이션으로 실행됩니다.

LIBRARY의 CICS 애플리케이션 재구성

추적 및 관리가 용이하도록 LIBRARY의 애플리케이션을 재구성할 수 있습니다. LIBRARY에 저장된 애플리케이션은 DFHRPL 연결에 DD 카드 데이터 세트로 정의됩니다.

시작하기 전에

필수 액세스 권한이 있는지 확인하십시오.

이 태스크 정보

이 태스크의 목적은 애플리케이션 구성을 LIBRARY에 재구성하는 것이므로 데이터 세트 이름은 운영 적합성이 아니라 이름에 포함된 애플리케이션과 관련됩니다.

프로시저

1. LIBRARY 데이터 세트에 애플리케이션 할당 방법을 결정하고 각 애플리케이션에 단일 LIBRARY를 사용할지 각 LIBRARY에 다중 애플리케이션을 사용할지 여부를 결정하십시오. 각 LIBRARY에 단일 애플리케이션을 사용하면 시스템 구성이 단

순해지고 유지보수가 용이합니다. LIBRARY에서 다중 데이터 세트 연결이 필요한 애플리케이션과 단일 데이터 세트가 필요한 애플리케이션도 판별해야 합니다.

2. DFHRPL에 남아 있는 애플리케이션과 동적 자원이 되는 애플리케이션을 판별하십시오.
3. 동적 LIBRARY에 정의될 애플리케이션 중에서 CICS 시동에 중요한 애플리케이션과 시동에 중요하지 않은 애플리케이션을 결정하십시오.
4. CICSplex SM, CEDA, DFHCSDUP 또는 EXEC CICS CREATE의 BAS(Business Application Services) 구성요소를 사용하여 동적 자원이 될 각 애플리케이션에 대해 또는 각 애플리케이션 세트에 대해(그룹화될 경우) LIBRARY 자원을 정의하십시오.
 - a. CICS 리전의 다른 라이브러리에 상대적인 순서 지정을 나타내는 순위를 각 LIBRARY에 지정하십시오. 일반적으로 애플리케이션은 다른 라이브러리와 교차하지 않으며 기본 순위 값을 사용할 수 있습니다.
 - b. CICS 실행에 중요한 상태를 각 LIBRARY에 지정하십시오. CICS 실행에 중요하지 않은 라이브러리에 대해서는 기본 상태인 NONCRITICAL을 유지하십시오.
 - c. LIBRARY에 데이터 세트 이름을 지정하십시오.
5. CEDA INSTALL LIBRARY 명령 또는 CICSplex SM WUI를 사용하여 새 LIBRARY 자원을 설치하십시오.
6. 다음 번 CICS 재시작 시 동적 LIBRARY 연결에 애플리케이션이 포함된 해당 데이터 세트를 DFHRPL 연결에서 제거하십시오.
7. 선택적으로 각 LIBRARY의 순위를 의도한 영구 값으로 다시 설정하십시오(테스트 목적으로 LIBRARY를 DFHRPL 앞에 배치한 값으로 순위를 설정한 경우).
8. GRPLIST를 사용하여 CICS를 재시작하는 동안, BAS 설치 중에 또는 CICS 재시작 후 새 LIBRARY 자원을 설치하십시오. 이를 수행하면 이제 DFHRPL 연결에 더 이상 없기 때문에 시스템이 새 LIBRARY 자원에서 프로그램을 로드합니다.

결과

CICS가 정상적으로 실행됩니다. 이제 LIBRARY 애플리케이션 구성이 향상되어 추적이 용이합니다. 설치된 CICS 시스템 애플리케이션을 판별하는 것도 용이합니다.

LIBRARY를 오프라인 상태로 두거나 CICS 시스템에서 애플리케이션 제거
LIBRARY를 오프라인 상태로 두거나(예: PDS 압축) 실행 중인 CICS 시스템에서 애플리케이션을 제거합니다.

이 태스크 정보

- 애플리케이션이 동적 LIBRARY 자원에서 알려진 데이터 세트 또는 데이터 세트의 세트입니다.

- 조치를 수행하는 사용자에게 적절한 액세스 권한이 있습니다.

프로시저

1. EXEC CICS SET LIBRARY 명령, CEMT, CICSplex SM WUI 또는 CICS Explorer를 사용하여 LIBRARY를 사용 안함으로 설정하십시오.
2. 애플리케이션 사용이 모두 완료되면, 로드된 프로그램 사본이 제거되게 하는 조작 (예: SET PROGRAM NEWCOPY)을 수행하거나 다음에 다시 로드할 때 실패하게 하십시오.
3. LIBRARY를 사용 안함으로 설정하는 하나의 이유는 데이터 세트를 압축한 후 LIBRARY를 다시 사용할 수 있게 하거나 LIBRARY 정의를 재설치하여 애플리케이션을 다시 사용하기 시작하기 위해서입니다. LIBRARY를 다시 사용할 수 있게 한 후에는 PROGRAM NEWCOPY 또는 PHASEIN을 실행하여 프로그램을 다시 사용하십시오.

결과

LIBRARY가 사용 안함으로 설정된 동안에는 검색 순서에서 사용 안함으로 설정된 LIBRARY 뒤에 오는 다른 LIBRARY에 사본이 있는 경우(이 경우 거기서 로드됨)가 아니면 새 사용자가 애플리케이션을 사용할 수 없습니다.

두 LIBRARY 연결 간의 전환

하나의 LIBRARY를 CICS에 도입하고 다른 LIBRARY를 오프라인 상태로 두어 새 LIBRARY의 프로그램이 로드되어 기존 LIBRARY의 해당 프로그램을 대체하도록 합니다.

이 태스크 정보

- CICS에 현재 설치된 애플리케이션을 구성하는 프로그램 또는 다양한 프로그램 아티팩트가 포함된 LIBRARY.
- 새 버전의 프로그램 또는 애플리케이션을 하나 이상의 PDS 또는 PDSE 데이터 세트에서 사용할 수 있습니다.
- 이 조치를 수행하는 사용자에게 적절한 액세스 권한이 있습니다.

프로시저

1. CICSplex SM BAS를 사용하여 새 애플리케이션 데이터 세트가 포함된 CICSplex SM LIBRARY 정의(LIBDEF)를 정의하십시오.
2. 새 LIBRARY를 설치하십시오.
3. PROGRAM NEWCOPY 또는 PHASEIN 명령을 실행하여 프로그램 새 사본 사용을 시작하십시오.
4. 기존 LIBRARY 자원을 사용 안함으로 설정하고 다시 사용할 가능성이 없으면 버리십시오.

5. 선택적으로 SET LIBRARY 명령, WUI 또는 CICSplex SM API를 사용하여 새 LIBRARY의 순위를 기존 LIBRARY의 순위로 다시 변경하십시오.

결과

CICS가 새 LIBRARY 및 새 버전의 애플리케이션으로 실행됩니다.

CICS 시스템에서 LIBRARY 자원에 대한 정보 검색

LIBRARY 자원에 대한 정보 검색

이 태스크 정보

다음과 같은 LIBRARY 자원에 대한 정보를 검색하십시오.

- CICS에 설치된 LIBRARY 자원.
- CICS에서 활성화된 LIBRARY 자원(예: 설치된 및 사용 가능한 LIBRARY 자원)을 통한 현재 검색 순서.
- 검색 순서에서 두 LIBRARY 연결의 상대적 위치.
- 중요한 LIBRARY 자원.
- LIBRARY 자원에 정의된 데이터 세트.

플랫폼에 전개된 애플리케이션의 일부로 정의 및 설치된 동적 프로그램 LIBRARY 연결은 해당 애플리케이션의 해당 버전 전용입니다. **EXEC CICS INQUIRE LIBRARY** 시스템 프로그래밍 명령을 사용하여 개인 자원을 조회하거나 찾아볼 수 있습니다. 기본적으로 CICS는 **EXEC CICS INQUIRE LIBRARY** 명령이 실행된 프로그램에 사용 가능한 자원을 검색합니다. 지정된 애플리케이션에 대해서도 개인 자원을 찾아보도록 선택할 수 있습니다. 다른 애플리케이션 컨텍스트에서 찾아보기 예제를 포함하여 개인 자원 찾아보기에 대한 자세한 정보는 시스템 프로그램 개발에서 자원 정의 찾아보기의 내용을 참조하십시오.

프로시저

- 실행 중인 CICS에서 **EXEC CICS INQUIRE LIBRARY** 명령을 실행하거나 공용 LIBRARY 자원의 경우만 CEMT 트랜잭션에서 **INQUIRE LIBRARY** 명령을 사용하십시오. LIBRARY 또는 특성을 지정하지 않을 경우 조회는 EXEC CICS INQUIRE 명령이 실행된 프로그램에 사용 가능한 설치된 LIBRARY 자원을 모두 표시합니다. LIBRARY 자원의 일부 특성을 지정하면 설치된 LIBRARY 자원 서브세트가 표시됩니다. 특정 LIBRARY 이름을 지정하면 해당 LIBRARY에 대한 세부사항이 표시됩니다. LIBRARY의 상세 보기는 연결의 데이터 세트를 표시합니다.
- CICSplex SM WUI에서는 기본 메뉴에서 **CICS 작업 보기 > 프로그램 작업 보기 > LIBRARY(DFHRPL 포함)**를 클릭하십시오. **DSNAME** 수 필드 이름을 클릭하여 LIBRARY 데이터 세트 레코드를 표시하십시오.

- CICS Explorer에서는 메뉴 표시줄에서 작업을 클릭하고 **LIBRARY**를 선택하거나 **LIBRARY DS** 이름을 선택하여 LIBRARY 데이터 세트 이름 레코드를 표시하십시오. CICS 시스템에 연결된 경우 보기에 표시된 자원은 해당 시스템의 자원입니다. CICSplex에 연결된 경우 보기에 표시된 자원은 설정한 찾아보기 범위에 따라 다릅니다.
- 플랫폼에 전개된 애플리케이션의 경우, CICS Explorer의 클라우드 탐색기 보기에서 설치된 애플리케이션을 두 번 클릭하여 온라인 애플리케이션 편집기를 여십시오. 자원 탭을 선택한 후 라이브러리 탭을 선택하여 애플리케이션의 개인 LIBRARY 자원을 보십시오. 자원이 설치된 CICS 시스템별로 자원을 필터링하거나 자원이 정의된 CICS 번들별로 필터링할 수 있습니다. 라이브러리 **DS** 이름 탭을 선택하여 z/OS가 개인 LIBRARY 자원에 대한 데이터 세트의 LIBRARY 연결에 대해 생성한 DD 이름을 보십시오.

결과

조회가 LIBRARY 자원의 중요 상태 및 사용 상태, 해당 순위를 표시하며 전체 검색 순서에서 절대 위치도 표시합니다. 사용 불가능한 LIBRARY 자원은 목록에 나타나지만 검색 순서에 참여하지 않습니다. 두 LIBRARY 연결의 검색 위치 순서를 비교하여 전체 검색 순서에서 다른 라이브러리보다 앞에 있는 라이브러리를 판별할 수 있습니다.

CICS 시스템에서 프로그램에 대한 LIBRARY 정보 검색

CICS 시스템에서 프로그램을 조회하여 프로그램이 로드된 해당 LIBRARY에서 LIBRARY와 데이터 세트를 검색합니다(예를 들어, 의도한 위치에서 로드되는지 유효성 검증).

시작하기 전에

- CICS가 실행 중입니다.
- 프로그램이 CICS 시스템에서 사용 중입니다.

프로시저

WUI 또는 CEMT INQUIRE PROGRAM 명령을 사용하여 프로그램을 조회하십시오.

결과

리턴된 프로그램 정보는 프로그램이 로드된 LIBRARY와 데이터 세트를 포함합니다.

- 설치된 LIBRARY에서 프로그램이 로드된 경우 LIBRARY 및 LIBRARYDSN 이름이 리턴됩니다.
- 사용 안함으로 설정된 LIBRARY에서 프로그램이 로드된 경우 LIBRARY 이름은 리턴되지만 LIBRARYDSN은 공백입니다.
- 버려진 LIBRARY에서 프로그램이 로드된 경우 LIBRARY와 LIBRARYDSN 둘 다 공백입니다.

- 프로그램이 로드되지 않은 경우 LIBRARY와 LIBRARYDSN 둘 다 공백입니다.
- LPA에서 프로그램이 로드된 경우 LIBRARY와 LIBRARYDSN 둘 다 공백입니다.

LIBRARY 자원의 CRITICAL 특성 수정

하나 이상의 CICS 리전의 LIBRARY가 CICS 시동에 중요함을 지정하십시오.

시작하기 전에

- CICS가 실행 중입니다.
- 하나 이상의 동적 LIBRARY가 CICS 시스템에서 활성화되어 있습니다.

프로시저

1. CICSplex SM WUI, CEMT 또는 SPI를 사용하여 설치된 LIBRARY의 중요 상태를 변경하십시오. LIBRARY 연결의 데이터 세트를 사용할 수 없거나 다른 문제점으로 인해 LIBRARY를 사용 가능으로 복구할 수 없는 경우 중요 상태가 CICS 시동이 계속해서 인터럽트되지 않는지 여부를 판별하면 이 변경이 다음 워치 또는 긴급 시작 시 적용됩니다.
2. CRITICAL 상태 영구 변경의 경우, CICSplex SM BAS 또는 CEDA를 사용하여 필수 중요 상태를 가진 LIBRARY 정의를 정의하고 CICSplex SM BAS 설치 또는 GRPLIST 설치를 사용하여 각 CICS 콜드 또는 초기 시작 시 정의를 설치하십시오.

결과

재시작 시 CICS 작동은 LIBRARY의 중요 설정과 LIBRARY의 데이터 세트 가용성에 따라 변경됩니다.

LIBRARY 구성 변경사항 추적

CICS 시스템에서 LIBRARY 구성 변경사항을 판별하려면 감사 로그를 사용하십시오.

이 태스크 정보

DFHLD0555I 및 DFHLD0556I 메시지는 LIBRARY 구성 변경사항의 감사 로그를 제공하기 위해 CSLB 트랜지언트 데이터 큐에 기록됩니다. 메시지는 검색 순서에 영향을 줄 수 있는 변경이 발생할 때마다 현재 LIBRARY 검색 순서를 표시하기 위해 발행됩니다. DFHLD0555I 메시지가 발행되고 뒤이어 검색 순서에서 활성화된 각 LIBRARY에 대해 하나의 DFHLD0556I 메시지 인스턴스가 옵니다. CICS 시스템에서 다음과 같은 LIBRARY 구성 변경사항을 판별하려면 감사 로그를 사용하십시오.

- 새 LIBRARY가 설치되었습니다.
- CICS에서 LIBRARY가 제거되었습니다(버려짐).
- LIBRARY의 순위, 중요 상태 또는 사용 상태가 변경되었습니다.
- 전체 LIBRARY 검색 순서가 변경되었습니다.

프로시저

1. CSLB 트랜지언트 데이터 큐에 기록된 감사 로그를 조사하여 CICS 시스템의 LIBRARY 구성 변경사항과 결과 LIBRARY 검색 순서를 확인하십시오. 감사 로그는 플랫폼에 설치된 애플리케이션의 일부로 정의된 LIBRARY 연결에 대한 LIBRARY 구성 변경사항도 표시합니다. 애플리케이션에 전용인 이러한 LIBRARY 구성은 애플리케이션의 경우에는 해당 검색 순서에 표시되지만 CICS 시스템의 경우에는 검색 순서의 일부가 아닙니다.
2. 선택적으로 사내에서 또는 벤더가 개발한 유틸리티를 사용하여 이 시스템 또는 다중 시스템에 대한 감사 로그를 분석하고 해석하십시오.

결과

DFHLD0555I 및 DFHLD0556I는 검색 순서에 영향을 줄 수 있는 변경이 발생한 후 그로 인한 새 LIBRARY 검색 순서를 표시합니다. 검색 순서에서 활성화된 각 LIBRARY에 대해 하나의 DFHLD0556I 메시지 인스턴스가 있습니다. 검색 순서가 변경되지 않더라도 이러한 메시지가 표시될 많은 시나리오가 있습니다. 예를 들어, 사용 불가능한 LIBRARY는 검색 순서에 참여하지 않기 때문에 사용 불가능한 LIBRARY의 순위 값 변경은 검색 순서에 영향을 주지 않지만 변경은 이러한 메시지를 트리거합니다. 마찬가지로, 사용 가능한 LIBRARY의 순위를 검색 순서의 라이브러리 양쪽에서 LIBRARY 순위 사이에 여전히 놓여 있는 숫자로 변경하면 이러한 메시지가 트리거되지만 순서는 변경되지 않습니다.

LIBRARY 구성 정리

임시 수정사항을 적용하는 데 사용되었거나 더 이상 사용되지 않는 애플리케이션이 포함된 LIBRARY 연결을 정리하십시오.

시작하기 전에

- CICS가 실행 중입니다.

프로시저

1. CICS에 설치된 LIBRARY 자원 이름과 LIBRARY 변경사항의 감사 로그를 조사하여 더 이상 필요하지 않은 임시 수정사항을 적용하는 데 사용된 LIBRARY 자원 또는 더 이상 사용되지 않는 애플리케이션에 대한 LIBRARY 자원을 검색하십시오.
2. 더 이상 필요하지 않은 LIBRARY 자원을 버리십시오.
3. 나중에 필요하다고 믿고 있고 운영 프로시저가 이런 방법으로 LIBRARY 정의 재사용을 허용하지 않는 한 이러한 LIBRARY 자원의 정의를 삭제하십시오.

결과

CICS가 이전과 같이 계속됩니다. CICS 시스템에 설치된 LIBRARY 자원 세트는 시스템에서 사용된 현재 애플리케이션에 필요한 유일한 자원입니다.

MVS 거주와 주소 지정 모드 정의

애플리케이션에서 MVS 상주 및 주소 지정 모드의 영향, 모드를 변경하는 방법 및 애플리케이션을 영구 상주로 작성하는 방법을 설명합니다.

이 태스크 정보

명령 레벨 프로그램은 16 MB 위에 상주할 수 있고 2GB 위의 영역에 주소 지정할 수 있습니다.

프로그램의 주소 지정 모드 설정

MVS에서 실행하는 모든 프로그램에는 두 개의 속성이 지정됩니다. 주소 지정 모드 (AMODE)와 상주 모드(RMODE)입니다. AMODE는 프로그램이 제어를 받도록 설계된 주소 지정 모드를 지정합니다.

이 태스크 정보

일반적으로 사용자의 프로그램은 해당 주소 지정 모드에서 실행하도록 설계되어 있습니다. 물론 프로그램에서 모드를 전환할 수 있고, 로드 모듈에서 서로 다른 시작점에는 서로 다른 AMODE 속성을 가질 수 있습니다. RMODE 속성은 프로그램이 상주할 수 있는 가상 저장영역을 나타냅니다.

표 9. 유효한 AMODE 및 RMODE 스펙

속성	의미
AMODE(24)	24비트 주소 지정 모드를 지정합니다.
AMODE(31)	31비트 주소 지정 모드를 지정합니다.
AMODE(ANY)	24비트 또는 31비트 주소 지정 모드를 지정합니다.
AMODE(64)	64비트 주소 지정 모드를 지정합니다.
RMODE(24)	모듈은 16MB 이하의 가상 저장영역에 상주해야 합니다. 24비트 종속성이 있는 31비트 또는 64비트 프로그램에 RMODE(24)를 지정할 수 있습니다.
RMODE(ANY)	모듈은 16MB 미만 또는 16MB 이상이지만 2GB 이하인 가상 저장영역에 상주할 수 있습니다.

참고: C 또는 C++ 언어 프로그램은 AMODE(31)로 링크 편집되어야 합니다.

프로그램의 AMODE 또는 RMODE 속성을 지정하지 않는 경우에는 MVS는 시스템 기본값 AMODE(24) 및 RMODE(24)를 지정합니다. 이러한 기본값을 덮어쓰려면 다음 장소 중 하나 이상에 AMODE 및 RMODE를 지정할 수 있습니다. 이 목록의 지정은 목록에서 나중에 지정된 항목을 겹쳐씹니다.

프로시저

1. 링크 편집 MODE 제어 문에 AMODE 및 RMODE를 지정할 수 있습니다.

```
MODE AMODE(31),RMODE(ANY)
```

2. 다음 방법 중 하나를 사용하여 AMODE 및 RMODE를 지정할 수 있습니다.

- 링크 편집 작업 단계의 EXEC 문의 PARM 문자열에서:

```
//LKED EXEC PGM=IEWL,PARM='AMODE(31),RMODE(ANY),...'
```

- 링크 편집 단계의 EXEC 문의 명령과 동등한 처리를 일으키는 LINK TSO 명령에서.

3. 어셈블러 프로그램의 소스 코드에 AMODE 또는 RMODE 문을 지정할 수 있습니다. 컴파일러 옵션을 사용하여 COBOL에 이러한 모드를 설정할 수도 있습니다. COBOL 컴파일러에 대한 관련 애플리케이션 정보를 참조하십시오.

CICS 주소 공간 고려사항

AMODE 및 RMODE 속성의 유효한 조합 및 이들의 효과가 나열되어 있습니다.

표 10. 유효한 AMODE 및 RMODE 스펙 및 해당 영향

AMODE	RMODE	거주	주소 지정
24	24	16MB 미만	24비트 모드
31	24	16MB 미만	31비트 모드
ANY	24	16MB 미만	31비트 모드
31	ANY	16MB 이상	31비트 모드
64	24	16MB 미만	64비트 모드
64	ANY	16MB 이상	64비트 모드

다음 예제는 31비트 표준으로 코딩된 프로그램을 위한 링크 편집 제어문을 보여줍니다. 여기서 *anyname*은 로드 모듈의 이름입니다.

```
//LKED.SYSIN DD *
  MODE AMODE(31),RMODE(ANY)
  NAME anyname(R)
/*
//
```

프로그램을 영구 상주로 만들기

상주 속성 RESIDENT(YES)가 있는 CSD에 프로그램을 정의하는 경우에는 이는 첫 번째 참조에 로드됩니다. 이는 RMODE(ANY) 또는 RMODE(24)와 링크 편집된 프로그램에 적용됩니다. 그러나 CICS가 사용하는 저장영역 압축 알고리즘은 상주 프로그램을 제거하지 않음을 유의하십시오.

이 태스크 정보

태스크가 프로그램을 로드하는 데 충분한 저장영역이 없는 경우에는 태스크는 충분한 저장영역이 사용 가능하게 될 때까지 일시중단됩니다. DSA 중 하나라도 저장영역이 부족에 가까워지면 CICS는 사용 중이 아닌 프로그램이 차지하고 있는 저장영역을 해제합니다. CICS에서 동적 저장영역에 대한 자세한 정보는 *CICS 성능 안내서*에서 CICS 동적 저장영역 『CICS 동적 저장영역』을 참조하십시오.

RMODE(24) 프로그램을 상주로 만드는 대신에 이를 비상주로 만들고 라이브러리 색인(LLA) 기능을 사용할 수 있습니다. 이러한 프로그램이 사용하는 공간은 사용 횟수가 0에 도달하여 더 많은 가상 저장영역이 사용 가능하게 될 때 해제됩니다. LLA는 해당 라이브러리 디렉토리를 저장영역에 보관하고 LLA 관리 라이브러리 모듈의 단계(장소) 사본을 가상 색인 기능(VLF)이 관리하는 데이터 공간에 보관합니다. CICS는 DASD에서 프로그램 디렉토리를 검색하는 대신 저장영역의 LLA 라이브러리 디렉토리로부터 프로그램 모듈을 찾습니다. CICS가 스테이지형 모듈을 요청하면 LLA는 입력/출력 활동 없이 저장영역으로부터 이를 얻습니다.

링크 팩 영역에서 애플리케이션 실행

프로그램은 프로그램이 특정 요구사항을 따른다는 전제 하에 링크 팩 영역(LPA)에 상주할 수 있습니다.

어셈블러 언어, C, COBOL 또는 PL/I 프로그램은 읽기 전용이어야 하고 다음과 같은 요구사항을 준수해야 합니다.

어셈블러

RENT 어셈블러 옵션을 사용합니다.

C RENT 컴파일러 옵션을 사용합니다.

COBOL

WORKING STORAGE를 겹쳐쓰지 마십시오(CICS 변환기는 필요한 컴파일러 RENT 옵션과 함께 CBL문을 생성함(변환기 옵션 NOCBLCARD를 지정하지 않는 경우)).

PL/I

STATIC 저장영역을 겹쳐쓰지 마십시오(CICS 변환기가 필수 REENTRANT 옵션을 PROCEDURE문에 삽입함).

모든 프로그램은 RENT 및 REFR 옵션으로 링크 편집되어야 합니다.

CICS가 이러한 표준에 맞게 작성하고 LPA에 설치한 모듈을 사용하게 하려면 CSD에서 프로그램 자원 정의에 USELPACOPY(YES)를 지정하십시오.

읽기 전용 DSA에서 애플리케이션 실행

16MB 위에 상주하기에 적합하고 읽기 전용인 프로그램은 CICS 확장 읽기 전용 DSA(ERDSA)에 상주할 수 있습니다. 16MB 위에 상주하기에 적합하지 않고 읽기 전용인 프로그램은 16MB 아래 CICS 읽기 전용 DSA(RDSA)에 상주할 수 있습니다.

다음 조건은 ERDSA에 적합한 프로그램에 적용됩니다.

- 프로그램은 읽기 전용 표준에 맞게 적절하게 작성되어야 합니다.
- 프로그램은 31비트 주소 지정 표준에 맞게 작성되어야 합니다.
- 프로그램은 RENT 속성 및 RMODE(ANY) 상주 속성으로 링크 편집되어야 합니다.

다음 조건은 RDSA에 적합한 프로그램에 적용됩니다.

- 프로그램은 읽기 전용 표준에 맞게 적절하게 작성되어야 합니다.
- 프로그램은 RENT 속성 및 RMODE(24) 상주 속성으로 링크 편집되어야 합니다.

참고: RENTPGM=PROTECT가 시스템 초기화 매개변수로 지정된 CICS를 실행할 때 RDSA는 키 0 읽기 전용 저장영역에서 할당됩니다.

특정 언어에 대한 ERDSA 요구사항은 다음 주제에서 설명합니다.

어셈블러

CICS가 ERDSA에서 어셈블러 프로그램을 로드하려면 다음 옵션을 사용하여 이를 어셈블하고 링크 편집하십시오. RENT 어셈블러 옵션, 링크 편집 RENT 속성, RMODE(ANY) 상주 모드입니다.

이러한 옵션을 지정하는 경우에는 프로그램이 완전히 읽기 전용인지 확인하십시오. 예를 들어, 정적 저장영역에 쓰지 않는지를 확인해야 합니다. 그렇지 않으면 저장영역 예외가 발생합니다. 프로그램은 31비트 또는 64비트 주소 지정 표준에 작성되어야 합니다. ERDSA에 상주하는 프로그램에서 저장영역 보호 예외의 가능한 원인은 문제점 해결에서 보호 예외의 원인의 내용을 참조하십시오.

AMODE(31) 프로그램

AMODE(31) 애플리케이션, DFHEITAL을 위한 CICS 제공 프로시저에는 XREF 및 LIST 옵션만을 지정하는 LNKPARM 매개변수가 있습니다. ERDSA 적합한 프로그램을 링크 편집하려면 호출 작업으로부터 LNKPARM을 덮어써서 필요한 다른 옵션에 추가로 RENT 및 RMODE(ANY) 옵션을 지정하십시오.

예를 들어, 다음과 같습니다.

```
//ASMPROG JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//EITAL   EXEC DFHEITAL,
```

```

                (other parameters as necessary)
                .
//          LNKPARM='LIST,XREF,RMODE(ANY),RENT'

```

어셈블러 프로그램(DFHEAI)의 CICS EXEC 인터페이스 모듈은 AMODE(ANY) 및 RMODE(ANY)를 지정합니다. 그러나 어셈블러는 애플리케이션을 AMODE(24) 및 RMODE(24)로 기본적으로 설정하므로 결과로 나오는 로드 모듈 또한 AMODE(24) 및 RMODE(24)가 됩니다.

애플리케이션을 AMODE(31) 및 RMODE(ANY)로서 링크 편집하게 하려면 어셈블러 프로그램에서 적합한 명령문을 사용하십시오. 예를 들어, 다음과 같습니다.

```

MYPROG      CSECT
MYPROG      AMODE 31
MYPROG      RMODE ANY

```

다음과 같은 방법으로 AMODE 및 RMODE를 설정할 수 있습니다.

- JCL PARM 키워드에서 링크 편집(또는 바인더) 제어 정보를 사용하여 필수 AMODE 및 RMODE 스펙을 설정할 수 있습니다. 예를 들어, 다음과 같습니다.

```

//EITAL EXEC DFHEITAL,
          LNKPARM='LIST,XREF,RENT,AMODE(31),RMODE(ANY)'

```

- 또는 JCL에서 링크 편집 또는 바인더, 단계에서 SYSIN 데이터 세트에서 MODE 제어 명령문을 사용할 수 있습니다.

바인더를 사용할 때에는 충돌하는 AMODE 및 RMODE 스펙에 대한 예기치 않은 경고 메시지를 볼 수도 있습니다.

AMODE(64) 프로그램

AMODE(64) 애플리케이션, DFHEGTAL에 대한 CICS 제공 프로시저에는 XREF 및 LIST 옵션만을 지정하는 LNKPARM 매개변수가 있습니다. ERDSA 적합한 프로그램을 링크 편집하려면 호출 작업으로부터 LNKPARM을 덮어써서 필요한 다른 옵션에 추가로 RENT 및 RMODE(ANY) 옵션을 지정하십시오.

예를 들어, 다음과 같습니다.

```

//ASMPROG JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//EGTAL   EXEC DFHEGTAL,
          .
          (other parameters as necessary)
          .
//          LNKPARM='LIST,XREF,RMODE(ANY),RENT'

```

AMODE(64) 어셈블러 프로그램(DFHEAG)의 CICS EXEC 인터페이스 모듈은 AMODE(64) 및 RMODE(ANY)를 지정합니다. 그러나 어셈블러는 기본적으로 애플리케이션을 AMODE(24) 및 RMODE(24)로 설정하므로 결과로 나오는 로드 모듈 또한 AMODE(24) 및 RMODE(24)가 됩니다.

애플리케이션을 AMODE(64) 및 RMODE(ANY)로서 링크 편집하게 하려면 어셈블러 프로그램에서 적합한 명령문을 사용하십시오. 예를 들어, 다음과 같습니다.

```
MYPROG    CSECT
MYPROG    AMODE 64
MYPROG    RMODE ANY
```

다음과 같은 방법으로 AMODE 및 RMODE를 설정할 수 있습니다.

- JCL PARM 키워드에서 링크 편집(또는 바인더) 제어 정보를 사용하여 필수 AMODE 및 RMODE 스펙을 설정할 수 있습니다. 예를 들어, 다음과 같습니다.

```
//EITAL EXEC DFHEGTAL,
          LNKPARM='LIST,XREF,RENT,AMODE(64),RMODE(ANY)'
```

- 또는 JCL에서 링크 편집 또는 바인더, 단계에서 SYSIN 데이터 세트에서 MODE 제어 명령문을 사용할 수 있습니다.

바인더를 사용할 때에는 충돌하는 AMODE 및 RMODE 스펙에 대한 예기치 않은 경고 메시지를 볼 수도 있습니다.

C 및 C++

CICS가 C 및 C++ 프로그램을 ERDSA에 로드하기를 원하면 RENT 컴파일러 옵션을 사용하여 이를 컴파일하고 링크 편집하십시오.

CICS 제공 프로시저 DFHYITDL 또는 DFHYITFL(C의 경우)과 DFHYITEL 또는 DFHYITGL(C++의 경우)에는 수많은 링크 편집 옵션을 지정하는 LNKPARM 매개 변수가 있습니다. ERDSA 적합한 프로그램을 링크 편집하려면 호출 작업으로부터 이 매개변수를 덮어쓰고 필요한 기타 옵션에 RENT를 추가하십시오. C(DFHELII)의 CICS EXEC 인터페이스 모듈은 AMODE(31) 및 RMODE(ANY)를 사용하여 링크 편집되므로 RMODE(ANY) 옵션을 추가할 필요가 없습니다. 그러므로, 프로그램은 사용자가 CICS EXEC 인터페이스 스텝을 포함할 때 자동으로 AMODE(31) 및 RMODE(ANY)로서 링크 편집됩니다. 116 페이지의 『CICS 제공 인터페이스 모듈』의 내용을 참조하십시오.

다음 샘플 작업 문은 RENT 옵션이 추가된 LNKPARM 매개변수를 보여줍니다.

```
//CPROG    JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//YITDL    EXEC DFHYITDL,
          .
          (other parameters as necessary)
          .
//          LNKPARM='LIST,MAP,LET,XREF,RENT'
```

CICS로 배포된 순서 번호, 예를 들면 C 또는 C++ 샘플 프로그램을 포함하는 C 또는 C++ 코드를 컴파일하려면, SEQ을 지정하기 위해 CPARM 매개변수를 덮어써야 합니다. 예를 들어, 다음과 같습니다.

```
//EXAMPLE EXEC DFHYITEL,
// CPARM='/CXX OPT(1) SEQ NOMAR SOURCE'
//TRN.SYSIN DD *
    ... source code goes here ...
/*
//LKED.SYSIN DD *
    NAME EXAMPLE(R)
/*
```

COBOL

통합된 CICS 변환기를 사용하면, 컴파일러 RENT 컴파일러 옵션을 요구하므로 변환 동안 다른 CBL 카드를 추가할 필요가 없습니다.

별도의 변환 단계를 사용하는 COBOL 프로그램은 다음과 같은 이유로 자동으로 ERDSA에 적합합니다.

- CBLCARD 변환기 옵션(기본값)은 필수 컴파일러 옵션, RENT를 CICS 변환기가 생성하는 CBL 문에 자동으로 포함되도록 합니다. NOCBLCARD 변환기 옵션을 사용하는 경우에는 컴파일 작업 단계의 PARM 문에 RENT 옵션을 지정하거나 COBOL 매크로 IGYCOPT를 사용하여 설치 정의 옵션을 설정할 수 있습니다.
- COBOL 컴파일러는 읽기 전용 및 31비트 주소 지정 표준을 준수하는 코드를 자동으로 생성합니다.
- COBOL(DFHELII)의 CICS EXEC 인터페이스 모듈은 AMODE(31) 및 RMODE(ANY)로 링크 편집됩니다. 그러므로 프로그램은 CICS EXEC 인터페이스 스텝을 포함할 때 자동으로 AMODE(31) 및 RMODE(ANY)로서 링크 편집됩니다. 116 페이지의 『CICS 제공 인터페이스 모듈』의 내용을 참조하십시오.

링크 편집에 재진입 가능 속성을 지정해야 합니다. CICS 제공 프로시저 DFHYITVL에는 수많은 링크 편집 옵션을 지정하는 LNKPARM 매개변수가 있습니다. ERDSA 적합한 프로그램을 링크 편집하려면 호출 작업으로부터 이 매개변수를 덮어쓰고 필요한 기타 옵션에 RENT를 추가하십시오. 예를 들어, 다음과 같습니다.

```
//COBPROG JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//YITVL EXEC DFHYITVL,
    .
    (other parameters as necessary)
    .
// LNKPARM='LIST,XREF,RENT'
```

PL/I

CICS PL/I 프로그램은 일반적으로 정적 저장영역을 변경하지 않는 한 ERDSA에 적합합니다.

다음 요구사항은 CICS 또는 PL/I에 의해 강제 실행됩니다.

- PL/I 컴파일 프로시저는 RENT 컴파일러 옵션을 지정해야 합니다. PL/I 컴파일하기 위한 CICS 제공 프로시저(예: DFHYITPL)는 이 옵션을 자동으로 포함합니다.

- PL/I 컴파일러는 31비트 주소 지정 표준을 준수하는 코드를 자동으로 생성합니다.
- PL/I (DFHELII)의 CICS EXEC 인터페이스 모듈은 AMODE(31) 및 RMODE(ANY)로 링크 편집됩니다. 그러므로 프로그램은 CICS EXEC 인터페이스 스텝을 포함할 때 자동으로 AMODE(31) 및 RMODE(ANY)로서 링크 편집됩니다. 116 페이지의 『CICS 제공 인터페이스 모듈』의 내용을 참조하십시오.

링크 편집에 재진입 가능 속성을 지정해야 합니다. CICS 제공 프로시저 DFHYITPL에는 수많은 링크 편집 옵션을 지정하는 LNKPARM 매개변수가 있습니다. ERDSA 적합한 프로그램을 링크 편집하려면 호출 작업으로부터 이 매개변수를 덮어쓰고 필요한 기타 옵션에 RENT를 추가하십시오. 예를 들어, 다음과 같습니다.

```
//PLIPROG JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//YITPL EXEC DFHYITPL,
.
      (other parameters as necessary)
.
//          LNKPARM='LIST,XREF,RENT'
```

프로그램이 완전히 읽기 전용임이 확실하지 않는 한(예: 정적 저장영역에 쓰지 않음) 링크 편집 단계에서 RENT 속성을 지정하지 마십시오. 그렇지 않은 경우 저장영역 예외가 발생합니다. ERDSA에 상주하는 프로그램에서 저장영역 보호 예외의 가능한 원인은 문제점 해결에서 보호 예외의 원인의 내용을 참조하십시오.

애플리케이션에서 BMS 맵핑 세트 사용

애플리케이션에서 BMS 맵핑 세트를 사용하기 위한 프로시저가 요약되어 있습니다.

이 태스크 정보

애플리케이션을 설치하기 전에 다음 프로시저를 사용하여 CICS에서 실행하십시오.

프로시저

- 183 페이지의 제 10 장 『맵핑 세트 및 파티션 세트 설치』에 설명된 대로 프로그램에서 사용되는 BMS 맵핑 세트를 작성하십시오.
- DFHRPL 또는 동적 LIBRARY 연결에 있는 데이터 세트에 물리적 맵핑 세트(형식화 활동에서 BMS에 의해 사용됨)를 포함시키십시오.
- 기호 맵핑 세트(애플리케이션에 복사됨)를 사용자 복사 라이브러리에 포함시키거나 직접 애플리케이션 소스에 삽입하십시오.

DFHMAPS 프로시저는 기본적으로 CICSTS53.CICS.SDFHMAC 라이브러리로 설정되는 DSCTLIB 매개변수에 지정되는 라이브러리에 기호 맵핑 세트 출력을 기록합니다. 사용자 복사 라이브러리에 기호 맵핑 세트를 포함시키려면 다음 단계를 사용하십시오.

1. 물리적 및 기호 맵핑 세트를 함께 설치하는 데 사용되는 DFHMAPS 프로시저의 EXEC문에 DSCTLIB=name 피연산자로 라이브러리 이름을 지정하십시오.
2. 애플리케이션을 어셈블하고 컴파일하는 데 사용되는 작업 스트림의 SYSLIB 연결에 사용자 사본 라이브러리에 대한 DD문을 포함시키십시오.

DFHMAPS 프로시저가 CICSTS53.CICS.SDFHMAC 라이브러리(기본값)에 기호 맵핑 세트를 작성하도록 선택하는 경우 CICSTS53.CICS.SDFHMAC 라이브러리에 대한 DD문을 애플리케이션을 컴파일하는 데 사용되는 작업 스트림의 SYSLIB 연결에 포함시키십시오. 이는 어셈블러 언어 프로그램을 어셈블하는 데 사용되는 DFHEITAL 프로시저에는 필요하지 않습니다. 이러한 작업에는 이미 CICSTS53.CICS.SDFHMAC 라이브러리에 대한 DD문이 SYSLIB 연결에 포함되어 있기 때문입니다.

3. PL/I의 경우 32760바이트의 블록 크기를 가지는 라이브러리를 지정하십시오. 이는 PL/I 컴파일러에서 블록 크기 제한사항을 극복하는 데 필요합니다.

다음에 수행할 작업

맵핑 세트 설치에 대한 자세한 정보는 183 페이지의 제 10 장 『맵핑 세트 및 파티션 세트 설치』의 내용을 참조하십시오. BMS 서비스를 사용하기 위한 프로그램 작성에 대한 정보는 911 페이지의 제 45 장 『BMS(Basic Mapping Support)』의 내용을 참조하십시오.

CICS 제공 프로시저를 사용하여 애플리케이션 설치

CICS는 지원되는 각 프로그래밍 언어에 대한 별도의 카탈로그화된 프로시저에서 변환(필요한 경우), 컴파일 및 링크 편집 단계를 위해 작업 제어문(JCL)을 제공합니다.

CICS가 설치된 후에 CICSTS53.CICS.SDFHPROC 라이브러리에 설치되는 다음 프로시저를 프로시저 라이브러리에 복사하십시오. 프로시저 이름의 양식은 DFHwxTyL이고 여기서 변수 *w*, *x* 및 *y*는 프로그램의 유형(EXCI 일괄처리 또는 CICS 온라인), 컴파일러의 유형 및 프로그래밍 언어에 따라 다릅니다. 다음 테이블은 프로시저 이름을 보여줍니다.

표 11. 애플리케이션 설치 프로시저: *비Language Environment* 준수 컴파일러

언어	독립형 변환기	EXCI
어셈블러	DFHEITAL(AMODE(24) 및 AMODE(31) 애플리케이션) DFHEGTAL (AMODE(64) 애플리케이션)	DFHEXTAL

표 12. 애플리케이션 설치를 위한 프로시저: Language Environment 준수 컴파일러

언어	독립형 변환기	통합된 변환기	독립형 변환기가 있는 EXCI	통합된 변환기가 있는 EXCI
C	DFHYITDL(주 1 참조)	DFHZITDL(주 1 참조)	DFHYXTDL	DFHZXTDL
XPLINK 컴파일러 옵션을 사용하는 C	DFHYITFL(주 2 참조)	DFHZITFL(주 1 참조)	-	-
C++	DFHYITEL(주 1 참조)	DFHZITEL(주 1 참조)	DFHYXTEL	DFHZXTEL
XPLINK 컴파일러 옵션을 사용하는 C++	DFHYITGL(주 2 참조)	DFHZITGL(주 1 참조)	-	-
COBOL(주 3 참조)	DFHYITVL	DFHZITCL(주 2 참조)	DFHYXTVL	DFHZXTCL
PL/I(주 4 참조)	DFHYITPL(주 2 참조)	DFHZITPL(주 2 참조)	DFHYXTPL	DFHZXTPL(주 6 참조)

주:

1. DFHYITEL은 C에 대해서도 사용할 수 있습니다. **COMPILER** 매개변수에 C 컴파일러의 정확한 이름을 지정해야 합니다.
2. 생성된 모듈의 출력 라이브러리는 PDSE(PDS가 아님)입니다.
3. DFHZITCL은 COBOL 모듈 컴파일링에 권장되는 프로시저입니다. 통합 CICS 변환기를 포함하는 Enterprise COBOL 컴파일러의 버전을 사용하기 때문입니다.
4. DFHZITPL은 PL/I 모듈 컴파일링에 권장되는 프로시저입니다. 통합 CICS 변환기를 포함하는 Enterprise PL/I 컴파일러의 버전을 사용하기 때문입니다.
5. Language Environment(IMS™ 루틴)의 일괄처리 환경에서 EXEC DLI 명령을 실행하는 프로그램에 대해 다음과 같은 특수 프로시저를 사용하십시오.

DFHYBTPL

PL/I 애플리케이션

DFHYBTVL

COBOL 애플리케이션입니다.

참고: 이 프로시저에는 매크로 DFHLI000이 필요합니다.

6. Enterprise PL/I 버전 4.2, 4.3 또는 4.4에 적용할 APAR PI34113 및 Enterprise PL/I 버전 4.5에 적용할 APAR PI35108이 필요합니다.

로드 라이브러리 2차 익스텐트에 프로그램 설치

CICS는 CICS가 실행 중일 때 작성되는 로드 라이브러리 2차 익스텐트를 지원합니다. 1차 및 2차 익스텐트가 있는 DFHRPL 또는 동적 LIBRARY 집합에 라이브러리를 정의하고, 2차 익스텐트가 CICS가 실행 중일 때 로드 라이브러리에 링크 편집되는 결과

로서 추가되는 경우에는 라이브러리를 다시 여십시오. 이는 프로그램이 새 사본이 새 라이브러리 익스텐트를 초래했다라도 CEMT NEWCOPY 명령을 사용하여 새 버전을 소개할 수 있음을 의미합니다.

그러나 이는 2차 익스텐트에서 모듈을 로드할 때 검색 시간을 늘릴 수 있습니다. 가능한 경우 2차 익스텐트를 사용하기를 피해야 합니다.

참고: DFHXITPL을 사용 중인 경우 바인더 단계의 SYSLMOD DD 문은 PDSE(이전 PL/I 컴파일러에 관한 PDS가 아니라)를 참조해야 합니다.

CICS 제공 인터페이스 모듈 포함

CPI 통신 또는 SAA[®] 자원 복구를 애플리케이션에서 사용하려면 해당 인터페이스 모듈을 프로그램에 사용 가능하도록 해야 합니다.

CICS 라이브러리에 온라인 애플리케이션을 설치하기 위한 CICS 제공 프로시저에서는 해당 언어 EXEC 인터페이스 모듈에 대한 INCLUDE문을 포함하는 CICS 라이브러리 멤버를 지정합니다. 예를 들어, DFHYITVL 프로시저는 다음 명령문을 사용합니다.

```
//COPYLINK EXEC PGM=IEBGENER,COND=(7,LT,COB)
//SYSUT1 DD DSN=&INDEX..SDFHSAMP(&STUB),DISP=SHR
//SYSUT2 DD DSN=&&COPYLINK,DISP=(NEW,PASS),
//          DCB=(LRECL=80,BLKSIZE=400,RECFM=FB),
//          UNIT=&WORK,SPACE=(400,(20,20))
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN DD DUMMY

//SYSLIN DD DSN=&&COPYLINK,DISP=(OLD,DELETE)
//          DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//          DD DDNAME=SYSIN
```

이 COBOL 예제에서 기호 매개변수 STUB은 DFHEILID로 기본값이 설정됩니다. DFHEILID 멤버는 명령문 INCLUDE SYSLIB(DFHELII)를 포함합니다.

PL/I 및 C에 대해 제공된 프로시저는 DFHEILID를 참조하며 이는 DFHELII 스텝이 사용됨을 의미합니다.

애플리케이션이 CPI 통신 또는 SAA 자원 복구 기능을 사용할 예정이면 다음 중 하나를 수행하십시오.

- 애플리케이션을 설치하기 위해 CICS 제공 프로시저를 호출하는 데 사용되는 작업에서 LKED.SYSIN 덮어쓰기에 적절한 INCLUDE문을 추가하십시오. 다음 INCLUDE문을 추가하십시오.
 - INCLUDE SYSLIB(DFHCPLC)(프로그램이 CPI 통신을 사용하는 경우)
 - INCLUDE SYSLIB(DFHCPLRR)(프로그램이 SAA 자원 복구를 사용하는 경우)

링크 편집 단계 중에 DFHEI1 항목에 대한 DUPLICATE 정의를 표시하는 경고 메시지가 표시될 수 있습니다. 이러한 메시지는 무시할 수 있습니다.

링크 편집 요구사항에 대한 자세한 정보는 179 페이지의 『자체 작업 스트림 사용』의 내용을 참조하십시오.

어셈블러 언어 애플리케이션 변환, 어셈블 및 링크 편집

DFHEITAL 또는 DFHEXTAL 프로시저를 사용하여 어셈블러 언어로 작성된 AMODE(24) 및 AMODE(31) 애플리케이션을 변환 및 어셈블하고 링크 편집할 수 있습니다. DFHEGTAL 프로시저를 사용하여 어셈블러 언어로 작성된 AMODE(64) 애플리케이션을 변환 및 어셈블하고 링크 편집할 수 있습니다.

이 태스크 정보

그림 18에 표시된 샘플 작업 제어문을 사용하여 어셈블러 언어로 작성된 애플리케이션을 처리할 수 있습니다. 프로시저 이름에서 *x*는 프로그램이 CICS 애플리케이션인지, 그러한 프로그램의 AMODE인지 또는 EXCI 일괄처리 프로그램인지에 따라 다릅니다. CICS 제공 프로시저의 이름은 164 페이지의 『CICS 제공 프로시저를 사용하여 애플리케이션 설치』의 내용을 참조하십시오.

```
//jobname      JOB      accounting info,name,MSGLEVEL=1
//              EXEC      PROC=DFHEXTAL                      1
//TRN.SYSIN     DD      *
*ASM           XOPTS(translator options . . .)                2
               .
               assembler language source statements
               .
/*
//LKED.SYSIN     DD      *
               NAME      anyname(R)
/*
//
```

그림 18. DFHEXTAL 프로시저를 호출하기 위한 샘플 작업 제어문

*anyname*은 로드 모듈의 이름입니다.

참고:

1. 프로그램을 읽기 전용 DSA에 설치하려면 추가 세부사항은 159 페이지의 『읽기 전용 DSA에서 애플리케이션 실행』의 내용을 참조하십시오.

LPA에서 사용할 프로그램을 설치하려면 다음 옵션을 추가하십시오.

- DFHEXTAL 프로시저의 ASM 단계에 대한 EXEC문에서 RENT에서 PARM 옵션

- DFHE_xTAL 프로시저에 대한 호출에서 LNKPARM 매개변수에 대한 RENT 및 REFR 옵션

(158 페이지의 『링크 팩 영역에서 애플리케이션 실행』 참조.)

2. XOPTS문에 포함할 수 있는 변환기 옵션에 대한 자세한 정보는 113 페이지의 『변환기 옵션 정의』의 내용을 참조하십시오.

다음 예제는 CICS 제공 프로시저 DFHEGTAL을 사용하여 AMODE(64) 애플리케이션을 변환하고 어셈블하며 링크 편집하기 위한 작업 제어문을 보여줍니다.

```
//APPLPROG EXEC DFHEGTAL
//TRN.SYSIN DD *
.
. Application program
.
/*
//LKED.SYSIN DD *
  ENTRY program_name
  NAME program_name(R)
/*
```

*program_name*은 AMODE(64) 애플리케이션의 이름입니다.

169 페이지의 그림 19에서는 변환기 목록 및 출력 파일을 생성하기 위해 CICS.SDFHLOAD에 대한 참조와 함께 명령 레벨 변환기로 처리되는 어셈블러 소스 프로그램을 보여줍니다. 그러면 이 출력 파일은 CICS.SDFHMAC에 대한 참조와 함께 어셈블러에 의해 처리되어 어셈블러 목록 및 추가 출력 파일을 생성합니다. 이 출력 파일은 연계 편집기에서 처리되어 연계 편집기 목록 및 애플리케이션 라이브러리에 저장되는 로드 모듈을 생성합니다.

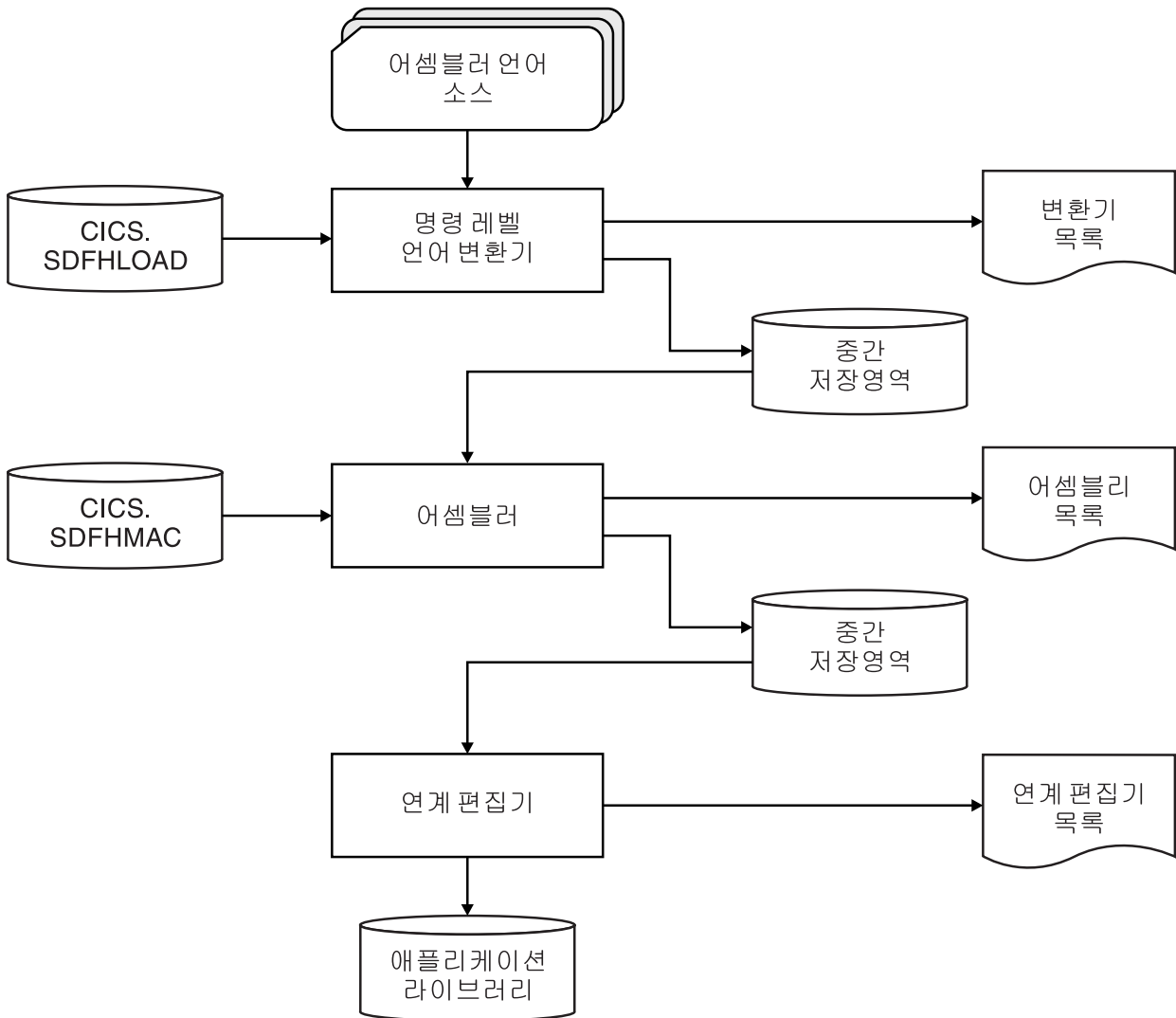


그림 19. DFHEITAL 또는 DFHEGTAL 프로시저를 사용하여 어셈블러 언어 프로그램 설치

COBOL 애플리케이션 설치

이 다이어그램은 별도의 변환기 단계를 필요로 하는 COBOL 및 PL/I 프로그램에 대해 카탈로그화된 프로시저에서 제어의 플로우를 보여줍니다. 통합 변환기를 사용하는 경우에는 별도의 변환기 단계가 없습니다. 상위 레벨 언어 소스 및 CICS.SDFHLOAD 둘 다 컴파일러에 입력하고 결합된 변환기와 컴파일러 목록이 생성됩니다.

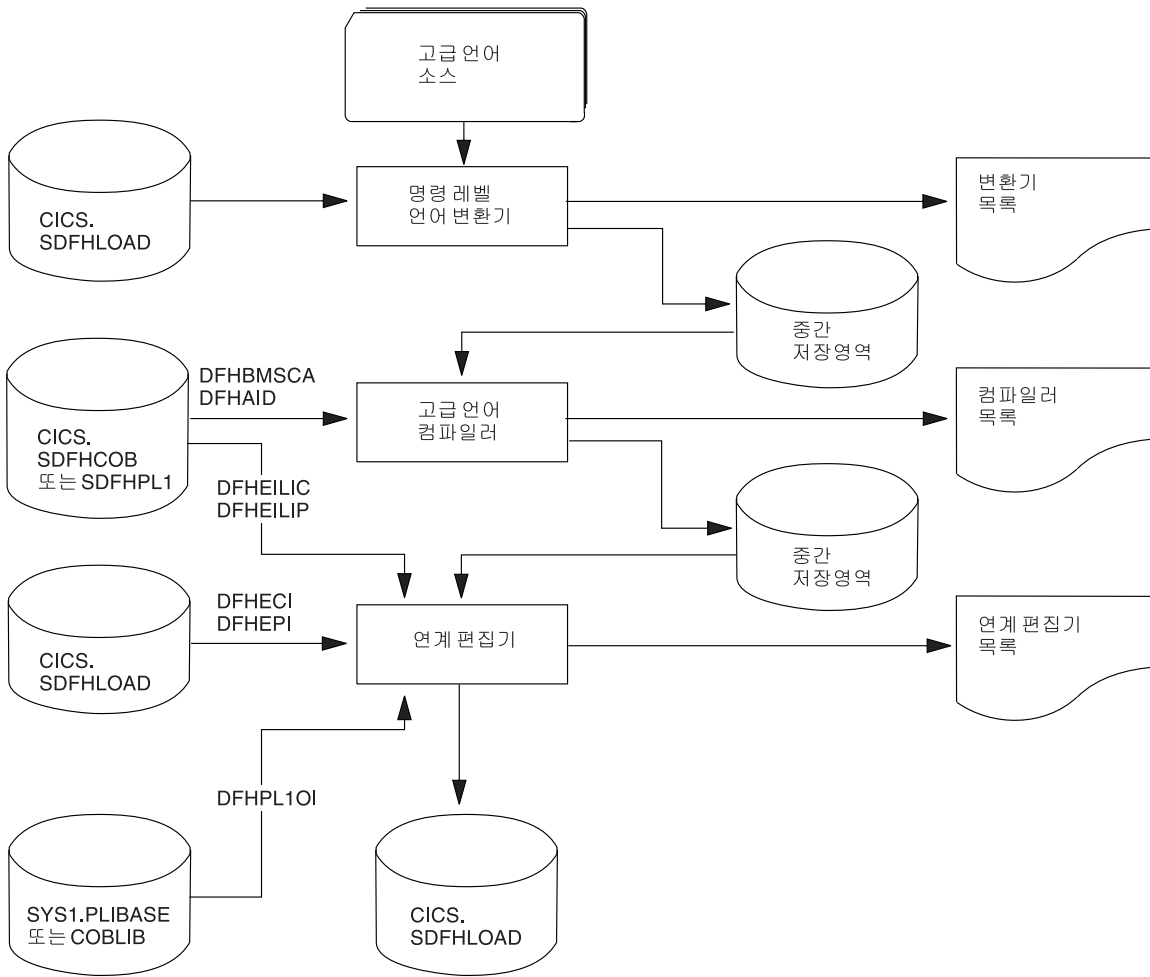


그림 20. COBOL과 PL/I 프로그램 설치

COBOL 애플리케이션을 설치하기 위한 샘플 JCL

이 예제에 표시된 작업 제어문을 사용하여 별개 또는 통합된 변환기가 있는 COBOL 애플리케이션을 처리할 수 있습니다.

프로시저 이름은 CICS 애플리케이션 또는 EXCI 일괄처리 프로그램인지에 따라 다릅니다. CCIS와 함께 제공된 COBOL 프로시저의 이름은 165 페이지의 표 12의 내용을 참조하십시오.

```
//jobname      JOB  accounting info,name,MSGLEVEL=1
//              EXEC PROC=procname                      1
//TRN.SYSIN DD      *                                    2
CBL  XOPTS(Translator options . . .)                    3
      .
      COBOL source statements
      .
/*
//LKED.SYSIN DD      *                                    4
              NAME      anyname(R)
/*
//
```

그림 21. DFHYITVL 또는 DFHYXTVL 프로시저를 호출하기 위한 샘플 작업 제어문

여기서 *procname*은 프로시저의 이름이고 *anyname*은 로드 모듈 이름입니다.

통합된 변환기를 호출하기 위해 프로시저 DFHZITCL 또는 DFHZXTCL을 사용하려면 그림 22에 표시된 작업 제어문을 사용할 수 있습니다.

```
//jobname      JOB  accounting info,name,MSGLEVEL=1
//              EXEC PROC=procname,PROGLIB=dsnname        1
//COBOL.SYSIN DD *
      .
      COBOL source statements
      .
/*
//LKED.SYSIN DD *
              NAME anyname(R)
/*
//
```

그림 22. DFHZITCL 또는 DFHZXTCL 프로시저를 사용하기 위한 샘플 작업 제어문

여기서 *anyname*은 로드 모듈 이름입니다.

참고:

1. 변환기 옵션: 컴파일 단계에서 필요한 COBOL 기능의 버전에 따라 COBOL3 또는 COBOL2 변환기 옵션을 지정하십시오.
2. 컴파일러 옵션:

COBOL 프로그램을 컴파일하려면 컴파일러 옵션 RENT, NODYNAM 및 LIB가 필요합니다.

변환기 옵션, CBLCARD(기본값)를 사용하는 경우에는 CICS 변환기는 이러한 옵션을 포함하는 CBL 문을 자동으로 생성합니다. 변환기 옵션 NOCBLCARD를 지정하여 CBL 또는 PROCESS 카드의 생성을 방지할 수 있습니다.

CICS가 제공한 COBOL 프로시저에서 COB 단계의 PARM 문은 컴파일러 옵션의 값을 지정합니다. 예를 들어, 다음과 같습니다.


```
//COB      EXEC PGM=IGYCRCTL,REGION=&REG,
//          PARM='NODYNAM,LIB,OBJECT,RENT,APOST,MAP,XREF'
```

통합된 변환기가 있는 컴파일러로 COBOL 프로그램을 컴파일하려면 CICS 컴파일러 옵션을 사용하여 컴파일러가 변환기를 호출하기를 원함을 표시하십시오. DFHZITCL 프로시저에는 이 컴파일러 옵션이 포함됩니다.

```
CBLPARM='NODYNAM,LIB,MAP,CICS(''COBOL3'')'
```

참고: PARM 문자열에서 통합된 변환기에 대한 CICS 변환기 옵션을 지정하는 경우에는 이 예제에 표시된 대로 이중 어퍼스트로피가 필요합니다. 그러나 소스 프로그램에 옵션을 지정하는 경우에는 단일 어퍼스트로피가 필요합니다(예를 들어, 소스 프로그램에서 CBL 문으로서 CBL CICS('COBOL3,SP') APOST가 있을 수 있습니다.)

CICS가 제공하는 COBOL 프로시저는 SIZE 및 BUF 옵션에 값을 지정하지 않습니다. 기본값은 SIZE=MAX 및 BUF=4K입니다. SIZE는 컴파일러에 사용 가능한 가상 저장영역의 양을 정의하고, BUF는 각 컴파일러 버퍼 작업 파일에 할당될 동적 저장영역의 양을 정의합니다. 프로시저를 호출하는 EXEC 문에서 PARM.COBL 매개변수를 사용하여 이러한 옵션을 변경할 수 있습니다. 예를 들어, 다음과 같습니다.

```
EXEC PROC=procname,PARM.COBL='SIZE=512K,BUF=16K,.,.,.'
```

다음 메소드 중 하나를 사용하여 컴파일러 옵션을 변경하십시오.

- COBOL 프로시저의 COB 단계에 정의된 PARM 문을 덮어쓰십시오.

프로시저를 호출하는 작업에 PARM 문을 지정하는 경우에는 프로시저 JCL에 지정된 모든 옵션을 덮어씁니다. 원하는 모든 옵션이 덮어쓰기 또는 CBL 문에 지정되는지 확인하십시오.

- COBOL 프로시저를 호출하는 데 사용된 작업 스트림에서 소스 문의 시작부에 CBL 문을 지정하십시오.
- COBOL 설치 기본 매크로, IGYCOPT를 사용하십시오. 이 매크로는 CBL 문을 사용하지 않는 경우 필수입니다(즉, 변환기 옵션 NOCBLCARD를 지정하는 경우).
- COBOL 프로그램에 컴파일러 옵션을 포함하는 데이터 세트를 정의하고, 이 데이터 세트는 CICS 컴파일러 옵션 및 해당 서브매개변수를 포함해야 합니다.

다음 방법 중 하나로 SYSOPTF DD 문을 코딩하십시오.

```
// SYSOPTF DD DSNAME=dsname,UNIT=SYSDA,VOLUME=(subparms),DISP=SHR
```

이 코드 단편에서, 컴파일러 옵션은 데이터 세트 *dsname*에 저장됩니다.

```
//COBOL EXEC PGM=IGYCRCTL,REGION=4M,PARM=(OPTFILE)
//SYSOPTF DD *
APOST
LIB
TRUNC(OPT)
CICS('COBOL3,SP')
NODYNAM
RENT
LIST
MAP
XREF
OPT
TEST(ALL,SEPARATE)
//STEPLIB DD DSN=PP.COBOL390.V410.SIGYCOMP,DISP=SHR
```

이 코드 단편에서는 컴파일러 옵션은 **OPTFILE** 매개변수 후에 코드에 직접 배치됩니다.

SYSOPTF 문에 대한 자세한 정보는 Enterprise COBOL for z/OS Information Center: 를 참조하십시오.

변환기 옵션 CBLCARDINOCBLCARD에 대한 자세한 정보는 113 페이지의 『변환기 옵션 정의』의 내용을 참조하십시오. NOCBLCARD 옵션을 사용하기로 선택한 경우에는 IGYOS4006-E 오류 메시지가 발행되는 것을 막기 위해 COBOL 컴파일러 옵션 ALLOWCBL=NO 또한 지정해야 합니다. ALLOWCBL 컴파일러 옵션에 대한 자세한 정보는 COBOL 버전에 해당하는 관련 설치 및 사용자 정의 매뉴얼을 참조하십시오.

3. 변환기에 입력이 없는 경우에는 DD * 대신에 DD DUMMY를 지정할 수 있습니다. 그러나 DD DUMMY를 지정하는 경우에는 적합한 DCB 피연산자 또한 코딩하십시오. 변환기는 SYSIN 데이터 세트에 모든 데이터 제어 블록 정보를 제공하지 않습니다.
4. CICS TS와 함께 제공된 독립형 변환기가 사용되는 경우에는, XOPTS 문에서 변환기 옵션은 COBOL 프로시저에서 유사한 옵션을 덮어씁니다.

XOPTS문에 포함할 수 있는 변환기 옵션에 대한 자세한 정보는 113 페이지의 『변환기 옵션 정의』의 내용을 참조하십시오.

통합된 CICS 변환기가 사용될 때 COBOL 컴파일러는 XOPTS가 아니라 변환기 옵션을 정의하기 위한 키워드 CICS만을 인식합니다.

5. 링크 편집하여 미해결된 약한 외부 참조를 무시할 수 있습니다.

링크 편집 작업 단계에는 적합한 CICS의 환경 특정 모듈 및 Language Environment 링크 편집 모듈을 포함하는 라이브러리에 대한 액세스가 필요합니다. 모듈 및 라이브러리 서브루틴이 라이브러리에 다른 이름으로 설치된 경우 이러한 라이브러리의 이름을 덮어쓰거나 변경하십시오.

프로그램을 읽기 전용 DSA에 설치 중인 경우에는 자세한 내용은 159 페이지의 『읽기 전용 DSA에서 애플리케이션 실행』의 내용을 참조하십시오.

LPA에서 사용될 프로그램을 설치 중인 경우에는 COBOL 프로시저에 대한 호출에서 LNKPARM 매개변수에 RENT 및 REFR 옵션을 추가하십시오. 자세한 정보는 158 페이지의 『링크 팩 영역에서 애플리케이션 실행』의 내용을 참조하십시오.

PL/I 애플리케이션 설치

DFHYxTPL 프로시저를 사용하여 개별 변환기로 PL/I 애플리케이션을 처리할 수 있습니다. 값“x”는 CICS 애플리케이션인지 또는 EXCI 일괄처리 프로그램인지에 따라 다릅니다. 또는 DFHZxTPL 프로시저를 통합된 변환기를 호출하는 데 사용할 수 있습니다.

170 페이지의 그림 20에서는 PL/I 프로그램에 대해 카탈로그화된 프로시저에서 제어의 플로우를 설명합니다.

PL/I 프로그램 준비에 대한 자세한 정보는 *PL/I 프로그래밍 안내서*를 참조하십시오.

C 애플리케이션 설치

이 다이어그램은 별도의 변환기 단계를 필요로 하는 C 명령 레벨 프로그램에 대해 DFHYxTzL 카탈로그화된 프로시저에서 제어의 플로우를 보여줍니다. 통합 변환기를 사용하는 경우에는 별도의 변환기 단계가 없습니다. 상위 레벨 언어 소스 및 CICS.SDFHLOAD 둘 다 컴파일러에 입력하고 결합된 변환기와 컴파일러 목록이 생성됩니다.

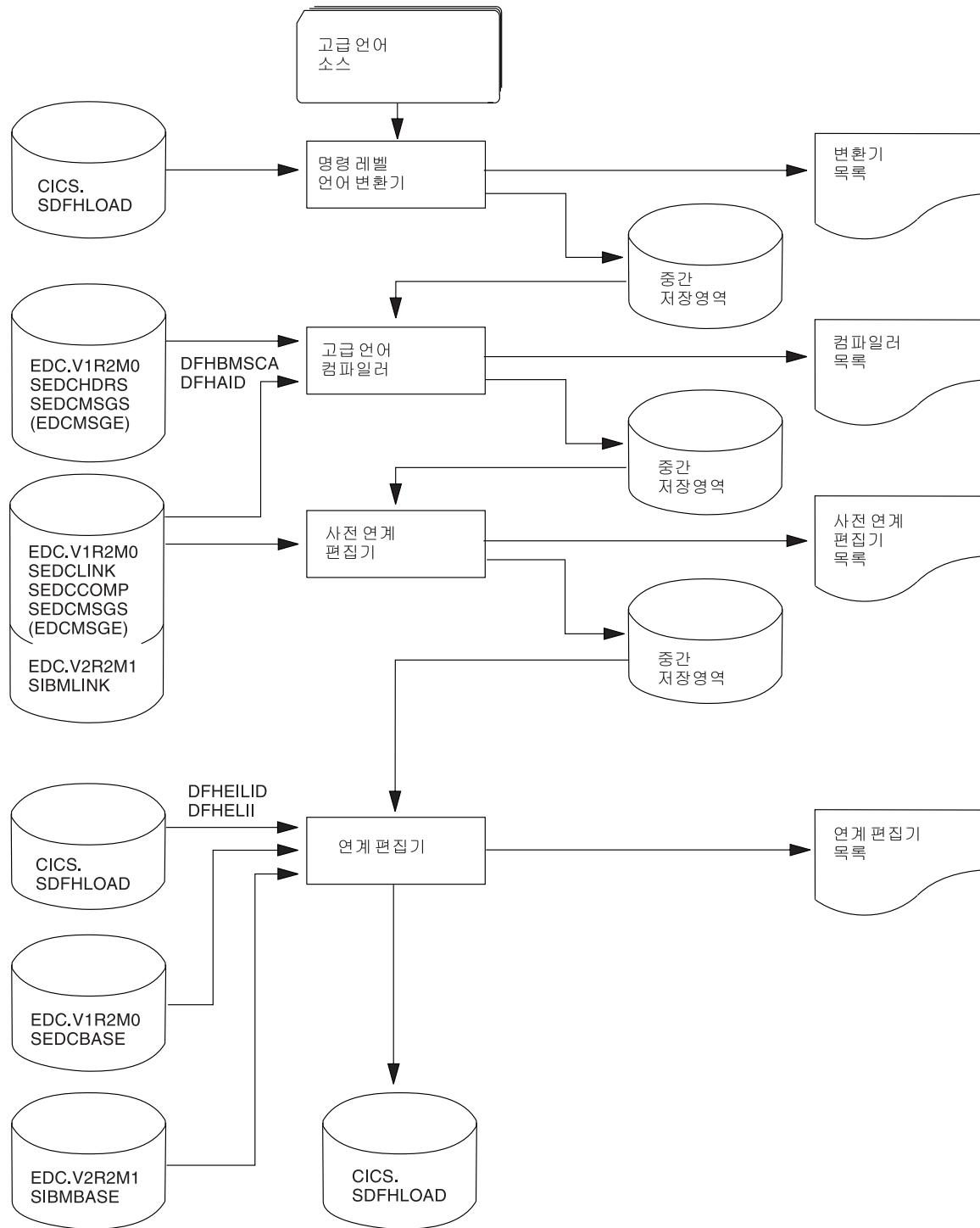


그림 23. DFHYxTzL 프로시저를 사용하여 C 프로그램 설치

변환기, 컴파일러, 예비 연계 편집기 및 연계 편집기 단계가 있으며 각각의 단계는 다음 단계로 전달되는 중간 파일 및 목록을 생성합니다. C 라이브러리는 컴파일러, 예비 연계 편집기 및 연계 편집기에서 참조됩니다.

참고: XPLINK 컴파일러 옵션을 선택하면 이 다이어그램에 미리 링크 단계가 없습니다.

C 프로그램을 설치하려면 먼저 C 라이브러리 및 컴파일러와 C에 대해 생성된 CICS 지원이 설치되어 있어야 합니다(z/OS용 CICS Transaction Server 설치 안내서 참조).

C 애플리케이션을 설치하기 위한 샘플 JCL

이 예제에 표시된 작업 제어문을 사용하여 C 애플리케이션을 처리할 수 있습니다. 프로시저 이름에서 x는 프로그램이 CICS 애플리케이션 또는 EXCI 일괄처리 프로그램인지 여부에 따라 달라집니다.

CICS 제공 프로시저의 이름은 165 페이지의 표 12의 내용을 참조하십시오.

```
//jobname    JOB  accounting info,name,MSGLEVEL=1
//          EXEC PROC=DFHYxTzL                      1
//TRN.SYSIN  DD      *                              2
#pragma XOPTS(Translator options . . .)             3
.
C source statements
.
/*
//LKED.SYSIN DD      *                              4
                NAME    anyname(R)

/*
//
```

여기서 anyname은 로드 모듈 이름입니다.

그림 24. DFHYxTzL 프로시저를 호출하기 위한 샘플 JCL

C 프로그램을 설치하기 위한 참고

1. 컴파일러 옵션: 프로시저를 호출하는 EXEC 문에서나 #pragma 옵션 지시문에서 매개변수 덮어쓰기 **CPARM**을 사용하여 컴파일러 옵션을 코딩할 수 있습니다.

C 또는 C++ 소스가 DFHYITEL, DFHZITEL 또는 DFHYXTEL JCL 프로시저에서 사용 가능하게 하려면 소스 입력 여백을 제한해야 합니다. 소스 입력 여백을 다음으로 제한할 수 있습니다.

- 다음 pragma 선언을 지정하여 C 또는 C++ 소스를 직접 변경:

```
#pragma margins(m, n)
```

여기서 *m* 및 *n*은 C 또는 C++ 소스가 있는 열입니다. 예제: #pragma margins(1,72).

- JCL 프로시저에서 **CPARM** 덮어쓰기 옵션 변경: 소스에 순서 번호가 있는지 여부에 따라 SEQ 또는 NOSEQ를 지정하십시오.

2. 변환기에 입력이 없는 경우에는 DD * 대신에 DD DUMMY를 지정할 수 있습니다. 그러나 DD DUMMY를 지정하는 경우에는 적합한 DCB 피연산자 또한 코딩하십시오. (변환기는 SYSIN 데이터 세트에 모든 데이터 제어 블록 정보를 제공하지 않습니다.)
3. 변환기 옵션: XOPTS문에 포함할 수 있는 변환기 옵션에 대한 자세한 정보는 113 페이지의 『변환기 옵션 정의』의 내용을 참조하십시오.
4. 프로그램을 읽기 전용 DSA에 설치 중인 경우에는 자세한 내용은 159 페이지의 『읽기 전용 DSA에서 애플리케이션 실행』의 내용을 참조하십시오.

LPA에서 사용될 프로그램을 설치 중인 경우에는 DFHYxTzL 프로시저에 대한 호출에서 LNKPARM 매개변수에 RENT 및 REFR 옵션을 추가하십시오. (자세한 정보는 158 페이지의 『링크 팩 영역에서 애플리케이션 실행』의 내용을 참조하십시오.)

C 언어 프로그램은 AMODE(31)로 링크 편집되어야 하므로 DFHYxTzL 프로시저는 기본적으로 AMODE(31)를 지정하십시오.

XL C를 위한 통합된 CICS 변환기 호출

XL C의 통합된 변환기를 호출하기 위해 프로시저를 사용하려면 그림 25에 표시된 작업 제어문을 사용할 수 있습니다.

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC DFHZITxL,PROGLIB=dsname          1
//C.SYSIN DD *
.
. C source statements
.
//LKED.SYSIN DD *
      NAME anyname(R)
//
```

여기서 anyname은 로드 모듈 이름입니다.

그림 25. XL C의 통합 변환기를 위한 샘플 JCL

1. 변환기 이름: XPLINK가 없는 C 프로그램에는 DFHZITDL를 지정하거나 XPLINK가 있는 C 프로그램에는 DFHZITFL을 지정하십시오.

XL C++를 위한 통합된 CICS 변환기 호출

XL C++의 통합된 변환기를 호출하기 위해 프로시저를 사용하기 위해 178 페이지의 그림 26에 표시된 작업 제어문을 사용할 수 있습니다.

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC DFHZITxL,PROGLIB=dsnnname          1
//CPP.SYSIN DD *
.
. C++ source statements
.
//LKED.SYSIN DD *
      NAME anyname(R)

//
```

여기서 anyname은 로드 모듈 이름입니다.

그림 26. XL C++의 통합 변환기를 위한 샘플 JCL

1. 변환기 이름: XPLINK가 없는 C++ 프로그램에는 DFHZITEL를 지정하거나 XPLINK가 있는 C++ 프로그램에는 DFHZITGL을 지정하십시오.

C 소스 코드에 사전에 변환된 코드 포함

변환기는 dfhexec 또는 DFHEXEC를 생성할 수 있습니다. 두 버전이 모두 프로그램에 있으면 IEW2456E 오류 메시지가 표시됩니다. 이 오류를 막는 데에는 두 가지 방법이 있습니다. dfhexec를 포함하는 이전 코드를 재컴파일하거나 작업에서 prelinker RENAME 제어문을 사용하십시오.

이 태스크 정보

다음 샘플 JCL은 RENAME 제어문을 사용하는 방법을 보여줍니다.

```
//jobname    JOB  accounting info,name,MSGLEVEL=1
//          EXEC PROC=DFHYxTzL
//TRN.SYSIN  DD      *
#pragma XOPTS(Translator options . . .)
.
. C source statements
.
/*
//PLKED.SYSLIN DD      *
      RENAME dfhexec DFHEI1
//LKED.SYSLIN DD      *
      NAME anyname(R)
/*
//
```

여기서 anyname은 로드 모듈 이름입니다.

그림 27. dfhexec의 이름을 바꾸기 위한 샘플 JCL

자체 작업 스트림 사용

카탈로그화된 제공된 프로시저를 모델로 사용하여 자체 JCL을 작성하여 애플리케이션을 변환, 어셈블(또는 컴파일), 링크 편집할 수 있습니다.

프로시저는 CICSTS53.CICS.SDFHPROC 라이브러리에 설치됩니다.

다음 정보는 변환기에 대한 중요한 사항과 프로그램의 각 주요 범주를 요약합니다. 단 순성을 위해 설명에서는 프로그램이 CICSTS53.CICS.SDFHLOAD 또는 IMS.PGMLIB에 로드된다고 가정합니다. 임의의 라이브러리에 프로그램을 로드할 수 있지만 CICS 작업 스트림의 DFHRPL 또는 동적 LIBRARY 연결이나 일괄처리 작업 스트림의 STEPLIB 라이브러리 연결(독립형 IMS 일괄처리 프로그램의 경우)에 포함된 경우에만 가능합니다.

참고: 작업 스트림에 참조된 IMS 라이브러리는 IMS.libnam(예: IMS.PGMLIB)을 통해 식별됩니다. IMS 라이브러리에 자체 이름 지정 규칙을 사용하는 경우, IMS 라이브러리의 이름도 그에 따라 바뀌어야 합니다.

변환기 요구사항

CICS 변환기에는 최소 256KB의 가상 저장영역이 필요합니다. 변환기 옵션 CICS 및 DLI를 사용해야 합니다.

EXEC CICS 또는 EXEC DLI 명령을 사용하는 온라인 프로그램

1. 항상 변환기 옵션 CICS를 사용하십시오. 프로그램이 EXEC DLI 명령을 실행하는 경우 변환기 옵션 DLI를 사용하십시오.
2. SYSLIN DD문이 정의한 링크 편집 입력은 오브젝트 텍 앞에 올바른 인터페이스 모듈을 포함해야 합니다. 그러므로 오브젝트 텍 앞에 인터페이스 모듈에 대한 INCLUDE문을 배치하십시오. 또한 INCLUDE문 앞에 ORDER문을, 모든 INCLUDE문 뒤에 ENTRY문을 배치하십시오.

인터페이스 모듈은 다음과 같습니다.

DFHEAI

어셈블러

DFHELII

모든 HLL 언어

CICS 제공 프로시저에서 SYSLIN DD문이 정의한 링크 편집 단계에 대한 입력은 라이브러리 멤버를 오브젝트 텍과 연결합니다. 이 멤버는 필수 인터페이스 모듈에 대한 INCLUDE문을 포함합니다. 예를 들어, DFHYITVL 프로시저는 다음 INCLUDE문을 포함하는 라이브러리 멤버 DFHEILID를 연결합니다.

```
INCLUDE SYSLIB(DFHELII)
```


3. SYSLMOD DD문이 정의한 링크 편집의 로드 모듈 출력을

CICSTS53.CICS.SDFHLOAD 또는 자체 프로그램 라이브러리에 배치하십시오.

그림 28은 COBOL 애플리케이션을 설치하는 데 사용할 수 있는 샘플 JCL과 인라인 프로시저(CICS 제공 프로시저 DFHYITVL을 기반으로 함)를 표시합니다. 이 프로시저는 DFHYITVL 프로시저에 포함된 것처럼 필수 인터페이스 모듈에 대한 INCLUDE 문을 포함하는 라이브러리 멤버 DFHEILID 연결과 COPYLINK 단계를 포함하지 않습니다. 대신에 JCL은 다음 INCLUDE문을 제공합니다.

```
INCLUDE SYSLIB(DFHELIB)
```

이 명령문이 제공되지 않은 경우, 링크 편집은 미해결 외부 참조에 대해 오류 메시지를 리턴하며 프로그램 출력은 실행 불가능으로 표시됩니다.

그림 28. COBOL 프로그램 설치를 위한 샘플 사용자 정의 JCL

```
/*      The following JCL could be used to execute this procedure
/*
//APPLPROG EXEC MYIITVL,
//      INDEX='CICSTS53.CICS
//      PROGLIB='CICSTS53.CICS.SDFHLOAD',
//      DSCTLIB='CICSTS53.CICS.SDFHCOB',
//      INDEX2='user.qualif'
//      OUTC=A,                Class for print output
//      REG=4M,                Region size for all steps
//      LNKPARM='LIST,XREF',    Link edit parameters
//      WORK=SYSDA             Unit for work data sets

//TRN.SYSIN DD *
/*      .
/*      . Application program
/*      .
/*
//LKED.SYSIN DD *
//      INCLUDE SYSLIB(DFHELIB)
//      NAME anyname(R)

/*
//MYIITVL PROC SUFFIX=1$,      Suffix for translator module
//      INDEX='CICSTS53.CICS', Qualifier(s) for CICS libraries
//      PROGLIB='CICSTS53.CICS.SDFHLOAD', Name of o/p library
//      DSCTLIB='CICSTS53.CICS.SDFHCOB', Private macro/dsect
//      AD370HLQ='SYS1',       Qualifier(s) for AD/Cycle compiler
//      LE370HLQ='SYS1',       Qualifier(s) for Language Environment libraries
//      OUTC=A,                Class for print output
//      REG=4M,                Region size for all steps
//      LNKPARM='LIST,XREF',    Link edit parameters
//      WORK=SYSDA             Unit for work data sets
/*

/*      This procedure contains 3 steps
/*      1. Exec the COBOL translator (using the supplied suffix 1$)
/*      2. Exec the COBOL compiler
/*      3. Linkedit the output into data set &PROGLIB
```

```
//TRN      EXEC PGM=DFHECP &SUFFIX,,
//          PARM='COBOL3',
//          REGION=&REG

//STEPLIB DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSPUNCH DD DSN=&&SYSCIN,
//          DISP=(,PASS),UNIT=&WORK,
//          DCB=BLKSIZE=400,
//          SPACE=(400,(400,100))
//*
//COB      EXEC PGM=IGYCRCTL,REGION=&REG,
//          PARM='NODYNAM,LIB,OBJECT,RENT,APOST,MAP,XREF'
//STEPLIB DD DSN=&AD370HLQ..SIGYCOMP,DISP=SHR
//SYSLIB DD DSN=&DSCTLIB,DISP=SHR
//          DD DSN=&INDEX..SDFHCOB,DISP=SHR
//          DD DSN=&INDEX..SDFHMAC,DISP=SHR
//          DD DSN=&INDEX..SDFHSAMP,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN DD DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),
//          UNIT=&WORK,SPACE=(80,(250,100))
//SYSUT1 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT2 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT3 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT4 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT5 DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT6 DD UNIT=&WORK,SPACE=(460,(350,100))
//*
//LKED     EXEC PGM=IEWL,REGION=&REG,
//          PARM='&LNKPARM',COND=(5,LT,COB)
//SYSLIB DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//          DD DSN=&LE370HLQ..SCEELKED,DISP=SHR
//SYSLMOD DD DSN=&PROGLIB,DISP=SHR
//SYSUT1 DD UNIT=&WORK,DCB=BLKSIZE=1024,
//          SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=&OUTC
//SYSLIN DD DSN=&&COPYLINK,DISP=(OLD,DELETE)
//          DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//          DD DDNAME=SYSIN
//PEND
//*
```

CALL DLI 인터페이스를 사용하는 온라인 프로그램

1. 변환기 옵션 CICS를 지정하고 변환기 옵션 DLI를 지정하지 마십시오.

참고: CICS 명령을 사용하지 않으며 실행 중인 트랜잭션에 의해서만 호출되고 CICS 태스크 시동에 의해서는 직접 호출되지 않는 프로그램의 경우 변환기 단계는 필요하지 않습니다.

2. 인터페이스 모듈 DFHDLIAI는 링크 편집에 의해 자동으로 포함됩니다. 링크 편집 입력에서 INCLUDE문을 사용하는 경우 오브젝트 텍 뒤에 배치하십시오.
3. 프로그램에 카피북 DLIUIB를 포함시키십시오.

4. SYSLMOD DD문이 정의한 링크 편집의 로드 모듈 출력을
CICSTS53.CICS.SDFHLOAD 또는 사용자 정의 애플리케이션 라이브러리에 배치
하십시오.

EXEC DLI 명령을 사용하는 일괄처리 또는 BMP 프로그램

1. 변환기 옵션 DLI는 필수입니다. 변환기 옵션 CICS를 지정하지 마십시오.
2. 인터페이스 모듈에 대한 INCLUDE문은 SYSLIN DD문이 정의한 링크 편집에 대한 입력에서 오브젝트 텍스트 다음에 와야 합니다. IMS.RESLIB에 상주하는 인터페이스 모듈 DFSLI000은 모든 프로그래밍 언어에 동일합니다. SYSLIB DD문이 정의한 링크 편집에 대한 입력에 CICSTS53.CICS.SDFHLOAD를 포함시키는 경우, IMS.RESLIB 뒤에 연결하십시오.
3. SYSLMOD DD문이 정의한 링크 편집의 로드 모듈 출력을 IMS.PGMLIB 또는 일괄처리 작업 스트림의 STEPLIB DD문에 연결된 라이브러리에 배치하십시오.

DL/I CALL 명령을 사용하는 일괄처리 또는 BMP 프로그램

DL/I CALL 인터페이스를 사용하는 어셈블러, COBOL 또는 PL/I 프로그램을 준비하려면 CICS 제공 프로시저를 사용하지 마십시오. CALL ASMTDLI, CALL CBLTDLI 또는 CALL PLITDLI를 포함하는 프로그램은 IMS 애플리케이션으로 어셈블 또는 컴파일, 링크 편집해야 하며, CICS 요구사항의 영향을 받지 않습니다. DL/I CALL 인터페이스를 사용하는 애플리케이션 준비 방법에 대한 정보는 관련 IMS 매뉴얼을 참조하십시오.

제 10 장 맵핑 세트 및 파티션 세트 설치

맵핑 세트 및 파티션 세트를 어셈블하고 링크 편집하려면 CICS의 BMS(Basic Mapping Support) 기능을 사용하십시오. BMS 매크로를 사용하여 BMS 맵에서 생성되는 HTML 템플리트를 설치할 수 있습니다.

프로그램이 BMS 맵을 사용하는 경우 맵을 작성해야 합니다. 이를 수행하는 전통적인 방법은 BMS 매크로에서 맵을 코딩하고 이들 매크로를 어셈블링하는 것입니다. 서로 다른 출력 옵션을 사용하여 어셈블리를 두 번 수행합니다.

- 한 어셈블리는 정의 세트를 작성합니다. 적합한 언어 명령문을 사용하여 이들 정의를 프로그램에 복사하며, 이를 통해 맵의 필드를 이름으로 참조할 수 있습니다.
- 두 번째 어셈블리는 프로그램이 실행할 때 사용되는 오브젝트 모듈을 작성합니다.

프로세스가 다음 다이어그램에서 설명됩니다.

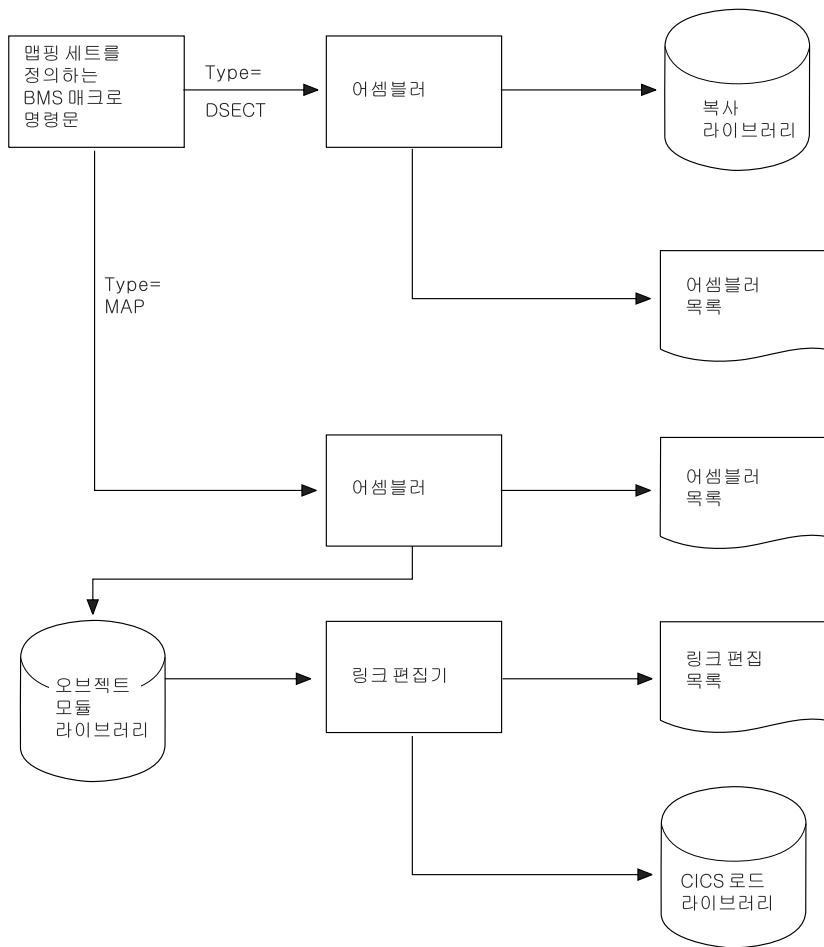


그림 29. 맵 준비

어떤 방법으로 맵을 생성하든지 간에, 해당 맵을 사용하는 임의의 프로그램을 컴파일 (어셈블)하기 전에 맵을 작성해야 합니다. 또한 맵을 변경하는 경우 일반적으로 해당 맵을 사용하는 모든 프로그램을 재컴파일(리어셈블)해야 합니다. 일부 변경사항은 물리적 맵핑에만 영향을 주고 프로그램이 사용하는 대응하는 기호 맵핑에는 반영되지 않습니다. 이들 중 하나는 필드의 순서를 변경하지 않는 필드 위치의 변경입니다. 그러나 데이터 유형, 필드 길이, 필드 순서 및 기타에서의 변경사항은 기호 맵핑에 영향을 주고, 항상 재컴파일(리어셈블)하는 것이 가장 안전합니다.

CICS는 또한 IBM Screen Definition Facility II(SDF II) 같은 라이선스가 있는 프로그램을 사용하여 대화식으로 BMS 맵핑 세트 및 파티션 세트의 정의를 지원합니다. SDF II에 대한 자세한 정보는 Screen Definition Facility II Library의 내용을 참조하십시오.

맵핑 세트 및 파티션 세트에 대한 상주 모드를 링크 편집 단계에서 RMODE(ANY)로서 지정하는 경우 CICS는 16MB 라인 이상의 BMS 맵핑 세트 및 파티션 세트를 로드합니다. 이전 릴리스의 CICS의 맵핑 세트 또는 파티션 세트 중 하나를 사용 중인 경

우, RMODE(ANY)를 사용하여 다시 링크 편집하여 16MB 라인 이상으로 로드할 수 있습니다. RMODE(ANY)를 지정하는 링크 편집 단계의 예제에 대해서는 이 섹션의 샘플 작업 스트림을 참조하십시오.

이 섹션에는 다음이 포함되어 있습니다.

- 『맵핑 세트 설치』
- 194 페이지의 『파티션 세트 설치』
- 196 페이지의 『CICS에 프로그램, 맵핑 세트 및 파티션 세트 정의』

관련 개념:

911 페이지의 제 45 장 『BMS(Basic Mapping Support)』

BMS(Basic Mapping Support)는 CICS 프로그램과 터미널 디바이스 사이의 API(Application Programming Interface)입니다.

관련 태스크:

192 페이지의 『DFHMAPT 프로시저를 사용하여 BMS 맵으로부터 HTML 템플릿 설치』

DFHMAPT 프로시저는 DFHMAPS와 비슷하며, BMS 맵으로부터 생성된 HTML 템플릿을 설치하는 추가 단계를 갖습니다.

맵핑 세트 설치

물리적 맵핑 세트 및 기호 설명 맵핑 세트를 별도로 및 함께 설치하는 방법을 배우려면 이들 예제를 사용하십시오.

이 섹션은 먼저 맵핑 세트의 유형, 이들을 정의하는 방법 및 CICS가 인식하는 방법을 설명합니다. 이 뒤에 물리적 맵핑 세트와 기호 설명 맵핑 세트를 별도로 준비하는 방법의 설명이 뒤따릅니다. 마지막으로, 하나의 작업에서 물리적 및 기호 설명 맵핑 세트를 둘 다 준비하는 방법의 설명이 있습니다. 이들 설명에서는 SYSPARM 매개변수가 맵핑 세트의 두 유형을 구분하는 데 사용된다고 가정합니다.

다음은 참조하십시오.

- 『맵핑 세트의 유형』
- 187 페이지의 『물리적 맵핑 세트 설치』
- 189 페이지의 『기호 설명 맵핑 세트 설치』
- 191 페이지의 『물리적 및 기호 설명 맵을 함께 설치』

맵핑 세트의 유형

맵핑 세트를 설치하려면 물리적 맵핑 세트와 기호 설명 맵핑 세트의 두 유형의 맵핑 세트를 준비해야 합니다.

- 물리적 맵핑 세트는 BMS가 데이터를 애플리케이션이 사용하는 양식과 독립적인 표준 디바이스에서 터미널에서 필요한 디바이스 종속 양식으로 변환하는 데 사용됩니다.
- 기호 설명 맵핑 세트는 애플리케이션에서 사용자 데이터의 표준 디바이스 독립 양식을 정의하는 데 사용됩니다. 이것은 어셈블리 언어에서는 DSECT, COBOL에서는 데이터 정의, PL/I에서는 BASED 또는 AUTOMATIC 구조, C/370에서는 『struct』입니다.

물리적 맵핑 세트는 CICS 로드 라이브러리에서 카탈로그화되어야 합니다. 기호 설명 맵핑 세트는 사용자 사본 라이브러리에서 카탈로그화되거나, 애플리케이션 자체에 직접 삽입될 수 있습니다.

맵핑 세트 정의 매크로는 두 번 어셈블됩니다. 형식화 활동에서 BMS가 사용하는 물리적 맵핑 세트를 생성하기 위해 한 번, 애플리케이션으로 복사되는 기호 설명 맵핑 세트를 생성하기 위해 한 번 어셈블됩니다.

필요한 맵핑 세트 유형 정의

맵핑 세트를 어셈블링하는 데 사용되는 작업의 EXEC 문에서 SYSPARM 피연산자 또는 DFHMSD 매크로의 TYPE 피연산자를 사용하여 두 가지 유형의 맵핑 세트를 구별할 수 있습니다.

이 용도로 SYSPARM 피연산자를 사용하는 경우에는 DFHMSD 매크로의 TYPE 피연산자는 무시됩니다. SYSPARM을 사용하면 물리적 맵핑 세트와 기호 설명 맵핑 세트 둘 모두를 BMS 맵핑 세트 정의 매크로의 동일한 변경되지 않은 세트에서 생성할 수 있습니다.

맵핑 세트는 비정렬 또는 정렬 맵으로서 어셈블될 수 있습니다. 정렬된 맵은 길이 필드가 반자 경계에서 정렬되어 있는 맵입니다. 애플리케이션 패키지가 정렬된 맵을 사용하기 위해 필요한 경우를 제외하고는 비정렬 맵을 사용하십시오.

SYSPARM 값은 단독은 맵핑 세트가 정렬되어 있거나 비정렬되어 있는지 여부를 판별하고 EXEC PROC=DFHMAPS 문에 지정되어 있습니다. SYSPARM 피연산자는 물리적 맵핑 세트 또는 기호 설명 맵핑 세트(DSECT)가 어셈블되는지 여부를 지정하는 데 사용할 수도 있습니다. 이 경우에는 TYPE 피연산자를 덮어씁니다. 피연산자가 둘 모두 지정되지 않은 경우에는 비정렬 DSECT가 생성됩니다.

DFHMSD 매크로의 TYPE 피연산자는 물리적 또는 기호 설명 맵핑 세트가 필요한지 여부만을 정의할 수 있습니다.

가능한 피연산자 조합이 다양한 유형의 맵핑 세트를 생성하게 하려면 187 페이지의 표 13의 내용을 참조하십시오.

표 13. 맵 어셈블리의 SYSPARM 및 DFHMSD 피연산자 조합

맵핑 세트 유형	EXEC DFHMAPS 문의 SYSPARM 피연산자	DFHMSD 매크로의 TYPE 피연산자
정렬된 기호 설명 맵 세트(DSECT)	A A ADSECT	지정되지 않은 DSECT 임의의(SYSPARM 사용)
정렬된 물리적 맵핑 세트	A AMAP	MAP 임의의(SYSPARM 사용)
정렬되지 않은 기호 설명 맵 세트(DSECT)	지정되지 않은 지정되지 않은 DSECT	지정되지 않은 DSECT 임의의(SYSPARM 사용)
정렬되지 않은 물리적 맵핑 세트	지정되지 않은 MAP	MAP 임의의(SYSPARM 사용)

물리적 맵핑 세트는 정렬 또는 비정렬 맵에 대해 어셈블되었는지 여부를 나타냅니다. 이 정보는 실행 시간에 테스트되고 적합한 맵 정렬이 사용됩니다. 따라서 정렬 및 비정렬 맵핑 세트를 혼합할 수 있습니다.

확장 데이터 스트림 터미널 사용

물리적 맵핑 세트를 리어셈블하여 수정된 확장 데이터 스트림 속성을 사용하거나, 동적 속성 수정의 경우 물리적 및 기호 설명 맵핑 세트 둘 모두를 리어셈블하여 사용할 수 있습니다.

3270 정보 표시 시스템에 설계된 애플리케이션 및 맵은 디바이스에서 변경되지 않은 상태로 실행되어 색상, 확장 강조표시, 프로그래밍된 기호 및 유효성 검사 등과 같은 확장을 3270 데이터 스트림에 지원합니다. 색상 등과 같은 수정된 확장 속성을 사용하려면 물리적 맵핑 세트만을 리어셈블하면 됩니다. 애플리케이션에 의한 동적 속성 수정이 필요한 경우에는 물리적 및 기호 설명 맵핑 세트 둘 모두를 리어셈블해야 하고 애플리케이션을 리어셈블하거나 리컴파일해야 합니다.

물리적 맵핑 세트 설치

이들 예제는 물리적 맵핑 세트 및 예제 작업 스트림 설치를 위한 어셈블러 및 연계 편집기 단계를 보여줍니다.

188 페이지의 그림 30은 물리적 맵핑 세트 설치를 위한 어셈블러 및 연계 편집기 단계를 보여줍니다.

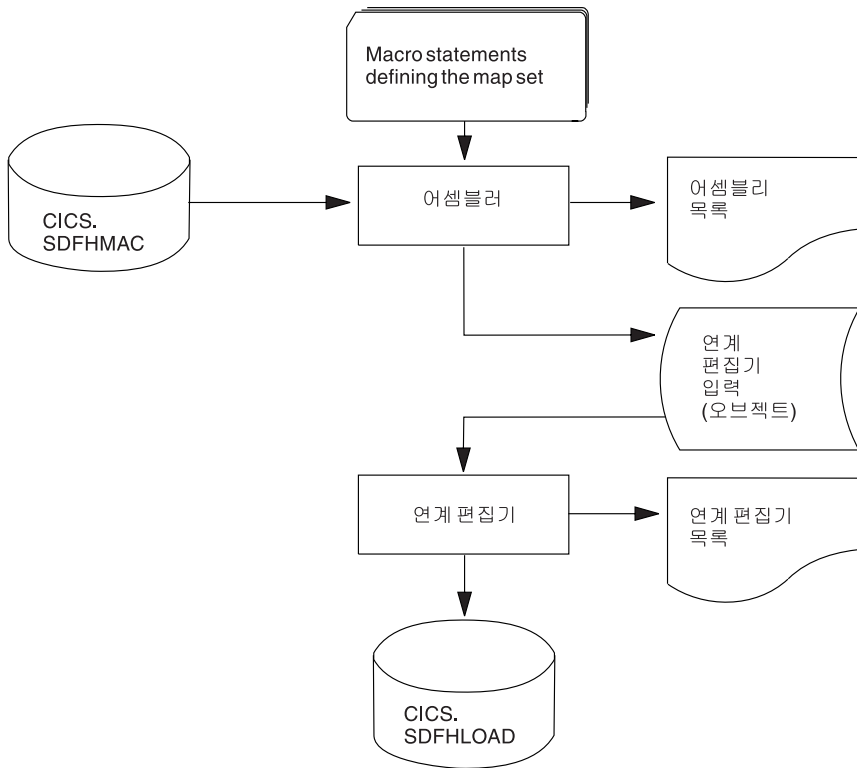


그림 30. 물리적 맵핑 세트 설치

그림 31은 물리적 맵핑 세트의 어셈블리 및 링크 편집을 위한 작업 스트림 예제를 제공합니다.

```
//PREP    JOB  'accounting information',CLASS=A,MSGLEVEL=1
//STEP1   EXEC PROC=DFHASMVS,PARM.ASSEM='SYSPARM(MAP)'          1
//SYSPUNCH DD  DSN=&&TEMP,DCB=(RECFM=FB,BLKSIZE=2960),
//          SPACE=(2960,(10,10)),UNIT=SYSDA,DISP=(NEW,PASS)
//SYSIN    DD  *
```

Macro statements defining the map set

```
/*
//STEP2   EXEC PROC=DFHLNKVS,PARM='LIST,LET,XREF'              2
//SYSLIN  DD  DSN=&&TEMP,DISP=(OLD,DELETE)
//          DD  *
//          MODE RMODE(ANY|24)                                  3
//          NAME mapsetname(R)                                  4
/*
//
```

그림 31. 물리적 맵핑 세트 어셈블리 및 링크 편집

참고

1. 반어 정렬된 길이 필드의 경우 SYSPARM(MAP) 대신 SYSPARM(AMAP) 옵션을 지정하십시오.

2. 물리적 맵핑 세트는 RMODE(ANY) 및 RENT 옵션을 사용하여 링크 편집되지 않는 한 CICS 키 저장영역으로 로드됩니다. 이들 옵션으로 링크 편집되는 경우, RENTPGM=PROTECT가 RENTPGM 초기화 매개변수에서 지정되면 맵핑 세트는 키-0 보호 저장영역으로 로드됩니다. 그러나 맵핑 세트(3270 또는 LU1 디바이스로만 전송되는 것은 예외)는 일부 경우에 CICS가 맵핑 세트를 수정하기 때문에 RENT 또는 REFR 옵션을 사용하여 링크 편집되지 않을 것을 권장합니다. 일반적으로, 3270 또는 LU1 디바이스로만 전송되는 맵핑 세트에 대해 RENT 또는 REFR 옵션을 사용하십시오. CICS에서 사용 가능한 저장영역 보호 기능에 대한 자세한 정보는 성능 개선의 저장영역 보호의 내용을 참조하십시오.
3. MODE문은 맵핑 세트가 16MB 라인 이상(RMODE(ANY)) 또는 이하(RMODE(24))로 로드될지 여부를 지정합니다. RMODE(ANY)는 CICS가 가상 저장영역의 어디에나 맵핑 세트를 로드할 수 있음을 표시하지만, 가능한 경우 16MB 라인 이상에서 로드하려고 합니다.
4. BMS가 저장영역으로 로드하는 물리적 맵핑 세트의 이름을 지정하려면 NAME문을 사용하십시오. 맵핑 세트가 디바이스에 종속되는 경우, 애플리케이션에서 사용되는 원래의 1 - 7자 맵핑 세트 이름에 디바이스 접미부를 추가하여 맵핑 세트 이름을 파생하십시오. CICS BMS가 지원하는 다양한 터미널에 대해 추가될 접미부는 맵핑 세트를 정의하는 데 사용된 DFHMSD 매크로의 TERM 또는 SUFFIX 피연산자에서 지정된 매개변수에 따라 다릅니다.

물리적 맵핑 세트를 사용하려면 그에 대한 자원 정의를 정의 및 설치해야 합니다. 196 페이지의 『CICS에 프로그램, 맵핑 세트 및 파티션 세트 정의』에서 설명하는 대로 프로그램 자동 설치 기능을 사용하거나 **CEDA DEFINE MAPSET** 및 **INSTALL** 명령을 사용하여 이를 수행할 수 있습니다.

기호 설명 맵핑 세트 설치

이들 예제는 DFHASMVS 프로시저를 사용하여 기호 설명 맵핑 세트를 설치하는 단계를 보여줍니다.

기호 설명 맵핑 세트를 사용하면 애플리케이션 프로그래머가 물리적 맵핑 세트의 필드에 대한 기호 참조를 만들 수 있습니다. 190 페이지의 그림 32는 BMS에 대한 기호 설명 맵핑 세트의 준비를 보여줍니다.

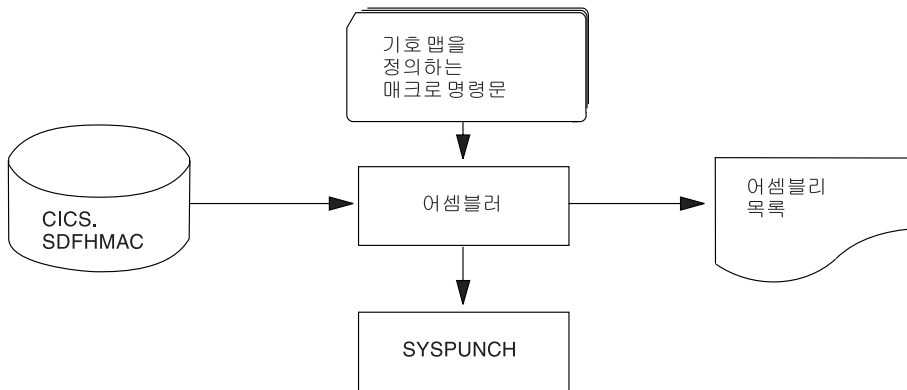


그림 32. DFHASMVS 프로시저를 사용한 기호 설명 맵핑 세트 설치

프로그램에서 기호 설명 맵핑 세트를 사용하려면 맵핑 세트에 대한 소스 명령문을 어셈블하고 SYSPUNCH를 통해 저장영역 정의의 천공된 사본을 얻어야 합니다. 처음 이것이 수행될 때, SYSPUNCH 출력을 SYSOUT=A로 보내서 기호 설명 맵핑 세트의 목록을 얻을 수 있습니다. 사용자 설치에서 많은 맵핑 세트가 사용되거나 공통 맵핑 세트의 다중 사용자가 있는 경우, 사용하는 각 언어에 대한 개인용 사용자 사본 라이브러리를 설정하십시오.

기호 설명이 둘 이상의 프로그래밍 언어에 대해 동일한 이름에서 준비될 때, 기호 설명 맵핑 세트의 별도 사본이 각 사용자 사본 라이브러리에 위치되어야 합니다. 사용자 사본 라이브러리가 SYSLIB와 올바르게 연결되었는지 확인해야 합니다.

물리적 맵핑 세트의 모든 다른 접미부의 버전에 대응하는 단 하나의 기호 설명 맵핑 세트가 필요합니다. 예를 들어 서로 다른 화면 크기를 갖는 터미널에서 동일한 애플리케이션을 실행하려면 다음을 수행합니다.

1. 동일한 필드를 갖지만 화면 크기에 맞게 위치되는 두 개의 맵핑 세트를 정의하십시오. 각 맵핑 세트는 동일한 이름을 갖지만 서로 다른 접미부를 갖는데, 이것은 터미널에 대해 지정된 접미부와 일치합니다.
2. 서로 다른 물리적 맵핑 세트를 개별적으로 어셈블하고 링크 편집하지만, 기호 설명 맵핑 세트가 모든 물리적 맵핑 세트에 대해 동일하기 때문에 단 하나의 기호 설명 맵핑 세트를 작성하십시오.

191 페이지의 그림 33에서 샘플 작업 스트림을 사용하여 기호 설명 맵핑 세트의 목록을 얻을 수 있습니다. CICS가 지원하는 모든 프로그래밍 언어에 적용됩니다.

```
//DSECT    JOB  'accounting information',CLASS=A,MSGLEVEL=1
//ASM      EXEC PROC=DFHASMVS,PARM.ASSEM='SYSPARM(DSECT)'
//SYSPUNCH DD  SYSOUT=A
//SYSIN    DD  *
```

Macro statements defining the map set

```
/*
//
```

그림 33. 기호 설명 맵핑 세트의 목록

길이 필드가 반단어 정렬되는 기호 설명 맵핑 세트를 어셈블하려는 경우, 그림 33의 샘플 작업의 EXEC문을 다음으로 변경하십시오.

```
//ASSEM EXEC PROC=DFHASMVS,PARM.ASSEM='SYSPARM(ADSECT)'
```

기호 설명 맵핑 세트의 천공된 사본을 얻으려면 이전 예제에서 //SYSPUNCH 문을 코딩하여 출력을 천공 데이터 스트림으로 보내십시오. 예를 들어, 다음과 같습니다.

```
//SYSPUNCH DD  SYSOUT=B
```

기호 설명 맵핑 세트를 개인용 사본 라이브러리에 저장하려면 다음과 비슷한 작업 제어문을 사용하십시오.

```
//SYSPUNCH DD  DSN=USER.MAPLIB.ASM(map set name),DISP=OLD
//SYSPUNCH DD  DSN=USER.MAPLIB.COB(map set name),DISP=OLD
//SYSPUNCH DD  DSN=USER.MAPLIB.PLI(map set name),DISP=OLD
```

물리적 및 기호 설명 맵을 함께 설치

이들 예제는 DFHMAPS 프로시저를 사용하여 물리적 및 기호 설명 맵핑 세트를 설치하는 단계를 보여줍니다.

192 페이지의 그림 34는 물리적 및 기호 설명 맵을 함께 설치하기 위한 DFHMAPS 프로시저를 보여줍니다. DFHMAPS 프로시저는 192 페이지의 그림 34에 표시된 다음의 4단계로 구성됩니다.

1. 맵핑 세트에 대해 코딩한 BMS 매크로가 임시 순차 데이터 세트에 추가됩니다.
2. 매크로가 어셈블되어 물리적 맵핑 세트를 작성합니다. MAP 옵션이 EXEC문의 SYSPARM 글로벌 변수에 코딩됩니다(PARM='SYSPARM(MAP)').
3. 물리적 맵핑 세트가 CICS 로드 라이브러리로 링크 편집됩니다.
4. 마지막으로, 매크로가 다시 어셈블되고 이번에는 기호 설명 맵핑 세트를 생성합니다. 이 단계에서, DSECT가 EXEC문의 SYSPARM 글로벌 변수에 코딩됩니다(PARM='SYSPARM(DSECT)'). 출력은 //SYSPUNCH DD 명령문에서 지정된 대상으로 보내집니다. DFHMAPS 프로시저에서 해당 대상은 CICSTS53.CICS.SDFHMAC 라이브러리입니다.

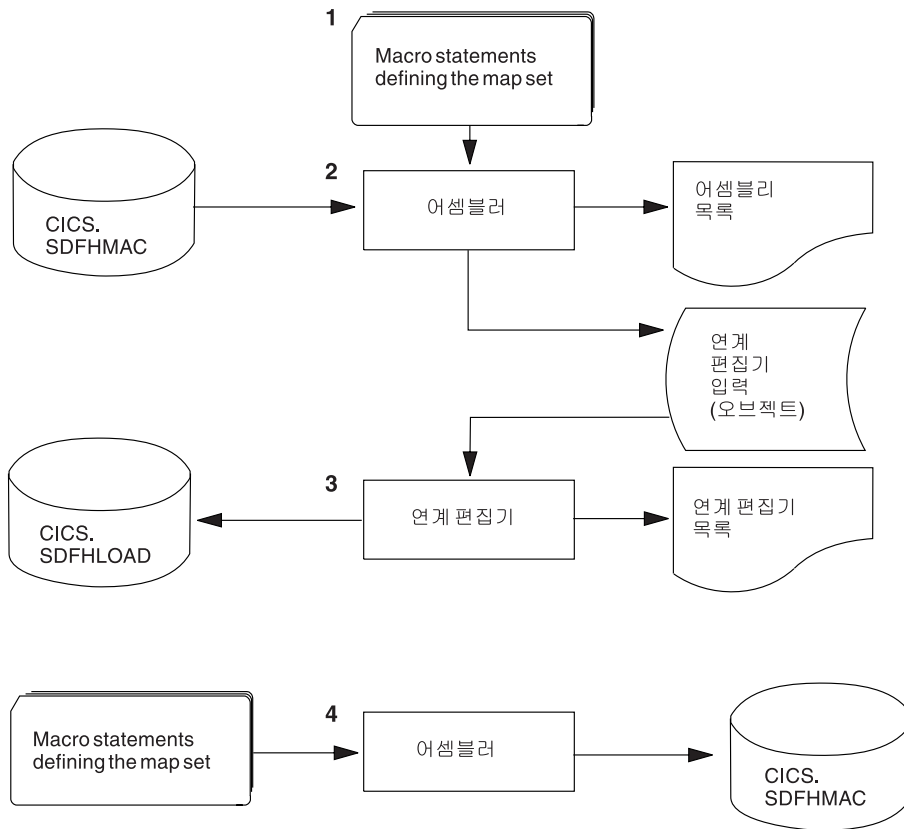


그림 34. 물리적 맵핑 세트 및 기호 설명 맵핑 세트를 함께 설치

DFHMAPT 프로시저를 사용하여 BMS 맵으로부터 HTML 템플릿 설치

DFHMAPT 프로시저는 DFHMAPS와 비슷하며, BMS 맵으로부터 생성된 HTML 템플릿을 설치하는 추가 단계를 갖습니다.

이 태스크 정보

이 단계에서, TEMPLATE는 EXEC문의 SYSPARM 글로벌 변수에서 코딩됩니다 (PARM='SYSPARM(TEMPLATE)'). DFHMAPT 프로시저에서 출력은 CICSTS53.CICS.SDFHHTML로 보내집니다.

사용자 자신의 매크로를 사용하여 HTML 템플릿을 사용자 정의하기 원하고 매크로를 BMS 소스에 추가하지 않으려는 경우 ASMTEMPL 단계를 수정해야 합니다.

1. EXEC문의 PARM 매개변수를 다음과 같이 변경하십시오.

```
PARM='SYSPARM(TEMPLATE,macro_name),DECK,NOOBJECT'
```

2. 매크로를 포함하는 라이브러리를 SYSLIB 연결에 추가하십시오.

물리적 및 기호 설명 맵을 설치하기 위한 JCL

이 예제는 기호 맵핑 세트와 기호 설명 맵핑 세트를 함께 설치하는 데 필요한 JCL 작업 스트림을 보여줍니다.

물리적 맵핑 세트의 어셈블리 및 기호 설명 맵핑 세트의 소스 문으로부터의 로드 모듈은 그림 35에서 샘플 작업 스트림을 사용하여 동일 작업에서 생성될 수 있습니다.

```
//PREPARE JOB 'accounting information',CLASS=A,MSGLEVEL=1
//ASSEM EXEC PROC=DFHMAPS,MAPNAME=mapsetname,RMODE=ANY|24 (see note)
//SYSUT1 DD *
```

Macro statements defining the map set

```
/*
//
```

그림 35. 물리적 및 기호 설명 맵을 함께 설치

참고: RMODE 문은 맵핑 세트가 16MB 라인 이상(RMODE=ANY) 또는 이하(RMODE=24)에 로드되는지 여부를 지정합니다. RMODE=ANY는 CICS는 가상 저장영역의 어느 곳이나 맵핑 세트를 로드할 수 있지만 가능한 경우 16MB 라인 위에 이를 로드하려고 시도합니다.

DFHMAPS 프로시저는 반자 정렬되지 않은 맵핑 세트를 생성합니다. 입력 맵에서 길이 필드가 반자 정렬되기를 원하는 경우에는 EXEC 문에서 A=A를 코딩해야 합니다. 그림 35의 샘플 작업에서 EXEC 문을 변경하여 다음을 수행하십시오.

```
//ASSEM EXEC PROC=DFHMAPS,MAPNAME=mapsetname,A=A
```

이렇게 변경하면 어셈블리 단계에서 SYSPARM 피연산자가 각각 SYSPARM(AMAP) 및 SYSPARM(ADSECT)으로 변경됩니다.

DFHMAPS 프로시저는 기호 설명 맵핑 세트 출력(SYSPUNCH)을 CICSTS53.CICS.SDFHMAC 라이브러리로 지시합니다. EXEC 문에 DSCTLIB=name을 지정하여 이를 덮어쓰십시오. 여기서 『name』은 선택된 사용자 사본 라이브러리의 이름입니다.

맵 어셈블리에 CSECT 추가

이 예제는 맵 어셈블리에 CSECT 이름 및 AMODE 및 RMODE 문 둘 모두를 추가하는 방법을 보여줍니다.

CSECT로 BMS 맵을 생성해야 할 수도 있습니다. 예를 들어, 맵이 16MB 이상에 상주하는지 확인하기 위해 AMODE 및 RMODE 옵션을 지정해야 하거나 변경 관리의 이유로 DFSMS 바인더 IDENTIFY 문을 사용해야 할 수도 있습니다. 이 경우 BMS 매크로 문의 앞에 적합한 CSECT를 포함해야 할 뿐만 아니라, CSECT 문이 기호 설

명 맵에 포함되지 않도록 하기 위해 몇몇 조건부 어셈블리 문을 추가해야 합니다. 다음 예제는 CSECT 이름 및 AMODE 및 RMODE 문 둘 모두를 추가하는 방법을 보여줍니다.

```
//PREPARE JOB 'accounting information',CLASS=A,MSGLEVEL=1
//ASSEM EXEC PROC=DFHMAPS,MAPNAME=mapsetname,RMODE=ANY|24
//SYSUT1 DD *
.
AIF ('&SYSPARM' EQ 'DSECT').SKIPSD
AIF ('&SYSPARM' EQ 'ADSECT').SKIPSD
ANYNAME CSECT Binder IDENTIFY requires CSECT name
ANYNAME AMODE 31
ANYNAME RMODE ANY
.SKIPSD ANOP ,
DFH0STM DFHMSD TYPE=DSECT,MODE=INOUT,CTRL=FREEKB,LANG=COBOL, C
TIOAPFX=YES,TERM=3270-2,MAPATTS=(COLOR,HILIGHT), C
DSATTS=(COLOR,HILIGHT)
SPACE
DFH0STM DFHMDI SIZE=(24,80)
.
.
.
SPACE
DFHMSD TYPE=FINAL
END
.
/*
//
```

그림 36. 맵 어셈블리에 CSECT에 추가

파티션 세트 설치

물리적 맵핑 세트와 동일한 방식으로 파티션 세트를 설치할 수 있습니다. 기호 설명 파티션 세트에 대한 개념이 없습니다.

195 페이지의 그림 37의 작업 스트림은 파티션 세트의 어셈블리 및 링크 편집의 예입니다.

```
//PREP      JOB  'accounting information',CLASS=A,MSGLEVEL=1
//STEP1     EXEC PROC=DFHASMVS
//SYSPUNCH DD  DSN=&&TEMP,DCB=(RECFM=FB,BLKSIZE=2960),
//           SPACE=(2960,(10,10)),UNIT=SYSDA,DISP=(NEW,PASS)
//SYSIN     DD  *
.
      Macro statements defining the partition set
.
/*
//STEP2     EXEC PROC=DFHLNKVS,PARM='LIST,LET,XREF'           1
//SYSLIN    DD  DSN=&&TEMP,DISP=(OLD,DELETE)
//          DD  *
          MODE RMODE(ANY|24)                                   2
          NAME partitionsetname(R)                             3
/*
//
```

그림 37. 파티션 세트 어셈블 및 링크 편집

참고

1. 파티션 세트는 RMODE(ANY) 및 RENT 옵션을 사용하여 링크 편집되지 않는 한 CICS 키 저장영역에 로드됩니다. 이들 옵션을 사용하여 링크 편집되는 경우, RENTPGM=PROTECT가 RENTPGM 시스템 초기화 매개변수에서 지정되는 경우 키-0 보호 설정된 저장영역에 로드됩니다.

CICS에서 사용 가능한 저장영역 보호 기능에 대한 자세한 정보는 성능 개선의 저장영역 보호의 내용을 참조하십시오.

2. MODE문은 파티션 세트가 16MB 라인 이상(RMODE(ANY)) 또는 이하(RMODE(24))로 로드될지 여부를 지정합니다. RMODE(ANY)는 CICS가 가상 저장영역의 어디에나 파티션 세트를 로드할 수 있음을 표시하지만, 가능한 경우 16MB 라인 이상에서 로드하려고 합니다.
3. BMS가 저장영역으로 로드하는 파티션 세트의 이름을 지정하려면 NAME문을 사용하십시오. 파티션 세트가 디바이스에 종속되는 경우, 애플리케이션에서 사용되는 원래의 1 - 7자 파티션 세트 이름에 디바이스 접미부를 추가하여 파티션 세트 이름을 파생하십시오. BMS가 다양한 터미널에 대해 추가하는 접미부는 파티션 세트를 정의한 DFHPSD 매크로의 SUFFIX 피연산자에서 지정된 매개변수에 따라 다릅니다.

파티션 세트 접미부의 전체 세트를 제공하는 프로그래밍 정보에 대해서는 CICS 애플리케이션 프로그래밍 참조서를 참조하십시오.

파티션 세트를 사용하려면 그에 대한 자원 정의를 정의하고 설치해야 합니다. CICS 자원 정의 안내서에서 설명하는 대로 프로그램 자동 설치 기능을 사용하거나 CEDA DEFINE PARTITIONSET 및 INSTALL 명령을 사용하여 이를 수행할 수 있습니다.

CICS에 프로그램, 맵핑 세트 및 파티션 세트 정의

CICS 시동 JCL에서 지정된 로드 라이브러리 중 하나에서 설치한 프로그램을 사용할 수 있으려면 프로그램 및 프로그램이 사용하는 모든 맵핑 세트 및 파티션 세트가 CICS에 정의되어야 합니다. 이를 수행하기 위해 CICS는 자원 정의 MAPSET(맵핑 세트의 경우), PARTITIONSET(파티션 세트의 경우), PROGRAM(프로그램의 경우)을 사용합니다.

이 태스크 정보

다음 방법 중 하나로 그런 자원 정의를 작성 및 설치할 수 있습니다.

- CICS는 처음 로드될 때 프로그램 함수에 대한 자동 설치를 사용하여 프로그램, 맵핑 세트 또는 파티션 세트에 대한 정의를 동적으로 작성, 설치 및 카탈로그화할 수 있습니다.
- 프로그램, 맵핑 세트 또는 파티션 세트에 대한 특정 자원 정의를 작성하고 해당 자원 정의를 CICS 리전에 설치할 수 있습니다.

다음 방법 중 하나로 자원 정의를 설치할 수 있습니다.

- CICS 초기화 시에, GRPLIST 시스템 초기화 매개변수에서 지정되는 그룹 목록에 자원 정의 그룹을 포함.
- CICS가 실행 중인 동안, CEDA INSTALL 명령에 의해.

CICS에 프로그램 정의에 대한 정보는 *CICS 자원 정의 안내서*의 내용을 참조하십시오.

제 11 장 애플리케이션 테스트

다음 방법으로 CICS 애플리케이션을 테스트할 수 있습니다. 이 지침은 Java 애플리케이션 테스트와는 관련이 없습니다.

단일 스레드 테스트

단일 스레드 테스트는 한 번에 한 개의 애플리케이션 트랜잭션을 사용하고 그렇지 않으면 『비어 있는』 CICS 시스템을 사용한 후 어떻게 작동하는지 확인합니다. 이를 사용하면 프로그램 로직을 테스트하고 기본 CICS 정보(예: 자원 정의)가 올바른지 확인할 수도 있습니다. 정상 온라인 프로덕션 CICS 시스템이 다른 리전에서 활성화되어 있는 동안 임의의 CICS 리전에서 이 단일 애플리케이션을 테스트할 수도 있습니다.

멀티스레드 테스트

멀티스레드 테스트에는 몇 개의 동시 활성 트랜잭션이 사용됩니다. 일반적으로 모든 트랜잭션은 동일한 CICS 리전에 있기 때문에 새 트랜잭션 기능이 이들과 공존하도록 간편하게 테스트할 수 있습니다.

단일 스레드 테스트에서 완벽하게 작동하는 트랜잭션이 멀티스레드 테스트에서 실패하는 것을 확인할 수도 있습니다. 또는, 다른 트랜잭션이 실패하거나 CICS가 종료될 수도 있습니다.

회귀 테스트

회귀 테스트는 시스템의 모든 트랜잭션이 시스템에 변경을 적용하기 전후 모두에서 동일하게 처리되는지 확인하기 위해 사용됩니다. 그렇게 해서 한 개의 문제를 해결하기 위해 적용되는 모든 수정사항이 추후 문제를 일으키지 않도록 합니다. 변경사항에 대해 적은 데이터 파일을 평가하는 것이 더 쉽기 때문에 테스트를 수행할 작은 파일 세트를 빌드하는 것은 좋은 생각입니다.

좋은 회귀 테스트는 모든 프로그램의 모든 코드를 실행합니다. 즉, 모든 테스트 및 가능한 조건을 탐색합니다. 시스템이 더 많은 트랜잭션, 더 많은 가능한 조건 등을 포함하도록 개발되면 이를 테스트 시스템에 추가하여 적절하게 유지되도록 하십시오. 각 테스트 결과는 이전 테스트 라운드와 동일해야 합니다. 모든 불일치가 문제의 가능성이 됩니다. 터미널 출력, 파일 변경, 로그 항목의 유효성을 비교할 수 있습니다.

순차 터미널 지원(747 페이지의 제 31 장 『순차 터미널 지원 사용』에서 설명)은 회귀 테스트에 유용할 수 있습니다. 예전에 작동했으며 지금은 수정 중인 모듈이 있다면 이전 테스트를 다시 실행하여 기능이 여전히 작동하는지 확인해야 합니다. 순차 터미널 지원이 이전 테스트케이스 『라이브러리』를 유지보수하고 필요할 때 이를 다시 실행하기 쉽도록 해줍니다.

순차 터미널 지원을 사용하면 전자 통신 장비를 사용하지 않고도 프로그램을 테스트할 수 있습니다. 시스템 프로그래머는 순차 디바이스가 터미널로 사용되도록(터미널 제어 테이블(TCT) 사용) 지정할 수 있습니다. 이 순차 디바이스는 카드 리더, 라인 프린터, 디스크 장치 또는 자기 테이프 장치가 될 수 있습니다. 또한 다음과 같은 순차 디바이스 조합일 수도 있습니다.

- 카드 리더 및 라인 프린터(CRLP)
- 입력으로 하나 이상의 디스크 또는 테이프 데이터 세트
- 출력으로 하나 이상의 디스크 또는 테이프 데이터 세트

프로그램 모듈에 대한 기본 테스트를 수행하기 위해 트랜잭션 테스트케이스 스트림을 준비할 수 있습니다. 테스트가 진행되는 동안 일반적으로 추가 트랜잭션 스트림을 생성하여 프로그램의 멀티프로그래밍의 유효성을 검증하거나 트랜잭션 테스트케이스가 동시에 실행되도록 할 수 있습니다.

애플리케이션을 테스트하고 디버그하기 전에 두 개의 기본 태스크를 수행해야 합니다.

1. 『애플리케이션의 테스트 준비』
2. 199 페이지의 『테스트에 대한 시스템 준비』

애플리케이션의 테스트 준비

이 목록에서는 애플리케이션 및 시스템 테이블 항목을 준비할 때 고려해야 할 것을 보여줍니다.

1. 각 프로그램을 변환, 어셈블 또는 컴파일한 후 링크 편집합니다. 테스트를 시작하기 전에 프로그램에 대해 세 단계 중 임의 단계에서 오류 메시지가 없는지 확인하십시오.
2. 실행 진단 기능(EDF) 화면과 함께 변환기 명령문 번호를 사용할 수 있도록, 변환기 단계에서 DEBUG 및 EDF 옵션을 사용합니다.
3. 덤프 및 EDF 표시장치의 저장영역 위치를 원래 COBOL 소스 명령문과 관련시키고, 작업 중인 저장영역에서 변수를 찾을 수 있도록 COBOL 컴파일러 옵션 CLIST 및 DMAP를 사용합니다.
4. 사용할 트랜잭션에 대해 PROFILE 자원 정의를 작성하고 정의가 설치되어 있는지 확인합니다.
5. 애플리케이션에서 트랜잭션마다 TRANSACTION 자원 정의를 작성하고 정의가 설치되어 있는지 확인합니다.
6. 시스템이 프로그램 자동 설치를 사용하지 않는 경우, 애플리케이션에서 사용되는 프로그램마다 PROGRAM 자원 정의를 작성하고, 정의가 설치되어 있는지 확인합니다.
7. 시스템이 프로그램 자동 설치를 사용하지 않는 경우, 애플리케이션에서 설정된 맵마다 MAPSET 자원 정의를 작성하고, 각 정의가 설치되어 있는지 확인합니다.

8. 사용되는 파일마다 FILE 자원 정의를 작성하고 각 정의가 설치되어 있는지 확인합니다.
9. 최소한 필수 파일 각각의 테스트 버전에서 빌드합니다.
10. 애플리케이션에서 사용될 트랜지언트 데이터 큐 처리 각각을 정의합니다.
11. 애플리케이션에서 사용되는 파일마다 시동 작업 스트림에 작업 제어 DD 카드를 놓습니다.
12. 일부 테스트 데이터를 준비합니다.

테스트에 대한 시스템 준비

이 목록은 디버깅을 위해 시스템을 준비할 때 고려해야 할 사항을 보여줍니다.

1. GRPLIST 시스템 초기화에서 지정하는 목록에 그룹 DFHEDF를 포함시켜서, 사용자 시스템에서 EDF가 사용 가능한지 확인하십시오.
2. 애플리케이션에 적절한 추적 옵션을 설정하십시오. 추적 옵션 설정에 대한 세부사항은 *CICS 문제점 판별 안내서*의 내용을 참조하십시오.
3. 트랜잭션 덤프가 모든 트랜잭션 덤프 코드에 대해 사용 가능하고, 시스템 덤프가 모든 시스템 덤프 코드에 대해 사용 가능한지 확인하십시오. 이 덤프는 기본 설정입니다. 덤프 옵션 설정에 대한 정보는 *CICS 문제점 판별 안내서*의 내용을 참조하십시오.
4. 덤프 인쇄 준비가 되어 있는지 확인하십시오. DFHDU700 작업 스트림 또는 프로시저가 준비되어 있고, CICS 덤프 데이터 세트가 시동 프로시저에 정의되어 있도록 하십시오.
5. 시스템에서 사용 가능한 SDUMP 데이터 세트와 처리를 위한 JCL에 대한 액세스 관련 정보는 시스템 프로그래머에게 문의하십시오.
6. SIT에서 **ICVR** 매개변수를 0보다 큰 숫자로 설정하여 CICS가 루프를 발견할 수 있도록 하십시오. 5 -10초(ICVR=5000 - ICVR=10000) 사이의 값은 일반적으로 작동 가능한 값입니다.
7. 통계를 생성하십시오. 통계 사용에 대한 자세한 정보는 모니터링의 CICS 통계의 내용을 참조하십시오.

제 12 장 진단 기능 실행(EDF)

실행 진단 기능(EDF)을 사용하여 프로그램이나 프로그램 준비 프로시저를 수정하지 않고도 애플리케이션을 온라인으로 테스트할 수 있습니다. CICS 실행 진단 기능은 DFHEDFP 프로그램을 호출하는 CICS 제공 트랜잭션인 CEDF로 지원됩니다.

참고: 디버그하려는 트랜잭션 이름을 지정할 수 있도록 하는 다른 CICS 제공 트랜잭션인 CEDX를 통해 CEDF를 간접 호출할 수도 있습니다. 이 섹션에서 CEDF 트랜잭션을 참조하는 경우(예를 들어, CICS가 아래의 새 CEDF 태스크 시작에 대해 설명하는 경우) 이는 CEDX 명령으로 호출될 수도 있음을 기억하십시오.

『DFH』 접두부는 CICS 시스템 모듈 및 샘플에 사용되기 때문에 프로그램 이름은 이 접두부로 시작되면 안됩니다. CICS 제공 트랜잭션에서 EDF를 사용하려고 해도 아무런 효과가 없습니다. 그렇지만 CICS 샘플 프로그램 및 일부 사용자 대체 가능 모듈에는 EDF를 사용할 수 있습니다. (예를 들어, DFHPEP 디버깅에 EDF를 사용할 수 있습니다.)

EDF는 애플리케이션의 다양한 위치에서 CICS 명령 실행을 인터셉트할 수 있으며 어떤 결과가 발생하는지 확인할 수 있습니다. 각 명령은 실행 전에 표시되고 대부분은 실행 완료 후에도 표시됩니다. 애플리케이션에서 전송된 화면은 보존되기 때문에 테스트 중에 사용자가 프로덕션 시스템에서 하는 것처럼 애플리케이션과 대화할 수 있습니다.

트랜잭션이 EDF 제어로 실행되면 EDF는 다음 위치에서 이를 인터셉트하며 사용자는 이에 대해 상호작용할 수 있습니다.

- 프로그램 초기화 중에 실행 인터페이스 블록(EIB)이 갱신된 후 프로그램에 제어가 지정되기 전.
- 각 CICS 명령 실행 시작 시. 이 인터럽트는 초기 추적 항목이 작성된 후 명령이 수행되기 전에 발생합니다. 표준 CICS 명령 및 FEPI(Front End Programming Interface) 명령 모두 인터셉트됩니다. EXEC DLI 및 EXEC SQL 명령과 자원 관리자 인터페이스를 통해 처리되는 모든 요청도 이 시점에서 인터셉트됩니다.
- 모든 명령의 실행 종료 시(ABEND, XCTL, RETURN 명령은 제외, 해당 명령은 EDF가 표시하는 오류 조건을 발생시킬 수 있지만 제외). EDF는 트랜잭션이 명령 처리를 종료할 때 이를 인터셉트하며 HANDLE CONDITION 메커니즘 호출 전 및 응답 추적 항목 작성 전에 인터셉트합니다.
- 프로그램 종료 시.
- 정상 태스크 종료 시.
- ABEND가 발생하는 경우 비정상 태스크 종료 후.

예제 EDF에 대해 작업하려면 샘플 EDF 세션에 대해 안내하는 *CICS 애플리케이션 디자인 및 프로그래밍*을 참조하십시오.

참고: NOEDF 옵션으로 변환되는 프로그램의 경우, 이 인터셉트 위치는 각 명령 실행 전후에 상관없이 여전히 적용됩니다. 해당 자원 정의에 또는 프로그램 자동 설치 종료로 CEDF가 NO로 정의된 프로그램의 경우, 프로그램 초기화 및 종료 화면도 표시되지 않습니다.

EDF가 애플리케이션 실행을 인터럽트할 때마다 새 CEDF 태스크가 시작됩니다. 각 CEDF 태스크는 짧으며 적절한 표시장치가 처리될 동안만 유지됩니다.

EDF 상호작용에 사용 중인 터미널은 트랜스미버(ATI/TTI) 상태여야 하며 데이터를 송수신할 수 있어야 합니다. 이는 터미널을 표시하는 가장 공통적인 상태이지만 시스템 프로그래머에게 해당 상태 확인을 요청하여 알아내거나 CEMT를 사용할 수 있습니다.

터미널에서 시작된 트랜잭션의 경우, 테스트 중인 트랜잭션과 동일한 터미널이나 다른 터미널에서 EDF를 사용할 수 있습니다. 동일한 터미널에서 반드시 화면을 지우고 트랜잭션 코드 CEDF를 입력하여 시작해야 하며 그렇지 않으면 예상치 못한 결과가 발생할 수 있습니다. 메시지 **THIS TERMINAL: EDF MODE ON**이 빈 화면 맨 위에 표시됩니다. 화면을 다시 지우고 정상적인 방법으로 트랜잭션을 실행합니다.

EDF를 사용 중인 경우 사용자 태스크는 직접 삭제할 수 없습니다. 태스크를 종료해야 하면 우선 CEDF 태스크를 강제로 삭제하고 EDF 화면이 표시된 동안 Enter 키를 누르십시오. Enter 키를 누르면 응답이 없으며 두 번째에는 CEDF 태스크를 강제로 삭제합니다. CEDF가 종료되고 사용자 트랜잭션은 AED3 이상 종료를 수신합니다.

이 장에서는 다음에 대해 설명합니다.

- 『EDF 사용 시 제한사항』
- 205 페이지의 『EDF에서 표시하는 내용』
- 212 페이지의 『EDF를 사용하여 프로그램 테스트』
- 219 페이지의 『EDF를 사용하여 정보 변경』
- 221 페이지의 『EDF 메뉴 기능 사용』

EDF 사용 시 제한사항

EDF를 사용하여 애플리케이션을 디버그하는 경우, 제한사항 수를 인지해야 합니다.

TCB 및 EDF 열기

프로그램이 일반적으로 OPEN TCB(L8, L9, X8 또는 X9)를 사용하여 실행되더라도 CEDF 자체가 스레드세이프가 아니기 때문에 강제로 프로그램이 QR TCB에서 실행되도록 합니다.

매개변수 목록 스택

CEDF는 EXEC CICS 매개변수 목록의 해당 사본에 대한 한 개의 스택 레벨만 가집니다. 애플리케이션이 EXEC 가능 글로벌 사용자 엑시트 또는 사용자 대체 가능 모듈(URM)을 호출하면 글로벌 사용자 엑시트 또는 URM으로 실행되는 EXEC CICS 명령에 대한 매개변수 목록이 기본 프로그램에서 실행되는 EXEC CICS 명령의 매개변수 목록을 오버레이할 수도 있습니다.

보안 고려사항

EDF는 설치가 접속 시간 보안으로 해당 사용을 제한할 수도 있는 강력한 도구입니다. 설치에서 사용되는 외부 보안 관리자는 EDF 트랜잭션용 보안 속성을 정의합니다. CEDF를 사용할 권한이 없는 경우에는 트랜잭션을 시작할 수 없습니다.

애플리케이션 전제조건

EDF를 사용하여 디버그되는 사용자 애플리케이션은 기본값인 변환기 옵션 EDF를 사용하여 어셈블(또는 컴파일)해야 합니다. NOEDF를 지정하는 경우 프로그램은 EDF를 사용하여 디버그할 수 없습니다. NOEDF를 지정할 때 성능 상의 이점은 없지만 옵션은 EDF 표시장치에 표시되는 이미 디버그된 서브프로그램에서 명령을 방지하는 데 유용할 수 있습니다.

EDF를 사용하여 디버그되는 애플리케이션은 자원 정의에 기본값인 속성 CEDF(YES)도 포함해야 합니다. 프로그램이 CEDF(YES)로 정의되고 변환기 옵션 EDF를 사용하여 컴파일되면 EDF 진단 화면이 프로그램에 대해 표시됩니다. 프로그램이 CEDF(YES)로 정의되지만 변환기 옵션 NOEDF로 컴파일되는 경우, 프로그램 초기화 및 종료 화면만 표시됩니다. CEDF(NO)가 지정되면 EDF 화면이 표시되지 않습니다.

속성 CEDF(NO)가 포함된 프로그램이 속성 CEDF(YES)가 포함된 프로그램에 링크되는 경우, 트랜잭션에 EDF를 사용하지 못할 수도 있습니다. 예를 들어, CICSplex SM 동적 트랜잭션 라우팅 프로그램 EYU9XLOP가 속성 CEDF(NO)로 정의되고 사용자 대체 가능 프로그램 EYU9WRAM(워크로드 관리 처리용)이 속성 CEDF(YES)로 정의되면 EDF를 사용하여 EYU9WRAM을 디버그할 수 없습니다. 트랜잭션 내에서 여러 프로그램을 제대로 디버깅하려면 모든 프로그램이 CEDF(YES)로 정의되도록 하십시오.

단일 화면 모드에 대한 제한사항

한 개의 특정 화면 모드를 사용하도록 권장하거나 반드시 사용하도록 하는 EDF 사용에는 몇 개의 제한사항이 있습니다.

- EDF는 원격 트랜잭션을 실행하는 경우 단일 화면 모드에서만 사용할 수 있습니다.
- VM PASSTHRU는 단일 화면 모드에서 테스트 중인 경우 EDF로 지원되지 않습니다.

- 단일 화면 모드에서, 메시지는 EDF 표시장치에 개입하기 때문에 사용자 트랜잭션 및 CEDF는 메시지 저널링을 지정하면 안됩니다. 메시지 저널링은 각 트랜잭션에 대한 프로파일 정의로 제어됩니다.
- 단일 화면 모드에서는 CEDF 트랜잭션의 프로파일 정의에 PROTECT=YES를 지정하지 마십시오. 이 옵션을 지정하면 CEDF 트랜잭션에 대한 메시지 보호가 무시됩니다. 사용자 트랜잭션은 CEDF에서 실행되는 경우에도 여전히 PROTECT=YES 옵션을 지정할 수 있습니다. 이 제한사항은 이중 화면 모드에는 적용되지 않습니다.
- SEND LAST 명령이 실행되면 단일 화면 모드를 사용 중인 경우 명령이 처리되기 전에 EDF가 종료됩니다.
- 화면 파티션을 사용하거나 자체적인 고유 단위(RU) 체인을 실행 중인 경우 애플리케이션을 테스트하려면 이중 화면 모드에서 실행하십시오.
- 단일 화면 모드에서 사용자 트랜잭션의 프로파일이 INBFMH=ALL 또는 INBFMH=DIP를 지정하면 CEDF용 프로파일이 동일한 INBFMH 값을 포함해야 합니다. 그렇지 않으면 사용자 트랜잭션이 ADIR 이상 종료로 종료됩니다. 이중 화면 모드의 경우 프로파일은 그런 점에서는 일치할 필요가 없습니다.
- 속성 설정 키가 사용되도록 인바운드 응답 모드가 문자로 설정된 경우 EDF는 단일 화면 모드에서 키가 사용되지 않도록 합니다.
- 이중 화면 모드에서 EDF에 CECI를 사용하면 특정 명령(예: ASSIGN 및 ADDRESS)이 트랜잭션 터미널이 아닌 EDF 터미널에 대해 실행됩니다. CEDF에서 CECI를 호출하는 방법에 대한 정보는 INVOKE CECI를 참조하십시오.
- 이중 화면 모드에서 EDF를 사용하는 경우, EDF 터미널에서 두 번째 태스크를 시작하는 것(예: START 명령을 실행)은 삼가하십시오. EDF는 의사 대화식 트랜잭션이기 때문에 두 번째 태스크가 사용 중인 터미널에서 시작되는 것을 방지하지 못합니다. 이로 인해 특정 상황에서 교착상태가 발생할 수 있습니다.
- 이중 화면 모드에서 EDF 화면 억제를 사용하면 장기 대기를 초래하는 명령(예: DELAY, WAIT 또는 두 번째 RECEIVE)으로 인해 EDF가 완료된 것으로 표시될 수도 있습니다. 태스크가 비정상 종료되면 EDF는 터미널 모니터링 시에 다시 활성화됩니다.

두 화면 모드에 대한 제한사항

두 화면 모드에는 기타 제한사항이 적용됩니다.

- 트랜잭션이 FREE 명령을 실행하면 EDF는 경고없이 꺼집니다.
- EDF는 CPI 통신 인터페이스(CPI-C) 또는 SAA 자원 복구 인터페이스(CPI-RR)에 대한 호출을 인터셉트하지 않습니다. EDF에서 CPI 호출을 사용하는 트랜잭션을 테스트할 수는 있지만 호출 위치에서 EDF 표시장치를 확인할 수는 없습니다.
- SIGNON 명령을 처리할 때 CEDF는 실수로 암호 또는 암호 문구 값이 노출되는 것을 줄이기 위해 해당 값이 표시되는 것을 방지합니다.

- 연결에 EDF를 사용하는 경우, 연결을 통해 전달되는 파일 제어 명령 기능은 CEDF 구현으로 인해 표시되지 않습니다.

EDF에서 표시하는 내용

모든 EDF 표시장치는 동일한 일반 형식이 있지만 내용은 태스크가 인터럽트된 지점에 따라 다릅니다. 표시장치는 어떤 인터럽션 지점 중에 어느 것에 도달했는지를 표시하고 해당 지점과 관련된 정보를 표시합니다.

그림 38에서는 SEND MAP 명령의 실행 후에 발생하는 일반적인 표시장치를 보여줍니다.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00032 APPLID: 1234567 DISPLAY:00
STATUS:  COMMAND EXECUTION COMPLETE                                1
EXEC CICS SEND MAP
  MAP ('T1      ')
  FROM ('.....'.....'...')
  LENGTH (154)
  MAPSET ('DFH0T1 ')
  CURSOR                                2
  TERMINAL
  ERASE
  NOFLUSH
  NOHANDLE

  OFFSET:X'002522'   LINE:00673      EIBFN=X'1804'
  RESPONSE: NORMAL      EIBRESP=0    3
ENTER:  CONTINUE      4
F1 : UNDEFINED          F2 : SWITCH HEX/CHAR  F3 : END EDF SESSION
F4 : SUPPRESS DISPLAYS  F5 : WORKING STORAGE  F6 : USER DISPLAY
F7 : SCROLL BACK        F8 : SCROLL FORWARD  F9 : STOP CONDITIONS
F10: PREVIOUS DISPLAY   F11: EIB DISPLAY      F12: ABEND USER TASK
```

그림 38. 일반적 EDF 표시장치

참고: 1헤더 2본문 3메시지 행 4기능의 메뉴

표시장치는 헤더, 본문(기본 표시장치 영역), 메시지 행 및 이 지점에서 선택할 수 있는 기능 메뉴로 구성됩니다. 본문이 한 화면에 맞지 않으면 EDF는 여러 화면을 작성하고, 사용자는 기능 키 F7과 F8을 사용하여 화면이동할 수 있습니다. 헤더, 메뉴 및 메시지 영역은 각 화면에서 반복됩니다.

헤더

표시장치의 헤더에는 실행되는 트랜잭션의 ID, 실행되는 프로그램 이름, 트랜잭션, CICS가 트랜잭션에 지정하는 내부 태스크 번호, 트랜잭션이 실행되는 CICS 리전의 applid, 표시장치 번호, EDF에 의한 인터셉션의 이유인 STATUS가 포함됩니다.

본문

표시장치의 본문 또는 메인 파트에는 인터셉트의 위치에 따라 달라지는 정보가 포함됩니다. 다음 화면에서는 프로그램 초기화 시, 시작 시, CICS 명령 실행 시작 및 종료 시, 프로그램 및 태스크 종료 시, 이상 종료 시 본문 콘텐츠를 표시합니다.

프로그램 시동 시

이 예제는 프로그램 시동 시의 표시장치를 보여줍니다. EDF는 COMMAREA(있는 경우) 및 EIB에 있는 프린시פל 필드의 내용을 표시합니다.

이들 EIB 필드에 대한 프로그래밍 정보는 *CICS 애플리케이션 프로그래밍 참조서*를 참조하십시오. COMMAREA가 없는 경우, 화면의 행 4는 공백으로 남아 있으며 EIBCALEN은 0의 값을 갖습니다.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00032 APPLID: 1234567 DISPLAY:00
STATUS: PROGRAM INITIATION
```

```
COMMAREA      = '3476559873'
EIBTIME       = 92920
EIBDATE       = 91163
EIBTRNID      = 'AC20'
EIBTASKN      = 32
EIBTRMID      = 'S246'

EIBCPOSN      = 4
EIBCALEN      = 10
EIBAID        = X'7D'                AT X'032F059A'
EIBFN         = X'0000'              AT X'032F059B'
EIBRCODE      = X'00000000000000'    AT X'032F059D'
EIBDS         = '.....'
+ EIBREQID     = '.....'
```

```
ENTER: CONTINUE
PF1 : UNDEFINED      PF2 : SWITCH HEX/CHAR    PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE    PF6 : USER DISPLAY
PF7 : SCROLL BACK     PF8 : SCROLL FORWARD    PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY       PF12: UNDEFINED
```

그림 39. 프로그램 시동 시의 전형적인 EDF 표시장치

CICS 명령의 실행 시작 시

이 예제는 CICS 명령의 시작 시의 표시장치를 보여줍니다. EDF는 키워드, 옵션, 인수 값을 포함하여 명령을 표시합니다.

CICS 명령의 실행 시작 시 전형적인 EDF 표시장치가 207 페이지의 그림 40에 표시됩니다. PF2를 눌러서 정보를 16진 또는 문자 양식으로 표시(하고 하나의 양식에서 다른 양식으로 전환)할 수 있습니다. 문자 형식이 요청되면 숫자 인수는 부호 있는 숫자 문자 형식으로 표시됩니다.

```

TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00032 APPLID: 1234567 DISPLAY:00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS SEND MAP
MAP ('T1      ')
FROM ('.....'..)
LENGTH (154)
MAPSET ('DFH0T1 ')
CURSOR
TERMINAL
ERASE
NOFLUSH
NOHANDLE

OFFSET:X'002522'   LINE:00673   EIBFN=X'1804'

ENTER: CONTINUE
PF1 : UNDEFINED      PF2 : SWITCH HEX/CHAR    PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE    PF6 : USER DISPLAY
PF7 : SCROLL BACK     PF8 : SCROLL FORWARD    PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY      PF12: ABEND USER TASK
    
```

그림 40. CICS 명령의 실행 시작 시에 전형적인 EDF 표시장치

그림 41은 DB2 버전 2.3과 함께 실행 중인 EXEC SQL 명령의 실행 시작에 대해 비슷한 화면을 보여줍니다.

```

TRANSACTION: LOKO PROGRAM: TLOKO TASK: 00082 APPLID: 1234567 DISPLAY:00
STATUS: ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER DSNCSQL
EXEC SQL UPDATE
DBRM=TLOK0, STMT=00242, SECT=00001
IVAR 001: TYPE=CHAR, LEN=00010 AT X'001E5A99'
DATA=X'F0F0F0F0F0F1F0F0F0F0'

OFFSET:X'000298'   LINE: UNKNOWN EIBFN= X'0A02'
ENTER: CONTINUE
PF1 : UNDEFINED      PF2 : UNDEFINED      PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE    PF6 : USER DISPLAY
PF7 : SCROLL BACK     PF8 : SCROLL FORWARD    PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY      PF12: ABEND USER TASK
    
```

그림 41. SQL 명령의 실행 시작 시에 전형적인 SQL 표시장치

옵션과 값 외에, 명령은 프로그램 내에서 16진 오프셋으로 식별됩니다. 프로그램이 DEBUG 변환기 옵션을 갖고 변환된 경우, 그림 40에서 보는 것처럼 행 번호도 나타납니다. (이 옵션에 대한 정보는 113 페이지의 『변환기 옵션 정의』의 내용을 참조하십시오.)

EXEC SQL 또는 EXEC DLI 명령의 시작 시에, EDF의 본문은 명령이 변환하는 CALL의 매개변수 목록을 표시합니다. DLI 명령이 다중 CALL문을 생성하는 경우 마지막 CALL문만 보게 됩니다.

명령의 실행 종료 시

이 예제는 명령의 실행 종료 시의 표시장치를 보여줍니다. EDF는 명령의 시작에서와 같은 형식으로 표시장치를 제공합니다. 이 시점에서 리턴되거나 변경된 변수의 값과 응답 코드에서 명령 실행의 효과를 볼 수 있습니다.

EDF는 ABEND, XCTL, RETURN 명령의 경우(이들 명령이 EDF가 표시하는 오류 상태를 발생할 수 있다 하더라도) 이 표시장치를 제공하지 않습니다. 207 페이지의 그림 40에 있는 **about to execute** 화면에 대응하는 완료 화면이 그림 42에 표시됩니다.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00054 APPLID: 1234567 DISPLAY:00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS SEND MAP
MAP ('T1      ')
FROM ('.....'...)
LENGTH (154)
MAPSET ('DFH0T1 ')
CURSOR
TERMINAL
ERASE
NOFLUSH
NOHANDLE

OFFSET:X'002522'   LINE:00673           EIBFN=X'1804'
RESPONSE: NORMAL   EIBRESP=0

ENTER: CONTINUE
PF1 : UNDEFINED      PF2 : SWITCH HEX/CHAR    PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE    PF6 : USER DISPLAY
PF7 : SCROLL BACK     PF8 : SCROLL FORWARD    PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY       PF12: ABEND USER TASK
```

그림 42. CICS 명령의 완료 시 전형적인 EDF 표시장치

CICS 명령의 경우, 응답 코드가 이름(예: NORMAL 또는 NOTFND) 및 10진수 양식의 대응하는 EIBRESP 값에 의해 설명됩니다. DL/I의 경우, 응답 코드는 2문자 DL/I 상태 코드이며, EIBRESP 값은 없습니다. EIBRESP 코드를 포함한 프로그래밍 정보는 CICS 애플리케이션 프로그래밍 참조서에 있으며, DL/I 코드는 애플리케이션 프로그래밍: EXEC DLI 명령에서 문서화됩니다.

209 페이지의 그림 43 및 209 페이지의 그림 44는 EXEC DLI 명령의 전형적인 화면을 보여줍니다.

```

TRANSACTION: XDLI PROGRAM: UPDATE TASK: 00111 APPLID: 1234567 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC DLI GET NEXT
USING PCB (+00003)
FIRST
SEGMENT ('A      ')
INTO ('          ')
SEGLNGTH (+00012)
FIRST
VARIABLE
+SEGMENT ('B      ')

OFFSET:X'000246'  LINE: 00000510          EIBFN:X'000C'
RESPONSE: 'AD'

ENTER: CONTINUE
PF1 : UNDEFINED          PF2 : SWITCH HEX/CHAR    PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS  PF5 : WORKING STORAGE    PF6 : USER DISPLAY
PF7 : SCROLL BACK        PF8 : SCROLL FORWARD    PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: EIB DISPLAY      PF12: ABEND USER TASK

```

그림 43. DLI 명령의 완료 시 전형적인 EDF 표시장치(화면 1)

```

TRANSACTION: XDLI PROGRAM: UPDATE TASK: 00111 APPLID: 1234567 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC DLI GET NEXT
+
FIRST
SEGMENT ('C          ')
SEGLNGTH (+00010)
LOCKED
INTO ('SMITH      ')
WHERE (ACCOUNT = '12345')
FIELDLENGTH (+00005)

OFFSET:X'000246'  LINE: 00000510          EIBFN:X'000C'
RESPONSE: 'AD'

ENTER: CONTINUE
PF1 : UNDEFINED          PF2 : SWITCH HEX/CHAR    PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS  PF5 : WORKING STORAGE    PF6 : USER DISPLAY
PF7 : SCROLL BACK        PF8 : SCROLL FORWARD    PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: EIB DISPLAY      PF12: ABEND USER TASK

```

그림 44. DLI 명령의 완료 시 전형적인 EDF 표시장치(화면 2)

```

TRANSACTION: LOKO PROGRAM: TLOKO TASK: 00111 APPLID: 1234567 DISPLAY: 00
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER DSNCSQL
EXEC SQL UPDATE
PLAN=TLOKO,   DBRM=TLOKO,   STMT=00242, SECT=00001
SQL COMMUNICATION AREA:
SQLCABC      = 136                      AT X'001E5A18'
SQLCODE      = 000                      AT X'001E5A1C'
SQLERRML     = 000                      AT X'001E5A20'
SQLERRMC     = ''                      AT X'001E5A22'
SQLERRP      = 'DSN'                   AT X'001E5A68'
SQLERRD(1-6) = 000, 000, 00001, -1, 00000, 000 AT X'001E5A70'
SQLWARN(0-A) = '-----'              AT X'001E5A88'
SQLSTATE     = 00000                    AT X'001E5A93'

OFFSET:X'000298'  LINE: UNKNOWN EIBFN= X'0A02'
RESPONSE:

ENTER:  CONTINUE
PF1 : UNDEFINED      PF2 : UNDEFINED      PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE  PF6 : USER DISPLAY
PF7 : SCROLL BACK    PF8 : SCROLL FORWARD  PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY      PF12: ABEND USER TASK

```

그림 45. SQL 명령의 완료 시 전형적인 SQL 표시장치

프로그램 및 태스크 종료 시

이 예제는 프로그램 종료 및 일반 태스크 종료 시의 표시장치를 보여줍니다. 본문 정보는 없습니다. 즉, 모든 관련 정보가 헤더에 있습니다.

그림 46 및 211 페이지의 그림 47은 프로그램 및 태스크 종료에 대한 요약된 화면을 보여줍니다.

```

TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00054 APPLID: 1234567 DISPLAY:00
STATUS:  PROGRAM TERMINATION

ENTER:  CONTINUE
PF1 : UNDEFINED      PF2 : SWITCH HEX/CHAR  PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE  PF6 : USER DISPLAY
PF7 : SCROLL BACK    PF8 : SCROLL FORWARD  PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY      PF12: ABEND USER TASK

```

그림 46. 프로그램 종료 시 전형적인 EDF 표시장치

TRANSACTION: AC20 TASK: 00054 APPLID: 1234567 DISPLAY: 00
STATUS: TASK TERMINATION

CONTINUE EDF? (ENTER YES OR NO) REPLY: YES
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: UNDEFINED

그림 47. 태스크 종료 시 전형적인 EDF 표시장치

비정상 종료 시

이 예제는 이상 종료 또는 비정상 태스크 종료가 발생할 때의 표시장치를 보여줍니다.

이상 종료 또는 비정상 태스크 종료가 발생할 때, EDF는 그림 48 및 212 페이지의 그림 49에 표시된 화면을 표시합니다.

TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK:00054 APPLID: 1234567 DISPLAY: 00
STATUS: AN ABEND HAS OCCURRED
COMMAREA = '1287656678'
EIBTIME = 135510
EIBDATE = 91163
EIBTRNID = 'AC20'
EIBTASKN = 76
EIBTRMID = 'S232'
EIBCPOSN = 4
EIBCALEN = 10
EIBAID = X'7D' AT X'032F059A'
EIBFN = X'1804' SEND AT X'032F059B'
EIBRCODE = X'000000000000' AT X'032F059D'
EIBDS = '.....'
+ EIBREQID = '.....'

ABEND : ABCD

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: UNDEFINED

그림 48. 이상 종료 발생 시 전형적인 EDF 표시장치


```

TRANSACTION: AC20          TASK: 00054 APPLID: 1234567 DISPLAY: 00
STATUS: ABNORMAL TASK TERMINATION
  COMMAREA      = '2934564671'
  EIBTIME       = 135510
  EIBDATE       = 91163
  EIBTRNID      = 'AC20'
  EIBTASKN      = 76
  EIBTRMID      = 'S232'
  EIBCPOSN      = 4
  EIBCALEN      = 10
  EIBAID        = 'X'7D'          AT X'032F059A'
  EIBFN         = 'X'1804' SEND    AT X'032F059B'
  EIBRCODE      = 'X'000000000000' AT X'032F059D'
  EIBDS         = '.....'
+ EIBREQID      = '.....'

```

```

ABEND : ABCD
CONTINUE EDF? (ENTER YES OR NO)          REPLY: YES
ENTER: CONTINUE
PF1 : UNDEFINED      PF2 : SWITCH HEX/CHAR  PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK    PF8 : SCROLL FORWARD  PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY    PF12: UNDEFINED

```

그림 49. 비정상 태스크 종료 시 전형적인 EDF 표시장치

본문은 COMMAREA 및 EIB에 있는 필드의 값뿐 아니라 다음 항목을 표시합니다.

- 이상 종료 코드
- 이상 종료 코드가 ASRA인 경우(즉, 프로그램 인터럽트가 발생했음), 인터럽트 시점에서 프로그램 상태 단어(PSW) 및 PSW에 의해 표시되는 인터럽트의 소스
- PSW가 인터럽트를 발생시키는 명령어가 애플리케이션 내에 있음을 표시하는 경우, 기본 시작점에 상대적인 해당 명령어의 오프셋

EDF를 사용하여 프로그램 테스트

CEDF 또는 CEDX 트랜잭션을 사용하여 EDF를 실행할 수 있습니다. 터미널 이외 트랜잭션을 테스트 중인 경우 트랜잭션의 이름을 지정할 수 있는 CEDX 트랜잭션을 사용하십시오. 터미널과 연관된 트랜잭션을 테스트 중인 경우에는 CEDF 트랜잭션을 사용하십시오.

일반적으로 EDF 실행을 위한 몇 가지 방법 중 하나를 선택할 수 있지만, 특정 방법이 요구되는 몇 가지 상황이 있습니다. 예를 들어, 원격 트랜잭션에 대해 단일 화면 모드를 사용해야 합니다. 사용자의 선택사항에 영향을 미치는 다른 조건에 대해 202 페이지 지의 『EDF 사용 시 제한사항』의 내용을 참조하십시오.

프로그램 실행 인터럽트

EDF를 사용할 때 디버깅을 돕기 위해 프로그램에서 방해 지점에서 다른 작업을 수행할 수 있습니다.

EDF의 장점은 각 방해 위치에서 할 수 있는 작업에 있습니다. 예를 들어, 다음을 수행할 수 있습니다.

- 명령이 실행되기 전에 인수값을 변경합니다. CICS 명령의 경우 명령 자체를 변경하거나 옵션을 추가하거나 삭제할 수 없지만 옵션과 연관된 값을 변경할 수 있습니다. NOOP를 사용하여 전적으로 명령의 실행을 억제할 수도 있습니다. 자세한 정보는 219 페이지의 『EDF를 사용하여 정보 변경』의 내용을 참조하십시오.
- 실행에 의해 리턴된 인수값을 변경하거나 응답 코드를 변경하여 명령의 결과를 변경하십시오. 이를 사용하면 일반 테스트 데이터를 사용해서는 도달하기 어려운 프로그램의 분기를 테스트할 수 있습니다(예를 들어, 입출력(I/O) 오류 시에 발생하는 일). 또한 이것이 문제점을 제거하는지 여부를 확인하기 위해 오류의 영향을 무시할 수도 있습니다.
- 프로그램의 작업 중인 저장영역, EXEC 인터페이스 블록(EIB) 및 DL/I 프로그램의 경우 DL/I 인터페이스 블록(DIB)을 표시하십시오.
- 명령 해석기(CECI)를 호출하십시오. CECI 아래에서 추가 정보를 얻거나 실행 환경을 변경하기 위해 프로그램에 없는 명령을 실행할 수 있습니다.
- CICS 리전에 다른 위치를 표시합니다.
- EIB 및 DIB에서 프로그램과 대부분의 필드의 작업 중인 저장영역을 변경합니다. EDF는 저장영역의 다른 영역을 변경하지 못하게 하여 태스크가 다른 태스크를 방해하는 것을 막습니다.
- 임시 저장영역과 트랜지언트 데이터 큐의 내용을 표시합니다.
- 하나 이상의 특정 조건 세트가 충족될 때까지 EDF 표시장치를 비활성화합니다. 이는 테스트 속도를 높입니다.
- 최대 10개의 이전 EDF 표시장치 또는 저장된 화면을 검색합니다.
- EDF 모드를 끄고 애플리케이션을 정상적으로 실행합니다.
- 이상 종료가 있는 태스크를 중지합니다.

변경의 처음 두 유형은 명령 표시장치의 본문에 값을 덮어써서 작성됩니다. 219 페이지의 『EDF를 사용하여 정보 변경』에서는 이를 수행하는 방법을 알려줍니다. 다른 메뉴에서는 기능 키를 사용합니다. others; 221 페이지의 『EDF 메뉴 기능 사용』에서는 할 수 있는 작업과 이를 수행하는 방법에 대해 정확히 알려줍니다.

```
TRANSACTION: DLID PROGRAM: DLID TASK: 00049 APPLID: IYAHZCIB DISPLAY:00
ADDRESS: 00000000
```

```
WORKING STORAGE IS NOT AVAILABLE
```

```
ENTER: CURRENT DISPLAY
```

```
PF1 : UNDEFINED      PF2 : BROWSE TEMP STORAGE  PF3 : UNDEFINED
PF4 : EIB DISPLAY    PF5 : INVOKE CECI          PF6 : USER DISPLAY
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF  PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: REMEMBER DISPLAY
```

그림 50. CECI를 호출할 수 있는 일반 EDF 표시장치

단일 화면 모드에서 EDF 사용

한 개의 터미널에 대해서만 EDF를 사용하면 EDF 입력 및 출력은 트랜잭션의 입력 및 출력으로 인터리브됩니다.

복잡하게 들릴 수도 있지만 실제로는 사용하기 매우 쉽습니다. SEND 명령 다음에 RECEIVE 명령이 오는 경우에만 조금 특별한데 SEND 명령으로 전송된 표시장치가 두 번 표시됩니다. SEND가 실행될 때 한 번 RECEIVE가 실행될 때 다시 한 번 표시되어 두 번 표시됩니다. 첫 번째 표시장치에는 응답할 필요가 없으며 응답하면 EDF가 첫 번째 표시장치에서 응답한 내용을 두 번째 표시장치까지 보존합니다.

다음과 같은 두 가지 방법으로 EDF를 시작할 수 있습니다.

- 깨끗한 화면에 입력 트랜잭션 CEDF를 입력
- 적절한 기능 키 누르기(EDF에 대한 기능 키가 정의된 경우)

그런 다음 다음 단계를 완료하여 트랜잭션이 테스트되도록 하십시오.

1. CLEAR 키를 눌러 화면을 지우십시오.
2. 테스트하려는 트랜잭션의 트랜잭션 코드를 입력하십시오.

EDF와 사용자 트랜잭션 모두 동일한 터미널을 공유하는 경우, EDF는 다음과 같은 상황에서 사용자 트랜잭션 표시장치를 복원합니다.

- 트랜잭션에서 운영자의 입력이 필요한 경우
- 트랜잭션 표시장치를 변경하는 경우

- 트랜잭션 종료 시
- EDF 표시장치를 비활성화하는 경우
- USER DISPLAY를 요청하는 경우

복원이 사용되도록 다음과 같은 상황에서 사용자 표시장치가 기억됩니다.

1. 태스크 시작 시에 태스크에 대한 첫 번째 EDF 화면이 표시되기 전
2. 사용자 표시장치가 변경된 경우, 다음 EDF 화면이 표시되기 전
3. SCREEN SUPPRESS 모드 종료 시

NOEDF 옵션으로 변환된 애플리케이션 또는 해당 자원 정의에 CEDF에 대해 아무것도 지정되지 않은 애플리케이션에 CEDF를 사용하는 경우, EDF는 해당 애플리케이션이 표시장치를 변경하는 시기를 확인할 수 없습니다. 따라서 EDF는 나중에 사용할 수 있도록 해당 표시장치의 사본을 저장할 수 없습니다. 다음 EDF 표시장치가 애플리케이션에서 전송된 모든 표시장치를 겹쳐쓰고 이를 복원할 수 없습니다.

유사하게 CEDF는 애플리케이션에서 변경되려는 시점 또는 트랜잭션에 운영자의 입력이 필요할 때 현재 표시장치를 복원할 수 없습니다. 따라서, 애플리케이션의 프린시플 기능에 대한 출력 명령으로 인해 이전 EDF 표시장치의 랜덤 백그라운드 정보가 화면에 표시될 수 있습니다.

입력 명령은 애플리케이션의 표시장치가 아니라 이전 EDF 표시장치에 대해 실행 가능하며 또는 트랜잭션에서 처음 수신되었다면 초기 TIOA의 내용만으로 충족되지 않고 CEDF 패널의 명시적 입력이 필요할 수도 있습니다.

이 고려사항은 애플리케이션에서 수행되는 모든 화면 I/O 조작에 적용됩니다.

EDF가 트랜잭션 표시장치를 복원하면 사용자 트랜잭션과 동일한 방식으로 알람이 울리지 않거나 키보드에 영향을 주지 않습니다. 사용자 트랜잭션 옵션의 영향은 SEND 명령이 처리될 때 표시되며 화면이 복원될 때는 표시되지 않습니다. 단일 화면 모드 NOEDF를 지정하면 사용자 프로그램은 데이터를 볼 수 없기 때문에 이를 전송하거나 수신하지 않습니다.

EDF가 색상, 프로그램화된 기호 또는 확장 강조표시를 사용하는 디바이스에서 트랜잭션 표시장치를 복원하면 이 속성은 더 이상 표시되지 않으며 표시장치는 프로그램화된 기호 또는 확장 강조표시 없이 흑백으로 표시됩니다. 또한, 애플리케이션의 인바운드 응답 모드가 속성 설정 키를 사용하기 위해 문자로 설정되면 EDF는 이 모드를 다시 설정하여 해당 키를 사용할 수 없게 됩니다. 이 키 변경으로 인해 사용자의 트랜잭션은 제대로 실행되지 않는 경우 이중 화면 모드에서 테스트하십시오.

트랜잭션을 통해 EDF 세션 파트를 종료하는 경우, 가장 최근의 RECEIVE 명령 뒤에 SEND 명령이 오지 않으면 EDF는 키보드가 잠금된 상태로 화면을 복원합니다. 그렇지 않으면 키보드가 잠금 해제됩니다.

의사 대화식 프로그램

EDF는 단일 터미널에서 의사 대화식 트랜잭션을 테스트하기 위해 특수 조항을 작성합니다. 터미널이 의사 대화식 트랜잭션을 구성하는 여러 태스크 사이의 EDF 모드에서 사용 중인 경우, 첫 번째 태스크 후에 디버깅을 수행하기는 매우 어렵습니다. 따라서 태스크가 종료되면 EDF는 운영자에게 EDF 모드를 다음 태스크로 계속할지 여부를 확인합니다. 의사 대화식 태스크를 디버깅하는 경우, Enter를 눌러 기본값 예를 승인하십시오. 완료했으면 아니오로 응답하십시오.

이중 화면 모드에서 EDF 사용

이중 화면 모드에서는 EDF 상호작용을 위해 한 터미널을 사용하고 테스트되고 있는 트랜잭션으로 입력을 보내고 출력을 받기 위해 한 터미널을 사용합니다.

EDF 터미널에서 CEDF *tttt* 트랜잭션을 입력하여 시작하는데, *tttt*는 트랜잭션이 테스트될 터미널의 이름입니다.

CEDF가 이 트랜잭션에 대한 응답으로 제공하는 메시지는 두 번째 터미널에서 실행 중인 트랜잭션이 이미 있는지 여부에 따라 다릅니다. 두 번째 터미널이 사용 중이 아닌 경우, 첫 번째 터미널에 표시되는 메시지는 다음과 같습니다.

```
TERMINAL tttt: EDF MODE ON
```

PROGRAM INITIATION 표시장치가 나타날 때 두 번째 터미널에서 트랜잭션이 시작될 때까지 더 이상 아무 일도 발생하지 않습니다.

또한 이중 화면 모드에서 EDF를 사용하여 이미 두 번째 터미널에서 실행 중인 트랜잭션을 모니터링할 수 있습니다. 예를 들어 특정 터미널에 있는 트랜잭션이 루프 상태에 있을 수 있다고 믿는 경우, 다른 터미널로 이동하여 트랜잭션이 실행 중인 터미널을 지명하는 CEDF 트랜잭션에 들어갈 수 있습니다. 첫 번째 터미널에 표시되는 메시지는 다음과 같습니다.

```
TERMINAL tttt: TRANSACTION RUNNING: EDF MODE ON
```

EDF는 실행되는 다음 **EXEC CICS** 명령에서 제어를 얻고, 그러면 사용자는 하나 이상의 **EXEC CICS** 명령이 실행 중이라고 가정할 때 루프를 유발하고 있는 명령 순서를 관찰할 수 있습니다.

EDF 및 원격 트랜잭션

테스트되는 트랜잭션이나 이 트랜잭션을 호출하는 터미널이 다른 CICS 리전에서 소유하고 있는 경우 이중 화면 모드에서 EDF를 사용할 수 없습니다.

CRTE 라우팅 세션을 사용하는 EDF 아래에서 원격 트랜잭션이 비정상적으로 종료되는 경우, EDF는 비정상 태스크 종료 화면을 표시하고, 사용자 트랜잭션에 대한 메시지 DFHAC2206을 표시합니다. CRTE 세션은 사용자 태스크 이상 종료의 영향을 받지 않

습니다. 또한 이상 종료 후에 EDF에 대해 계속할 것을 선택한 경우, 터미널은 CRTE 라우팅 세션 내에서 EDF 모드로 유지됩니다.

실행에도 차이가 있습니다. 원격 트랜잭션의 경우, EDF는 EDF가 계속되는지 여부에 관계없이 각 트랜잭션의 종료 시 세션의 해당 메모리를 제거합니다. 이는 설정한 옵션과 저장된 화면이 의사 대화식 순서에서 개별 태스크 사이에 손실됨을 의미합니다.

EDF 및 비터미널 트랜잭션

터미널 없이 실행하는 트랜잭션, 예를 들어 **EXEC CICS START** 명령에 의해 시작되는 트랜잭션이나 트랜지언트 데이터 트리거 레벨에 의해 시작되는 트랜잭션을 테스트하려면 EDF를 사용하십시오. 비터미널 트랜잭션을 테스트하려면 **CEDX trnx** 명령을 사용하십시오. 여기서 *trnx*는 트랜잭션 ID입니다.

CEDX를 사용하여 트랜잭션을 테스트하려면 다음 조건이 만족되어야 합니다.

- CEDX 명령을 입력하고 EDF 표시장치를 위해 사용하는 터미널은 지정된 트랜잭션이 실행할 CICS 리전에 로그인되어야 합니다.
- CEDX 명령이 지정된 트랜잭션이 CICS에 의해 시작되기 전에 실행되어야 합니다. CEDX 명령을 실행할 때 이미 실행 중인 동일한 트랜잭션의 다른 인스턴스는 무시됩니다.
- CEDX 명령에서 지정하는 트랜잭션은 로컬 CICS 리전에서 실행해야 합니다. CEDX 트랜잭션 뒤에 CRTE 라우팅 트랜잭션을 사용할 수 없습니다.

CEDX를 사용하여 트랜잭션을 디버그할 때, CICS는 CEDX 명령에서 지정되는 트랜잭션의 정의를 수정하여 특수 트랜잭션 클래스인 DFHEDFTC를 참조하여 EDF 조작을 제어합니다. EDF를 끝 때(CEDX *tranid*,OFF를 사용하여) CICS는 트랜잭션 정의를 다시 정상 트랜잭션 클래스로 수정합니다.

EDF 및 DTP 프로그램

실행 진단 기능(EDF)에 링크의 세션을 모니터하도록 지시하여 원격 링크 사이에서 DTP(Distributed Transaction Processing)를 사용 중인 트랜잭션을 테스트할 수 있습니다.

CICS에서 실행 중이고 EDF가 설치된 참여 시스템 중 하나(또는 둘 다)에서 이를 수행할 수 있습니다. 트랜잭션이 또 다른 CICS 리전으로부터 라우팅된 경우 원격 트랜잭션에 대해 단일 화면 모드를 사용해야 하기 때문에 이를 수행할 수 없습니다.

APPC 및 MRO 링크의 경우, 원격 시스템의 시스템 ID를 지명할 수 있습니다.

CEDF sysid

그러면 EDF가 자신을 지정된 시스템에 속하는 모든 세션 사이에서 접속된 임의의 트랜잭션과 연관시킵니다.

APPC, MRO, LU6.1 링크의 경우, 트랜잭션이 사용 중인 세션 ID를 사용할 수 있습니다.

CEDF sessionid

INQUIRE TERMINAL 명령으로 세션 ID를 판별할 수 있지만, 이것은 트랜잭션이 실행 중이고 EDF를 시작하기 전에 세션 설정 지점에 도달했어야 함을 의미합니다.

분산 트랜잭션 처리를 사용하는 트랜잭션이 그와 연관된 터미널도 갖는 경우 또는 터미널(하나를 사용하지 않는 경우에도)에서 트랜잭션을 호출하는 경우, EDF를 사용하여 해당 터미널에서 일반적인 방법으로 테스트할 수 있습니다.

원격 시스템에서 트랜잭션 테스트를 완료했을 때, CESF를 사용하여 CICS에서 로그오프하기 전에 해당 시스템 ID 또는 세션 ID에서 EDF를 끄십시오. 예를 들어, 다음과 같습니다.

CEDF sysid,OFF

EDF를 끄지 못하면 해당 시스템에 대한 링크를 사용 중인 또 다른 트랜잭션이 일시중단될 수 있습니다.

EDF 및 DPL(Distributed Program Link) 명령

단일 또는 이중 터미널 모드에서 EDF를 사용하여 DPL(Distributed Program Link) 명령을 포함하는 트랜잭션을 테스트할 수 있습니다. 그러나, EDF는 DPL 명령 호출 및 응답 화면만 표시합니다. 원격 프로그램에 의해 실행되는 CICS 명령은 표시되지 않지만, 원격 이상 종료 및 원격 이상 종료 발생했음 메시지가 이상 종료가 수신된 시스템의 시스템 ID와 함께 EDF 터미널로 리턴됩니다. 제어가 로컬 프로그램으로 돌아간 후, EDF는 정상적으로 테스트를 계속하지만 이상 종료가 원격 프로그램에 있는 경우 프로그램 상태 단어(PSW)가 표시되지 않습니다.

EDF 중지

터미널의 EDF 제어를 종료하려는 경우 사용하는 방법은 테스트에서 사용자 위치에 따라 다릅니다.

테스트 중인 트랜잭션이 계속 실행 중이고 EDF 없이 계속하기를 원하면, END EDF SESSION 기능 키를 누르십시오. 테스트 종료 인터셉트에 도달한 경우, EDF는 계속할 것인지 여부를 묻습니다. 계속하지 않을 경우, 응답을 NO로 겹쳐지게 하십시오. (YES가 기본값입니다.) 터미널에서 어떤 트랜잭션도 실행되고 있지 않으면, 화면을 지우고 다음을 입력하십시오.

CEDF ,OFF

(공백 및 쉼표가 필수입니다.)

이중 화면 모드에서 로그 오프 중이면, 화면을 지우고 CEDF *tttt*,OFF를 입력하십시오.

모든 경우에서, 메시지 THIS TERMINAL: EDF MODE OFF가 비어 있는 화면의 맨 위에 표시됩니다.

EDF를 사용하여 정보 변경

EDF를 사용하는 대부분의 변경에서는 메모리의 정보가 변경됩니다. 변경하려면 화면에 표시된 정보에 필요한 정보를 입력하십시오.

화면 아래 부분에 있는 메뉴를 제외하고는 Tab 키를 사용하여 커서를 이동해서 영역을 변경할 수 있습니다.

화면을 변경할 때 다음과 같은 규칙을 준수해야 합니다.

- CICS 명령 화면에서 모든 인수 값을 다시 입력할 수 있지만 인수의 키워드는 안됩니다. 선택적 인수는 제거할 수 없으며 옵션은 추가하거나 삭제할 수 없습니다.
- 작업 중인 저장영역 화면이 아니라 명령 표시장치에서 인수를 변경하면 표시장치에 표시된 파트만 변경할 수 있습니다. 표시된 값 이외의 값을 입력하면 변경되지 않으며 진단 메시지도 생성되지 않습니다. 인수가 너무 길어서 일부만 화면에 표시되는 경우, 작업 중인 저장영역에서 인수가 지시하는 부분으로 영역을 변경하십시오. 주소를 판별하려면 인수 위치 주소도 표시되도록 인수를 16진 형식으로 표시하십시오.
- 풀워드인 인수 값에 입력하는 경우 입력 가능한 최대 값은 2147483639입니다.
- 문자 형식에서 숫자 값에는 항상 부호 필드가 있습니다. 마이너스 문자(-) 또는 공백으로만 부호 필드에 입력할 수 있습니다.
- 인수가 문자 형식으로 표시되면 일부 문자는 표시되지 않을 수도 있습니다(소문자 포함). EDF는 표시되지 않는 각 문자를 마침표로 대체합니다. 마침표 위에 입력하려는 경우, 저장영역에 표시되지 않는 문자가 포함되어 있을 수도 있음에 주의하십시오.

마침표가 포함된 문자에 입력하여 이를 변경할 수는 없습니다. 변경은 무시되고 진단 메시지가 표시되지 않습니다. 문자를 마침표로 변경하려면 표시장치를 16진 형식으로 전환하고 F2 키를 사용하며 X'4B' 값으로 입력하십시오.

- 저장영역이 문자 및 16진 형식 모두로 표시될 때 두 형식을 변경하고 변경사항이 서로 충돌하는 경우 16진 필드 값이 우선적으로 사용됩니다. 진단 메시지는 표시되지 않습니다.
- 일부 명령 인수(예: HANDLE CONDITION)는 숫자나 문자 데이터가 아니라 프로그램 레이블입니다. EDF의 양식이 이 인수를 표시(및 해당 수정사항 승인)하는 것은 사용 중인 프로그래밍 언어에 따라 달라집니다.

- COBOL의 경우 널(null) 인수가 표시되며 이를 수정할 수 없습니다. 예: ERROR ().
- C 및 C++의 경우 레이블은 유효하지 않습니다.
- PL/I의 경우, 레이블 상수 주소가 사용됩니다. 예: ERROR (X'001D0016').
- 어셈블러 언어의 경우, 프로그램 레이블 주소가 사용됩니다. 예: ERROR (X'00030C').
- AMODE(64) 어셈블러 언어의 경우, 레이블이 지원되지 않습니다.

HANDLE CONDITION 명령에 레이블 값이 지정되지 않으면 EDF는 소괄호 없이 조건 이름만 표시합니다.

- 현재 함수에 대해 발생 가능한 모든 예외 조건 이름으로 응답 필드에 입력할 수 있습니다(ERROR 또는 NORMAL 단어 포함). EDF가 계속되면 지정된 응답에 대해 지정된 조치를 프로그램이 수행합니다. EIB 표시장치의 EIBRESP 필드를 해당 값을 변경해도 동일한 결과를 얻을 수 있습니다. EIBRESP 값 또는 『명령 실행 완료』 화면의 응답 필드를 변경하면 EIBRCODE가 갱신됩니다. EIBRESP는 두 번째 EIB 화면에 표시되며 유일하게 변경할 수 있습니다(EIBRCODE 보호). EIB 표시장치의 EIBRESP 값을 변경해도 동일한 결과를 얻을 수 있습니다. EDF는 EIB 및 명령 화면에서 관련된 값을 적절하게 변경합니다.
- 대문자 변환이 사용 중인 터미널에 대해 지정되지 않은 경우, 항상 대문자로 입력하십시오.
- 명령 처리 억제를 처리하기 전에 NOOP 또는 NOP를 사용하여 모든 명령에 입력할 수 있습니다. 공백을 입력하거나 ERASE EOF 키를 사용해서도 동일한 결과를 얻을 수 있습니다. 화면이 NOOP를 사용하여 다시 표시되면 ERASE EOF를 사용하여 전체 verb 행을 지우고 Enter 키를 눌러 원래 verb 행을 복원할 수 있습니다.
- 인수가 이미 64비트 주소인 경우 64비트 주소를 입력할 수 있습니다(중간에 밑줄이 포함된 주소 포함) 예: AAAAAAAAABBBBBBB 또는 AAAAAA_BBBBBBB.

프로그램의 데이터 영역을 나타내는 필드에 입력하는 경우, 애플리케이션 저장영역에서 직접 변경이 수행되고 이는 영구적으로 유지됩니다. 그렇지만 상수(프로그램 리터럴)를 나타내는 필드를 변경하는 경우, 프로그램 저장영역은 변경되지 않습니다. 그 이유는 이 변경이 동일한 상수 또는 프로그램을 사용하는 다른 태스크를 사용하는 프로그램의 다른 파트에도 영향을 줄 수 있기 때문입니다. 명령은 변경된 데이터로 실행되지만 처리 후에 명령이 표시되면 원래 인수 값이 다시 표시됩니다. 예를 들어, 다음 코드가 포함된 프로그램을 테스트할 수 있습니다.

```
EXEC CICS SEND MAP('MENU') END-EXEC.
```

명령을 실행하기 전에 MENU에서 MENU2로 이름을 변경하기 위해 EDF를 사용하는 경우 사용되는 맵은 MENU2지만 응답에 표시되는 맵은 MENU입니다. 『이전 표시장

치』 키를 사용하여 사용한 맵 이름을 확인할 수 있습니다. 두 번 이상 동일한 명령을 처리하는 경우에는 매번 변경에 대해 이 유형을 입력해야 합니다.

EDF 응답

이 규칙 목록을 사용하여 키보드 입력에 대한 EDF 응답을 배울 수 있습니다.

규칙은 표시된 순서대로 다음과 같습니다.

1. CLEAR 키가 사용되면 EDF는 변경사항을 무시한 채 화면을 재표시합니다.
2. 올바르지 않은 변경이 만들어지면, EDF는 유효한 변경을 승인하고 진단 메시지로 화면을 재표시합니다.
3. 표시 번호가 변경되면 EDF는 다른 변경을 승인하고 요청된 표시를 표시합니다.
4. 기능 키가 사용되면 EDF는 변경을 승인하고 기능 키가 요청하는 조치를 수행합니다. 화면 맨 아래에 있는 메뉴에서 기능 키 정의 아래에서 커서로 ENTER를 누르는 것은 기능 키를 누르는 것과 동일합니다.
5. ENTER 키를 누르고 화면이 수정되면(REPLY 필드 이외) EDF는 변경이 포함된 상태로 화면을 재표시합니다.
6. ENTER 키를 누르고 화면이 수정되지 않은 경우에는(REPLY 필드 이외) 효과는 ENTER 키의 의미에 따라 다릅니다. ENTER 키가 CONTINUE를 의미하면 사용자 트랜잭션은 계속 실행됩니다. CURRENT DISPLAY를 의미하는 경우에는, EDF는 상태 표시를 재표시합니다.

EDF 메뉴 기능 사용

각 위치에서 사용할 수 있는 기능 키는 모든 EDF 표시장치의 맨 아래에 있는 메뉴에 표시됩니다.

모든 표시장치에 적용되는 기능은 항상 동일한 키에 지정되지만, 일부 키의 정의는 표시장치 및 인터셉트 위치에 따라 다릅니다. 옵션을 선택하려면, 표시된 기능 키를 누르십시오. 터미널에 24개의 기능 키가 있는 경우, EDF는 PF13 - PF24를 PF1 - PF12의 중복으로 처리합니다. 터미널에 PF 키가 없는 경우, 원하는 옵션 아래에 커서를 놓고 ENTER 키를 누르십시오.

ABEND USER TASK

모니터 중인 태스크를 종료합니다. 이 조치를 준수하기 위해 ENTER ABEND CODE AND REQUEST ABEND AGAIN 메시지가 표시됩니다. 커서 위치에서 이상 종료 코드를 입력한 후, 지정된 코드로 식별되는 트랜잭션 덤프로 태스크를 이상 종료하도록 이 기능을 다시 요청하십시오. NO를 입력하는 경우, 태스크는 덤프 없이, 그리고 네 개의 물음표 (????)의 4자 기본 이상 종료 코드로 이상 종료됩니다.

문자 A로 시작하는 이상 종료 코드는 CICS에서 사용하도록 예약되어 있습니다. CICS 이상 종료 코드를 사용하면 예기치 않은 결과가 발생할 수 있습니다.

이상 종료가 이미 진행 중이거나 태스크가 종료되는 경우 이 기능을 사용할 수 없습니다.

BROWSE TEMP STORAGE

임시 저장영역 큐 CEBRxxxx의 표시장치를 생성합니다. 여기서 xxxx는 EDF를 실행 중인 터미널의 터미널 ID입니다. 이 기능은 작업 중인 저장영역(PF5) 화면에서만 사용할 수 있습니다. 임시 저장영역 큐를 표시 또는 수정하고 트랜지언트 데이터 큐를 읽거나 쓰기 위해 CEBR 명령을 사용할 수 있습니다.

CONTINUE

변경사항을 통합하기 위해 현재 화면을 다시 표시합니다. 어떤 변경사항도 작성하지 않은 경우, CONTINUE는 테스트할 트랜잭션이 다음 인터셉트 위치까지 실행을 재개하도록 합니다. 계속하려면 ENTER를 누르십시오.

CURRENT DISPLAY

변경사항을 통합하기 위해 현재 화면을 다시 표시합니다. 어떤 변경사항도 작성하지 않은 경우, EDF는 마지막 인터셉트 위치에 대해 명령 화면을 표시합니다. 이 기능을 실행하려면, 적절한 화면에서 ENTER를 누르십시오.

DIB DISPLAY

DL/I 인터페이스 블록(DIB)의 내용을 표시합니다. 이 기능은 작업 중인 저장영역 화면(PF5)에서만 사용할 수 있습니다. DIB 필드에 대한 정보는 *IMS: Application programming for EXEC DLI*의 내용을 참조하십시오.

EIB DISPLAY

EXEC 인터페이스 블록(EIB)의 내용을 표시합니다. EIB 표시장치의 예제에 대해서 206 페이지의 그림 39의 내용을 참조하십시오. EIB에 대한 프로그래밍 정보에 대해서 참조 -> 애플리케이션 개발에서 EIB 필드의 내용을 참조하십시오. COMMAREA가 존재하면, EDF는 해당되는 주소와 한 행의 데이터를 덤프 형식으로 표시합니다.

INVOKE CECI

명령 레벨 해독기(CECI)에 액세스합니다. 이 기능은 작업 중인 저장영역(PF5) 화면에서만 사용할 수 있습니다. CECI가 호출되는 화면의 예제는 214 페이지의 그림 50의 내용을 참조하십시오. 그런 다음 241 페이지의 제 14 장 『명령 레벨 해독기(CECI)』에서 논의되는 CECI 명령을 사용할 수 있습니다. 이 CECI 명령에는 명령 실행 전후에 원래 명령에 의해 참조된 자원에 대해 **INQUIRE** 및 **SET** 명령이 포함됩니다. 이중 화면 모드에서 CECI를 실행할 때 제한사항에 대해서는 인바운드 응답 모드를 참조하십시오. 이 패널에서 CECI를 사용하는 것은 CEDF 내에서 CEBR을 사용하는 것과 유사합니다.

END EDF SESSION

트랜잭션의 EDF 제어를 종료합니다. 트랜잭션은 해당 위치에서 실행을 계속하지만, 더 이상 EDF 모드에서 실행되지 않습니다.

NEXT DISPLAY

이전 표시장치로 리턴한 경우, 앞으로 다음 표시장치를 표시하고 표시장치 번호를 1씩 증가시킵니다. 이 옵션은 PREVIOUS DISPLAY의 반대입니다.

PREVIOUS DISPLAY

다른 표시장치를 저장하지 않으면, 이전 표시장치를 화면으로 전송합니다. 현재 인터셉트 위치에서 표시장치 번호는 항상 00입니다. 이전 표시장치를 요청하는 대로, 표시장치 번호는 첫 번째 이전 표시장치에 대해 -01로, 그 이전의 표시장치에 대해 -02로 1씩 줄어들고 가장 오래된 표시장치는 -10으로 떨어집니다. 더 이상 사용 가능한 이전 화면이 없으면, PREVIOUS 옵션은 메뉴에서 더 이상 사용 가능하지 않고 해당하는 기능 키가 작동하지 않습니다.

REGISTERS AT ABEND

로컬 ASRA 이상 종료가 발생하는 경우 레지스터의 값을 포함하는 저장영역을 표시합니다. 저장영역의 레이아웃은 이상 종료 시 16바이트 프로그램 상태 단어(PSW)가 뒤에 오는 64비트 레지스터입니다. 그림 51에는 일반적이 화면이 나와 있습니다. 일부 경우에는, EDF가 레지스터의 값을 캡처하기 전에 두 번째 프로그램 확인이 리전에서 발생할 때, 이 기능이 이상 종료 표시장치의 메뉴에 나타나지 않습니다. 이러한 상황이 발생하면, 실행되는 두 번째 테스트에서 자세한 정보가 제공됩니다.

```
TRANSACTION: UT  PROGRAM: ASRA31  TASK: 0000487  APPLID: IYK2ZKE1  DISPLAY: 00
ADDRESS: 0006F010
0006F010  000000  00000000 00000090 00000000 0010003C  ..... 1
0006F020  000010  00000000 00041800 00000000 AE410028  ..... 1
0006F030  000020  00000000 2C2E35A8 00000000 00000000  .....y..... 1
0006F040  000030  00000000 2C2D7800 00000000 7F2C9918  .....".r. 1
0006F050  000040  00000000 2E410000 00000000 2C2D7818  ..... 1
0006F060  000050  00000000 00100008 00000000 00100100  ..... 1
0006F070  000060  00000000 009AF000 00000000 00100690  .....0..... 1
0006F080  000070  00000000 AE410158 00000000 00100690  ..... 1
0006F090  000080  079D0000 80000000 00000000 2E410178  ..... 2
0006F0A0  000090  00040004 00000000 00000000 00000000  .....
0006F0B0  0000A0  00000000 00000000 00000000 00000000  .....
0006F0C0  0000B0  00000000 00000000 00000000 00000000  .....
0006F0D0  0000C0  00000000 00000000 00000000 00000000  .....
0006F0E0  0000D0  00000000 00000000 D4F0F0F0 F0F4F8F9  .....M0000489
0006F0F0  0000E0  D4F0F0F0 F0F4F8F9 00000000 00000000  M0000489.....
0006F100  0000F0  00000000 000000E4 E3404000 00F00000  .....UT ..0..
```

ENTER: CURRENT DISPLAY

```
PF1 : UNDEFINED          PF2 : BROWSE TEMP STORAGE PF3 : UNDEFINED
PF4 : EIB DISPLAY        PF5 : INVOKE CECI          PF6 : USER DISPLAY
PF7 : SCROLL BACK HALF  PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL  PF11: SCROLL FORWARD FULL PF12: REMEMBER DISPLAY
```

그림 51. REGISTERS AT ABEND에 대한 일반적인 EDF 표시장치

참고:

1. 레지스터 값
2. PSW

REMEMBER DISPLAY

일반적으로 메모리에 보존되지 않는 표시장치(예: EIB 표시장치)를 EDF 메모리에 위치시킵니다. EDF는 각 명령의 시작 및 완료 시 자동으로 표시장치를 저장합니다. 메모리는 10개까지 표시장치를 보유할 수 있습니다. 표시장치에는 역시간 순서로 번호가 매겨집니다. (즉, -10은 가장 오래된 표시장치이고 -01은 최근 표시장치입니다.) 표시장치와 연관되는 모든 페이지는 메모리에서 보존되며 다시 호출 시 화면 이동될 수 있습니다. 그러나 작업 중인 저장영역 표시장치를 저장할 경우 보기의 화면만 저장됩니다.

SCROLL BACK

표시장치에서 이전 화면을 표시합니다. 이 기능은 하나의 화면에 모두 맞춰지지 않는 EIB, DIB 또는 명령 표시장치에 적용됩니다. 보기의 화면이 표시장치의 첫 번째 화면이 아니고, 첫 번째 옵션 또는 필드 이전에 더하기 부호(+)가 있는 경우, 표시장치에서 이전 화면을 보려면 이 기능을 선택하십시오. 예제는 206 페이지의 그림 39의 내용을 참조하십시오.

SCROLL FORWARD

표시장치에서 다음 화면을 표시합니다. 이 기능은 하나의 화면에 모두 맞춰지지 않는 EIB, DIB 또는 명령 표시장치에 적용됩니다. 표시장치가 맞지 않는 경우, 추가 화면이 있음을 표시하기 위해 표시장치의 마지막 옵션 또는 필드에 더하기 부호(+)가 나타납니다. 다음 화면을 표시하려면 이 기능을 선택하십시오.

SCROLL BACK FULL

작업 중인 저장영역 표시장치에서 이전 화면을 표시합니다. 이 기능은 작업 중인 저장영역의 표시장치에 적용되며 EIB 및 DIB 표시장치에 대한 SCROLL BACK 옵션과 유사합니다. SCROLL BACK FULL은 작업 중인 저장영역 표시장치에 하나의 전체 화면을 뒤로 제공하여, 현재 화면의 주소보다 저장영역에서 더 낮은 주소를 보여줍니다.

SCROLL FORWARD FULL

작업 중인 저장영역 표시장치에서 다음 화면을 표시합니다. 이 기능은 작업 중인 저장영역의 표시장치에 적용되며 EIB 및 DIB 표시장치에 대한 SCROLL FORWARD 옵션과 유사합니다. SCROLL FORWARD FULL은 작업 중인 저장영역 표시장치에 하나의 전체 화면을 앞으로 제공하여, 현재 화면의 주소보다 저장영역에서 더 높은 주소를 보여줍니다.

SCROLL BACK HALF

화면의 반만큼 작업 중인 저장영역의 표시를 되돌립니다. 이 기능은 작업 중인 저장영역의 표시장치가 화면의 반만큼 되돌려지는 것을 제외하고 SCROLL BACK FULL과 유사합니다.

SCROLL FORWARD HALF

화면의 반만큼 작업 중인 저장영역의 표시장치를 진행합니다. 이 기능은 작업 중인 저장영역의 표시장치가 화면의 반만큼 진행되는 것을 제외하고 SCROLL FORWARD FULL과 유사합니다.

STOP CONDITIONS

EDF가 SUPPRESS DISPLAYS 함수를 사용한 후 표시장치를 재개하도록 하는 상태를 지정합니다. 그림 52에 있는 메뉴 화면이 표시됩니다. STOP CONDITIONS 및 SUPPRESS DISPLAYS 함수를 함께 사용하여 알고 있는 프로그램이 부분적으로 작동 중인지 확인할 때 상호작용을 감소할 수 있습니다.

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 0086 APPLID: 1234567 DISPLAY: 00
DISPLAY ON CONDITION:-
```

```
COMMAND:                EXEC CICS
OFFSET:                  X'.....'
LINE NUMBER:
CICS EXCEPTION CONDITION:  ERROR
ANY CICS CONDITION        NO
TRANSACTION ABEND         YES
NORMAL TASK TERMINATION   YES
ABNORMAL TASK TERMINATION YES
```

```
DLI ERROR STATUS:
ANY DLI ERROR STATUS
```

```
ENTER: CURRENT DISPLAY
```

```
PF1 : UNDEFINED      PF2 : UNDEFINED      PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : UNDEFINED      PF8 : UNDEFINED      PF9 : UNDEFINED
PF10: UNDEFINED      PF11: UNDEFINED     PF12: REMEMBER DISPLAY
```

그림 52. STOP CONDITIONS의 일반적인 EDF 표시장치

기본적으로, 그림 52에 표시된 것처럼, 다음 상태가 발생하는 경우 EDF가 표시장치를 재개합니다.

- CICS 예외 상태
- 트랜잭션 이상 종료
- 보통의 태스크 종료
- 비정상 태스크 종료

STOP CONDITIONS 메뉴를 사용하여 적용되지 않는 기본값을 끄고, 프로그램에 특정한 상태를 추가할 수 있습니다.

다음 이벤트 중 임의의 이벤트나 모두를 STOP CONDITIONS로 지정할 수 있습니다.

- 특정 유형의 함수 및 옵션(예: READNEXT 파일 또는 ENQ 자원)이 발견됩니다(예: FEPI ADD 또는 GDS ASSIGN).

- 특정 오프셋에 있거나 특정 행 번호에 있는 명령(프로그램이 DEBUG 옵션으로 변환된 것으로 가정함)이 발견됩니다.
- DL/I 오류 상태가 발생하거나, 특정 DL/I 오류 상태가 발생합니다.
- 특정 예외 상태가 발생합니다. ERROR(기본값)가 CICS EXCEPTION CONDITION에 대해 지정된 경우, EDF는 오류 상태(예: NOTOPEN, EOF 또는 INVREQ)에 대한 응답으로 화면을 다시 표시합니다. CICS EXCEPTION CONDITION에 대해 EOF과 같은, 특정 상태를 지정하고, NO(기본값)가 ANY CICS CONDITION에 대해 지정되는 경우, EDF는 해당 상태(EOF)가 발생할 때만 화면을 다시 표시합니다.

ANY CICS CONDITION에 대해 YES를 지정하면, EDF는 CICS 예외 상태를 대체하고 명령이 NOTOPEN, EOF 또는 QBUSY와 같은 0이 아닌 EIBRESP 값을 생성할 때마다 화면을 다시 표시합니다.

- CICS 조치에서 ERROR가 발생할 예외 조건이 발생합니다(예: INVREQ 또는 NOTFND).
- 이상 종료가 발생합니다.
- 태스크가 정상적으로 종료합니다.
- 태스크가 비정상적으로 종료합니다.

STOP CONDITIONS에 대한 오프셋을 사용할 때, 명령에 해당하는 BALR 지시사항의 오프셋을 지정해야 합니다. 오프셋은 컴파일러 또는 어셈블러에 의해 생성된 코드 목록에서 판별될 수 있습니다. COBOL, C, C++ 또는 PL/I에서, 관련 BALR 지시사항을 판별하기 위해 어셈블러 목록을 생성하는 컴파일러 옵션을 사용해야 합니다.

행 번호를 사용할 때, 선행 0을 포함하여 목록에 나타나는 그대로 정확하게 번호를 지정하고 명령이 시작되는 행이어야 합니다. NUM 또는 SEQUENCE 변환기 옵션을 사용한 경우, 변환기는 소스에 나타나는 대로 행 번호를 사용합니다. 그렇지 않으면, 변환기가 행 번호를 지정합니다.

SOURCE 또는 VBREF 변환기 옵션을 사용한 경우 변환기 목록에서 행 번호를 찾을 수 있습니다(변환기 단계에서 SYSPRINT). STOP CONDITIONS에 대해 행 번호를 사용해야 하는 것처럼 DEBUG 변환기 옵션을 사용한 경우, 행 번호는 CALL 문에서 매개변수로 명령의 변환된 양식에 임베드된 컴파일(어셈블리) 목록에도 나타납니다.

EDF가 CICS 명령과 DL/I 명령에서 표시장치를 재개하도록 지정할 수 있습니다. DLI가 있는 명령행에 CICS 규정자를 입력하고 표시장치 억제를 중지할 DL/I 명령의 유형을 입력하십시오. DL/I 프로그램을 실행하고 있거나 동일한 태스크에서 이전에 이 프로그램을 실행해 있어야 합니다. 일찍 프로그램 초기화 패널에서 DL/I 명령을 억제할 수 있습니다.

특정 DL/I 상태 코드가 발생할 때 표시장치 억제를 중지할 수 있습니다. 사용할 수 있는 상태 코드에 대한 정보는 *IMS: Application programming for EXEC DLI* 에서 DL/I 인터페이스 블록(DIB)에서 코드 목록을 참조하십시오.

SUPPRESS DISPLAYS

지정된 STOP CONDITIONS 중 하나가 발생할 때까지 모든 EDF 표시장치를 억제합니다. 그러나 상태가 발생할 때, 원래 작성될 때 화면에 전송되지 않은 경우에도 10개의 이전 명령 표시장치에 대한 액세스를 여전히 가지고 있습니다.

SWITCH HEX/CHAR

문자 및 16진 양식 사이에 표시장치를 전환합니다. 전환은 명령 표시장치에만 적용되고, 이전에 기억하고 있는 표시장치, STOP CONDITIONS 표시장치 또는 작업 중인 저장영역 표시장치에 영향을 주지 않습니다.

WHERE 옵션을 포함하는 DL/I 명령 화면 표시에서, 키 값(각 비교 연산자 뒤의 표현식)만 16진수로 변환될 수 있습니다.

UNDEFINED

표시된 기능 키는 현재 인터셉트 위치에서 현재 표시장치에 대해 정의되지 않습니다.

USER DISPLAY

트랜잭션이 EDF 모드에서 실행되고 있지 않은 경우 화면에 있을 것을 표시합니다. (단일 터미널 체크아웃에 대해서만 이를 사용할 수 있습니다.) 이 키를 사용한 후 EDF로 리턴하려면 ENTER 키를 누르십시오.

WORKING STORAGE

사용자 프로그램의 24비트 또는 31비트 작업 중인 저장영역 내용이나 CICS 리전에서 다른 주소의 내용을 보여줍니다. 228 페이지의 그림 53에는 일반 작업 중인 저장영역 화면이 나와 있습니다.

이 기능은 64비트 저장영역을 지원하지 않습니다.


```

TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00030 APPLID: 1234567 DISPLAY:00
ADDRESS: 035493F0 WORKING STORAGE
035493F0 000000 E3F14040 00000000 00010000 00000000 T1 .....
03549400 000010 00000000 00000000 F1000000 00000000 .....1.....
03549410 000020 F0000000 00000000 F0000000 00000000 0.....0.....
03549420 000030 F0000000 00000000 F0000000 00000000 0.....0.....
03549430 000040 00000000 00000000 00000000 00000000 .....
03549440 000050 D7C1D5D3 00000000 D9C5C3C4 00000000 PANL....RECD....
03549450 000060 D3C9E2E3 00000000 C8C5D3D7 00000000 LIST....HELP....
03549460 000070 84000000 00000000 A4000000 00000000 d.....u.....
03549470 000080 82000000 00000000 C4000000 00000000 b.....D.....
03549480 000090 E4000000 00000000 C2000000 00000000 U.....B.....
03549490 0000A0 D5000000 00000000 E2000000 00000000 N.....S.....
035494A0 0000B0 7B000000 00000000 6C000000 00000000 #.....%.....
035494B0 0000C0 4A000000 00000000 F1000000 00000000 ¢.....1.....
035494C0 0000D0 F2000000 00000000 F3000000 00000000 2.....3.....

ENTER: CURRENT DISPLAY
PF1 : UNDEFINED PF2 : BROWSE TEMP STORAGE PF3 : UNDEFINED
PF4 : EIB DISPLAY PF5 : INVOKE CECI PF6 : USER DISPLAY
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: REMEMBER DISPLAY

```

그림 53. 작업 중인 저장영역에 대한 일반 EDF 표시장치

작업 중인 저장영역 내용은 덤프 목록과 유사한 양식(즉, 16진수 및 문자 표시 둘 다로 된)으로 표시됩니다. 작업 중인 저장영역의 주소는 화면의 맨 위에 표시됩니다. 화면 이동 명령을 사용하여 전체 영역을 통해 찾아보거나 화면의 맨 위에서 새 주소를 입력할 수 있습니다. 이 주소는 CICS 리전 어디에도 있을 수 있습니다. 작업 중인 저장영역 표시장치는 두 개의 추가적 화면 이동 키와, EIB(명령이 DL/I 명령인 경우 DIB)를 표시할 키를 제공합니다.

작업 중인 저장영역의 의미는 다음과 같이 애플리케이션의 프로그래밍 언어에 따라 다릅니다.

COBOL

프로그램의 WORKING-STORAGE 섹션에 정의된 모든 데이터 저장영역

C, C++ 및 PL/I

현재 프로시저의 동적 저장영역(DSA)

어셈블러 언어

현재 DFHEISTG DSECT에서 정의된 저장영역

어셈블러 언어 프로그램은 항상 작업 중인 저장영역을 획득하지는 않습니다. 예를 들어, 프로그램이 CICS 명령을 실행하지 않는 경우에는 필요하지 않을 수 있습니다. 이와 같은 프로그램에 LINK할 때, Register 13 does not address DFHEISTG 메시지가 발행될 수 있습니다. 이 메시지는 반드시 오류를 의미하지는 않지만, 찾아 볼 작업 중인 저장영역이 전혀 없습니다.

COBOL 프로그램을 제외하고, 작업 중인 저장영역은 72바이트 표준 형식 저장 영역으로 시작합니다. 즉, 레지스터 14 - 12는 오프셋 12에서 시작하고, 레지스터 13은 오

프셋 4에서 저장됩니다. AMODE(64) 프로그램의 경우, 작업 중인 저장영역은 형식 4 저장 영역(F4SA)에서 시작합니다. 레지스터 14 - 12는 오프셋 8에서 시작하고, 레지스터 13은 오프셋 128에서 저장됩니다.

작업 중인 저장영역은 화면에서 변경될 수 있습니다. 16진수 섹션이나 문자 섹션을 사용할 수 있습니다. 16진 주소를 사용하는 표시장치의 헤드에서 ADDRESS 필드 위에 입력할 수 있습니다. 그러면 ENTER를 누를 때 해당 주소에서 시작하는 저장영역이 표시됩니다. 주소 공간에서 위치를 조사할 수 있습니다. 자세한 정보는 219 페이지의 『EDF를 사용하여 정보 변경』의 내용을 참조하십시오.

테스트할 특정 트랜잭션에 고유한, 현재 실행 중인 프로그램의 작업 중인 저장영역의 파트가 아닌 프로그램 저장영역을 조사 중인 경우, 화면에서 해당 필드는 다른 태스크에 속하거나 영향을 미칠지도 모르는 저장영역 위에 겹쳐쓰지 않도록 보호됩니다.

작업 중인 저장영역 표시장치 행의 초기 파트가 비어 있으면, 공백 부분은 작업 중인 저장영역의 일부가 아닙니다. 이 상황은 표시장치가 2배 단어 정렬이어서 발생할 수 있습니다.

태스크 맨 앞이나 끝에서는 작업 중인 저장영역을 사용할 수 없습니다. 이러한 상황에서, 주소 필드 위에 입력하여 리전에서 저장영역을 계속 조사할 수 있도록 EDF는 공백 저장영역 표시장치를 생성합니다.

보통의 비CICS 리턴으로 PL/I 또는 Language Environment 프로그램을 종료하면, EDF는 리턴을 인터셉트하지 않으므로 사용자가 작업 중인 저장영역을 볼 수 없습니다. 그 대신 RETURN 명령을 사용하면, 실행 이전과 프로그램 종료 시 EDF 표시장치가 표시됩니다.

Language Environment가 가능한 프로그램을 사용 중인 경우, 프로그램이 비CICS 리턴을 사용하여 종료되는 경우, 프로그램 종료 시 작업 중인 저장영역이 사용 가능하게 됩니다. 이러한 경우, 작업 중인 저장영역은 표시장치에 대해 사용할 수 없습니다.

제 13 장 임시 저장영역 찾아보기(CEBR)

찾아보기 트랜잭션(CEBR)을 사용하여 임시 저장영역 큐를 찾고 삭제할 수 있습니다. CEBR 트랜잭션을 사용하여 트랜지언트 데이터 큐의 내용을 확인하고 완료 시 트랜지언트 데이터 큐를 재설정하기 위해 임시 저장영역으로 전송할 수도 있습니다.

이 전송을 수행하는 CEBR 명령으로 트랜지언트 데이터 큐에 레코드를 추가하고 트랜지언트 데이터 큐에서 모든 레코드를 제거할 수 있습니다.

일부 설치의 의도하지 않았거나 권한이 없는 수정을 피하기 위해 특히, 프로덕션 시스템의 CEBR 사용을 제한합니다. 설치의 임시 저장영역 및 트랜지언트 데이터 큐를 포함하여 개별 자원을 보호할 수도 있습니다. CEBR 트랜잭션을 사용 중이며 보안 실패로 설명된 이상 종료를 경험한 경우에는 아마도 사용자 ID에 권한이 없는 큐에 액세스하려 시도한 것입니다.

이 장에서는 다음에 대해 설명합니다.

- 『CEBR 트랜잭션 사용』
- 233 페이지의 『CEBR 트랜잭션이 표시하는 내용』
- 234 페이지의 『CEBR 기능 키 사용』
- 236 페이지의 『CEBR 명령 사용』
- 238 페이지의 『트랜지언트 데이터에 CEBR 트랜잭션 사용』

CEBR 트랜잭션 사용

트랜잭션 ID CEBR을 입력하고 그 뒤에 찾으려는 큐의 이름을 지정해서 CEBR 트랜잭션을 시작합니다.

최대 16자의 이름을 입력할 수 있습니다. 예를 들어, AXBYQUEUENAME111이란 임시 저장영역 큐의 이름을 표시하려면 CEBR AXBYQUEUENAME111을 입력하고 ENTER를 누릅니다. 큐 이름에 소문자가 있으면 사용 중인 터미널에 대한 대문자 변환이 비활성화되었는지 확인한 후 대문자와 소문자의 올바른 조합을 입력하십시오. CICS는 예를 들어, 232 페이지의 그림 54에 표시된 것처럼 큐 표시로 응답합니다.

또는 CEDF 트랜잭션에서 CEBR 트랜잭션을 시작할 수 있습니다. 초기 CEDF 화면(205 페이지의 그림 38 참조)에서 PF5를 누른 후(그러면 작업 중인 저장영역으로 이동함) 해당 화면에서 PF2를 눌러 임시 저장영역을 찾아서(즉, CEBR 트랜잭션을 호출해서) 이를 수행합니다. CEBR은 찾을 큐에 'b'를 입력하여 CEMT I TSQ에서도 시작할 수 있습니다. CEBR 트랜잭션은 이름이 네 개의 문자 CEBR 및 뒤이어 터미널 ID 중 네 개의 글자로 이루어진 임시 저장영역 큐를 표시해서 응답합니다. (CICS는 사용자가

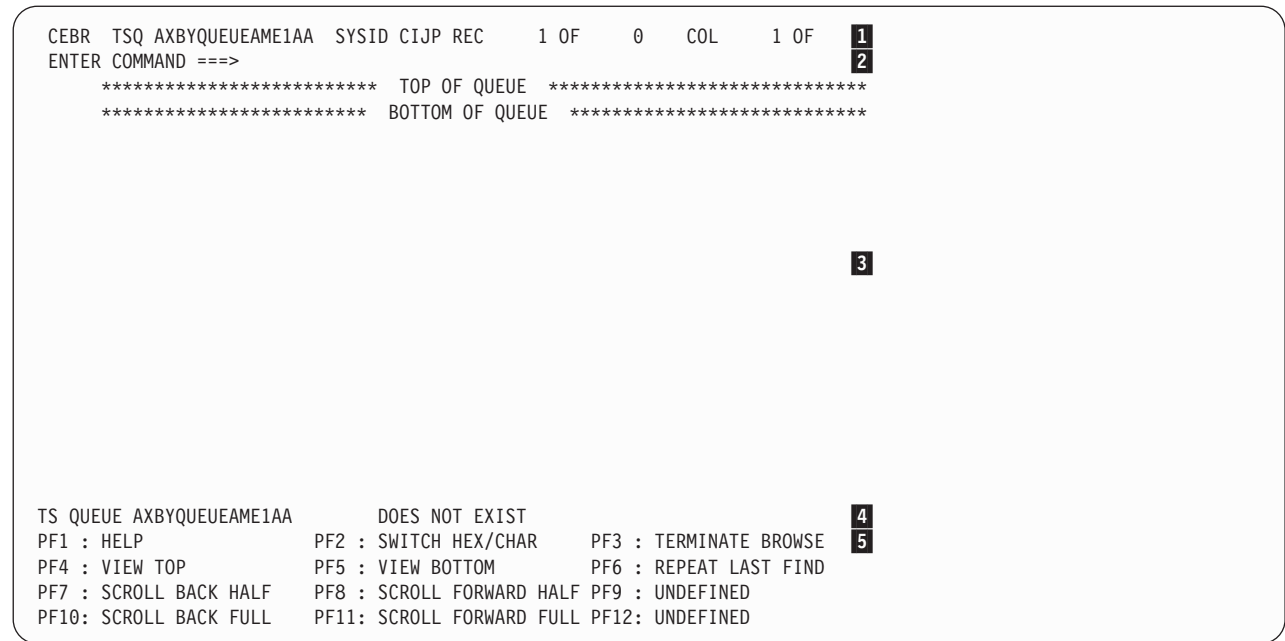
CEBR 트랜잭션을 직접 호출하고 큐 이름을 제공하지 않을 경우 이 동일한 기본 큐 이름을 사용합니다.) S21A 터미널의 EDF 세션에서 또는 큐 이름 없이 CEBR 트랜잭션을 호출한 결과는 233 페이지의 그림 55에 표시되어 있습니다. CEDF 트랜잭션에서 CEBR 트랜잭션을 입력할 경우 CEBR 화면에서 PF3을 누르면 EDF 패널로 돌아옵니다.

CEBR을 사용하여 큐 이름에 하나 이상의 임베디드 공백이 있는 임시 저장영역 큐에서 데이터를 표시하려는 경우에는 그림 54에 표시된 대로, CEBR 화면에서 TS QUEUE 필드에 큐 이름을 입력해야 합니다. CEBR을 입력한 직후에 이러한 큐 이름을 입력할 경우 예측 불가능한 결과가 발생합니다.

```
CEBR TSQ AXBYQUEUENAME111 SYSID CIJP REC 1 OF 3 COL 1 OF 5
ENTER COMMAND ==>
***** TOP OF QUEUE *****
00001 HELLO
00002 HELLO
00003 HELLO
***** BOTTOM OF QUEUE *****

PF1 : HELP          PF2 : SWITCH HEX/CHAR    PF3 : TERMINATE BROWSE
PF4 : VIEW TOP      PF5 : VIEW BOTTOM      PF6 : REPEAT LAST FIND
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL PF12: UNDEFINED
```

그림 54. 임시 저장영역 큐 내용의 일반 CEBR 표시



- 1 헤더
- 2 명령 영역
- 3 본문
- 4 메시지 행
- 5 옵션의 메뉴

그림 55. 기본 임시 저장영역 큐의 일반 CEBR 표시

CEBR 트랜잭션이 표시하는 내용

CEBR 트랜잭션 표시장치는 헤더, 명령 영역, 본문(기본 표시 영역), 메시지 행, 현재 선택할 수 있는 함수 메뉴로 이루어져 있습니다.

헤더

헤더에는 다음 사항이 표시됩니다.

- 실행 중인 트랜잭션 즉, CEBR.
- 임시 저장영역 큐의 ID(232 페이지의 그림 54의 AXBYQUEUEAME111 및 그림 55의 AXBYQUEUEAME1AA). 화면을 또 다른 큐로 전환하려는 경우 헤더의 이 필드를 겹쳐쓸 수 있습니다. 큐 이름에 소문자가 있으면 사용 중인 터미널에 대한 대문자 변환이 비활성화되었는지 확인한 후 대문자와 소문자의 올바른 조합을 입력하십시오.

- 임시 저장영역 풀 이름이나 원격 시스템에 해당하는 시스템 이름. 이름을 지정하지 않은 경우 로컬 시스템의 이름이 표시됩니다. 공유 또는 원격 큐를 찾아보려는 경우 헤더의 이 필드를 겹쳐쓸 수 있습니다.
- 강조표시된 레코드 수.
- 큐의 레코드 수(AXBYQUEUEAME111에 3개 그리고 AXBYQUEUEAME1AA에는 없음)
- 각 레코드에서 화면이 시작하는 위치(두 경우 모두 위치 1) 및 가장 긴 레코드의 길이(AXBYQUEUEAME111 큐의 경우 22 및 AXBYQUEUEAME1AA 큐의 경우 0).

명령 영역

명령 영역은 표시할 사항과 수행할 기능을 제어할 명령을 입력하는 곳입니다.

이러한 명령은 236 페이지의 『CEBR 명령 사용』에서 설명합니다. 화면 맨 아래의 옵션 메뉴에 표시된 기능 키로 화면을 수정할 수도 있습니다. 기능 키는 『CEBR 기능 키 사용』에서 설명합니다.

본문

본문은 큐 레코드가 표시되는 곳입니다. 화면의 각 행이 큐 레코드에 일치합니다.

레코드가 행에 비해 너무 긴 경우에는 잘립니다. 하지만 후속 화면에서 전체 레코드를 볼 수 있도록 표시되는 레코드 부분을 변경할 수 있습니다. 화면에 맞는 것보다 더 많은 레코드가 큐에 있으면 페이지를 앞으로 및 뒤로 넘기거나 표시를 시작할 레코드를 지정해서 원하는 모든 레코드를 볼 수 있습니다.

메시지 행

CEBR은 본문과 메뉴 사이의 메시지 행을 사용하여 사용자에게 메시지를 표시합니다.

예를 들어, 233 페이지의 그림 55에 표시된 『Does not exist』 메시지가 있습니다.

CEBR 기능 키 사용

언제든지 사용할 수 있는 기능 키는 모든 CEBR 트랜잭션 화면의 맨 아래에 표시되며 모든 화면에서 의미가 동일합니다.

터미널에 PF 키가 없으면 설명 아래에 커서를 두고 ENTER를 눌러서 사용을 시뮬레이션할 수 있습니다. 터미널의 기능 키가 24개인 경우 CEBR 트랜잭션은 PF13 - PF24를 각각 PF1 - PF12의 중복으로 처리합니다.

PF1 HELP

CEBR 트랜잭션이 실행 중일 때 사용할 수 있는 모든 명령을 나열하는 도움말 화면을 표시합니다. ENTER를 눌러서 기본 화면으로 돌아갈 수 있습니다.

PF2 SWITCH HEX/CHAR

화면을 문자에서 16진 형식으로 그리고 다시 반대로 전환합니다.

PF3 TERMINATE BROWSE

CEBR 트랜잭션을 종료합니다. CEBR 트랜잭션을 직접 입력한 경우 다음 트랜잭션을 위해 터미널을 해제합니다. EDF 세션에서 입력했으면 입력한 작업 중인 저장 영역 화면을 리턴합니다. CEMT I TSQ에서 입력한 경우에는 CEMT 화면을 리턴합니다.

PF4 VIEW TOP

큐의 첫 번째 레코드를 표시하며 TOP 명령과 효과가 동일합니다.

PF5 VIEW BOTTOM

큐의 마지막 레코드를 표시하며 BOTTOM 명령과 효과가 동일합니다.

PF6 REPEAT LAST FIND

이전 FIND 명령을 반복합니다.

PF7 SCROLL BACK HALF

화면의 위쪽 절반 레코드가 아래 절반으로 이동하도록 화면에 맞는 레코드 수 절반만큼 표시장치를 뒤로 이동시킵니다.

PF8 SCROLL FORWARD HALF

화면의 아래쪽 절반 레코드가 위쪽 절반으로 이동하도록 화면에 맞는 레코드 수 절반만큼 표시장치를 앞으로 이동시킵니다.

PF9 VIEW RIGHT(또는 VIEW LEFT)

표시장치에 현재 있는 열 바로 다음(오른쪽에)이나 바로 이전(왼쪽에)의 열을 표시하도록 화면을 변경합니다. 전체 레코드가 화면의 한 행과 잘 맞는 경우에는 키가 정의되지 않습니다. 레코드 끝에 도달할 때까지 오른쪽으로 이동한 후 반대쪽 레코드 처음을 향해 다시 왼쪽으로 이동합니다. COLUMN 명령을 사용하여 표시장치가 시작하는 열을 변경할 수도 있습니다.

PF10 SCROLL BACK FULL

표시장치에 현재 있는 레코드의 바로 이전 레코드를 표시하기 위해 화면에 잘 맞는 레코드 수만큼 화면을 뒤로 이동합니다.

PF11 SCROLL FORWARD FULL

표시장치에 현재 있는 레코드의 바로 다음 레코드를 표시하기 위해 화면에 잘 맞는 레코드 수만큼 화면을 앞으로 이동합니다.

CEBR 명령 사용

CEBR은 임시 저장영역 큐의 레코드를 보고 조작하는 데 사용할 수 있는 많은 명령을 제공합니다.

BOTTOM

(약어: B)

임시 저장영역 큐의 마지막 레코드를 표시합니다(마지막 행의 마지막 레코드로 화면의 본문을 채울 만큼).

COLUMN nnnn

(약어: C nnnn)

각 레코드의 문자 위치(열) nnnn에서 시작하는 레코드를 표시합니다. CEBR 트랜잭션을 시작할 때 가정한 기본 시작 위치는 레코드의 첫 번째 문자입니다.

FIND /string

(약어: F /string)

지정된 문자열의 다음 발생을 찾습니다. 현재 레코드 다음의 레코드에서 검색을 시작합니다. 현재 레코드는 강조표시된 레코드입니다. 초기 큐 표시에서 현재 레코드가 1로 설정되므로 검색은 레코드 2에서 시작합니다.

문자열을 찾으면 문자열을 포함한 레코드가 강조표시된 행이 되고 표시장치는 두 번째 행의 레코드를 표시하도록 변경됩니다. FIND가 성공한 후에도 검색 문자열을 찾을 수 없는 경우 이는 문자열이 표시장치의 열을 벗어난 레코드 열에 있기 때문입니다. 화면이동 키나 COLUMN 명령을 사용하여 문자열을 표시하도록 표시장치의 오른쪽 또는 왼쪽으로 이동하십시오.

예를 들어, 다음과 같습니다.

```
FIND /05-02-93
```

문자열 『05-02-93』의 다음 발생을 찾습니다. / 문자는 구분 기호입니다. 반드시 /를 사용할 필요는 없지만 검색 인수에 표시되는 문자를 구분 기호로 사용해선 안 됩니다. 예를 들어, 찾고 있는 문자열이 『05-02-93』이 아닌 『05/02/93』이면 다음을 사용할 수 없습니다.

```
FIND /05/02/93
```

검색 문자열에 슬래시가 있습니다. 다음 예제는 작동합니다.

```
FIND X05/02/93      or      FIND S05/07/93
```

문자열의 숫자 중 하나 또는 /를 제외한 구분 기호는 작동합니다. 검색 문자열에 공백이 있는 경우에는 문자열의 끝에서 구분 기호를 반복해야 합니다. 예를 들어, 다음과 같습니다.

```
FIND /CLARE JACKSON/
```

검색 문자열은 대소문자를 구분하지 않습니다. FIND 명령을 입력했으면 PF6을 눌러 이를 반복할 수 있습니다(즉, 문자열의 다음 발생을 찾음).

GET xxxx

(약어: G xxxx)

이름 지정된 트랜지언트 데이터 큐를 현재 표시장치에 있는 임시 저장영역 큐의 끝으로 전송합니다. 이를 통해 큐의 내용을 찾아볼 수 있습니다. xxxx는 리전 내부 트랜지언트 데이터 큐의 이름 또는 입력을 위해 연 리전 외부 트랜지언트 데이터 큐의 이름이어야 합니다. 트랜지언트 데이터 큐 찾아보기에 대한 자세한 정보는 238 페이지의 『트랜지언트 데이터에 CEBR 트랜잭션 사용』의 내용을 참조하십시오.

LINE nnnn

(약어: L nnnn)

nnnn보다 하나 앞선 큐 레코드에서 화면의 본문을 시작하고 현재 행을 nnnn으로 설정합니다. (이 배열은 후속 FIND 명령이 레코드 nnnn 다음에서 검색을 시작하도록 합니다.)

PURGE

찾아보고 있는 큐를 삭제합니다.

BMS 논리 메시지와 같이 내부적으로 생성된 큐의 내용을 삭제하기 위해 PURGE를 사용하지 마십시오.

참고: 복구 가능한 임시 저장영역 큐를 영구 제거하면 태스크가 종료될 때까지 다른 어떤 태스크도 큐를 갱신(레코드 추가, 레코드 변경 또는 영구 제거)할 수 없습니다.

PUT xxxx

(약어: P xxxx)

찾아보고 있는 임시 저장영역 큐를 이름 지정된 트랜지언트 데이터 큐에 복사합니다. xxxx는 리전 내부 트랜지언트 데이터 큐의 이름 또는 출력을 위해 연 리전 외부 트랜지언트 데이터 큐의 이름이어야 합니다. 트랜지언트 데이터 큐 작성 또는 복원에 대한 자세한 정보는 238 페이지의 『트랜지언트 데이터에 CEBR 트랜잭션 사용』의 내용을 참조하십시오.

QUEUE xxxxxxxxxxxxxxxxx

(약어: Q xxxxxxxxx)

찾고 있는 큐의 이름을 변경합니다. 지정하는 값은 최대 16자를 사용한 문자 형식(예: QUEUE ABCDEFGHIJKLMNOP) 또는 16진 형식(예: QUEUE X'C1C2C3C4')이 가능합니다. 큐 이름에 소문자가 있으면 사용 중인 터미널에 대한 대문자 변환이 비활성화되었는지 확인한 후 대문자와 소문자의 올바른 조합을 입력하십시오. CEBR 트랜잭션은 이름 지정된 큐에 있는 데이터를 표시해서 응답합니다.

헤더의 현재 값을 겹쳐써서 큐 이름을 변경할 수도 있습니다.

SYSID xxxx

(약어: S xxxx)

큐를 찾을 임시 저장영역 풀이나 원격 시스템의 이름을 변경합니다.

헤더의 현재 SYSID 값을 겹쳐써서 이 이름을 변경할 수도 있습니다.

참고: CEBR 트랜잭션이 실행 중인 CICS 시스템에서 ISC가 활성화되지 않은 경우 SYSID의 기본값은 로컬 SYSID입니다.

TERMINAL xxxx

(약어: TERM xxxx)

상수의 처음 네 자와 터미널 이름의 마지막 네 자로 터미널과 연관된 임시 저장영역 큐의 이름을 지정할 규칙을 사용하는 애플리케이션에 맞게 조정된, 찾고 있는 큐의 이름을 변경합니다. 새 큐 이름의 양식은 현재 큐 이름의 처음 네 자와 뒤따르는 xxxx로 구성됩니다.

TOP

(약어: T)

CEBR 트랜잭션이 큐의 첫 번째 레코드에 표시를 시작하도록 합니다.

트랜지언트 데이터에 CEBR 트랜잭션 사용

GET 명령은 지정한 트랜지언트 데이터 큐의 각 레코드를 읽고 찾아보고 있는 임시 저장영역 큐의 끝에 트랜지언트 데이터 큐가 비어 있게 될 때까지 이 레코드를 씁니다. 그러면 트랜지언트 데이터 큐에 있는 레코드를 볼 수 있습니다.

검사가 완료되면 임시 저장영역 큐를 다시 트랜지언트 데이터 큐에 복사할 수 있습니다(PUT 명령을 사용하여). 그러면 일반적으로 트랜지언트 데이터 큐는 처음 찾았을 때와 같이 되지만 항상 그런 것은 아닙니다. 다음은 GET 및 PUT 명령을 사용할 때 유의해야 하는 몇 가지 사항입니다.

- 트랜지언트 데이터 큐를 찾아본 후 변경되지 않은 상태로 복원하려면 GET 명령 시점에 표시장치의 임시 저장영역 큐가 비어 있는지 확인하십시오. 그렇지 않을 경우 후속 PUT 명령을 실행할 때 기존 임시 저장영역 레코드가 트랜지언트 데이터 큐에 복사됩니다.
- 트랜지언트 데이터 큐를 가져온 후 다시 제자리에 돌려놓기 전에 다른 태스크가 이 트랜지언트 데이터 큐에 쓸 수 있습니다. PUT 명령을 실행하면 임시 저장영역 큐의 레코드가 새 레코드 다음에 복사되어, 큐의 레코드가 더 이상 원래 작성된 순서가 아닙니다. 일부 애플리케이션은 큐의 순차 레코드 처리에 따라 다릅니다.
- 복구 가능 트랜지언트 데이터 큐를 가져오고 나면 트랜잭션이 종료될 때까지 다른 태스크가 이 큐처리에 액세스할 수 없습니다. CEBR 트랜잭션을 CEDF 트랜잭션에서

입력한 경우 트랜잭션의 의사 대화식 순서를 디버깅 중인지 묻는 『계속』 질문에 『예』로 응답할 수 있어도 CEDF 트랜잭션을 종료해야 합니다. CEBR 트랜잭션을 직접 입력한 경우 이를 종료해야 합니다.

- 이와 마찬가지로, 복구 가능 트랜지언트 데이터 큐에 PUT 명령을 실행하고 난 후에는 트랜잭션이 종료될 때까지 다른 태스크가 이 큐에 액세스할 수 없습니다.

GET 및 PUT 명령은 쌍으로 사용할 필요가 없습니다. PUT 명령으로 언제든지 임시 저장영역 큐에서 트랜지언트 데이터 큐에 추가할 수 있습니다. 트랜지언트 데이터 큐를 읽는 코드를 디버깅 중이면 임시 저장영역에 큐를 작성(CECI 트랜잭션이나 CEBR GET 명령으로 또는 프로그램으로)한 후 임시 저장영역에서 원하는 대로 트랜지언트 데이터 큐를 새로 고칠 수 있습니다. 마찬가지로 해당하는 PUT 명령 없이 GET 명령을 사용하여 트랜지언트 데이터 큐를 비울 수 있습니다.

제 14 장 명령 레벨 해독기(CECI)

명령 레벨 해독기(CECI) 트랜잭션을 사용하여 CICS 명령의 구문을 검사하고 3270 화면에서 이 명령을 대화식으로 처리할 수 있습니다. CECI를 통해 대부분의 명령을 실행하고 결과를 표시할 수 있습니다.

CECI는 전체 CICS 명령 레벨 애플리케이션 프로그래밍과 시스템 프로그래밍 인터페이스의 구문에 대한 참조도 제공합니다.

CECI는 테스트 시스템과 상호작용하여 테스트 데이터, 임시 저장영역 큐를 작성 또는 삭제하거나 오류 논리를 테스트하기 위해 의도적으로 잘못된 데이터를 소개할 수 있습니다. CECI를 사용하여 프로덕션 시스템의 손상된 데이터베이스 레코드를 수리할 수도 있습니다.

해석기는 이처럼 설치 시 접속 시간 보안으로 사용을 제한할 수 있는 강력한 도구입니다. (설치에 사용된 외부 보안 관리자는 CECI 및 CECS 트랜잭션의 보안 속성을 정의합니다.) 이와 같이 보안 속성이 정의되었고 사용자에게 선택한 해석기 트랜잭션을 사용할 권한이 없는 경우에는 트랜잭션을 시작할 수 없습니다.

이 장에서는 다음에 대해 설명합니다.

- 『CECI가 표시하는 내용』
- 247 페이지의 『CECI 사용』
- 250 페이지의 『명령 저장』
- 251 페이지의 『CECI의 실행 방식』

CECI가 표시하는 내용

모든 CECI 화면에는 동일한 기본 레이아웃이 있습니다. CECI 표시장치는 명령행, 상태 표시줄, 화면 본문, 메시지 행, 기능 키의 CECI 옵션으로 이루어집니다.

- 『명령행』
- 242 페이지의 『상태 표시줄』
- 245 페이지의 『화면 본문』
- 246 페이지의 『메시지 행』
- 246 페이지의 『기능 키의 CECI 옵션』

명령행

명령행은 화면의 첫 번째 행입니다. 처리할 명령이나 해당 구문을 확인할 명령을 여기에 입력합니다. 전체 또는 축약된 구문이 될 수 있습니다.

명령 입력 및 축약 규칙은 다음과 같습니다.

- 키워드 EXEC CICS는 선택사항입니다.
- 명령의 옵션은 고유하게 만들기에 충분한 문자 수로 축약될 수 있습니다. 올바른 약어는 화면 본문의 구문 표시장치에 대문자로 표시됩니다.
- 문자열을 묶는 따옴표(')는 선택사항이며, 모든 문자열은 앞에 앰퍼샌드(&)가 없으면 문자열 상수로 처리됩니다. 이러한 경우 문자열은 변수로 처리됩니다.
- 명령이 처리될 때 CICS로부터 값을 수신하는 명령의 옵션은 수신인이라고 하며, 지정하지 않아도 됩니다. 명령이 처리된 후, CICS에서 수신된 값은 구문 표시장치에 포함되고, 변수(지정된 경우)에 저장됩니다.
- 두 개의 키워드가 충돌되는 CECI 명령을 실행하는 경우, CECI는 첫 번째 키워드를 무시하고 오류 메시지(READ 명령의 다음 메시지와 같은)를 발행합니다.

E INTO option conflicts with SET option and is ignored

- 명령 앞에 물음표를 두면, 트랜잭션 코드 CECI를 사용한 경우에도 해독기가 구문 검사 후 중지됩니다. 실행을 진행하려면 물음표를 제거하십시오.

다음 예제는 명령의 축약된 양식을 보여줍니다. 다음 파일 제어 명령은

```
EXEC CICS READ FILE('FILEA') RIDFLD('009000') INTO(&REC)
```

다음과 같이 명령 입력 행에 입력할 수 있습니다.

```
READ FIL(FILEA) RID(009000)
```

또는 다음과 같이 최소한으로 입력할 수 있습니다.

```
READ F(FILEA) RI(009000)
```

첫 번째 양식에서, INTO 스펙은 데이터를 읽어들이는 변수 &REC를 작성합니다. 그러나 INTO는 수신인이므로(위에 정의된 것처럼) 생략할 수 있습니다. 생략하면, CICS에서 자동으로 변수가 작성됩니다.

상태 표시줄

명령을 해석하는 프로세스 과정에서, CECI는 일련의 표시장치를 제시합니다. 화면 본문의 형식은 근본적으로 모든 화면에 대해 동일합니다. 명령 구문과 선택된 옵션 값을 표시합니다.

이러한 화면의 상태 표시줄은 명령 처리 중 현재 위치를 알려주며, 다음 중 하나입니다.

- COMMAND SYNTAX CHECK
- ABOUT TO EXECUTE COMMAND
- COMMAND EXECUTION COMPLETE
- COMMAND NOT EXECUTED

이 화면에서, 추가 표시장치를 선택할 수 있습니다. 선택할 때, 화면 본문은 요청한 정보를 표시하고, 상태 표시줄은 다음과 같을 수 있는 표시장치를 식별합니다.

- EXPANDED AREA
- VARIABLES
- EXEC INTERFACE BLOCK
- SYNTAX MESSAGES

처리 중에 언제든지 표시장치를 요청하고 명령 해석 순서로 리턴할 수 있습니다.

상태 표시줄에는 NAME=이라고 하는 하나의 입력 필드도 있습니다. 이 필드는 변수를 작성하고 이름을 지정하는데 사용됩니다.

Command syntax check

상태 표시줄에 **command syntax check**가 표시되면, 이는 명령 입력 행에 입력한 명령에 대해 구문이 검사되었지만 처리되지 않음을 표시합니다.

사용자가 CECS를 입력하거나 명령 앞에 물음표를 입력하는 경우 항상 이 상태입니다. 또한 명령의 구문 검사에서 심각한 오류 메시지를 제공하는 경우의 상태이기도 합니다.

또한 해독기가 실행할 수 없는 명령 중 하나를 실행하려고 하는 경우에 이 상태가 됩니다. CECS 또는 CECI를 사용하여 어떤 명령도 구문을 검사할 수 있더라도, 해독기는 더 이상 다음 명령을 처리할 수 없습니다.

- 해독기가 제공하지 않는 환경에 의존하는 EXEC CICS 명령:
 - FREE
 - FREEMAIN
 - GETMAIN
 - HANDLE ABEND
 - HANDLE AID
 - HANDLE CONDITION
 - IGNORE CONDITION
 - POP HANDLE
 - PUSH HANDLE
 - SEND LAST
 - SEND PARTNSET
 - WAITCICS
 - WAIT EVENT
 - WAIT EXTERNAL

- 파티션을 참조하는 BMS 명령(화면이 파티션된 후 표시장치를 복원할 수 없기 때문입니다.)
- EXEC DLI
- CPI 통신(CPI-C) 명령
- SAA 자원 복구 인터페이스(CPI-RR) 명령

About to execute command

다음 예제는 about to execute command의 일반 CECI 표시장치를 보여줍니다.

이 화면(그림 56에 표시된)은 **command syntax check**에서 중지할 어떤 이유도 적용되지 않을 때 나타납니다.

```

READ FILE('FILEA') RIDFLD('009000')
STATUS: ABOUT TO EXECUTE COMMAND                                NAME=
EXEC CICS READ
  File( 'FILEA  ' )
  < SYsid() >
  SEt() | Into()
  < Length() >
  RIDfld( '009000' )
  < Keylength() < GEneric > >
  < RBa | RRn | DEBRec | DEBKey >
  < GTeq | Equal >
  < Update < Token() > >

```

PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF

그림 56. about to execute command의 일반 CECI 표시장치

화면을 변경하지 않고 ENTER 키를 누르면 CECI는 명령을 실행합니다. 그러나 계속 화면을 수정할 수 있습니다. 수정하는 경우 CECI는 이전 명령을 무시하고 처음부터 새 명령을 처리합니다. 표시되는 다음 화면은 명령을 실행할 수 없는 경우 **command syntax check**이고 명령이 올바르면 **about to execute command**임을 의미합니다.

CICS 명령 완료

다음 예제는 command execution complete의 일반 CECI 표시장치를 보여줍니다.

다음 표시장치(245 페이지의 그림 57에 표시된)는 해독기가 수정되지 않은 **about to execute command** 화면의 ENTER 키에 대한 응답으로 명령을 실행한 후 나타납니다.

```

INQUIRE FILE NEXT
STATUS:  COMMAND EXECUTION COMPLETE                                NAME=
EXEC CICS INQUIRE File( 'DFHCSD ' )
  < START | END | Next >
  < ACCESSMETHOD( +0000000003 ) >
  < ADD( +00000000041 ) >
  < BASEDSNAME( ' ' ) >
  < BLOCKFORMAT( +0000000016 ) >
  < BLOCKKEYLEN( -0000000001 ) >
  < BLOCKSIZE( -0000000001 ) >
  < BROWSE( +00000000039 ) >
  < CFDTPOOL( ' ' ) >
  < DELETE( +00000000043 ) >
  < DISPOSITION( +0000000027 ) >
  < DSNAME( 'CFV01.CICS03.PSK.CSD ' ) >
  < EMPTYSTATUS( +0000000032 ) >
  < ENABLESTATUS( +0000000033 ) >
  < EXCLUSIVE( +0000000001 ) >
  < FWDRECSTATUS( +0000000361 ) >
+ < JOURNALNUM( +00000 ) >

RESPONSE: NORMAL          EIBRESP=+0000000000 EIBRESP2=+0000000000
PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF

```

그림 57. *command execution complete*의 일반 CECI 표시장치

명령이 처리되었고 결과가 화면에 표시됩니다.

해당되는 CICS 제공 값과 함께 수신인(지정 여부에 관계없이)이 강화되어 표시됩니다.

화면 본문

CECI 화면 본문에는 세 개의 표시장치 모두에 공통되는 정보가 포함됩니다.

명령의 전체 구문이 표시됩니다. 명령행에 지정되거나 기본적으로 간주되는 옵션은 명령 실행 시 수신인인 것처럼 사용됨을 표시하기 위해 강조됩니다. < > 대괄호는 대괄호 내에서 옵션을 선택할 수 있음을 표시합니다. 구문에서 오류가 발생하면, CECI는 본문 다음에 있는 메시지 영역에서 이 오류를 진단하며, 246 페이지의 『메시지 행』에 설명되어 있습니다. 진단 메시지가 너무 많은 경우 나머지 메시지는 PF9를 사용하여 표시할 수 있습니다.

인수는 문자나 16진 형식으로 표시할 수 있습니다. PF2를 사용하여 형식 사이에 전환할 수 있습니다. 문자 형식에서, 일부 문자는 표시 가능하지 않습니다. (일부 터미널에 소문자가 표시되지 않습니다.) CECI는 이 문자를 마침표로 표시합니다. 실수 값을 표시하기 위해 16진수로 전환해야 하므로, 수정할 때 주의해야 합니다.

옵션의 값이 행에 비해 너무 긴 경우, 첫 번째 파트만 표시되고 그 뒤로 더 있음을 표시하기 위해 『...』가 있습니다. 옵션 값의 시작 부분에 커서를 위치시키고 Enter를 눌러서 전체 값을 표시할 수 있습니다. 이 조치는 펼쳐진 표시장치를 생성합니다.

명령에 하나의 화면에 맞출 수 있는 것보다 많은 옵션이 있는 경우, 더 있음을 표시하기 위해 현재 표시장치의 마지막 옵션 왼쪽에 더하기 부호(+)가 나타납니다. 이에 대한 예제가 244 페이지의 『CICS 명령 완료』에 있습니다. PF 키로 화면 이동하여 추가 페이지를 표시할 수 있습니다.

메시지 행

CECI는 오류 메시지를 표시하기 위해 메시지 행을 사용합니다. 명령 실행 후, 메시지 행은 응답 코드를 보여줍니다.

메시지 앞에 있는 **S**는 심각함을 나타냅니다(실행을 금지하기에 충분히 나쁨). 실행을 금지하지 않고 정보를 제공하는 경고 메시지(**W** 플래그 지정)와 오류 메시지(**E** 플래그 지정)도 있습니다. **E** 메시지는 실행을 진행하기 전에 명령 검토를 유도하고 보장하지 않을 만큼 충분히 비정상적인 옵션 조합을 표시합니다.

여러 개의 오류 메시지가 있는 경우, CECI는 모든 오류 메시지를 포함하는 별도의 표시장치를 작성하고, 메시지 수와 심각도를 알려주기 위해 메시지 행을 사용합니다. PF9로 메시지 표시장치를 가져올 수 있습니다.

245 페이지의 그림 57에서는 명령 실행 결과를 보여주기 위해, 메시지 행의 두 번째 사용을 보여줍니다. CECI는 텍스트(245 페이지의 그림 57의 예제에서 NORMAL) 및 10진수 양식(EIBRESP 및 EIBRESP2 값) 둘 다를 정보를 제공합니다.

기능 키의 CECI 옵션

화면의 맨 아래에 있는 단일 행은 표시장치의 기능 키 효과를 표시하는 메뉴를 제공합니다.

기능 키는 아래에 설명되어 있습니다. 터미널에 어떤 기능 키도 없는 경우, 메뉴에서 필수 항목 아래에 커서를 위치시키고 ENTER를 눌러서 동일한 효과를 얻을 수 있습니다.

F1 HELP

명령 해석기를 사용하는 방법과 기능 키의 의미에 대한 자세한 정보를 제공하는 HELP 패널을 표시합니다.

F2 HEX

(SWITCH HEX/CHAR) 16진수와 문자 형식 사이에 표시장치를 전환합니다. 이는 모드 전환입니다. 모든 후속 화면은 다음에 이 키를 누를 때까지 선택된 모드에서 유지됩니다.

F3 END

(END SESSION) 해독기의 현재 세션을 종료합니다.

F4 EIB

(EIB DISPLAY) EXEC 인터페이스 블록(EIB)의 내용을 보여줍니다.

F5 VAR

(VARIABLES) 현재 명령 해석기 세션과 연관되는 모든 변수를 표시하고, 각 변수의 이름, 길이 및 값을 제공합니다.

F6 USER

(USER DISPLAY) 사용자 표시장치 패널의 현재 내용(즉, 여기까지 처리된 명령이 해독기가 아니라 보통의 프로그램에 의해 실행된 경우 터미널에 나타나는 내용)을 표시합니다. 이 키는 터미널 명령이 실행될 때까지 의미가 없습니다(예: SEND MAP).

F7 SBH

(SCROLL BACK HALF) 본문 절반의 화면을 뒤로 이동합니다.

F8 SFH

(SCROLL FORWARD HALF) 본문 절반의 화면을 앞으로 이동합니다.

F9 MSG

(DISPLAY MESSAGES) 명령의 구문 검사 동안 생성된 모든 메시지를 표시합니다.

F10 SB

(SCROLL BACK) 본문을 하나의 전체 화면만큼 뒤로 이동합니다.

F11 SF

(SCROLL FORWARD) 본문을 하나의 전체 화면만큼 앞으로 이동합니다.

CECI 사용

두 개의 트랜잭션 ID, CECS 또는 CECI와 그 뒤에 테스트하려는 명령의 이름을 입력해서 명령 레벨 해독기를 시작합니다.

명령 옵션도 나열할 수 있습니다(나중에 수행할 수 있음). 예를 들어, 다음과 같습니다.

```
CECS READ FILE('FILEA')
```

또는

```
CECI READ FILE('FILEA')
```

CICS는 명령 및 연관된 함수, 옵션, 인수와 표시로 응답합니다. 명령을 제외할 경우 CECI는 시작하기 위한 가능한 모든 명령 목록을 제공합니다. 참조 -> 애플리케이션 개발에서 CICS 명령 요약 및 *CICS 시스템 프로그래밍 참조서*에 프로그래밍 용도로 설정된 명령을 사용할 수 있습니다.

트랜잭션 코드 CECS를 사용하면 해독기가 명령의 구문이 정확한지 확인합니다. CECI를 사용하는 경우에는 구문이 정확할 때 명령을 실행하는 옵션이 있습니다. CICS는 구문 검사와 실행에 상이한 보안이 지정되도록 두 개의 트랜잭션 ID를 사용합니다.

변경 수행

CICS가 명령을 실행할 때까지 명령행의 내용을 변경해서, 본문의 구문 표시에 표시된 옵션 값을 변경해서 또는 변수 화면에서 변수 값을 변경해서 명령을 변경할 수 있습니다. (명령이 실행된 후에도 여전히 변경할 수 있지만 또 다른 명령에 대비하지 않을 경우 효력이 없습니다.)

명령행이나 변수 화면에서 변경을 수행하면 CECI 트랜잭션의 기간 동안 변경사항이 지속됩니다. 하지만 구문 화면의 본문에서 변경할 경우에는 변경이 일시적입니다. 명령이 실행될 때까지만 지속되며 명령행에 반영되지 않습니다.

모든 문자를 모든 터미널에 표시할 수 있는 것은 아닙니다. 16진 형식이 아닌 문자 형식의 표시인 경우 CECI는 이 문자를 마침표(X'4B')로 표시합니다. 마침표를 겹쳐쓸 때 현재 값은 마침표가 아니라 표시 불가능한 문자일 수 있습니다.

또한 문자 모드 표시일 때에는 문자를 마침표로 변경할 수 없습니다. 이를 시도할 경우 CECI가 변경사항을 무시하고 진단 메시지를 발행하지 않습니다. 이러한 변경을 수행하려면 표시를 16진으로 전환하고 마침표를 나타내는 값(X'4B')을 입력하십시오.

16진 형식의 변경에도 제한사항이 있습니다. 문자를 공백으로 변경해야 하는 경우 16진 표시에서 코드(X'40')를 입력할 수 없습니다. 또 다시 변경사항이 무시되고 CECI는 메시지를 발행하지 않습니다. 대신에 사용자가 문자 모드로 전환한 후 문자를 지워야 합니다.

모든 수정이 끝나면 CECI는 구문을 다시 검사해서 오류가 없는지 확인합니다. 실행 스톱퍼가 있으면 command syntax check에서 처리를 재시작하고 그렇지 않은 경우에는 about to execute command에서 처리를 재시작합니다. 수정하지 않은 about to execute command 화면에서 Enter를 눌러야만 CECI가 명령을 실행합니다.

변수 정의

명령이 옵션의 값 때문에 명령 입력 영역의 행 길이를 초과하거나 옵션 값을 통해 두 명령을 연결하고 싶은 경우에는 변수를 정의할 수 있습니다.

변수 목록을 표시하려면 PF5를 누르십시오. 현재 해독기 세션과 연관된 각 변수에 대해 이름, 길이 및 값이 표시됩니다. CECI는 처음 세 개의 변수를 작성하고 이를 명시적으로 삭제하지 않는 한 이들은 항상 표시됩니다. 이들은 예제를 제공하고 명령 목록의 작성을 돕기 위한 것입니다.

이러한 처음 세 변수 후에 작성한 모든 변수가 표시됩니다. 예를 들어, 다음 명령을 입력할 수도 있습니다.

```
READ FILE('FILEA') RID('009000') INTO(&REC)
```

&REC는 변수로 표시됩니다.

일반적으로 명령행에서 옵션에 제공된 값은 문자열 상수로 사용됩니다. 그러나 이 값을 표시하기 위한 변수를 지정할 수 있습니다. 예를 들어, 옵션 값을 통해 두 명령을 연결하고 싶을 때입니다. 예를 들어, CECI로 레코드를 변경하려면 먼저 다음 명령을 입력해야 합니다.

```
EXEC CICS READ UPDATE INTO(&REC)
      FILE('FILEA') RID('009000')
```

그런 다음 변수 &REC를 변경하여 레코드를 수정한 후 다음을 입력할 수 있습니다.

```
EXEC CICS REWRITE FROM(&REC) FILE('FILEA')
```

첫 번째 위치의 앰퍼샌드는 CECI에게 변수를 지정하는 중임을 알립니다.

필수 값으로 변수를 작성하고 명령에 변수 이름을 지정하여 행 길이 제한사항을 극복할 수 있습니다. 변수는 데이터 유형의 문자, 이중 단어, 전자, 반자 또는 팩형 10진수를 가질 수 있습니다. 다음 방법으로 변수를 작성할 수 있습니다.

- 수신인에서 변수의 이름을 지정합니다. 변수는 명령이 처리될 때 작성됩니다. 이 데이터 유형과 길이는 옵션에 의해 암시됩니다.
- 이미 정의된 변수 목록에 새 항목을 추가하십시오. 새 변수를 작성하려면 변수 표시의 첫 번째 사용하지 않는 행의 적합한 열에 해당 이름과 길이를 입력한 다음 ENTER를 누르십시오. 문자 변수의 경우 변수가 정의될 때 사용된 길이를 사용하십시오. 이중 단어의 경우 **FD**를 입력하십시오. 전자의 경우 유형 **F**를 입력하고, 반자의 경우 유형 **H**를 입력하고, 팩형 변수의 경우 **P**가 앞에 오는 바이트 길이를 사용하십시오.

문자 변수는 공백으로 초기화됩니다. 나머지는 적합한 양식에서 0으로 초기화됩니다. 변수가 작성된 후 변수 표시에서 데이터 필드를 수정하여 값을 변경할 수 있습니다.

- 지정된 옵션의 확장된 영역 표시를 생성했을 때 상태 표시줄에 NAME 필드를 사용하십시오. 이를 수행하려면 구문 표시에서 옵션 아래에 커서를 놓고 ENTER를 누르십시오. 표시된 옵션 값과 연관시키고 싶은 변수 이름을 지정하려면 NAME 필드에 이를 입력하고 ENTER를 다시 누르십시오.
- 기존 변수를 복사하십시오. 이를 수행하려면 복사할 변수의 확장된 영역 표시를 구하고, 새 변수의 이름과 함께 표시된 이름 위에 입력한 다음 ENTER를 누르십시오.
- 구문 표시에 직접 NAME 필드를 사용하십시오. 그러면 250 페이지의 『명령 저장』에 설명된 대로 내용이 명령행의 문자열에 있는 문자 변수가 작성됩니다.

CECI는 세션 종료 시에 모든 변수를 제거하지만 변수를 삭제할 수도 있습니다. 세션 종료 전에 변수를 삭제하려면 이름을 시작하는 앰퍼샌드 아래에 커서를 놓고 ERASE EOF를 누른 다음 ENTER를 누르십시오.

명령 저장

간혹 명령이나 명령 시리즈를 CECI에서 반복적으로 실행할 수 있습니다. 이를 수행하기 위한 한 가지 기술은 명령을 포함하는 임시 저장영역 큐를 작성하는 것입니다. 그런 다음 큐에서 명령을 읽고 실행합니다.

CECI는 큐를 작성하고 큐에서 명령을 실행하기 위한 단축키를 제공합니다. 큐를 작성하려면 다음을 수행하십시오.

1. CECI 세션을 시작하십시오.
2. 명령행에 저장할 첫 번째(또는 다음) 명령을 입력하고, 상태 표시줄의 NAME 필드에 &DFHC를 놓은 후 ENTER를 누르십시오. 이 조치는 일반 구문 확인을 야기시키고, CECI가 항상 사용자 대신 정의하는 세 개의 변수 중 첫 번째 변수인 &DFHC의 값으로서 사용자 명령을 저장합니다. 변수 표시장치를 선택하면, &DFHC가 명령의 값이라는 것을 알게 됩니다.
3. 구문이 올바른지 확인하고 실행하기 전에(**about to execute command** 화면에서), 명령행을 &DFHW로 변경하고 ENTER를 누르십시오. 그러면 CECI가 실행할 명령에 대해 &DFHW의 값을 사용합니다. &DFHW는 CECI 제공 변수 중 두 번째이고, 이름이 『CI<ttt>』인 임시 저장영역 큐에 변수 &DFHC의 내용(즉, 사용자 명령)을 쓰기 위한 명령을 포함합니다. 여기서 『ttt』는 터미널의 이름이고 두 개의 공백이 문자 『CI』 앞에 붙습니다.
4. (**command execution complete** 화면을 통해) 이 WRITEQ 명령을 실행하십시오. 그러면 명령이 큐에 저장됩니다.
5. 두 개 이상의 명령을 저장하려면, 명령마다 4 - 2 단계를 반복하십시오.

목록에서 저장된 명령을 실행하려고 할 때, 다음을 수행하십시오.

1. 명령행에 &DFHR을 입력하고 ENTER를 누르십시오. &DFHR은 CECI 제공 변수 중 마지막이며, 이전에 쓴 큐를 읽기 위한 명령을 포함합니다. 이 명령을 실행하십시오. 그러면 변수 &DFHC에 저장한 명령 중 첫 번째(다음) 명령을 가져옵니다.
2. 그런 다음 명령행에서 &DFHC를 입력하고 ENTER를 누르십시오. CECI는 명령행을 사용자 명령인 &DFHC의 값으로 바꿉니다. 명령을 실행하려면 ENTER를 누르십시오.
3. 저장한 모든 명령을 실행할 때까지, 명령행에서 &DFHR 및 &DFHC를 대체하여, 이러한 두 단계를 반복하십시오.

사용자의 필요성에 적합하도록 이 프로시저를 변경할 수 있습니다. 예를 들어, 단계(2)를 건너뛰어서 순서에서 명령을 건너뛸 수 있습니다. 정상적으로 입력한 명령과 같은 방법으로 실행하기 전에 저장된 명령의 옵션을 변경할 수 있습니다.

저장된 순서의 실행을 반복하려면, 큐의 맨 앞에 읽기를 다시 배치하기 위해 READQ 명령의 첫 번째 실행에서 옵션 ITEM(1)을 지정해야 합니다.

CECI의 실행 방식

해석기는 CICS가 제공하는 프로그램을 사용하여 일반 트랜잭션으로 실행됩니다. 세션 시작과 종료 사이에 할 일은 단일 태스크의 단일 논리적 작업 단위(LUW)입니다.

명령을 실행해서 생성된 잠금과 큐에 넣기는 세션 기간 동안 보존됩니다. 예를 들어, 복구 가능한 파일에서 갱신을 위한 레코드를 읽으면 CECI를 종료할 때까지 다른 태스크는 이 레코드를 사용할 수 없습니다.

이상 종료

CECI는 실행 오류로 인한 일반적인 이상 종료가 발생하지 않도록 모든 명령을 NOHANDLE 옵션을 지정해서 실행합니다.

또한 CECI는 이상 종료가 발생하더라도 제어를 잃지 않도록 세션 초기에 **HANDLE ABEND** 명령을 실행합니다. 결과적으로 CECI가 처리하므로 자원 백아웃은 없습니다. 보호된 자원에 관련된 일련의 갱신을 수행 중인 경우 모든 갱신을 완료할 수 있는지 확인하십시오. 모든 갱신을 완료할 수 없으면 **SYNCPPOINT ROLLBACK** 명령 또는 **ABEND** 명령을 CANCEL 옵션과 함께 사용해서 복구 가능 자원에 대한 이전 명령의 효과를 제거하십시오.

예외 조건

일부 명령의 경우 지정된 모든 옵션이 올바른 경우에도 CECI가 예외 조건을 리턴할 때가 있습니다. 이 조건은 일부 명령에서 CECI가 명시적으로 지정하지 않은 옵션을 사용하기 때문에 리턴되는 것입니다. 예를 들어, **ASSIGN** 명령은 CECI에서 항상 예외 조건 INVREQ를 리턴합니다. CECI는 사용자가 제대로 요청한 정보를 리턴하더라도 일부가 올바르지 않은 다른 옵션에서 정보를 얻으려 시도합니다.

프로그램 제어 명령

해독기는 자체가 애플리케이션이기 때문에 일부 프로그램 제어 명령을 해독할 때 해당 명령을 실행하는 애플리케이션과 다른 결과를 생성할 수 있습니다. 예를 들어, CANCEL 옵션을 사용하지 않으면 **ABEND** 명령이 인터셉트됩니다.

LINK 명령을 사용하는 경우 대상 프로그램은 해독기의 환경에서 실행합니다. 특히, 링크된 프로그램 중에 사용자 표시를 수정할 경우 해독기는 변경사항을 인식하지 못합니다.

이와 마찬가지로, **XCTL** 명령을 해독하는 경우 CECI는 이름 지정된 프로그램에 제어를 전달하고 제어를 다시 가져오지 않으므로 CECI 세션이 종료됩니다.

터미널 공유

해독 중인 명령이 해독기와 동일한 화면을 사용하는 명령인 경우 명령 해독기는 해독기 표시장치와 사용자 표시장치 간에 화면 공유를 관리합니다.

사용자 표시장치는 다음의 경우에 복원됩니다.

- 처리 중인 명령에 운영자의 데이터가 필요할 때
- 처리 중인 명령이 사용자 표시장치를 수정하려 할 때
- USER DISPLAY가 요청될 때

SEND 명령 다음에 RECEIVE 명령이 지정되면 SEND 명령이 전송한 표시가 두 번 표시됩니다. SEND 명령이 처리될 때 한 번 표시되고 RECEIVE 명령이 처리될 때 다시 표시됩니다. SEND 명령에 응답할 필요가 없지만 응답하는 경우에는 RECEIVE 명령에 대한 화면이 복원될 때 해독기가 이를 저장하고 표시합니다.

해독기는 사용자 표시를 복원할 때 SEND 명령이 처리될 때와 동일한 방식으로 알람 경고음을 생성하지 않거나 키보드에 영향을 미치지 않습니다.

공유 저장영역: LENGTH 옵션 없는 ENQ 명령

일반적으로 LENGTH 옵션 없이 EXEC CICS ENQ 명령을 사용하는 경우 그 효과는 저장영역에 특정 위치(주소)의 데이터 영역을 자원으로 지정하는 것입니다. 다중 태스크를 통해 이 자원에서 큐에 넣을 수 있으며 저장영역의 동일한 위치를 참조해야 합니다. CECI는 공유 저장영역이 아닌 자체 작업 중인 저장영역을 사용하기 때문에 이 작동을 에뮬레이트할 수 없습니다.

ENQ 명령을 CECI에서 LENGTH 옵션 없이 실행하는 경우 CICS는 CECI 태스크가 소유한 저장영역 내의 주소에서 큐에 넣습니다. 다른 태스크(CECI이든 아니든)는 동일한 이 저장영역에서 큐에 넣을 수 없습니다. CECI는 변수에 대한 공유 저장영역 사용을 지원하지 않습니다.

저장영역 주소를 RESOURCE 옵션으로 지정하고 LENGTH 옵션을 추가(ENQ 명령이 CECI 태스크에서 실행될 때)한 후, 또 다른 CECI나 비CECI 태스크에 LENGTH 옵션 없이 동일한 저장영역 주소를 지정해서 의도한 작동을 에뮬레이트할 수 없습니다. LENGTH 옵션이 지정되면 CICS는 자신의 위치가 아닌 자원 값에서 큐에 넣습니다. 따라서 CICS는 LENGTH 옵션이 있거나 없는 큐에 넣기를 상이한 큐에 넣기로 간주하므로 태스크가 의도한 대로 직렬화되지 않습니다.

다중 태스크에서 실행한 동일한 ENQ 명령에 LENGTH 옵션이 지정되면 LENGTH 옵션이 지정될 때 데이터 영역의 위치(CCI가 소유한 저장영역 또는 다른 저장영역이든 상관없이)가 문제가 되지 않으므로 큐에 넣기가 예상대로 작동합니다.

제 15 장 CICS 애플리케이션에서 디버거 사용 준비

CICS는 버그 분리 및 수정과 애플리케이션 테스트를 위한 워크스테이션 기반 및 호스트 기반 디버거의 사용을 지원합니다. CICS 애플리케이션에서 디버거를 사용하려면 먼저 다음 작업을 수행해야 합니다.

시작하기 전에

이 작업 정보

프로시저

1. 워크스테이션 기반과 호스트 기반 디버거 사이에서 선택하십시오. 애플리케이션을 디버깅할 때 디버깅 도구를 통해 프로그램과 상호작용합니다. 예를 들어, 저장영역을 검사하거나 중단점을 설정하거나 코드를 스텝 스루하려고 합니다. 이 상호작용은 디버깅 세션입니다. CICS에서 디버깅 세션을 실행할 환경을 선택할 수 있습니다.

워크스테이션 기반

워크스테이션의 디버거 클라이언트는 디버깅 작업을 수행하는 데 사용하는 그래픽 사용자 인터페이스를 제공합니다. 디버거 클라이언트는 CICS 시스템에서 실행되고 디버깅 중인 프로그램과 상호작용하는 디버거 서버와 통신합니다.

자세한 정보는 261 페이지의 제 16 장 『워크스테이션에서 CICS 애플리케이션 디버깅』의 내용을 참조하십시오.

호스트 기반

CICS 시스템에서 실행되고 있는 디버깅 도구는 디버깅 작업을 수행하는 데 사용하는 터미널 인터페이스를 제공합니다. 디버깅 도구는 애플리케이션이 실행될 때 이 애플리케이션과 직접 상호작용합니다.

CICS는 호스트 기반 디버깅을 위한 디버깅 도구를 지원합니다. 자세한 정보는 263 페이지의 제 17 장 『CICS 애플리케이션에서 디버깅 도구 사용』의 내용을 참조하십시오.

애플리케이션에 따라 다른 디버깅 요구사항이 있을 수 있습니다(예를 들어, Java 프로그램은 호스트 기반 디버깅 세션에서 디버깅할 수 없음). CICS에서 다른 사용자가 워크스테이션 기반 및 호스트 기반 디버깅을 동시에 같은 리전에서 사용할 수 있습니다.

2. 애플리케이션이 디버깅 도구에 의해 인터셉트되고 그 외에는 인터셉트되지 않는지 확인하십시오. 테스트 또는 배치 시스템에서도 대부분의 애플리케이션은 대부분의 시간 동안 올바르게 작동합니다. 그리고 디버깅 중일 때 한 번에 한 애플리케이션에

만 집중하려고 할 수 있습니다. 동시에 동료는 다른 애플리케이션을 디버깅하려고 할 수 있습니다. 따라서 CICS 시스템에서 대부분의 프로그램은 시스템에서 정상적으로 실행되도록 두고 디버깅 세션과 상호작용할 프로그램 및 다른 사용자의 디버깅 세션과 상호작용할 프로그램을 지정할 수 있습니다.

디버깅 프로파일을 통해 이러한 모든 작업을 수행할 수 있습니다. 디버깅 프로파일은 함께 디버깅할 일련의 애플리케이션을 지정합니다. 프로파일을 활성화하면 프로파일이 정의하는 프로그램은 사용자가 지정한 디버깅 세션을 사용하여 디버거의 제어 하에 실행됩니다. 프로파일을 비활성화하면 디버깅 프로파일에서 참조되지 않는 프로그램이 수행하는 대로 프로그램이 다시 정상적으로 실행됩니다. 디버깅 프로파일을 사용하면 디버깅 세션의 특성을 정의하여 특정 프로그램의 디버깅에 사용할 수도 있습니다.

자세한 정보는 255 페이지의 『디버깅 프로파일』의 내용을 참조하십시오.

3. 디버거와 상호작용하려면 프로그램을 준비하십시오. CICS는 다양한 언어로 작성된 애플리케이션을 지원합니다. 컴파일 언어 프로그램(COBOL, PL/I, C, C++ 및 Language Environment 사용 어셈블러 서브루틴)은 Language Environment의 제어 하에 실행되고 Java 프로그램은 JVM(Java Virtual Machine)에서 실행됩니다. 프로그램에 대해 기본적으로 두 개의 서로 다른 런타임 환경이 있기 때문에 프로그램이 디버거와 상호작용하게 하는 두 가지 방법이 있습니다.
 - 컴파일 언어 프로그램의 경우 디버거와 상호작용하게 할 프로그램을 컴파일하는 시간을 결정하고 적절한 컴파일러 옵션을 지정해야 합니다. 자세한 정보는 컴파일러 문서를 참조하십시오.
 - Java 프로그램의 경우 런타임 시 디버거와 상호작용하게 할 프로그램을 결정하고 적절한 JVM 옵션을 지정할 수 있습니다. 자세한 정보는 CICS의 Java 애플리케이션의 내용을 참조하십시오.
4. CICS 시스템이 디버깅 환경을 지원하는지 확인하십시오. CICS 시스템에 디버깅 프로파일이 있는 경우 모든 프로파일이 비활성인 경우에도 프로그램 시작 시 오버헤드가 발생합니다. 이 오버헤드가 작더라도 고성능 프로덕션 시스템에서는 거의 허용되지 않습니다. 그러나 사용자는 보통 그러한 시스템에서는 애플리케이션을 디버깅하지 않습니다. 따라서 디버깅 프로파일의 사용은 선택적이고 디버깅 프로파일을 사용하려는 경우 시스템 프로그래머는 그에 따라 CICS를 구성해야 합니다.

예

다음에 수행할 작업

디버깅 프로파일

디버깅 프로파일은 함께 디버깅할 하나 이상의 일련의 애플리케이션을 지정합니다.

예를 들어, 다음과 같습니다.

- 시스템 CICS1에서 실행되는 프로그램 PYRL01의 모든 인스턴스
 - 이름이 『setBankAccount』로 시작되는 모든 Java 클래스
 - 사용자 APPDEV02에 의해 실행되고 이름이 『PYRL』로 시작되는 모든 프로그램
- CICS는 디버깅 프로파일에서 다음 정보를 사용하여 프로그램의 인스턴스가 디버거의 제어 하에 실행되어야 하는지 결정합니다. 매개변수는 다음을 지정합니다.
- 프로그램이 실행되는 트랜잭션
 - 트랜잭션과 연관된 터미널. 터미널 ID 또는 z/OS Communications Server 네트워크 이름을 지정할 수 있습니다.
 - 프로그램의 이름
 - COBOL 프로그램의 경우 장치 또는 컴파일의 이름(프로그램 또는 클래스 이름)
 - Java 오브젝트의 경우 클래스 이름
 - 로그인한 사용자의 사용자 ID
 - 트랜잭션이 실행되는 CICS 리전의 applid

대부분의 매개변수는 같은 문자로 시작되는 일련의 값을 지정할 수 있는 일반 매개변수일 수 있습니다(예: TRN0, TRN1, TRN2, TRNA, TRNB, ...).

디버깅 프로파일은 다음과 같은 추가 정보를 포함합니다.

상태 프로파일의 상태: 활성 또는 비활성:

- 프로파일이 활성인 경우 디버깅이 필요한 리전에서 프로그램이 시작될 때마다 프로파일이 검사됩니다.

참고: 프로파일이 활성인 동안 프로파일을 변경하면 변경사항이 즉시 적용됩니다. 다음에 프로그램을 시작할 때에는 변경된 매개변수가 프로그램이 디버거의 제어에서 실행되어야 하는지 여부를 결정합니다.

- 프로파일이 비활성인 경우 프로그램이 시작되면 변경사항을 무시합니다.

디버깅 표시장치 설정

디버깅 표시장치 설정은 디버거와 상호작용하는 방식을 지정합니다.

- Java 프로그램의 경우 워크스테이션에서 디버깅 도구를 사용할 수 있습니다.
- 컴파일 언어 프로그램에서 다음을 사용할 수 있습니다.

A3270 터미널

워크스테이션에서 디버깅 도구

JVM 프로파일 이름

Java 프로그램에서만 프로그램이 디버깅될 때 사용되는 JVM 프로파일을 지정할 수 있습니다.

디버그 도구 및 Language Environment 옵션

컴파일 언어 프로그램에서만 프로그램이 디버깅될 때 디버그 도구와 Language Environment에 전달될 옵션을 지정할 수 있습니다.

다음과 같은 종류의 프로그램에 대한 디버깅 프로파일을 작성할 수 있습니다.

컴파일 언어 프로그램

Java 애플리케이션

프로파일에 저장되는 정보는 각 유형의 프로그램에 따라 다릅니다.

프로파일은 둘 이상의 CICS 리전이 공유할 수 있는 CICS 파일에 저장됩니다. 여러 CICS 리전에서 공유하는 프로파일은 모든 리전에서 활성이거나 비활성입니다. 이러한 프로파일은 일부 리전에서는 활성이고 다른 리전에서는 비활성일 수 있습니다.

CICS는 시스템이 디버깅 프로파일을 사용하도록 설정되면 선택적으로 생성되는 일련의 샘플 프로그램을 제공합니다. 자체 프로파일을 작성하기 위한 시작점으로 이러한 프로파일을 사용할 수 있습니다.

디버깅할 프로그램을 선택하기 위해 디버깅 프로파일 사용

디버깅할 프로그램을 선택하려면 한 개 이상의 디버깅 프로파일을 작성해야 합니다. 각 프로파일은 CICS가 프로그램의 인스턴스가 디버거의 제어로 실행되는지 결정하는 데 사용하는 다수의 매개변수를 지정합니다.

프로파일은 활성이거나 비활성일 수 있습니다. 활성 프로파일 중 하나가 프로그램 인스턴스와 일치하는 경우 프로그램은 디버거의 제어로 실행됩니다. CICS가 프로그램을 시작할 때 비활성 프로파일은 검사하지 않습니다. 프로파일은 작성 시 비활성입니다.

표 14는 디버깅 프로파일의 매개변수가 컴파일 언어 프로그램의 프로그램 인스턴스를 선택하는 데 사용되는 방식을 보여주는 예입니다. 257 페이지의 표 15에서는 디버깅 프로파일의 매개변수가 Java 프로파일을 선택하는 데 사용되는 방식을 보여줍니다.

표 14. 컴파일 언어 프로그램의 디버깅 프로파일 매개변수 예

디버깅 프로파일	트랜잭션	터미널	프로그램	사용자	애플리케이션 ID
프로파일 1	PRLA	T001	PYRL01	TESTER5	CICSTST2
프로파일 2	PRLA	*	PYRL02	*	*
프로파일 3	PRL*	*	*	*	CICSTST3

표 15. Java 프로그램의 디버깅 프로파일 예제

디버깅 프로파일	트랜잭션	Bean	메소드	사용자	애플리케이션 ID
프로파일 4	PRLA	NewEmployee	setBasicSalary	TESTER5	CICSTST2

다음은 디버거의 제어로 실행되는 프로그램을 각 프로파일이 제어하는 방식입니다.

프로파일 1

이 예제에서 테이블의 모든 매개변수는 명시적으로 지정됩니다. 프로그램 PYRL01은 다음의 모든 조건이 충족되는 경우에만 디버거의 제어로 실행됩니다.

- 트랜잭션이 PRLA임
- 트랜잭션이 터미널 T001에서의 터미널 입력에 의해 시작됨
- 트랜잭션이 사용자 TESTER5에 의해 실행되고 있음
- 트랜잭션이 리전 CICSTST2에서 실행되고 있음

프로파일 2

이 예제에서 테이블의 일부 매개변수는 *로 지정되는 일반 매개변수입니다. 이 유형의 일반 매개변수는 모든 값과 일치합니다. 이 프로파일은 트랜잭션 PRLA에서 실행되는 프로그램 PYRL02의 각 인스턴스가 디버거의 제어를 받도록 지정합니다.

프로파일 3

이 예제에는 다른 종류의 일반 매개변수가 포함됩니다. PRL*는 문자 'PRL'로 시작하는 모든 값과 일치합니다. 이 프로파일은 리전 CICSTST3에서 ID가 문자 'PRL'로 시작되는 트랜잭션에서 실행되는 모든 프로그램이 디버거의 제어를 받도록 지정합니다.

프로파일 4

메소드 setBasicSalary는 다음의 모든 조건이 충족되는 경우에만 디버거의 제어로 실행됩니다.

- 트랜잭션이 PRLA임
- 메소드는 bean NewEmployee 메소드임
- 트랜잭션이 사용자 TESTER5에 의해 실행되고 있음
- 트랜잭션이 리전 CICSTST2에서 실행되고 있음

디버거의 제어를 받는 프로그램이 갑자기 시작되지 않도록 하기 위해 디버깅 프로파일에서 지정한 매개변수를 주의하여 선택해야 합니다.

- 가능한 경우 매개변수 모두 또는 대부분의 값을 지정하여 특정 환경의 특정 프로그램에 대한 디버깅을 제한하십시오. 가능한 경우 일반 값보다는 특정 값을 사용하십시오.

- 가능한 경우 각 디버깅 프로파일에서 사용자 ID 및 애플리케이션 ID를 명시적으로 지정하십시오.
- 프로덕션 리전에서 프로그램을 디버깅하는 것은 권장되지 않지만 필요한 경우가 있을 수 있습니다. 이러한 경우에는 모든 매개변수가 명시적으로 지정되는 디버깅 프로파일을 사용하십시오.
- 사용해야 하는 경우에만 디버깅 프로파일을 활성화하고 사용 후에는 즉시 비활성화하십시오.

디버깅 프로파일에서 일반 매개변수 사용

디버깅 프로파일에서 다수의 매개변수에 대한 일반 값을 제공할 수 있습니다. 일반 매개변수를 지정하려면 와일드카드 문자로 별표(*)를 사용하십시오. 와일드카드 문자만 사용하거나 와일드카드 문자를 매개변수 끝에 사용할 수 있습니다. 매개변수를 공백으로 두는 것은 별표를 지정하는 것과 같습니다.

이 태스크 정보

예를 들어, 다음과 같습니다.

*는 모든 가능한 값과 일치함

TR*는 TR, TRA, TRAA 및 TRAQ와 일치함

TRA*는 TRA, TRAA 및 TRAQ와 일치하지만 TR과는 일치하지 않음

와일드카드를 사용하면 시작 프로그램이 둘 이상의 활성 프로그램과 일치할 수 있습니다. 이 경우 CICS는 다음과 같은 원칙을 사용하여 가장 일치하는 프로파일을 선택합니다.

- 모든 매개변수는 정확하게 일치하거나 와일드카드를 고려하여 일치해야 합니다.
- 최상의 일치는 와일드카드를 포함하지 않는 프로파일입니다.
- 차선의 일치는 *를 포함하는 프로파일입니다. 이 그룹 내에서 최상의 일치는 가장 적은 수의 *를 포함하고 가장 많은 수의 명시적으로 지정된 문자를 포함하는 프로파일입니다.

예를 들어, 트랜잭션 TRAA를 고려할 때

- TRAA는 가능한 최상의 일치임(모든 문자 일치)
- TRA*는 TR*보다 정확한 일치임

디버깅 프로파일에서 복잡한 와일드 카드의 사용을 피하는 것이 좋습니다. 많은 프로파일 중 지정된 프로그램 인스턴스에 대해 최상의 일치가 되는 프로파일이 항상 명백한 것은 아니기 때문입니다. 그러나 복잡한 와일드카드 사용이 필요한 경우 여러 프로파일 중 최상 일치에 될 프로파일을 정확히 확인하기 위해 259 페이지의 그림 58의 정보를 사용할 수 있습니다.

각 필드에 대해 차례대로 다음을 수행하십시오.

1. 각 필드의 문자 수를 셉니다(후미 공백은 포함하고 *는 제외함)(C).
2. * 문자 수를 셉니다(A).
3. 필드의 길이를 판별합니다(L).
4. M 을 $C - (L * A)$ 로 계산합니다. M 이 음수일 수 있음에 유의하십시오.

각 프로파일에 대해 차례대로 모든 필드의 M 값을 합합니다(R).

R 값이 가장 큰 프로파일이 최상의 일치입니다. 둘 이상의 일치 프로파일이 동일한 가장 큰 R 값을 가지면 CICS는 프로파일이 작성된 순서 선택에 따라 그 중 하나를 선택합니다.

그림 58. 디버깅 프로파일 일치 알고리즘

제 16 장 워크스테이션에서 CICS 애플리케이션 디버깅

워크스테이션에서 실행되는 디버깅 도구를 사용하여 CICS 애플리케이션을 디버깅할 수 있습니다.

이 환경에는 디버깅 도구에 대한 두 가지 구성요소가 있습니다.

- **디버거 클라이언트**, 워크스테이션에서 실행됩니다. 애플리케이션과의 상호작용은 디버거 클라이언트에서 제공되는 그래픽 사용자 인터페이스(GUI)를 통해 이루어집니다. 예를 들어, 디버거 클라이언트를 사용하여 중단점을 설정하고 프로그램을 스텝 스루 하며 프로그램에서 사용되는 변수를 검사할 수 있습니다.
- **디버거 서버**, 애플리케이션과 같은 시스템에서 실행되고 디버거 클라이언트와 통신합니다.

워크스테이션에서 디버거 클라이언트를 사용하여 다음과 같은 종류의 CICS 애플리케이션을 디버깅할 수 있습니다.

- 컴파일 언어(COBOL, PL/I, C, C++)로 작성된 애플리케이션
- Language Environment가 사용 설정된 어셈블러 서브루틴
- JVM에서 실행 중인 Java 애플리케이션
- 컴파일 언어 프로그램 및 Java 프로그램의 조합을 사용하는 애플리케이션

워크스테이션에서 디버거 클라이언트를 사용하여 PLT 프로그램을 디버깅할 수 없습니다.

다음은 디버거 클라이언트로 사용할 수 있습니다.

WebSphere® Studio Enterprise Developer
WebSphere Studio Application Developer

컴파일 언어와 Language Environment가 사용 설정된 어셈블러 서브루틴의 경우 다음 제품을 디버거 서버로 사용할 수 있습니다.

- 디버그 도구

Java 프로그램에서는 디버거 서버는 디버그 모드로 실행되는 JVM(Java Virtual Machine)입니다.

워크스테이션에서 애플리케이션 디버그 준비

워크스테이션을 사용하여 CICS 애플리케이션을 디버깅하려면 먼저 시스템 프로그래머가 디버깅을 위해 CICS 리전을 준비해야 합니다.

이 태스크 정보

다음 태스크를 완료해야 합니다.

프로시저

1. 워크스테이션에 적합한 디버거 클라이언트를 설치하십시오. 다음 제품을 디버거 클라이언트로 사용할 수 있습니다.

WebSphere Studio Enterprise Developer

WebSphere Studio Application Developer

이러한 제품 문서에는 제품 설치 및 사용에 필요한 정보가 들어 있습니다.

2. 한 개 이상의 디버깅 프로파일을 작성하십시오. 디버깅 프로파일은 디버거의 제어를 받아 실행되는 프로그램을 지정합니다.

참고: 디버깅 프로파일은 JVM 프로파일과 같지 않습니다. Java 애플리케이션을 디버깅하려면 두 프로파일 모두 필요합니다.

3. COBOL, PL/I, C 또는 C++로 작성되는 프로그램 또는 Language Environment가 사용 설정된 어셈블러 서브루틴을 디버깅하려는 경우 디버그 세션을 실행하고 적절한 옵션으로 프로그램을 컴파일할 방식을 고려하십시오. 자세한 정보는 *z/OS용 디버그 도구 사용자 안내서*를 참조하십시오.
4. Java 프로그램을 디버깅하려면 프로그램이 디버깅에 사용 가능한 JVM(Java Virtual Machine)에서 실행되는지 확인해야 합니다. 이를 수행하려면 다음과 같이 하십시오.
 - a. JVM을 디버깅에 사용 가능하게 하는 JVM 프로파일을 작성하십시오. 자세한 정보는 *CICS의 Java 애플리케이션에서 CICS JVM에서 실행 중인 애플리케이션 디버깅*을 참조하십시오.
 - b. Java 프로그램에 대한 디버깅 프로파일을 작성할 때 JVM 프로파일을 지정하십시오. JVM 프로파일을 지정하지 않으면 JVM은 PROGRAM 정의에 지정된 프로파일을 사용합니다.
5. 워크스테이션에서 디버거 클라이언트를 시작하십시오.
6. 디버거로 WebSphere Studio를 사용하는 경우 프로그램에 최소 한 개 이상의 중단점을 설정하십시오.
7. 디버깅할 프로그램 인스턴스를 정의하는 디버깅 프로파일을 활성화하십시오. 컴파일 언어 프로그램에서 프로파일을 활성화할 때 프로그램이 실행되면 시작되는 디버깅 세션의 속성을 지정하는 디버깅 옵션을 정의해야 합니다.

결과

이러한 단계를 모두 완료하면 최종 단계에서 선택한 프로그램이 디버거의 제어를 받아 실행됩니다.

제 17 장 CICS 애플리케이션에서 디버그 도구 사용

디버그 도구는 프로그램을 테스트하고 애플리케이션의 실행을 시험, 모니터 및 제어하는 데 도움을 줍니다.

디버그 도구에 대한 자세한 정보는 *z/OS용 디버그 도구 사용자 안내서*를 참조하십시오.

디버그 도구 정보

디버그 도구는 프로그램을 테스트하고 CICS 애플리케이션의 실행을 검사하고, 모니터 하고 제어하는 데 도움이 됩니다.

디버그 도구를 사용하여 다음과 같은 종류의 CICS 애플리케이션을 디버깅할 수 있습니다.

- 컴파일 언어(COBOL, PL/I, C, C++)로 작성된 애플리케이션
- Language Environment가 사용 설정된 어셈블러 서브루틴
- 컴파일 언어 프로그램 및 Java 프로그램의 조합을 사용하는 애플리케이션. 디버그 도구는 이러한 애플리케이션의 Java 부분은 디버깅하지 않습니다.

디버그 도구를 사용하여 PLT 프로그램을 디버깅할 수 없습니다.

다음과 같은 네 가지 방식으로 디버그 도구를 사용할 수 있습니다.

단일 터미널 모드

디버그 도구가 애플리케이션과 같은 터미널에 화면을 표시합니다.

이중 터미널 모드

디버그 도구가 애플리케이션에서 사용된 것과 다른 터미널에서 화면을 표시합니다.

일괄처리 모드

디버그 도구에 터미널이 없지만 입력에 명령 파일을 사용하고 로그에 출력을 씁니다.

원격 디버그 모드

디버그 도구가 디버거 클라이언트와 함께 작업하여 워크스테이션에 그 결과를 표시합니다.

디버그 도구에 대한 자세한 정보는 *z/OS용 디버그 도구 사용자 안내서*를 참조하십시오.

참고: 단일 터미널 모드 또는 이중 터미널 모드에서 디버그 도구를 사용하는 경우 디버그 도구에서 사용되는 터미널은 애플리케이션이 실행되고 있는 리전에서 로컬 터미널이어야 합니다. 터미널 소유 리전에서 터미널을 사용하여 애플리케이션 소유 리전의 디버그 도구와 상호작용할 수 없습니다.

디버그 도구를 사용한 애플리케이션 디버그 준비

디버그 도구를 사용하여 CICS 애플리케이션을 디버깅하려면 먼저 시스템 프로그래머가 디버깅을 위해 CICS 리전을 준비해야 합니다.

시작하기 전에

자세한 정보는 *CICS 애플리케이션 프로그래밍 안내서*의 내용을 참조하십시오.

이 태스크 정보

그러면 다음 태스크를 완료해야 합니다.

프로시저

1. 디버그 세션을 실행하고 적절한 옵션으로 프로그램을 컴파일할 방식을 고려하십시오. 자세한 정보는 *z/OS용 디버그 도구 사용자 안내서*를 참조하십시오.
2. 한 개 이상의 디버깅 프로파일을 작성하십시오. 디버깅 프로파일은 디버거의 제어를 받아 실행되는 프로그램을 지정합니다.
3. 디버깅할 프로그램 인스턴스를 정의하는 디버깅 프로파일을 활성화하십시오. 프로파일을 활성화할 때 디버거와 상호작용하는 데 사용할 표시장치를 지정해야 합니다.

결과

이러한 단계를 모두 완료하면 최종 단계에서 선택한 프로그램이 디버그 도구의 제어를 받아 실행됩니다.

제 4 부 CICS 애플리케이션 프로그래밍 기술

CICS용 프로그램을 작성하기 위해 알아야 하는 개념 및 기술입니다.

제 18 장 애플리케이션 디자인

다음 기본 개념을 사용하면 CICS 애플리케이션 디자인에 도움이 됩니다.

성능 및 효율을 향상시킬 수 있는 변경사항이 제안되었습니다. 그러나 효율성에 대한 프로그래밍 추가 안내는 303 페이지의 『성능 향상을 위한 디자인』에서 제공됩니다.

CICS에서 구현되는 프로그래밍 모델은 3270s를 위해 디자인된 모델에서 상속되고 대화식, 터미널 지향 애플리케이션의 다양한 특성을 보여줍니다. 일반적으로 다음과 같이 세 가지 유형의 프로그래밍 모델이 있습니다.

- 터미널 시작(대화식 모델)
- DPL(Distributed Program Link) 또는 RPC 모델
- START(큐 처리 모델)

일단 시작되면 애플리케이션은 일반적으로 이러한 모델 및 자체적으로 계속하고 분배하는 기타 메소드를 사용합니다. 예를 들어, 의사 대화에서는 RETURN IMMEDIATE 또는 DTP입니다. 이러한 모델 사이의 주요 차이점은 모델이 상태를 유지하는 방식(예: 보안)에 있기 때문에 상태는 애플리케이션 디자인의 필수적인 파트가 됩니다. 이는 다른 애플리케이션으로 변환을 시도할 때 가장 심각한 문제점을 나타냅니다.

의사 대화식 모델은 대부분 터미널 시작 트랜잭션과 연관되고 대화식 모델의 효율적인 구현으로 개발되었습니다. 1-in 및 1-out 프로토콜(예: HTTP)의 사용 증가로 인해 DPL 또는 RPC 모델에 대한 의사 대화식 특성 추가가 필요하게 되었습니다.

의사 대화식 및 대화식 디자인

대화식 트랜잭션에서 각 사용자의 응답을 처리하는 데 소요된 시간의 길이는 입력 대기에 걸린 시간에 비해 매우 짧습니다. 대화식 트랜잭션은 트랜잭션과 사용자가 대화를 시작할 수 있도록 터미널로부터 둘 이상의 입력을 포함하는 트랜잭션입니다. 비대화식 트랜잭션에는 한 개의 입력만 있습니다(트랜잭션이 호출되는 원인이 되는 입력). 이 트랜잭션은 해당 입력을 처리한 후 터미널에 대해 응답하고 종료합니다.

프로세서 속도(데이터 세트 액세스도 고려된 속도)는 사용자 응답 시간보다 훨씬 더 빠른 터미널 전송 시간에 비해 상당히 빠릅니다. 이는 특히 사용자가 입력에 대해 생각해야 하는 경우 또는 많은 문자를 입력해야 하는 경우에 해당됩니다. 결과적으로 대화식 트랜잭션은 비대화식 트랜잭션에 비해 훨씬 더 길게 저장영역 및 기타 자원을 사용합니다.

의사 대화식 트랜잭션 순서에는 사용자를 여러 입력 화면이 포함된 단일 대화식 트랜잭션으로 간주하는 일련의 비대화식 트랜잭션이 포함됩니다. 순서에서 각 트랜잭션은 한 개의 입력을 처리한 후 응답을 다시 전송하고 종료합니다.

의사 대화식 트랜잭션은 종료하기 전에 같은 터미널에서 시작된 다음 트랜잭션에서 사용할 데이터를 해당 트랜잭션이 도착할 때마다 앞으로 전달할 수 있습니다. 의사 대화식 트랜잭션은 RETURN 명령에서 TRANSID 옵션을 사용하여 다음 트랜잭션이 될 트랜잭션을 지정할 수 있습니다. 그러나 해당 디바이스에 대한 다른 트랜잭션이 시작되면 RETURN 명령에 IMMEDIATE 옵션을 지정하지 않을 경우 사용자가 디자인한 의사 대화식 체인을 인터럽트할 수 있습니다. 이 경우 이 터미널에 대해 큐 처리된 다른 트랜잭션에 관계없이 TRANSID 명령에서 지정된 트랜잭션이 첨부됩니다.

응답이 작성되는 시간부터 사용자가 다음 입력을 전송하고 CICS가 그에 응답하기 위해 다음 트랜잭션을 시작할 때까지 터미널에 대한 트랜잭션은 존재하지 않습니다. 일반적으로 입력 간 프로그램에 저장되는 정보는 순서의 한 트랜잭션에서 COMMAREA를 사용하는 다음 트랜잭션 또는 CICS 제공자가 이러한 용도로 제공하는 기타 기능 중 하나로 전달됩니다. (세부사항은 314 페이지의 『트랜잭션 사이에 데이터 공유』의 내용을 참조하십시오.)

대화식 프로그래밍과 의사 대화식 프로그래밍 사이에서 선택할 때 고려해야 할 두 가지 주요 이슈가 있습니다.

- **경합 자원**(예: 저장영역 및 프로세서 사용)에 대한 트랜잭션의 영향. 트랜잭션을 구성하는 제어 블록, 데이터 영역 및 프로그램에 대한 저장영역이 필요하고 태스크를 시작하고 처리하고 종료하기 위한 프로세서가 필요합니다. 대화식 프로그램은 동등한 유사 대화식 순서를 구성하는 트랜잭션의 합계에 비해 매우 길게 지속되기 때문에 저장영역에 매우 큰 영향을 미칩니다. 그러나 프로세서 오버헤드는 오히려 적습니다. 모든 입력에 대해 하나씩이 아닌 한 개의 트랜잭션만 시작되기 때문입니다.
- **독점 사용 자원**(예: 복구 가능한 데이터 세트의 레코드, 복구 가능한 트랜지언트 데이터 큐, 큐에 넣기 항목 등)에 대한 영향. 다시, 대화식 트랜잭션은 이러한 자원을 해당 순서의 비대화식 트랜잭션에 비해 훨씬 더 오래 보존합니다. 이러한 관점에서 의사 대화식 트랜잭션은 빠른 응답에는 더 적합하지만 복구 및 무결성 함축은 대화식 트랜잭션 사용을 선호함을 의미할 수 있습니다.

요약하면, 대화식 태스크는 작성은 더 쉽지만 성능(가상 저장영역의 필요성) 및 이 태스크를 포함하는 CICS 시스템의 전반적인 운용성에 대한 영향 측면 둘 다에서 심각한 단점이 있습니다. 지금은 프로세서가 과거에 비해 늘어난 실제 저장영역과 더 강력한 파워로 더 커졌기 때문에 작은 양에 대해서는 대화식 태스크도 더 이상 어렵지 않습니다. 다만 대화식 애플리케이션을 사용할 경우 작업 초기에 가상 저장영역 제한조건이 발생할 수 있습니다. 보통 수준 이상으로 애플리케이션을 실행하면 가상 저장영역 제한조건이 발생하기 전에 ENQ 문제점이 발생할 수 있습니다.

CICS는 작업 단위(UOW)에 의한 복구 가능 자원(예: 데이터 세트, 트랜지언트 데이터 및 임시 저장영역)에 대한 변경이 완전하게 이루어지는지 또는 전혀 이루어지지 않는지 확인합니다. UOW는 트랜잭션이 SYNCPOINT 명령을 실행하지 않는 경우 이 트랜잭션과 동등하며 이 경우 UOW는 두 동기점 사이에서 유지됩니다. 동기점과 UOW에 대한 자세한 설명은 *CICS 복구 및 다시 시작 안내서*에서 기능 복구 및 재시작을 참조하십시오.

트랜잭션이 복구 가능 자원을 변경하면 CICS는 원래 트랜잭션이 완료될 때까지 해당 자원을 변경하려는 다른 트랜잭션에 대해 이 자원을 사용 불가능한 상태가 됩니다. 대화식 트랜잭션의 경우 해당 자원은 상대적으로 긴 기간 동안 다른 터미널에 대해 사용 불가능한 상태일 수 있습니다.

예를 들어, 한 사용자가 복구 가능 데이터 세트의 특정 레코드를 갱신하려고 하고 이 작업이 끝나기 전에 다른 사용자도 같은 작업을 시도하려고 하면 두 번째 사용자의 트랜잭션은 일시중단됩니다. 여기에는 장단점이 있습니다. 첫 번째 사용자가 레코드를 변경하는 동안 두 번째 사용자가 레코드 갱신을 시작하는 것을 원하지 않을 수 있습니다. 둘 중 한 명은 곧 더 이상 사용하지 않는 버전이 될 레코드에서 작업하게 되고 이러한 변경이 다른 사용자의 변경사항을 지울 수 있기 때문입니다. 다른 한편으로는 변경 중인 레코드를 이 트랜잭션이 갱신을 위해 읽으려고 할 때 발생하는 길고 설명이 없는 대기 시간을 두 번째 사용자가 겪는 것도 원하지 않습니다.

의사 대화식 트랜잭션을 사용하는 경우 자원은 매우 잠깐 동안만 사용 불가능합니다(즉, 짧은 구성요소 트랜잭션). 그러나 모든 복구 가능 자원을 이러한 트랜잭션 중 하나로만 갱신할 수 있는 경우가 아니라면 복구는 불가능합니다. UOW가 트랜잭션을 전체적으로 확장할 수 없기 때문입니다. 따라서 이러한 방식으로 복구 가능 자원에 대한 갱신을 격리할 수 없는 경우 대화식 트랜잭션을 사용해야 합니다.

이전 예제에서는 의사 대화식 트랜잭션의 추가 문제점이 제기됩니다. 순서의 최종 트랜잭션에 대한 모든 갱신을 제한한다고 해도 첫 번째 사용자가 계속 변경사항을 입력하는 동안 두 번째 사용자가 같은 레코드에 대해 트랜잭션 갱신을 시작하는 것을 막을 수 있는 방법은 없습니다. 이는 무결성을 보장하기 위해 추가 애플리케이션 논리가 필요함을 의미합니다. 일부 큐에 넣기 양식을 사용하거나 갱신을 적용하기 전에 트랜잭션이 레코드의 원래 버전과 현재 버전을 비교하게 할 수 있습니다.

터미널 중단 기능

대화 태스크가 실행 중일 때 CICS는 다른 메시지가 해당 터미널로 전송되는 것을 허용하지 않습니다. 의사 대화식 애플리케이션은 메시지가 대화의 메시지 쌍 사이를 통과하도록 허용합니다.

대화 태스크에는 장점과 단점이 있습니다. 장점은 예기치 않은 메시지(예: 브로드캐스트)가 사용자 시스템 대화를 중단할 수 없고, 더 나쁘게는 형식화된 화면을 손상시킬 수 없다는 점입니다. 단점은 사용자가 예를 들어, 제어 운영자가 10분 후에 CICS를 시

시스템 종료하려는 의도 등과 같은 중요 정보를 통지받을 수 없다는 점입니다. 더욱 중요하게는, 사용자가 대화 종료에 실패하면 정상 CICS 시스템 종료를 방지하거나 지연시킬 수 있습니다.

의사 대화식 애플리케이션은 메시지를 허용할 수 있으므로 시스템 종료 경고 등과 같은 통지는 전달될 수 있습니다. 이는 표시 화면 내용을 방해할 수도 있으며 때로는 TRANSID 옵션이 있는 RETURN 명령에 의해 제어되는 트랜잭션 순서를 방해할 수 있습니다. 링크된 상호작용 순서의 중간에 IMMEDIATE 옵션을 사용하거나 터미널을 NOATI 상태로 강제 실행하여 이를 방지하십시오.

태스크 시작 방법

작업은 CICS에서 임의 입력 또는 자동 태스크 초기화(ATI)으로부터 시작됩니다(태스크 시동).

자동 태스크 초기화는 다음과 같은 경우에 발생합니다.

- 기존 태스크가 CICS에게 다른 태스크를 작성할지 묻습니다. START 명령, RETURN 명령의 IMMEDIATE 옵션(726 페이지의 『RETURN IMMEDIATE』에서 논의함) 및 SEND PAGE 명령(969 페이지의 『SEND PAGE 명령』) 모두 이를 수행합니다.
- CICS는 트랜지언트 데이터 큐를 처리할 태스크를 작성합니다(809 페이지의 『ATI(Automatic Transaction Initiation)』 참조).
- CICS는 BMS ROUTE 요청에서 전송된 메시지를 전달한 태스크를 작성합니다(987 페이지의 『메시지 라우팅』 참조). CICS 제공 트랜잭션 CMSG를 사용한 후 표시되는 CSPG 태스크는 이러한 태스크의 예입니다. CMSG는 대상 목록에 있는 각 대상 터미널에 대해 CSPG 트랜잭션을 작성하는 ROUTE 명령을 사용합니다.

그러나 태스크를 시작하는 기본 메커니즘은 임의 입력입니다. 사용자가 기존 태스크의 **프린시펄** 기능이 아닌 터미널에서 입력을 전송하면 CICS는 이를 처리할 태스크를 작성합니다. 입력을 전송한 터미널은 새 태스크의 **프린시펄** 기능이 됩니다.

프린시펄 기능

CICS에서는 태스크가 오직 한 개의 터미널 즉, **프린시펄** 기능과 직접 통신할 수 있습니다. CICS는 태스크를 시동할 때 **프린시펄** 기능을 지정하고 태스크는 그 지속 기간 동안 이 기능을 『소유』합니다. 다른 태스크는 소유 태스크가 종료될 때까지 해당 터미널을 사용할 수 없습니다. 태스크가 해당 **프린시펄** 기능 외의 터미널과 통신해야 하는 경우 **프린시펄** 기능인 터미널이 있는 다른 태스크를 작성하여 이를 간접적으로 수행해야 합니다. 이 요구사항은 인쇄와 관련되어 가장 흔하게 발생하며 그러한 태스크를 작성하는 방법은 887 페이지의 『CICS 프린터 사용』에 설명되어 있습니다.

다른 시스템으로부터의 임의 입력도 같은 방식으로 처리됩니다. CICS는 입력을 처리할 태스크를 작성하고 입력을 전달한 대화를 **프린시펄** 기능으로 지정합니다. (따라서 다른

시스템과의 대화는 프린시펄 또는 대체 기능일 수 있습니다. 한 CICS 리전의 태스크가 다른 CICS와의 대화를 시작하는 경우 이 대화는 시동 태스크에 대해서는 대체 기능이지만 수신 시스템에서 작성된 파트너 태스크에 대해서는 프린시펄 기능입니다. 반대로 터미널은 항상 프린시펄 기능입니다.)

대체 기능

태스크는 한 개의 터미널과만 통신할 수 있지만 하나 이상의 원격 시스템과도 통신을 설정할 수 있습니다. 이러한 설정은 CICS에 해당 시스템과의 대화를 대체 기능으로 지정하도록 요청하여 수행합니다. 태스크는 프린시펄 기능을 소유하는 것과 같은 방식으로 대체 기능을 『소유』합니다. 소유권은 지정 시점부터 태스크가 종료하거나 태스크가 기능을 해제할 때까지 지속됩니다.

모든 태스크에 프린시펄 기능이 있는 것은 아닙니다. 임의 입력으로 시작된 태스크는 항상 프린시펄 기능을 가지지만 자동 태스크 초기화에서 시작된 태스크는 프린시펄 기능이 필요하거나 필요하지 않을 수 있습니다. 프린시펄 기능이 필요한 경우 CICS는 요청 기능을 태스크에 지정할 수 있을 때까지 태스크 시동을 기다립니다.

트랜잭션 선택

임의 입력을 수신하면 실행할 트랜잭션을 CICS가 판별하는 방식은 일반적으로 같은 프린시펄 기능을 가진 이전 태스크에서 최종 RETURN 시 TRANSID 옵션을 사용하여 CICS가 다음에 실행할 트랜잭션을 판별하는 것입니다.

이 상황은 거의 항상 의사 대화식 트랜잭션 순서에서 발생하고 일반적으로 메뉴 구동 애플리케이션에서도 발생합니다. 실패 시 순서를 시작하는 경우에 CICS는 입력의 처음 몇 문자를 트랜잭션 코드로 해석합니다. 그러나 실제로는 그보다 더 복잡합니다. 정확한 프로세스는 다음과 같이 진행됩니다. 단계 번호는 테스트가 수행되는 순서를 나타내고 272 페이지의 그림 59(이 로직의 다이어그램)을 참조합니다.

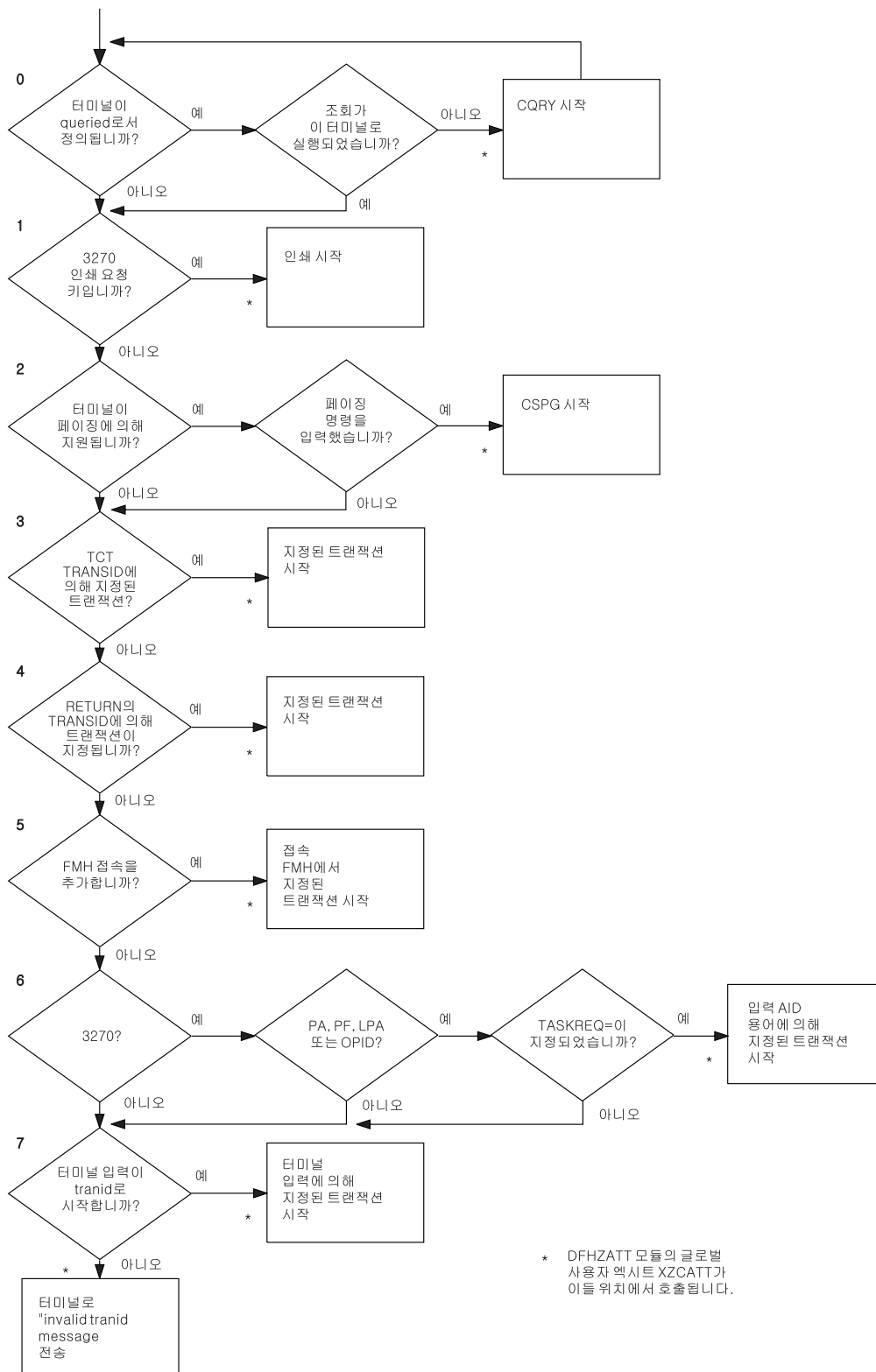


그림 59. 실행할 트랜잭션 판별

0. 터미널에서 처음 입력되면 CICS는 경우에 따라 입력을 처리하기 위한 태스크를 작성하기 전에 예비 태스크를 스케줄합니다. 이 태스크는 CICS 제공『조

회』 트랜잭션인 CQRY를 실행합니다. 그러면 터미널은 해당 하드웨어 특성 중 일부(예: 확장 속성, 문자 세트 등)의 인코딩된 설명을 전송하게 됩니다.

CQRY를 통해 시스템 프로그래머는 터미널 정의에서 이러한 상세사항을 생략하여 터미널 네트워크의 유지보수를 간소화할 수 있습니다. 이는 터미널 정의에서 그렇게 지정하고 입력을 처리하기 위해 사용할 후속 트랜잭션의 판별에 영향을 미치지 않는 경우에만 발생합니다. 후속 트랜잭션 판별은 다음과 같이 진행됩니다.

1. 터미널이 3270이고 입력이 "인쇄 요청 키"인 경우 화면의 내용을 인쇄하는 CICS 제공 트랜잭션인 CSPP가 시작됩니다. 이 기능에 대한 자세한 정보는 894 페이지의 『표시장치 화면 인쇄』의 CICS 인쇄 키를 참조하십시오. 이를 목적으로 『3270 논리 장치』 또는 3270 데이터 스트림을 승인하는 기타 디바이스는 3270으로 간주합니다.
2. 전체 BMS 지원이 있을 경우 터미널의 유형은 BMS 터미널 페이지징에서 지원되는 유형이고 입력은 페이지징 명령이며 CICS 제공 트랜잭션 CSPG가 요청을 처리하기 위해 시작됩니다. BMS 지원 레벨에 대한 설명은 912 페이지의 『BMS 지원 레벨』에 있고 같은 섹션에 BMS가 지원하는 목록도 포함되어 있습니다. 시스템 초기화 테이블에서 PGRET, SKRxxxx, PGCHAIN, PGCOPY 및 PGPURGE 옵션은 페이지징 명령을 정의합니다. 페이지징에는 전체 BMS가 필요하기 때문에 CICS 시스템에 해당 레벨 미만의 BMS가 있을 경우 이 단계는 건너뛸니다.
3. 터미널 정의가 특정 트랜잭션을 사용하여 해당 터미널로부터의 모든 임의 입력을 처리해야 한다고 표시하는 경우 표시되는 트랜잭션이 실행됩니다(있는 경우 이 정보는 TERMINAL 정의의 TRANSACTION 속성에 표시됨).
4. 터미널에서 이전 태스크가 태스크를 종료한 RETURN 명령의 TRANSID 옵션을 지정한 경우 이름 지정된 트랜잭션이 실행됩니다.
5. 입력에 연결 기능 관리 헤더가 있는 경우 헤더에 있는 연결 이름은 네 자로 된 CICS 트랜잭션 ID로 변환되고 이 트랜잭션이 실행됩니다.
6. 터미널이 3270이고 어텐션 ID가 트랜잭션으로 정의되는 경우 이 트랜잭션이 실행됩니다. 769 페이지의 『3270 터미널에서의 입력』의 어텐션 키는 어텐션 ID를 설명합니다. 해당 TRANSACTION 정의의 TASKREQ 속성을 사용하여 트랜잭션 ID를 정의합니다.
7. 모든 이전 테스트가 실패한 경우 입력의 처음 문자는 실행할 트랜잭션을 식별하는 데 사용됩니다. 사용되는 문자는 데이터 스트림의 제어 정보 뒤에 있고 첫 번째 필드 분리문자 앞에 있는 첫 번째 문자(최대 네 문자) 또는 다음 3270 제어 문자(X'00'부터 X'3F'까지)입니다. 필드 분리문자는 시스템 초기화 테이블(기본값은 공백임)의 FLDSEP 옵션에서 정의됩니다.

입력에 해당 문자가 없을 경우(CLEAR 키를 사용할 때 발생함) 또는 예를 들어, 입력과 일치하는 트랜잭션 정의가 없는 경우 CICS는 실행할 트랜잭션을 판별할 수 없기 때문에 『올바르지 않은 트랜잭션 ID』 메시지를 터미널로 전송합니다.

참고: 실행할 트랜잭션을 결정하기 위한 이 논리는 임의 입력을 처리하기 위해 시작된 태스크에만 적용됩니다. 자동 트랜잭션 초기화의 경우 항상 트랜잭션을 알 수 있습니다. START 또는 RETURN IMMEDIATE로 태스크를 작성할 때 TRANSID 옵션에서 트랜잭션을 지정합니다. 마찬가지로 큐 정의에서 트랜지언트 데이터 큐를 처리하는 데 사용할 트랜잭션을 지정합니다. 메시지를 라우팅하기 위해 작성된 태스크는 CICS 제공 트랜잭션 CSPG를 실행합니다.

비즈니스 및 표시 논리 분리

일반적으로 애플리케이션을 재사용 가능한 비즈니스 코드를 포함하는 파트와 클라이언트에 대한 표시에 책임이 있는 파트로 분할하는 것은 좋은 사례입니다. 이 기술을 통해 개별적으로 파트를 최적화하여 성능을 향상시키고 다른 표시 양식에서 비즈니스 로직을 재사용할 수 있습니다.

비즈니스 및 표시 논리를 분리할 때 다음을 고려해야 합니다.

- 애플리케이션의 두 파트 간 연관관계를 피하십시오.
- DPL 제한 API에 유의하고 세부사항은 *CICS 애플리케이션 프로그래밍 참조*를 참조하십시오.
- EIBTRMID 사용법과 같은 숨겨진 표시 종속 항목에 유의하십시오.

275 페이지의 그림 60에서는 사용자의 데이터를 승인하고, 파일의 레코드를 갱신하며, 사용자에게 다시 응답을 전송하는 단순 CICS 애플리케이션을 설명합니다. 이 프로그램을 실행하는 트랜잭션은 의사 대화에서 두 번째입니다. 첫 번째 트랜잭션은 BMS 맵을 사용자 터미널에 전송하고 두 번째 트랜잭션은 EXEC CICS RECEIVE MAP 명령을 사용하여 이 데이터를 읽고 파일의 레코드를 갱신한 후 EXEC CICS SEND MAP 명령을 사용하여 응답을 전송합니다.

EXEC CICS READ UPDATE 및 EXEC CICS REWRITE 명령이 비즈니스 로직의 일부인 반면 EXEC CICS RECEIVE 및 EXEC CICS SEND MAP 명령은 트랜잭션 표시 논리의 일부입니다.

```

..
EXEC CICS RECEIVE MAP
..
EXEC CICS READ UPDATE
..
EXEC CICS REWRITE
..
EXEC CICS SEND MAP
..

```

그림 60. 단일 애플리케이션의 CICS 함수

CICS 애플리케이션 디자인에서 모듈식 프로그래밍의 확실한 원칙은 비즈니스 로직에서 표시 논리를 분리하고 이를 단일 트랜잭션으로 만드는 데 통신 영역 및 EXEC CICS LINK 명령을 사용하는 것입니다. 그림 61 및 그림 62에서는 애플리케이션 디자인에 대한 이 접근법을 설명합니다.

```

..
EXEC CICS RECEIVE MAP
..
EXEC CICS LINK..
..
EXEC CICS SEND MAP
..

```

그림 61. 표시 논리

```

..
EXEC CICS ADDRESS COMMAREA
..
EXEC CICS READ UPDATE
..
EXEC CICS REWRITE
..
EXEC CICS RETURN..

```

그림 62. 비즈니스 로직

트랜잭션의 비즈니스 로직이 표시 논리에서 분리되고 통신 영역 인터페이스를 제공받으면 다른 표시 메소드에서 재사용할 수 있습니다. 예를 들어, 455 페이지의 『DPL(Distributed Program Link)』를 사용하여 2 티어 모델 또는 표시 논리가 HTTP 기반인 CICS 비즈니스 로직 인터페이스가 포함된 CICS 웹 지원을 구현할 수 있습니다.

멀티스레딩: 재진입 가능, 유사 재진입 가능 및 스레드 세이프 프로그램

멀티스레딩은 애플리케이션의 단일 사본을 여러 트랜잭션이 동시에 처리할 수 있는 기술입니다. 예를 들어, 한 개의 트랜잭션이 애플리케이션의 실행을 시작할 수 있습니다. CICS를 대기시키고 디스패처를 호출하게 하는 **EXEC CICS** 명령이 실행되면 다른 트랜잭션이 이 애플리케이션의 동일 사본을 실행할 수 있습니다.

이 기술을 완료될 때까지 프로그램을 실행하는 싱글스레딩과 비교합니다. 한 개의 트랜잭션에 의한 프로그램 처리가 완료되어야 다른 트랜잭션이 해당 프로그램을 사용할 수 있습니다.

멀티스레딩에서 모든 CICS 애플리케이션은 유사 재진입 가능이어야 합니다. 즉, 시작점과 종료점 사이에서 연속적으로 재사용가능해야 합니다. EXEC CICS 인터페이스를 사용하는 CICS 애플리케이션은 이러한 규칙을 자동으로 준수합니다. COBOL, C 및 C++ 프로그램의 경우 프로그램이 호출될 때마다 작업 중인 저장영역의 신규 사본을 확보하여 재진입성을 보장합니다. 컴파일 또는 미리 링크 유틸리티에서는 항상 RENT 옵션을 사용해야 하고 쓰기 가능 정적이 없고 자연적으로 재진입 가능한 C 및 C++ 프로그램에 대해서도 마찬가지입니다. CICS 변환기에서 삽입하는 임시 변수 및 DFHEIPTR 필드는 보통 쓰기 가능 정적 변수로 정의되고 RENT 옵션을 필요로 합니다. 이러한 프로그램을 재진입 가능 상태로 유지하려면 변수 데이터가 PL/I에서 정적 저장영역으로 표시되거나 어셈블러 언어 프로그램 CSECT에서 DC로 표시되지 않아야 합니다.

애플리케이션이 재진입 가능으로 컴파일 및 링크 편집되도록 요구하는 것 이외에도 CICS는 프로그램을 유사 재진입 가능 또는 스레드 세이프로 식별하고 이러한 속성은 PROGRAM 자원 정의에서 설정됩니다. 다음 테이블은 프로그램에서 사용 가능한 CONCURRENCY 및 API 설정과 애플리케이션이 실행되는 TCB의 유형을 표시합니다.

표 16. PROGRAM CONCURRENCY 및 API 설정과 사용된 TCB 유형의 조합

CONCURRENCY 속성	API 속성	CICS TCB
CONCURRENCY(QUASIRENT)	API(CICSAPI)	애플리케이션은 항상 CICS 유사 재진입 가능(QR TCB)에서 실행됩니다. 개방형 TCB로의 전환이 자원 관리자에서 발생하면 CICS는 애플리케이션으로 리턴하기 전에 항상 QR TCB로 리턴합니다.
CONCURRENCY(QUASIRENT)	API(OPENAPI)	올바르지 않은 조합입니다. OPENAPI 프로그램은 QR TCB에서 실행될 수 없습니다.
CONCURRENCY(THREADSAFE)	API(CICSAPI)	애플리케이션은 QR TCB 또는 개방형 TCB에서 실행될 수 있습니다. 개방형 TCB로의 전환이 자원 관리자에서 발생하면 CICS는 애플리케이션으로 리턴할 때 개방형 TCB에 남아 있습니다. QR TCB로의 전환이 발생하면 CICS는 애플리케이션으로 리턴할 때 QR TCB에 남아 있습니다.
CONCURRENCY(THREADSAFE)	API(OPENAPI)	CONCURRENCY(REQUIRED) API(OPENAPI)와 같습니다.
CONCURRENCY(REQUIRED)	API(CICSAPI)	이 애플리케이션은 항상 개방형 TCB에서 실행됩니다. CICS 서비스에서는 TCB 키 일치 비교가 필요하지 않기 때문에 애플리케이션은 항상 L8 개방형 TCB에서 실행됩니다. OPENAPI 자원 관리자는 L8 TCB를 사용하기 때문에 호출될 경우 TCB 전환이 필요하지 않습니다. QR TCB로의 전환이 발생하면 CICS는 애플리케이션으로 리턴할 때 개방형 TCB로 리턴합니다.

표 16. PROGRAM CONCURRENCY 및 API 설정과 사용된 TCB 유형의 조합 (계속)

CONCURRENCY 속성	API 속성	CICS TCB
CONCURRENCY(REQUIRED)	API(OPENAPI)	이 애플리케이션은 항상 개방형 TCB에서 실행됩니다. TCB의 키는 프로그램의 실행 키와 일치해야 합니다. CICS는 EXECKEY(USER)가 설정되는 경우에는 L9 TCB를 사용하고 EXECKEY(CICS)가 설정되는 경우에는 L8 TCB를 사용합니다. 애플리케이션이 사용자 키이고 OPENAPI 자원 관리자가 호출되면 L9 TCB에서 L8 TCB로의 전환이 발생합니다. CICS는 애플리케이션으로 리턴하기 전에 L9 TCB로 리턴합니다. QR TCB로의 전환이 발생하면 CICS는 애플리케이션으로 리턴할 때 개방형 TCB로 리턴합니다.

유사 재진입 가능 애플리케이션

유사 재진입 가능 프로그램은 각 **EXEC CICS** 명령 전후로 항목에서 둘 다 제어기 전달될 때 일정한 상태를 유지하는 프로그램입니다. 이러한 유사 재진입성은 애플리케이션의 각 호출이 이전 실행의 영향이나 여러 CICS 태스크로 인한 프로그램 동시 멀티스레딩의 영향을 받지 않도록 보장합니다.

CICS는 CICS 관리 태스크 제어 블록(TCB)에서 사용자 프로그램을 실행합니다. 프로그램이 유사 재진입 가능(프로그램 자원 정의의 CONCURRENCY 속성에서)으로 정의되는 경우 CICS는 항상 CICS 유사 재진입 가능(QR TCB)에서 그러한 프로그램을 호출합니다. 멀티스레딩 컨텍스트에서 준 재입력 프로그램의 요구사항은 프로그램이 다중 TCB에서 동시에 실행될 경우 덜 엄격합니다.

CICS에서는 애플리케이션이 일정한 상태를 보장할 수 있도록 재진입 가능이어야 합니다. 실제로는 애플리케이션이 재진입 가능이 아닐 수 있습니다. CICS는 『유사 재진입 가능』을 예상합니다.

예를 들어, 애플리케이션은 실행 코드나 프로그램 저장영역 내에서 정의된 변수를 수정할 수 있지만 태스크를 제어할 수 없고 다른 태스크가 같은 프로그램에서 실행될 가능성이 생기기 전에 그러한 변경을 실행 취소하거나 코드 및 변수를 다시 초기화해야 합니다.

QR TCB에서 CICS 디스패처가 CICS 유사 재진입 가능 사용자 프로그램(애플리케이션, 사용자 대체 가능 모듈, 글로벌 사용자 엑시트 및 태스크 관련 사용자 엑시트)에 제어를 제공합니다. 이 TCB에서 실행될 때 프로그램은 CICS 요청 중 프로그램이 제어를 포기할 때까지(이때 사용자 태스크는 일시중단되고 프로그램은 『사용 중』 상태로 유지됨) 다른 준 재입력 프로그램이 실행될 수 없음을 확신할 수 있습니다. 그러면 같은 프로그램을 다른 태스크 수행을 위해 재호출할 수 있습니다. 즉, 한 번에 한 개의 태스크만 실행될 수 있지만 애플리케이션이 둘 이상의 태스크에 의해 사용 중 상태일 수 있다는 의미입니다.

프로그램이 서로의 작업 중인 저장영역을 간섭할 수 없도록 하기 위해 CICS는 애플리케이션이 실행될 때마다 작업 중인 저장영역의 개별 사본을 얻습니다. 예를 들어, 사용자 애플리케이션을 11개의 사용자 태스크가 사용 중인 경우 해당 동적 저장영역(DSA)에는 11개의 작업 중인 저장영역의 사본이 있는 것입니다.

유사 재진입성을 통해 프로그램은 다른 프로그램의 동시 액세스로부터 자원을 보호할 필요 없이 글로벌로 공유하는 자원에 액세스할 수 있습니다(예: CICS CWA(Common Work Area)). 이러한 자원은 다음 CICS 요청이 발행되기 전까지 실행 중인 프로그램에 대해서만 효과적으로 잠깁니다. 예를 들어, 애플리케이션은 비교 및 스왑(CS) 지시 사항 또는 자원 잠금(큐에 넣기)을 수행하지 않고도 CWA에서 필드를 갱신할 수 있습니다.

참고: CICS QR TCB는 글로벌 자원에 액세스하는 모든 사용자 태스크가 QR TCB에서 실행되는 경우에만 글로벌 자원의 독점 제어를 통해 보호를 제공합니다. 다른 개방형 TCB에서 동시에 실행되는 다른 태스크에 대한 자동 보호는 제공하지 않습니다.

프로그램에 긴 계산이 포함되는 경우 주의하십시오. 애플리케이션은 한 EXEC CICS 명령에서 다른 명령이 있을 때까지 제어를 보유하기 때문에 QR TCB에서 다른 트랜잭션의 처리는 제외됩니다. 그러나 태스크 제어 SUSPEND 명령을 사용하여 다른 트랜잭션 처리가 진행되도록 할 수 있습니다. 세부사항은 781 페이지의 제 36 장 『태스크 제어』의 내용을 참조하십시오. 런어웨이 태스크 시간 간격은 트랜잭션 정의 및 초기화 매개변수 **ICVR**에 의해 제어됩니다. CICS는 지정된 시간 간격이 만료되기 전에 제어를 리턴하지 않는 태스크를 영구 제거합니다.

스레드 세이프 프로그램

CICS 개방형 트랜잭션 환경(OTE)에서 애플리케이션 프로그램, 태스크 관련 사용자 엑시트(TRUE), 글로벌 사용자 엑시트 프로그램 및 사용자 대체 가능 모듈이 CICS에 스레드 세이프로 정의되면 개방형 트랜잭션 환경(OTE)의 개방형 TCB에서 동시에 실행될 수 있습니다.

개방형 트랜잭션 환경에 액세스

ENABLE PROGRAM 명령에서 OPENAPI 옵션을 사용하여 사용 설정된 태스크 관련 사용자 엑시트(TRUE)를 포함하는 애플리케이션은 자동으로 개방형 트랜잭션 환경에 관련됩니다. 이러한 애플리케이션은 스레드 세이프 상태가 되면 성능상의 이점을 얻을 수 있습니다. DB2 자원에 액세스하는 CICS 애플리케이션이 사용하는 CICS DB2 태스크 관련 사용자 엑시트는 개방형 API TRUE이므로, CICS DB2 애플리케이션은 스레드 세이프 상태가 되면 성능상 이로우 수 있습니다. CICS DB2 애플리케이션의 스레드 세이프 프로그래밍에 대한 세부사항은 DB2 안내서에서 CICS DB2 애플리케이션이 스레드 안전 프로그래밍을 통해 개방 트랜잭션 환경(OTE)을 사용하도록 설정의 내용을 참조하십시오.

다른 사용자 애플리케이션, PLT 프로그램, 사용자 대체 기능 모듈 또는 태스크 관련 사용자 엑시트의 경우 OPENAPI 프로그램으로 정의하여 개방 트랜잭션 환경에서 사용하도록 선택할 수 있습니다. 이때 이러한 프로그램은 스레드 세이프여야 합니다. OPENAPI 프로그램의 스레드 세이프 프로그래밍에 대한 세부사항은 OPENAPI 프로그램의 내용을 참조하십시오.

프로그램을 CONCURRENCY(REQUIRED)로 정의하는 경우 이 프로그램은 항상 개방형 TCB에서 실행됩니다. 사용되는 개방형 TCB의 유형은 API 설정에 따라 달라집니다. CICSAPI 프로그램의 경우 CICS는 프로그램의 실행 키와 관계없이 L8 개방형 TCB를 사용합니다. OPENAPI 프로그램의 경우 CICS는 EXECKEY(USER)가 설정된 경우에는 L9 TCB를 사용하고 EXECKEY(CICS)가 설정된 경우에는 L8 TCB를 사용합니다. REQUIRED는 사용자 애플리케이션, PLT 프로그램 및 사용자 대체 기능 모듈에 적용할 수 있습니다. 글로벌 사용자 엑시트 프로그램 및 태스크 관련 사용자 엑시트 프로그램의 경우 CONCURRENCY(REQUIRED)를 지정하면 CONCURRENCY(THREADSAFE)를 지정한 것처럼 CICS가 프로그램을 처리합니다.

글로벌 사용자 엑시트는 OPENAPI 프로그램으로 정의할 수 없지만 글로벌 사용자 엑시트에 대해 **ENABLE PROGRAM** 명령에서 THREADSAFE 옵션을 사용하면 글로벌 사용자 엑시트가 스레드 세이프 프로그램으로 사용 설정되고 애플리케이션이 사용하는 것과 같은 개방형 TCB에서 애플리케이션으로 실행될 수 있습니다. **ENABLE PROGRAM** 명령이 CONCURRENCY 또는 API 옵션을 지정하지 않으면 프로그램 정의의 옵션이 사용됩니다.

일련화 기술

개방형 TCB에서 실행되고 있는 애플리케이션은 유사 재진입성에 의존하여 다른 프로그램의 동시 액세스로부터 공유 자원을 보호합니다. 또한 개방형 TCB에서 동시에 실행되고 있는 사용자 태스크도 액세스할 수 있는 공유 자원에 액세스하는 경우에 준 재입력 프로그램도 위험한 상태에 놓일 수 있습니다. 따라서 공유 자원에 액세스하기 위해 사용자 프로그램에서 사용하는 기술은 다른 프로그램에 의한 동시 액세스의 가능성을 고려해야 합니다.

공유 자원의 무결성을 유지하면서 개방형 트랜잭션 환경의 성능상 이점을 얻으려면 공유 자원에 대한 동시 액세스를 방지하는 일련화 기술을 사용해야 합니다. 공유 자원에 액세스할 때 적합한 일련화 기술을 사용하는 프로그램은 스레드 세이프 프로그램으로 설명됩니다.

파일, 트랜지언트 데이터 큐, 임시 저장영역 큐 및 DB2 테이블과 같은 대부분의 자원에 대한 CICS 처리에서는 자동으로 스레드 세이프 방식의 액세스를 보장합니다. 이러한 자원에 대해 작동하는 일부 CICS 명령은 개방형 TCB에서 실행될 수 있게 하는 적절한 일련화 기술을 사용하도록 코드화됩니다. 이러한 명령이 스레드 세이프 명령입니다. 그 외의 경우에는 CICS가 명령의 작동에 관계없이 자원에 대한 액세스가 일련화 되도록 QR TCB로의 전환을 강제 실행하여 스레드 세이프 프로그램으로 처리합니다.

공유 저장영역과 같이 사용자 프로그램이 직접 액세스하는 기타 자원의 경우 스레드 세이프 처리를 보장하는 것은 사용자 프로그램의 책임입니다. 사용자 애플리케이션의 스레드 세이프 프로그래밍에 대한 정보는 애플리케이션의 스레드 세이프화의 내용을 참조하십시오.

TCB 전환

스레드 세이프 프로그램은 개방형 TCB와 QR TCB 사이에서 전환하기 보다는 개방형 TCB에 남아서 성능상의 이점을 얻습니다. 프로그램이 스레드 세이프로 정의되고 개방형 TCB에서 실행되는 경우 개방형 TCB에서 QR TCB로의 TCB 전환은 다음 상황에서 발생합니다.

- 프로그램이 스레드 세이프가 아닌 EXEC CICS 명령을 실행하는 경우 CICS는 개방형 TCB에서 QR TCB로 다시 전환하여 비스레드 세이프 코드를 실행합니다. 프로그램이 OPENAPI 또는 CONCURRENCY(REQUIRED)로 정의되면 CICS는 개방형 TCB로 전환하여 애플리케이션 로직 실행을 계속합니다. 프로그램이 OPENAPI 또는 CONCURRENCY(REQUIRED)로 정의되지 않은 경우 프로그램은 QR TCB에서 계속 실행됩니다. CICS DB2 애플리케이션에서 프로그램이 OPENAPI 또는 CONCURRENCY(REQUIRED)로 정의되지 않고 더 이상 추가 DB2 요청을 하지 않을 경우 QR TCB로의 재전환은 불리합니다. 나머지 애플리케이션 코드를 실행하기 위해 소요되는 시간 동안 QR TCB의 사용량이 늘어나기 때문입니다. 그러나 프로그램이 추가 DB2 요청을 할 경우 CICS는 개방형 TCB로 다시 전환해야 합니다.
- 프로그램이 스레드 세이프로 정의되지 않은 태스크 관련 사용자 엑시트 프로그램을 호출하면 CICS가 QR TCB로 다시 전환하고 제어를 태스크 관련 사용자 엑시트 프로그램에 제공합니다. 태스크 관련 사용자 엑시트 프로그램이 처리를 완료하면 비스레드 세이프 EXEC CICS 명령 후의 상황과 같습니다. OPENAPI 또는 CONCURRENCY(REQUIRED) 프로그램이 개방형 TCB로 다시 전환하고 OPENAPI 또는 CONCURRENCY(REQUIRED)로 정의되지 않은 프로그램은 QR TCB에서 계속 실행됩니다.
- 프로그램이 스레드 세이프 CICS 명령을 실행하거나 DB2 요청을 하는 경우 명령 또는 요청에 대한 처리의 일부로 글로벌 사용자 엑시트가 호출되어야 합니다. 스레드 세이프로 정의되지 않은 글로벌 사용자 엑시트 프로그램이 사용되는 경우 CICS는 QR TCB로 다시 전환하고 글로벌 사용자 엑시트 프로그램에 제어를 제공합니다. 사용자 엑시트 프로그램이 처리를 완료하면 CICS가 개방형 TCB로 다시 전환하여 스레드 세이프 CICS 명령 처리를 계속하거나 DB2 요청을 완료합니다.
- 프로그램이 완료되면 CICS가 태스크 종료를 위해 QR TCB로 다시 전환합니다. 이러한 전환은 항상 필요합니다.

정적 또는 동적으로 호출된 루틴에 대한 스레드세이프 고려사항

CONCURRENCY(THREADSAFE) 또는 CONCURRENCY(REQUIRED)로 프로그램을 정의하는 경우에는 해당 프로그램에서 정적으로나 동적으로 호출되는 모든 루틴 (예: COBOL 루틴)은 또한 스레드세이프 표준으로 코딩되어야 합니다.

EXEC CICS LINK 명령이 한 프로그램에서 다른 프로그램으로 링크하는 데 사용되면 프로그램 링크 스택 레벨이 증가됩니다. 그러나 정적으로 호출되거나 동적으로 호출된 루틴이 CICS 명령 레벨 인터페이스 통과와 관련되지 않으므로 프로그램 링크 스택 레벨이 증가되지 않습니다. COBOL 루틴을 사용하면 정적 호출의 경우 단순 분기와 링크가 링크 편집 시에 해석되는 주소와 관련됩니다. 동적 호출의 경우 프로그램 정의가 관련되어 있지만 Language Environment가 프로그램을 로드할 수 있도록 허용하기 위해 서만 필수입니다. 이후에는 단순 분기와 링크가 실행됩니다. 이러한 메소드 중 하나에 의해 루틴이 호출되면 CICS는 이를 프로그램의 변경으로 간주하지 않습니다. 루틴을 호출한 프로그램은 여전히 실행 중인 것으로 간주되므로 해당 프로그램의 프로그램 정의는 여전히 최신인 것으로 간주됩니다.

호출 프로그램의 프로그램 정의가 CONCURRENCY(THREADSAFE) 또는 CONCURRENCY(REQUIRED)를 선언하면 호출된 루틴 또한 이 스펙을 준수해야 합니다. CONCURRENCY(THREADSAFE) 또는 CONCURRENCY(REQUIRED) 속성이 있는 프로그램은 DB2 호출에서 리턴될 때 열린 TCB에 남아 있고, 스레드세이프가 아닌 프로그램에는 적합하지 않습니다. 예를 들어, 트랜잭션의 초기 프로그램, 프로그램 A가 COBOL 루틴인 프로그램 B에 동적 호출을 실행하는 상황을 고려하십시오. CICS 명령 레벨 인터페이스가 관련되지 않았으므로 CICS는 프로그램 B에 대한 호출에 대해 알지 못하고, 현재 프로그램이 프로그램 A가 될 것으로 고려합니다. 프로그램 B가 DB2 호출을 실행합니다. DB2 호출로부터의 리턴 시에 CICS는 프로그램이 열린 TCB에 남아 있을 수 있는지 또는 프로그램이 스레드세이프 처리를 위해 QR TCB로 전환되어야 하는지 여부를 판별해야 합니다. 이를 수행하기 위해 CICS는 현재 프로그램으로 간주되는 프로그램(프로그램 A)의 CONCURRENCY 속성을 검사합니다. 프로그램 A가 CONCURRENCY(THREADSAFE) 또는 CONCURRENCY(REQUIRED)로 정의되는 경우에는 처리는 열린 TCB에서 계속될 수 있습니다. 이 시나리오에서는 프로그램 B가 실행 중이므로 처리가 안전하게 계속되는 경우에는 프로그램 B는 스레드세이프 표준으로 코딩되어야 합니다.

애플리케이션 스레드 세이프화

애플리케이션을 스레드 세이프화하면 개방형 트랜잭션 환경을 사용하고 TCB 전환을 피하며 성능상의 이점을 얻을 수 있습니다.

시작하기 전에

스레드 세이프 애플리케이션을 사용하려면 시스템 초기화 매개변수 **FORCEQR**이 YES로 설정되지 않는지 확인하십시오. **FORCEQR**은 스레드 세이프 정의된 프로그램이 QR TCB

에서 실행되도록 강제 실행하고 스레드 세이프 정의 프로그램과 관련된 문제점을 조사하고 해결하는 동안 임시 조치로 이 매개변수를 YES로 설정할 수 있습니다.

또한 파일 소유 CICS 리전에서 시스템 초기화 매개변수 **FCQRONLY**에 대해 적절한 설정을 선택하십시오. **FCQRONLY**가 YES로 설정된 경우 CICS는 CICS 리전의 모든 파일 제어 요청이 QR TCB에서 실행되도록 강제 실행합니다.

- 원격 CICS TS 4.2 또는 이후 리전에 대한 ship 파일 제어 요청을 실행하기 위해 IPIC 연결을 사용하는 경우 그러한 연결의 성능을 개선하려면 파일 소유 리전에서 **FCQRONLY**를 NO로 설정하십시오.
- MRO 링크 또는 SNA를 통한 ISC 연결만 사용하거나 파일 소유 리전이 CICS TS 4.2 이전인 경우 파일 소유 리전에서 **FCQRONLY**를 YES로 설정하십시오.

원격 CICS에서 실행할 함수 또는 프로그램을 요청하기 위해 CICS 상호 통신을 사용하는 경우 CICS 시스템 간의 TCP/IP 연결을 통한 IP 상호연결(IPIC)을 선택하여 스레드 세이프 애플리케이션에 최적의 지원을 제공하십시오. IPIC 연결에서 CICS는 개방형 TCB를 사용하여 원격 CICS 시스템에 대한 요청을 관리하는 미러 프로그램을 실행하고 처리량 향상을 제공합니다. 다른 두 개의 연결 유형에서 CICS는 미러 프로그램 실행에 개방형 TCB를 사용하지 않습니다. DPL(Distributed Program Link)에 사용되는 EXEC CICS LINK 명령은 장기 실행 미러가 사용되는 원격 CICS 리전에 대한 IPIC 연결에서 스레드 세이프이며 다른 연결 유형에서는 스레드 세이프가 아닙니다.

이 태스크 정보

스레드 세이프 프로그램은 프로그램이 스레드 세이프가 되는 것의 의미 및 개방형 TCB와 QR TCB 사이에서 TCB 전환이 발생하는 환경에 대해 설명합니다.

애플리케이션을 스레드 세이프화하고 개방형 TCB에 남을 수 있게 하려면 다음 프로시저를 사용하십시오.

프로시저

1. PROGRAM 자원 정의에서 CONCURRENCY(THREADSAFE)를 지정하여 CICS에 프로그램을 스레드 세이프로 정의하십시오.

OPENAPI로 정의되는 프로그램의 경우 CICS에는 CONCURRENCY(THREADSAFE) 옵션이 필요합니다. 스레드 세이프로 정의된 코드만 개방형 TCB에서 실행이 허용됩니다. CICS에 프로그램을 스레드 세이프로 정의하면 프로그램에 포함된 모든 EXEC CICS 명령이 스레드 세이프임을 지정하는 것이 아니고 애플리케이션 로직이 스레드 세이프인 것만 지정합니다. CICS는 TCB 전환을 사용하여 EXEC CICS 명령이 처리되도록 할 수 있지만 프로그램이 개방형 TCB에서 실행될 수 있도록 CICS에서 사용자는 애플리케이션 로직이 스레드 세이프임을 보장해야 합니다.

또는 프로그램을 CONCURRENCY(REQUIRED)로 정의하여 프로그램이 처음부터 개방형 TCB에서 실행되도록 할 수 있습니다. CONCURRENCY(REQUIRED)로 정의된 프로그램은 항상 개방형 TCB에서 실행되어야 하기 때문에 스레드 세이프 표준으로 코드화되어야 합니다. 사용되는 개방형 TCB의 유형은 API 설정에 따라 달라집니다.

2. 프로그램 로직 즉, EXEC CICS 명령 간 고유 언어 코드가 스레드 세이프인지 확인하십시오.

CICS에 스레드 세이프이지만 스레드 세이프가 아닌 애플리케이션 로직을 포함하는 것으로 프로그램을 정의하는 경우 결과는 예측할 수 없으며 CICS는 발생 가능한 결과로부터 사용자를 보호할 수 없습니다. 프로그램 로직을 스레드 세이프화하려면 공유 자원에 액세스할 때 적절한 일련화 기술을 사용해서 해당 자원에 대한 동시 액세스를 방지해야 합니다. EXEC CICS 명령을 사용하여 파일, 트랜지언트 데이터 큐, 임시 저장영역 큐 및 DB2 테이블과 같은 자원에 액세스하는 경우에는 CICS가 스레드 세이프 처리를 보장하지만 공유 저장영역과 같이 사용자 프로그램이 직접 액세스하는 모든 자원에 대해서는 사용자 프로그램이 스레드 세이프 처리를 보장해야 합니다.

공유 저장영역의 일반적인 예로는 CICS CWA, 글로벌 사용자 엑시트를 위한 글로벌 작업 영역 및 공유 옵션과 애플리케이션에 의해 명시적으로 획득되는 저장영역이 있습니다. 다음 EXEC CICS 명령이 발생했는지 찾아서 애플리케이션이 이러한 유형의 공유 저장영역을 사용하는지 여부를 확인하십시오.

- ADDRESS CWA
- EXTRACT EXIT
- GETMAIN SHARED

로드 모듈 스캐너 유틸리티에는 공유 저장영역에 대한 액세스를 제공하는 CICS 명령을 식별하는 데 도움이 되는 샘플 테이블 DFHEIDTH가 포함됩니다. 이러한 명령 중 일부는 자체가 스레드 세이프이지만 모두 글로벌 저장영역에 대한 액세스를 제공하기 때문에 이러한 명령을 따르고 글로벌 저장영역을 사용하는 애플리케이션 로직은 잠재적으로 스레드 세이프가 아닐 수 있습니다. 스레드 세이프 상태를 보장하려면 애플리케이션에 동시 갱신으로부터 보호하기 위해 필요한 동기화 로직이 포함되어야 합니다.

팁: 공유 자원을 사용하는 프로그램을 식별할 때 자체적으로 수정하는 프로그램도 포함시키십시오. 그러한 프로그램은 효과적으로 저장영역을 공유하지만 위험한 상태로 간주해야 합니다.

공유 자원에 액세스할 때 다음 기술을 사용하여 스레드 세이프 처리를 제공하십시오.

- 재시도 액세스, 자원이 다른 프로그램과 동시에 변경된 경우 비교 및 스왑 지시 사항을 사용합니다.
- 자원에 대한 큐에 넣기, 독점적 제어를 획득하고 다른 프로그램이 자원에 액세스 할 수 없도록 보장하기 위해 다음과 같은 기술 중 하나를 사용합니다.
 - 애플리케이션에서 EXEC CICS ENQ 명령
 - 글로벌 사용자 엑시트 프로그램에서 CICS 큐에 넣기(NQ) 도메인에 대한 XPI ENQUEUE 함수 호출
 - L8 TCB가 사용 가능한 경우에만 개방형 API 태스크 관련 사용자 엑시트에 서 ENQ와 같은 MVS 서비스 QR TCB에서 실행될 수 있는 애플리케이션 에서 MVS 서비스의 사용으로 TCB가 대기 상태가 되어 성능 저하가 발생할 수 있음에 유의하십시오.
- EXEC CICS LINK 명령으로 준 재입력 프로그램에 링크하여 유사 재진입 가능으로 정의되는 프로그램에서만 공유 자원에 대한 액세스를 수행합니다. 이 기술은 스레드 세이프 애플리케이션 및 개방형 API 태스크 관련 사용자 엑시트에 만 적용됩니다. 재진입 가능으로 정의된 링크 대상 프로그램은 QR TCB에서 실행되고 CICS 유사 재진입성으로 제공되는 일련화의 장점을 취할 수 있습니다. 유사 재진입 가능 모드라고 해도 일련화는 프로그램이 제어를 보유하고 대기하지 않는 경우에만 제공됨에 유의하십시오.
- 공유 자원에 액세스하는 모든 트랜잭션을 MAXACTIVE(1)로 지정되는 활성 태 스크 수를 사용하여 정의되는 제한된 트랜잭션 클래스(TRANCLASS)에 배치하 십시오. 이 접근법은 매우 단순한 잠금 메커니즘을 효과적으로 제공하지만 성능 에 심각한 영향을 미칠 수 있습니다.

참고: 스레드 세이프라는 용어가 개별 프로그램의 컨텍스트에서 정의되기는 하지만 공유 자원에 액세스하는 모든 애플리케이션이 규칙을 준수하는 경우에만 사용자 애플리케이션 전체를 스레드 세이프로 간주할 수 있습니다. 스레드 세이프 표준에 대 해 정확하게 작성되는 프로그램은 동일한 자원에 액세스하는 다른 프로그램이 스레 드 세이프 규칙을 준수하지 않는 경우에는 공유 자원을 안전하게 갱신할 수 없습니 다.

3. 최고의 성능을 얻으려면 프로그램이 스레드 세이프 EXEC CICS 명령만 사용하는 지 확인하십시오.

스레드 세이프가 아닌 EXEC CICS 명령을 스레드 세이프에 정의되어 개방형 TCB 에서 실행되고 있는 프로그램에 포함시키면 CICS가 개방형 TCB에서 QR TCB로 다시 전환하여 명령이 안전하게 처리되는지 확인합니다. 애플리케이션의 결과에는 영 향을 주지 않지만 그 성능에는 영향을 미칠 수 있습니다.

스레드 세이프인 명령은 CICS API 및 SPI 명령 주제의 명령 설명에 『이 명령은 스레드 세이프임』이라는 설명과 함께 표시되어 있습니다. 또한 참조 -> 애플리케이션

선 개발에서 스레드 안전 명령 및 참조 > 시스템 프로그래밍의 스레드세이프 (threadsafe) SPI 명령에도 나열되어 있습니다.

로드 모듈 스캐너 유틸리티에는 스레드 세이프가 아닌 애플리케이션에서 CICS 명령을 식별하는 데 도움이 되는 샘플 테이블 DFHEIDNT가 포함됩니다.

4. 프로그램이 사용하는 실행 경로에 사용자 엑시트 프로그램이 있을 경우 최고의 성능을 얻으려면 프로그램이 스레드 세이프 표준으로 코드화되고 스레드 세이프 CICS에 정의되는지 확인하십시오. 이러한 엑시트는 동적 계획 엑시트, 글로벌 사용자 엑시트 또는 태스크 관련 사용자 엑시트일 수 있습니다. 벤더 소프트웨어에서 제공된 사용자 엑시트 프로그램이 스레드 세이프 표준으로 코드화되는지 및 CICS에 스레드 세이프에 정의되는지도 확인하십시오.

스레드 세이프 사용자 엑시트 프로그램은 해당 프로그램을 호출하는 스레드 세이프 애플리케이션과 같은 개방형 TCB에서 사용할 수 있고 서브태스크 TCB를 작성 및 관리하지 않고도 비CICS API를 사용하고 스스로 개방형 트랜잭션 환경을 이용할 수 있습니다. 프로그램이 사용하는 실행 경로에 있는 사용자 엑시트 프로그램이 스레드 세이프가 아닌 경우 CICS는 QR TCB로 전환하여 해당 프로그램을 실행합니다. 이 경우 애플리케이션의 성능에 좋지 않은 영향을 미칠 수 있습니다.

다음과 같이 중요한 사용자 엑시트에 유의하십시오.

- 글로벌 사용자 엑시트 XEIN 및 XEIOU는 EXEC CICS 명령 전과 후에 호출됩니다.
 - 글로벌 사용자 엑시트 XPCFTCH는 CICS에 정의된 프로그램이 제어를 수신하기 전에 호출됩니다.
 - CICS DB2 요청의 경우 CICS DB2 태스크 관련 사용자 엑시트 DFHD2EX1은 스레드 세이프입니다. CICS DB2 요청에 대한 다른 중요한 엑시트에는 스레드 세이프에 정의되지 않는 기본 동적 계획 엑시트 DSNCEXT, 스레드 세이프에 정의되는 대체 동적 계획 엑시트 DFHD2PXT 및 글로벌 사용자 엑시트 XRMIIN과 XRMIOUT가 포함됩니다.
 - IPIC 연결을 통해 원격 CICS 리전에 대한 ship CICS 파일 제어, 트랜지언트 데이터 또는 임시 저장영역 요청을 실행할 경우 해당 요청은 스레드 세이프이고 개방 TCB의 원격 CICS 리전에서 실행될 수 있습니다. 파일 제어, 트랜지언트 데이터 또는 임시 저장영역 요청을 위해 원격 CICS 리전에서 호출되는 글로벌 사용자 엑시트 프로그램은 최고의 성능을 얻기 위해 스레드 세이프 프로그램으로 사용 설정되어야 합니다.
- a. CICS에 사용자 엑시트 프로그램을 스레드 세이프에 정의하기 위해 해당 PROGRAM 자원 정의에서 적절한 속성을 지정할 수 있습니다.
 - 태스크 관련 사용자 엑시트 프로그램의 경우 OPENAPI 및 THREADSAFE 또는 THREADSAFE만 지정하십시오.

- 글로벌 사용자 엑시트 프로그램의 경우 OPENAPI를 사용할 수 없지만 THREADSAFE를 지정할 수 있습니다.

글로벌 사용자 엑시트 프로그램 또는 태스크 관련 사용자 엑시트 프로그램에서 CONCURRENCY(REQUIRED)를 지정할 경우 CICS는 CONCURRENCY(THREADSAFE)를 지정한 것처럼 프로그램을 처리합니다.

- 또는 CICS에 사용자 엑시트 프로그램을 스레드 세이프코 정의하기 위해 **EXEC CICS ENABLE PROGRAM** 명령을 사용하여 프로그램을 사용으로 설정할 때 적절한 옵션을 지정할 수 있습니다.
 - 태스크 관련 사용자 엑시트 프로그램의 경우 OPENAPI 또는 THREADSAFE를 지정하십시오.
 - 글로벌 사용자 엑시트 프로그램의 경우 OPENAPI를 사용할 수 없지만 THREADSAFE를 지정할 수 있습니다.

OPENAPI 또는 THREADSAFE 옵션을 사용하여 엑시트 프로그램을 사용으로 설정할 때 이 조치는 프로그램 로직이 스레드 세이프코 CICS가 해당 엑시트의 프로그램 정의에서 CONCURRENCY 설정을 덮어쓰고 엑시트 프로그램을 스레드 세이프코 처리함을 CICS에 표시합니다.

- 첫 번째 단계 PLT 글로벌 사용자 엑시트 프로그램을 스레드 세이프코 정의하려면 **EXEC CICS ENABLE PROGRAM** 명령에서 THREADSAFE를 지정하십시오. CICS 초기화 중에 글로벌 사용자 엑시트 프로그램(예: 복구 엑시트 포인트에서 실행되는 글로벌 사용자 엑시트 프로그램)을 최대한 먼저 사용할 수 있도록 하기 위해 해당 프로그램을 첫 번째 단계 PLT 프로그램에서 사용으로 설정하는 것이 일반적인 사례입니다. 첫 번째 단계 PLT는 CICS 초기화 중 거의 처음에 실행되기 때문에 엑시트 프로그램을 정의하기 위해 프로그램 자동 설치 사용자 프로그램 또는 설치된 PROGRAM 자원 정의를 사용할 수 없습니다. CICS는 CONCURRENCY(QUASIRENT)가 설정된 첫 번째 단계 PLT 프로그램에서 사용 설정된 엑시트 프로그램을 자동으로 설치합니다. 그러나 **EXEC CICS ENABLE PROGRAM** 명령의 설정이 시스템 자동 설치된 프로그램 정의에서 CONCURRENCY(QUASIRENT) 설정을 덮어씁니다.

CONCURRENCY(REQUIRED) 프로그램

애플리케이션을 CONCURRENCY(REQUIRED)로 정의하는 것은 프로그램의 시작부터 프로그램이 기본 CICS 유사 재진입 가능 TCB(QR TCB) 대신 개방형 태스크 제어 블록(개방형 TCB)에서 계속 실행됨을 의미합니다. CICS가 QR TCB로 전환하여 **EXEC CICS** 명령을 처리하는 경우 CICS는 개방형 TCB로 다시 전환하여 애플리케이션에 제어를 리턴합니다.

사용되는 개방형 TCB의 유형은 프로그램이 사용할 API에 따라 다릅니다.

- 프로그램이 CICS가 지원하는 API(DB2, IMS 및 WebSphere MQ와 같은 외부 자원 관리자에 대한 액세스 포함)만 사용하는 경우 프로그램 속성 API(CICSAPI)를 사용하여 정의해야 합니다. 이 경우 CICS는 프로그램의 실행 키에 관계없이 항상 L8 개방형 TCB를 사용합니다. CICS 명령은 TCB의 키에 의존하지 않기 때문입니다.
- 프로그램이 다른 비CICS API를 사용하는 경우 프로그램 속성 API(OPENAPI)를 사용하여 정의되어야 합니다. 이 경우 CICS는 프로그램의 실행 키에 따라 L9 TCB 또는 L8 TCB를 사용합니다. 이는 비CICS API가 올바르게 작동할 수 있게 합니다.

글로벌 사용자 엑시트는 CONCURRENCY(REQUIRED)로 정의할 수 있지만 ENABLE PROGRAM 명령에서 THREADSAFE 옵션을 사용하여 사용 설정될 경우 필요 시 개방형 TCB에서 실행될 수 있습니다.

태스크 관련 사용자 엑시트(TRUE)는 CONCURRENCY(REQUIRED)로 정의하거나 사용 설정할 수 있습니다. 이 경우 스레드 세이프 표준에 맞게 작성되어야 합니다. CONCURRENCY(REQUIRED) API(OPENAPI)로 정의되는 TRUE는 항상 L8 TCB에서 실행됩니다. CONCURRENCY(REQUIRED) API(CICSAPI)로 정의되는 TRUE는 애플리케이션 환경의 요구에 따라 T8, L8 또는 X8 TCB에서 실행될 수 있습니다. CICS-DB2 TRUE는 CONCURRENCY(REQUIRED) API(CICSAPI)로 사용 설정됩니다.

(DB2 호출과 같은 TCB에서 실행될 수 있는 성능상의 이점을 이용하기 위해) THREADSAFE CICSAPI로 정의되는 기존 스레드 세이프 CICS-DB2 애플리케이션을 REQUIRED CICSAPI로 정의하여 추가로 개선할 수 있습니다. 이 정의는 첫 번째 DB2 호출이 프로그램을 개방형 TCB로 이동시킬 때까지 대기하지 않고도 프로그램이 L8 개방형 TCB에서 실행될 수 있음을 의미합니다. 추가 이점은 애플리케이션이 실행하는 비스레드 세이프 CICS 명령의 수(있는 경우)에 따라 다릅니다.

OPENAPI 프로그램

개방형 트랜잭션 환경(OTE)은 CICS 애플리케이션 코드가 CICS 주소 공간 내에서 다른 트랜잭션의 간섭 없이 비CICS 서비스를 사용할 수 있는 환경입니다. PROGRAM 자원 정의에서 OPENAPI 속성을 통해 사용자 애플리케이션, PLT 프로그램, 사용자 대체 가능 모듈 또는 태스크 관련 사용자 엑시트(TRUE)를 OPENAPI 프로그램으로 정의하여 OTE를 사용할 수 있습니다.

속성 API(OPENAPI)를 사용하여 프로그램을 OPENAPI 프로그램으로 정의하는 것은 프로그램이 시작부터 항상 기본 CICS 유사 재진입 가능 TCB(QR TCB)가 아닌 L8 또는 L9 모드 개방형 태스크 제어 블록(개방형 TCB)에서 실행됨을 의미합니다. 글로벌 사용자 엑시트는 OPENAPI 프로그램으로 정의될 수 없지만 **ENABLE PROGRAM** 명령에서 THREADSAFE 옵션을 사용하여 사용으로 설정할 경우 필요 시 개방형 TCB에서 실행될 수 있습니다.

QR TCB의 애플리케이션 워크로드를 여러 개방형 TCB로 분산시키면 처리량 특히, CPU 집약 프로그램의 처리량이 향상될 수 있습니다. 다른 비CICS API를 사용할 수 있지만 CICS 내에서 비CICS API를 사용하는 것은 완전히 사용자의 재량이고 사용자의 위험입니다. CICS 내에서 비CICS API 테스트가 수행되지 않았으며 그러한 API의 사용은 IBM 서비스에서 지원되지 않습니다.

개방형 TCB에서 실행되기 위해서는 OPENAPI 프로그램이 스레드 세이프해야 합니다. 스레드 세이프 프로그램에 대한 요구사항은 다음과 같습니다.

1. 이 프로그램은 CICS에 대해 CONCURRENCY(REQUIRED)로 정의되어야 합니다. 즉, 프로그램이 개방형 TCB에서 실행되어야 한다는 의미입니다.
2. 프로그램의 로직(즉, EXEC CICS 명령 간 고유 언어 코드)은 스레드 세이프해야 합니다. 개방형 TCB에서 실행되지만 스레드 세이프가 아닌 애플리케이션 로직을 포함하는 프로그램을 CICS에 정의하는 경우 결과를 예측할 수 없으며 CICS는 발생 가능한 결과로부터 사용자를 보호할 수 없습니다.
3. 최고의 성능을 얻으려면 프로그램이 스레드 세이프 EXEC CICS 명령만 사용해야 합니다. 개방형 TCB에서 실행 중인 프로그램에 스레드 세이프가 아닌 EXEC CICS 명령을 포함시키는 경우 CICS는 개방형 TCB에서 QR TCB로 다시 전환하여 명령이 안전하게 처리되도록 합니다. TCB 전환은 애플리케이션의 성능에 나쁜 영향을 미칠 수 있습니다.
4. 최고의 성능을 얻으려면 프로그램이 사용하는 실행 경로에 있는 사용자 엑시트 프로그램도 스레드 세이프 표준으로 코드화하고 CICS에 스레드 세이프 프로그램으로 정의되어야 합니다. 프로그램이 사용하는 실행 경로에 있는 사용자 엑시트 프로그램이 스레드 세이프가 아닌 경우 CICS가 QR TCB로 전환하여 해당 프로그램을 실행합니다. 이 경우 애플리케이션의 성능에 좋지 않은 영향을 미칠 수 있습니다.

278 페이지의 『스레드 세이프 프로그램』에서는 스레드 세이프 프로그램의 요구사항에 대해 자세히 설명합니다.

OPENAPI 프로그램에는 몇 가지 추가적 의무와 제한사항이 있습니다. 예를 들어, 이러한 프로그램은 특히 종료 태스크를 대신해서 획득한 모든 비CICS 자원이 사용 가능한 상태가 되고 특정 MVS 시스템 서비스를 사용하지 않도록 해야 합니다. 289 페이지의 『OPENAPI 프로그램을 위한 스레드세이프 제한사항』에서는 이러한 요구사항을 설명합니다.

REQUIRED OPENAPI(애플리케이션 로직이 스레드 세이프인 것으로 가정)로 정의할 후보 프로그램은 자기 책임 하에 다른 비CICS API를 사용하려고 하는 프로그램입니다.

OPENAPI 프로그램을 위한 TCB

다음 TCB는 OPENAPI 프로그램에서 사용됩니다.

- L8 모드 TCB는 웹 서비스 요청 처리, XML 구문 분석 및 CICS 웹 지원용 z/OS UNIX 파일에 액세스 등을 수행하기 위해 L8 TCB에서 실행되는 일부 CICS 프로그램을 포함한 CICS 키 OPENAPI 애플리케이션에 사용됩니다.
- L9 모드 TCB는 사용자 키 OPENAPI 애플리케이션에 사용됩니다.

L8 모드 TCB는 프로그램이 ENABLE PROGRAM 명령의 OPENAPI 옵션을 사용하여 사용으로 설정된 TRUE를 통해 자원 관리자에 액세스해야 하는 경우에도 사용됩니다. 개방형 API TRUE는 L8 모드 TCB에서 제어를 제공받고 서브태스크 TCB를 작성하지 않고도 비CICS API를 사용할 수 있습니다.

CICS는 L8 및 L9 모드 개방형 TCB의 풀에서 L8 및 L9 TCB의 수를 자동으로 제어합니다.

OPENAPI 프로그램을 사용하면 보통 스레드 세이프 프로그램에 비해 더 많은 TCB 전환이 발생할 수 있습니다. OPENAPI 프로그램이 EXEC CICS 명령 또는 스레드 세이프가 아닌 사용자 엑시트 프로그램을 사용하여 QR TCB로의 전환을 발생시키는 경우 추가 전환이 발생합니다. CICS가 애플리케이션 로직을 계속해서 실행하기 위해 개방형 TCB로 다시 전환하기 때문입니다. TCB의 키가 OPENAPI 프로그램에 대해 정확해야 한다는 요구사항으로 인해 추가 TCB 전환이 포함될 수 있습니다. OPENAPI TRUE는 항상 L8 TCB의 CICS 키에서 실행됩니다. 예를 들어, 사용자 키 OPENAPI 프로그램이 L9 TCB에서 실행되지만 DB2 호출을 하는 경우 CICS는 DB2를 호출하기 위해 L8 TCB로 전환한 다음 L9 TCB로 리턴하여 프로그램을 계속해서 실행합니다. 이 전환으로 인해 CICS DB2 애플리케이션은 보통 OPENAPI 프로그램이 아닌 (CICSAPI) 스레드 세이프 프로그램으로 정의됩니다. CICS 키 CICS DB2 애플리케이션은 필요한 경우 OPENAPI 프로그램으로 정의될 수 있습니다.

OPENAPI 프로그램을 위한 스레드세이프 제한사항

OPENAPI 프로그램은 QR TCB에 의해 부과되는 제한사항으로부터는 자유롭지만 여전히 전체적인 CICS 시스템 또는 사용 중인 L8 또는 L9 TCB의 미래 사용자 둘 모두에게는 의무가 있습니다.

L8 또는 L9 TCB는 이들이 할당된 CICS 태스크가 전용으로 사용하도록 지정되었지만 CICS 태스크가 완료된 후에는 TCB는 여전히 "정리" 상태에 있더라도 이러한 TCB의 디스패처 관리 풀로 돌아갑니다. (이 컨텍스트에서 정리되지 않은 TCB는 L8 또는 L9 모드 TCB를 사용하는 태스크가 OPENAPI 프로그램에서 처리되지 않은 이상 종료료를 겪었음을 의미합니다. 프로그램이 CICS가 발견하지 못한 스레드세이프 제한사항을 위반했다는 의미는 아닙니다.) TCB는 특정 OPENAPI 프로그램이 전용으로 사용하지는 않지만 L8 모드 TCB가 할당된 CICS 태스크에 의해 호출된 모든 OPENAPI 프로그램 및 OPENAPI TRUE에 의해 사용됩니다. 또한 OPENAPI 프로그램을 호출

하는 애플리케이션이 스프레드세이프 표준으로 코딩되고 CICS에 스프레드세이프로서 정의되는 경우에는 이는 프로그램으로부터 리턴 시에 L8 모드 TCB를 계속해서 실행합니다.

OPENAPI 프로그램은 열린 TCB 환경 실행을 다음과 같은 문제를 일으키는 방식으로 처리하지 않아야 합니다.

- 열린 TCB에서 실행할 수 있는 애플리케이션 로직
- 동일 태스크가 호출한 OPENAPI TRUE
- 열린 TCB를 사용할 수도 있는 미래 태스크
- CICS 관리 코드

사용자의 책임 하에 OPENAPI 프로그램이 다른 비CICS API를 사용하기로 결정하는 경우에는 다음을 인식해야 합니다.

- CICS 서비스를 호출하거나 CICS로 돌아갈 때 OPENAPI 프로그램은 프로그램에 대한 입력에 있는 것처럼 MVS 프로그래밍 환경을 복원하는지 확인해야 합니다. 여기에는 추가된 ESTAE를 취소하는 것에 추가로, 교차 메모리 모드, ASC 모드, 요청 블록(RB) 레벨, 연계 스택 레벨, TCB 디스패치 우선순위가 포함됩니다.
- CICS 태스크 종료 시에 OPENAPI 프로그램은 열린 TCB를 다른 CICS 트랜잭션이 재사용하기에 적합한 상태에 두는지 확인해야 합니다. 특히 태스크를 종료하는 대가로 획득된 모든 비CICS 자원이 자유롭게 되는지 확인해야 합니다. 이러한 자원은 다음을 포함할 수 있습니다.
 - 동적으로 할당된 데이터 세트
 - 열린 ACB 또는 DCB
 - STIMERM 요청
 - MVS 관리 저장영역
 - ENQ 요청
 - 첨부된 서브태스크
 - 로드된 모듈
 - 소유 데이터 공간
 - 추가된 액세스 목록 항목
 - 이름/토큰 쌍
 - 수정된 페이지
 - 보안 설정(TCBSENV가 0으로 설정되어야 함)
- OPENAPI 프로그램은 전체 CICS 운영에 영향을 미치는 다음 MVS 시스템 서비스를 사용하지 않아야 합니다.
 - CHKPT
 - ESPIE

- QEDIT
- SPIE
- STIMER
- TTIMER
- XCTL / XCTLX
- 모든 TSO/E 서비스
- OPENAPI 프로그램은 L8 또는 L9 모드 TCB 하에서 MVS Language Environment 서비스를 사용 중인 Language Environment 프로그램을 호출해서는 안됩니다. L8 및 L9 모드 TCB는 CICS 서비스를 사용하여 Language Environment에 대해 초기화되기 때문입니다.

FORCEQR 시스템 초기화 매개변수 사용

OTE를 사용하기 위해 프로그램이 스레드세이프 상태로 애플리케이션을 실행하면(예: CICS DB2 애플리케이션에서) 하나 이상의 프로그램이 스레드세이프가 아닌 경우 문제점을 일으킬 수 있습니다. 이 경우 FORCEQR 시스템 초기화 매개변수를 사용하여 모든 애플리케이션을 QR TCB로 강제 실행할 수 있습니다.

이는 문제점을 조사하는 동안 작동되지 않는 애플리케이션을 감당할 수 없는 프로덕션 리전에서 유용할 수 있습니다.

이 매개변수의 기본값은 FORCEQR=NO이고, 이는 CICS가 프로그램 자원 정의에서 CONCURRENCY 속성을 존중함을 의미합니다. 임시 조치로서 스레드 정의 프로그램과 연관된 문제점을 조사하고 해결하는 동안 FORCEQR=YES를 설정할 수 있습니다. 프로그램이 OTE 하에서 열린 TCB의 사용을 재개할 준비가 되었을 때 이를 다시 FORCEQR=NO로 변경해야 함을 기억하십시오.

재진입 불가능 프로그램

CICS에서 재진입 불가능 애플리케이션을 실행하는 것을 막을 수 없습니다. 그러나 이러한 애플리케이션은 멀티스레딩 환경에서 일관된 결과를 제공하지 못합니다.

재진입 불가능 애플리케이션 또는 연관된 애플리케이션의 실행으로 수정 가능한 제어 블록 또는 테이블을 사용하려면 해당 자원 정의에서 RELOAD(YES) 옵션을 지정하십시오. RELOAD(YES)를 선택하면 요청할 때마다 프로그램 또는 모듈의 신규 사본이 저장영역에 로드됩니다. 이 옵션은 재진입 불가능 프로그램 또는 테이블에 액세스하는 멀티스레딩 태스크가 각각 자체 프로그램 사본에서 작업하고 CICS 리전에서 실행되고 있는 다른 동시 태스크에 의해 프로그램의 다른 버전에 작성되는 변경사항의 영향을 받지 않도록 합니다.

RELOAD(YES)에 대한 정보는 CICS 자원 정의 안내서에서 PROGRAM 정의 속성의 내용을 참조하십시오.

CICS는 RENT 속성으로 링크 편집되는 모든 프로그램을 CICS 읽기 전용 동적 저장 영역(DSA)에 로드합니다. CICS는 RMODE(24) 프로그램에 RDSA를 사용하고 RMODE(ANY) 프로그램에는 ERDSA를 사용합니다. 기본적으로 이러한 DSA의 저장 영역은 읽기 전용 key-0로 보호 설정된 저장영역에서 할당되어 key-zero 또는 수퍼바이저 상태에서 실행되고 있는 모든 제외 프로그램에서 로드되는 모든 모듈을 보호합니다(CICS가 RENTPGM=NOPROTECT 시스템 초기화 매개변수를 사용하여 초기화하는 경우 읽기 전용 key-0 저장영역을 사용하지 않고 CICS 키 저장영역을 대신 사용합니다).

재진입 불가능 프로그램 또는 모듈을 실행하려는 경우 비읽기 전용 DSA에 로드해야 합니다. SDSA 및 ESDSA는 재진입 불가능 사용자 키 프로그램 및 모듈에 대한 사용자 키 저장영역입니다.

CICS DSA에 대한 자세한 정보는 성능 개선의 CICS 동적 저장영역의 내용을 참조하십시오.

트랜잭션에서 데이터 저장

CICS는 트랜잭션 내 및 사이에 데이터를 저장하는 다양한 기능을 제공합니다. 각 기능은 트랜잭션의 다른 프로그램 및 다른 트랜잭션에 대해 데이터에 남겨지는 사용 가능한 공간, 구현되는 방식 및 오버헤드, 복구 및 큐에 넣기 특성에 따라 다릅니다.

트랜잭션의 수명 동안 존재하는 저장영역 기능은 다음을 포함합니다.

- TWA(Transaction Work Area)
- 사용자 저장영역(SHARED 옵션 없이 실행된 GETMAIN 명령에 의해)
- COMMAREA
- 프로그램 저장영역

이러한 영역은 모두 소스가 동적 저장영역(DSA)이거나, 확장된 동적 저장영역(EDSA)이거나 기준 위 동적 저장영역(GDSA)인 기본 저장영역 기능입니다. 이러한 영역 모두는 복구가 불가능하고 자원 보안 키로 보호할 수 없습니다. 접근성 및 지속 기간에 따라 다르기 때문에 각 저장영역 기능은 서로 다른 일련의 저장영역 요구를 충족시킵니다.

TWA(Transaction Work Area)

TWA(Transaction Work Area)는 트랜잭션이 시작될 때 할당되고 2진 0 값으로 초기화됩니다. 트랜잭션 전 기간 동안 지속되고 트랜잭션의 모든 로컬 프로그램에서 액세스할 수 있습니다.

DPL(Distributed Program Link) 명령으로 링크되는 원격 프로그램에는 클라이언트 트랜잭션의 TWA에 대한 액세스 권한이 없습니다. TWA의 크기는 트랜잭션 자원 정의

의 TWASIZE 옵션에 의해 판별됩니다. 이 크기의 값이 0이 아닌 경우 TWA는 항상 할당됩니다. TWASIZE 판별에 대한 자세한 정보는 *CICS* 자원 정의 안내서의 내용을 참조하십시오.

TWA 사용과 관련된 프로세서 오버헤드는 최소입니다. TWA 액세스에 GETMAIN 명령이 필요하지 않고 한 개의 ADDRESS 명령을 사용하여 처리할 수 있습니다. TASKDATAKEY 옵션은 TWA를 CICS에서 얻는지 또는 사용자 키 저장영역에서 얻는지 여부를 결정합니다. (CICS 키 및 사용자 키 저장영역에 대한 전체 설명은 787 페이지의 『저장영역 제어』의 내용을 참조하십시오.) 트랜잭션 정의의 TASKDATALOC 옵션은 확보한 저장영역이 16MB 라인 위에 있는지 또는 아닌지 여부를 결정합니다.

TWA는 둘 다 크기가 상대적으로 고정되고 트랜잭션 지속 기간 동안 대부분 사용되는 상당히 작은 데이터 저장영역 요구사항 및 상대적으로 큰 요구사항에 적합합니다. TWA는 전체 트랜잭션 동안 존재하기 때문에 TWA 크기가 클 경우 비대화식 트랜잭션에 비해 대화식 트랜잭션에 더 큰 영향을 미칩니다.

사용자 저장영역

사용자 저장영역은 트랜잭션에 있는 모든 프로그램에 사용 가능하지만 LINK 또는 XCTL 명령을 사용하여 프로그램 간에 이를 전달하기 위해서는 약간의 노력이 필요합니다. 해당 크기는 수정되지 않고 트랜잭션이 이를 요구할 때 GETMAIN 명령을 사용하여 구할 수 있고 필요가 없어지는 즉시 리턴됩니다.

사용자 저장영역은 크기에 변수가 있거나 트랜잭션보다 단명한 큰 저장영역 요구사항에 유용합니다.

GETMAIN 명령의 USERDATAKEY 및 CICS DATAKEY 옵션이 트랜잭션 자원 정의의 TASKDATAKEY 옵션을 덮어쓰는 방법에 대한 자세한 정보는 787 페이지의 『저장영역 제어』의 내용을 참조하십시오.

GETMAIN 명령의 SHARED 옵션은 획득된 저장영역이 태스크 종료 뒤에도 유지되게 합니다. 저장영역은 통신 영역에서 한 태스크로부터 동일 터미널에 있는 다음 태스크로 전달될 수 있습니다. 첫 번째 태스크는 RETURN 명령의 COMMAREA 옵션에서 통신 영역의 주소를 리턴합니다. 두 번째 태스크는 ADDRESS 명령의 COMMAREA 옵션에서 주소에 액세스합니다. 저장영역이 공통 저장영역에 있는지 확인하기 위해 GETMAIN 명령의 SHARED 옵션을 사용해야 합니다.

GETMAIN 명령에 관련된 프로세서 오버헤드 때문에 작은 양의 저장영역에는 사용자 저장영역을 사용하지 마십시오. 더 작은 양의 경우에는 TRANSACTION WORK AREA(TWA)를 사용하거나 이를 더 큰 요청으로 그룹화하십시오. GETMAIN 명령으로 획득한 저장영역은 결합된 요청을 사용할 때 더 오래 보관될 수 있지만 프로세서 오버헤드 및 참조 세트 크기 둘 모두가 감소됩니다.

LINK 및 XCTL 명령의 COMMAREA

통신 영역(COMMAREA)은 트랜잭션 내 두 프로그램 사이에서 또는 같은 터미널의 두 트랜잭션 사이에서 정보를 전송하는 데 사용되는 기능입니다.

트랜잭션 간 COMMAREA 사용에 대한 정보는 320 페이지의 『RETURN 명령에서 COMMAREA 사용』의 내용을 참조하십시오.

두 참가 프로그램이 트랜잭션에서 나중에 호출될 수 있는 다른 프로그램에 대해 데이터가 사용 가능하도록 명시적으로 지정하지 않으면 COMMAREA의 정보는 두 참가 프로그램에만 사용 가능합니다.

- 한 프로그램이 다른 프로그램에 링크될 때 COMMAREA는 링크 프로그램이 액세스할 수 있는 모든 데이터 영역이 될 수 있습니다. 이 영역은 해당 프로그램의 작업 중인 저장영역 또는 LINKAGE SECTION에 있습니다. 이 영역에서 링크 프로그램은 데이터를 호출하는 프로그램에 전달하고 해당 프로그램으로부터 결과를 수신할 수 있습니다.
- 프로그램이 다른 프로그램에 제어를 전송할 때(XCTL 명령) CICS는 지정된 COMMAREA를 저장영역의 새 영역에 복사할 수 있습니다. 호출 프로그램 및 그 제어 블록이 제어를 전송한 후에는 더 이상 사용 가능하지 않을 수 있기 때문입니다.

두 가지 경우 모두에서 영역의 주소는 제어를 수신하는 프로그램으로 전달되고 CICS 명령 레벨 인터페이스는 주소 지정성을 설정합니다. 자세한 정보는 364 페이지의 『프로그램 제어』의 내용을 참조하십시오.

XCTL이 사용될 때 CICS는 수신 프로그램의 주소 지정 모드를 준수하는 영역에서 COMMAREA를 작성하여 모든 COMMAREA에 이 명령을 수신하는 프로그램이 주소를 지정할 수 있도록 합니다. 수신인이 AMODE(24)인 경우 COMMAREA는 16MB 라인 아래에 작성되고 수신인이 AMODE(31)인 경우 COMMAREA는 16MB 위 2GB 아래에서 작성됩니다.

COMMAREA는 주소 지정 모드 및 수신 프로그램의 EXECKEY 속성에 따라 필요한 경우 USERKEY 저장영역으로 복사됩니다. EXECKEY에 대한 자세한 정보는 787 페이지의 『저장영역 제어』의 내용을 참조하십시오.

CICS에는 전송할 바이트 수를 줄이도록 디자인된 알고리즘이 있습니다. 이 알고리즘은 전송 전에 COMMAREA에서 이어지는 2진 0을 제거하고 전송 후에 복원합니다. 이러한 알고리즘은 항상 전체 크기 COMMAREA를 보는 애플리케이션에 대해 투명하게 운영합니다.

LINK 명령에서 COMMAREA 사용으로 인한 오버헤드는 최소입니다. CICS가 프로그램에서 사용되는 큰 영역의 저장영역에서 COMMAREA를 작성할 때 XCTL 및 RETURN 명령에서보다 약간 더 높은 정도입니다.

LINK 및 XCTL 명령의 채널

통신 영역을 사용하는 대신에 CICS 프로그램 간에 데이터를 전송하는 보다 근대적인 방식은 채널을 사용하는 것입니다.

채널에는 COMMAREA보다 여러 가지 장점이 있습니다. 330 페이지의 『채널의 이점』의 내용을 참조하십시오. LINK 또는 XCTL 명령에서 채널을 전달하려면 COMMAREA 옵션 대신에 CHANNEL 옵션을 사용하십시오.

채널은 321 페이지의 『채널을 사용하는 프로그램 간 데이터 통신』에 설명되어 있습니다.

프로그램 저장영역

CICS는 프로그램을 사용하는 각 트랜잭션에 대해 CICS 프로그램 변수 영역의 개별 사본을 작성합니다. 이 영역은 프로그램 저장영역이라고 합니다.

이 영역은 COBOL에서 WORKING-STORAGE SECTION, C, C++ 및 PL/I에서 자동 저장영역 그리고 어셈블러 언어에서 DFHEISTG 섹션이라고 합니다. TWA와 같이 이 영역의 크기는 고정되고 GETMAIN 명령을 실행하지 않고도 CICS에서 할당됩니다. EXEC CICS 인터페이스는 주소 지정성을 자동으로 설정합니다. 그러나 TWA와는 달리 이 저장영역은 트랜잭션의 지속 기간이 아닌 프로그램이 실행되는 동안에만 지속됩니다. 이 경우 프로그램 외부에서 필요하지 않고 크기가 작은 또는 큰 경우에는 크기가 고정되며 프로그램의 실행 시간 전체 또는 대부분 동안 필요한 데이터 영역에 유용합니다.

임시 저장영역 큐

임시 저장영역은 데이터를 저장하기 위한 기본 CICS 기능이며 다중 트랜잭션에 대해 사용 가능해야 합니다. 임시 저장영역에 있는 데이터 항목은 임시 저장영역 큐에서 보관됩니다. 이러한 항목은 시작 태스크 또는 다른 태스크에서 임시 저장영역 큐에 지정된 기호 이름을 사용하여 검색할 수 있습니다.

여러 항목을 포함하고 있는 임시 저장영역은 작은 데이터 세트로 간주할 수 있습니다. 큐에 있는 특정 항목(논리 레코드)은 상대 위치 번호로 참조됩니다. 항목은 항목 번호에 따라 순차적으로 또는 직접 처리할 수 있습니다. 큐에 단일 항목만 포함된 경우 이름 지정된 스크래치 패드 영역으로 간주할 수 있습니다.

임시 저장영역 큐는 최대 16자의 기호 이름으로 식별됩니다. 중복된 이름으로 인해 발생할 수 있는 충돌을 피하려면 이름 지정 규칙을 설정하십시오. 예를 들어, 운영자 ID 또는 터미널 ID를 각 프로그래머 제공 기호 이름에 대한 접미부로 사용할 수 있습니다. 임시 저장영역 큐가 작성될 때 이름을 지정할 수 있다는 점은 저장된 데이터에 대

한 강력한 양식의 직접 액세스를 제공합니다. 큐 이름에 터미널 이름 또는 레코드 키를 포함시켜 터미널 및 데이터 세트 레코드와 같은 자원의 스크래치 패드 영역에 액세스할 수 있습니다.

태스크에서 태스크로 데이터를 전달하는 다른 메소드에 비해 임시 저장영역 큐는 더 많은 프로세서 사용을 필요로 할 수 있습니다. 임시 저장영역 큐와의 모든 상호작용에 EXEC CICS 명령을 사용하면 CICS가 그 내부 색인을 사용하여 데이터를 찾거나 삽입해야 합니다. 따라서 기본 임시 저장영역에서의 프로세서 사용량은 CWA 또는 TCTUA에서 사용할 때보다 큼니다. 일반적으로 보조 저장영역에서도 데이터 세트 I/O가 있습니다. 공유 임시 저장영역 풀에는 임시 저장영역 서버가 필요하며 애플리케이션은 데이터를 검색하기 위해 결합 기능에 액세스해야 합니다.

그러나 임시 저장영역 큐에는 데이터를 전달하는 다른 메소드에 비해 많은 장점이 있습니다. 필요할 때까지 임시 저장영역을 할당하지 않아도 됩니다. 임시 저장영역은 필요한 동안만 보관하고 항목 크기는 항목을 작성하는 명령을 실행할 때까지 정해지지 않습니다. 따라서 임시 저장영역 큐는 상대적으로 높은 볼륨의 데이터와 길이 또는 지속 기간에 따라 달라지는 데이터에 상대적으로 유리한 선택입니다. 임시 저장영역 큐에 자원 보호도 사용할 수 있습니다.

임시 저장영역 큐는 시작 태스크나 다른 태스크에 의해 삭제될 때까지 초기 또는 콜드 스타트 시까지 저장영역에 남아 있습니다. 애플리케이션은 **DELETEQ TS** 명령을 사용하여 유효 수명이 끝나면 임시 저장영역 큐를 삭제할 수 있습니다. 애플리케이션이 항상 임시 저장영역 큐를 삭제할 수 있는 것은 아닌 경우 자동 삭제 설정을 고려해 보십시오. CICS가 최근에 액세스되지 않은 기본 저장영역 또는 보조 저장영역의 복구 불가능 임시 저장영역 큐를 자동으로 삭제하게 할 수 있습니다. TSMODEL 자원 정의의 만료 간격은 자동 삭제를 제어합니다.

임시 저장영역 큐의 위치 선택

CICS 리전의 임시 저장영역은 기본 저장영역, 보조 저장영역 또는 z/OS 결합 기능의 공유 임시 저장영역 풀에 있을 수 있습니다. 여러 임시 저장영역 위치 및 각 위치에서 임시 저장영역 큐에 사용 가능한 기능에 대한 개요는 성능 개선의 CICS 임시 저장영역 개요의 내용을 참조하십시오.

애플리케이션은 **WRITEQ TS** 명령을 사용하여 데이터의 첫 번째 항목을 임시 저장영역 큐에 씁니다. 이 명령은 임시 저장영역 큐의 기호 이름을 지정합니다. 이 이름이 설치된 TSMODEL 자원 정의와 일치하는 경우 CICS는 TSMODEL 자원 정의로 지정된 임시 저장영역 위치에서 임시 저장영역 큐를 작성합니다. 일치하는 TSMODEL 자원 정의가 없는 경우 CICS는 애플리케이션이 **WRITEQ TS** 명령에서 지정하는 임시 저장영역 위치를 사용합니다. 기본 위치는 보조 임시 저장영역입니다.

임시 저장영역 큐가 있는 위치를 선택하려면 다음과 같은 요소를 고려하십시오.

사용 수명 및 빈도

- 기본 임시 저장영역은 단기간 필요하고 액세스 빈도가 높은 임시 저장영역 큐에 적합합니다.
- 보조 임시 저장영역 및 공유 임시 저장영역 풀은 수명 기간이 상대적으로 길고 액세스 빈도가 낮은 임시 저장영역 큐에 적합합니다. 1초 수명 절사점을 사용하여 큐가 기본 저장영역으로 이동할지 또는 보조나 결합 기능 저장영역으로 이동할지 결정할 수 있습니다.

복구

- 기본 저장영역에 있는 임시 저장영역 큐는 복구 기능으로 정의할 수 없습니다.
- 보조 저장영역에 있는 임시 저장영역 큐는 복구 기능으로 정의할 수 있습니다.
- CICS 복구는 공유 임시 저장영역 풀의 임시 저장영역 큐에 사용할 수 없습니다. 그러나 결합 기능은 CICS 재시작의 영향을 받지 않기 때문에 공유 임시 저장영역 풀의 임시 저장영역 큐는 지속적인 것으로 간주할 수 있습니다.

자동 삭제

CICS가 해당 임시 저장영역 모델에 만료 간격을 추가하여 더 이상 필요하지 않은 해당 임시 저장영역 큐를 자동으로 삭제하도록 지정할 수 있습니다.

- 기본 저장영역에서 임시 저장영역 큐에 만료 간격을 설정할 수 있습니다.
- 보조 임시 저장영역에서 복구 불가능 큐에 만료 간격을 설정할 수 있습니다. 복구 가능 큐는 자동으로 삭제할 수 없습니다.
- 공유 임시 저장영역 풀에 있는 임시 저장영역 큐는 자동으로 삭제할 수 없습니다.

저장영역 유형

기본 임시 저장영역은 CICS 리전의 64비트 저장영역에 있습니다. 복구 가능 임시 저장영역이 필요하지 않은 경우 애플리케이션이 기본 임시 저장영역을 사용하도록 지정할 수 있습니다. 결과적으로 31비트 저장영역에서 공간에 대한 압박이 줄고 디스크에 데이터를 쓰기 위한 I/O 활동도 줄어듭니다.

임시 저장영역 큐에 대한 잠금 및 대기

CICS 임시 저장영역 도메인은 여러 요청을 동시에 처리할 수 있지만 같은 임시 저장영역 큐에 대한 요청을 일련화합니다. 이 큐는 각 요청 기간 동안 잠깁니다.

한 번에 한 개의 트랜잭션만 복구 가능 임시 저장영역 큐를 작성하거나 삭제할 수 있습니다. 큐를 복구 가능한 상태로 설정하기로 선택한 경우 큐에 넣기 가능성에 대해서도 기억해 두십시오.

태스크가 임시 저장영역에 쓰려고 시도하고 사용 가능한 공간이 없는 경우 CICS는 일반적으로 이 태스크를 일시중단합니다. 이 태스크는 **HANDLE CONDITION NOSPACE** 명령 또는 **WRITEQ TS** 명령에서 RESP 또는 NOHANDLE 옵션을 사용하여 제어를 다시 획득할 수 있습니다. 일시중단된 경우 이 태스크는 기본 저장영역 또는 VSAM 데이터 세트에서 일부 다른 태스크가 필요한 공간을 사용 가능하게 할 때까지 재개되지 않습니다. 이 상황에서 특히, 대기 태스크가 독점 사용 자원을 소유하고 있어서 해당 자원을 필요로 하는 다른 모든 태스크도 대기해야 하는 경우 무응답 지연이 발생할 수 있습니다.

관련 참조

- ☞ 참조 -> 애플리케이션 개발에서 WRITEQ TS
- ☞ 참조 -> 애플리케이션 개발에서 READQ TS
- ☞ 참조 -> 애플리케이션 개발에서 DELETEQ TS
- ☞ 참조 -> 시스템 정의에서 TSMODEL 자원

리전 내부 트랜지언트 데이터

리전 내부 트랜지언트 데이터는 CICS가 기본 저장영역에서 유지보수하는 색인과 함께 단일 데이터 세트에 함께 보관되는 데이터의 큐로 구성됩니다.

리전 내부 트랜지언트 데이터는 보조 임시 저장영역과 같은 일부 특성을 가집니다(리전 외부 트랜지언트 데이터에 대한 정보는 313 페이지의 『효율적 순차 데이터 세트 액세스』 참조).

보조 임시 저장영역의 사용 용도와 같은 다양한 용도로 트랜지언트 데이터를 사용할 수 있지만 몇 가지 중요한 차이점이 있습니다.

- 트랜지언트 데이터는 임시 저장영역과 같은 동적 특성을 가지지 않습니다. 임시 저장영역 큐와 달리 트랜지언트 데이터 큐는 애플리케이션이 데이터를 쓸 때 작성할 수 없습니다. 그러나 트랜지언트 데이터 큐는 CICS가 실행되는 동안 RDO를 사용하여 정의 및 설치할 수 있습니다.
- 트랜지언트 데이터 큐는 순차적으로 읽어야 합니다. 각 항목은 한 번만 읽을 수 있습니다. 트랜잭션이 항목을 읽으면 해당 항목은 큐에서 제거되고 다른 트랜잭션이 사용할 수 없습니다. 반대로 임시 저장영역 큐에 있는 항목은 순차적으로 또는 직접(항목 번호별로) 읽을 수 있습니다. 이러한 항목은 임의의 횟수로 읽을 수 있고 전체 큐가 영구 제거될 때까지 큐에서 제거되지 않습니다.

이러한 두 가지 특성으로 인해 트랜지언트 데이터는 스크래치 패드 데이터로는 적합하지 않지만 큐 처리된 데이터(예: 감사 추적 및 인쇄될 출력)로는 적합합니다. 실제로 순차적으로 한 번만 읽는 데이터로는 트랜지언트 데이터가 임시 저장영역보다 선호됩니다.

- 임시 저장영역 큐에 있는 항목은 변경할 수 있고 트랜지언트 데이터의 항목은 변경할 수 없습니다.
- 트랜지언트 데이터 큐는 항상 데이터 세트에 기록됩니다. (기본 임시 저장영역에 해당하는 트랜지언트 데이터 양식은 없습니다.)
- 큐에 항목을 써서 특정 트랜잭션을 시작할 수 있도록(예: 큐 처리) 트랜지언트 데이터 큐를 정의할 수 있습니다. 임시 저장영역에서 유사한 기능을 수행하기 위해 START 명령을 사용할 수 있기는 하지만 임시 저장영역에는 이 『트리거』 메커니즘에 해당하는 항목이 없습니다.
- 트랜지언트 데이터에는 임시 저장영역보다 많은 복구 옵션이 있습니다. 트랜지언트 데이터 큐는 물리적으로 또는 논리적으로 복구 가능할 수 있습니다.
- 리전 내부 및 리전 외부 트랜지언트 데이터에 대한 명령이 동일하기 때문에 두 가지 유형의 데이터 세트 사이에서 전환할 수 있습니다. 이를 수행하려면 트랜지언트 데이터 큐 정의만 변경하고 애플리케이션 자체는 변경하지 마십시오. 임시 저장영역에는 이러한 유형의 해당 기능이 없습니다.

GETMAIN SHARED 명령

GETMAIN 또는 GETMAIN64 명령의 SHARED 옵션을 사용하여 획득한 저장영역은 획득 태스크가 종료될 때 릴리스되지 않습니다. 이를 사용하면 한 태스크는 다른 태스크가 사용할 수 있도록 저장영역에 데이터를 남길 수 있습니다.

저장영역은 FREEMAIN 또는 FREEMAIN64 명령이 실행될 때까지는 획득 태스크나 또 다른 태스크에 의해서 릴리스되지 않습니다.

사용자 고유의 데이터 세트

사용자 고유의 데이터 세트를 사용하여 트랜잭션 간 데이터를 저장할 수도 있습니다. 이 방법은 아마 처리된 지시사항, 버퍼, 제어 블록 및 사용자 프로그래밍 요구사항의 관점에서 최대의 오버헤드를 가지지만 추가 기능과 유연성을 제공합니다.

데이터 세트를 복구 가능 자원으로 정의할 수 있을 뿐만 아니라 정방향 복구를 위해 이에 대한 변경사항을 로깅할 수 있습니다. 데이터 세트의 문자열 수를 지정할 수 있을 뿐만 아니라 임시 저장영역 및 트랜지언트 데이터 세트에서 액세스 충돌을 방지하기 위해 자원 보안을 사용할 수 있습니다.

CICS에 전달된 영역의 길이

CICS 명령이 LENGTH 옵션을 포함하는 경우 일반적으로 이 명령은 부호 있는 반자 2진 값 길이를 승인합니다. 부호 있는 반자 2진 값을 사용하면 길이에 이론적 32KB 상한이 지정됩니다. 실제로 이러한 한계는 더 낮고 각 명령에 따라 다릅니다.

한계는 데이터 세트 정의, 복구 가능성 요구사항, 버퍼 크기 및 로컬 네트워크 특성에 따라 다릅니다.

LENGTH 옵션

COBOL, C, C++, PL/I 및 어셈블러 언어에서 변환기는 길이를 처리합니다.

LENGTH 옵션을 지정해야 하는 경우에 대한 세부사항을 포함한 프로그래밍 정보는 애플리케이션 개발 참조서의 내용을 참조하십시오. 가능한 경우 CICS 명령에서 지정되는 길이가 24KB를 초과하지 않도록 하십시오. 자세한 정보는 참조 -> 애플리케이션 개발에서 CICS 명령의 LENGTH 옵션의 내용을 참조하십시오.

다수의 명령에는 애플리케이션과 CICS 간 데이터 전송이 포함됩니다. 모든 경우에 애플리케이션이 전송할 데이터 길이를 제공해야 합니다.

EXEC CICS LINK 명령을 사용하여 COMMAREA에서 데이터를 전달할 때 COMMAREA에서 전달되고 있는 데이터 길이와 일치하는 LENGTH 값을 지정해야 합니다. LENGTH에 0(영)을 지정하지 마십시오. 0을 지정할 경우 결과로 나타나는 동작을 예측할 수 없고 **EXEC CICS LINK** 명령이 실패할 수 있기 때문입니다. 데이터 전달에 COMMAREA를 사용하는 경우 링크되는 프로그램에서는 태스크의 EIB에 있는 EIBCALEN 필드가 프로그램이 예상하는 값과 일치하는지 확인해야 합니다. 불일치할 경우 저장영역 위반 또는 시스템 장애가 발생할 수 있습니다. 자세한 정보는 367 페이지의 『COMMAREA』의 내용을 참조하십시오.

대부분의 경우 LENGTH 옵션을 지정해야 합니다. SET 옵션이 사용될 경우 각 명령의 구문 및 연관된 옵션은 이 규칙이 적용되는지 여부를 표시합니다.

WAIT EXTERNAL 명령 및 다수의 **QUERY SECURITY** 명령에는 자원 상태 또는 정의를 제공하는 옵션이 있습니다. CICS는 이러한 옵션과 연관된 값 즉, 이름, CICS 값 데이터 영역을 제공합니다. 옵션은 소괄호 안에 용어 *cvda*가 포함된 명령의 구문에 표시됩니다. CVDA에 대한 프로그래밍 정보는 참조 -> 애플리케이션 개발에서 CICS-값 데이터 영역(CVDA)의 내용을 참조하십시오.

저널 명령의 경우 LENGTH 및 PFXLENG 값의 합계에 제한사항이 적용됩니다. 418 페이지의 『저널링』의 내용을 참조하십시오.

저널 레코드

저널 레코드의 경우 저널 버퍼 크기는 64KB 미만의 한계를 부과할 수 있습니다. 이 한계는 LENGTH 및 PFXLENG 값의 합계에 적용됩니다.

데이터 세트 정의

임시 저장영역, 트랜지언트 데이터 및 파일 제어에서 데이터 세트 정의는 24KB 미만의 한계를 부과할 수 있습니다.

데이터 세트 작성에 대한 정보는 구성에서 데이터 세트 정의의 내용을 참조하십시오. 파일의 자원 정의에 대한 정보는 참조 -> 시스템 정의에서 FILE 자원의 내용을 참조하십시오.

권장사항

모든 시스템의 모든 명령에서 32,000바이트가 LENGTH 스펙의 적절한 작업 한계입니다. 사용자 지정 레코드 및 버퍼 크기에 따라 이 한계가 오류를 발생시키거나 애플리케이션에 제한조건을 지정할 가능성은 거의 없습니다.

참고: LENGTH 옵션의 값은 명령에서 처리하는 데이터 영역의 길이를 초과할 수 없습니다.

오류 최소화

다음 기술을 사용하면 애플리케이션에 오류가 발생하지 않도록 하는 데 도움이 됩니다. 다음 제안사항 중 일부는 프로그래밍 외에 조작 및 시스템에도 적용됩니다.

자체적으로 완전하게 실행되는 두 애플리케이션 시스템이 함께 실행될 때 성능이 저하되고 『락아웃』 또는 대기해야 하는 상황이 빈번히 발생합니다. 각 시스템의 범위가 충분히 정의되지 않았습니다.

올바르게 디자인된 애플리케이션 시스템에서 중요한 점은 다음과 같습니다.

- 모든 레벨에서 각 함수가 잘 정의된 입력 및 출력으로 명확하게 정의됩니다.
- 시스템이 사용하는 자원이 적절하게 정의됩니다.
- 다른 시스템과의 상호작용을 알 수 있습니다.

CICS를 애플리케이션 오류로부터 보호하기

애플리케이션에서 오류를 최소화하기 위해 이러한 저장영역 도구와 기술을 사용하십시오.

이 태스크 정보

- 저장영역 보호 기능을 사용하여 애플리케이션이 CICS 코드와 제어 블록을 겹쳐쓰지 못하도록 막을 수 있습니다. CICS 시스템 초기화 매개변수를 사용하여 이 기능을 사용할지 여부를 선택할 수 있습니다. 이 기능에 대한 자세한 정보는 성능 개선의 저장영역 보호의 내용을 참조하십시오.
- GETMAIN 명령의 사용 등과 같이 기술이 초래할 수 있는 문제점을 피하는 표준을 사용하는 것을 고려해 보십시오.

애플리케이션 테스트

애플리케이션 테스트에 적용되는 몇몇 일반 규칙이 나열되어 있습니다.

이 테스트 정보

- 프로덕션 CICS 시스템을 테스트하지 마십시오. 라이브 데이터베이스에 영향을 미치지 않고 오류를 고립시킬 수 있는 테스트 시스템을 사용하십시오.
- 가능한 경우, 테스트를 하려면 애플리케이션 개발자 이외의 다른 사람을 이용하십시오.
- 테스트에 사용하는 데이터를 문서화하십시오.
- 애플리케이션을 여러 차례 테스트하십시오. 197 페이지의 제 11 장 『애플리케이션 테스트』의 내용을 참조하십시오.
- 초기 테스트에 CEDF 트랜잭션을 사용하십시오. 201 페이지의 제 12 장 『진단 기능 실행(EDF)』의 내용을 참조하십시오.
- 단일 사용자 환경에서 발견되지 않은 문제점을 찾기 위해 스트레스 또는 볼륨 테스트를 사용하십시오. 이를 위한 좋은 도구는 Teleprocessing Network Simulator(TPNS, 라이선스가 있는 프로그램 번호 5740-XT4)입니다.

TPNS는 애플리케이션을 설치하기 전에 테스트하고 평가할 수 있게 해주는 통신 테스트 패키지입니다. TPNS를 사용자 조직의 애플리케이션에서 사용되는 테스트 로직, 사용자 엑시트 루틴, 메시지 로깅, 데이터 암호화 및 디바이스 종속성을 위해 사용할 수 있습니다. 이는 시스템 성능과 응답 시간, 스트레스 테스트를 조사하고 TP 네트워크 디자인을 평가할 때 유용합니다. 자세한 정보는 *TPNS* 일반 정보 매뉴얼을 참조하십시오.

- 애플리케이션이 올바른 데이터와 올바르지 않은 데이터를 처리할 수 있는지 여부를 테스트하십시오.
- 관련된 데이터베이스의 전체 사본을 테스트하십시오.
- 다중리전 운영을 사용하는 것을 고려하십시오. 다중리전 운영의 내용을 참조하십시오.
- 애플리케이션을 프로덕션 시스템으로 이동하기 전에 프로덕션 데이터베이스 사본에 마지막 테스트 세트를 실행하여 오류를 찾으십시오.

특히 파괴된 저장영역 체인을 찾으십시오.

어셈블러 언어 프로그램(데이터 영역을 제대로 수정하지 않은 경우) 또 다른 트랜잭션에 영향을 미치거나 이상종료하는 무언가를 변경할 수 있으므로 이를 식별하기가 더 어렵습니다.

문제점 해결에 대한 자세한 정보는 문제점 해결에서 문제점 판별에 대한 접근방식의 내용을 참조하십시오.

비터미널 트랜잭션 보안

CICS는 비터미널 트랜잭션에 사용되는 자원을 무단 사용으로부터 보호할 수 있습니다.

이러한 트랜잭션의 유형은 다음과 같습니다.

- START 명령으로 시작되고 터미널 ID는 지정하지 않은 트랜잭션입니다.
- 터미널 없이 시작되는 트랜잭션으로, 리전 내부 트랜지언트 데이터 큐에 대해 트리거 레벨에 도달하면 발생합니다.
- CICS 내부 트랜잭션(CPLT)으로, CICS 시동 시 실행되며 프로그램 목록 테이블(PLT)에서 지정된 프로그램을 실행합니다. 이 트랜잭션은 PLT의 첫 번째와 두 번째 단계 둘 다 실행합니다.

또한 CICS 종료 중에 실행되는 PLT에 대해 자원 보안 확인을 수행할 수 있습니다. PLT 종료 프로그램은 종료를 요청하는 트랜잭션의 일부로 실행되기 때문에 종료 명령을 발행하는 사용자의 권한 하에 실행됩니다.

START 명령은 START 명령으로 시작된 비터미널 트랜잭션의 보안을 처리합니다.

다른 사용자에게 대한 트랜잭션을 첨부하거나 그 트랜잭션이 첨부될 수 있게 하는 권한이 부여된 대리 사용자 또는 해당 트랜잭션에 대한 모든 자원 액세스 권한을 상속받은 사용자는 사용자를 대리할 수 있습니다.

CICS는 환경에 따라 단일 START 명령에서 최대 세 명의 대리 사용자 보안 확인을 실행할 수 있습니다.

1. USERID가 지정되는 경우 START 명령을 실행하는 트랜잭션의 사용자 ID
2. START 명령을 실행하는 트랜잭션이 CEDF 이중 화면 모드에서 실행 중인 경우 CEDF 트랜잭션의 사용자 ID
3. START 명령이 다른 CICS 시스템에 제공되는 함수이고 링크 보안이 유효한 경우 원격 시스템의 CICS 리전 사용자 ID

트랜잭션이 첨부되기 전에 이러한 사용자 ID 각각에 대해 필요에 따라 개별 대리 사용자 보안 확인이 수행됩니다.

USERID 옵션, USERIDERR 조건 및 INVREQ 및 NOTAUTH 조건에 대한 프로그래밍 정보는 *CICS 애플리케이션 프로그래밍 참조서*의 내용을 참조하십시오.

성능 향상을 위한 디자인

애플리케이션의 디자인을 변경하여 성능과 효율성을 개선할 수 있습니다.

- 304 페이지의 『프로그램 크기』
- 305 페이지의 『가상 저장영역』
- 309 페이지의 『자원 독점 제어』

- 310 페이지의 『운영 제어』
- 311 페이지의 『운영 체제 대기』
- 311 페이지의 『NOSUSPEND 옵션』
- 313 페이지의 『효율적 순차 데이터 세트 액세스』
- 314 페이지의 『효율적 로깅』

애플리케이션 디자인의 다른 측면은 267 페이지의 제 18 장 『애플리케이션 디자인』에 설명되어 있습니다.

특정 상황에서 적용되는 성능 문제점이 있으면 영향이 해당 상황에서만 적용되도록 변경사항을 분리하십시오. 문제점을 수정하고 변경사항을 테스트한 후에는 성능에 미치는 영향이 가장 두드러진 가장 일반적으로 사용되는 프로그램과 트랜잭션에서 사용하십시오.

프로그램 크기

소형 프로그램에 대한 조기 강조는 CICS 프로그래머가 프로그램을 가능한 작은 단위로 분할하고 XCTL 명령을 사용하여 제어를 전송하거나 LINK 명령을 사용하여 링크를 전송하도록 유도했습니다.

그러나 현재 시스템에서는 각 전송에 대한 CICS 처리 오버헤드가 있고 LINK 명령의 경우에는 레지스터 저장 영역(RSA)에 대한 저장영역 오버헤드도 있기 때문에 프로그램을 그렇게 작은 단위로 분할하는 것이 항상 좋은 것만은 아닙니다.

순차적으로 처리되는 보통 크기의 코드 블록의 경우, 인라인 코드가 가장 효율적입니다. 이 규칙의 예외는 다음 코드 블록입니다.

- 상당히 길고 애플리케이션의 여러 위치에서 독립적으로 사용되는 코드 블록
- 자주 변경되는 코드 블록(이 경우, 용이한 유지보수로 LINK 또는 XCTL 명령의 오버헤드 밸런스를 맞춤)
- 드물게 사용되는 코드 블록(예: 오류 복구 논리 및 흔하지 않은 데이터 조합을 처리하기 위한 코드)

이러한 이유 중 하나로 서브루틴으로 작성해야 하는 코드 블록이 있는 경우, 성능 관점에서 이를 처리하는 최상의 방법은 호출 프로그램 내에서 처리완료된 서브루틴(예: COBOL에서 PERFORM 명령을 통해 처리된 코드)을 사용하는 것입니다. 다른 프로그램이 이를 필요로 하는 경우 별도의 프로그램이어야 합니다. CALL문(매크로)을 사용하여 별도의 프로그램을 호출하거나 XCTL 또는 LINK 명령을 사용하여 별도로 보존하고 처리할 수 있습니다. CICS 서비스가 호출되지 않기 때문에 CALL의 경우 실행 오버헤드가 가장 적습니다. 예를 들어, 호출되는 프로그램의 작업 중인 저장영역은 복사되지 않습니다. 그러나 호출된 프로그램을 호출 프로그램에 링크해야 하므로 특수 COBOL, C 또는 PL/I 기능을 사용하지 않는 한 이를 필요로 하는 다른 프로그램이

공유할 수 없습니다. 호출된 서브루틴은 이를 호출(CALL)하는 각 프로그램의 일부로 로드되므로 저장영역을 더 많이 사용합니다. 따라서 프로그램을 사용하는 후속 트랜잭션은 호출된 프로그램에 대해 수행된 변경사항이 작업 중인 저장영역에 있을 수도 있고 없을 수도 있습니다. 이는 전적으로 CICS가 프로그램의 새 사본을 저장영역에 로드했는지 여부에 따라 다릅니다.

XCTL 및 LINK 명령을 사용하면 오버헤드가 가장 많지만 유연성도 최고입니다. XCTL 명령의 경우보다 LINK 명령의 경우 프로세서와 저장영역 요구사항이 둘 다 훨씬 많습니다. 따라서 호출 프로그램이 호출된 프로그램이 처리된 후 리턴된 제어를 갖지 않아도 되면 XCTL 명령을 사용해야 합니다.

애플리케이션으로 인한 로드 모듈은 최대 2GB의 기본 저장영역을 차지할 수 있습니다. 분명히 초대형 로드 모듈 로드 및 초기화와 연관된 추가 비용이 있으며 CICS 동적 저장영역 한계(EDSA)를 그에 상응하여 높게 설정해야 합니다. 가능하면 대형 로드 모듈 사용을 피해야 합니다. 그러나 객체 지향 언어(예: C++)로 작성된 대형 애플리케이션은 크기면에서 16M을 쉽게 초과할 수 있습니다. 단일 DLL에 바인드된 C++ 클래스 경험은 두 개 이상의 DLL에서 단일 DLL을 재구성하면 이 클래스 성능이 저하된다는 것입니다. 이는 다중 DLL 간의 함수 참조를 해석하는 데 필요한 처리 때문입니다.

프로그램이 동적 저장영역(DSA)에서 사용 가능한 저장영역을 모두 차지하는 경우 이상 종료 코드 APCG를 수신할 수 있습니다.

가상 저장영역

신중한 디자인을 통해 사용되는 가상 저장영역의 양을 최소화하고 애플리케이션의 오버헤드를 줄일 수 있습니다.

진정한 대화식 CICS 태스크는 각 쓰기 이후에 터미널 읽기 요청을 실행하여 몇몇 또는 다수의 상호작용에 대해 터미널 사용자와 대화하는 태스크입니다(예를 들어, SEND 명령 뒤에 RECEIVE 명령 또는 CONVERSE 명령 사용). 이는 태스크가 터미널 사용으로부터 다음 입력을 기다리기 위해 확장 수명의 대부분을 소비함을 의미합니다.

CICS 태스크에는 수명 전반에서 가상 저장영역이 필요하며, 대화식 태스크에서는 태스크가 터미널 I/O를 기다리고 있는 기간 동안 이 가상 저장영역 중 일부가 전달됩니다. 관련된 저장영역은 TCA 및 연관된 태스크 제어 블록(EIS 또는 EIB 포함)과 터미널 읽기 요청이 실행될 때 사용 중인 모든 프로그램에 필요한 저장영역을 포함합니다. 이 태스크의 해당 프로그램 사용과 연관된 작업 영역(예: COBOL 작업 중인 저장영역 사본)도 포함합니다.

신중한 디자인을 통해 대화 기간 동안 때로는 하나의 초소형 프로그램만 보유하도록 배열할 수 있습니다. 다른 사용자가 필요한 저장영역을 공유할 수 있습니다. 나머지 가상 저장영역 요구사항에 해당 코드를 사용하는 동시 대화식 세션 수를 곱해야 합니다.

대조적으로, 의사 대화식 태스크 순서에서는 메시지 쌍을 처리하느라 소비한 기간 동안 만 거의 모든 가상 저장영역이 필요합니다. 일반적으로 1분 중 1 - 3초의 기간이 걸립니다(나머지는 운영자 입력을 기다리는 시간이 됨). 따라서 다중 동시 사용자에 대한 전체 요구사항은 대화식 태스크에 필요한 요구사항의 약 5%입니다. 그러나 각 태스크에서 다음 태스크로 전달되는 데이터 영역을 감안해야 합니다. 이는 몇 바이트의 COMMAREA 또는 임시 저장영역의 큰 영역일 수 있습니다. 후자인 경우, 일반적으로 기본 저장영역이 아닌 디스크에서 임시 저장영역을 사용하는 것이 좋지만 이는 의사 대화식 설정에서 추가 임시 저장영역 I/O 오버헤드 증가를 의미합니다(대화식 처리에는 필요하지 않음).

대화식 애플리케이션에 필요한 추가 가상 저장영역은 일반적으로 그에 상응하여 상당한 양의 실제 저장영역이 필요함을 의미합니다. 저장영역을 제어하는 데 필요한 페이징은 추가 오버헤드와 가상 저장영역을 수반합니다. 페이징의 부작용은 트랜잭션 비율이 올라갈수록 증가하므로 가능한 페이징 사용을 최소화해야 합니다. 이렇게 하는 방법에 대한 정보는 『페이징 영향 축소』의 내용을 참조하십시오.

페이징 영향 축소

페이징 영향을 줄이는 것은 가상 저장영역 환경에서 CICS가 사용하는 기술입니다. 이 환경에서 프로그래밍의 주요 목적은 페이지 결함을 줄이는 것입니다. 프로그램이 실제 저장영역에 상주하지 않는 명령어나 데이터를 참조하는 경우 페이지 결함이 발생합니다. 이 경우 참조된 명령어나 데이터를 포함하는 가상 저장영역의 페이지를 실제 저장영역에 페이징해야 합니다. 페이징이 많이 필요할수록 전체 시스템 성능이 저하됩니다.

이 태스크 정보

애플리케이션이 운영 체제와 직접 통신할 수 있긴 하지만 이러한 조치의 결과는 예측할 수 없으며 성능을 저하시킬 수 있습니다.

가상 저장영역 환경에서 실행될 애플리케이션을 작성하려면 다음 용어를 이해해야 합니다.

참조 국소성

애플리케이션 실행 중에 비교적 오랜 기간 동안 프로그램의 총 페이지 수와 비교하여 비교적 적은 수의 페이지 내 명령어 및 데이터에 대한 일관된 참조입니다.

작업 세트

주어진 기간 동안 필요한 프로그램의 페이지 수 및 조합입니다.

참조 세트

쓸모 없는 데이터를 검색하는 중간 저장영역 참조 없이 필요한 페이지에 대한 직접 참조입니다.

참조 국소성:

처리된 지시사항과 프로그램에서 상대적으로 적은 수의 페이지 내에서(4096바이트 세그먼트) 사용된 데이터를 유지하십시오. 프로그램에서 이 품질은 참조 국소성으로 알려져 있습니다.

이 태스크 정보

다음을 수행하여 이를 수행할 수 있습니다.

- 프로그램의 실행을 가능한 선형으로 만듭니다.
- 일반 실행 순서에서 사용하는 서브루틴을 가능한 이를 호출한 코드와 가깝게 유지합니다.
- 몇몇 장소에서만 호출되는 짧은 서브루틴이 있는 경우에는 코드를 반복해야 하더라도 코드를 인라인에 둡니다.
- 오류 처리와 기타 자주 처리되지 않는 코드를 프로그램의 기본 플로우로부터 분리합니다.
- 이러한 코드가 사용하는 데이터를 일반 실행에 사용되는 데이터로부터 분리합니다.
- 데이터 항목을 이들이 언급되는 순서로 정의합니다(특히 배열과 기타 거대 구조).
- 데이터 구조의 요소를 이들이 언급되는 대략적인 순서로 정의합니다. 예를 들어, PL/I에서는 행 행의 모든 요소가 저장된 후 그 다음 요소가 저장되는 등의 순서입니다. 열이 아니라 행별로 처리할 수 있도록 배열을 정의합니다.
- 데이터를 처음 사용되는 것과 가능한 가깝게 초기화합니다.
- 서브루틴 호출로 확장되므로 COBOL 변수 MOVE 운영을 피합니다.
- 가능한 몇몇 GETMAIN 명령만을 실행합니다. 요구사항의 지속 기간이 크게 다르지 않는 한 프로그램이 여러 개의 작은 명령을 수행하는 것보다 요구사항을 합쳐서 하나의 GETMAIN 명령을 실행하는 것이 낫습니다.
- 가능하면 GETMAIN 명령에서 INITIMG 옵션의 사용을 피하십시오. 이는 동일 영역에 대한 다른 참조가 있을 때 획득한 저장영역에 대한 즉각적인 페이지 참조를 야기시킵니다. 그렇지 않으면 프로그램에서 한참 나중에게까지 발생하지 않을 것입니다.

참고: 이전 제안사항 중 일부는 가상 저장영역 환경에서 실행 속도가 아니라 코드의 가독성과 유지보수성을 목표로 하는 경우 설치의 프로그래밍 표준과 충돌할 수 있습니다. 몇몇 구조화된 프로그래밍 방법, 특히 모듈식 프로그래밍 기술은 COBOL(및 C, PL/I 및 어셈블러 언어로 된 동등한 프로그래밍 기술)로 PERFORM 명령을 광범위하게 사용하여 프로그램의 구조를 명확하게 만듭니다. 이는 비구조화된 프로그램에서 발견되는 것보다 순차 처리에 대한 더 많은 예외를 발생시킬 수 있습니다. 그럼에도 불구하고 구조화된 코드와 연관된 생산성이 크므로 참조 국소성의 가능한 손실을 만회하고도 남을 가치가 있습니다.

작업 세트:

작업 세트는 지정된 기간 동안 필요한 프로그램의 페이지의 수와 조합입니다.

이 태스크 정보

작업 세트의 크기를 최소화하려면 프로그램이 지정된 기간에 참조하는 저장영역의 양은 가능한 작아야 합니다. 다음을 수행하여 이를 수행할 수 있습니다.

- 모듈별 프로그램을 쓰고 모듈을 참조의 예상 시간과 빈도에 따라 구조화합니다. 단지 크기만을 위해 모듈화하지 마십시오. 중복된 코드를 서브루틴 또는 별개의 모듈과는 대조적으로 인라인을 고려하십시오.
- 실행을 제안하는 프로그램의 플로우가 순차적이지 아닐 때마다 별개의 서브프로그램을 사용하십시오.
- 터미널 사용자로부터 응답을 기다리는 기본 저장영역을 묶지 마십시오.
- 이동정보라는 명령 레벨 파일 제어 위치 지정형 입출력(I/O)을 사용하십시오.
- COBOL 프로그램에서 상수를 WORKING STORAGE 섹션에 데이터 변수가 아니라 PROCEDURE DIVISION에 리터럴로서 지정하십시오.
- C, C++ 및 PL/I 프로그램에서 상수 데이터를 위한 정적 저장영역을 사용하십시오.
- 기본 저장영역에 대한 요청을 생성하므로 가능하면 LINK 명령의 사용을 피하십시오.

참조 세트:

프로그램이 정상 작동 동안에 사용하는 전체 페이지 수를 가능한 적게 유지하십시오. 이러한 페이지를 참조 세트라고 부르고 프로그램의 실제 저장영역 요구사항을 표시합니다.

이 태스크 정보

참조 세트는 다음을 수행하여 줄일 수 있습니다.

- COBOL 프로그램에서 상수를 WORKING STORAGE SECTION에서 데이터 변수로서가 아니라 PROCEDURE DIVISION에서 리터럴로 지정하기. 그 이유는 프로그램을 실행하는 모든 태스크에는 작업 중인 저장영역의 별개의 사본이 있는 반면 리터럴은 CICS에서 단 하나의 사본만이 사용되는 프로그램 자체의 일부로 간주되기 때문입니다.
- 앞서와 동일한 이유로 C, C++ 및 PL/I에서 온전히 상수인 데이터에 정적 저장영역을 사용하기.
- 프로그램에서 가능하면 많이 데이터 영역을 재사용하기. 이를 COBOL에서 REDEFINES 절, C 및 C++에서는 union 절, PL/I에서는 기본 저장영역 및 어셈블러 언어에서는 ORG 또는 이와 동등한 것을 사용하여 이를 수행할 수 있습니다. 특히, 한 번에 하나의 맵만을 사용하는 맵핑 세트가 있는 경우에는

STORAGE=AUTO 또는 BASE 피연산자를 지정하지 않고 DFHMSD 맵핑 세트 정의를 코딩하십시오. 그러면 i맵핑 세트에 있는 맵이 서로를 재정의할 수 있습니다.

다음을 수행하여 데이터를 직접 참조하십시오.

- 테이블에서 데이터의 긴 검색을 피하기
- 체인과 같이 검색되어야 하는 구조 대신에 배열과 같이 직접 주소 지정할 수 있는 데이터 구조 사용하기.
- 간접 주소 지정을 시뮬레이트하는 메소드 피하기

애플리케이션에서 오버레이(페이징 기술)를 사용하려고 시도하지 마십시오. 시스템 페이징은 자동으로 제공되고 우수한 성능이 있습니다. 가상 저장영역의 애플리케이션의 디자인은 실제 환경의 디자인과 유사합니다. 시스템에는 언급되지 않은 페이지의 코드를 페이징할 필요가 없도록 모든 모듈을 상주시켜야 합니다.

프로그램이 동적인 경우에는 전체 프로그램은 실행이 시작되기 전에 인접한 페이지 전체에 걸쳐 로딩되어야 합니다. 동적 프로그램이 사용 중이 아니고 만족되지 않은 저장 영역 요청이 있는 경우 저장영역에서 이를 제거할 수 있습니다. 제거를 방지하기 위해 충분한 동적 영역을 허용하는 것은 이를 상주로 만드는 것보다 더 비쌉니다. 동적 프로그램은 페이지에서 사용되지 않는 공간을 다른 프로그램과 공유하지 않기 때문입니다.

자원 독점 제어

CICS가 제공하는 기본적인 강렬한 복구 기능은 성능에 영향을 줍니다. 다양한 접근방식을 채택하여 자원에 대한 경합 지연을 줄일 수 있습니다.

CICS는 트랜잭션이 실패하면 다른 트랜잭션이 수행한 변경사항과 관계없이 해당 자원에 대한 변경사항을 백아웃할 수 있도록 복구 가능 자원에 대한 갱신을 일련화합니다. 따라서 복구 가능 자원을 갱신하는 트랜잭션은 종료되거나 SYNCPOINT 명령을 사용하여 해당 변경사항을 확약하려 함을 표시할 때까지 해당 자원을 제어합니다. 동일한 자원이 필요한 다른 트랜잭션은 첫 번째 트랜잭션이 완료될 때까지 기다려야 합니다.

이러한 잠금 지연을 생성하는 1차 자원은 데이터 세트, DL/I 데이터베이스, 임시 저장 영역, 트랜지언트 데이터 큐입니다. 보호 기반 단위는 데이터 세트의 경우 개별 레코드(키), DL/I 데이터베이스의 경우 프로그램 스펙 블록(PSB), 임시 저장영역의 경우 큐 이름입니다. 트랜지언트 데이터의 경우, 큐『읽기』종료를『쓰기』종료와 별도의 자원으로 간주합니다(즉, 다른 트랜잭션이 큐에 쓰는 동안 한 트랜잭션이 큐에서 읽을 수 있음).

자원 소유권 경합으로부터 트랜잭션 지연을 줄이려면 자원 청구와 해제(UOW 끝) 간의 시간 길이를 최소화해야 합니다. 특히 대화식 트랜잭션은 터미널 읽기에서 중요한 자원을 소유해서는 안됩니다.

참고: 복구 불가능한 데이터 세트의 경우에도 VSAM은 두 개의 트랜잭션이 갱신을 위해 동일한 레코드를 동시에 읽지 못하게 합니다. 그러나 이 큐에 넣기는 UOW 끝에서가 아니라 갱신이 완료되는 즉시 종료됩니다. FCT(File Control Table)에 『독점 제어 없음』(SERVREQ=NOEXCTL)을 사용하여 정의함으로써 BDAM 데이터 세트의 경우 이 보호를 포기할 수도 있습니다.

특정 규칙이 관찰되지 않는 한 보호 설계는 지연은 물론 교착상태도 생성할 수 있습니다. 두 개의 트랜잭션이 하나 이상의 복구 가능 자원을 갱신하는 경우 항상 동일한 순서로 자원을 갱신해야 합니다. 예를 들어, 두 데이터 세트를 각각 갱신하는 경우에는 모든 트랜잭션에서 데이터 세트 『B』를 갱신하기 전에 데이터 세트 『A』를 갱신해야 합니다. 마찬가지로, 트랜잭션이 단일 데이터 세트에서 몇 개의 레코드를 갱신하는 경우 항상 일부 예측 가능한 순서(낮은 키에서 높은 키로, 또는 반대로). 각 READ UPDATE 명령과 함께 TOKEN 키워드를 포함시키는 것도 고려할 수 있습니다. TOKEN 키워드에 대한 정보는 709 페이지의 『TOKEN 옵션』의 내용을 참조하십시오. 트랜지언트 데이터, 임시 저장영역, 사용자 저널을 이러한 자원 간에 포함시켜야 합니다. CICS 복구 및 다시 시작 안내서의 애플리케이션에서 자원 잠금에는 자원 보호 범위에 대한 추가 정보가 들어 있습니다.

여기서 VSAM 제어 간격의 CICS 데이터 세트와 데이터 세트에 대한 VSAM 내부 잠금 간의 차이점에 주목하는 것이 적절합니다. CICS에는 VSAM 큐에 넣기에 대한 정보가 없으므로 일괄처리와 CICS에서 동시에 갱신되는 SHARE OPTION 4 제어 간격으로 인해 기껏해야 성능 감소를 초래하고 최악의 경우 일괄처리와 CICS 간에 발견 불가능한 교착 상황을 초래할 수 있습니다. 일괄처리와 CICS 간의 이러한 동시 갱신을 피해야 합니다. 어떤 경우라도 일괄처리와 CICS 둘 다 데이터 세트를 갱신하는 경우에는 CICS가 데이터 무결성을 보장할 수 없습니다.

운영 제어

다수의 운영 기술을 사용하여 CICS 시스템의 성능과 효율성에 영향을 줄 수 있습니다.

MXT

CICS 시스템 초기화 매개변수 MXT는 CICS 시스템에서 동시에 있을 수 있는 최대 사용자 태스크 수를 지정합니다. MXT는 저장영역 부족(SOS) 조건을 방지하고 CICS 시스템에서 자원 경합을 제어하는 데 매우 유용합니다. CICS 시스템에 활동이 이미 너무 많으면 입력 메시지를 처리하기 위해 사용자 태스크 작성을 지연시켜 작동합니다. 특히, 처리를 기다리는 메시지가 차지하는 가상 저장영역은 태스크가 메시지를 처리하는 데 필요한 영역보다 훨씬 적으므로 신속히 처리할 수 있을 때까지 메시지 처리를 지연시켜 가상 저장영역을 절약합니다.

트랜잭션 클래스는 특정 사용자 정의 유형 또는 클래스(자원을 많이 사용하는 사용자인 경우)의 태스크 수를 제한하는 데 유용합니다.

이탈 태스크

CICS는 태스크가 일시중단되면 태스크의 이탈 시간(ICVR)만 다시 설정합니다. 각 CICS 구현의 고유 네이처로 인해 EXEC CICS 명령이 처리 중에 태스크 일시중단을 야기한다고 보장할 수 없습니다. 이탈 시간을 초과하여 태스크가 AICA를 이상 종료할 수 있습니다. 애플리케이션에서 EXEC CICS SUSPEND 명령을 코드화하여 이 이상 종료를 방지할 수 있습니다. 이렇게 하면 디스패처가 요청을 실행한 태스크를 일시중단하고 우선순위가 더 높은 태스크를 실행하도록 허용합니다. 실행할 준비가 된 태스크가 없으면 일시중단을 실행한 프로그램이 재개됩니다. AICA에 대한 추가 정보는 *CICS 문제점 판별 안내서*의 CICS가 발견하지 않은 루프 조사를 참조하십시오.

보조 추적

보조 추적을 사용하여 애플리케이션을 검토하십시오. 예를 들어, 명백히 불필요한 코드를 표시할 수 있습니다(예: SETL 이후 대신 데이터 세트 처음부터 데이터 세트 찾아보기, 너무 많거나 너무 큰 GETMAIN 명령, 더 이상 필요가 없을 때 저장영역 해제 실패, 의도하지 않은 논리 루프, 더 이상 필요하지 않은 독점 제어에 대해 보유된 레코드 잠금 해제 실패).

운영 체제 대기

운영 체제 대기를 야기하는 기능 사용을 피해야 합니다. 이러한 대기 중 하나가 발생하면 모든 CICS 활동이 중단되고 모든 트랜잭션이 응답 지연을 겪습니다.

운영 체제 대기의 주요 소스는 다음과 같습니다.

- 리전 외부 트랜지언트 데이터 세트(313 페이지의 『효율적 순차 데이터 세트 액세스』 참조).
- CICS 프로그램에서 사용해서는 안되고 CICS가 일반적으로 대체 기능을 제공하는 해당 COBOL, C, C++ 및 PL/I 언어 기능. 언어 제한사항에 대한 지침 정보는 25 페이지의 제 3 장 『COBOL 애플리케이션 개발』, 53 페이지의 제 4 장 『C 및 C++로 프로그래밍』, 65 페이지의 제 5 장 『PL/I 애플리케이션 개발』의 내용을 참조하십시오.
- 운영 체제 서비스를 호출하는 SVC 및 어셈블러 언어 매크로(예: WTO(Write-To-Operator)).

NOSUSPEND 옵션

어떤 명령의 기본 조치는 필수 자원을 사용할 수 있게 될 때까지 애플리케이션을 일시 중단하는 것입니다. NOSUSPEND 옵션을 사용하여 이 대기를 금지하고 애플리케이션의 다음 명령어로 리턴할 수 있습니다.

ENQBUSY, NOJBUFSP, NOSPACE, NOSTG, QBUSY, SESSBUSY 및 SYSBUSY 조건의 기본 조치는 필수 자원(예: 저장영역)을 사용할 수 있게 될 때까지 애플리케이션 실행을 일시중단한 후 명령 처리를 재개하는 것입니다. 이러한 조건을 초래할 수 있는 명령은 다음과 같습니다.

- ALLOCATE
- ENQ
- GETMAIN
- WRITE JOURNALNAME
- WRITE JOURNALNUM
- READQ TD
- WRITEQ

NOSUSPEND 옵션(ALLOCATE 명령의 경우 NOQUEUE 옵션으로도 알려짐)을 사용하여 이 대기를 금지하고 명령을 뒤따르는 애플리케이션의 명령어로 즉시 리턴할 수 있습니다.

CICS는 COBOL 애플리케이션에서 HANDLE CONDITION 및 IGNORE CONDITION 명령이 참조하는 조건 테이블을 유지보수합니다.

제한사항: HANDLE CONDITION 및 IGNORE CONDITION 명령은 COBOL, PL/I, 어셈블러 언어 애플리케이션에서만 지원됩니다(AMODE(64) 어셈블러 언어 애플리케이션은 제외). 지원되는 기타 모든 고급 언어에서는 지원되지 않습니다.

이러한 명령 실행은 기존 항목을 갱신하거나 조건이 아직 이러한 명령의 주제가 되지 않은 경우에는 새 항목이 작성되게 합니다. 각 항목은 다음 세 가지 상태 중 하나를 표시합니다.

- 레이블이 현재 지정되었습니다.

```
HANDLE CONDITION condition(label)
```

- 조건을 무시합니다.

```
IGNORE CONDITION
```

- 현재 지정된 레이블이 없습니다.

```
HANDLE CONDITION
```

조건이 발생하면 다음 테스트가 수행됩니다.

1. 명령에 NOHANDLE 또는 RESP 옵션이 있으면 애플리케이션의 다음 명령어로 제어가 리턴됩니다. 그렇지 않으면, 조건 테이블을 스캔하여 수행할 작업을 확인합니다.
2. 조건에 대한 항목이 있으면 이 항목이 조치를 판별합니다.

3. 항목이 없고 이 조건에 대한 기본 조치가 실행을 일시중단하는 것이면 다음과 같습니다.
 - 명령에 NOSUSPEND 또는 NOQUEUE 옵션이 있는 경우 다음 명령어로 제어가 리턴됩니다.
 - 명령에 이러한 옵션 중 하나가 없으면 태스크가 일시중단됩니다.
4. 항목이 없고 이 조건에 대한 기본 조치가 이상 종료하는 것이면 두 번째 검색이 수행되어 ERROR 조건을 찾습니다.
 - 찾으면 이 항목이 조치를 판별합니다.
 - ERROR를 검색하고 찾을 수 없으면 태스크가 이상 종료됩니다.

효율적 순차 데이터 세트 액세스

CICS는 다수의 여러 순차 처리 옵션을 제공합니다. 각각 성능 특성이 다릅니다.

295 페이지의 『임시 저장영역 큐』 및 298 페이지의 『리전 내부 트랜지언트 데이터』에서 이미 설명한 임시 저장영역과 리전 내부 트랜지언트 데이터 큐는 사용하기 가장 효율적이지만 전적으로 CICS 내에서 작성하고 처리해야 합니다.

리전 외부 트랜지언트 데이터는 표준 순차(QSAM/BSAM) 데이터 세트를 처리하는 CICS 방법입니다. BDAM을 처리할 때와 마찬가지로 데이터 세트를 처리하려면 CICS가 운영 체제 대기를 실행해야 하므로 나열된 세 가지 양식의 순차 지원 중 가장 비효율적입니다. 또한 리전 외부 트랜지언트 데이터 세트는 복구 불가능합니다. 반면에 VSAM ESDS는 제한사항 내에서 복구 가능하며 처리가 더 효율적입니다. 복구 제한사항은 VSAM 제한사항으로 인해 완료되지 않은 UOW 중에 ESDS에 추가된 레코드를 백아웃 프로세스 중에 물리적으로 제거할 수 없다는 것입니다. 그러나 사용자 엑시트 루틴이 삭제한 것으로 플래그를 지정할 수 있습니다.

CICS 저널은 출력 데이터 세트에만 해당되긴 하지만 리전 외부 트랜지언트 데이터에 대한 다른 좋은 대안을 제공합니다. 저널은 MVS 시스템 로거를 통해 관리되지만 유연한 처리 옵션은 매우 효율적인 처리를 허용합니다. 각 저널 명령은 조작 특성(예: 동기 또는 비동기)을 지정하는 반면, 리전 외부 조작은 트랜지언트 데이터 큐 정의의 매개변수에 의해 전적으로 제어됩니다.

트랜잭션은 저널링과 관련하여 태스크 대기를 최소화하기 위해 가능하면 비동기로 저널링해야 합니다. 그러나 무결성 고려사항이 저널 레코드를 태스크 끝 앞에 물리적으로 기록하도록 요구하는 경우에는 동기 쓰기를 사용해야 합니다. 몇 개의 저널 쓰기가 있을 경우, 트랜잭션은 마지막 논리 레코드를 제외한 모두에 비동기 쓰기를 사용해야 하므로 태스크에 대한 논리 레코드는 최소 물리적 I/O 수와 한 번의 대기만 사용하여 기록됩니다.

CICS가 실행 중인 동안 입력(일괄처리)은 물론 출력(온라인)에도 저널을 사용할 수 있습니다. 예를 들어, 인쇄 또는 복사를 통해 제공된 일괄처리 유틸리티 DFHJUP를 저널 데이터에 대한 액세스에 사용할 수 있습니다. 일괄처리로 저널 읽기는 다음 제한사항을 수반함에 유의하십시오.

- MVS 시스템 로거 로그 스트림 데이터에 대한 액세스는 서브시스템 인터페이스인 LOGR을 통해 제공됩니다.
- 저널에서 레코드 읽기는 일괄처리 작업을 통해서만 오프라인으로 가능합니다.

효율적 로깅

CICS는 데이터 세트에 대해 활동의 일부 또는 모든 유형을 로깅하기 위한 옵션을 제공합니다.

로깅 갱신을 사용하면 필요한 경우 데이터 세트를 백업 사본으로부터 재구성할 수 있습니다. 보안상의 이유로 읽기를 로깅하고 싶을 수도 있습니다. 또한, 데이터 무결성과 보안의 필요성을 로깅이 성능에 미치는 영향과 균형을 맞춰야 합니다. 이들은 로깅 및 로깅이 암시하는 독점 제어 때문에 초래된 가능한 지연을 수행하는 데 필요한 실제 작업입니다.

트랜잭션 사이에 데이터 공유

CICS에는 트랜잭션 사이에 데이터를 공유하기 위한 몇 가지의 기능이 있습니다. CWA(Common Work Area), TCTTE 사용자 영역(TCTUA), COMMAREA, 표시장치 화면, 또는 채널 및 컨테이너를 사용할 수 있습니다.

TCTUA 및 CWA에 저장된 데이터는 시스템의 모든 트랜잭션에 사용 가능합니다. 자원 보안 및 저장영역 보호 제한사항에 따라, 임의 트랜잭션이 트랜잭션에 쓰고, 임의 트랜잭션이 트랜잭션을 읽을 수 있습니다.

이러한 기능 중 일부를 사용하면 트랜잭션간 연관관계가 발생할 수 있습니다. 트랜잭션 연관관계에 대한 자세한 정보는 379 페이지의 『연관관계』의 내용을 참조하십시오.

이 섹션에서는 다음에 대해 설명합니다.

- 315 페이지의 『CWA(Common Work Area) 사용』
- 319 페이지의 『TCTTE 사용자 영역(TCTUA) 사용』
- 320 페이지의 『RETURN 명령에서 COMMAREA 사용』
- 320 페이지의 『RETURN 명령에서 채널 사용』
- 321 페이지의 『표시장치 화면을 사용하여 데이터 공유』

CWA(Common Work Area) 사용

CWA(Common Work Area)는 시스템 시동 시 할당되고 해당 CICS 세션의 지속 기간 동안 존재하는 단일 제어 블록입니다. 시스템 초기화 매개변수 WRKAREA에 지정된 크기는 고정됩니다.

CWA의 특성은 다음과 같습니다.

- CWA에서 데이터를 저장하거나 검색할 때 오버헤드는 거의 없습니다. 명령 레벨 프로그램은 하나의 ADDRESS 명령을 실행하여 영역 주소를 가져오지만, 이후에는 직접 액세스할 수 있습니다.
- CWA의 데이터는 트랜잭션이나 시스템이 실패하는 경우 복구되지 않습니다.
- 자원 보안을 따르지 않습니다.
- CICS는 CWA 사용을 통제합니다. CWA를 사용하는 모든 애플리케이션의 모든 프로그램은 공유 사용에 대해 동일한 규칙을 따릅니다. 이 규칙은 보통 애플리케이션 개발자와 함께 시스템 프로그래머에 의해 정해지며, 모든 프로그램이 영역의 레이아웃을 설명하기 위해 동일한 『사본』 모듈을 사용하도록 요구합니다.

CWA의 길이를 초과하면 저장영역 위반이 발생하므로 초과하지 않도록 해야 합니다. 게다가, 하나의 트랜잭션에서 사용되는 데이터가 다른 트랜잭션에서 사용되는 데이터를 대체하지 않도록 해야 합니다. CWA 데이터를 보호하는 한 가지 방법은 사용자 키 애플리케이션에 의해 쓰여지지 않도록 CWA를 보호하는 저장영역 보호 기능을 사용하는 것입니다. 자세한 정보는 『CWA 보호』의 내용을 참조하십시오.

- CWA는 특히 애플리케이션에서 여러 프로그램에 의해 자주 읽혀지거나 갱신되는 적은 양의 데이터(상태 정보와 같은)에 적합합니다.
- CWA는 큰 볼륨이나 단기 수명 데이터에는 적합하지 않습니다. 이 경우 항상 할당되기 때문입니다.

CWA 보호

CWAKEY 시스템 초기화 매개변수는 CWA가 CICS 키 또는 사용자 키 저장영역에서 할당될 것인지 여부를 지정할 수 있도록 합니다.

CWAKEY 매개변수에 대한 세부사항은 CWAKEY 매개변수 CICS 시스템 정의 안내서의 내용을 참조하십시오.

쓰기 액세스를 CWA로 제한하려면 CWAKEY=CICS를 지정하면 됩니다. 이는 CICS가 CICS 키 저장영역에서 CWA를 할당함을 의미하며, EXECKEY(USER)로 정의된 애플리케이션을 CWA에 대한 읽기 전용 액세스로 제한합니다. CICS 키 저장영역에서 할당된 CWA에 쓰는 것이 허용되는 유일한 프로그램은 EXECKEY(CICS)로 정의하는 프로그램입니다.

CICS 키로 실행되는 프로그램은 CICS 저장영역에도 쓸 수 있어서, 이와 같은 프로그램이 CICS 저장영역을 겹쳐쓰지 않도록 완전하게 테스트되도록 합니다.

CWA보다 오히려 CICS를 보호하는 것을 우선으로 하려면, CWA에 대해 CWAKEY=USER를, CWA에 쓰는 모든 프로그램에 대해 EXECKEY(USER)를 지정하십시오. 그러면 프로그램이 CWA의 길이를 초과하는 경우 CICS 저장영역에 겹쳐 쓰지 않게 됩니다. 저장영역 보호에 대한 자세한 정보는 787 페이지의 『저장영역 제어』의 내용을 참조하십시오.

그림 63에서는 CWA 자체가 CWAKEY=CICS에 의해 사용자 키 애플리케이션으로부터 보호되는 CWA의 특정 사용에 대해 보여줍니다.

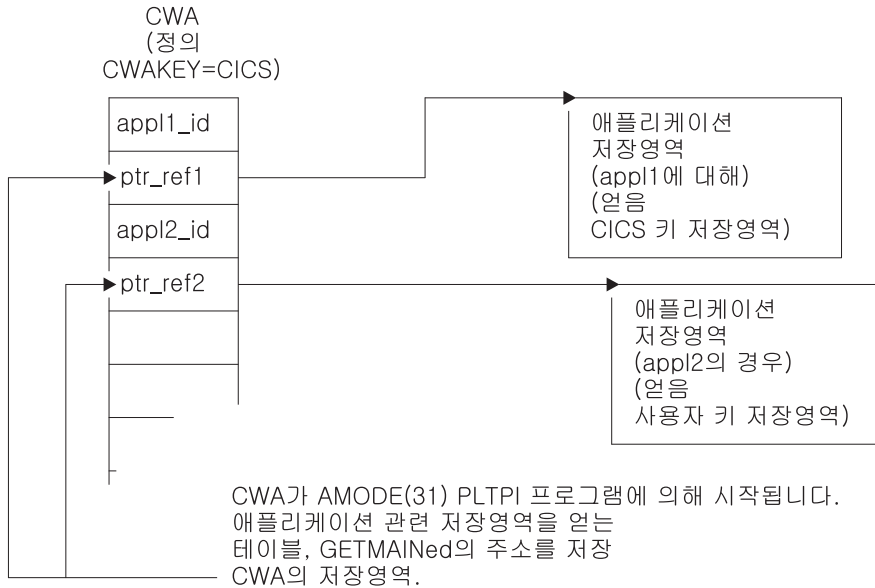


그림 63. CICS 키 저장영역에서 CWA의 사용 예제. 이 예제는 애플리케이션이 사용하도록 사용자 키 또는 CICS 키 저장영역에서 확보되는 저장영역을 참조하기 위해 CWA를 사용할 수 있는 방법을 보여주며, CWA 자체는 CICS 키 저장영역에서 보호됩니다.

이 그림에서, CWA는 애플리케이션 데이터와 상수를 저장하기 위해 직접 사용되지 않습니다. CWA에는 애플리케이션 ID 및 연관된 주소의 쌍이 포함되며, 주소 필드에는 애플리케이션 관련 데이터를 보유하는 데이터 영역의 주소가 포함됩니다. 보호에 대해, CWA는 CWAKEY=CICS로 정의되므로, 이 설명에서 프로그램 목록 테이블 사후 초기설정(PLTPI) 목록에 정의된 프로그램이며 주소 및 애플리케이션 ID와 함께 CWA를 로드하는 프로그램은 EXECKEY(CICS)로 정의해야 합니다. CWA에 대한 액세스가 요구되는 애플리케이션은 EXECKEY(USER)로 정의해야 하며, CWA가 애플리케이션에 의해 겹쳐서 쓰여지지 않도록 보호합니다. 그림 63에서, 데이터 영역 중 하나는 CICS 키 저장영역에서 확보되고, 다른 데이터 영역은 사용자 키 저장영역에서 확보됩니다.

318 페이지의 그림 64에 표시된 샘플 코드에서, 프로그램 목록 테이블 사후 초기설정 (PLTPI) 프로그램은 CWA에 지정된 데이터에 대한 포인터로 애플리케이션 데이터 영역을 설정합니다.

이 예제는 CWA에 저장된 데이터의 주소로, 애플리케이션에서 사용하도록 글로벌 데이터를 작성하는 방법을 보여줍니다(예를 들어, PLTPI 프로그램으로). 첫 번째 데이터 영역은 PLTPI 프로그램에 의해 실행된 GETMAIN 명령의 기본값인 CICS 키 저장영역에서 확보되고, 두 번째는 USERDATAKEY 옵션을 지정하여 사용자 키 저장영역에서 확보됩니다. CWA 자체는 CICS 키 저장영역에 있으며, PLTPROG는 EXECKEY(CICS)로 정의됩니다.

```

ID DIVISION.
PROGRAM-ID. PLTPROG.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 APPLID          PIC X(8)      VALUE SPACES.
77 SYSID           PIC X(4)      VALUE SPACES.
01 COMM-DATA.
   03 AREA-PTR     USAGE IS POINTER.
   03 AREA-LENGTH  PIC S9(8)    COMP.
LINKAGE SECTION.
01 COMMON-WORK-AREA.
   03 APPL-1-ID    PIC X(4).
   03 APPL-1-PTR   USAGE IS POINTER.
   03 APPL-2-ID    PIC X(4).
   03 APPL-2-PTR   USAGE IS POINTER.
PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.
* Obtain APPLID and SYSID values
  EXEC CICS ASSIGN APPLID(APPLID)
                SYSID(SYSID)

  END-EXEC.
* Set up addressability to the CWA
  EXEC CICS ADDRESS
                CWA(ADDRESS OF COMMON-WORK-AREA)

  END-EXEC.
* Get 12KB of CICS-key storage for the first application ('APP1')
  MOVE 12288    TO AREA-LENGTH.
  EXEC CICS GETMAIN SET(AREA-PTR)
                FLENGTH(AREA-LENGTH)
                SHARED

  END-EXEC.
* Initialize CWA fields and link to load program
* for storage area 1.
  MOVE 'APP1'   TO APPL-1-ID.
  SET APPL-1-PTR TO AREA-PTR.
  EXEC CICS LINK PROGRAM('LOADTAB1')
                COMMAREA(COMM-DATA)

  END-EXEC.
* Get 2KB of user-key storage for the second application ('APP2')
  MOVE 2048     TO AREA-LENGTH.
  EXEC CICS GETMAIN SET(AREA-PTR)
                FLENGTH(AREA-LENGTH)
                SHARED
                USERDATAKEY

  END-EXEC.
* Initialize CWA fields and link to load program
* for storage area 2.
  MOVE 'APP2'   TO APPL-2-ID.
  SET APPL-2-PTR TO AREA-PTR.
  EXEC CICS LINK PROGRAM('LOADTAB2')
                COMMAREA(COMM-DATA)

  END-EXEC.
  EXEC CICS RETURN
  END-EXEC.
MAIN-PROCESSING-EXIT.
  GOBACK.

```

그림 64. CWA를 로드하기 위한 샘플 코드

TCTTE 사용자 영역(TCTUA) 사용

TCT 사용자 영역(TCTUA)은 터미널 제어 테이블 항목(TCTTE)의 선택적 확장입니다. TCT의 각 항목은 이 확장이 존재하는지 여부와, 존재하는 경우 존재 시간을 지정합니다(터미널에 사용되는 TYPETERM 자원 정의의 USERAREALEN 속성으로).

TYPETERM 자원 정의에 대한 자세한 정보는 *CICS 자원 정의 안내서*에서 모델 터미널 정의 자동 설치를 참조하십시오.

시스템 초기화 매개변수 TCTUALOC 및 TCTUAKEY는 모든 TCTUA의 위치 및 저장영역 키를 지정합니다.

- TCTUALOC=BELOW 또는 ANY는 TCTUA에 대해 24비트 또는 31비트 주소 지정 가능성을 원하는지 여부와, TCTCUA가 16MB 라인 아래에 저장되어야 하거나 이 라인 위 또는 아래에 있을 수 있는지 여부를 지정합니다.
- TCTUAKEY=USER 또는 CICS는 TCTUA가 사용자 키 또는 CICS 키 저장영역에서 할당되도록 할 것인지 여부를 지정합니다.

TCTUA에는 CWA와 공통적으로 다음 특성을 가지고 있습니다.

- 최소 프로세서 오버헤드(단 하나의 ADDRESS 명령이 필요함)
- 복구 안함
- 자원 보안 없음
- CICS에 의한 사용 조절 없음
- 고정 길이
- 대형 블록 또는 단기 데이터에 대한 부적절성

그러나 CWA와 달리, 특정 터미널에 대한 TCTUA는 보통 해당 터미널을 사용하는 트랜잭션 사이에서만 공유됩니다. 따라서 이는 의사 대화식 순서에서 일련의 트랜잭션 사이에 작은 양의 표준 길이 데이터를 저장하는데 유용합니다. 다른 차이점은 영구적으로 할당할 필요가 없다는 것입니다. TCTTE가 설정되어 있는 동안 TCTUA만 존재하기 때문입니다. 자동 설치가 아닌 터미널의 경우 TCTUA는 시스템 시동 시 할당되고, 자동 설치 터미널의 경우에는 TCTTE가 생성될 때 TCTUA가 할당됩니다.

이 방식으로 TCTUA를 사용하면 사용하는 트랜잭션 사이에 특수 훈련이 필요하지 않습니다. 데이터가 항상 데이터를 쓴 트랜잭션 다음의 트랜잭션에 의해 읽혀지기 때문입니다. 그러나, TCTUA를 사용하여 장기 데이터를 저장하는 경우(예를 들어, 전체 애플리케이션에서 필요한 터미널 또는 운영자 정보), 하나의 트랜잭션에서 사용되는 데이터가 다른 트랜잭션에서 사용되는 데이터를 덮어쓰지 않도록 CWA와 같이 주의해야 합니다. 할당된 TCTUA의 길이를 초과하면 저장영역 위반이 발생하므로 초과하지 않도록 합니다.

RETURN 명령에서 COMMAREA 사용

RETURN 명령의 COMMAREA 옵션은 의사 대화식 순서에서 연속 트랜잭션 사이에 데이터를 전달하기 위해 특별히 디자인되었습니다. 이 옵션은 애플리케이션보다 EXEC 인터페이스가 GETMAIN 및 FREEMAIN 요청을 실행하지만, 사용자 저장영역의 특수 양식으로 구현됩니다.

COMMAREA는 기본 저장영역에서 CICS 공유 서브풀로부터 할당되고, 의사 대화식 애플리케이션의 태스크 사이에, TCTTE에 의해 주소 지정됩니다. 다음 태스크에 전달되지 않으면 COMMAREA는 사용 가능하게 해제됩니다.

프로그램이 LINK 명령 또는 XCTL 명령에 의해 호출되었던 것처럼(294 페이지의 『LINK 및 XCTL 명령의 COMMAREA』 참조), 다음 태스크에서 첫 번째 프로그램이 전달된 COMMAREA에 대한 자동 주소 지정 가능성을 갖습니다. COMMAREA의 주소를 확보하기 위해 ADDRESS 명령의 COMMAREA 옵션을 사용할 수도 있습니다.

분산 환경의 의사 대화식 순서에서 연속 트랜잭션 사이에 전달되는 COMMAREA의 경우, SNA용 z/OS Communications Server는 총 데이터 길이 크기를 32KB로 제한합니다. 이 한계는 전체 전송된 패키지(Communications Server가 추가한 제어 데이터 포함)에 적용됩니다. 전송에서 중간 링크가 사용되면 제어 데이터의 양이 증가합니다.

요약하면 다음과 같습니다.

- 프로세서 오버헤드가 낮습니다. (XCTL 명령에서 COMMAREA를 사용하는 것과 같고, 기본 임시 저장영역을 사용하는 것과 거의 같습니다.)
- 복구 가능하지 않습니다.
- 자원 보안이 없습니다.
- 대량의 데이터에 적합하지 않습니다. (기본 저장영역이 사용되므로, 터미널 사용자가 응답할 때까지 보유됩니다.)
- 프로그램 사이에 데이터를 전송하기 위해 COMMAREA를 사용하는 것과 같이, 해당 프로그램이 명백하게 데이터 또는 해당 주소를 후속 프로그램으로 전달하지 않는 한, 트랜잭션의 첫 번째 프로그램에만 사용 가능합니다.

RETURN 명령에서 채널 사용

통신 영역을 사용하는 대신에 데이터를 허위 대화식에서 다음 프로그램으로 데이터를 전달하는 보다 근대적인 방식은 채널을 사용하는 것입니다.

채널에는 COMMAREA보다 여러 가지 장점이 있습니다. 330 페이지의 『채널의 이점』의 내용을 참조하십시오. RETURN 명령에서 채널을 전달하려면 COMMAREA 옵션 대신에 CHANNEL 옵션을 사용하십시오.

채널은 321 페이지의 『채널을 사용하는 프로그램 간 데이터 통신』에 설명되어 있습니다.

표시장치 화면을 사용하여 데이터 공유

데이터는 표시장치 화면 자체에 3270 표시장치 터미널로부터의 허위 대화식 트랜잭션 사이에 저장될 수 있습니다.

예를 들면, 데이터 입력에서 사용자에게 의해 만들어진 오류는 데이터를 처리하여 트랜잭션에 의해(강조표시 또는 메시지로) 강조표시됩니다. 그런 다음 다음 번 트랜잭션 ID는 자체를 가리키도록 설정되고(수정된 항목을 처리할 수 있도록) CICS로 돌아갑니다.

트랜잭션에는 올바른 데이터를 사용하는 두 가지 방법이 있습니다. 이를 저장할 수 있고(예를 들어, COMMAREA에), 이를 다음 번 실행을 위해 전달할 수 있습니다. 이 경우에는 트랜잭션은 화면에서 변경된 데이터를 이전 항목의 데이터와 병합해야 합니다. 또는 키 지정 필드의 수정된 데이터 태그를 끄지 않고 화면에 데이터를 저장할 수도 있습니다.

화면에 데이터를 저장하는 것은 코딩하기 쉽지만 이는 안전하지 않습니다. 변경되지 않은 데이터를 재전송하기 위해서는 추가 네트워크 트래픽이 필요하기 때문에 오류가 발생할 수 있으므로 대량의 데이터를 포함하는 화면은 저장하지 않는 것이 좋습니다. (이 제한사항은 로컬에 접속된 터미널에는 적용되지 않습니다.)

두 번째로 사용자가 CLEAR 키를 누르면 화면 데이터는 유실되고, 트랜잭션은 여기에서 복구할 수 있어야 합니다. 이는 관련된 애플리케이션에 해당하는 경우, CANCEL 또는 QUIT를 의미하는 CLEAR 키를 정의하여 피할 수 있습니다.

키 지정 데이터 이외의 데이터는 또한 화면에 저장될 수 있습니다. 이 데이터는 CLEAR에 의해 초래된 것을 제외하고는 변경으로부터 보호될 수 있고, 필요한 경우 표시되지 않을 수 있습니다.

채널을 사용하는 프로그램 간 데이터 통신

채널은 COMMAREA에 적용되는 32KB 한계를 훨씬 초과하는 양으로 프로그램 간 데이터를 전송하는 메소드입니다.

이 섹션에는 다음 정보가 포함되어 있습니다.

- 322 페이지의 『채널: 빠른 시작』
- 327 페이지의 『채널 사용: 일부 일반 시나리오』
- 332 페이지의 『채널 작성』
- 333 페이지의 『현재 채널』
- 338 페이지의 『채널의 범위』
- 343 페이지의 『프로그램에 전달된 컨테이너 발견』
- 343 페이지의 『링크에서 리턴된 컨테이너 발견』
- 350 페이지의 『CICS 읽기 전용 컨테이너』
- 346 페이지의 『채널 설계: 우수 사례』

- 348 페이지의 『채널 구성 및 사용: 예제』
- 350 페이지의 『채널 및 BTS 활동』
- 350 페이지의 『JCICS에서 채널 사용』
- 353 페이지의 『채널을 사용한 동적 라우팅』
- 353 페이지의 『데이터 변환』
- 330 페이지의 『채널의 이점』
- 360 페이지의 『COMMAREA에서 채널로 마이그레이션』
- 344 페이지의 『채널과 컨테이너 삭제 및 해당 저장영역의 사용 가능화』

채널: 빠른 시작

채널 및 컨테이너에 대한 간략한 소개입니다.

채널 및 컨테이너

컨테이너는 프로그램 간에 정보를 전달할 용도로 설계된 이름 지정된 데이터 블록입니다. 여러 프로그램은 서로 간에 임의의 수의 컨테이너를 전달할 수 있습니다. 컨테이너는 채널이라는 세트로 그룹화됩니다. 채널은 매개변수 목록과 비슷합니다.

이름 지정된 컨테이너를 작성해서 채널에 지정하기 위해 프로그램은 **EXEC CICS PUT CONTAINER(container-name) CHANNEL(channel-name)** 명령을 사용합니다. 프로그램은 그런 다음 **EXEC CICS LINK, XCTL, START** 또는 **RETURN** 명령의 **CHANNEL(channel-name)** 옵션을 사용하여 채널과 컨테이너를 두 번째 프로그램에 전달합니다.

두 번째 프로그램은 **EXEC CICS GET CONTAINER(container-name)** 명령을 사용하여 전달된 컨테이너를 읽을 수 있습니다. 이 명령은 프로그램을 호출한 채널에 속한 이름 지정된 컨테이너를 읽습니다.

두 번째 프로그램이 **EXEC CICS LINK** 명령으로 호출된 경우 호출 프로그램에 컨테이너를 리턴할 수도 있습니다. 새 컨테이너를 작성하거나 기존 컨테이너를 재사용해서 이를 수행할 수 있습니다.

채널과 컨테이너는 채널과 컨테이너를 작성하는 프로그램 및 채널과 컨테이너가 전달되는 프로그램에만 표시됩니다. 이 프로그램이 종료되면 CICS는 컨테이너와 해당 저장영역을 자동으로 제거합니다. 프로그램은 이 시기 이전에 채널과 컨테이너를 삭제할 명령을 실행할 수도 있습니다.

AMODE(64) 프로그램은 **EXEC CICS PUT64 CONTAINER** 및 **EXEC CICS GET64 CONTAINER** 명령을 통해 채널과 컨테이너를 사용해서 동일한 방식으로 64비트 저장영역의 데이터를 전송할 수 있습니다. 이러한 명령은 비Language Environment(LE) AMODE(64) 어셈블리 언어 애플리케이션에만 사용되며, CICS 비즈니스 트랜잭션 서비스(BTS) 컨테이너는 지원되지 않습니다.

채널 컨테이너는 복구 가능하지 않습니다. RETURN TRANSID CHANNEL()로 시작된 의사 대화식 트랜잭션은 재시작할 수 없습니다. 복구 가능 컨테이너를 사용해야 하는 경우에는 CICS 비즈니스 트랜잭션 서비스(BTS) 컨테이너를 사용하십시오.

트랜잭션 채널 DFHTRANSACTION

채널은 일반적으로 링크 레벨이 변경될 때 범위를 벗어납니다. 따라서 트랜잭션의 모든 프로그램에 대해 사용 가능하지 않을 수 있습니다. 이름이 DFHTRANSACTION인 채널을 작성하는 경우 이 채널은 링크 레벨이 변경될 때 범위를 벗어나지 않습니다. 따라서 API 사용 종료점을 포함하여, 트랜잭션의 모든 프로그램이 이 채널을 사용할 수 있습니다. 하지만 트랜잭션 채널은 CICS TS 5.2 이전 버전의 CICS 리전에 전달할 수 없습니다.

DFHTRANSACTION은 채널 이름을 승인하는 모든 API 명령에서 사용할 수 있습니다.

기본 예제

채널을 작성하고 두 번째 프로그램에 전달하는 프로그램의 단순 예제입니다.

325 페이지의 그림 65에서는 다음을 수행하는 COBOL 프로그램 CLIENT1을 보여줍니다.

1. PUT CONTAINER(*container-name*) CHANNEL(*channel-name*) 명령을 사용하여 inqcustrec라고 하는 채널을 작성하고 이 채널에 고객 번호 및 분기 번호를 각각 포함하는 두 개의 컨테이너 custno 및 branchno를 추가합니다.
2. LINK PROGRAM(*program-name*) CHANNEL(*channel-name*) 명령을 사용하여 프로그램 SERVER1에 링크하고 inqcustrec 채널을 전달합니다.
3. GET CONTAINER(*container-name*) CHANNEL(*channel-name*) 명령을 사용하여 SERVER1에서 리턴된 고객 레코드를 검색합니다. 고객 레코드는 inqcustrec 채널의 custrec 컨테이너에 있습니다.

클라이언트 및 서버 프로그램 둘 다에서 동일한 COBOL 카피북인 INQINTC를 사용할 것을 참고하십시오. 카피북의 3행 및 5행에서 7행까지는 INQUIRY-CHANNEL 및 해당 컨테이너를 나타냅니다. 이러한 행이 프로그램의 작동을 위해 꼭 필요한 것은 아닙니다. 채널 및 컨테이너는 예를 들어, PUT CONTAINER와 같은 명령에서 이름 지정되어 작성되고 정의할 필요가 없기 때문입니다. 그러나 두 프로그램에서 사용되는 카피북에 이러한 행을 포함시키면 유지보수가 용이해지고 사용된 컨테이너의 이름을 기록합니다.

권장사항

채널을 사용하는 클라이언트/서버 애플리케이션을 보다 쉽게 유지보수하려면 사용된 컨테이너의 이름을 기록하는 카피북을 작성하고 컨테이너로 맵핑되는 데이터 필드를 정의하십시오. 클라이언트 및 서버 프로그램 둘 다에 카피북을 포함시키십시오.

참고: 이 예제는 두 개의 COBOL 프로그램을 보여줍니다. CICS에서 지원되는 다른 언어로 같은 기술을 사용할 수 있습니다. *COBOL* 프로그램에만 해당됨, 그러나 서버 프로그램이 **EXEC CICS GET CONTAINER** 명령에서 SET 옵션(INTO 대신)을 사용하는 경우 SET에 의해 지정되는 저장영역의 구조를 프로그램의 LINKAGE 섹션에서 정의해야 합니다. 이는 한 개가 아닌 두 개의 카피북이 필요함을 의미합니다. 첫 번째로 프로그램의 WORKING-STORAGE 섹션에서 사용되는 채널 및 컨테이너의 이름을 지정합니다. 두 번째로 LINKAGE 섹션에서 저장영역 구조를 정의합니다.

IDENTIFICATION DIVISION.
PROGRAM-ID. CLIENT1.

WORKING-STORAGE SECTION.

```

COPY INQINTC
*           copybook INQINTC
* Channel name
* 01 INQUIRY-CHANNEL PIC X(16) VALUE 'inqcustrec'.
* Container names
* 01 CUSTOMER-NO      PIC X(16) VALUE 'custno'.
* 01 BRANCH-NO        PIC X(16) VALUE 'branchno'.
* 01 CUSTOMER-RECORD  PIC X(16) VALUE 'custrec'.
* Define the data fields used by the program
* 01 CUSTNO           PIC X(8).
* 01 BRANCHNO         PIC X(5).
* 01 CREC.
* 02 CUSTNAME         PIC X(80).
* 02 CUSTADDR1        PIC X(80).
* 02 CUSTADDR2        PIC X(80).
* 02 CUSTADDR3        PIC X(80).

```

PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.

```

*
* INITIALISE CUSTOMER RECORD
*
*     ... CREATE CUSTNO and BRANCHNO
*
* GET CUSTOMER RECORD
*
*     EXEC CICS PUT CONTAINER(CUSTOMER-NO) CHANNEL(INQUIRY-CHANNEL)
*                   FROM(CUSTNO) FLENGTH(LENGTH OF CUSTNO)
*                   END-EXEC
*     EXEC CICS PUT CONTAINER(BRANCH-NO) CHANNEL(INQUIRY-CHANNEL)
*                   FROM(BRANCHNO) FLENGTH(LENGTH OF BRANCHNO)
*                   END-EXEC
*
*     EXEC CICS LINK PROGRAM('SERVER1') CHANNEL(INQUIRY-CHANNEL) END-EXEC
*
*     EXEC CICS GET CONTAINER(CUSTOMER-RECORD) CHANNEL(INQUIRY-CHANNEL)
*                   INTO(CREC) END-EXEC
*
*
* PROCESS CUSTOMER RECORD
*
*     ... FURTHER PROCESSING USING CUSTNAME and CUSTADDR1 etc...
*
*     EXEC CICS RETURN END-EXEC
*
* EXIT.

```

그림 65. 채널을 작성하고 작성한 채널을 두 번째 프로그램으로 전달하는 프로그램의 단순 예제입니다.

그림 66에서는 CLIENT1에 의해 링크되는 SERVER1 프로그램을 보여줍니다. 데이터가 전달된 custno 및 branchno 컨테이너에서 SERVER1은 데이터를 검색하고 검색한 데이터를 사용하여 데이터베이스에서 전체 고객 레코드를 찾습니다. 그런 다음 같은 채널에서 새 컨테이너 custrec를 작성하고 이 컨테이너에서 고객 레코드를 리턴합니다.

프로그래머가 SERVER1의 GET 및 PUT 명령에서 CHANNEL 키워드를 지정하지 않았음을 참고하고 채널이 명시적으로 지정되지 않은 경우 현재 채널(즉, 프로그램을 호출하는 데 사용된 채널)이 사용됩니다.

IDENTIFICATION DIVISION.
PROGRAM-ID. SERVER1.

WORKING-STORAGE SECTION.

```
COPY INQINTC
*               copybook INQINTC
* Channel name
* 01 INQUIRY-CHANNEL PIC X(16) VALUE 'inqcustrec'.
* Container names
* 01 CUSTOMER-NO      PIC X(16) VALUE 'custno'.
* 01 BRANCH-NO        PIC X(16) VALUE 'branchno'.
* 01 CUSTOMER-RECORD  PIC X(16) VALUE 'custrec'.
* Define the data fields used by the program
* 01 CUSTNO           PIC X(8).
* 01 BRANCHNO        PIC X(5).
* 01 CREC.
* 02 CUSTNAME         PIC X(80).
* 02 CUSTADDR1        PIC X(80).
* 02 CUSTADDR2        PIC X(80).
* 02 CUSTADDR3        PIC X(80).
```

PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.

```
EXEC CICS GET CONTAINER(CUSTOMER-NO)
                INTO(CUSTNO) END-EXEC
EXEC CICS GET CONTAINER(BRANCH-NO)
                INTO(BRANCHNO) END-EXEC
```

... USE CUSTNO AND BRANCHNO TO FIND CREC IN A DATABASE

```
EXEC CICS PUT CONTAINER(CUSTOMER-RECORD)
                FROM(CREC)
                FLENGTH(LENGTH OF CREC) END-EXEC
```

```
EXEC CICS RETURN END-EXEC
```

```
EXIT.
```

그림 66. 데이터가 전달된 채널에서 데이터를 검색하는 프로그램에 대한 링크의 단순 예제입니다. 이 프로그램은 325 페이지의 그림 65에 표시된 프로그램 CLIENT1에 의해 링크됩니다.

채널 사용: 일부 일반 시나리오

채널과 컨테이너는 프로그램 간에 데이터를 전달할 강력한 방법을 제공합니다. 이 시나리오는 채널의 사용 방식에 대한 몇 가지 예제를 보여줍니다.

한 개의 채널, 한 개의 프로그램

이 예제는 단일 채널이 있는 독립형 프로그램을 보여줍니다.

그림 67에서는 가장 단순한 시나리오(『독립형』 프로그램과 이 프로그램을 호출하는 데 사용할 수 있는 단일 채널)를 보여줍니다.

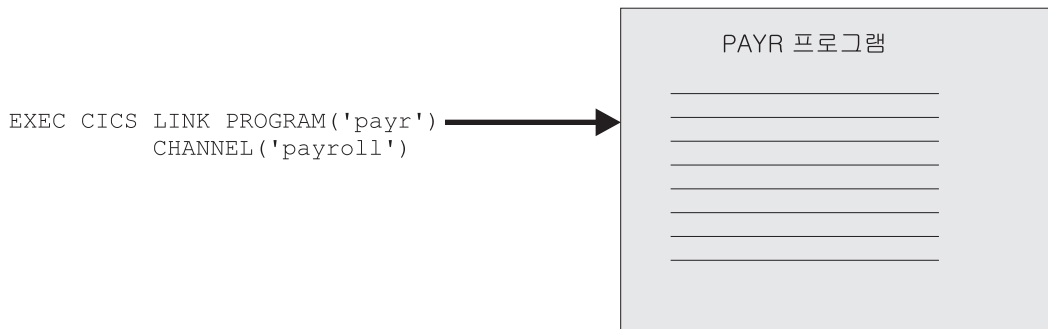


그림 67. 단일 채널이 있는 독립형 프로그램

한 개의 채널, 여러 개의 프로그램(구성요소)

이 예제는 단일 채널을 통해 호출되는 일련의 관련 프로그램(구성요소)을 보여줍니다.

328 페이지의 그림 68에는 일련의 상관된 프로그램에서 최상위 레벨 프로그램에 대한 단일 채널이 있습니다. 음영 표시된 영역 내 일련의 프로그램은 『구성요소』로 간주할 수 있습니다. 클라이언트 프로그램은 외부 채널만 『보고』 발생하는 처리 및 백엔드 프로그램의 존재에 대해서는 알지 못합니다.

구성요소 내부에서 프로그램은 서로 채널을 전달할 수 있습니다. 또는 구성요소 프로그램은 예로, 새 채널을 작성하고 원래 채널의 컨테이너를 한 개 이상 추가하여 원래 채널의 서브세트를 전달할 수 있습니다.

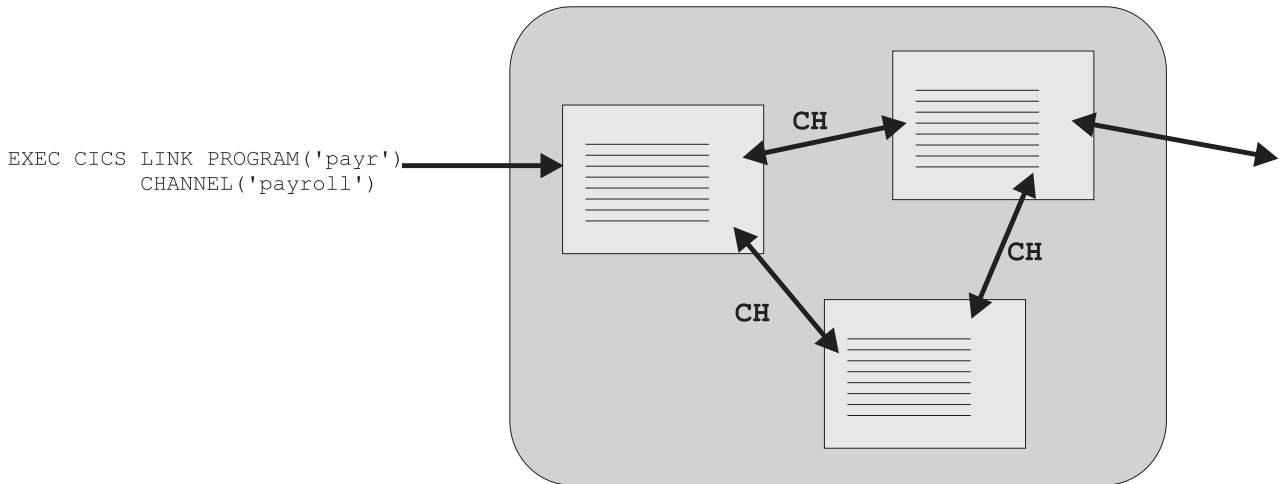


그림 68. 『구성요소』—단일 외부 채널을 통해 호출되는 일련의 관련 프로그램. 『CH』는 구성요소 내에서 프로그램이 서로 채널을 전달할 수 있음을 나타냅니다.

여러 개의 채널, 한 개의 구성요소

이 예제는 두 개의 대체 채널을 통해 호출할 수 있는 일련의 관련 프로그램(구성요소)을 보여줍니다.

이전 예제에서와 같이 구성요소로 간주할 수 있는 일련의 상관 프로그램이 있습니다. 그러나 여기에는 구성요소를 호출하는 데 사용할 수 있는 두 개의 대체 외부 채널이 있습니다. 구성요소에서 최상위 레벨 프로그램은 EXEC CICS ASSIGN CHANNEL 명령을 실행하여 구성요소를 호출하는 데 사용된 채널을 판별하고 그에 따라 조정하여 처리합니다.

클라이언트 프로그램과 구성요소 간 『느슨한 결합』은 쉬운 전개를 허용합니다. 즉, 클라이언트와 구성요소를 서로 다른 시간에 업그레이드할 수 있습니다. 예를 들어, 먼저 구성요소를 세 번째 채널(첫 번째 또는 두 번째 채널의 여러 컨테이너 세트로 구성됨)을 처리하도록 업그레이드할 수 있습니다. 다음에는 클라이언트 프로그램을 업그레이드(또는 새 클라이언트 작성)하여 세 번째 채널을 전달할 수 있습니다.

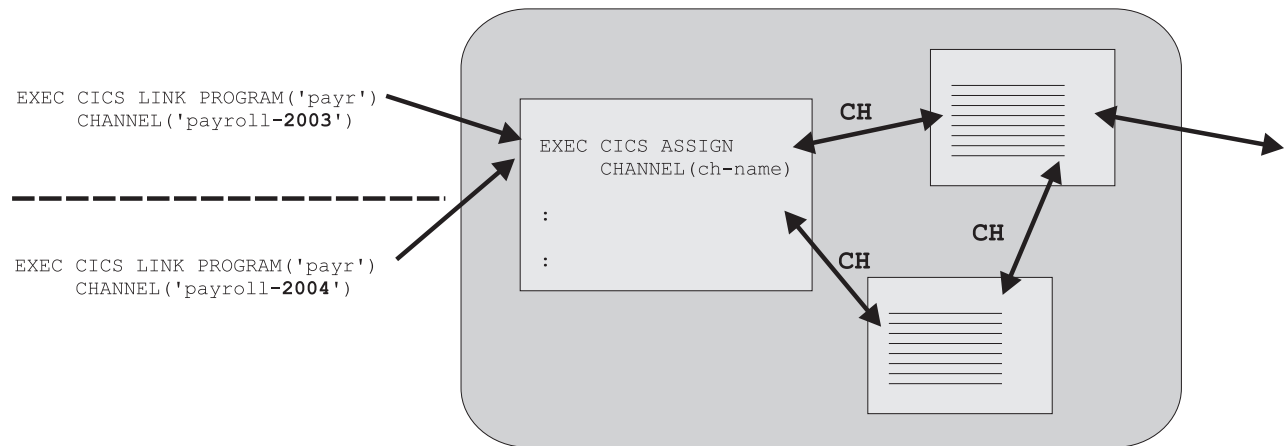


그림 69. 동일 구성요소에 대한 다중 외부 채널. 『CH』는 구성요소 내에서 프로그램이 서로 채널을 전달할 수 있음을 나타냅니다.

다중 대화식 구성요소

이 예제는 다중 구성요소가 채널을 통해 상호작용할 수 있는 방법을 보여줍니다.

330 페이지의 그림 70에서는 각각 호출에 사용할 수 있는 한 개의 채널을 포함하는 『인적 자원』 구성요소와 『급여』 구성요소를 보여줍니다. 급여 구성요소는 독립형 프로그램 및 인적 자원 구성요소에서 호출합니다.

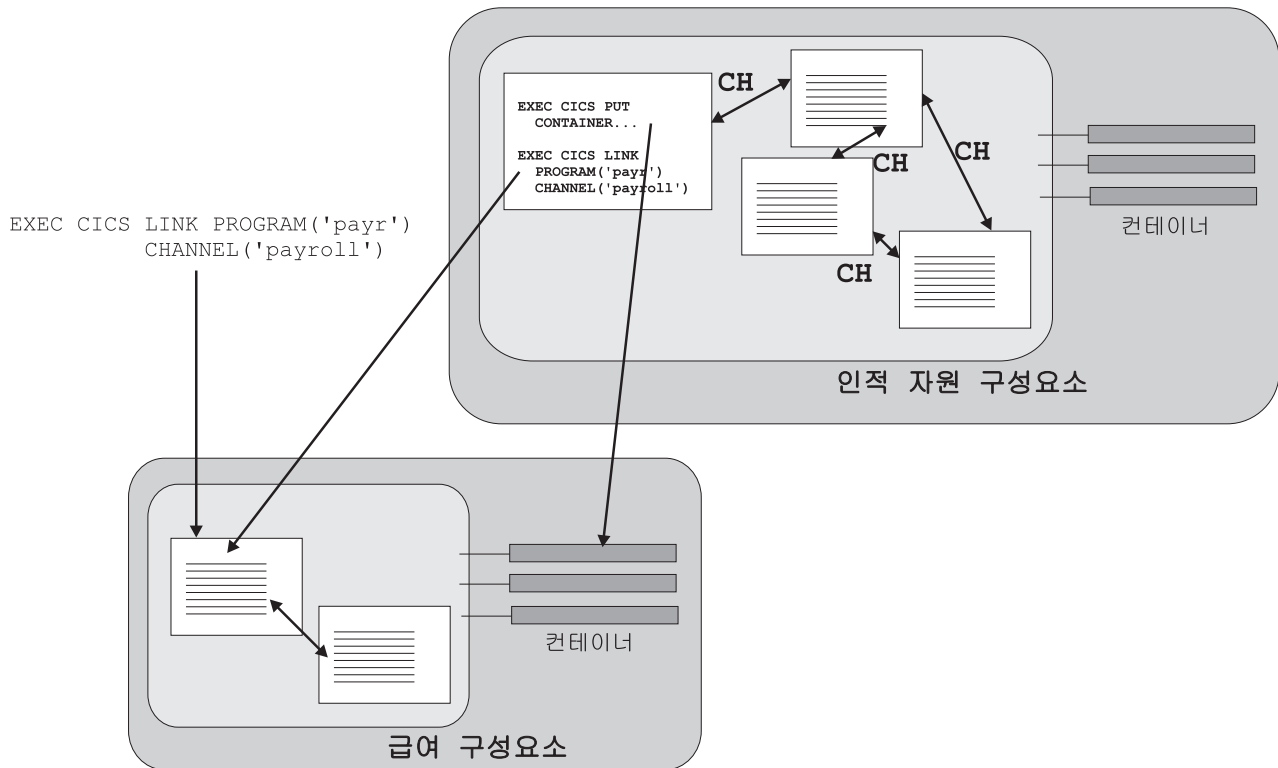


그림 70. 채널을 통해 상호작용하는 다중 구성요소

채널의 이점

CICS 프로그램에서 채널과 컨테이너 모델을 사용하여 데이터를 교환하는 방식은 통신 영역 사용에 비해 여러 장점이 있습니다.

- 채널에 추가할 수 있는 컨테이너 수에 대한 한계가 없고 개별 컨테이너의 크기는 사용자가 사용할 수 있는 저장영역의 크기를 통해서만 제한됩니다.

다른 애플리케이션에 사용 가능한 저장영역 한계를 초과하는 너무 큰 컨테이너를 작성하지 않도록 주의하십시오.

- 채널은 구조화된 방식으로 데이터를 전달하는 여러 컨테이너로 구성될 수 있습니다.
- 채널은 리턴된 데이터의 정확한 크기를 알기 위해 사용하는 프로그램을 필요로 하지 않습니다.
- 채널은 CICS 프로그램 간에 데이터를 교환하기 위한 표준 메커니즘입니다. **LINK**, **START**, **XCTL**, **RETURN** 명령에 채널을 전달할 수 있습니다. DPL(Distributed Program Link)이 지원되며 **START CHANNEL** 및 **RETURN TRANSID** 명령으로 시작한 트랜잭션은 원격일 수 있습니다.
- 트랜잭션 채널, DFHTRANSACTION은 링크 레벨이 변경될 때조차도 범위에 남아 있으므로 트랜잭션의 프로그램 간에 데이터를 교환하는 용도로 이 채널을 사용할 수 있습니다.

- CICS가 지원하는 언어로 쓴 어느 CICS 애플리케이션이나 채널을 사용할 수 있습니다. 예를 들어, CICS 리전의 Java 클라이언트 프로그램은 채널을 사용하여 백엔드 AOR의 COBOL 서버 프로그램과 데이터를 교환할 수 있습니다.
- 여러 채널을 처리할 서버 프로그램을 쓸 수 있습니다. 이 프로그램은 예를 들어, 다음을 수행할 수 있습니다.
 - 호출된 채널을 동적을 발견합니다.
 - 채널의 컨테이너를 계수하고 찾습니다.
 - 전달된 채널에 따라 처리를 다양화합니다.
- 하나 이상의 채널을 통해 호출된 관련 프로그램 세트에서 『구성요소』를 빌드할 수 있습니다.
- 클라이언트와 구성요소 간의 느슨한 결합으로 인해 전개가 간편합니다. 클라이언트와 구성요소는 각기 다른 시기에 업그레이드할 수 있습니다. 예를 들어, 먼저 구성요소를 새 채널을 처리하도록 업그레이드한 후 클라이언트 프로그램을 새 채널을 사용하도록 업그레이드할 수 있습니다(또는 새 클라이언트를 쓸 수 있음).
- 프로그래머의 저장영역 관리 문제가 경감됩니다. CICS는 범위를 벗어날 때 컨테이너(및 해당 저장영역)를 자동으로 제거합니다. 프로그램은 범위를 벗어나기 전에 개별 컨테이너나 전체 채널 및 채널의 모든 컨테이너를 삭제할 명령을 실행할 수도 있습니다.
- 채널 애플리케이션에 사용하는 데이터 변환 모델이 단순하며 애플리케이션 개발자가 단순 API 명령을 사용하여 이 모델을 제어합니다.
- CICS 비즈니스 트랜잭션 서비스(BTS)에 대한 경험이 있는 프로그래머는 비BTS 애플리케이션에서 간편하게 컨테이너를 사용할 수 있습니다.
- 컨테이너를 사용하는 프로그램은 채널과 BTS 애플리케이션 모두에서 호출할 수 있습니다.
- 컨테이너를 사용하는 비BTS 애플리케이션은 전체 BTS 애플리케이션으로 마이그레이션할 수 있습니다. (BTS로의 마이그레이션 경로를 형성합니다.)

채널이 모든 경우에 최상의 솔루션인 것만은 아닙니다. 애플리케이션을 설계하는 경우 채널 사용으로 인한 하나 또는 두 가지 영향을 고려해야 합니다.

- 원격 프로그램 또는 트랜잭션에 채널을 전달할 때 특히 로컬 및 원격 리전이 MRO가 아닌 ISC 연결로 연결되면 많은 양의 데이터 전달 시 성능에 영향을 미칠 수 있습니다.
- 채널은 둘 이상의 위치에 컨테이너 데이터를 보유할 수 있기 때문에 동일한 데이터를 전달하도록 지정된 COMMAREA보다 더 많은 저장영역을 사용할 수 있습니다. COMMAREA는 포인터로 액세스하는 반면, 컨테이너의 데이터는 프로그램 간에 복사됩니다.

채널 작성

많은 API 명령 중 하나에 이름을 지정해서 채널을 작성할 수 있습니다. 현재 프로그램 범위에 채널이 없는 경우 CICS가 채널을 작성합니다.

이 태스크 정보

다음 명령 중 하나에 이름을 지정해서 채널을 작성합니다.

```
LINK PROGRAM CHANNEL
MOVE CONTAINER CHANNEL TOCHANNEL
PUT CONTAINER CHANNEL
PUT64 CONTAINER
RETURN TRANSID CHANNEL
START TRANSID CHANNEL
XCTL PROGRAM CHANNEL
WEB RECEIVE TOCHANNEL
WEB CONVERSE TOCHANNEL
```

채널을 작성하고 데이터의 컨테이너로 채널을 채우는 가장 간단한 방법은 **EXEC CICS PUT CONTAINER(container-name) CHANNEL(channel-name) FROM(data_area)** 명령을 연속해서 실행하는 방법입니다. 첫 번째 PUT 명령은 채널을 작성해서(존재하지 않을 경우) 컨테이너에 추가하고, 다음 명령은 추가 컨테이너를 채널에 추가합니다. 컨테이너가 존재하면 내용을 새 데이터로 겹쳐씹니다.

AMODE(64) 프로그램은 **EXEC CICS PUT64 CONTAINER** 명령을 사용하여 이러한 방식으로 채널을 작성 및 사용할 수 있습니다. 이 명령은 비Language Environment(LE) AMODE(64) 어셈블러 언어 애플리케이션에만 사용됩니다.

참고: 이름이 DFHTRANSACTION인 트랜잭션 채널을 제외한 새 채널은 링크 레벨이 변경될 때까지 이 범위에 남습니다. 트랜잭션 채널은 링크 레벨이 변경될 때 범위를 벗어나지 않으며 트랜잭션에서 항상 액세스 가능합니다. 채널 범위에 대한 자세한 정보는 338 페이지의 『채널의 범위』의 내용을 참조하십시오.

CICS가 API의 일부로 정의하는 문자를 제외하고, *DFH* 문자로 시작하는 이름을 채널에 지정하지 마십시오. 이 채널은 해당 API에 설명된 용도로만 사용해야 합니다.

채널에 컨테이너를 추가하는 대체 방법은 또 다른 채널로부터 컨테이너를 이동시키는 것입니다. 이를 위해서는 다음 명령을 사용하십시오.

```
EXEC CICS MOVE CONTAINER(container-name) AS(container-new-name)
CHANNEL(channel-name1) TOCHANNEL(channel-name2)
```

참고:

1. CHANNEL 또는 TOCHANNEL 옵션을 지정하지 않을 경우 현재 채널이 내포됩니다.
2. 소스 채널은 프로그램 범위에 있어야 합니다.

3. 대상 채널이 현재 프로그램 범위에 없는 경우 작성됩니다.
4. 소스 컨테이너가 없는 경우 오류가 발생합니다.
5. 대상 컨테이너가 없을 경우 작성되고 존재하는 경우에는 내용을 겹쳐씹니다.
6. 채널은 작성되면 범위를 벗어날 때까지 존재합니다. 연관된 저장영역인 컨테이너나 설정된 저장영역도 채널이 범위를 벗어날 때까지 존재합니다.

채널 간에 데이터를 전송하는 보다 효율적인 방법으로 GET CONTAINER 및 PUT CONTAINER 대신 MOVE CONTAINER를 사용할 수 있습니다.

다음 명령에 이름 지정된 채널이 현재 프로그램 범위에 없는 경우에는 비어 있는 채널이 작성됩니다.

- EXEC CICS LINK PROGRAM CHANNEL(channel-name)
- EXEC CICS RETURN TRANSID CHANNEL(channel-name)
- EXEC CICS START TRANSID CHANNEL(channel-name)
- EXEC CICS XCTL PROGRAM CHANNEL(channel-name)

현재 채널

프로그램의 현재 채널은 프로그램이 호출된 채널입니다(있는 경우). 이 예제는 프로그램 간에 현재 채널과 컨테이너가 전달되는 방식을 보여줍니다.

프로그램은 다른 채널을 작성할 수 있습니다. 하지만 특정 프로그램의 특정 호출의 경우에는 현재 채널이 변경되지 않습니다. 매개변수 목록과 비슷합니다.

LINK 명령을 사용한, 현재 채널 예제

이 예제는 EXEC CICS LINK 명령을 사용하여 현재 채널과 컨테이너를 또 다른 프로그램에 전달하는 방식을 보여줍니다.

다음 그림은 프로그램의 현재 채널의 기점을 설명합니다. 다섯 개의 대화식 프로그램을 표시합니다. 프로그램 A는 예를 들어, 터미널 사용자가 시작한 최상위 레벨 프로그램입니다. 프로그램을 통해 시작되지 않으며 현재 채널이 없습니다.

B, C, D, E는 두 번째, 세 번째, 네 번째, 다섯 번째 레벨 프로그램입니다.

프로그램 B의 현재 채널은 프로그램 A가 실행한 EXEC CICS LINK 명령의 CHANNEL 옵션을 통해 전달된 X입니다. 프로그램 B는 하나의 컨테이너를 추가하고 또 다른 컨테이너를 삭제해서 채널 X를 수정합니다.

프로그램 C의 현재 채널도 프로그램 B가 실행한 EXEC CICS LINK 명령의 CHANNEL 옵션을 통해 전달된 X입니다.

프로그램 D는 현재 채널이 없으며 이는 C가 채널을 전달하지 않기 때문입니다.

프로그램 E의 현재 채널은 D가 실행한 EXEC CICS LINK 명령의 CHANNEL 옵션을 통해 전달된 Y입니다.

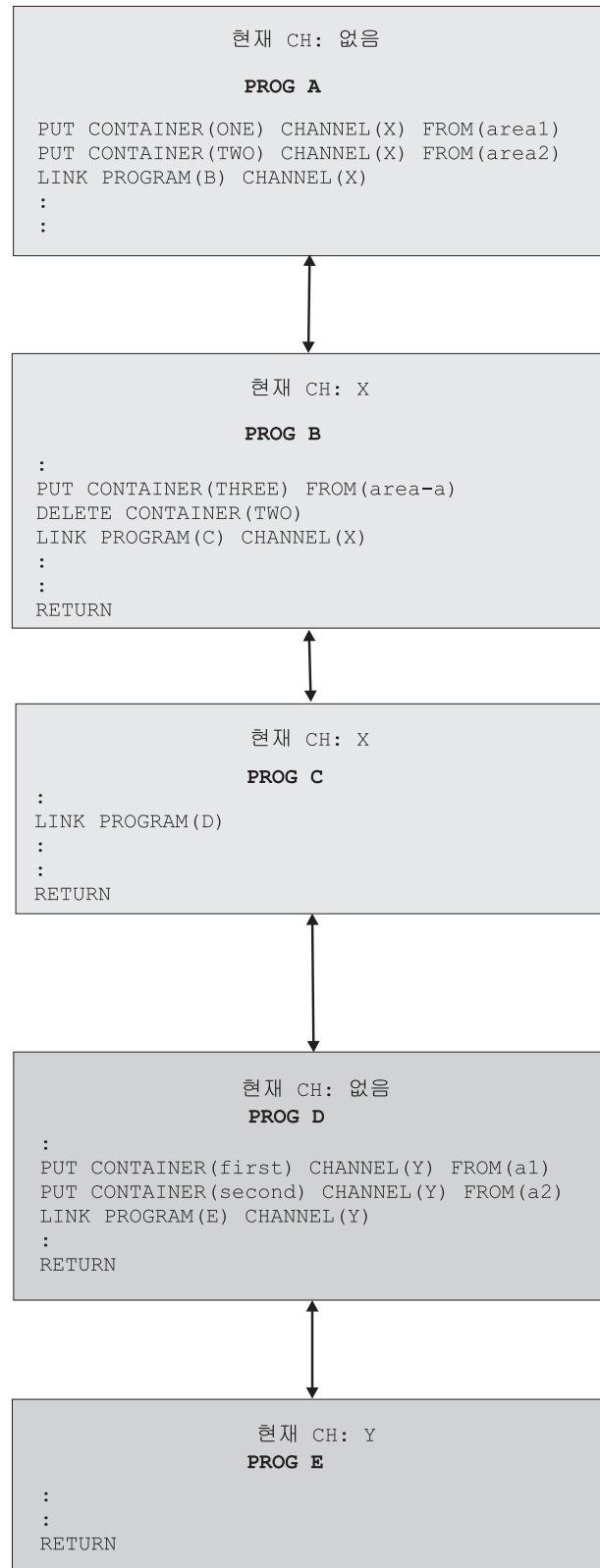


그림 71. 현재 채널: LINK 명령 사용 예제

다음 테이블에는 이전 그림에 표시된 5개 각 프로그램의 현재 채널(있는 경우) 이름이 나열됩니다.

표 17. LINK 명령을 사용한 대화식 프로그램 예제의 현재 채널

프로그램	현재 채널	실행 명령	설명
A	없음	<pre> . EXEC CICS PUT CONTAINER(ONE) CHANNEL(X) FROM(area1) EXEC CICS PUT CONTAINER(TWO) CHANNEL(X) FROM(area2) EXEC CICS LINK PROGRAM(B) CHANNEL(X) . </pre>	프로그램 A는 채널 X를 작성해서 프로그램 B에 전달합니다. 프로그램 B가 프로그램 A에 제어를 리턴할 때 X 채널은 이미 수정된 상태이기 때문에 이 채널이 프로그램 A에 의해 작성되었을 때와 같은 컨테이너 세트를 포함하지 않음을 참고하십시오. (프로그램 B가 TWO 컨테이너를 삭제하고 THREE 컨테이너를 추가했습니다.)
B	X	<pre> . EXEC CICS PUT CONTAINER(THREE) FROM(area-a) EXEC CICS DELETE CONTAINER(TWO) EXEC CICS LINK PROGRAM(C) CHANNEL(X) . . EXEC CICS RETURN </pre>	프로그램 B는 컨테이너를 추가 및 삭제해서 채널 X(현재 채널)를 수정하고 수정한 채널을 프로그램 C에 전달합니다. 프로그램 B는 PUT CONTAINER 및 DELETE CONTAINER 명령에 CHANNEL 옵션을 지정할 필요가 없습니다. 현재 채널이 내포되어 있습니다.
C	X	<pre> . EXEC CICS LINK PROGRAM(D) . . EXEC CICS RETURN </pre>	프로그램 C는 프로그램 D로 링크하지만 채널을 전달하지는 않습니다.
D	없음	<pre> . EXEC CICS PUT CONTAINER(first) CHANNEL(Y) FROM(a1) EXEC CICS PUT CONTAINER(second) CHANNEL(Y) FROM(a2) EXEC CICS LINK PROGRAM(E) CHANNEL(Y) . . EXEC CICS RETURN </pre>	프로그램 D는 프로그램 E에 전달할 새 채널, Y를 작성합니다.
E	Y	<pre> . RETURN . </pre>	프로그램 E는 전달된 데이터를 일부 처리한 후 리턴합니다.

XCTL 명령을 사용한, 현재 채널 예제

이 예제는 EXEC CICS XCTL 명령을 사용하여 현재 채널과 컨테이너를 또 다른 프로그램에 전달하는 방식을 보여줍니다.

337 페이지의 그림 72는 네 개의 대화식 프로그램을 보여줍니다. A1은 예를 들어, 터미널 사용자가 시작한 최상위 레벨 프로그램입니다. 프로그램을 통해 시작되지 않으며 현재 채널이 없습니다. B1, B2, B3은 모두 두 번째 레벨 프로그램입니다.

B1의 현재 채널은 A1이 실행한 EXEC CICS LINK 명령의 CHANNEL 옵션을 통해 전달된 X입니다.

B2는 현재 채널이 없으며 이는 B1이 채널을 전달하지 않기 때문입니다.

B3의 현재 채널은 B2가 실행한 EXEC CICS XCTL 명령의 CHANNEL 옵션을 통해 전달된 Y입니다.

B3이 리턴될 때 CICS는 채널 Y 및 해당 컨테이너를 삭제합니다.

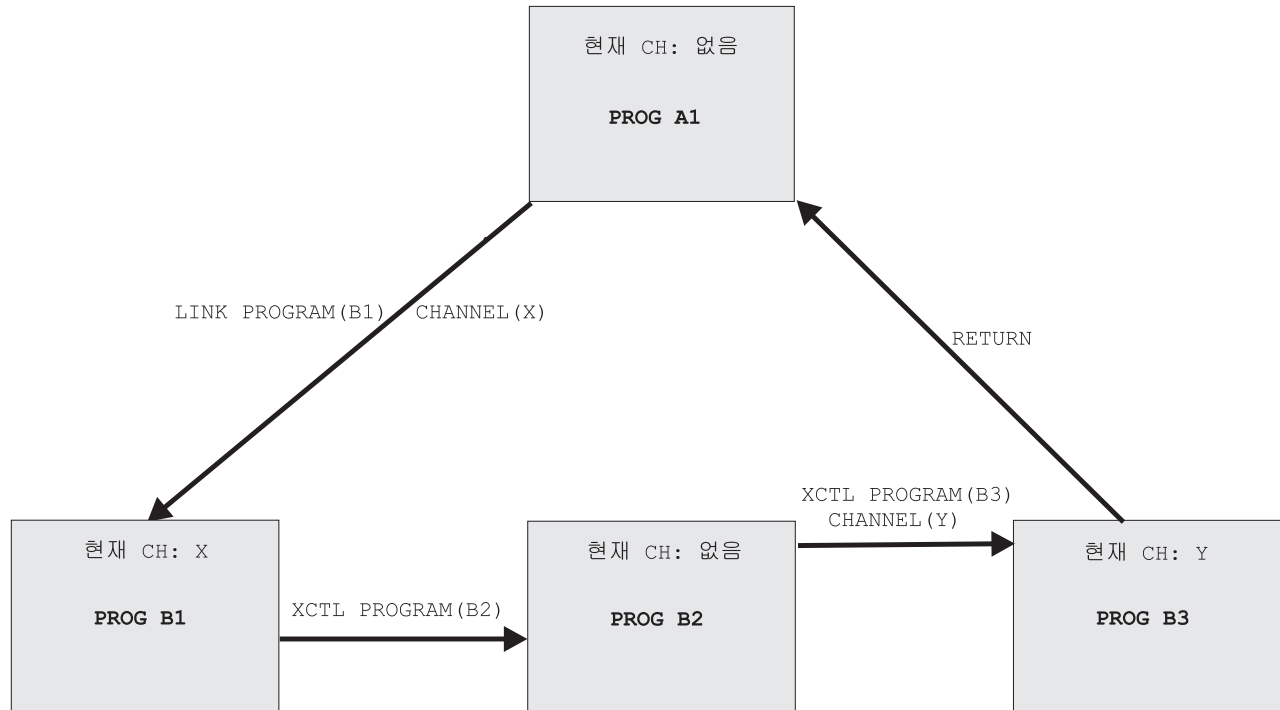


그림 72. XCTL 명령을 사용한 현재 채널 예제

다음 테이블은 그림 72에 표시된 각 4개 프로그램의 현재 채널(있는 경우) 이름을 나열합니다.

표 18. 대화식 프로그램의 현재 채널-예제

프로그램	현재 채널	명령 실행
A1	없음	. EXEC CICS LINK PROGRAM(B1) CHANNEL(X) .
B1	X	. EXEC CICS XCTL PROGRAM(B2) .
B2	없음	. EXEC CICS XCTL PROGRAM(B3) CHANNEL(Y) .
B3	Y	. EXEC CICS RETURN .

현재 채널: START 및 RETURN 명령

LINK 및 XCTL 명령 외에, START 및 RETURN 명령으로 채널을 전달할 수 있습니다.

EXEC CICS START TRANSID(*transid*) CHANNEL(*channel-name*)

시작된 트랜잭션(또는 둘 이상인 경우 첫 번째 프로그램)을 구현하는 프로그램에 채널이 전달되며 이 채널이 현재 채널이 됩니다.

참고: 채널을 지정하는 ALL EXEC CICS START 요청은 INTERVAL, AT, FOR 또는 UNTIL을 지정해서 지연시킬 수 없습니다(지원되지 않으므로). 이는 JCICS가 아닌 CICS 변환기로 모니터링합니다. 이 명령을 사용하여 EXEC CICS START 요청을 지연시키면 변환기가 실패합니다.

EXEC CICS RETURN TRANSID(*transid*) CHANNEL(*channel-name*)

CHANNEL 옵션은 다음의 경우에만 올바릅니다.

- 의사 대화식 RETURN에—즉, TRANSID 옵션에 사용자 터미널에서 실행할 다음 트랜잭션을 지정하는 RETURN 명령에 사용할 때.
- 최고 논리 레벨에서 프로그램을 실행한 경우—즉, CICS에 제어를 리턴하는 프로그램.

다음 트랜잭션을 구현하는 프로그램에 채널이 제공되고 이 채널이 현재 채널이 됩니다.

채널의 범위

채널의 범위는 채널에 액세스할 수 있는 코드입니다. 다음 예제는 다이어그램으로 각 채널의 범위를 보여줍니다.

범위는 채널 및 컨테이너 저장영역의 수명을 정의하기 때문에 중요합니다. 자세한 정보는 344 페이지의 『채널과 컨테이너 삭제 및 해당 저장영역의 사용 기능화』의 내용을 참조하십시오.

트랜잭션 채널 DFHTRANSACTION

채널은 일반적으로 링크 레벨이 변경될 때 범위를 벗어납니다. 따라서 트랜잭션의 모든 프로그램에 대해 사용 가능하지 않을 수 있습니다. 이름이 DFHTRANSACTION인 채널을 작성하는 경우 이 채널은 링크 레벨이 변경될 때 범위를 벗어나지 않습니다. 이 채널은 API가 사용 설정된 종료 코드를 포함하여 트랜잭션에 있는 모든 프로그램에 사용 가능합니다. 그러나 5.2 이전 버전에서는 트랜잭션 채널을 CICS 리전에 전달할 수 없습니다.

DFHTRANSACTION은 채널 이름을 승인하는 모든 API 명령에서 사용할 수 있습니다. 그러나 프로그램은 **EXEC CICS DELETE CHANNEL** 명령을 사용하여 DFHTRANSACTION 채널을 삭제할 수 없습니다.

LINK 명령이 포함된 범위 예제

이 예제에서는 『LINK 명령이 포함된 현재 채널 예제』의 다이어그램을 추가하여 각 채널의 범위를 보여줍니다.

LINK 명령에 대한 자세한 정보는 333 페이지의 『LINK 명령을 사용한, 현재 채널 예제』의 내용을 참조하십시오.

다음 그림은 이전에 설명한 5개의 동일한 대화식 프로그램을 보여줍니다.

X 채널(엑세스를 시작할 수 있는 코드)의 범위는 프로그램 A, B 및 C입니다.

Y 채널의 범위는 프로그램 D 및 E입니다.

이러한 채널 중 어느 채널도 트랜잭션 채널인 DFHTRANSACTION이 아니며 이 채널의 범위는 전체 트랜잭션일 수 있습니다.

프로그램 B가 프로그램 A에 제어를 리턴할 때 X 채널은 이미 수정된 상태이기 때문에 이 채널이 프로그램 A에 의해 작성되었을 때와 같은 컨테이너 세트를 포함하지 않음을 참고하십시오.

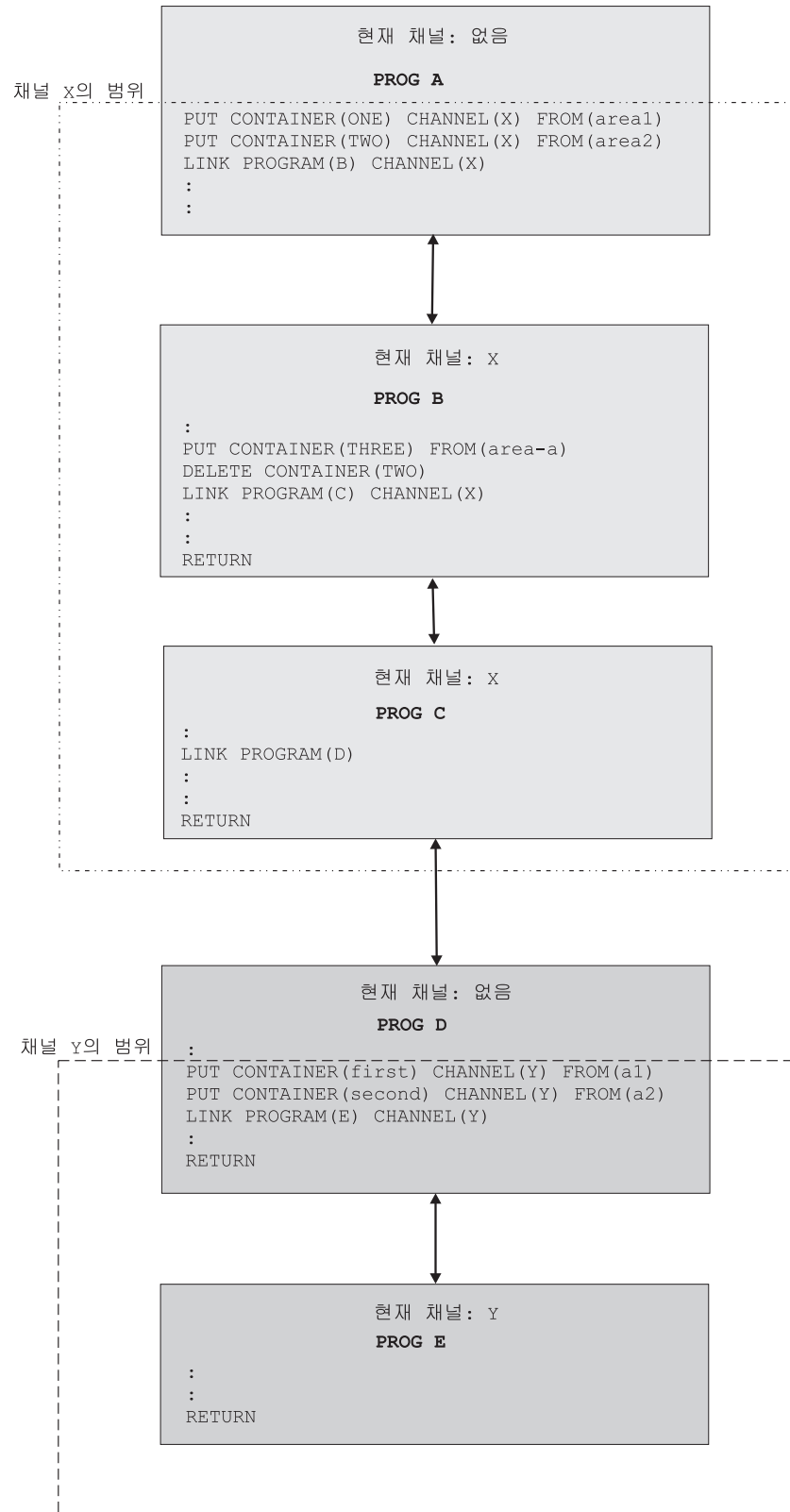


그림 73. 채널의 범위-LINK 명령을 보여주는 예제

다음 테이블은 이전 그림에서 표시된 다섯 개의 프로그램 각각의 현재 채널(있는 경우)의 이름 및 범위를 나열합니다.

표 19. 채널의 범위—LINK 명령 포함 예제

프로그램	현재 채널	채널의 범위
A	없음	해당사항 없음
B	X	A, B, C
C	X	A, B, C
D	없음	해당사항 없음
E	Y	D, E

LINK 및 XCTL 명령이 포함된 범위 예제

이 예제에서는 『XCTL 명령이 포함된 현재 채널 예제』의 다이어그램을 추가하여 각 채널의 범위를 보여줍니다.

XCTL 명령에 대한 자세한 정보는 336 페이지의 『XCTL 명령을 사용한, 현재 채널 예제』의 내용을 참조하십시오.

342 페이지의 그림 74에서는 이전에 설명한 네 개의 동일한 대화식 프로그램과 프로그램 B1에서 EXEC CICS LINK 명령으로 호출되는 세 번째 레벨 프로그램 C1을 보여줍니다.

X 채널의 범위는 A1과 B1로 제한됩니다.

Y 채널의 범위는 프로그램 B2 및 B3입니다.

Z 채널의 범위는 프로그램 B1 및 C1입니다.

이러한 채널 중 어느 채널도 트랜잭션 채널인 DFHTRANSACTION이 아니며 이 채널의 범위는 전체 트랜잭션일 수 있습니다.

프로그램 B3이 프로그램 A1에 제어를 리턴할 때 X 채널은 프로그램 B1에 의해 이미 수정되었을 수 있고 이 채널은 프로그램 A1에 의해 작성되었을 때와 같은 컨테이너 세트를 포함하지 않을 수 있음을 참고하십시오.

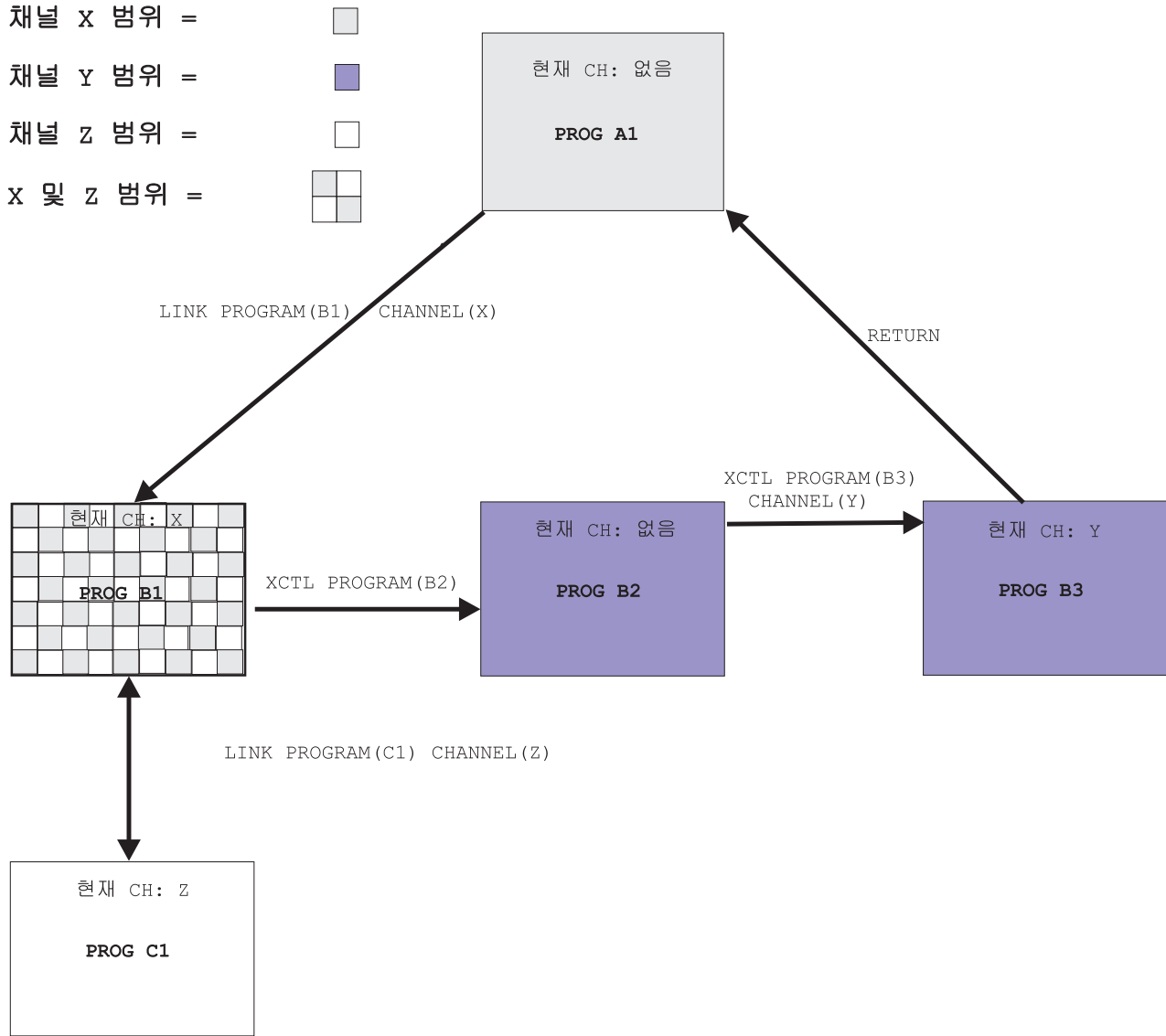


그림 74. 채널의 범위-LINK 및 XCTL 명령을 보여주는 예제

다음 테이블은 그림 74에서 표시된 다섯 개의 프로그램 각각의 현재 채널(있는 경우)의 이름 및 범위를 나열합니다.

표 20. 채널의 범위-LINK 및 XCTL 명령을 포함하는 예제

프로그램	현재 채널	채널의 범위
A1	없음	해당사항 없음
B1	X	A1 및 B1
B2	없음	해당사항 없음
B3	Y	B2 및 B3
C1	Z	B1 및 C1

프로그램에 전달된 컨테이너 발견

프로그램이 호출되면 프로그램에 전달된 컨테이너 및 채널의 이름을 판별할 수 있습니다.

일반적으로 채널을 교환하는 프로그램은 해당 채널을 처리하기 위해 작성됩니다. 즉, 클라이언트와 서버 프로그램 둘 다 채널의 이름 및 채널에 있는 컨테이너의 이름과 수를 압니다. 그러나 예를 들어, 서버 프로그램 또는 구성요소가 둘 이상의 채널을 처리하기 위해 작성되는 경우 호출 시 이 서버 프로그램 또는 구성요소에 전달된 채널을 발견해야 합니다.

프로그램은 EXEC CICS ASSIGN CHANNEL 명령을 실행하여 현재 채널(즉, 프로그램 호출에 사용된 채널)을 발견할 수 있습니다(현재 채널이 없는 경우 명령은 공백을 리턴함).

프로그램은 EXEC CICS QUERY CHANNEL 명령을 실행하여 현재 채널에 있는 컨테이너의 수를 계수할 수 있습니다. 이 명령은 모든 채널에서 사용할 수 있기 때문에 프로그램은 먼저 현재 채널의 이름을 발견하고 명령에서 이 이름을 지정해야 합니다. 프로그램은 찾아보기를 통해 현재 채널에 있는 컨테이너의 이름을 가져올 수도 있습니다. 일반적으로 이러한 조작은 필요하지 않습니다. 여러 채널을 처리할 수 있도록 작성된 프로그램은 종종 가능한 각 채널 내 컨테이너의 이름과 수를 인식할 수 있도록 코드화됩니다.

현재 채널에 있는 컨테이너의 이름을 가져오려면 다음과 같은 찾아보기 명령을 사용하십시오.

- EXEC CICS STARTBROWSE CONTAINER BROWSETOKEN(*data-area*) .
- EXEC CICS GETNEXT CONTAINER(*data-area*) BROWSETOKEN(*token*).
- EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(*token*).

현재 채널의 이름 및 필요에 따라 채널에 있는 컨테이너의 수 또는 이름을 검색한 경우 서버 프로그램은 전달된 데이터의 유형에 맞도록 그 처리를 조정할 수 있습니다.

링크에서 리턴된 컨테이너 발견

LINK 명령에 따라 프로그램은 링크된 프로그램에서 리턴된 컨테이너의 이름을 발견할 수 있습니다.

프로그램은 EXEC CICS LINK PROGRAM(*program-name*) CHANNEL(*channel-name*) 명령을 통해 두 번째 프로그램으로 전달하는 채널을 작성합니다. 두 번째 프로그램은 전달된 데이터에 대한 일부 처리를 수행하고 같은 채널(현재 채널)로 결과를 리턴합니다. 리턴 시 첫 번째 프로그램은 리턴된 채널의 이름은 알지만 채널에 있는 컨테이너의 수 및 이름까지 아는 것은 아닙니다. (리턴된 채널이 전달된 채널과 동일한 컨테이너를 포함합니까? 또는 두 번째 프로그램이 일부 채널을 삭제하거나 다른 채널을 작성했습니까?)

첫 번째 프로그램은 EXEC CICS QUERY CHANNEL 명령을 실행하고 현재 채널의 이름을 지정하여 채널에 있는 컨테이너의 수를 계수할 수 있습니다. 첫 번째 프로그램은 찾아보기를 통해 컨테이너 이름을 발견할 수도 있습니다. 이를 수행하기 위해 다음과 같은 명령을 사용합니다.

- EXEC CICS STARTBROWSE CONTAINER BROWSETOKEN(*data-area*) CHANNEL(*channel-name*).
- EXEC CICS GETNEXT CONTAINER(*data-area*) BROWSETOKEN(*token*).
- EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(*token*).

채널과 컨테이너 삭제 및 해당 저장영역의 사용 기능화

컨테이너는 대용량 오브젝트이기 때문에 CICS 리전의 저장영역이 부족해지지 않도록 이러한 오브젝트의 저장영역을 올바르게 관리하는 것이 중요합니다. CICS는 애플리케이션에 액세스 권한은 있지만 더 이상 사용할 수 없는 컨테이너 및 채널을 자동 삭제합니다. 애플리케이션은 명령을 실행하여 개별 컨테이너 또는 전체 채널을 삭제할 수도 있습니다.

애플리케이션이 명령에서 처음으로 이름을 지정하여 채널을 작성하고 작성한 채널에 컨테이너를 추가할 때 CICS는 컨테이너에 보관되는 데이터에 대해 64비트 저장영역(2GB 초과)을 사용합니다. 채널 및 컨테이너에 64비트 저장영역을 사용하면 CICS 리전에 적용되는 z/OS **MEMLIMIT** 매개변수에 대해 선택하는 값에 영향을 미칩니다. 64비트 저장영역을 사용하는 다른 CICS 기능도 고려해야 합니다. 자세한 정보는 성능 개선에서 **MEMLIMIT** 추정, 검사 및 설정의 내용을 참조하십시오.

애플리케이션이 **SET** 옵션이 포함된 **GET CONTAINER** 명령을 발행하면 CICS가 프로그램 자원 정의의 **DATALOCATION** 설정에 따라 24비트 저장영역(16MB 미만) 또는 31비트 저장영역(16MB 초과 2GB 미만) 중 한 태스크 저장영역에서 컨테이너 사본을 작성합니다. 애플리케이션은 컨테이너의 내용에 대해 작업한 다음 **PUT CONTAINER** 명령을 사용하여 이 콘텐츠를 다시 64비트 저장영역(2GB 초과)에 복사할 수 있습니다. Language Environment(LE) AMODE(64) 외의 어셈블리 언어 애플리케이션은 **PUT64 CONTAINER** 및 **GET64 CONTAINER** 명령을 사용하여 64비트 저장영역에서 채널 및 컨테이너에 대해 직접 작업할 수 있습니다.

다른 애플리케이션에 사용 가능한 저장영역 한계를 초과하는 너무 큰 컨테이너를 작성하지 않도록 주의하십시오. 대용량 데이터를 전달해야 하는 경우 해당 데이터를 한 개의 컨테이너에 넣는 대신 여러 컨테이너로 분할하여 넣으십시오. 컨테이너를 작성할 때 CICS는 이 조치로 인해 트랜잭션에 할당된 64비트 저장영역이 **MEMLIMIT** 값의 5%를 초과하게 되는지 확인합니다. 이 상황에서 CICS는 경고 메시지 DFHPG0400을 발행하고 컨테이너를 작성하지 않습니다.

컨테이너 삭제 및 해당 저장영역의 사용 가능화

CICS는 채널이 범위를 벗어나면 채널에 있는 모든 컨테이너의 64비트 저장영역(2GB 초과)을 자동으로 사용 가능화합니다. 채널의 범위는 채널을 작성한 프로그램 및 채널이 전달된 프로그램을 포함하여 채널에 액세스할 수 있는 코드입니다. 예외는 트랜잭션 채널인 DFHTRANSACTION이며 이 채널은 트랜잭션 내 모든 프로그램에 사용 가능하고 링크 레벨이 변경될 때 범위를 벗어나지 않습니다. 채널 범위에 대한 자세한 정보는 338 페이지의 『채널의 범위』의 내용을 참조하십시오.

채널이 범위를 벗어나기 전에 컨테이너가 삭제되게 하려는 경우 애플리케이션에서 이름 지정된 컨테이너를 명시적으로 삭제하는 **DELETE CONTAINER** 명령을 실행할 수 있습니다. 이 명령이 실행되면 채널에 포함된 모든 데이터가 버려지고 컨테이너의 사용된 64비트 저장영역은 사용 가능해집니다. 한 채널을 소유하는 애플리케이션은 **DELETE CHANNEL** 명령을 실행하여 채널에 있는 모든 컨테이너를 삭제할 수도 있습니다.

프로그램이 **GET CONTAINER** 명령을 실행하고 CICS가 사용할 프로그램의 컨테이너 사본을 작성하는 경우 다음 중 어느 한 상황이 발생할 때까지 CICS는 컨테이너 사본을 24비트 저장영역 또는 31비트 저장영역에서 유지보수합니다.

- 이 저장영역에 액세스할 수 있는 프로그램이 동일한 채널의 동일한 컨테이너에 대해 SET 옵션과 함께 후속 **GET CONTAINER** 또는 **GET64 CONTAINER** 명령을 실행합니다.
- 컨테이너가 **DELETE CONTAINER** 명령으로 삭제됩니다.
- 컨테이너가 **MOVE CONTAINER** 명령으로 이동됩니다.
- 채널이 프로그램 범위를 벗어납니다.
- 채널 및 채널의 컨테이너가 **DELETE CHANNEL** 명령으로 삭제됩니다.

이러한 이벤트 중 하나가 발생하면 24비트 저장영역 또는 31비트 저장영역에서 컨테이너의 사본이 삭제됩니다.

채널 삭제 및 해당 저장영역 사용 가능화

CICS는 채널이 범위를 벗어나면 채널에 있는 모든 컨테이너의 64비트 저장영역(2GB 초과)을 자동으로 사용 가능화합니다. 이때 24비트 저장영역 또는 31비트 저장영역에 여전히 존재하는 채널의 컨테이너 사본도 삭제됩니다.

채널이 범위를 벗어나기 전에 전체 채널을 삭제하려는 경우 애플리케이션에서 **DELETE CHANNEL** 명령을 실행할 수 있습니다. 이 명령은 한 번의 조작으로 채널에 남아 있는 모든 컨테이너를 삭제합니다. 채널 및 채널의 남아 있는 컨테이너에 사용되는 모든 64비트 저장영역이 사용 가능해지고 24비트 저장영역 또는 31비트 저장영역에 여전히 존재하는 채널의 컨테이너 사본도 삭제됩니다.

DELETE CHANNEL 명령을 실행하는 애플리케이션은 채널을 소유한 프로그램이어야 합니다. 채널을 소유한 프로그램은 다음 명령 중 하나에 이름을 지정해서 채널을 작성한 프로그램입니다.

- **LINK PROGRAM CHANNEL**
- **MOVE CONTAINER CHANNEL TOCHANNEL**
- **PUT CONTAINER CHANNEL**
- **PUT64 CONTAINER**
- **RETURN TRANSID CHANNEL**
- **START TRANSID CHANNEL**
- **XCTL PROGRAM CHANNEL**
- **WEB RECEIVE TOCHANNEL**
- **WEB CONVERSE TOCHANNEL**

애플리케이션은 다음 채널을 삭제할 수 없습니다.

- 애플리케이션의 현재 채널 즉, 프로그램이 호출된 채널.
- 애플리케이션이 작성하지 않은 채널.
- 읽기 전용 채널.
- 트랜잭션 채널 DFHTRANSACTION.

채널 설계: 우수 사례

컨테이너는 프로그램 간에 정보를 전달하는 용도로 사용됩니다. 이 컨테이너는 채널이라는 세트로 그룹화됩니다. 일련의 우수 사례에 따라 채널을 설계할 것을 권장합니다.

프로시저

- DPL 호출 종료 시 서버 프로그램이 변경하지 않은 입력 컨테이너는 클라이언트로 리턴되지 않습니다. 서버 프로그램이 내용을 변경한 입력 컨테이너와 서버 프로그램이 작성한 컨테이너가 리턴됩니다. 따라서 최적의 DPL 성능을 위해서는 다음 우수 사례를 사용하십시오.
 - 입력 및 출력 데이터에 별도의 컨테이너를 사용하십시오.
 - 클라이언트가 아닌 서버 프로그램이 출력 컨테이너를 작성하는지 확인하십시오.
 - 읽기 전용 및 읽기/쓰기 데이터에 별도의 컨테이너를 사용하십시오.
 - 구조가 선택적이면 개별 컨테이너로 작성하십시오.
 - 오류 정보에는 전용 컨테이너를 사용하십시오.
- 채널 지정에 관한 다음 일반 팁은 최적의 DPL 성능을 위한 권장사항을 포함하고 있으며 이를 기반으로 확장합니다.
 - 입력 및 출력 데이터에 별도의 컨테이너를 사용하십시오. 다음과 같은 혜택이 있습니다.

- 프로그램을 보다 간편하게 유지보수할 수 있도록 데이터를 제대로 캡슐화합니다.
- 더 작은 컨테이너가 각 방향으로 플로우되기 때문에 DPL 호출에 채널이 전달될 때 효율성이 증가합니다.
- 클라이언트가 아닌 서버 프로그램이 출력 컨테이너를 작성하는지 확인하십시오. 클라이언트가 작성하면 비어 있는 컨테이너가 서버 리전에 전송됩니다.
- 읽기 전용 및 읽기/쓰기 데이터에 별도의 컨테이너를 사용하십시오. 다음과 같은 혜택이 있습니다.
 - 프로그램을 보다 쉽게 이해할 수 있어서 카피복 구조가 단순화됩니다.
 - REORDER 오버레이에 관한 문제를 피할 수 있습니다.
 - 서버 리전으로 전송한 읽기 전용 컨테이너가 리턴되지 않으므로 CICS 리전 간의 전송 효율성이 향상됩니다.
- 각 구조마다 별도의 컨테이너를 사용하십시오. 다음과 같은 혜택이 있습니다.
 - 프로그램을 보다 간편하게 이해하고 유지보수할 수 있도록 데이터를 제대로 캡슐화합니다.
 - 전체 구성요소를 다시 컴파일하지 않아도 되기 때문에 구조 중 하나를 훨씬 간편하게 변경할 수 있습니다.
 - **MOVE CONTAINER** 명령을 사용하여 채널 간에 컨테이너를 이동시킴으로써 채널의 서브세트를 하위 구성요소에 전달할 수 있습니다.
- 구조가 선택적이면 개별 컨테이너로 작성하십시오. 그러면 컨테이너가 존재할 경우에만 구조가 전달되므로 효율성이 향상됩니다.
- 오류 정보에는 전용 컨테이너를 사용하십시오. 다음과 같은 혜택이 있습니다.
 - 오류 정보를 더 간편하게 식별할 수 있습니다.
 - 다음과 같은 이유로 효율성이 향상됩니다.
 - 오류가 발생할 경우에만 오류 정보를 포함한 구조가 다시 전달됩니다.
 - 채널에서 컨테이너 찾아보기를 시작하는 대신 **GET CONTAINER**(known-error-container-name) 또는 **GET64 CONTAINER**(known-error-container-name) 명령을 실행하여(그리고 아마도 NOTFOUND 조건을 수신하여) 오류 컨테이너의 존재를 확인하는 것이 훨씬 더 효율적입니다.
- code page1의 문자 데이터와 code page2의 문자 데이터처럼 여러 다른 유형의 데이터를 전달해야 하는 경우에는 복잡한 구조의 한 컨테이너보다는 각 구조마다 별도의 컨테이너를 사용하십시오. 그러면 여러 다른 코드 페이지 간의 이동성이 개선됩니다.
- 대용량 데이터를 전달해야 하는 경우 해당 데이터를 한 개의 컨테이너에 넣는 대신 여러 컨테이너로 분할하여 넣으십시오.

원격 프로그램 또는 트랜잭션으로 채널이 전달될 때 많은 양의 데이터를 전달하면 성능에 영향이 미칠 수 있습니다. 이는 특히, 로컬 및 원격 리전이 MRO가 아닌 ISC 연결로 연결된 경우에 더 그러합니다.

경고: 다른 애플리케이션에 사용 가능한 저장영역 한계를 초과하는 너무 큰 컨테이너를 작성하지 않도록 주의하십시오.

- 채널 및 컨테이너는 2GB 미만(기준 아래) 및 64비트 저장영역(기준 위)을 사용합니다. 저장영역의 사용과 해제에 대한 정보는 344 페이지의 『채널과 컨테이너 삭제 및 해당 저장영역의 사용 가능화』의 내용을 참조하십시오. 채널 및 컨테이너에 64비트 저장영역을 사용하면 CICS 리전에 적용되는 z/OS MEMLIMIT 매개변수에 대해 선택하는 값에 영향을 미칩니다. 64비트 저장영역을 사용하는 다른 CICS 기능도 고려해야 합니다. 자세한 정보는 성능 개선에서 MEMLIMIT 추정, 검사 및 설정의 내용을 참조하십시오.

관련 개념:

360 페이지의 『COMMAREA에서 채널로 마이그레이션』

데이터를 교환하기 위해 기존 통신 영역을 사용하는 CICS 애플리케이션은 전과 같이 계속해서 작업할 수 있습니다. 채널을 마이그레이션하려는 경우 다음은 COMMAREA를 사용하지 않고 채널 및 컨테이너를 사용하기 위해 기존 애플리케이션의 여러 유형을 마이그레이션하는 방법의 예제입니다.

관련 참조:

330 페이지의 『채널의 이점』

CICS 프로그램에서 채널과 컨테이너 모델을 사용하여 데이터를 교환하는 방식은 통신 영역 사용에 비해 여러 장점이 있습니다.

채널 구성 및 사용: 예제

이 예제에서, 클라이언트 프로그램은 채널을 구성하여 서버 프로그램으로 전달하고 서버의 출력을 검색합니다. 서버 프로그램은 채널의 컨테이너에서 데이터를 검색하고 클라이언트에 출력을 리턴합니다.

349 페이지의 그림 75는 다음과 같은 CICS 클라이언트 프로그램을 보여줍니다.

1. EXEC CICS PUT CONTAINER 명령을 사용하여 컨테이너 세트를 구성(및 데이터를 넣음)합니다. 컨테이너는 동일하게 이름 지정된 channel—『payroll-2004』의 모든 파트입니다.
2. EXEC CICS LINK 명령을 실행하여 PAYR 서버 프로그램을 호출하고 payroll-2004 채널을 전달합니다.
3. EXEC CICS GET CONTAINER 명령을 실행하여 payroll-2004 채널의 status 컨테이너에 있음을 알고 있는 서버의 프로그램 출력을 검색합니다.

```
* create the employee container on the payroll-2004 channel
EXEC CICS PUT CONTAINER('employee') CHANNEL('payroll-2004') FROM('John Doe')

* create the wage container on the payroll-2004 channel
EXEC CICS PUT CONTAINER('wage') CHANNEL('payroll-2004') FROM('100')

* invoke the payroll service, passing the payroll-2004 channel
EXEC CICS LINK PROGRAM('PAYR') CHANNEL('payroll-2004')

* examine the status returned on the payroll-2004 channel
EXEC CICS GET CONTAINER('status') CHANNEL('payroll-2004') INTO(stat)
```

그림 75. 클라이언트 프로그램의 채널 구성, 서버 프로그램에 채널 전달, 서버의 출력 검색 방식

그림 76은 클라이언트가 호출한 PAYR 서버 프로그램의 일부를 보여줍니다. 서버 프로그램은 다음을 수행합니다.

1. 호출된 채널을 조회합니다.
2. EXEC CICS GET CONTAINER 명령을 실행하여 payroll-2004 채널의 employee 및 wage 컨테이너에서 입력을 검색합니다.
3. 입력 데이터를 처리합니다.
4. EXEC CICS PUT CONTAINER 명령을 실행하여 payroll-2004 채널의 status 컨테이너에 출력을 리턴합니다.

"PAYR", CICS COBOL server program

```
* discover which channel I've been invoked with
EXEC CICS ASSIGN CHANNEL(ch_name)
:
WHEN ch_name 'payroll-2004'
  * my current channel is "payroll-2004"
  * get the employee passed into this program
  EXEC CICS GET CONTAINER('employee') INTO(emp)
  * get the wage for this employee
  EXEC CICS GET CONTAINER('wage') INTO(wge)
  :
  * process the input data
  :
  :
  * return the status to the caller by creating the status container
  * on the payroll channel and putting a value in it
  EXEC CICS PUT CONTAINER('status') FROM('OK')
  :
  :
WHEN ch_name 'payroll-2005'
  * my current channel is "payroll-2005"
  :
  :
  :
```

그림 76. 서버 프로그램이 전달된 채널을 조회하고, 채널의 컨테이너에서 데이터를 검색하며, 호출자에게 출력을 리턴하는 방식

CICS 읽기 전용 컨테이너

CICS는 자체 용도로 채널 및 컨테이너를 작성하여 사용자 프로그램에 전달할 수 있습니다. 일부 경우에 사용자 프로그램에서 리턴 시 CICS에 필요한 데이터를 사용자 프로그램이 수정할 수 없도록 CICS는 이러한 컨테이너를 읽기 전용으로 표시합니다.

사용자 프로그램은 읽기 전용 컨테이너를 작성할 수 없습니다.

읽기 전용 컨테이너는 겹쳐쓰기, 이동 또는 삭제할 수 없습니다. **PUT CONTAINER**, **PUT64 CONTAINER**, **MOVE CONTAINER** 또는 **DELETE CONTAINER** 명령에서 읽기 전용 컨테이너를 지정하는 경우 **INVREQ** 조건이 발생합니다. 프로그램은 **EXEC CICS DELETE CHANNEL** 명령을 사용하여 읽기 전용 채널을 삭제할 수 없습니다.

JCICS에서 채널 사용

CICS는 CICS Java 프로그램이 채널을 전달하고 수신하는 데 사용할 수 있는 JCICS 클래스를 제공합니다.

JCICS의 채널 사용에 대한 정보는 *CICS의 Java 애플리케이션*에서 채널 및 컨테이너를 참조하십시오.

채널 및 BTS 활동

빌드 및 채널과의 상호작용에 사용되는 **PUT**, **GET**, **MOVE**, **DELETE CONTAINER** 명령은 CICS 비즈니스 트랜잭션 서비스(BTS) 애플리케이션에 사용된 명령과 유사합니다.

따라서 BTS 경험이 있는 프로그래머는 비BTS 애플리케이션에서 쉽게 컨테이너를 사용할 수 있음을 발견합니다. 또한 컨테이너를 사용하는 서버 프로그램을 채널 및 BTS 애플리케이션 모두에서 호출할 수 있습니다. 이에 대한 예제는 351 페이지의 그림 77에 표시됩니다.

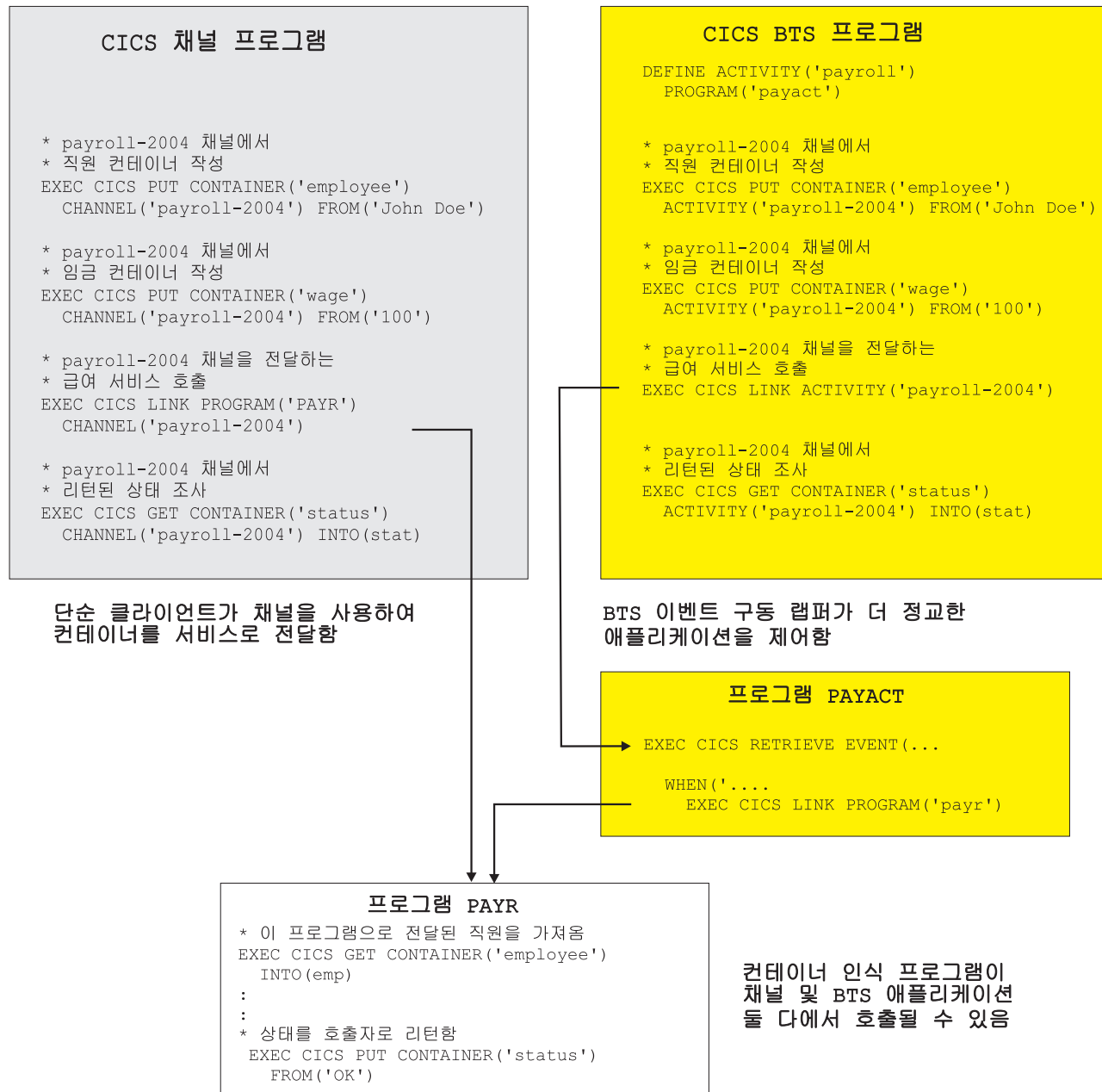


그림 77. 채널 및 BTS 활동

컨텍스트

컨테이너 명령을 실행하는 프로그램은 변경 없이도 채널 애플리케이션의 일부 또는 BTS 활동의 일부로 사용할 수 있습니다.

두 채널과 BTS 컨텍스트에서 사용할 프로그램의 경우 실행하는 컨테이너 명령은 해당 명령을 채널 또는 BTS 명령으로 식별하는 옵션을 지정해서는 안됩니다. 각 컨테이너 명령에서 피해야 할 옵션은 다음과 같습니다.

DELETE CONTAINER

ACQACTIVITY(BTS 특정)
 ACQPROCESS(BTS 특정)
 ACTIVITY(BTS 특정)
 CHANNEL(채널 특정)
 PROCESS(BTS 특정)

GET CONTAINER

ACQACTIVITY(BTS 특정)
 ACQPROCESS(BTS 특정)
 ACTIVITY(BTS 특정)
 CHANNEL(채널 특정)
 INTOCCSID(채널 특정)
 PROCESS(BTS 특정)

MOVE CONTAINER

FROMACTIVITY(BTS 특정)
 CHANNEL(채널 특정)
 FROMPROCESS(BTS 특정)
 TOACTIVITY(BTS 특정)
 TOCHANNEL(채널 특정)
 TOPROCESS(BTS 특정)

PUT CONTAINER

ACQACTIVITY(BTS 특정)
 ACQPROCESS(BTS 특정)
 ACTIVITY(BTS 특정)
 CHANNEL(채널 특정)
 DATATYPE(채널 특정)
 FROMCCSID(채널 특정)
 PROCESS(BTS 특정)

컨테이너 명령이 실행될 때 CICS는 명령을 처리하는 방식을 판별하기 위해서 명령이 발생하는 컨텍스트(채널, BTS 또는 둘 다 아님)를 분석합니다. 컨텍스트를 판별하기 위해 CICS는 다음 테스트 순서를 사용합니다.

1. 채널: 프로그램에 현재 채널이 있습니까?
2. BTS: 프로그램이 BTS 활동의 일부입니까?
3. 없음: 프로그램에 현재 채널이 없고 프로그램이 BTS 활동의 일부가 아닙니다. 따라서 컨테이너 명령이 실행되는 컨텍스트가 없습니다. 이 명령은 INVREQ 조건 및 RESP2 값 4로 거부됩니다.

채널을 사용한 동적 라우팅

채널을 전달하는 EXEC CICS LINK 및 EXEC CICS START 명령을 동적으로 라우팅할 수 있습니다.

따라서 동적으로 라우팅할 수 있는 채널 관련 요청 유형은 다음과 같습니다.

- 프로그램 링크(DPL) 요청
- 터미널 관련 START 요청으로 시작된 트랜잭션
- 비터미널 관련 START 요청

라우팅 프로그램에는 통신 영역의 DYRCHANL 필드에 프로그램 링크 또는 START 명령과 연관된 채널 이름(있는 경우)이 전달됩니다. DYRCHANL 필드는 이전에 나열된 대로 세 가지 유형의 요청에만 적용됩니다. 다른 유형의 요청의 경우 또는 요청과 연관된 채널이 없는 경우에는 공백을 포함합니다.

참고: 라우팅 프로그램의 통신 영역은 DFHDYPDS DSECT를 통해 매핑됩니다.

라우팅 프로그램에는 주소가 아닌 채널 이름이 주어지므로 DYRCHANL 필드를 사용하여 컨테이너의 콘텐츠를 조사하거나 변경할 수 없습니다.

채널을 사용하는 애플리케이션은 채널 내에 DFHROUTE라는 특수 컨테이너를 작성할 수 있습니다. 애플리케이션이 동적으로 라우팅될 LINK 또는 터미널 관련 START 요청(비터미널 관련 START 요청 제외)을 실행하면, 동적 라우팅 프로그램에는 DFHDYPDS의 DYRACMAA 필드에 DFHROUTE 컨테이너의 주소가 제공되며 콘텐츠를 조사하고 변경할 수 있습니다.

COMMAREA가 아닌 채널을 전달하도록 프로그램을 마이그레이션하는 경우, 기존 COMMAREA 구조를 사용하여 DFHROUTE를 매핑할 수 있습니다.

관련 정보:

CICS 동적 라우팅 소개

동적 라우팅 프로그램 쓰기

데이터 변환

채널과 컨테이너를 자주 사용하는 애플리케이션은 데이터를 한 코드 페이지에서 다른 코드 페이지로 변환해야 합니다.

데이터 변환이 필요한 이유

여기에 데이터 변환이 필요한 몇 가지 경우가 있습니다.

- 서로 다른 인코딩 표준(예: EBCDIC 및 ASCII)을 사용하는 플랫폼 사이에서 문자 데이터가 전달되는 경우

- 한 CCSID(Coded Character Set Identifier)에서 다른 인코딩으로 일부 문자 데이터의 인코딩을 변경하려는 경우. CCSID 및 CICS에서 지원되는 CCSID의 목록에 대한 설명은 참조 -> 연결성 및 표준에서 CICS 지원 변환의 내용을 참조하십시오.

채널을 사용한 코드 페이지 변환 준비

CICS는 z/OS 변환 서비스를 사용한 문자 변환을 지원합니다.

시작하기 전에

Unicode 데이터를 포함하는 코드 페이지 변환을 준비하는 경우 z/OS용 CICS Transaction Server 설치 안내서에서 z/OS에 의한 Unicode 데이터 변환 사용의 정보를 참조하십시오.

이 태스크 정보

UTF-8 또는 UTF-16 및 EBCDIC 및 ASCII 코드 페이지로(부터)의 데이터 변환은 적합한 변환 이미지 선택에 따라 다릅니다. Unicode의 UTF-8 및 UTF-16 양식 간 변환도 지원됩니다.

이러한 서비스를 통해 지원되는 변환에 대해서는 z/OS Unicode Services User's Guide and Reference에서 관련 부록을 참조하십시오. 이러한 변환에는 광범위한 문자 인코딩(EBCDIC, ASCII 및 Unicode) 간 변환 기능이 포함됩니다.

참고:

1. 예를 들어 WebSphere MQ 전송을 사용한 037과 500 사이의 변환은 EBCDIC 대 EBCDIC 변환이며 CICS 및 WebSphere MQ에서 사용되는 문자 인코딩의 작은 차이로부터 발생합니다.
2. 각 코드 페이지의 모든 위치가 다른 코드 페이지에 직접 대응 상대를 가지는 것은 아닙니다(예: EBCDIC 문자 NL). Java와 z/OS 변환 서비스는 수행하는 변환에 따라 다를 수 있습니다. 기술 노트 및 기타 인터넷 토론에서 특정 위치에 대한 지침을 제공할 수 있습니다. 특정 환경에서 보다 적합한 변환이 무엇인지에 대한 질문에 따라 프로그래밍 커뮤니티 자체가 나뉘어집니다.

CICS는 이제 z/OS 변환 서비스를 사용하여 이러한 문자 변환을 지원합니다. CICS의 이전 릴리스는 변환을 수행하기 위해 일련의 테이블을 사용했고 해당 테이블을 사용하는 변환은 계속해서 지원됩니다. CICS가 이러한 테이블에서 지원되지 않는 한 쌍의 CCSID 간 변환을 수행해야 하는 경우 z/OS 변환 서비스가 사용됩니다.

필요한 변환 이미지가 사용 가능한지 확인

이러한 애플리케이션이 실행하는 CICS 리전에 대해 특정 변환이 사용 가능하도록 CICS 애플리케이션의 일부로 사용되는 해당 CCSID를 z/OS 변환 이미지 유지보수를 담당하는 시스템 프로그래머에게 알려야 합니다.

CCSID 1200 처리

CICS는 CCSID 1200, 1201 및 1202를 사용하여 UTF-16 데이터가 포함된 변환을 지원합니다. z/OS 변환 서비스는 해당 빅엔디언 양식에 CCSID 1200이 사용되는 것을 허용하지만 리틀엔디언 양식 또는 CCSID 1201 또는 1202에 대한 지원은 포함하지 않습니다. CICS는 변환을 위해 데이터를 z/OS에 전달하기 전에 이러한 지원되지 않는 양식으로 식별되는 소스 데이터를 빅엔디언 양식 1200으로 변환합니다. 대상 데이터가 지원되지 않는 양식 중 하나인 경우 CICS는 빅엔디언 양식인 1200으로 데이터를 수신하고 이 데이터를 필요한 CCSID로 변환합니다. 대상 CCSID가 1200인 경우 CICS는 인코딩이 빅엔디언 양식일 것으로 예상합니다. 이러한 CCSID 사이에서 발생하는 변환인 경우 CICS는 z/OS 변환 서비스를 호출하지 않고도 변환을 수행합니다.

이러한 양식의 UTF-16을 포함하는 변환을 위해 z/OS 변환 이미지를 설정할 때 CCSID 1200을 지정해야 합니다. 변환 이미지를 작성하려고 할 때 CCSID 1201 및 1202는 z/OS에서 인식되지 않습니다.

CICS는 인바운드 변환에서 바이트 순서 표시자를 준수하지만 관련 아웃바운드 변환을 처리할 때 해당 정보를 보존할 수 없습니다. CCSID 1200에 대한 모든 아웃바운드 데이터는 UTF16입니다. BE. 애플리케이션 프로그래머는 이에 대해 알고 필요한 변환을 BE to LE 수행해야 합니다.

Java 프로그램

코드 페이지 변환 기능은 Java 내에 있으므로 이러한 기능을 CICS에서 복제하지 않아도 됩니다. 여기에 설명한 변환 기능은 Java 프로그램으로 확장되지 않습니다. 예제는 CICS의 Java 애플리케이션에서 컨테이너에 데이터 넣기를 참조하십시오.

채널을 사용한 데이터 변환

데이터 교환에 채널을 사용하는 애플리케이션은 단순 데이터 변환 모델을 사용합니다. 변환이 필요하지 않은 경우도 많으며 변환이 필요한 경우에는 단일 프로그래밍 명령어를 사용하여 CICS에 변환을 자동 처리하도록 지시할 수 있습니다.

다음 사항에 주의하십시오.

- 일반적으로 (비Java) CICS TS 프로그램이 다른 (비Java) CICS TS 프로그램을 호출하는 경우 두 프로그램 모두 EBCDIC 인코딩을 사용하기 때문에 데이터 변환이 필요하지 않습니다. 예를 들어, CICS TS C 언어 프로그램이 CICS TS COBOL 프로그램을 호출하여 문자 데이터를 보관 중인 일부 컨테이너를 전달하는 경우 데이터 변환을 사용하는 유일한 이유는 데이터의 CCSID를 변경하려고 하는 비정상적인 경우입니다.
- 채널 애플리케이션에서 사용되는 데이터 변환 모델은 단순합니다. 데이터 채널 컨테이너는 API 명령을 사용하여 애플리케이션 프로그래머의 제어에 따라 변환됩니다.

- 애플리케이션 프로그래머는 사용자 데이터(즉, 애플리케이션에서 작성되는 컨테이너에 있는 데이터)의 변환에만 책임이 있습니다. 시스템 데이터는 필요한 경우 CICS에 의해 자동으로 변환됩니다.
- 애플리케이션 프로그래머는 문자 데이터의 변환에만 관여합니다. 2진 데이터의 변환(빅엔디언과 리틀엔디언 간 변환)은 지원되지 않습니다.
- 애플리케이션은 한 코드 페이지에서 또 다른 코드 페이지로 문자 데이터를 변환하는 간단한 방법으로 컨테이너 API를 사용할 수 있습니다.

데이터 변환을 목적으로 CICS는 다음 두 가지 유형의 데이터를 인식합니다.

CHAR

문자 데이터입니다. 즉, 텍스트 문자열입니다. 컨테이너의 데이터는 데이터를 검색하는 애플리케이션의 코드 페이지로 변환됩니다(필요한 경우). 데이터를 검색하는 애플리케이션이 ASCII 기반 시스템의 클라이언트인 경우 컨테이너 데이터는 ASCII 코드 페이지가 됩니다. CICS Transaction Server for z/OS 애플리케이션인 경우에는 EBCDIC 코드 페이지가 됩니다.

컨테이너에 있는 모든 데이터는 단일 문자열이었던 것처럼 변환됩니다. 1바이트 문자 세트(SBCS) 코드 페이지의 경우 여러 문자 필드로 구성되는 구조는 1바이트 문자열과 같습니다. 그러나 2바이트 문자 세트(DBCS) 코드 페이지의 경우 이는 해당되지 않습니다. DBCS 코드 페이지를 사용하는 경우 데이터 변환이 올바르게 실행되도록 하려면 각 문자열을 개별 컨테이너에 넣어야 합니다. DBCS에 대한 자세한 정보는 IBM 용어집을 참조하십시오.

BIT 모든 비문자 데이터입니다. 즉, 유형 CHAR로 지정되지 않는 모든 데이터입니다. 컨테이너의 데이터를 변환할 수 없습니다. 이 값이 기본값입니다.

다음과 같이 컨테이너에 있는 데이터의 데이터 변환을 위한 코드 페이지를 지정하는 두 가지 방식이 있습니다.

- 코드화된 문자 세트 ID 또는 CCSID로 지정합니다. CCSID는 특정 코드 페이지를 식별하는 10진수입니다. 예를 들어, ASCII 문자 세트 ISO 8859-1에 대한 CCSID는 819입니다.
- 코드 페이지의 IANA 등록 문자 세트 이름입니다. 이는 HTTP 헤더에 charset= 값 형식으로 지정할 수 있는 영숫자 이름입니다. 예를 들어, ISO 8859-1에 대한 CICS에서 지원되는 IANA 문자 세트 이름은 iso-8859-1 및 iso_8859-1입니다.

애플리케이션 프로그래머가 데이터 변환을 위한 코드 페이지를 지정하지 않는 경우 CICS는 CICS 리전 전체에 기본 코드 페이지를 사용합니다. 이 기본 코드 페이지는 **LOCALCCSID** 시스템 초기화 매개변수에서 지정됩니다.

데이터 변환에 다음과 같은 API 명령을 사용할 수 있습니다.

•

```
EXEC CICS PUT CONTAINER [CHANNEL] [DATATYPE] [FROMCCSID | FROMCODEPAGE]
```

-

```
EXEC CICS GET CONTAINER [CHANNEL] [INTOCCSID | INTOCODEPAGE]
```

비Language Environment(LE) AMODE(64) 어셈블러 언어 애플리케이션의 경우 64 비트 저장영역에 있는 데이터의 데이터 변환에 다음과 같은 API 명령을 사용할 수도 있습니다.

-

```
EXEC CICS PUT64 CONTAINER [CHANNEL] [DATATYPE] [FROMCCSID | FROMCODEPAGE]
```

-

```
EXEC CICS GET64 CONTAINER [CHANNEL] [INTOCCSID | INTOCODEPAGE]
```

CICS가 문자 데이터를 자동 변환하게 하는 방법:

PUT CONTAINER 또는 PUT64 CONTAINER 명령의 DATATYPE (DFHVALUE(CHAR)) 옵션을 사용하여 컨테이너가 변환에 적합한 문자 데이터를 보유하도록 지정할 수 있습니다. 클라이언트 및 서버 플랫폼이 다른 경우 GET CONTAINER 또는 GET64 CONTAINER 명령이 데이터를 자동으로 변환합니다.

이 태스크 정보

다음 프로시저는 PUT CONTAINER 및 GET CONTAINER 명령을 사용하여 문자 데이터를 자동으로 변환하는 방법을 보여줍니다.

AMODE(64) 프로그램에서는 PUT64 CONTAINER 및 GET64 CONTAINER 명령을 사용하여 같은 방식으로 64비트 저장영역에 있는 문자 데이터를 변환할 수 있습니다. 이러한 명령은 비Language Environment(LE) AMODE(64) 어셈블러 언어 애플리케이션에서만 사용할 수 있고 CICS 비즈니스 트랜잭션 서비스(BTS) 컨테이너는 지원되지 않습니다.

프로시저

1. 클라이언트 프로그램에서는 PUT CONTAINER 명령의 DATATYPE (DFHVALUE(CHAR)) 옵션을 사용하여 컨테이너가 문자 데이터를 보유하고 데이터가 변환에 적합한 데이터여야 함을 지정합니다. 예를 들어, 다음과 같습니다.

```
EXEC CICS PUT CONTAINER(cont_name) CHANNEL('payroll')
FROM(data1) DATATYPE(DFHVALUE(CHAR))
```

데이터가 클라이언트 플랫폼의 기본 CCSID에 있으면 FROMCCSID 또는 FROMCODEPAGE 옵션을 지정할 필요가 없습니다. (CICS TS 리전에서 기본 CCSID는 LOCALCCSID 시스템 초기화 매개변수에 지정됩니다.) 기본 CCSID가 내재되어 있습니다.

2. 서버 프로그램에서는 다음과 같은 GET CONTAINER 명령을 실행하여 프로그램의 현재 채널에서 데이터를 검색합니다.

```
EXEC CICS GET CONTAINER(cont_name) INTO(data_area1)
```

데이터가 서버 플랫폼의 기본 CCSID로 리턴됩니다. 기본값 외의 CCSID로 데이터를 변환하려는 경우가 아니면 INTOCCSID 또는 INTOCODEPAGE 옵션을 지정할 필요가 없습니다. 클라이언트 및 서버 플랫폼이 다른 경우 데이터 변환이 자동으로 발생합니다.

3. 서버 프로그램에서는 다음과 같은 PUT CONTAINER 명령을 실행하여 클라이언트로 값을 리턴합니다.

```
EXEC CICS PUT CONTAINER(status) FROM(data_area2)
          DATATYPE(DFHVALUE(CHAR))
```

DATATYPE(DFHVALUE(CHAR)) 옵션은 컨테이너가 문자 데이터를 보유하고 데이터가 변환에 적합해야 함을 지정합니다. 데이터가 서버 플랫폼의 기본 CCSID에 있으면 FROMCCSID 또는 FROMCODEPAGE 옵션을 지정할 필요가 없습니다.

4. 클라이언트 프로그램에서는 다음과 같은 GET CONTAINER 명령을 실행하여 서버 프로그램에 의해 리턴되는 상태를 검색합니다.

```
EXEC CICS GET CONTAINER(status) CHANNEL('payroll')
          INTO(status_area)
```

상태는 클라이언트 플랫폼의 기본 CCSID로 리턴됩니다. 기본값 외의 CCSID로 데이터를 변환하려는 경우가 아니면 INTOCCSID 또는 INTOCODEPAGE 옵션을 지정할 필요가 없습니다. 클라이언트 및 서버 플랫폼이 다른 경우 데이터 변환이 자동으로 발생합니다.

컨테이너를 사용하여 코드 페이지 변환:

애플리케이션에서 컨테이너 API를 사용하여 한 코드 페이지에서 다른 코드 페이지로 문자 데이터를 변환할 수 있습니다.

이 태스크 정보

다음 예제는 데이터를 code page1에서 code page2로 변환합니다.

```
EXEC CICS PUT CONTAINER(temp) DATATYPE(DFHVALUE(CHAR))
          FROMCCSID(code_page1) FROM(input-data)
EXEC CICS GET CONTAINER(temp) INTOCCSID(code_page2)
          SET(data-ptr) FLENGTH(data-len)
```

다음 예제는 데이터를 리전의 기본 EBCDIC 코드 페이지에서 지정된 UTF8 코드 페이지로 변환합니다.

```
EXEC CICS PUT CONTAINER(temp) DATATYPE(DFHVALUE(CHAR))
          FROM(ebcdic-data)
EXEC CICS GET CONTAINER(temp) INTOCCSID(utf8_ccsid)
          SET(utf8-data-ptr) FLENGTH(utf8-data-len)
```

컨테이너 API를 이러한 방식으로 사용할 때에는 다음에 유의하십시오.

- 변환된 길이가 알려지지 않은 경우 GET CONTAINER 명령에 INTO 대신 SET 옵션을 사용하십시오. (GET CONTAINER(*cont_name*) NODATA FLENGTH(*len*) 명령을 실행해서 변환된 데이터의 길이를 검색할 수 있습니다.)
- 코드 페이지에 10진수 CCSID가 아닌 지원되는 IANA 문자 세트 이름을 지정하려는 경우 또는 CCSID를 영숫자순으로 지정하려는 경우에는 FROMCCSID 및 INTOCCSID 대신 FROMCODEPAGE 및 INTOCODEPAGE 옵션을 사용하십시오.
- 저장영역 오버헤드를 피하려면 변환 후 변환한 데이터를 복사하고 컨테이너를 삭제하십시오.
- 채널 제공을 피하려면 임시 채널을 사용하십시오. 임시 저장영역 및 채널 사용에 대한 자세한 정보는 397 페이지의 『임시 저장영역 사용 시 연관관계 피하기』의 내용을 참조하십시오.
- 전체 대 전체 변환은 가능하지 않습니다. 즉, 지정된 코드 페이지와 채널의 코드 페이지가 지원되지 않는 조합이면 코드 페이지 변환 오류가 발생합니다.

SOAP 예제:

CICS TS SOAP 애플리케이션을 사용하여 소켓 또는 WebSphere MQ 메시지 큐에서 UTF-8 또는 UTF-16 메시지를 검색하거나, 메시지를 UTF-8 형식으로 컨테이너에 넣거나, EBCDIC 데이터 구조를 같은 채널에 있는 다른 컨테이너에 넣거나, 분배 프로그램 링크(DPL)가 백엔드 AOR에서 핸들러 프로그램을 호출하게 하여 채널을 전달할 수 있습니다.

CICS TS에서도 실행되는 백엔드 핸들러 프로그램은 EXEC CICS GET CONTAINER 명령을 사용하여 EBCDIC 데이터 구조 또는 메시지를 검색할 수 있습니다. 이 프로그램은 UTF-8 또는 UTF-16 형식 또는 프로그램 고유 또는 리전의 EBCDIC 코드 페이지 형식으로 메시지를 가져올 수 있습니다. 마찬가지로 이 프로그램은 EXEC CICS PUT CONTAINER 명령을 사용하여 데이터를 UTF-8, UTF-16 또는 EBCDIC 형식으로 컨테이너에 배치할 수 있습니다.

리전의 EBCDIC 코드 페이지 형식 메시지 중 하나를 검색하기 위해 핸들러는 다음 명령을 실행할 수 있습니다.

```
EXEC CICS GET CONTAINER(input_msg) INTO(msg)
```

INTOCCSID 및 INTOCODEPAGE 옵션은 지정되지 않기 때문에 메시지 데이터는 리전의 EBCDIC 코드 페이지로 자동 변환됩니다(여기에서는 메시지 데이터를 채널에 저장하는 데 사용되는 PUT CONTAINER 명령이 DATATYPE을 CHAR로 지정한 것으로 가정하며 이 명령이 DATATYPE을 기본값 BIT로 지정한 경우에는 변환이 불가능함).

리전의 EBCDIC 코드 페이지 형식으로 일부 출력을 리턴하기 위해서 핸들러는 다음 명령을 실행할 수 있습니다.

```
EXEC CICS PUT CONTAINER(output) FROM(output_msg)
```

CHAR을 지정하지 않기 때문에 데이터 변환은 허용되지 않습니다. FROMCCSID 및 FROMCODEPAGE 옵션이 지정되지 않기 때문에 메시지 데이터는 리전의 EBCDIC 코드 페이지가 됩니다.

UTF-8에서 메시지 중 하나를 검색하려면 INTOCCSID 또는 INTOCODEPAGE 옵션을 지정하여 코드 페이지를 식별하고 데이터가 리전의 EBCDIC 코드 페이지로 자동 변환되는 것을 막아야 합니다. 핸들러는 다음 명령을 실행할 수 있습니다.

```
EXEC CICS GET CONTAINER(input_msg) INTO(msg) INTOCCSID(utf8)
```

이 경우 utf8은 전자로 정의되고 UTF-8에 대해 1208로 초기화되는 변수이며 코드화된 문자 세트 ID 또는 CCSID입니다. 코드 페이지에 IANA 문자 세트 이름을 사용하는 경우 INTOCCSID 옵션 대신 INTOCODEPAGE 옵션을 사용할 수 있습니다.

```
EXEC CICS GET CONTAINER(input_msg) INTO(msg) INTOCODEPAGE(utf8)
```

이 경우 utf8은 길이 56인 문자열로 정의되고 'utf-8'로 초기화됩니다.

일부 출력을 UTF-8 형식으로 리턴하기 위해 서버 프로그램은 다음 명령을 실행할 수 있습니다.

```
EXEC CICS PUT CONTAINER(output) FROM(output_msg) FROMCCSID(utf8)
```

또는 대안으로

```
EXEC CICS PUT CONTAINER(output) FROM(output_msg) FROMCODEPAGE(utf8)
```

여기서 변수 utf8은 INTOCCSID 및 INTOCODEPAGE에서와 같은 방식으로 정의되고 초기화됩니다. FROMCCSID 또는 FROMCODEPAGE 옵션은 메시지 데이터가 현재 UTF-8 형식임을 지정합니다. FROMCCSID 또는 FROMCODEPAGE의 지정은 DATATYPE이 CHAR임을 의미하므로 데이터 변환이 허용됩니다.

COMMAREA에서 채널로 마이그레이션

데이터를 교환하기 위해 기존 통신 영역을 사용하는 CICS 애플리케이션은 전과 같이 계속해서 작업할 수 있습니다. 채널을 마이그레이션하려는 경우 다음은 COMMAREA를 사용하지 않고 채널 및 컨테이너를 사용하기 위해 기존 애플리케이션의 여러 유형을 마이그레이션하는 방법의 예제입니다.

단일 컨테이너를 포함하는 채널로 COMMAREA를 바꿀 수 있습니다. COMMAREA에서 채널 및 컨테이너로 이동시키는 것이 가장 단순한 방법처럼 보일 수도 있지만 이를 수행하는 것이 우수 사례는 아닙니다. 이 새 기능을 이용하기 위해 애플리케이션을 변경하는 데 시간이 소요되기 때문에 채널 및 컨테이너에 대한 『우수 사례』를 구현해

야 합니다. 346 페이지의 『채널 설계: 우수 사례』의 내용을 참조하십시오. 채널은 COMMAREA에 비해 여러 가지 장점이 있고(330 페이지의 『채널의 이점』 참조) 이러한 개선사항 대부분을 구현하기 위해 채널을 디자인하는 것은 도움이 됩니다.

또한 채널이 같은 데이터를 전달하도록 디자인된 COMMAREA에 비해 더 많은 저장 영역을 사용할 수 있음에 유의하십시오(330 페이지의 『채널의 이점』 참조).

사용자 작성 동적 또는 분배 라우팅 프로그램은 채널 및 컨테이너를 사용자 자신의 애플리케이션에서 구현할 계획인지 여부에 따라 작업이 필요합니다. 워크로드 관리를 위해 CICSplex SM 대신 사용자 작성 동적 또는 분배 라우팅 프로그램을 채택하는 경우 프로그램이 DFHDYPDS 통신 영역의 DYRLEVEL, DYRTYPE 및 DYRVER 필드로 전달될 수 있는 새 값을 처리하도록 수정해야 합니다. CICS 사용자 정의 안내서에서 동적 라우팅 프로그램으로 전달되는 매개변수를 참조하십시오.

COMMAREA를 전달하는 LINK 명령 마이그레이션

구조를 교환하기 위해서 LINK 명령에서 COMMAREA를 사용하는 두 개의 프로그램을 마이그레이션하려면 이 테이블에 표시된 명령어를 변경합니다.

이 태스크 정보

다음 명령어에서 structure는 정의한 데이터 구조의 이름입니다. EXEC CICS GET CONTAINER 및 PUT CONTAINER 명령은 16자 필드를 사용하여 컨테이너를 식별합니다. 유용한 규칙은 컨테이너에 사용 중인 데이터 구조와 같은 이름(여기에서 structure-name으로 표시됨)을 지정하는 것입니다.

표 21. COMMAREA를 전달하는 LINK 명령 마이그레이션

프로그램	이전	이후
PROG1	EXEC CICS LINK PROGRAM(PROG2) COMMAREA(structure)	EXEC CICS PUT CONTAINER(structure-name) CHANNEL(channel-name) FROM(structure) EXEC CICS LINK PROGRAM(PROG2) CHANNEL(channel-name) : EXEC CICS GET CONTAINER(structure-name) CHANNEL(channel-name) INTO(structure)
PROG2	EXEC CICS ADDRESS COMMAREA(structure-ptr) ... RETURN	EXEC CICS GET CONTAINER(structure-name) INTO(structure) ... EXEC CICS PUT CONTAINER(structure-name) FROM(structure) RETURN

참고: COMMAREA 예제에서 데이터를 COMMAREA에 넣은 PROG2는 데이터를 PROG1에 리턴하기 위해 RETURN 명령을 실행하기만 하면 됩니다. 채널 예제에서 데이터를 리턴하려면 PROG2가 RETURN 전에 PUT CONTAINER 명령을 실행해야 합니다.

COMMAREA를 전달하는 XCTL 명령 마이그레이션

구조를 전달하기 위해서 LINK 명령에서 COMMAREA를 사용하는 두 개의 프로그램을 마이그레이션하려면 이 테이블에 표시된 명령어를 변경합니다.

이 태스크 정보

다음 명령어에서 `structure`는 정의한 데이터 구조의 이름입니다. EXEC CICS GET CONTAINER 및 PUT CONTAINER 명령은 16자 필드를 사용하여 컨테이너를 식별합니다. 유용한 규칙은 컨테이너에 사용 중인 데이터 구조와 같은 이름(여기에서 `structure-name`으로 표시됨)을 지정하는 것입니다.

표 22. COMMAREA를 전달하는 XCTL 명령 마이그레이션

프로그램	이전	이후
PROG1	EXEC CICS XCTL PROGRAM(PROG2) COMMAREA(structure)	EXEC CICS PUT CONTAINER(structure-name) CHANNEL(channel-name) FROM(structure) EXEC CICS XCTL PROGRAM(PROG2) CHANNEL(channel-name) : :
PROG2	EXEC CICS ADDRESS COMMAREA(structure-ptr) ...	EXEC CICS GET CONTAINER(structure-name) INTO(structure) ...

RETURN 명령에서 의사 대화식 COMMAREA 마이그레이션

의사 대화의 일부로 구조를 교환하기 위해 COMMAREA를 사용하는 두 개의 프로그램을 마이그레이션하려면 이 테이블에 표시된 명령어를 변경합니다.

이 태스크 정보

다음 명령어에서 `structure`는 정의한 데이터 구조의 이름입니다. EXEC CICS GET CONTAINER 및 PUT CONTAINER 명령은 16자 필드를 사용하여 컨테이너를 식별합니다. 유용한 규칙은 컨테이너에 사용 중인 데이터 구조와 같은 이름(여기에서 `structure-name`으로 표시됨)을 지정하는 것입니다.

표 23. RETURN 명령에서 의사 대화식 COMMAREA 마이그레이션

프로그램	이전	이후
PROG1	EXEC CICS RETURN TRANSID(PROG2) COMMAREA(structure)	EXEC CICS PUT CONTAINER(structure-name) CHANNEL(channel-name) FROM(structure) EXEC CICS RETURN TRANSID(TRAN2) CHANNEL(channel-name)
PROG2	EXEC CICS ADDRESS COMMAREA(structure-ptr)	EXEC CICS GET CONTAINER(structure-name) INTO(structure)

START 데이터 마이그레이션

구조를 교환하기 위해서 START 데이터를 사용하는 두 개의 프로그램을 마이그레이션하려면 이 테이블에 표시된 명령어를 변경합니다.

이 태스크 정보

다음 명령어에서 `structure`는 정의한 데이터 구조의 이름입니다. EXEC CICS GET CONTAINER 및 PUT CONTAINER 명령은 16자 필드를 사용하여 컨테이너를 식별합니다. 유용한 규칙은 컨테이너에 사용 중인 데이터 구조와 같은 이름(여기에서 `structure-name`으로 표시됨)을 지정하는 것입니다.

표 24. START 데이터 마이그레이션

프로그램	이전	이후
PROG1	EXEC CICS START TRANSID(TRAN2) FROM(structure)	EXEC CICS PUT CONTAINER(structure-name) CHANNEL(channel-name) FROM(structure) EXEC CICS START TRANSID(TRAN2) CHANNEL(channel-name)
PROG2	EXEC CICS RETRIEVE INTO(structure)	EXEC CICS GET CONTAINER(structure-name) INTO(structure)

PROG2의 새 버전은 의사 대화식 예제의 버전과 같음을 참고하십시오.

임시 저장영역을 사용하여 데이터를 전달하는 마이그레이션 프로그램

채널을 사용할 수 없었던 이전 CICS 릴리스에서 일부 애플리케이션은 임시 저장영역 큐(TS 큐)를 사용하여 한 프로그램에서 다른 프로그램으로 32K를 초과하는 데이터를 전달했습니다. 일반적으로 여기에는 여러 번의 TS 큐에 대한 쓰기 및 TS 큐로부터 읽기가 포함됩니다.

이 태스크 정보

채널을 사용하기 위해 이러한 애플리케이션 중 하나를 마이그레이션하는 경우 다음 사항에 대해 알고 있어야 합니다.

- 기존 애플리케이션에서 사용된 TS 큐가 기본 저장영역에 있는 경우 새 애플리케이션, 마이그레이션된 애플리케이션의 저장영역 요구사항은 기존 애플리케이션의 저장영역 요구사항과 비슷할 가능성이 높습니다.
- 기존 애플리케이션에서 사용된 TS 큐가 보조 저장영역에 있는 경우 마이그레이션된 애플리케이션의 저장영역 요구사항은 기존 애플리케이션의 저장영역 요구사항보다 더 클 가능성이 높습니다. 이는 컨테이너 데이터가 디스크에 기록되기 보다는 저장영역에 보관되기 때문입니다.

동적으로 라우팅되는 애플리케이션 마이그레이션

COMMAREA 또는 채널을 전달할 수 있는 EXEC CICS LINK 및 EXEC CICS START 명령은 동적으로 라우팅할 수 있습니다. 이러한 명령을 마이그레이션하여 COMMAREA 대신 채널을 사용할 수 있습니다.

LINK 또는 START 명령이 채널이 아닌 COMMAREA를 전달하는 경우 라우팅 프로그램은 요청의 유형에 따라 COMMAREA의 내용을 검사하거나 변경할 수 있습니다.

비터미널 관련 START 요청(*distributed* 라우팅 프로그램이 처리함)이 아닌 터미널 관련 START 요청(동적 라우팅 프로그램이 처리함)으로 시작되는 LINK 요청 및 트랜잭션의 경우 라우팅 프로그램이 해당 통신 영역 즉, 애플리케이션 COMMAREA의 주소에 제공되고 내용을 검사 및 변경할 수 있습니다.

참고: 라우팅 프로그램의 통신 영역은 DFHDYPDS DSECT를 통해 맵핑됩니다.

COMMAREA가 아닌 채널을 사용하기 위해 동적으로 라우팅된 EXEC CICS LINK 또는 START 명령을 마이그레이션하는 경우 라우팅 프로그램에 DFHDYPDS의 DYRCHANL 필드로 채널의 이름이 전달됩니다. 라우팅 프로그램에는 주소가 아닌 채널의 이름이 제공되므로 채널의 컨테이너의 내용을 검사하거나 변경하는 데 DYRCHANL 필드를 사용할 수 없음에 유의하십시오.

라우팅 프로그램에 채널과 같은 유형의 기능을 제공하기 위해 채널을 사용하는 애플리케이션이 채널 내에서 DFHROUTE라고 하는 특수 컨테이너를 작성할 수 있습니다. 애플리케이션이 동적으로 라우팅될 LINK 또는 터미널 관련 START 요청(비터미널 관련 START 요청 제외)을 실행하면, 동적 라우팅 프로그램에는 DFHDYPDS의 DYRACMAA 필드에 DFHROUTE 컨테이너의 주소가 제공되며 콘텐츠를 조사하고 변경할 수 있습니다.

COMMAREA가 아닌 채널을 전달하도록 프로그램을 마이그레이션하는 경우, 기존 COMMAREA 구조를 사용하여 DFHROUTE를 맵핑할 수 있습니다.

동적 및 분배 라우팅에 대한 소개 정보는 CICS 상호 통신 안내서에서 CICS 동적 라우팅에 대한 소개를 참조하십시오. 동적 또는 분배 라우팅 프로그램 작성에 대한 정보는 CICS 사용자 정의 안내서에서 동적 라우팅 프로그램 작성을 참조하십시오.

프로그램 제어

CICS 프로그램 제어 기능은 CICS 시스템에서 애플리케이션 사이의 제어 플로우를 관리합니다.

Java 및 C++

여기에 설명된 API(Application Programming Interface)는 Java 프로그램에서 사용되지 않은 CICS API입니다. 프로그램 제어 서비스에 액세스하기 위해 JCICS 클래스를 사용하는 Java 프로그램에 대한 정보는 애플리케이션 개발에서 JCICS를 사용하는 Java 프로그래밍 및 JCICS Javadoc 문서를 참조하십시오. CICS C++ 클래스를 사용하는 C++ 프로그램에 대한 정보는 애플리케이션 개발에서 CICS C++ OO 기반 클래스 사용의 내용을 참조하십시오.

CICS API를 사용하는 프로그램 제어

프로그램 제어 명령에서 참조되는 애플리케이션 이름은 CICS에 프로그램으로 정의되어야 합니다. 제어 프로그램 명령을 다음과 같이 사용할 수 있습니다.

- 애플리케이션 중 하나를 다른 애플리케이션에 링크할 수 있으며 이로 인해 요청 프로그램에 대한 후속 리턴이 예상됩니다(LINK 명령). 이 명령에 COMMAREA, CHANNEL, INPUTMSG 옵션을 사용하면 데이터가 요청된 애플리케이션으로 전달됩니다.
- 애플리케이션 중 하나를 별도의 CICS 리전에 있는 다른 프로그램으로 링크할 수 있으며 요청 프로그램에 대한 후속 리턴이 예상됩니다(LINK 명령). 이 명령에 COMMAREA 또는 CHANNEL 옵션을 사용하면 데이터가 요청된 애플리케이션으로 전달됩니다. 이를 DPL(Distributed Program Link)이라고 합니다. (DPL을 사용하는 경우 LINK 명령에 INPUTMSG 및 INPUTMSGLEN 옵션을 사용할 수 없습니다. DPL에 대한 자세한 정보는 452 페이지의 『CICS 상호 통신』의 내용을 참조하십시오.
- 요청 프로그램에 대한 리턴없이 애플리케이션 중 하나의 제어를 다른 애플리케이션으로 전송할 수 있습니다(XCTL 명령). 이 명령에 COMMAREA, CHANNEL, INPUTMSG 옵션을 사용하면 데이터가 요청된 애플리케이션으로 전달됩니다. (DPL을 사용하는 경우 XCTL 명령에 INPUTMSG 및 INPUTMSGLEN 옵션을 사용할 수 없습니다. DPL에 대한 자세한 정보는 452 페이지의 『CICS 상호 통신』의 내용을 참조하십시오.
- 애플리케이션 중 하나의 제어를 다른 애플리케이션 또는 CICS로 리턴할 수 있습니다(RETURN 명령). 이 명령에 COMMAREA, CHANNEL, INPUTMSG 옵션을 사용하면 데이터가 새로 시작된 트랜잭션으로 전달됩니다. DPL을 사용하는 경우 RETURN 명령에 INPUTMSG 및 INPUTMSGLEN 옵션을 사용할 수 없습니다. DPL에 대한 자세한 정보는 452 페이지의 『CICS 상호 통신』의 내용을 참조하십시오.
- 지정된 애플리케이션, 테이블 또는 맵핑을 기본 저장영역으로 로드할 수 있습니다(LOAD 명령).

LOAD 및 RELEASE 명령에 HOLD 옵션을 사용하여 읽기 전용이 아닌 프로그램, 테이블 또는 맵핑을 로드하면 동적 트랜잭션 라우팅을 수행하는 기능에 악영향을 줄 수 있는 트랜잭션간 연관관계를 작성할 수 있습니다.

이러한 명령을 실행하는 프로그램의 잠재된 문제점을 쉽게 식별하기 위해 CICS Interdependency Analyzer를 사용할 수 있습니다. 이 기능에 대한 자세한 정보는 CICS Interdependency Analyzer for z/OS의 내용을 참조하고 트랜잭션 연관관계에 대한 자세한 정보는 379 페이지의 『연관관계』의 내용을 참조하십시오.

- 이전에 로드한 애플리케이션, 테이블 또는 맵핑을 기본 저장영역에서 삭제할 수 있습니다(RELEASE 명령).

RESP 옵션을 사용하여 비정상 종료를 처리할 수 있습니다.

프로그램 링크

LINK 명령은 제어를 한 논리 레벨에 있는 애플리케이션에서 더 낮은 다음 논리 레벨에 있는 애플리케이션으로 전달하는 데 사용됩니다.

애플리케이션 논리 레벨

CICS 아래에서 실행 중인 애플리케이션은 다양한 논리 레벨에서 실행됩니다. 태스크 내에서 제어를 수신할 첫 번째 프로그램은 최상위 논리 레벨에 있습니다.

애플리케이션이 다른 애플리케이션에 링크될때(최후 제어 리턴을 예상하고) 링크 대상 프로그램은 다음으로 낮은 논리 레벨에 상주할 것으로 간주됩니다. 제어 리턴을 예상하지 않고 제어가 하나의 애플리케이션에서 다른 애플리케이션으로 전송될 때, 두 프로그램은 동일한 논리 레벨에서 상주할 것으로 간주됩니다.

리턴이 예상되는 다른 프로그램에 대한 링크

제어를 수신하는 프로그램이 아직 기본 저장영역에 없으면 로드됩니다. RETURN 명령이 링크된 프로그램에서 처리되면 제어는 프로그램으로 리턴되어 다음 순차 프로세스 명령에서 링크를 시작합니다.

링크된 프로그램은 예외 조건, 어텐션 ID 및 이상 종료 처리와 관련하여 LINK 명령을 실행하는 프로그램과는 독립적으로 작동합니다. 예를 들어, 링크 프로그램에서 HANDLE 명령의 영향은 링크 대상 프로그램에 의해 상속되지 않지만 원본 HANDLE 명령은 링크 프로그램으로 리턴 시에 복원됩니다. HANDLE ABEND 명령을 사용하여 다른 링크 레벨에서 비정상 종료를 처리할 수 있습니다. 367 페이지의 그림 78에서는 논리 레벨의 개념을 보여줍니다.

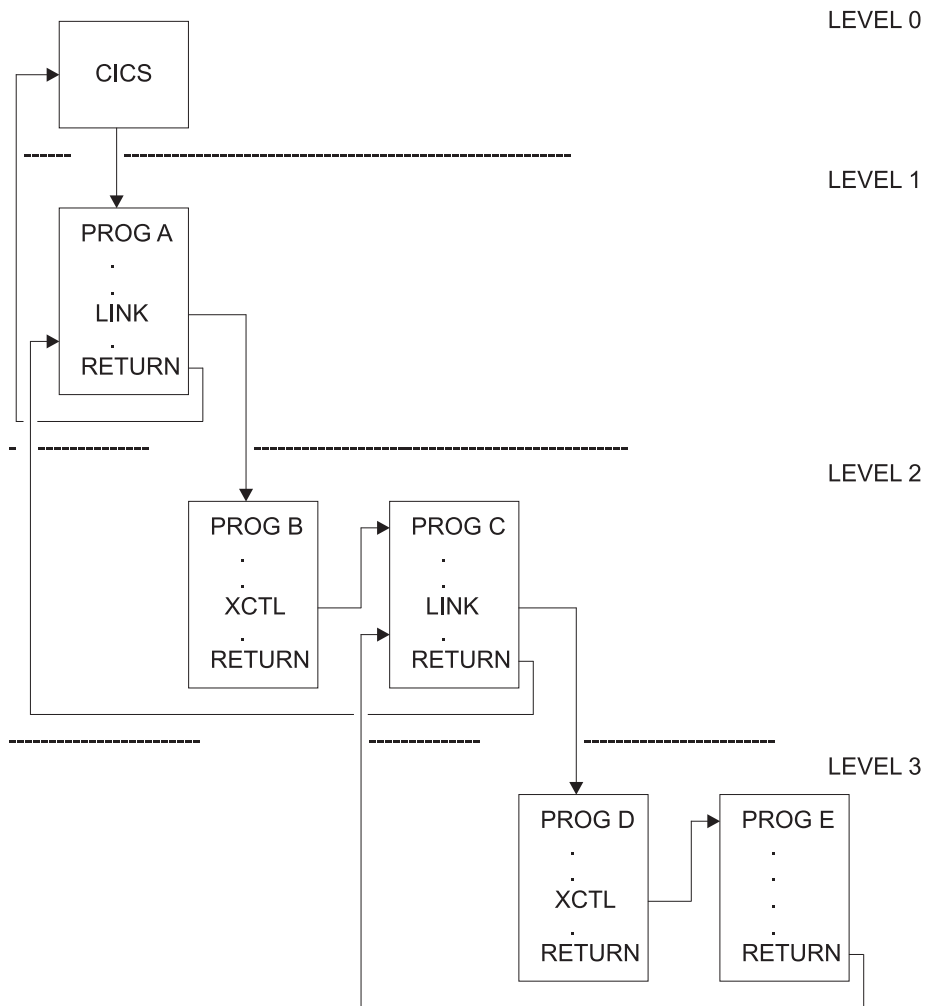


그림 78. 애플리케이션 논리 레벨

다른 프로그램으로 데이터 전달

EXEC CICS 프로그램 제어 명령 LINK, XCTL 및 RETURN을 사용하고 해당 명령의 COMMAREA, CHANNEL 또는 INPUTMSG 옵션을 사용하여 다른 프로그램으로 데이터를 전달할 수 있습니다. COMMAREA 및 CHANNEL은 상호 독점적입니다.

COMMAREA

COMMAREA는 데이터가 프로그램 또는 트랜잭션으로 전달되는 데이터 영역(통신 영역이라고 함) 이름을 지정합니다. 이는 LINK, XCTL 및 RETURN 명령의 옵션입니다.

LINK 및 XCTL 명령의 COMMAREA 옵션은 데이터가 호출 중인 프로그램에 전달되는 데이터 영역의 이름을 지정합니다.

RETURN 명령의 COMMAREA 옵션은 TRANSID 옵션으로 식별된 트랜잭션에 데이터가 전달되는 통신 영역의 이름을 지정합니다. TRANSID 옵션은 다음 입력이 태스크에 연관된 터미널에서 수신될 때 시작되는 트랜잭션을 지정합니다.

호출된 프로그램은 매개변수로 데이터를 수신합니다. 프로그램은 전달된 데이터에 액세스할 수 있도록 데이터 영역의 정의를 포함해야 합니다.

수신 COBOL 프로그램에서 데이터 영역에 이름을 DFHCOMMAREA로 지정해야 합니다. COBOL 프로그램에서 프로그램이 COMMAREA를 LINK, XCTL 또는 RETURN 명령의 파트로 전달하면 작업 중인 저장영역이나 LINKAGE SECTION은 데이터 영역을 포함할 수 있습니다. COMMAREA를 수신 중인 프로그램은 데이터를 LINKAGE SECTION에 지정해야 합니다. 이는 프로그램이 다음 중 하나일 때 적용됩니다.

- COMMAREA가 전달되는 LINK 또는 XCTL 명령 중의 수신 프로그램
- 이전에 호출된 태스크의 RETURN 명령이 COMMAREA 및 TRANSID를 지정하는 초기 프로그램

COMMAREA를 수신 중인 C 또는 C++ 프로그램에서 COMMAREA는 구조에 대한 포인터로 정의되어야 합니다. 그러면 프로그램이 ADDRESS COMMAREA 명령을 실행하여 전달된 데이터에 대한 주소지정가능성을 확보해야 합니다.

PL/I 프로그램에서 데이터 영역은 모든 이름을 사용할 수 있지만 프로그램에 전달된 매개변수를 기반으로 기본 변수로 선언되어야 합니다. 이 기본 변수에 대한 포인터는 영역에 대한 선언에 표시되어 문맥상으로 선언되는 것이 아니라 포인터로 명시적으로 선언되어야 합니다. 그러면 PL/I 오류 메시지가 생성되는 것이 방지됩니다. ALLOCATE 문은 이 포인터를 기반으로 하는 모든 변수에 대해 수신 프로그램 내에서 처리할 수 없습니다. 이 포인터는 애플리케이션으로 갱신해야 합니다.

어셈블러 언어 프로그램에서 데이터 영역은 DSECT 맵핑이어야 합니다. 이 데이터 영역의 주소 지정에 사용되는 레지스터는 DFHEISTG DSECT로 맵핑되는 DFHEICAP(통신 영역 포인터)에서 로드해야 합니다. COMMAREA가 64비트 저장영역에 있으면 안 됩니다.

수신 데이터 영역은 원래 통신 영역 길이와 동일하거나 더 짧을 수는 있지만 길면 안 됩니다. 데이터의 첫 번째 파트에만 액세스해야 하는 경우, 수신 데이터 영역은 더 짧을 수 있습니다. 수신 데이터 영역이 원래 통신 영역보다 길면 트랜잭션이 이미 전달된 영역 외부의 데이터를 읽으려고 할 수도 있습니다. 영역 외부의 데이터를 겹쳐쓰려고 할 수도 있으며 이로 인해 CICS가 이상 종료될 수 있습니다.

이 문제를 방지하려면 프로그램은 태스크의 EIB에서 EIBCALEN 필드에 액세스하여 전달된 모든 통신 영역의 길이가 예상대로 인지 확인해야 합니다. 전달된 통신 영역이 없는 경우, EIBCALEN 값은 0입니다. 그렇지 않으면 EIBCALEN은 항상 호출된 프

로그래밍의 데이터 영역 크기에 상관없이 LINK, XCTL 또는 RETURN 명령의 LENGTH 옵션에 지정된 값을 포함합니다. EIBCALEN 값이 프로그램의 DSECT 값과 일치하는지 확인하고 트랜잭션이 해당 영역의 데이터에 액세스하는지 확인하십시오.

전달 중인 데이터에서 추가 확인을 위해 COMMAREA에 ID를 추가할 수도 있습니다. 이 ID는 전송 트랜잭션과 같이 전송되며 수신 트랜잭션을 사용하여 확인됩니다.

LINK 명령을 사용하여 통신 영역이 전달되면 호출된 프로그램은 통신 영역 자체에 대한 포인터로 전달됩니다. 호출된 프로그램의 데이터 영역 내용에 대한 모든 변경사항은 제어기가 리턴할 때 호출 프로그램에서 사용할 수 있습니다. 이런 변경사항에 액세스하기 위해 프로그램은 원래 COMMAREA 옵션에 지정된 데이터 영역의 이름을 지정합니다.

XCTL 명령을 사용하여 통신 영역이 전달되면 전달되는 영역에 명령을 실행하는 프로그램에 전달되는 영역과 동일한 주소 및 길이가 있는 경우를 제외하고는 해당 영역의 사본이 작성됩니다. 예를 들어, 프로그램 A가 LINK 명령을 프로그램 B에 대해 실행하고 프로그램 B는 결국 XCTL 명령을 프로그램 C에 대해 실행하며 B가 A가 B에 전달한 것과 동일한 통신 영역을 C에 전달하는 경우, 프로그램 C는 A에 속하는 통신 영역에 주소지정가능성 영역을 전달하게 되고(사본이 아님) C가 작성한 모든 변경사항은 제어기가 리턴될 때 A에게 사용 가능해집니다.

LINK 명령으로 액세스된 하위 레벨 프로그램이 RETURN 명령을 실행할 때 제어는 제어를 리턴하는 프로그램보다 한 논리 레벨 위로 다시 전달합니다. 태스크가 터미널과 연관된 경우, TRANSID 옵션은 해당 터미널과 연관되는 다음 트랜잭션에 대한 트랜잭션 ID를 지정하기 위해 낮은 레벨에서 사용할 수 있습니다. 가장 높은 논리 레벨이 RETURN 명령을 사용하여 CICS로 제어를 주고 입력이 터미널에서 수신된 후에만 트랜잭션 ID가 사용 가능합니다. 터미널에서 입력된 모든 입력은 어텐션 키와는 별개로 데이터로 완벽하게 해석됩니다. 임의의 링크 레벨에서 리턴할 때 COMMAREA 없이 TRANSID 옵션을 사용할 수는 있지만 나중에 RETURN 명령으로 덮어쓰기 됩니다. RETURN 명령이 올바르게 실행된 COMMAREA로 인해 최상위 레벨에서 실패하면 TRANSID는 널(null)이 됩니다. 또한, COMMAREA 또는 IMMEDIATE는 최상위 레벨에서만 지정할 수 있으며 그렇지 않으면 RESP2=2가 포함된 INVREQ를 수신합니다.

또한, COMMAREA 옵션은 데이터를 시작되는 새 태스크로 전달할 때 사용할 수도 있습니다.

호출된 프로그램은 EIB에서 EIBFN 필드에 액세스하여 이를 호출하는 명령 유형을 판별할 수 있습니다. 이 필드는 모든 CICS 명령 실행 전에 테스트해야 합니다. 프로그램이 LINK 또는 XCTL 명령으로 호출되는 경우, 적절한 코드는 EIBFN 필드에서 제공됩니다. RETURN 명령으로 호출되는 경우에는 태스크에서 CICS 명령이 실행되지 않으며 필드에는 0이 포함됩니다.

채널

통신 영역을 사용하는 대신에 CICS 프로그램 간에 데이터를 전송하는 보다 근대적인 방식은 채널을 사용하는 것입니다.

@@채널에는 COMMAREA 보다 몇몇 장점이 있습니다. 330 페이지의 『채널의 이점』. 채널은 COMMAREA 대신에 LINK, XCTL 및 RETURN 명령에 전달될 수 있습니다.

채널은 321 페이지의 『채널을 사용하는 프로그램 간 데이터 통신』에 설명되어 있습니다.

INPUTMSG

LINK, XCTL, RETURN 명령의 INPUTMSG 옵션은 호출 중인 프로그램에 전달되는 데이터 영역 이름을 지정하는 다른 방법입니다.

이 경우, 호출된 프로그램은 RECEIVE 명령을 처리하여 데이터를 가져옵니다. 이 옵션을 사용하면 터미널에서 직접 호출되도록 작성된 초기 터미널 입력을 가져오는 RECEIVE 명령이 포함된 (『프론트 엔드』) 애플리케이션을 호출할 수 있습니다.

LINK 명령으로 액세스한 프로그램이 터미널에서 초기 입력을 가져오려고 RECEIVE 명령을 실행하지만 초기 RECEIVE 요청이 더 높은 레벨의 프로그램으로 실행되었다면 프로그램이 수신할 수 있는 데이터는 없습니다. 이 경우, 애플리케이션은 터미널의 입력을 대기합니다. INPUTMSG 옵션으로 호출하여 원래 터미널 입력이 링크된 프로그램에서 계속 사용되도록 할 수 있습니다.

LINK(또는 XCTL 또는 RETURN) 명령에 INPUTMSG를 지정하여 애플리케이션이 다른 프로그램을 호출하면 INPUTMSG에 지정된 데이터는 링크된 프로그램 자체가 RECEIVE 명령을 실행하지 않고 대신 다른 애플리케이션을 호출하더라도 계속 사용 가능합니다. INPUTMSG 설명은 371 페이지의 그림 79의 내용을 참조하십시오.

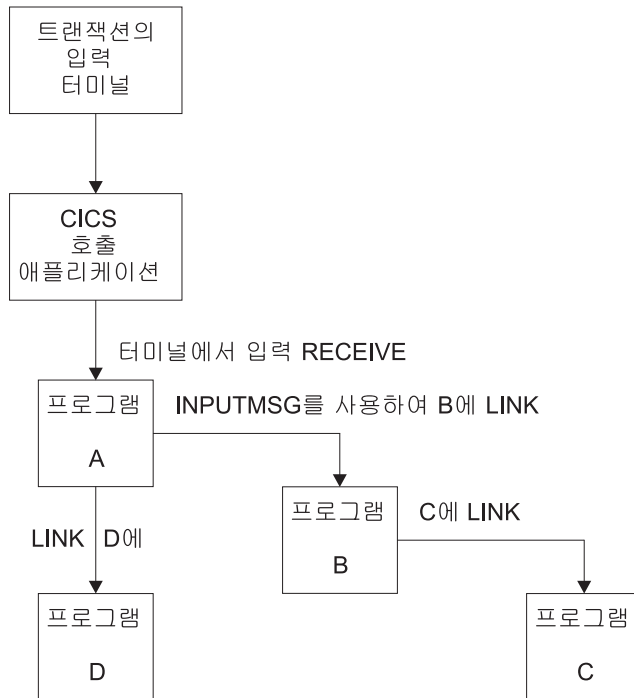


그림 79. 링크된 체인에 INPUTMSG 사용

참고:

- 이 예제에서 『실제』 첫 번째 RECEIVE 명령은 프로그램 A에서 실행됩니다. INPUTMSG 옵션으로 프로그램 B에 링크하여 RECEIVE 요청을 발행하는 다음 프로그램도 터미널 입력을 수신할 수 있습니다. 이는 프로그램 B 또는 프로그램 C 일 수 있습니다.
- 프로그램 A가 수신한 수정되지 않은 터미널 입력을 전달하려고 하는 경우, RECEIVE 명령에 사용되는 INPUTMSG 옵션에 대해 동일한 데이터 영역의 이름을 지정할 수 있습니다. 예를 들어, 다음과 같습니다.

```
EXEC CICS RECEIVE
      INTO(TERMINAL-INPUT)
```

```
EXEC CICS LINK
      PROGRAM(PROGRAMB)
      INPUTMSG(TERMINAL-INPUT)
```

- LINKchain의 한 프로그램이 RECEIVE 명령을 실행하는 즉시 INPUTMSG 데이터는 모든 후속 RECEIVE 명령에 대해 사용 가능하던 것이 중지됩니다. 즉, 표시된 예제에서 B가 C 링크 전에 RECEIVE 요청을 발행한 경우, INPUTMSG 데이터 영역은 C에 사용 불가능합니다.
- 임의 프로그램에서 다른 프로그램으로의 이 데이터 통신 방법은 모든 데이터에 사용할 수 있으며 사용자 터미널에서 시작될 필요는 없습니다. 이 예제에서 프로그램 A는 모든 데이터를 이름 지정된 데이터 영역으로 이동하고 데이터를 참조하는 INPUTMSG로 프로그램 B를 호출할 수 있습니다.

5. INPUTMSG에서 전달된 『터미널 데이터』도 제어가 결국 INPUTMSG로 링크를 받은 프로그램으로 리턴되면 더 이상 사용 가능하지 않습니다. 이 예에서 C가 B로 리턴되고 B가 A로 리턴되면 B나 C는 RECEIVE 명령을 실행하지 않고 데이터는 A에서 수신되는 것으로 간주됩니다. 그런 다음 A가 다른 프로그램(예: 프로그램 D)을 호출하면 원래 INPUTMSG 데이터는 INPUTMSG 옵션을 지정한 경우를 제외하고 더 이상 D에서 사용할 수 없습니다.
6. INPUTMSG 데이터는 SEND 또는 CONVERSE 명령이 실행될 때 사용 불가능하게 됩니다.

RETURN 명령에서 INPUTMSG 옵션 사용

INPUTMSG를 지정하여 데이터를 TRANSID 옵션이 사용된 RETURN 명령에 지정된 다음 트랜잭션으로 전달할 수 있습니다. 이를 수행하려면 RETURN이 가장 높은 논리 레벨에서 실행되어 제어를 CICS로 리턴해야 하고 명령도 IMMEDIATE 옵션을 지정해야 합니다. TRANSID가 포함된 INPUTMSG를 지정했지만 IMMEDIATE는 지정하지 않은 경우, 터미널의 다음 실제 입력은 INPUTMSG 데이터를 덮어쓰기 때문에 결국 유실됩니다.

TRANSID가 포함된 INPUTMSG를 SEND 명령 실행 후에 지정하면 SEND 메시지는 즉시 터미널로 분출됩니다.

TRANSID 옵션이 없는 RETURN 명령에서 INPUTMSG를 사용하는 다른 방법은 동적 트랜잭션 라우팅 프로그램으로 사용하는 것입니다. 사용자 대체 기능 동적 트랜잭션 라우팅 프로그램에 대한 프로그래밍 정보는 *CICS 사용자 정의 안내서*의 동적 라우팅 프로그램 작성을 참조하십시오.

혼합 주소 지정 모드 사용

CICS는 다양한 주소 지정 모드를 가지고 있는 프로그램과 동일한 주소 지정 모드를 가지고 있는 프로그램 사이에 LINK, XCTL 및 RETURN 명령을 사용하는 것을 지원합니다.

다음 제한사항은 COMMAREA 옵션이라고 하는 통신 영역을 사용하여 데이터를 전달하는 프로그램에 적용됩니다.

- AMODE(31) 프로그램에 통신 영역 내에서 전달된 주소 길이는 31비트여야 합니다. 호출된 프로그램이 특히 상위 바이트를 무시하도록 디자인되지 않은 경우에는 플래그 데이터가 상위 바이트로 패딩 3바이트 주소를 사용하지 마십시오.
- 통신 영역 내에서 AMODE(64)로 전달되는 주소는 64비트 또는 31비트 길이가 가능합니다. 사용하기 전에 31비트 주소를 64비트 주소로 변경해야 합니다.
- AMODE(24) 프로그램에 데이터로 전달되는 주소는 호출된 프로그램에 의해 올바르게 액세스되는 경우 16MB 행 아래여야 합니다.

이 제한사항은 통신 영역 자체의 주소에 적용되며, 그 안에 있는 주소에도 적용됩니다. 그러나 16MB 위지만 2GB 아래(행 위) 통신 영역은 AMODE(24) 서브프로그램으로 전달될 수 있습니다. CICS는 처리를 위해 16MB 아래 영역으로 통신 영역을 복사합니다. 제어가 링크 프로그램으로 리턴될 때 다시 복사합니다.

CICS는 주소 지정 모드가 다른 프로그램 사이의 통신 영역(COMMAREA) 내에서 전달되는 데이터 주소의 유효성을 검증하지 않습니다.

COMMAREA가 64비트 저장영역에 있으면 안됩니다.

LINK를 사용한 데이터 전달

이들 예제는 LINK 명령이 데이터가 링크될 프로그램으로 전달되게 만드는 방법을 보여줍니다. XCTL 명령이 비슷한 방식으로 코딩됩니다.

이 태스크 정보

374 페이지의 그림 80부터 375 페이지의 그림 83까지는 COBOL, C, C++, PL/I, 어셈블러 언어에서 LINK 명령을 사용하는 방법을 보여줍니다.

이들 예제는 COMMAREA에서 전달되는 데이터를 보여줍니다. 채널을 사용하여 데이터를 전달하는 LINK 명령의 예제에 대해서는 321 페이지의 『채널을 사용하는 프로그램 간 데이터 통신』의 내용을 참조하십시오.

LINK 명령에서 지정하는 LENGTH 값이 COMMAREA에서 전달될 데이터의 길이와 일치하는지 확인하십시오. LENGTH에 대해 0(영)을 지정하지 마십시오. 그 결과 작동을 예측할 수 없으며 EXEC CICS LINK 명령이 실패할 수 있기 때문입니다.

데이터 전달에 COMMAREA를 사용하는 경우 링크되는 프로그램에서는 태스크의 EIB에 있는 EIBCALEN 필드가 프로그램이 예상하는 값과 일치하는지 확인해야 합니다. 불일치할 경우 저장영역 위반 또는 시스템 장애가 발생할 수 있습니다. 자세한 정보는 367 페이지의 『COMMAREA』의 내용을 참조하십시오.

```

                Invoking program
IDENTIFICATION DIVISION.
PROGRAM ID. 'PROG1'.
.
WORKING-STORAGE SECTION.
01 COM-REGION.
    02 FIELD PICTURE X(3).
.
PROCEDURE DIVISION.
    MOVE 'ABC' TO FIELD.
    EXEC CICS LINK PROGRAM('PROG2')
        COMMAREA(COM-REGION)
        LENGTH(3) END-EXEC.
.
                Invoked program
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG2'.
.
LINKAGE SECTION.
01 DFHCOMMAREA.
    02 FIELD PICTURE X(3).
.
PROCEDURE DIVISION.
IF EIBCALEN GREATER ZERO
THEN
    IF FIELD EQUALS 'ABC' ...

```

그림 80. LINK 명령: COBOL 예제

다음 예제에서 COMMAREA는 문자열을 포함합니다. 구조를 포함하는 COMMAREA의 예제에 대해서는 376 페이지의 『RETURN을 사용한 데이터 전달』의 내용을 참조하십시오.

```

                Invoking program
main()
{
    unsigned char field[3];
    memcpy(field, "ABC", 3);
    EXEC CICS LINK PROGRAM("PROG2")
        COMMAREA(field)
        LENGTH(sizeof(field));
}
                Invoked program
main()
{
    unsigned char *commarea;
    EXEC CICS ADDRESS COMMAREA(commarea) EIB(dfheiptr);
    if (dfheiptr->eibcalen > 0)
    {
        if (memcmp(commarea, "ABC", 3) == 0)
        {

```

그림 81. LINK 명령: C 예제

```

                Invoking program
PROG1: PROC OPTIONS(MAIN);
DCL 1 COM_REGION AUTOMATIC,
      2 FIELD CHAR(3),
.
FIELD='ABC';
EXEC CICS LINK PROGRAM('PROG2')
      COMMAREA(COM_REGION) LENGTH(3);
END;

```

```

                Invoked program
PROG2:
PROC(COMM_REG_PTR) OPTIONS(MAIN);
DCL COMM_REG_PTR PTR;
DCL 1 COM_REGION BASED(COMM_REG_PTR),
      2 FIELD CHAR(3),
.
IF EIBCALEN>0 THEN DO;
    IF FIELD='ABC' THEN ...
.
    END;
END;

```

그림 82. LINK 명령: PL/I 예제

```

                Invoking program
DFHEISTG DSECT
COMREG   DS 0CL20
FIELD    DS CL3
.
PROG1    CSECT
.
        MVC FIELD,=C'XYZ'
        EXEC CICS LINK
            PROGRAM('PROG2')
            COMMAREA(COMREG) LENGTH(3)
.
        END
                Invoked program
COMREG   DSECT
FIELD    DS CL3
.
PROG2    CSECT
.
        L COMPTR,DFHEICAP
        USING COMREG,COMPTR
        CLC FIELD,=C'ABC'
.
        END

```

그림 83. LINK 명령: 어셈블러 언어 예제

RETURN을 사용한 데이터 전달

이들 예제는 RETURN 명령을 사용하여 데이터를 새 트랜잭션으로 전달하는 방법을 보여줍니다.

이 태스크 정보

그림 그림 84 - 379 페이지의 그림 87은 COBOL, C, C++, PL/I, 어셈블러 언어에서 RETURN 명령을 사용하는 방법을 보여줍니다.

이들 예제는 COMMAREA에서 리턴될 데이터를 보여줍니다. 채널을 사용하여 데이터를 리턴하는 RETURN 명령의 예제에 대해서는 321 페이지의 『채널을 사용하는 프로그램 간 데이터 통신』의 내용을 참조하십시오.

```

                Invoking program
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG1'.

.
WORKING-STORAGE SECTION.
01  TERMINAL-STORAGE.
    02  FIELD PICTURE X(3).
    02  DATAFLD PICTURE X(17).

.
PROCEDURE DIVISION.
    MOVE 'XYZ' TO FIELD.
    EXEC CICS RETURN TRANSID('TRN2')
        COMMAREA(TERMINAL-STORAGE)
        LENGTH(20) END-EXEC.

.
                Invoked program
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG2'

.
LINKAGE SECTION.
01  DFHCOMMAREA.
    02  FIELD PICTURE X(3).
    02  DATAFLD PICTURE X(17).

.
PROCEDURE DIVISION.
    IF EIBCALEN GREATER ZERO
    THEN
        IF FIELD EQUALS 'XYZ'
            MOVE 'ABC' TO FIELD.
    EXEC CICS RETURN END-EXEC.

```

그림 84. RETURN 명령: COBOL 예제

```

                                Invoking program
struct ter_struct
{
    unsigned char field[3];
    unsigned char datafld[17];
};
main()
{
    struct ter_struct ter_stor;
    memcpy(ter_stor.field,"XYZ",3);
    EXEC CICS RETURN TRANSID("TRN2")
        COMMAREA(&ter_stor)
        LENGTH(sizeof(ter_stor));
}

                                Invoked program
struct term_struct
{
    unsigned char field[3];
    unsigned char datafld[17];
};
main()
{
    struct term_struct *commarea;
    EXEC CICS ADDRESS COMMAREA(commarea) EIB(dfheiptr);
    if (dfheiptr->eibcalen > 0)
    {
        if (memcmp(commarea->field, "XYZ", 3) == 0)
            memcpy(commarea->field, "ABC", 3);
    }
    EXEC CICS RETURN;
}

```

그림 85. RETURN 명령: C 예제

```

                Invoking program
PROG1: PROC OPTIONS(MAIN);
DCL 1 TERM_STORAGE,
      2 FIELD CHAR(3),
.
FIELD='XYZ';
EXEC CICS RETURN TRANSID('TRN2')
      COMMAREA(TERM_STORAGE);
END;

                Invoked program
PROG2:
PROC(TERM_STG_PTR) OPTIONS(MAIN);
DCL TERM_STG_PTR PTR;
DCL 1 TERM_STORAGE
      BASED(TERM_STG_PTR),
      2 FIELD CHAR(3),
.
IF EIBCALEN>0 THEN DO;
  IF FIELD='XYZ' THEN FIELD='ABC';
END;
EXEC CICS RETURN;
END;

```

그림 86. RETURN 명령: PL/I 예제

Invoking program

```

DFHEISTG DSECT
TERMSTG  DS 0CL20
FIELD    DS CL3
DATAFLD  DS CL17
:
:
PROG1    CSECT
:
:
        MVC FIELD,=C'ABC'
        EXEC CICS RETURN
        TRANSID('TRN2')
        COMMAREA(TERMSTG)
:
:
        END

```

Invoked program

```

TERMSTG  DSECT
FIELD    DS CL3
DATAFLD  DS CL17
:
:
PROG2    CSECT
:
:
        CLC EIBCALEN,=H'0'
        BNH LABEL2
        L  COMPTR,DFHEICAP
        USING TERMSTG,COMPTR
        CLC FIELD,=C'XYZ'
        BNE LABEL1
        MVC FIELD,=C'ABC'
LABEL1    DS 0H
:
:
LABEL2    DS 0H
:
:
        END

```

그림 87. RETURN 명령: 어셈블러 언어 예제

연관관계

CICS 트랜잭션 및 프로그램은 많은 다양한 기술을 사용하여 서로 데이터를 전달합니다. 이러한 기술 중 일부에서는 데이터를 교환하는 트랜잭션 또는 프로그램이 동일한 CICS 리전에서 실행되어야 합니다. 여기에는 트랜잭션 및 DPL(Distributed Program Link) 요청이 동적으로 라우팅될 수 있는 리전에 대해 제한사항이 있으며 요청 사이에 연관관계를 작성하는 것을 말합니다.

Java

트랜잭션 사이의 연관관계에 대한 안내에서는 EXEC CICS API를 사용하여 작성된 애플리케이션에 대해 설명합니다. 그러나 많은 설명들은 CICSplex에서 실행 중인 Java 애플리케이션에 대해 고르게 적용됩니다. Java 애플리케이션 개발에 대한 안내는 애플리케이션 개발에서 JCICS를 사용하는 Java 프로그래밍의 내용을 참조하십시오.

트랜잭션, 프로그램 링크 요청, EXEC CICS START 요청 및 CICS 비즈니스 트랜잭션 서비스(BTS) 활동은 모두 동적으로 라우팅될 수 있습니다.

동적 라우팅 프로그램을 사용하여 동적으로 라우팅할 수 있습니다.

- 터미널에서 시작된 트랜잭션
- 적합한 터미널 관련 EXEC CICS START 명령에 의해 시작된 트랜잭션
- 적합한 CICS 대 CICS DPL 요청
- CICS 외부로부터 수신된 적합한 프로그램 링크 요청

분산 라우팅 프로그램을 사용하여 동적으로 경로를 지정할 수 있습니다.

- 적합한 BTS 프로세스 및 활동.
- 적합한 비터미널 관련 EXEC CICS START 요청.

동적 라우팅 및 분산 라우팅에 대한 자세한 소개 정보는 *CICS 상호 통신 안내서*에 있는 CICS 동적 라우팅 소개를 참조하십시오.

중요사항:

다음 절에서는 트랜잭션 사이의 연관관계에 대해 독점적으로 설명합니다.

- 연관관계는 프로그램 사이에도 존재할 수 있습니다. (엄밀히 말하면, 연관관계가 있는 프로그램과 연관되는 트랜잭션이라고 말할 수 있습니다.) 여기에는 프로그램 링크 요청을 라우팅할 수 있는 리전에 대해 제한사항이 있을 수 있습니다.
- 안전한, 안전하지 않은, 그리고 의심스러운 프로그래밍 기술에 대한 절은 트랜잭션의 라우팅 뿐만 아니라, 프로그램 링크 및 START 요청의 라우팅에 적용됩니다.

연관관계 유형

트랜잭션간 연관관계 및 트랜잭션 시스템 연관관계의 두 가지 유형의 연관관계가 동적 라우팅에 영향을 미칩니다. 트랜잭션간 연관관계는 공통 자원을 공유하거나 해당 처리를 조정하는 트랜잭션 세트 사이에서 발생합니다. 트랜잭션 시스템 연관관계는 트랜잭션과 특정 CICS 리전 간의 연관관계입니다. 여기서 트랜잭션은 해당 CICS 리전의 특성에 신호를 보내거나 변경합니다.

트랜잭션간 연관관계

둘 이상의 CICS 트랜잭션 사이의 트랜잭션 연관관계는 동일 CICS 리전에서 실행하기 위해서는 트랜잭션을 필요로 하는 방식으로 정보를 서로 전달하거나 서로 간의 활동을 동기화하는 기술을 사용하는 트랜잭션에 의해 발생합니다.

이 연관관계 유형은 다음 상황에서 발생할 수 있습니다.

- 한 트랜잭션은 두 번째 트랜잭션이 첫 번째 트랜잭션과 동일한 CICS 리전에서 실행해서만 액세스할 수 있는 장소에 '상태 데이터'를 남기고 종료됩니다.
- 한 트랜잭션은 첫 번째 트랜잭션이 여전히 실행 중일 때 두 번째 트랜잭션이 액세스하는 데이터를 작성합니다. 이것이 안전하게 작동하려면 첫 번째 트랜잭션은 일반적으로 두 번째 트랜잭션이 첫 번째 트랜잭션이 작성한 데이터를 읽었을 때 두 번째 트랜잭션이 게시하는 몇몇 이벤트를 기다립니다. 이 방법에서는 두 트랜잭션 모두가 동일한 CICS 리전으로 라우팅되어야 합니다.
- 두 트랜잭션은 이벤트 제어 블록(ECB) 메커니즘을 사용하는 동기화됩니다. CICS에는 이 기술에 대한 Function Shipping 지원이 없으므로 이 연관관계 유형은 두 트랜잭션이 동일 CICS 리전으로 라우팅되어야 함을 의미합니다.

참고: ENQ에 시스플렉스 전체 범위를 제공하기 위해 적합한 ENQMODEL 자원 정의를 사용하지 않는 한 두 트랜잭션이 큐에 넣기(ENQ) 메커니즘을 사용하여 동기화되는 경우에도 이는 참입니다. ENQMODEL 자원 정의 *CICS* 자원 정의 안내서의 내용을 참조하십시오.

트랜잭션 시스템 연관관계

다른 트랜잭션이 아닌 특정 시스템에 대한 연관관계가 있는 트랜잭션은 동적 라우팅에 적합하지 않습니다. 일반적으로 이들은 INQUIRE 및 SET 명령을 사용하는 트랜잭션이거나 글로벌 사용자 엑시트 프로그램에 대한 약간의 종속성이 있는 트랜잭션이며 이 또한 특정 CICS 리전을 가진 연관관계가 있습니다.

INQUIRE 및 SET 명령 및 글로벌 사용자 엑시트 사용

INQUIRE 및 SET 명령에는 원격(즉, Function Shipping) 지원이 없고 이들에 대한 SYSID 옵션도 없으므로 이러한 명령을 사용하는 트랜잭션은 이들이 참조하는 자원을 소유하는 CICS 리전으로 라우팅되어야 합니다. 일반적으로 이러한 트랜잭션은 대상 리전에 동적으로 라우팅될 수 없으므로 INQUIRE 및 SET를 사용하는 트랜잭션은 동적으로 라우팅되어야 합니다.

서로 다른 CICS 리전에서 실행 중인 글로벌 사용자 엑시트는 데이터를 교환할 수 없습니다. 사용자 트랜잭션은 사용자 엑시트를 사용하여 데이터나 매개변수를 전달하는 경우가 거의 없지만 이러한 트랜잭션이 있는 경우에는 이들은 글로벌 사용자 엑시트와 동일한 대상 리전에서 실행되어야 합니다.

프로그래밍 기술 및 연관관계

트랜잭션간 연관관계의 관점에서 동적 또는 분산된 라우팅 환경에서 애플리케이션이 사용하는 프로그래밍 기술은 안전, 위험 또는 의심으로 분류될 수 있습니다.

- 안전한 기술은 트랜잭션간 연관관계를 야기시키지 않습니다.
- 위험한 기술은 본질적으로 트랜잭션간 연관관계를 야기시킵니다.
- 의심스러운 기술은 정확히 구현되는 방법에 따라 연관관계를 작성하거나 작성하지 않을 수 있습니다.

안전한 기술

안전한 프로그래밍 기술은 연관관계의 필요성을 피합니다. 안전한 범주에서 프로그래밍 기술은 다음과 같습니다.

- 수많은 CICS 명령에서 CICS API가 지원하는 통신 영역의 사용. 그러나 허위 대화형 트랜잭션에서 다음 트랜잭션에 전달되는 RETURN 명령에 COMMAREA이기 때문에, 트랜잭션 연관관계에 관하여 동적이거나 분포된 라우팅 환경에 흥미가 있는 CICS RETURN 명령만 COMMAREA 옵션입니다.
- 선택적으로 CICS에 정의된 각 터미널에 사용 가능한 TCT 사용자 영역(TCTUA)의 사용.
- 동기화 또는 태스크의 직렬화가 CICS 명령을 사용합니다.
 - 시스플렉스 전체 범위를 ENQ에 제공하기 위해 적합한 ENQMODEL 자원 정의를 사용한 경우 ENQ/DEQ. ENQMODEL의 설명에 대해서는 387 페이지의 『ENQMODEL 자원 정의와 함께 ENQ 및 DEQ 명령 사용』 및 CICS 자원 정의 안내서의 ENQMODEL 자원 정의를 참조하십시오.
- 데이터를 CICS 비즈니스 트랜잭션 서비스(BTS) 활동 간에 전달하기 위해 컨테이너의 사용. 컨테이너 데이터는 RLS-enabled BTS VSAM 파일에 저장됩니다.

COMMAREA와 TCTUA에 대한 자세한 정보는 384 페이지의 『연관관계를 피하는 프로그래밍 기술』의 내용을 참조하십시오.

위험한 기술

안전하지 않은 프로그래밍 기술은 연관관계의 작성을 요구할 수 있는 기술입니다. 위험한 범주에 있는 프로그래밍 기술은 다음과 같습니다.

- 수명이 긴 공유 저장영역의 사용:
 - CWA(Common Work Area)(CWA)
 - GETMAIN SHARED 저장영역
 - LOAD PROGRAM HOLD에 의해 획득된 저장영역
- 동기화된 태스크에 의해 공유된 태스크 생애 로컬 저장영역의 사용

한 태스크가 일부 태스크 생애 저장영역의 주소를 또 다른 태스크로 전달하는 것이 가능합니다.

소유 태스크가 종료하지 않는 경우 수신 태스크가 전달된 주소를 사용하는 것이 안전할 수도 있습니다. 수신 태스크가 수신인이 주소로 완료하기 전에 전송 태스크가 종료되거나 다른 방법으로 저장영역을 해제하지 못하게 막을 수 있도록 CICS 태스크 동기화 기술을 사용하는 것이 가능한 하지만 권장하지는 않습니다. 그러나 이러한 설계는 전송 태스크가 몇몇 외부 에이전시에 의해 제거될 위험이 있기 때문에 강력하지는 않습니다.

세부사항은 393 페이지의 『태스크 수명 저장영역 공유』의 내용을 참조하십시오.

- 동기화 또는 태스크의 직렬화가 CICS 명령을 사용합니다.
 - WAIT EVENT/WAIT EXTERNAL/WAITCICS
 - 시스플렉스 전체 범위를 ENQ에 제공하기 위해 적합한 ENQMODEL 자원 정의를 사용하지 않는 한 ENQ/DEQ. ENQMODEL의 설명은 387 페이지의 『ENQMODEL 자원 정의와 함께 ENQ 및 DEQ 명령 사용』 및 CICS 자원 정의 안내서의 내용을 참조하십시오.

안전하지 않은 프로그래밍 기술에 대한 자세한 정보는 388 페이지의 『연관관계를 작성하는 프로그래밍 기술』의 내용을 참조하십시오.

의심스러운 기술

몇몇 프로그래밍 기술은 정확히 어떻게 구현되는지에 따라서 연관관계를 작성하거나 작성하지 않을 수도 있습니다. 좋은 예제는 임시 저장영역을 사용하는 것입니다. 이 범주의 기술을 사용하는 애플리케이션은 동적 또는 분배된 라우팅 환경에서 제한사항 없이 작업하는지 여부를 판별하기 위해 확인해야 합니다. 의심스러운 범주의 프로그래밍 기술은 다음과 같습니다.

- 제한적인 이름 지정 규칙이 있는 임시 저장영역 큐의 사용
- 트랜지언트 데이터 큐 및 트리거 레벨의 사용
- 동기화 또는 태스크의 직렬화가 CICS 명령을 사용합니다.
 - RETRIEVE WAIT / START
 - START / CANCEL REQID
 - DELAY / CANCEL REQID
 - POST / CANCEL REQID
- INQUIRE 및 SET 명령 및 글로벌 사용자 엑시트

의심스러운 프로그래밍 기술에 대한 자세한 정보는 396 페이지의 『연관관계를 작성할 수 있는 프로그래밍 기술』의 내용을 참조하십시오.

권장사항

트랜잭션간 연관관계를 처리하기 위한 최상의 방법은 트랜잭션간 연관관계를 처음부터 작성하는 것을 피하는 것입니다. 연관관계를 피하는 것이 가능하지 않은 경우에는 다음을 수행해야 합니다.

- 적합한 이름 지정 규칙을 사용하여 트랜잭션간 연관관계를 쉽게 인식 가능하게 만듭니다.
- 연관관계의 수명을 가능한 짧게 유지합니다.

애플리케이션을 변경하여 트랜잭션간 연관관계를 피할 수 있다라도 연관관계에 대처하기 위해 동적 또는 분배된 라우팅 프로그램에 로직을 포함하는 경우에는 이는 필요하지 않습니다. 마침내 영향 받은 트랜잭션을 정적으로 라우팅할 수 있습니다.

연관관계를 피하는 프로그래밍 기술

트랜잭션 사이에 데이터를 전달하기 위한 일부 기술은 트랜잭션간 연관관계를 작성하지 않는 안전한 일반적인 방법입니다. 여기에는 통신 영역(COMMAREA), 터미널 제어 테이블 사용자 영역(TCTUA) 또는 BTS 컨테이너의 사용이 포함됩니다.

연관관계 작성을 피하려면, 주소를 포함하기 위해 COMMAREA, TCTUA 및 BTS 컨테이너를 사용하지 마십시오. 일반적으로, 이와 같은 주소가 참조하는 저장영역은 수명이 긴 저장영역여야 합니다. 이러한 저장영역 사용은 동적 트랜잭션 라우팅 환경에서 불안정한 프로그래밍 기술입니다.

COMMAREA

RETURN 명령에서 COMMAREA 옵션의 사용은 CICS 의사 대화식 트랜잭션의 연속적인 트랜잭션 사이에 데이터를 전달하는 데 사용할 수 있는 안전한 프로그래밍 기술의 주요 예제입니다.

애플리케이션이 아니라 저장영역에 대해 GETMAIN 및 FREEMAIN 요청을 발행하는 것이 CICS임에도 불구하고 CICS는 COMMAREA를 사용자 저장영역의 특수한 양식으로 취급합니다.

CICS는 RETURN 명령에 지정되는 COMMAREA의 내용이 항상 다음 트랜잭션의 첫 번째 프로그램에 사용 가능하게 되도록 보장합니다. 이것은 전송 및 수신 트랜잭션이 서로 다른 대상 리전에서 실행할 때도 성립합니다. 의사 대화식에서, 동적 라우팅 프로그램이 다음 트랜잭션을 라우팅하는 대상 리전과 상관 없이 CICS는 이전 RETURN 명령에 지정된 COMMAREA가 대상 리전에서 사용 가능하게 되도록 합니다. 이것이 385 페이지의 그림 88에 설명되어 있습니다.

COMMAREA의 일부 일반적인 특성은 다음과 같습니다.

- 프로세서 오버헤드가 낮습니다.
- 복구 가능하지 않습니다.

- RETURN의 COMMAREA 길이는 최대 32 763바이트의 이론적 상한까지 트랜잭션에 따라 다를 수 있습니다. (그러나 안전하려면 이론적 최대값에서 한계를 줄일 수 있는 많은 요인 때문에 애플리케이션 프로그래밍 참조 매뉴얼에서 권장하는 대로 24KB(1KB = 1024바이트)를 초과해서는 안됩니다.)
- CICS는 터미널 사용자가 다음 트랜잭션으로 응답할 때까지 CICS 기본 저장영역에 COMMAREA를 보유합니다. 이것은 큰 COMMAREA를 사용 중일 때 중요한 고려사항일 수 있습니다. CICS가 보유하는 COMMAREA 수가 임의의 시점에서 리전에 있는 태스크의 최대 수가 아니라 터미널 사용량과 관련되기 때문입니다.
- 다음 트랜잭션의 첫 번째 프로그램이 명시적으로 데이터를 또 다른 프로그램이나 후속 트랜잭션으로 전달하지 않는 한, COMMAREA는 해당 프로그램에만 사용 가능합니다.

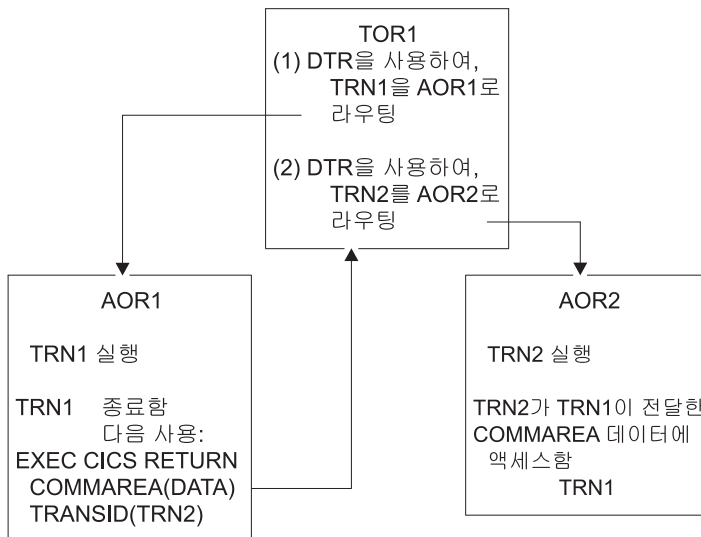


그림 88. 동적 트랜잭션 라우팅 환경에서 의사 대화에 COMMAREA의 사용

그림 88에 표시된 것처럼 의사 대화식 트랜잭션에서 사용되는 COMMAREA는 CICSplex를 통해 트랜잭션 사이에서 전달될 수 있으며, COMMAREA가 데이터만 포함하고 저장영역의 주소를 포함하지 않는 경우 트랜잭션간 연관관계는 작성되지 않습니다.

TCTUA

TCTUA는 터미널 제어 테이블 항목(TCTTE)의 선택적 확장 기능으로, 각 항목은 확장 기능이 존재하는지 여부 및 해당 길이를 지정합니다.

TYPETERM 자원 정의의 USERAREALEN 속성에서 해당 길이를 정의하여 TCTUA가 터미널과 연관되기 원함을 지정합니다. 이것은 TCTUA가 동일한 TYPETERM 정의를 사용하여 작성되는 모든 터미널에 대해 고정 길이임을 의미합니다.

TCTUA(TERMINAL CONTROL TABLE USER AREA)는 의사 대화식 트랜잭션에서 연속적인 트랜잭션 사이에서 데이터를 전달하는 수단으로서 동적 트랜잭션 라우팅

환경에서 안전하게 사용할 수 있습니다. COMMAREA 처럼, TCTUA는 의사 대화의 트랜잭션이 서로 다른 대상 리전으로 라우팅될 때도 항상 사용자 터미널에서 시작된 트랜잭션에 액세스할 수 있습니다. 이것이 그림 89에 설명되어 있습니다. TCTUA의 일부 다른 일반 특성은 다음과 같습니다.

- 최소 프로세서 오버헤드(주소를 얻기 위해 단 하나의 CICS 명령이 필요합니다.)
- 복구 가능하지 않습니다.
- 길이는 주어진 TYPETERM 정의와 연관된 터미널 그룹에 대해 고정됩니다. 허용되는 최대 크기가 255바이트인 소량의 데이터에 대해서만 적합합니다.
- 터미널이 자동 설치되는 경우, TCTUA는 AILDELAY 시스템 초기화 매개변수에 의해 판별되는 보유 기간인 TCTTE 동안 지속됩니다. TCTTE 및 모든 연관된 TCTUA는 CICS와 터미널 사이의 세션이 종료된 후 AILDELAY 간격이 만기할 때 삭제됩니다.

터미널이 명시 터미널 정의에 의해 CICS에 정의된 경우, TCTTE 및 그의 연관된 TCTUA는 터미널이 설치될 때 작성되며 CICS의 다음 초기 또는 콜드 스타트까지 남아 있습니다.

TCTUA는 라우팅 리전의 동적 라우팅 환경뿐 아니라 대상 리전의 애플리케이션에 사용 가능합니다. 트랜잭션의 동적 라우팅과 관련된 정보를 저장하는 데 사용할 수 있습니다. 예를 들어, TCTUA를 사용하여 트랜잭션이 라우팅되는 선택된 대상 리전의 이름을 저장할 수 있습니다.

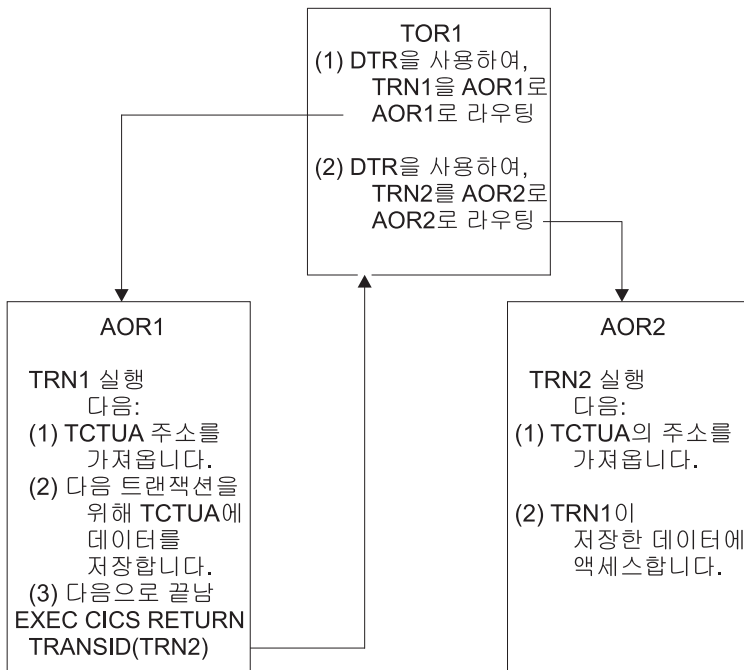


그림 89. 동적 라우팅 환경에서 의사 대화에 의한 TCTUA의 사용

TCTUA를 안전하지 않은 방법으로 사용:

EXEC CICS ADDRESS TCTUA(ptr-ref)는 TCTUA에 직접적인 주소 지정 능력을 제공하고 이는 TCTUA에 대한 액세스를 필요로 하는 각 태스크가 TCTUA 주소를 획득해야 하는 방법입니다. 태스크가 TCTUA의 주소를 임시 저장영역 큐 등과 같은 다른 방법으로 전달하려고 시도하거나 TCTUA 자체를 사용하여 다른 저장영역의 주소를 전달하려고 하면 TCTUA는 동적 트랜잭션 라우팅 환경에서 사용하기 위해 안전한 프로그래밍 방법의 제공을 중지합니다.

태스크가 다른 태스크와 연관된 터미널을 이름 지정하는 INQUIRE TERMINAL 명령을 실행하여 자체 프린시פל 기능이 아니라 프린시פל 기능의 TCTUA를 획득하는 것도 가능합니다. (INQUIRE TERMINAL 명령은 지정된 터미널의 TCTUA 주소를 리턴합니다.) 태스크의 자체 프린시פל 기능이 아니라 터미널의 TCTUA 주소를 사용하는 것은 TCTUA 기능의 안전하지 않은 사용의 또 다른 예입니다. 상황에 따라서 특히, 동적 라우팅 환경에서는 조회하는 태스크의 프린시פל 기능이 아닌 터미널의 TCTUA는 주소를 획득한 후에는 삭제할 수 있습니다. 예를 들어, 대상 리전에서 INQUIRE TERMINAL 명령은 동적으로 라우팅된 트랜잭션을 실행 중인 대리 터미널과 연관된 TCTUA 주소를 리턴할 수 있습니다. 터미널로부터 다음 트랜잭션이 다른 대상 리전으로 라우팅되면 TCTUA 주소는 더 이상 유효하지 않게 됩니다.

ENQMODEL 자원 정의와 함께 ENQ 및 DEQ 명령 사용

ENQ 및 DEQ 명령은 공유 자원에 대한 액세스를 일련화하는 데 사용됩니다. CICS는 ENQMODEL 자원을 z/OS 글로벌 자원 직렬화와 함께 사용하여 여러 애플리케이션이 사용하는 자원의 시스플렉스 전체 보호를 제공합니다.

시스플렉스 큐에 넣기는 트랜잭션간 연관관계의 중요한 원인을 제거하고 시스플렉스 전체에 걸쳐 단일 스레딩과 동기화를 허용합니다. 이는 공유 임시 저장영역 큐에 대한 동시 갱신의 직렬화를 가능하게 하고 서로 다른 CICS 리전에서 동시 태스크가 작성한 레코드를 원격 트랜지언트 데이터 큐에 인터리빙하는 것을 방지합니다. 시스플렉스 전체 큐에 넣기는 주소에 큐에 넣기가 아니라 자원에 대해서만 지원됩니다. 이는 복구 가능한 자원의 잠금을 위해 설계되지 않았습니니다.

단일 CICS 리전 내에 있는 로컬 큐에 넣기는 CICS 주소 공간에서 관리됩니다. 둘 이상의 CICS 리전에 영향을 미치는 시스플렉스 전체 큐에 넣기는 z/OS 글로벌 자원 직렬화에 의해 관리됩니다. 글로벌 자원 직렬화에 대한 자세한 정보는 z/OS MVS Planning: Global Resource Serialization의 내용을 참조하십시오. CICS의 z/OS 글로벌 자원 직렬화 구성에 대한 자세한 정보는 글로벌 CICS 큐 넣기 및 큐 해제: 성능 개선에서 성능 개선의 내용을 참조하십시오.

EXEC CICS ENQ 및 EXEC CICS DEQ 명령이 자원에 대해 발행되면 CICS는 일치하는 설치된 ENQMODEL 정의를 확인합니다. ENQMODEL 자원의 ENQSCOPE 속성은 동일한 큐에 넣기 범위를 공유하는 리전 세트를 정의합니다. ENQSCOPE 속성이 공백

(기본값)으로 남아 있으면 CICS는 일치하는 ENQ 또는 DEQ 요청을 발행 중인 CICS 리전에 로컬인 것으로 처리합니다. ENQSCOPE가 공백이 아니면 CICS는 ENQ 또는 DEQ를 시스플렉스 전체로 처리하고 큐에 넣기를 관리하기 위해 큐 이름과 자원 이름을 z/OS 글로벌 자원 직렬화로 전달합니다. 시스플렉스 전체 큐에 넣기 또는 큐에서 해제 기능을 사용해야 하는 CICS 리전은 모두 필수 ENQMODEL 자원이 정의되고 설치되어 있어야 합니다. ENQMODEL 자원에 대한 자세한 정보는 참조 -> 시스템 정의에서 ENQMODEL 자원의 내용을 참조하십시오.

애플리케이션은 적합한 ENQMODEL 자원을 정의할 때 시스플렉스 큐에 넣기를 사용합니다. 애플리케이션을 변경할 필요가 없습니다. 자원 이름이 동적으로 구성된 애플리케이션의 경우 미리 알려지지 않았으므로 큐에 넣기 EXEC 인터페이스 프로그램 종료 XNQREQ 및 XNQREQC를 사용하여 자원 이름의 시작 시에 적합한 ENQMODEL 자원 정의와 일치하는 문자를 제공할 수 있습니다. 이러한 사용자 엑시트에 대한 자세한 정보는 참조 -> 시스템 프로그래밍에서 XNQREQ 및 XNQREQC, 큐 넣기 EXEC 인터페이스 프로그램 엑시트의 내용을 참조하십시오.

BTS 컨테이너

컨테이너는 BTS 활동의 소유이며 BTS 활동 사이에서 또는 동일한 활동의 서로 다른 활성화 사이에서 데이터를 전달하는 데 사용될 수 있습니다.

활동 밖에서는 컨테이너를 사용할 수 없습니다. 자세한 정보는 *CICS Business Transaction Services* 매뉴얼을 참조하십시오. 활동은 GET 및 PUT 컨테이너를 사용하여 컨테이너의 내용을 갱신합니다. CICS는 RLS 사용 가능 VSAM 파일에 BTS 활동과 연관된 모든 정보(컨테이너 포함)를 저장하여 적합한 컨테이너가 활동에 사용 가능하게 합니다. 이 때문에 BTS 환경은 sysplex 밖으로 확장할 수 없음을 주의하십시오(*CICS Business Transaction Services* 참조). 그러나 데이터를 컨테이너로 전달하는 sysplex 내에서 동적 라우팅을 사용할 수 있습니다.

컨테이너의 일부 일반적 특성은 다음과 같습니다.

- 활동이 임의 숫자의 컨테이너를 소유할 수 있습니다. 하나로 제한되지 않습니다.
- 크기 제한이 없습니다.
- 복구 가능합니다.
- 연관된 활동이 실행 중인 동안에만 기본 저장영역에 존재합니다. 그렇지 않으면 디스크에 보유됩니다. 그러므로, 터미널 COMMAREA와는 달리 저장영역 요구사항에 대해 과도하게 걱정할 필요가 없습니다.

연관관계를 작성하는 프로그래밍 기술

일부 CICS 애플리케이션 프로그래밍 기술(특히 공유 저장영역에 주소를 전달하거나 확보하는 기술)은 트랜잭션 사이에 연관관계를 작성합니다.

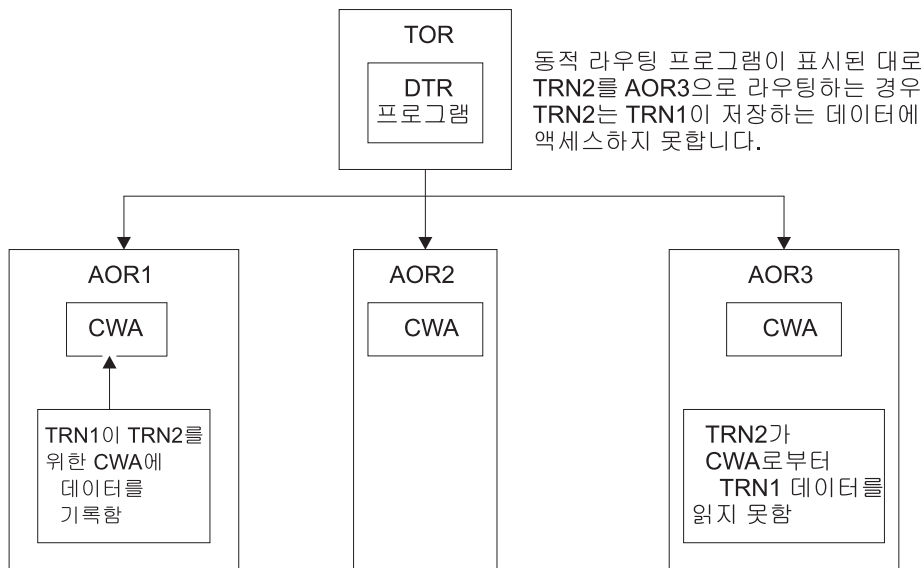
일반적으로 안전하지 않은 프로그래밍 기술은 다음 절에서 설명됩니다.

- 『CWA(Common Work Area) 사용』
- 390 페이지의 『GETMAIN SHARED 저장영역 사용』
- 391 페이지의 『LOAD PROGRAM HOLD 명령 사용』
- 393 페이지의 『태스크 수명 저장영역 공유』
- 394 페이지의 『WAIT EVENT 명령 사용』
- 395 페이지의 『ENQMODEL 자원 정의 없이 ENQ 및 DEQ 명령 사용』

CWA(Common Work Area) 사용

CICS 리전의 CWA는 CICS 초기화 동안(선택적으로) 작성되고, CICS가 종료할 때까지 존재하며, CICS 재시작(웜 또는 긴급) 시에 복구되지 않습니다. ADDRESS CWA (*ptr-ref*) 명령이 CWA에 대한 직접 주소지정 가능성을 제공합니다.

CWA 같이 수명이 긴 공유 저장영역의 사용이 연관관계를 작성할 수 있는 방법의 좋은 예가 한 태스크가 CWA에 데이터를 저장하고 나중 태스크가 그로부터 데이터를 읽을 때입니다. 명백하게, 데이터를 수신하는 태스크는 데이터를 저장한 태스크와 동일한 대상 리전에서 실행해야 합니다. 그렇지 않으면 다른 주소 공간의 다른 저장영역을 참조합니다. 이것은 그림 90에서 보는 것처럼 동적 또는 분산 라우팅 프로그램의 워크로드 밸런스 기능을 제한합니다.



CWA

그림 90. CWA 사용에 의해 작성되는 트랜잭션간 연관관계의 설명. 동적 라우팅 프로그램은 이 CWA 연관관계를 인식해야 하며, TRN2를 TRN1과 동일한 대상 리전으로 라우팅하도록 보장합니다.

그러나 CWA가 읽기 전용 데이터를 포함하고 이 데이터가 둘 이상의 대상 리전에 복제되는 경우, CWA를 사용하고 계속해서 동적 라우팅의 전체 이점을 갖는 것이 가능

합니다. 예를 들어, 수많은 선택된 대상 리전의 각각에 있는 읽기 전용 데이터와 함께 CWA를 로드하는 CICS 시동(PLTPI 프로그램)의 사후 초기화 단계 중에 프로그램을 실행할 수 있습니다. 이 방법으로 동일한 CWA 데이터와 함께 로드되는 대상 리전으로 라우팅되는 모든 트랜잭션은 트랜잭션이 라우팅되는 대상 리전과 상관없이 데이터에 대한 동일한 액세스 권한을 갖습니다. CICS 서브시스템 저장영역 보호를 사용하면 CICS 키 저장영역에서 CWA를 요청하여 CWA 데이터의 읽기 전용 무결성을 보장하고, 사용자 키에서 실행하기 위해 CWA를 읽는 모든 프로그램을 정의할 수 있습니다.

GETMAIN SHARED 저장영역 사용

공유 저장영역은 GETMAIN SHARED 명령에 의해 할당되고, 동일한 또는 다른 태스크에 의해 명시적으로 해제될 때까지 할당된 채로 남아 있습니다.

공유 저장영역을 사용하면 공유 저장영역의 수명 동안 실행하는 모든 CICS 태스크 사이에서 데이터를 교환할 수 있습니다. 이 방식으로 설계된 트랜잭션은 동일한 CICS 리전에서 실행해야 올바르게 작업합니다. 동적 또는 분산 라우팅 프로그램은 공유 저장영역을 사용하는 트랜잭션이 동일한 대상 리전으로 라우팅되도록 보장해야 합니다.

391 페이지의 그림 91은 공유 저장영역의 사용법을 보여줍니다.

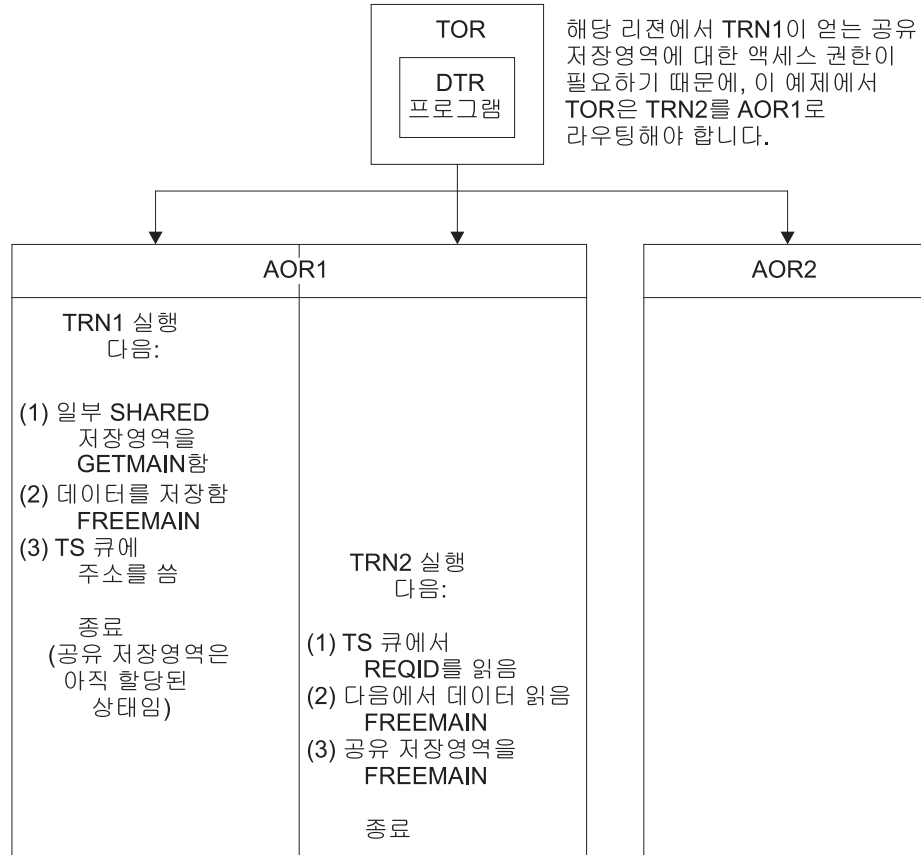


그림 91. 공유 저장영역의 사용에 의해 작성되는 트랜잭션간 연관관계의 설명. 동적 라우팅 프로그램은 이 연관관계를 인식해야 하며, TRN2를 TRN1과 동일한 대상 리전으로 라우팅하도록 보장합니다.

그림 91에 표시된 두 트랜잭션이 의사 대화식 트랜잭션의 일부인 경우, 공유 저장영역의 사용은 **COMMAREA**에 의해 대체되어야 합니다(저장영역의 양이 **COMMAREA** 크기 한계 안에 맞는 경우).

LOAD PROGRAM HOLD 명령 사용

CICS가 **LOAD PROGRAM HOLD** 명령에 대한 응답으로 로드하는 프로그램(또는 테이블)은 동일하거나 서로 다른 태스크에 의해 명시적으로 해제될 때까지 직접 주소 지정 가능한 저장영역에 남아 있습니다.

로드된 프로그램(또는 테이블)이 저장영역에 보유되는 동안 실행하는 모든 CICS 태스크는 다음 경우에 로드된 프로그램의 저장영역을 사용하여 데이터를 교환할 수 있습니다.

- 프로그램이 읽기 전용 저장영역에 로드되지 않거나,
- 프로그램이 **RELOAD(YES)**로 CICS에 정의되지 않습니다.

임시 저장영역 큐를 사용하여 로드된 프로그램 저장영역의 주소를 다른 태스크에 사용 가능하게 만들 수 있지만, 보다 일반적인 방법은 다른 태스크가 SET(ptr_ref) 옵션과 함께 LOAD PROGRAM 명령을 실행하여 CICS가 보유된 프로그램의 주소를 리턴할 수 있게 하는 것입니다.

LOAD PROGRAM HOLD 명령의 사용에 의해 유발되는 연관관계의 네이처는 GETMAIN SHARED 저장영역의 사용에 의해 유발되는 것과 동일하며(391 페이지의 그림 91 및 그림 92 참조), 동일한 규칙이 적용됩니다. 애플리케이션 설계를 보존하기 위해, 동적 또는 분산 라우팅 프로그램은 로드된 프로그램(또는 테이블)의 주소를 사용하는 모든 트랜잭션이 동일한 대상 리전으로 라우팅되도록 보장해야 합니다.

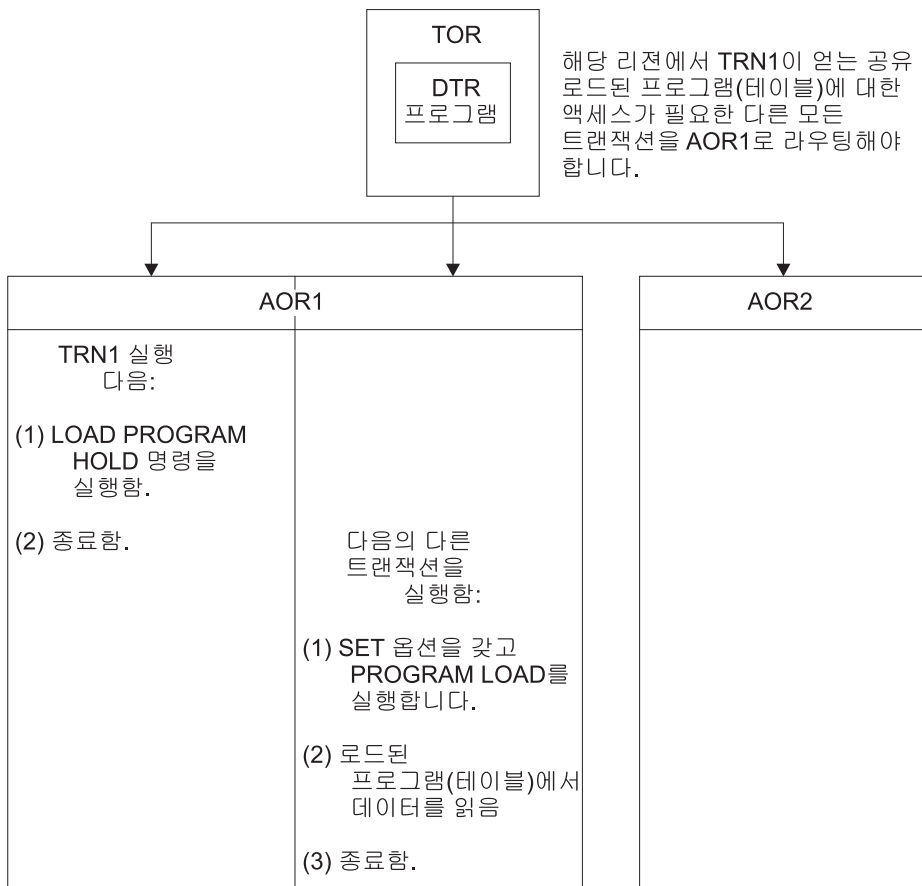


그림 92. LOAD PROGRAM HOLD의 사용에 의해 작성되는 트랜잭션간 연관관계의 설명. 동적 라우팅 프로그램은 이 연관관계를 인식해야 하며, TRN2를 TRN1과 동일한 대상 리전으로 라우팅하도록 보장합니다.

참고: 이 규칙은 로드된 프로그램의 자원 정의에서(HOLD 옵션이 LOAD 명령에서 지정되었는지 여부) RESIDENT 옵션으로 정의되는 프로그램에도 적용됩니다. 그러나, 연관관계 고려사항과 상관없이, RESIDENT와 함께 정의되는 프로그램이 SET

PROGRAM(*program_name*) NEWCOPY 명령에 따르므로 변경될 수 없기 때문에 RESIDENT 옵션을 사용하여 트랜잭션이 데이터를 공유할 수 있게 하는 것은 안전하지 않습니다.

규칙은 또한 통신 태스크가 동기화되는 상주하지 않고 보유되지 않는 로드된 프로그램에도 적용됩니다.

태스크 수명 저장영역 공유

소유 태스크가 동일한 CICS 주소 공백에서 다른 태스크로 주소를 전달할 수 있는 경우, 하나의 태스크에 속하는 태스크 수명 저장영역의 사용은 다른 태스크와 공유할 수 있습니다.

이 기술은 통신 태스크 사이에 연관관계를 작성하고, 전달된 주소를 검색하고 사용하는 태스크는 태스크 수명 저장영역을 소유하는 태스크와 동일한 대상 리전에서 실행해야 합니다.

예를 들어, PL/I 자동 변수의 주소나 COBOL 작업 중인 저장영역 구조의 주소를 전달하기 위해 임시 저장영역 큐를 사용할 수 있습니다. (예제는 그림 93의 내용을 참조하십시오.)

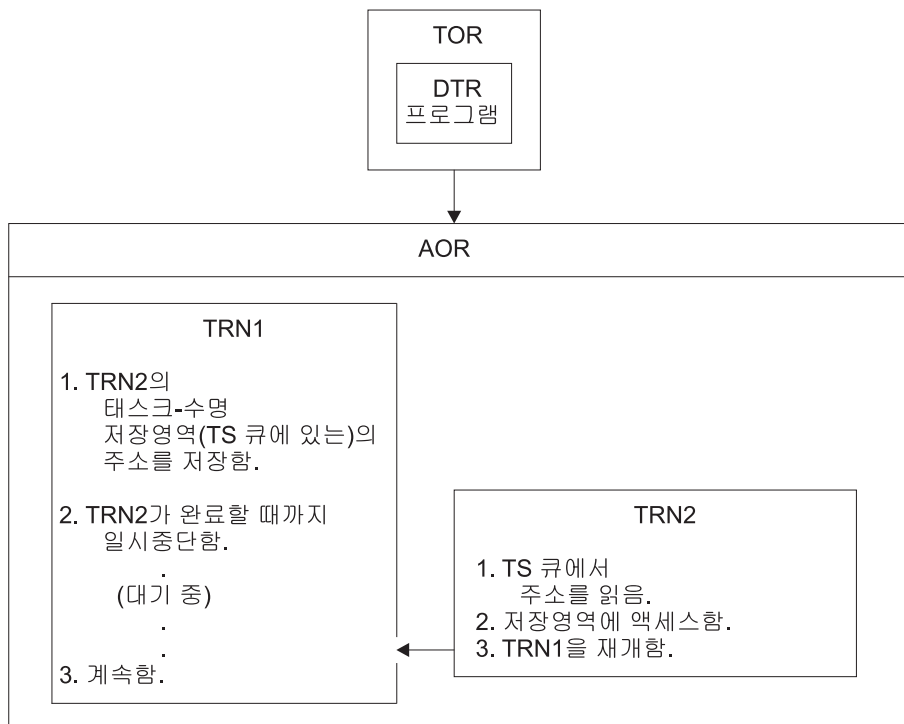


그림 93. 태스크 수명 저장영역 사용으로 작성된 트랜잭션간 연관관계의 그림. TRN2는 TRN1과 동일한 대상 리전에서 실행해야 합니다. 또한 TRN2가 해당 태스크 수명 저장영역을 사용하여 완료될 때까지 TRN1은 종료하면 안됩니다.

두 개의 태스크가 태스크 중 하나에 속하는 태스크 수명 저장영역을 공유하도록 하려면 태스크가 동일한 방식으로 동기화해야 합니다. 해당되는 로컬 저장영역의 주소를 전달하는 태스크를 일시중단하고 재개하는 방법을 제공하는 명령에 대해서는 표 25의 내용을 참조하십시오.

표 25. 일시중단 및 재개(동기화) 태스크에 대한 방법

일시중단 조작	재개 조작
WAIT EVENT, WAIT EXTERNAL, WAITCICS	POST
RETRIEVE WAIT	START
DELAY	CANCEL
POST	CANCEL
START	CANCEL
ENQ	DEQ

이러한 기술 중 일부에서는 이 기술을 사용하는 트랜잭션이 동일한 대상 리전에서 실행되어야 하며, 이에 대해서는 나중에 설명합니다. 그러나 다른 대상 리전에서 실행 중인 태스크가 동기화될 수 있는 경우에도, 서로 태스크 수명 저장영역의 주소를 전달하는 것이 안전하지 않습니다. 동적 라우팅이 없어도, 표 25에 있는 동기화 기술을 기반으로 하는 디자인은 근본적으로 위험합니다. 저장영역 소유 태스크가 제거될 수 있기 때문입니다.

참고:

1. 태스크 수명 저장영역의 공유를 허용하기 위해 RETRIEVE WAIT/START와 같은 동기화 기술을 사용하는 것은 권장되지 않습니다.
2. START 명령이 원격 시스템으로 라우팅되거나 function-ship되는 경우를 제외하고, 다른 태스크의 태스크 수명 저장영역을 공유하는 태스크가 START 명령으로 시작되는 경우에는 어떠한 트랜잭션간 연관관계도 발생하지 않습니다.

WAIT EVENT 명령 사용

WAIT EVENT 명령은 태스크를 다른 CICS 또는 MVS 태스크에 의해 수행되는 이벤트의 완료와 동기화하는 데 사용됩니다.

이벤트의 완료는 이벤트 제어 블록(ECB)으로 비트 패턴 설정에 의해 신호화(게시)됩니다. 대기하는 태스크와 게시하는 태스크가 둘 다 ECB에 대한 직접 주소 지정 가능성을 가져야 하므로, 두 태스크는 모두 동일한 대상 리전에서 실행해야 합니다. 임시 저장영역 큐의 사용이 대기하는 태스크가 다른 태스크로 ECB의 주소를 전달할 수 있는 한 가지 방법입니다.

이 동기화 기술은 395 페이지의 그림 94에서 설명됩니다.

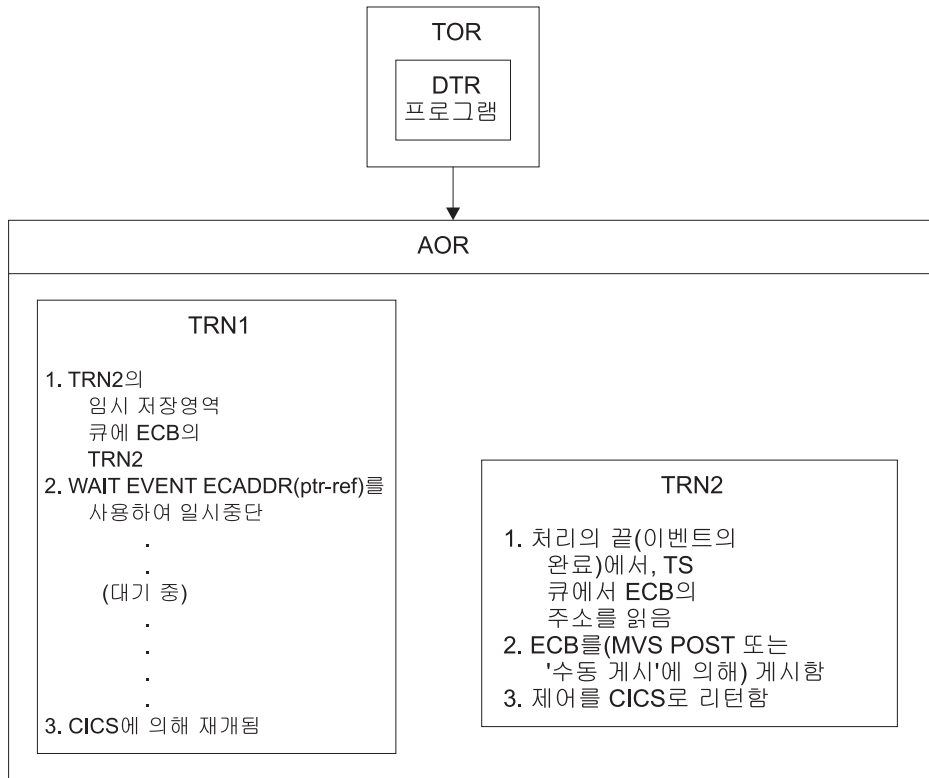


그림 94. *WAIT EXTERNAL* 명령의 사용에 의해 작성되는 트랜잭션간 연관관계의 설명. TRN2는 TRN1과 동일한 대상 리전에서 실행해야 합니다.

그림 94에 표시된 TRN2가 TRN1과 다른 대상 리전에서 실행한 경우, *ptr-ref*의 값이 올바르지 않으며 게시 조작이 예측 불가능한 결과를 갖고 대기하는 태스크가 절대 재개되지 않습니다. 이런 이유로, 동적 또는 분산 라우팅 프로그램은 애플리케이션 설계를 보존하기 위해 게시하는 태스크가 대기하는 태스크와 동일한 대상 리전에서 실행하도록 보장해야 합니다. 동일한 고려사항이 태스크 동기화를 위한 *WAIT EXTERNAL* 및 *WAITCICS* 명령의 사용에 적용됩니다.

ENQMODEL 자원 정의 없이 ENQ 및 DEQ 명령 사용

ENQ 및 DEQ 명령은 공유 자원에 대한 액세스를 일련화하는 데 사용됩니다. 이들 명령은 동일한 리전에서 실행하는 CICS 태스크에 대해서만 작업합니다.

ENQ 및 DEQ 명령은 sysplex 전체 범위를 갖도록 적절한 ENQMODEL 자원 정의에 의해 지원되지 않는 한 서로 다른 리전의 태스크가 공유하는 자원에 대한 액세스를 일련화하는 데 사용될 수 없습니다. ENQMODEL의 설명에 대해서는 387 페이지의 『ENQMODEL 자원 정의와 함께 ENQ 및 DEQ 명령 사용』 및 *CICS* 자원 정의 안내서의 ENQMODEL 자원 정의를 참조하십시오.

LENGTH 옵션을 지정하지 않는 모든 ENQ는 주소의 큐에 넣기로서 취급되므로 로컬 범위만 가짐을 주의하십시오. 직렬화를 위한(sysplex 전체 범위를 제공하기 위한

ENQMODEL 정의 없이) ENQ 및 DEQ의 사용은 그림 95에서 설명되어 있습니다.

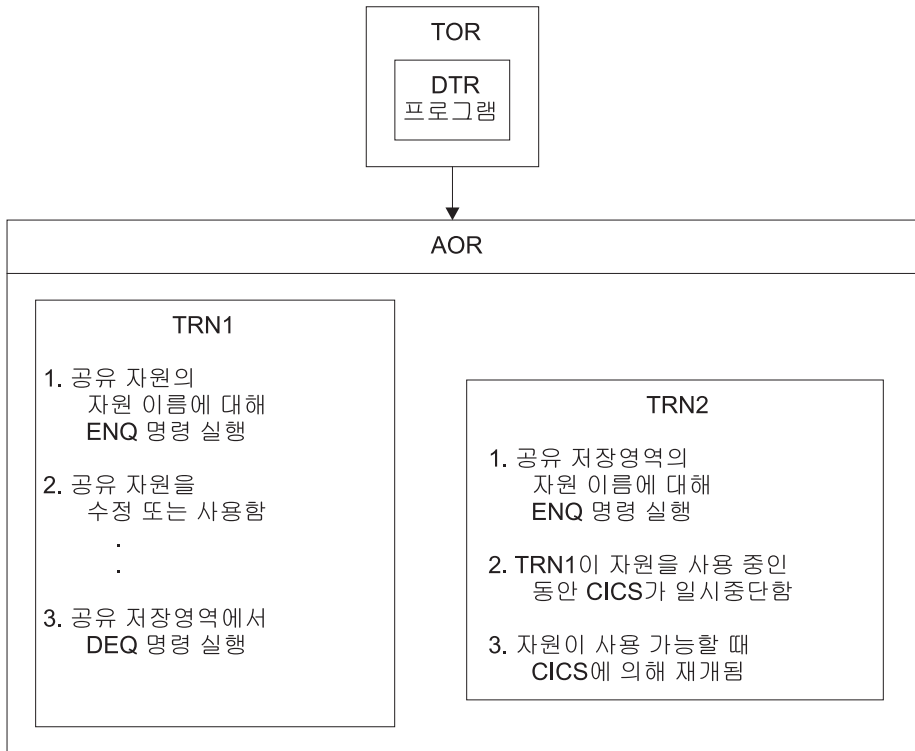


그림 95. ENQ/DEQ 명령을 사용하여 작성된 트랜잭션간 연관관계의 설명. TRN2는 TRN1과 동일한 대상 리전에서 실행해야 합니다.

그림 95에 표시된 TRN2가 TRN1과 다른 대상 리전에서 실행한 경우, TRN1이 공유 자원에 액세스한 동안 TRN2가 일시중단되지 않습니다. 이런 이유로 동적 또는 분산 라우팅 프로그램은 주어진 이름에서 큐에 넣는 모든 태스크가 애플리케이션 설계를 보존하기 위해 동일한 대상 리전에서 실행해야 함을 보장해야 합니다. 물론 TRN2는 대상 리전의 동일한 자원 이름에 대해 ENQ 명령을 실행하는 다른 CICS 태스크와 일련 화됩니다.

연관관계를 작성할 수 있는 프로그래밍 기술

일부 CICS 애플리케이션 프로그래밍 기술은 구현되는 방법에 따라 트랜잭션 사이에 연관관계를 작성할 수 있습니다. 원하지 않는 연관관계를 방지하기 위해 다양한 접근방식을 채택할 수 있습니다.

- 397 페이지의 『임시 저장영역 사용 시 연관관계 피하기』
- 400 페이지의 『트랜지언트 데이터 사용』
- 401 페이지의 『RETRIEVE WAIT 및 START 명령 사용 시 연관관계 피하기』
- 403 페이지의 『START 및 CANCEL REQID 명령 사용 시 연관관계 피하기』
- 405 페이지의 『DELAY 및 CANCEL REQID 명령을 사용한 연관관계 피하기』

- 407 페이지의 『POST 및 CANCEL REQID 명령을 사용한 연관관계 피하기』

임시 저장영역 사용 시 연관관계 피하기

CICS 애플리케이션은 공통적으로 임시 저장영역 큐를 사용하여 임시 애플리케이션 데이터베이스를 보유하고 스크래치 패드로서 작용합니다.

가끔 임시 저장영역 큐가 트랜잭션의 한 인스턴스에서 실행하는 애플리케이션 사이에서 (예를 들어, 다중 프로그램 트랜잭션에서 LINK 또는 XCTL 명령에 의해 제어를 전달하는 프로그램 사이에서) 데이터를 전달하는 데 사용됩니다. 그런 임시 저장영역 큐의 사용은 큐가 트랜잭션 인스턴스의 수명 동안만 존재할 것을 요구하며, 따라서 트랜잭션 인스턴스가 단 하나의 대상 리전에서 실행하기 때문에 서로 다른 대상 리전 사이에서 공유될 필요가 없습니다.

참고: 이 나중 문장은 프로그램 중 하나가 분산 프로그램 링크 명령에 의해 링크되고 링크 대상 프로그램이 원격 시스템에 상주하는 다중 프로그램 트랜잭션의 경우에는 엄격히 참이 아닙니다. 이 경우에는 DPL 명령에 의해 링크되는 프로그램이 원격 리전의 다른 CICS 태스크 아래에서 실행합니다. DPL 프로그램으로 데이터 전달을 위한 선호되는 방법은 채널 또는 COMMAREA에 의한 것이지만, 임시 저장영역 큐가 DPL 애플리케이션에서 데이터 전달에 사용되는 경우 큐가 두 리전 사이에서 공유되어야 합니다.

가끔 임시 저장영역 큐가 대상 리전에 특정한 정보를 보유하거나 읽기 전용 데이터를 보유합니다. 이 경우 임시 저장영역 큐가 각 대상 리전에서 복제될 수 있으며, 대상 리전 사이의 데이터 공유는 필요하지 않습니다.

그러나 많은 경우에 임시 저장영역 큐는 트랜잭션 사이에서 데이터를 전달하는 데 사용되며, 이 경우에 큐는 해당 큐를 사용하는 트랜잭션이 임의의 동적으로 선택된 대상 리전에서 실행할 수 있도록 글로벌하게 액세스 가능해야 합니다. 트랜잭션간 연관관계를 피하면서 임시 저장영역 큐를 글로벌하게 액세스 가능하게 만들기 위해, 큐 소유 리전(QOR)의 원격 큐를 사용하거나 결합 기능의 임시 저장영역 데이터 공유 풀에 상주하는 공유 임시 저장영역 큐를 사용할 수 있습니다.

큐 소유 리전의 원격 임시 저장영역 큐

임시 저장영역 요청을 큐 소유 리전(QOR)으로 이동하려면 다음을 수행하십시오.

- 애플리케이션 소유 리전에서 REMOTESYSTEM 및 REMOTEPREFIX(또는 XREMOTEPFX) 속성을 사용하여 TSMODEL 자원을 작성하여 임시 저장영역 큐를 원격으로 정의하십시오.
- 큐 소유 리전에서, 대응하는 TSMODEL 자원을 작성하여 임시 저장영역 큐의 특성을 정의하십시오.

- 중복을 피하려는 경우, 로컬 및 원격 리전 모두에서 설치될 수 있는 TSMODEL 자원을 작성하고, 이들 자원을 애플리케이션 소유 리전과 큐 소유 리전 사이에서 공유할 수 있습니다.

임시 저장영역 큐 이름이 동적으로 생성되는 경우, 주어진 터미널에 고유한 큐 이름을 생성하는 프로세스를 따라야 합니다. 유용한 규칙은 4문자 접두부(예를 들어, 트랜잭션 ID) 및 그 뒤에 오는 접미부로서의 4문자 터미널 ID입니다. 이 형식의 일반 큐 이름은 원격 큐로 정의될 수 있습니다. 이름 지정 규칙이 이 규칙을 준수하지 않는 경우, 큐가 원격으로 정의될 수 없으며 해당 큐에 액세스하는 모든 트랜잭션은 큐가 작성된 대상 리전으로 라우팅되어야 합니다.

임시 저장영역 요청에 대해 글로벌 사용자 엑시트를 사용하여 임시 저장영역 큐 이름을 로컬에서 원격 큐 이름으로 변경할 수 없습니다.

그림 96은 원격 큐 소유 리전의 사용을 설명합니다.

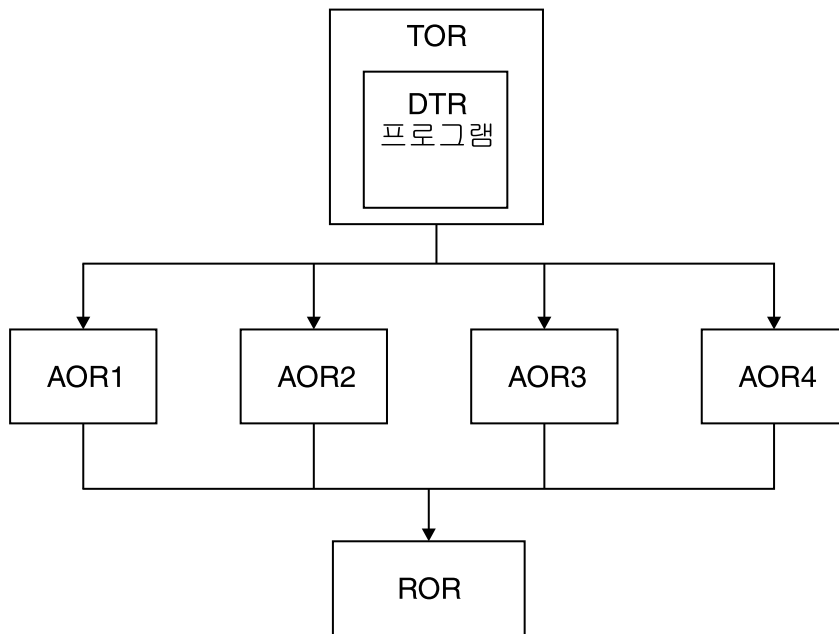


그림 96. 임시 저장영역과 관련된 트랜잭션간 연관관계를 피하기 위한 원격 큐 사용. 이 예제는 결합된 파일 소유 및 큐 소유 리전(자원 소유 리전 또는 ROR)을 보여줍니다. 별도의 리전을 사용할 수 있지만, 파일 제어 및 임시 저장영역 제어에 의해 독립적으로 관리되는 데이터를 복구할 때 발생할 수 있는 '인다우트' 창 때문에 이들은 복구 조작 중에 특별히 주의해야 합니다.

공유 임시 저장영역 큐

결합 기능의 임시 저장영역 데이터 공유 풀에 상주하는 공유 임시 저장영역 큐를 사용하려면, 애플리케이션 소유 리전에서 POOLNAME 및 PREFIX(또는 XPREFIX) 속성을 사용하여 임시 저장영역 요청을 공유 임시 저장영역 큐의 이름 지정된 풀에 맵핑하는 TSMODEL 자원을 작성하십시오.

결합 기능에서 정의되는 임시 저장영역 큐의 각 풀을 지원하도록 임시 저장영역 서버를 설정해야 합니다. 자세한 정보는 구성에서 임시 저장영역 서버 설정 및 실행의 내용을 참조하십시오.

TSMODEL 자원 정의는 임시 저장영역 데이터 공유 풀에 상주하는 큐에 대해 명시 SYSID를 지정하는 애플리케이션을 지원하지 않음을 주의하십시오. 애플리케이션은 대신 QUEUE 또는 QNAME 옵션을 사용해야 합니다. 애플리케이션이 공유 큐 풀에 대해 명시 SYSID를 사용하는 경우, 이것은 임시 저장영역 테이블(TST)의 지원이 필요 합니다.

400 페이지의 그림 97은 결합 기능에서 공유 임시 저장영역 큐의 동일한 풀을 사용 중인 4개의 AOR을 보여줍니다. 임시 저장영역 서버는 항상 큐 서버 리전 프로그램 DFHXQMN에 의해 관리됩니다.

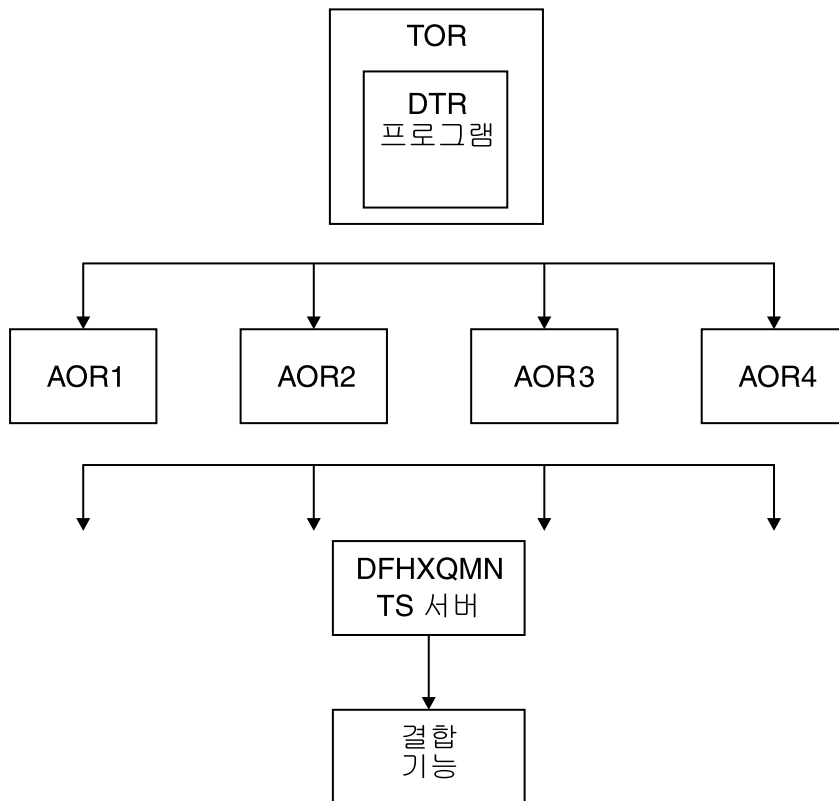


그림 97. 임시 저장영역 데이터 공유 서버 사용의 예

글로벌하게 액세스 가능한 큐에 대한 예외 조건

글로벌 QOR을 사용하여 TS 큐에 관련된 트랜잭션간 연관관계를 제거할 때, 예외 조건 처리도 주의깊게 검토해야 합니다. 트랜잭션 및 큐가 동일한 리전에서 로컬일 때 이전에는 가능하지 않았던 일부 예외 조건이 발생할 수 있습니다.

이 상황은 대상 리전 및 QOR이 독립적으로 실패하여 다음과 같은 상황을 유발할 수 있기 때문에 발생합니다.

- QOR은 계속되는 반면 대상 리전만 실패했기 때문에 큐가 이미 존재합니다.
- 대상 리전이 계속하는 반면 QOR만 실패했기 때문에 큐를 찾을 수 없습니다.

트랜지언트 데이터 사용

CICS 애플리케이션은 일반적으로 트랜지언트 데이터 큐(TD)를 사용합니다. 공유되어야 하는 TD 큐를 사용하는 트랜잭션이 대상 리전에 동적으로 라우팅될 수 있게 하려면 TD 큐가 글로벌하게 액세스 가능한지 확인해야 합니다.

TD 큐에 대한 동적 트랜잭션 라우팅 고려사항은 임시 저장영역의 고려사항과 많은 공통점을 갖습니다.

모든 트랜지언트 데이터 큐가 CICS에 정의되어야 하며, 자원이 CICS 리전에 사용 가능해지기 전에 설치되어야 합니다. 이들 정의는 원격 트랜지언트 데이터 큐 소유 리전 (QOR)을 지원하도록 변경될 수 있습니다.

그러나, 트리거 기능을 사용하는 TD 큐에 대한 제한사항이 있습니다. 트리거 레벨에 도달될 때 호출될 트랜잭션은 큐가 상주하는(QOR의) 리전에서 로컬 트랜잭션으로 정의되어야 합니다. 따라서 트리거 트랜잭션은 QOR에서 실행해야 합니다. 그러나, 큐와 연관된 모든 터미널이 QOR에서 로컬 터미널로서 정의될 필요는 없습니다. 이것은 트랜잭션간 연관관계를 작성하지 않습니다.

그림 98은 원격 트랜지언트 데이터 큐 소유 리전의 사용을 설명합니다.

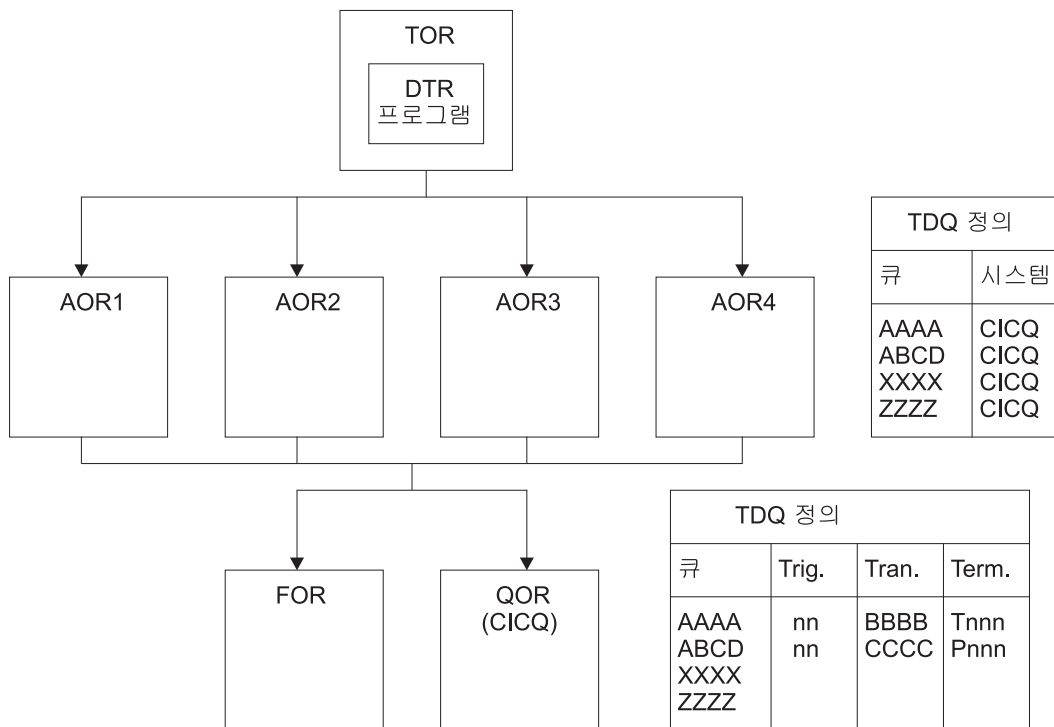


그림 98. 트랜지언트 데이터와 관련된 트랜잭션간 연관관계를 피하기 위한 원격 큐 사용. 대상 리전에 설치된 트랜지언트 데이터 큐 정의는 QOR이 소유하는 것으로 정의됩니다(CICQ). QOR에 설치된 모든 트랜지언트 데이터 큐 정의는 로컬이며 일부는 트리거 레벨을 갖습니다.

글로벌하게 액세스 가능한 큐에 대한 예외 조건

글로벌 QOR을 사용하여 TD 큐에 관련된 트랜잭션간 연관관계를 제거할 때, 어떤 새로운 예외 조건도(시스템 정의 오류 또는 실패가 있는 경우 SYSIDERR 이외의) 있어서는 안됩니다.

RETRIEVE WAIT 및 START 명령 사용 시 연관관계 피하기

일부 동기화 기술의 사용은 두 동기화된 태스크 사이의 태스크 수명 저장영역의 공유를 허용합니다. 이 목적을 위해서는 RETRIEVE WAIT 및 START 명령을 사용하십시오.

그림 99에서 설명하는 예제에서, TRN1은 비동기 태스크 TRN2에서 데이터를 검색하도록 설계되며, 따라서 TRN2가 데이터를 사용 가능하게 할 때까지 대기해야 합니다. 이 메커니즘이 작동하려면 TRN1이 터미널 관련 트랜잭션이어야 함을 주의하십시오.

단계는 다음과 같습니다.

1. TRN1이 TS 큐에 TRANSID 및 TERMID를 포함한 데이터를 씁니다.
2. 스스로 일시중단하게 만들기 위해 TRN1은 RETRIEVE WAIT 명령을 실행하는데, 이 명령은 RETRIEVE가 충족될 수 있을 때까지 CICS가 태스크를 일시중단하게 만들며 TRN2가 FROM 매개변수에 의해 전달되는 데이터와 함께 START 명령을 실행할 때까지가 아닙니다.
3. 그러나 일시중단된 태스크(이 예제에서는 TRN1)의 TRANSID 및 TERMID를 아는 경우에는 TRN2는 START 명령을 실행하여 TRN1을 재개할 수 있습니다. 따라서 TS 큐를 읽어서 TRN1에 의해 기록된 정보를 얻습니다. 임시 저장영역 큐 사용은 일시중단 태스크가 이 정보를 전달할 수 있는 한 가지 방법입니다.
4. TS 큐의 정보를 사용하여 TRN2가 TRN1에 대한 START 명령을 실행하여 CICS가 미해결 RETRIEVE WAIT를 충족하여 TRN1을 재개하게 만듭니다.

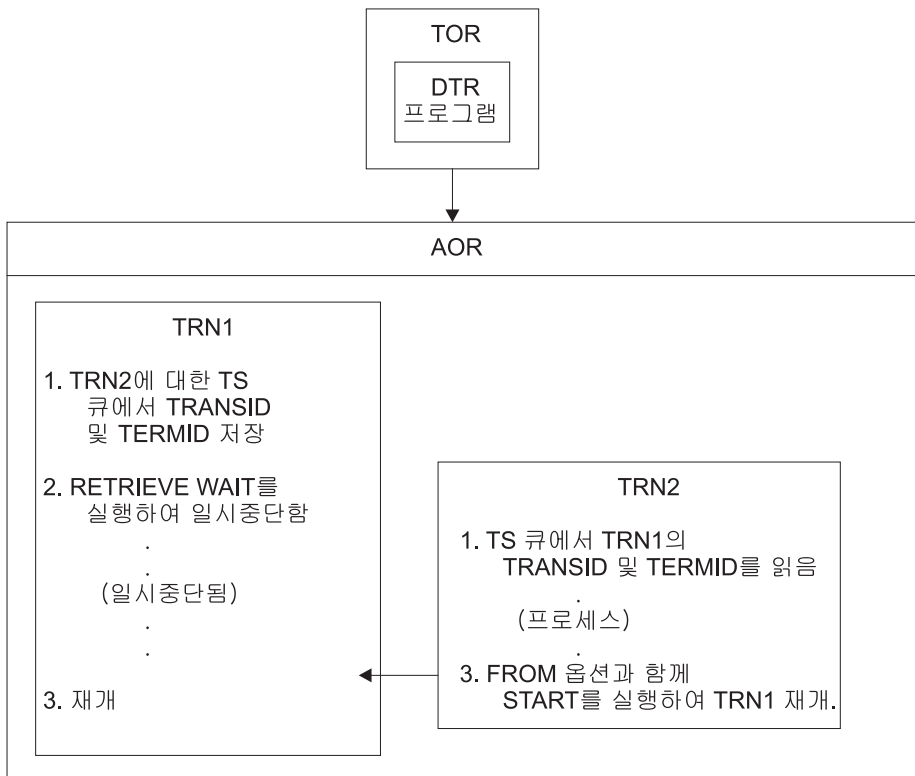


그림 99. RETRIEVE WAIT 및 START 명령을 사용한 태스크 동기화의 설명

402 페이지의 그림 99에 표시된 RETRIEVE WAIT 및 START 명령을 사용한 태스크 동기화의 예에서, RETRIEVE WAIT를 충족하는 START 명령은 다음과 같아야 합니다.

- RETRIEVE WAIT 명령을 실행하는 트랜잭션(본 예제에서는 TRN1)과 동일한 대상 리전에서 실행되어야 합니다. 또는
- RETRIEVE WAIT 명령이 실행된 대상 리전의 SYSID를 지정해야 합니다. 또는
- RETRIEVE WAIT 명령을 실행한 대상 리전에 상주하는 원격 트랜잭션으로 정의되는 TRANSID(본 예제에서는 TRN1)를 지정해야 합니다.

원격 TRANSID 기술에 기반한 애플리케이션 디자인은 두 개의 대상 리전에 대해서만 작동합니다. START 명령에서 SYSID 옵션을 사용하는 애플리케이션 디자인은 모든 대상 리전이 다른 모든 대상 리전에 연결된 경우(바람직하지 않을 수 있음) 다중 대상 리전에 대해서만 작동합니다. 어느 경우이나 애플리케이션이 수정되어야 합니다. 라우팅 프로그램에 제한사항을 부과하는 경우를 제외하면 동적 또는 분산 라우팅 프로그램에서 이 프로그래밍 기술을 사용하는 허용 가능한 방법은 없습니다. 일반적으로 이것은 동적 또는 분산 라우팅 프로그램이 애플리케이션 디자인을 보존하기 위해서는 TRN2가 TRN1과 동일한 리전에서 실행하도록 보장해야 합니다.

START 및 CANCEL REQID 명령 사용 시 연관관계 피하기

CICS START 명령은 트랜잭션이 또 다른 트랜잭션을 시작하기 위해 사용됩니다. 또 다른 트랜잭션은 CANCEL 명령을 사용하고 START 명령과 연관된 REQID를 지정하여 이 명령을 취소할 수 있습니다.

이 CICS 애플리케이션 프로그래밍 기술로 한 트랜잭션이 START 명령을 실행하여 지정된 간격 후에 또 다른 트랜잭션을 시작합니다. 또 다른 트랜잭션(START 명령에서 요청된 것이 아님)이 더 이상 요청된 트랜잭션(REQID 매개변수에 의해 지정된 트랜잭션)을 실행할 필요가 없음을 판별하고 START 요청을 취소합니다.

임시 저장영역 큐가 REQID가 태스크 사이에서 전달될 수 있는 한 가지 방법입니다.

참고: CANCEL 명령은 지정된 간격이 아직 만기하지 않은 경우에만 효과적입니다. 이 기술을 사용하려면 CANCEL 명령이 REQID 옵션을 지정해야 하지만, START 명령은 필요하지 않습니다. 이것은 NOCHECK 옵션이 START 명령에서 지정되지 않는 경우 CICS가 요청에 대한 REQID를 생성하고 이를 EIBREQID 필드의 실행 인터페이스 블록(EIB)에 저장하기 때문입니다.

404 페이지의 그림 100은 이 프로그래밍 기술을 설명합니다..

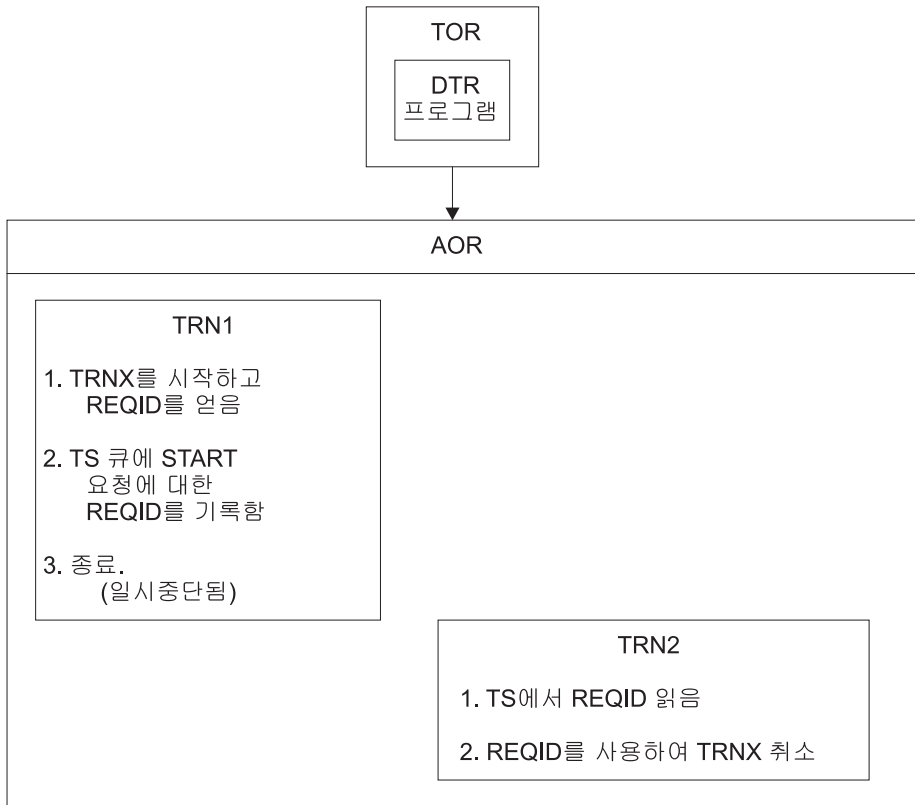


그림 100. START 및 CANCEL REQID 명령 사용의 설명

이 애플리케이션 프로그래밍 기술을 사용할 때 START 요청을 취소하는 CANCEL 명령은 다음과 같아야 합니다.

- START 명령이 실행된 것과 동일한 대상 리전에서 실행되어야 합니다. 또는
- START 명령이 실행된 대상 리전의 SYSID를 지정해야 합니다. 또는
- START 명령이 실행된 대상 리전에 상주하는 원격 트랜잭션으로 정의되는 TRANSID(본 예제에서는 TRNX)를 지정합니다.

참고: START 명령이 반드시 해당 명령을 실행하는 애플리케이션과 동일한 리전에서 실행되지는 않습니다. 제공되는 기능(또는 비터미널 관련 START의 경우, 라우팅)이고 다른 CICS 리전에서 실행될 수 있습니다. 이전 규칙은 START 명령이 최종적으로 실행되는 리전에 적용됩니다.

원격 TRANSID 기술에 기반한 애플리케이션 디자인은 두 개의 대상 리전에 대해서만 작동합니다. 취소 명령에서 SYSID 옵션을 사용하는 애플리케이션 디자인은 모든 대상 리전이 다른 모든 대상 리전에 연결된 경우 다중 대상 리전에 대해서만 작동합니다. 어느 경우이나 애플리케이션이 수정되어야 합니다. 라우팅 프로그램에 제한사항을 부과하는 경우를 제외하면 동적 또는 분산 라우팅 프로그램에서 이 프로그래밍 기술을 사용하는 허용 가능한 방법은 없습니다.

일반적으로, 이것은 동적 또는 분산 라우팅 프로그램이 애플리케이션 디자인을 보존하기 위해 TRN2가 TRN1과 동일한 리전에서 실행하도록 보장하며, 또한 TRNX가 동일한 리전에서 로컬 트랜잭션으로 정의되도록 보장해야 함을 의미합니다.

DELAY 및 CANCEL REQID 명령을 사용한 연관관계 피하기

CICS DELAY 명령은 태스크를 일시중단하는 데 사용됩니다. 또 다른 태스크가 CANCEL 명령을 사용하고 DELAY 명령과 연관된 REQID를 지정하여 이 명령을 취소할 수 있습니다.

이 CICS 애플리케이션 프로그래밍 기술을 사용하면 한 태스크가 DELAY 명령에 의해 일시중단된 다른 태스크를 재개할 수 있습니다.

DELAY 요청은 DELAY 명령을 실행하는 것과는 다른 태스크에 의해서만 취소될 수 있으며, CANCEL 명령은 DELAY 명령과 연관된 REQID를 지정해야 합니다. DELAY 및 CANCEL 명령은 둘 다 이 기술을 사용하기 위해 REQID 옵션을 지정해야 합니다.

임시 저장영역 큐를 사용하여 REQID를 전달하는 이 기술에 포함된 단계는 다음과 같습니다.

1. 태스크(TRN1)가 사전 정의된 DELAY REQID를 TS 큐에 씁니다. 예를 들어, 다음과 같습니다.

```
EXEC CICS WRITEQ TS
      QUEUE('DELAYQUE') FROM(reqid_value)
```

2. 태스크는 DELAY 명령을 실행하고 reqid_value를 REQID로 사용하여 또 다른 태스크를 기다립니다. 예를 들어, 다음과 같습니다.

```
EXEC CICS DELAY
      INTERVAL(1000) REQID(reqid_value)
```

3. 또 다른 태스크인 TRN2가 'DELAYQUE'라는 TS 큐에서 DELAY 요청의 REQID를 읽습니다.

4. TRN2가 처리를 완료하고, DELAY 요청을 취소하여 TRN1을 재개합니다.

TS 큐를 사용한 프로세스가 406 페이지의 그림 101에서 설명됩니다.

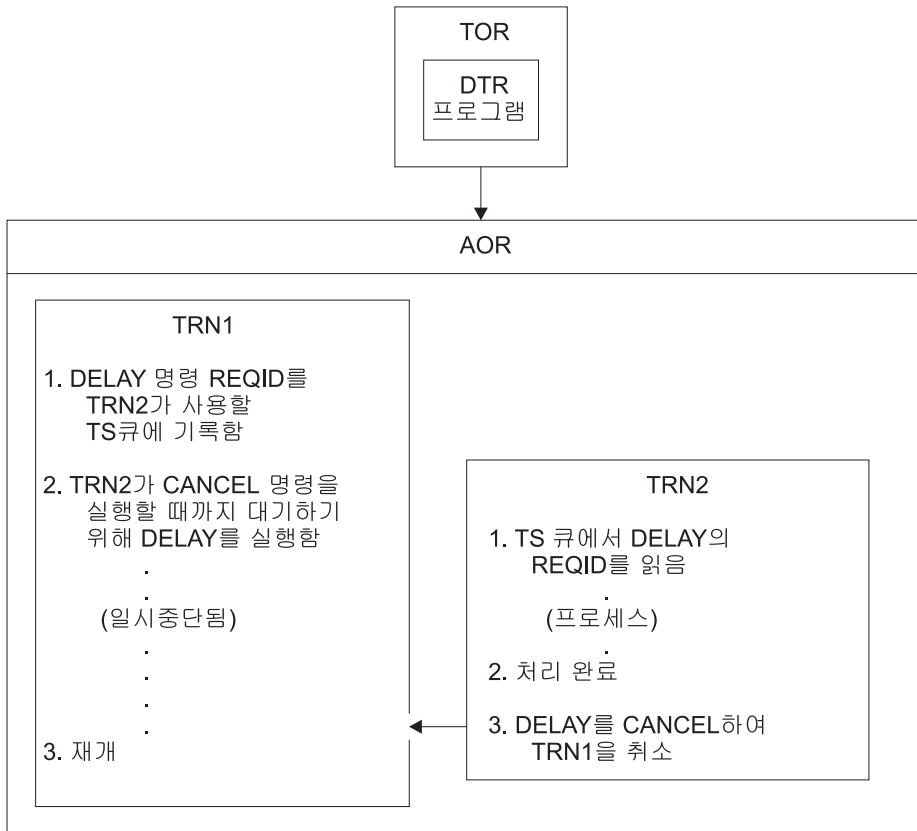


그림 101. DELAY 및 CANCEL REQID 명령 사용의 설명

이 기술을 채택할 때 REQID를 전달하는 또 다른 방법은 TRN1이 FROM 및 TERMID 옵션을 갖는 START 명령을 사용하여 TRN2를 시작하는 것입니다. TRN2는 INTO 옵션을 사용하여 RETRIEVE 명령으로 REQID를 얻을 수 있습니다.

이 애플리케이션 프로그래밍 기술을 사용할 때 DELAY 요청을 취소하는 CANCEL 명령은 다음과 같아야 합니다.

- DELAY 명령이 실행된 것과 동일한 대상 리전에서 실행되어야 합니다. 또는
- DELAY 명령이 실행된 대상 리전의 SYSID를 지정해야 합니다. 또는
- DELAY 명령이 실행된 대상 리전에 상주하는 원격 트랜잭션으로 정의되는 TRANSID(본 예제에서는 TRN1)를 지정합니다.

원격 TRANSID 기술에 기반한 애플리케이션 디자인은 두 개의 대상 리전에 대해서만 작동합니다. 취소 명령에서 SYSID 옵션을 사용하는 애플리케이션 디자인은 모든 대상 리전이 다른 모든 대상 리전에 연결된 경우 다중 대상 리전에 대해서만 작동합니다. 어느 경우여나 애플리케이션이 수정되어야 합니다. 라우팅 프로그램에 제한사항을 부과하는 경우를 제외하면 동적 또는 분산 라우팅 환경에서 이 프로그래밍 기술을 사용하는 허용 가능한 방법은 없습니다.

CANCEL 명령이 터미널에서 시작되는 트랜잭션에 의해 실행되는 경우, 트랜잭션이 동적으로 잘못된 대상 리전으로 라우팅될 수 있습니다.

POST 및 CANCEL REQID 명령을 사용한 연관관계 피하기

CICS POST 명령은 지정된 시간이 만기했다는 알람을 요청하는 데 사용됩니다. 또 다른 트랜잭션(TRN2)은 POST 요청의 CANCEL을 실행하여 지정된 시간이 만기한 것처럼 알람을 강제 실행할 수 있습니다.

시간 한계는 CICS에 의해 이벤트 제어 블록(ECB)에서 비트 패턴을 설정하여 신호화됩니다. 알람이 수신되었는지 여부를 판별하기 위해, 요청하는 트랜잭션(TRN1)은 ECB를 주기적으로 테스트하거나 ECB에 대해 WAIT 명령을 실행합니다.

TS 저장영역 큐를 사용하여 POST 요청의 REQID를 태스크 사이에서 전달할 수 있습니다.

그림 102에서 이 기술을 설명합니다.

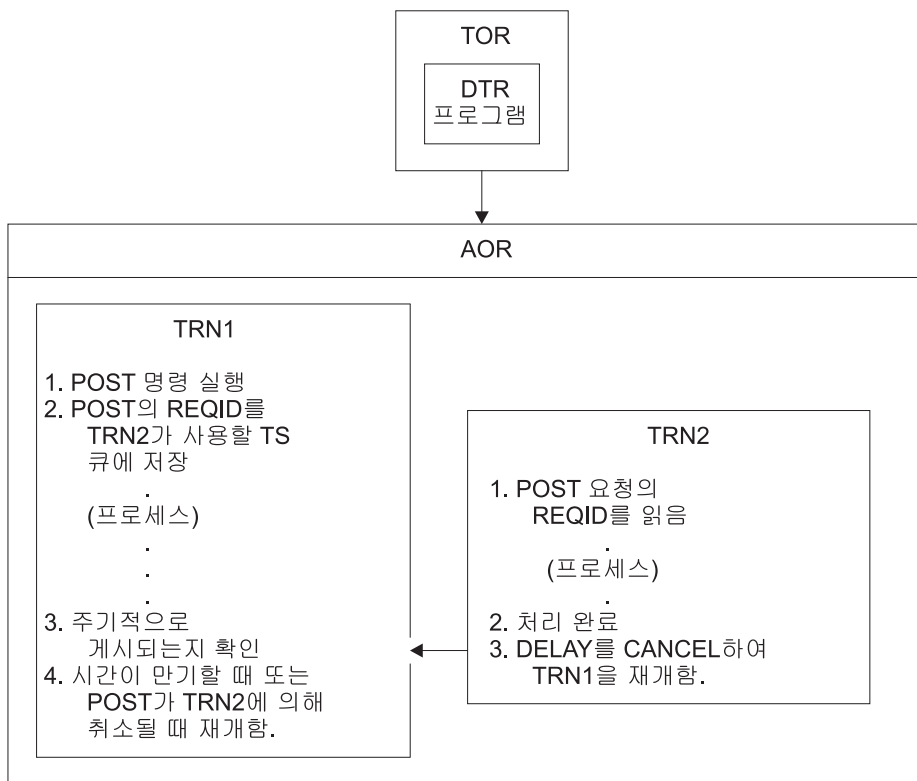


그림 102. POST 명령 사용의 설명

이 기술이 사용되는 경우, 동적 또는 분산 라우팅 프로그램은 애플리케이션 디자인을 보존하기 위해 TRN2가 TRN1과 동일한 CICS 리전에서 실행하는지 확인해야 합니다.

POST를 실행한 태스크를 알리는 CANCEL 명령은 다음과 같아야 합니다.

- POST 명령이 실행된 것과 동일한 대상 리전에서 실행되어야 합니다. 또는

- POST 명령이 실행된 대상 리전의 SYSID를 지정해야 합니다. 또는
- POST 명령이 실행된 대상 리전에 상주하는 원격 트랜잭션으로 정의되는 TRANSID(본 예제에서는 TRN1)를 지정합니다.

원격 TRANSID 기술에 기반한 애플리케이션 디자인은 두 개의 대상 리전에 대해서만 작동합니다. 취소 명령에서 SYSID 옵션을 사용하는 애플리케이션 디자인은 모든 대상 리전이 다른 모든 대상 리전에 연결된 경우 다중 대상 리전에 대해서만 작동합니다. 어느 경우이나 애플리케이션이 수정되어야 합니다. 라우팅 프로그램에 제한사항을 부과하는 경우를 제외하면 동적 또는 분산 라우팅 프로그램에서 이 프로그래밍 기술을 사용하는 허용 가능한 방법은 없습니다.

일반적으로 이것은 동적 또는 분산 라우팅 프로그램이 애플리케이션 디자인을 보존하기 위해서 TRN2가 TRN1과 동일한 리전에서 실행하도록 보장해야 합니다.

명백하게, CANCEL이 POST를 실행한 동일한 태스크에 의해 실행되는 경우 문제가 없습니다. 다른 태스크가 POST 명령을 취소하는 경우, REQID를 지정하여 해당 명령의 특정 인스턴스를 식별해야 합니다. 따라서 REQID를 갖는 CANCEL 명령은 트랜잭션 간 연관관계 문제를 표시합니다. 그러나, CICS는 자동으로 REQID를 생성하고 이를 EIBREQID에서 애플리케이션으로 전달하기 때문에 POST 명령에서 REQID를 지정할 필요는 없습니다.

트랜잭션간 연관관계 발견

동적 라우팅 환경에서 트랜잭션 연관관계를 관리하려면 먼저 어떤 트랜잭션에 연관관계가 있는지를 검색해야 합니다. CICS Interdependency Analyzer를 사용하여 연관관계를 발견하십시오.

이 태스크 정보

유틸리티와 이를 사용하는 방법에 대한 설명은 CICS Interdependency Analyzer for z/OS의 내용을 참조하십시오.

참고: 프로그램 링크 요청을 동적으로 라우팅하는 경우에는 어떤 프로그램(또는 해당 연관된 트랜잭션)에 연관관계가 있는지를 발견해야 합니다. 이를 수행하기 위해 CICS Interdependency Analyzer를 사용할 수 없습니다.

CICS Interdependency Analyzer를 사용하지 않으면 다음 메소드 중 하나를 사용하여 연관관계를 발견할 수 있습니다.

- CICS 제공 로드 라이브러리 스캐너 CAULMS를 사용하십시오.

참고: z/OS용 CICS TS, 버전 3.2부터는 이전에는 CICS 트랜잭션 연관관계 유틸리티의 일부로서 제공된 검출기와 리포터 구성요소는 CICS TS에 포함되지 않습니다. 로드 라이브러리 스캐너 구성요소뿐만 아니라 이러한 구성요소는 상호의존과 연관관계 둘 모두를 분석할 수 있는 CICS Interdependency Analyzer에 통합되어 있

습니다. 로드 라이브러리 스캐너는 단독으로 z/OS용 CICS TS, 버전 3.2에 남아 있고, 잠재적인 연관관계를 가지고 있는 애플리케이션에 대한 보고서를 생성할 수 있습니다.

CICS 로드 라이브러리 스캐너 사용 방법에 대한 세부사항은 CICS TS 2.3 *트랜잭션 유틸리티 안내서*를 참조하십시오.

- 애플리케이션 디자인을 검토하고 트랜잭션간 통신에 사용된 기술에 특별히 주의를 기울이십시오.
- 애플리케이션의 소스를 검색하여 트랜잭션간 연관관계를 초래할 수 있는 EXEC CICS 명령의 인스턴스를 찾아보십시오.
- CICS 보조 추적을 분석할 수 있는 추적 분석 프로그램을 실행하십시오. 예를 들어, CICS 추적 유틸리티 프로그램, DFHTU700을 ABBREV 옵션과 함께 실행하여 CICS 보조 추적 출력을 형식화하는 경우에는 결과로 나오는 요약된 추적 데이터를 분석하여 의심되는 명령의 인스턴스를 찾을 수 있습니다.

애플리케이션 생성기에 의해 초래된 트랜잭션간 연관관계

애플리케이션 생성기는 트랜잭션간 연관관계의 여러 문제점을 일으킬 수 있습니다.

- 연관관계는 애플리케이션 프로그래머로부터 숨겨질 수 있습니다.
- 애플리케이션 생성기는 CICS에 대한 트랜잭션의 다양한 개념이 있을 수 있습니다. 이는 동적으로 라우팅되는 엔티티이므로 중요한 CICS 트랜잭션 중에 연관관계입니다.
- 몇몇 애플리케이션 생성기는 애플리케이션 내의 모든 트랜잭션에 단일 트랜잭션 코드를 사용하여 라우터가 연관관계가 있는 트랜잭션의 인스턴스를 선택하기가 어렵게 합니다.

트랜잭션간 연관관계의 지속 기간 및 범위

동적 또는 분산 라우팅 계획을 수행하고, 트랜잭션간 연관관계를 관리하는 방법을 계획할 때, 연관관계 및 연관관계 수명 개념을 이해하는 것이 중요합니다. 트랜잭션간 연관관계의 수명 및 관련사항은 트랜잭션간 연관관계의 지속 기간 및 범위를 정의합니다.

명확하게, 동적 또는 분배된 라우팅 프로그램에 대한 이상적인 상황은 트랜잭션간 연관관계가 전혀 없는 경우로, 이는 동적 라우팅에 사용 가능한 대상 리전 선택 시 제한사항이 없음을 의미합니다. 그러나 트랜잭션간 연관관계가 존재하는 경우에도, 이 연관관계의 범위, 연관관계로 판별되는 연관관계의 범위 및 연관관계 수명에 대한 한계가 있습니다.

트랜잭션 연관관계의 관련사항 및 수명을 이해하는 것은 동적 라우팅 환경에서 이를 관리할 방법을 결정할 때 중요합니다.

이 절에서는 다음 주제에 대해 다룹니다.

- 『연관관계 트랜잭션 그룹』
- 『연관관계 및 연관관계 수명』

연관관계 트랜잭션 그룹

동적 라우팅 환경에서 연관관계를 관리하려면 먼저 트랜잭션을 연관관계에 따라 분류해야 합니다. 이를 수행하는 한 가지 방법은 트랜잭션을 트랜잭션 그룹에 놓는 것인데, 그룹은 트랜잭션간 연관관계를 갖는 트랜잭션의 세트입니다.

각 연관관계 트랜잭션 그룹(또는 연관관계 그룹)은 서로 연관관계를 갖는 트랜잭션의 그룹을 나타냅니다. 연관관계 그룹 정의는 동적 또는 분산 라우팅 프로그램이 트랜잭션이 어떤 대상 리전으로 라우팅되어야 하는지 판별할 수 있는 하나의 방법입니다.

CICS 워크로드에서 더 많은 트랜잭션간 연관관계를 가질수록, 동적 라우팅 프로그램이 CICSplex 사이에서 워크로드의 균형을 잡는 것은 덜 효율적입니다. 트랜잭션간 연관관계의 영향을 최소화하기 위해, 연관관계 그룹의 연관관계가 연관관계 및 연관관계 수명으로 특징화됩니다. 이러한 관련사항 및 수명 속성은 연관관계의 범위와 지속 기간을 판별합니다. 그러므로 연관관계 트랜잭션 그룹은 그룹과 연관된 연관관계 및 연관관계 수명과 함께 연관관계 그룹 ID, 연관관계 그룹을 구성하는 트랜잭션 세트로 구성됩니다.

연관관계 및 연관관계 수명

트랜잭션 그룹을 작성할 때, **AFFINITY** 속성을 사용하여 그룹에 연관관계를 지정하고 **AFFLIFE** 속성을 사용하여 연관관계 수명을 지정할 수 있습니다. 연관관계는 동적 또는 분산 라우팅 프로그램이 연관관계와 연관된 트랜잭션 인스턴스에 대한 대상 리전을 선택하는 방법을 판별합니다. 연관관계 수명은 연관관계가 종료하는 시기를 판별합니다.

연관관계 그룹에 지정할 수 있는 5가지 가능한 연관관계는 BAPPL, Global, LOCKED, LUname 및 userid입니다. 연관관계에 지정하는 연관관계 수명은 연관관계에 따라 다릅니다. 예를 들어, 연관관계가 LOCKED인 경우, 연관관계 수명은 UOW여야 하는 반면, 연관관계가 BAPPL인 경우에는 연관관계 수명을 활동, 영구, 프로세스 또는 시스템으로 설정할 수 있습니다.

BAPPL 관계:

BAPPL과 트랜잭션 그룹 간의 연관관계를 정의할 때 동일한 BTS 프로세스와 연관된 모든 트랜잭션 인스턴스는 연관관계 수명 동안 동일한 대상 리전에서 실행되어야 합니다.

연관관계가 BAPPL이면 연관관계 수명은 다음 값 중 하나를 가지고 있어야 합니다.

프로세스

연관관계는 연관된 프로세스가 존재하는 한 지속됩니다.

작업 연관관계는 연관된 활동이 존재하는 한 지속됩니다.

시스템 연관관계는 대상 리전이 존재하는 한 지속되며, 대상 리전이 종료될 때마다(정상, 즉시 또는 비정상 종료) 종료됩니다.

영구 연관관계는 모든 CICS 재시작에서 확장합니다. CICSplex SM을 실행 중인 경우, 이 연관관계는 워크로드를 사용하여 CICSplex를 관리할 때 포함되는 CMAS가 활성 상태일 때 계속 지속합니다.

BAPPL 관계가 있는 일반적인 트랜잭션 예제는 로컬 임시 저장영역 큐가 BTS 활동이나 프로세스 내에 트랜잭션 간에 데이터를 전달하는 데 사용되는 곳입니다.

그림 103에서는 BAPPL 관계가 있는 연관관계 그룹의 예제를 보여줍니다.

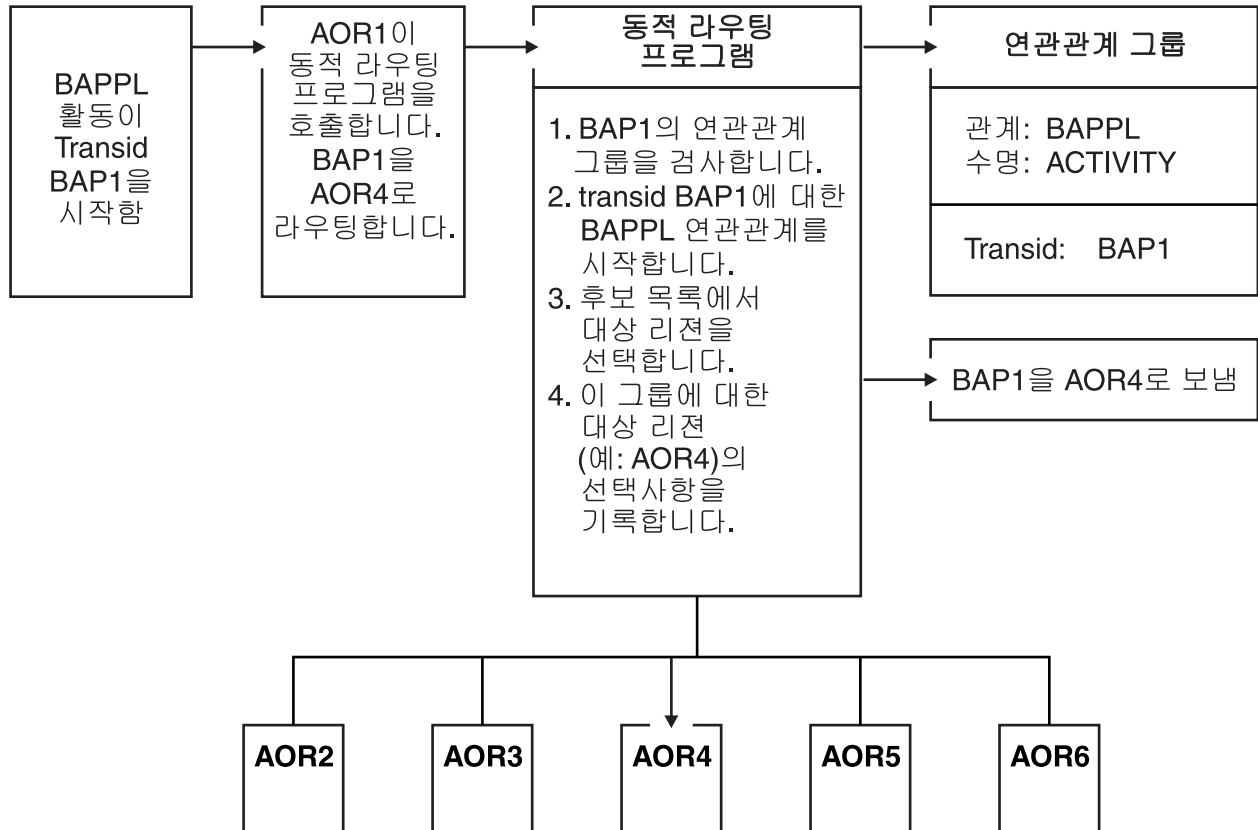


그림 103. BAPPL 관계 및 활동 수명이 있는 트랜잭션간 연관관계 관리

이 예제에서 BTS 트랜잭션 BAP1의 첫 번째 인스턴스는 BAPPL 활동 연관관계를 시작합니다. BAP1의 첫 번째 인스턴스는 적합한 대상 리전(AOR1 - AOR6)에 라우팅될 수 있지만 활동의 다른 모든 인스턴스는 BAP1에 대해 선택되는 대상 리전으로 라우팅되어야 합니다.

BTS 자체는 연관관계를 도입하지 않고 이를 도입하는 프로그래밍 기술을 권장하지 않지만 연관관계를 도입하는 기존 코드를 지원합니다. 워크로드 관리에 이러한 연관관계를 정의해야 합니다. 이는 특히 각 연관관계의 수명을 지정하기 위해 중요합니다. 이를 실패하면 워크로드 관리 라우팅 옵션을 불필요하게 제한할 수도 있습니다.

지정된 활동을 동시에 및 비동기적으로 실행할 수 있음을 유의해야 합니다. 워크로드 관리는 비동기적으로 작성되는 호출만을 인식할 수 있습니다. 게다가 이러한 연관관계는 BTS 세트에 걸쳐 동기화되므로 이러한 연관관계, 특히 활동 및 프로세스 연관관계를 작성하지 않는 것이 좋습니다. 이러한 연관관계는 시스템에 심각한 성능 영향을 미칠 수 있습니다.

CICSplex SM에서는 연관관계를 유지보수할 수 있는 가장 긴 시간은 워크로드에 관련된 CMAS가 활성인 동안입니다. 즉, PERMANENT의 연관관계 수명입니다. 전체 시스템 장애 또는 계획된 시스템 종료 발생하면 연관관계는 유실되지만 CICS에서는 활동은 BTS RLS 데이터 세트에서 복구됩니다.

글로벌 관련사항:

트랜잭션 그룹에 대해 글로벌의 연관관계를 정의할 때, START 명령 또는 CICS BTS 프로세스에 의해, 터미널에서 시작된 모든 트랜잭션의 모든 인스턴스는 연관관계의 수명에 대해 동일한 대상 리전에서 실행해야 합니다.

연관관계가 글로벌인 경우, 연관관계 수명은 다음 값 중 하나를 가지고 있어야 합니다.

시스템 연관관계는 대상 리전이 존재하는 한 지속되며, 대상 리전이 종료될 때마다(정상, 즉시 또는 비정상 종료) 종료됩니다.

영구 연관관계는 모든 CICS 재시작에서 확장합니다. 이는 모든 트랜잭션간 연관관계의 가장 큰 제한입니다. CICSplex SM을 실행 중인 경우, 이 연관관계는 워크로드를 사용하여 CICSplex를 관리할 때 포함되는 CMAS가 활성 상태일 때 계속 지속합니다.

영구 수명의 글로벌 트랜잭션간 연관관계 예제는 트랜잭션이 로컬, 복구 가능, 임시 저장영역 큐를 사용하는(읽거나 쓰는) 경우와 TS 큐 이름이 터미널에서 파생되지 않는 경우입니다. (큐가 로컬인 CICS 리전에서 TS 큐가 복구 가능함을 지정할 수 있습니다.)

일반적으로, 이 연관관계 범주의 트랜잭션은 동적 라우팅에 적합한 후보가 아니므로, 정적으로 라우팅되는 트랜잭션으로 만들 것을 고려해야 합니다.

413 페이지의 그림 104에서는 글로벌 관련사항의 예제를 보여줍니다.

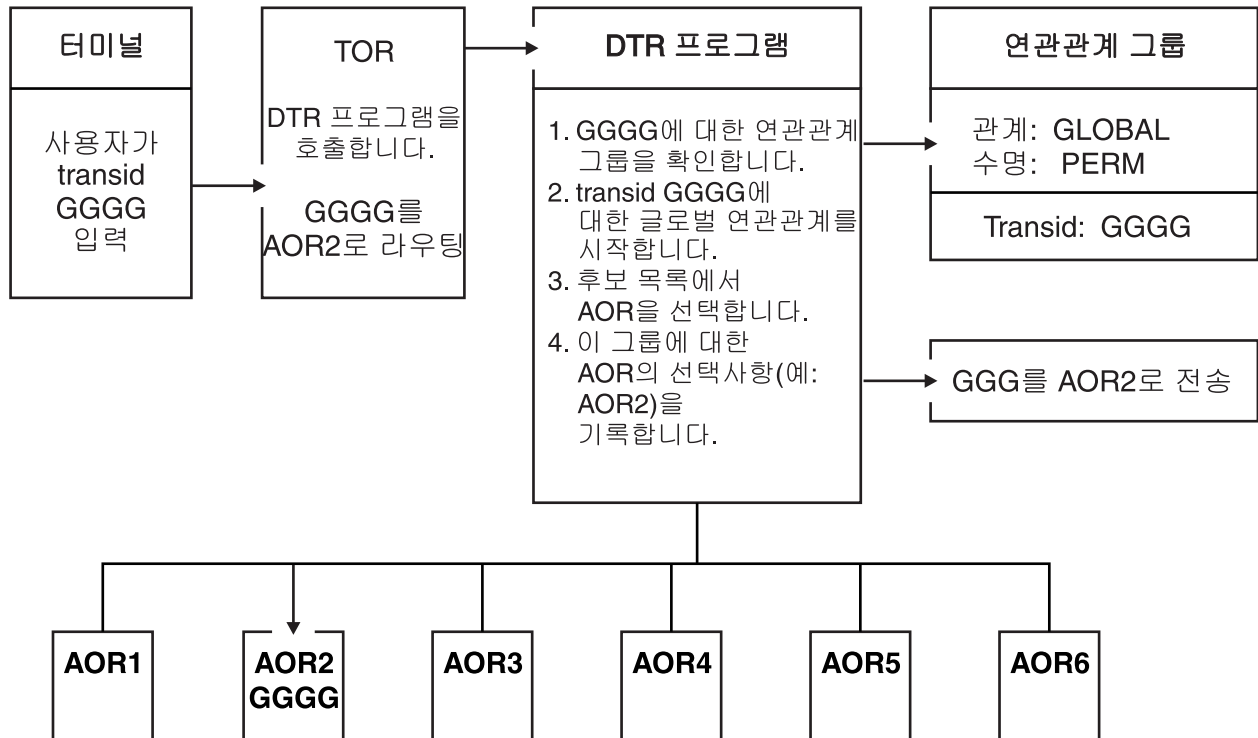


그림 104. 글로벌 관련사항 및 영구 수명을 가지고 있는 트랜잭션간 연관관계 관리

이 예제에서, 트랜잭션 GGGG는 영구 글로벌 연관관계를 가지고 있는 그룹에 정의됩니다. 터미널에서 transid GGGG의 첫 번째 인스턴스는 영구 수명 연관관계를 시작합니다. GGGG의 첫 번째 인스턴스는 적절한 대상 리전으로 라우팅될 수 있습니다. 이 예제에서, AOR2는 가능한 범위 AOR1 - AOR6에서 선택되지만, 임의 터미널의 다른 모든 인스턴스 역시 동일한 리전 AOR2로 라우팅되어야 합니다.

LOCKED 관계:

트랜잭션 그룹에 LOCKED 연관관계를 정의하는 경우, 작업 단위가 같은 동적으로 링크된 프로그램과 연관된 그룹의 모든 트랜잭션 인스턴스는 작업 단위 수명 동안 동일한 대상 리전에서 실행되어야 합니다.

LOCKED 관계는 작업 단위(UOW)와 연관됩니다. LOCKED 관계의 연관관계 수명은 항상 UOW입니다. 작업 단위는 CICS SYNCPOINT 또는 ROLLBACK 요청이 실행될 때 또는 원래 태스크가 종료될 때 종료됩니다. UOW는 동적 프로그램 링크 요청 간의 LOCKED 연관관계에 대해서만 유효합니다.

LOCKED 관계를 가진 일반적인 트랜잭션 예제는 동적으로 라우팅된 동일한 프로그램 호출이 공통 자원에 액세스하는 경우입니다. 프로그램의 각 호출마다 호출된 프로그램이 대상 시스템에서 실행되는 트랜잭션 코드가 있습니다. 해당 트랜잭션 코드를 LOCKED 연관관계를 지정하는 트랜잭션 그룹의 일부로 지정하면, 해당 프로그램 이름에 대한 모든 동적 프로그램 링크 인스턴스는 연관된 원격 트랜잭션이 호출자의 작업

단위가 종료될 때까지 동일한 대상 리전으로 재라우팅되도록 합니다. 해당 후속 프로그램 링크가 다른 리전으로 라우팅된 경우 교착상태가 발생할 수 있습니다. 따라서 자원을 잠근 리전과 다른 곳으로 작업을 라우팅하지 않아야 합니다. 추가 정보는 *CICS 상 호 통신 안내서*의 동일한 서버 리전으로의 다중 링크를 참조하십시오.

그림 105는 LOCKED 관계를 가진 연관관계 예제를 보여줍니다.

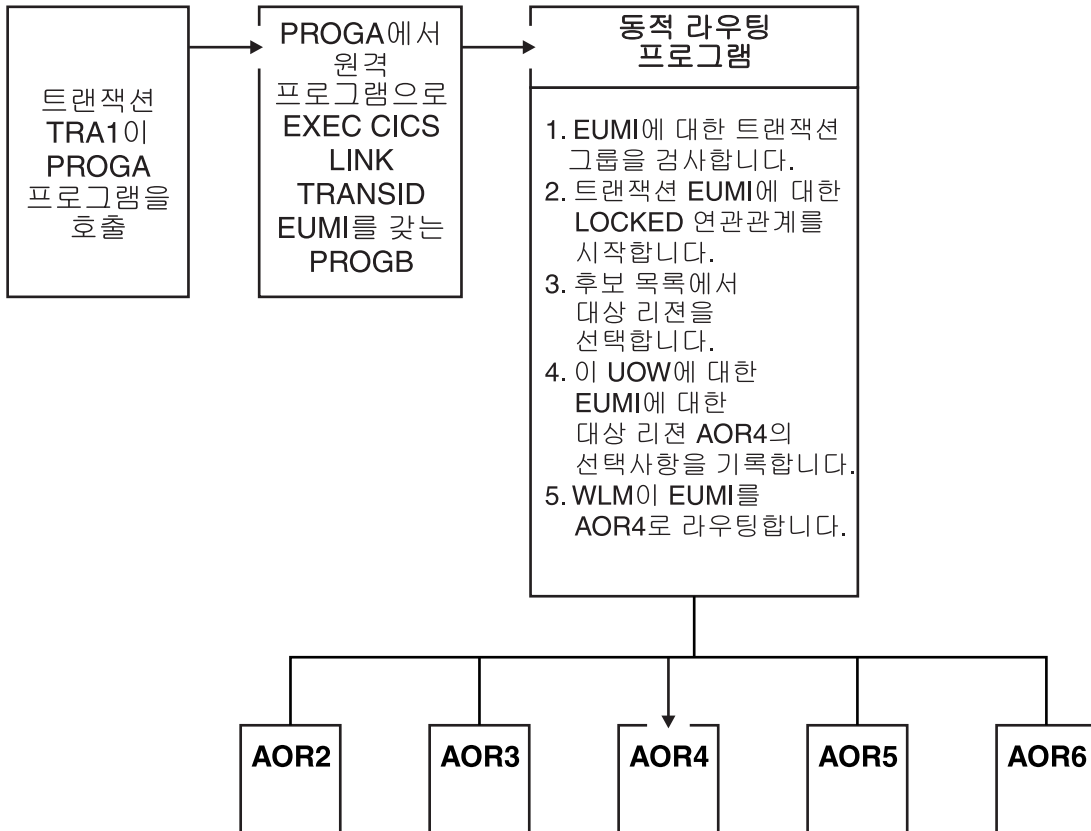


그림 105. LOCKED 관계 및 UOW 수명을 가진 트랜잭션간 연관관계 관리

이 예제에서 트랜잭션 TRA1은 WLMATAFF 자원 테이블의 **NETUOWID** 속성이 식별한 작업 단위의 일부로 PROGA 프로그램을 호출합니다. PROGA는 원격 시스템에서 PROGB 프로그램에 **EXEC CICS LINK**를 요청합니다. PROGB는 프로그램 정의에 **TRANSID** 속성이 정의한 동적 및 연관된 트랜잭션(EUMI)으로 정의됩니다. EUMI는 연관관계가 LOCKED이고 연관관계 수명이 UOW인 정의된 트랜잭션 그룹에 속한 사용자 정의 미러 트랜잭션입니다. 워크로드 관리는 이 트랜잭션을 CICS 리전 AOR4로 라우팅합니다. 연관관계가 이미 존재하는지 여부를 확인한 후 동적 라우팅 프로그램이 EUMI에 대해 새 연관관계를 시작합니다. 모든 후속 EUMI 인스턴스는 작업 단위 수명 동안 이 연관관계를 공유하며 작업 단위가 종료될 때까지 AOR4로 라우팅됩니다.

LUnicode(터미널) 관련사항:

트랜잭션 그룹에 대해 LU 이름의 연관관계를 정의할 때, 동일한 터미널과 연관되는 그룹의 모든 트랜잭션의 모든 인스턴스는 연관관계 수명에 대해 동일한 대상 리전에서 실행해야 합니다.

연관관계가 LU 이름인 경우, 연관관계 수명은 다음 값 중 하나를 가지고 있어야 합니다.

의사 대화

연관관계는 전체 의사 대화 동안 지속되고 터미널에서 의사 대화가 종료될 때 종료합니다. 각 트랜잭션은 END의 의사 대화 모드가 아니라 EXEC CICS RETURN TRANSID로 종료해야 합니다.

로그온 연관관계는 터미널이 CICS에 로그인한 상태에 있는 동안 지속되며 터미널이 로그오프할 때 종료됩니다.

시스템 연관관계는 대상 리전이 존재하는 한 지속되며, 대상 리전이 종료될 때마다(정상, 즉시 또는 비정상 종료) 종료됩니다.

영구 연관관계는 모든 CICS 재시작에서 확장합니다. CICSplex SM을 실행 중인 경우, 이 연관관계는 워크로드를 사용하여 CICSplex를 관리할 때 포함되는 CMAS가 활성 상태일 때 계속 지속합니다.

구분 연관관계는 END 의사 대화 모드의 트랜잭션이 발견될 때까지 계속됩니다.

LU 이름 관련사항을 가지고 있는 트랜잭션의 일반적 예제는 다음과 같습니다.

- 의사 대화에서 트랜잭션 사이에 데이터를 전달하기 위해 로컬 TS 큐를 사용합니다. 그리고,
- 터미널 이름에서 부분적으로 파생되는 TS 큐 이름을 사용합니다. (TS 큐 이름에 대한 정보는 397 페이지의 『임시 저장영역 사용 시 연관관계 피하기』의 내용을 참조하십시오.)

이러한 유형의 트랜잭션은 의사 대화의 수명 및 터미널의 관련사항을 가지고 있는 연관관계에 놓을 수 있습니다. 동적 라우팅 프로그램이 특정 터미널(LU 이름)에 의해 초기화된 의사 대화에서 첫 번째 트랜잭션을 발견할 때, 이 트랜잭션의 올바른 후보인 대상 리전으로 트랜잭션을 라우팅하여 가능하도록 해제합니다. 그러나 동일 터미널에서 초기화되는 연관관계 그룹의 후속 트랜잭션은 의사 대화를 시작한 트랜잭션과 동일한 대상 리전으로 라우팅해야 합니다. 연관관계가 종료될 때(지정된 터미널에서 의사 대화 종료 시), 동적 라우팅 프로그램은 다시 첫 번째 트랜잭션을 후보 대상 리전으로 라우팅하여 가능하도록 해제합니다.

이러한 연관관계 양식은 관리 가능하므로, 동적 트랜잭션 라우팅에서 너무 심한 제한조건을 부과하지 않고, 많은 CICSplexes에서 발생할 수 있습니다. 동적 라우팅 프로그램에 의해 쉽게 관리할 수 있으므로, 동적 라우팅의 사용을 금지하면 안됩니다.

그림 106에서는 이러한 유형의 예제를 보여줍니다.

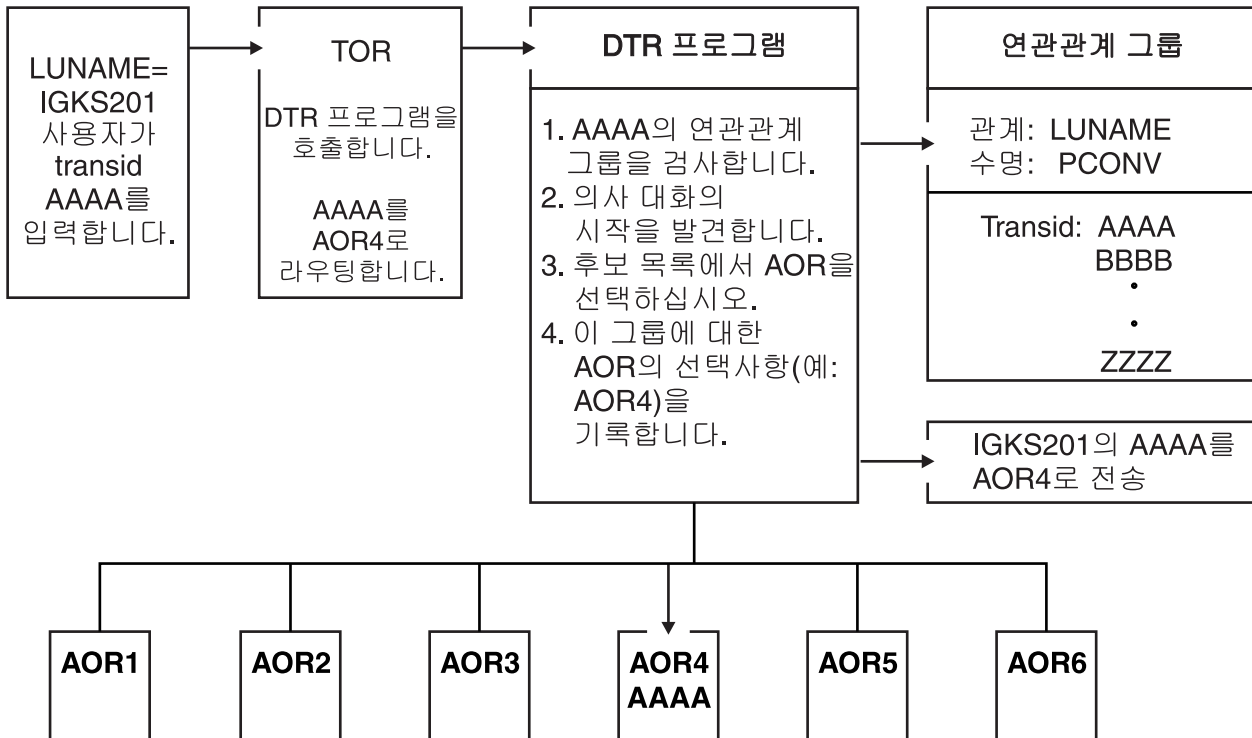


그림 106. LU 이름 관련사항 및 의사 대화 수명을 가지고 있는 트랜잭션간 연관관계 관리

이 예제에서, 터미널에서 트랜잭션 ID AAAA의 인스턴스가 의사 대화 수명 연관관계를 시작합니다. AAAA는 적절한 어떤 대상 리전(AOR1 - AOR6)에도 라우팅될 수 있지만, 동일한 터미널(이 예제에서 IGKS201)의 동일한 의사 대화에서, 그룹에 있는 다른 트랜잭션은 AAAA에 대해 선택되는 대상 리전으로 라우팅해야 합니다.

사용자 ID 관련사항:

트랜잭션 그룹에 대해 사용자 ID의 연관관계를 정의할 때, 터미널에서, **START** 명령으로, 또는 CICS BTS 활동으로 초기화되고, 동일한 사용자 ID 대신 실행된 모든 트랜잭션의 모든 인스턴스는 연관관계 수명에 대해 동일한 대상 리전에서 실행해야 합니다.

연관관계가 사용자 ID인 경우, 연관관계 수명은 다음 값 중 하나를 갖습니다.

의사 대화

연관관계는 전체 의사 대화 동안 지속되고 해당 사용자 ID에 대해 의사 대화가 종료될 때 종료합니다. 각 트랜잭션은 END의 의사 대화 모드가 아니라 **EXEC CICS RETURN TRANSID**로 종료해야 합니다.

사인온 연관관계는 사용자가 사인온한 상태에 있는 동안 지속되며 사용자가 사인오프할 때 종료됩니다. 이 수명은 사용자 ID마다 한 명의 사용자만 허용되는 상황

에서만 가능합니다. 여러 사용자가 (다른 터미널에서) 동시에 동일한 사용자 ID로 사인온하도록 허용되는 경우 사인온 수명을 발견할 수 없습니다.

시스템 연관관계는 대상 리전이 존재하는 한 지속되며, 대상 리전이 종료될 때마다(정상, 즉시 또는 비정상 종료) 종료됩니다.

영구 연관관계는 모든 CICS 재시작에서 확장합니다. CICSplex SM을 실행 중인 경우, 이 연관관계는 워크로드를 사용하여 CICSplex를 관리할 때 포함되는 CMAS가 활성 상태일 때 계속 지속합니다.

구분 연관관계는 END 의사 대화 모드의 트랜잭션이 발견될 때까지 계속됩니다.

사용자 ID 관련사항을 가지고 있는 트랜잭션의 일반적 예제는 TS 큐와 같은 자원 식별을 위해 동적으로 사용자 ID가 사용되는 것입니다. 이 범주에서 연관관계의 최소 검색은 사용자가 사인온 상태로 유지되는 동안만 지속되는 검색입니다.

그림 107에서는 사용자 ID 관련사항과 사인온 수명이 있는 연관관계 그룹의 예제를 보여줍니다.

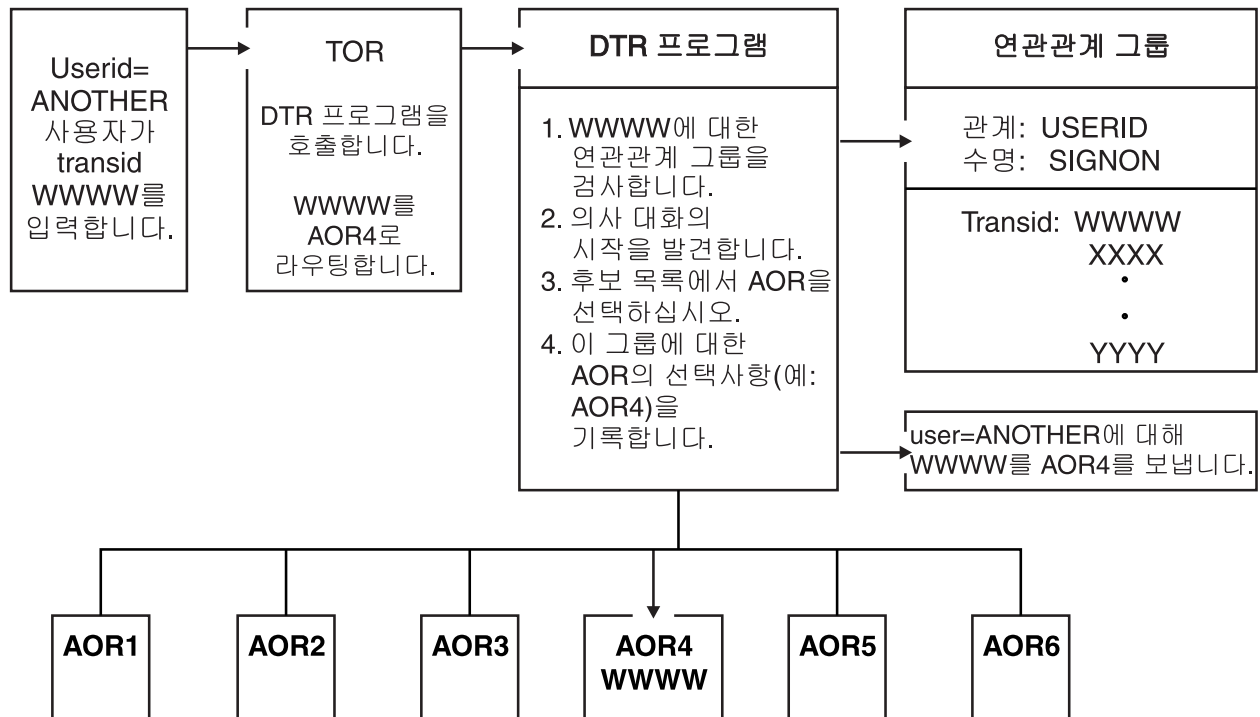


그림 107. 사용자 ID 관련사항 및 사인온 수명을 가지고 있는 트랜잭션간 연관관계 관리

이 예제에서, 터미널의 트랜잭션 인스턴스가 사인온 수명 연관관계를 시작합니다. 적절한 어떤 대상 리전(AOR1 - AOR6)에도 라우팅될 수 있지만, 동일한 사용자(이 예제에서 ANOTHER)에 대해 그룹에 있는 다른 트랜잭션은 그룹에서 트랜잭션의 첫 번째 인스턴스로 선택되는 대상 리전으로 라우팅해야 합니다.

복구 디자인

CICS는 CICS 실행 중에 이벤트 또는 데이터 변경사항을 복구하거나 재구성하는 데 도움이 되는 두 가지 기술(저널링과 동기점 지정)을 제공합니다.

이름 지정된 카운터 복구 기술은 871 페이지의 『이름 지정된 카운터 복구』에 설명되어 있습니다.

저널링

CICS는 CICS 처리 중에 저널 작성 및 관리를 위한 기능을 제공합니다. 저널은 사용자가 이벤트 또는 데이터 변경사항 후속 재구성을 용이하게 하는 데 필요한 데이터를 포함할 수 있습니다.

예를 들어, 저널은 감사 추적, 데이터베이스 갱신 및 추가 변경 파일 또는 시스템을 통과하는 트랜잭션 레코드(종종 로그라고 함)의 역할을 할 수 있습니다. 임의 태스크에서 각 저널을 작성할 수 있습니다.

애플리케이션 프로그래머가 다음을 수행할 수 있도록 저널 제어 명령이 제공됩니다.

- 저널 레코드 작성(WRITE JOURNALNAME 또는 WRITE JOURNALNUM 명령)
- 저널 출력과 동기화(완료 대기)(WAIT JOURNALNAME 또는 WAIT JOURNALNUM 명령)

저널 제어 명령 실행 중 발생하는 예외 조건은 423 페이지의 『예외 조건 처리』에 설명되어 있습니다(이전 JFILEID 옵션은 호환성 목적으로만 지원됨).

저널 레코드

각 저널은 저널 ID로 알려진 이름 또는 숫자로 식별됩니다. 이 수는 1부터 99에 이를 수 있습니다. 이름 DFHLOG는 시스템 로그로 알려진 저널에 예약되어 있습니다.

저널 레코드가 빌드될 때, 데이터는 저널 버퍼 영역으로 이동됩니다. 저널 운영에 필요한 모든 버퍼 공간 및 기타 작업 영역은 CICS에 의해 획득되고 관리됩니다. 사용자 태스크는 저널에 작성될 데이터만을 제공합니다. 로그 관리자는 출력 서비스를 요청하는 애플리케이션 프로그래머가 자세한 레이아웃과 정확한 저널 레코드 내용을 걱정할 필요가 없도록 설계되어 있습니다. 프로그래머는 어떤 저널을 사용할지, 어떤 사용자 데이터를 지정할지 및 어떤 사용자 ID를 제공할지에 대해서만 알면 됩니다.

저널 출력 동기화

동기 저널 레코드는 WAIT 옵션과 함께 WRITE JOURNALNAME 또는 WRITE JOURNALNUM 명령을 실행하여 작성되며, 요청 태스크는 출력이 완료될 때까지 기다릴 수 있습니다.

이 명령을 지정하여 애플리케이션 프로그래머는 처리를 계속하기 전에 저널과 연관된 외부 저장영역에 저널 레코드가 기록되게 합니다. 즉, 태스크가 출력 작업과 동기화됩니다.

애플리케이션 프로그래머는 비동기 저널 출력도 요청할 수 있습니다. 따라서 저널 레코드는 저널 버퍼에 작성되지만 요청 태스크가 제어를 보유할 수 있으므로 다른 처리를 계속할 수 있습니다. 태스크는 WAIT JOURNALNAME 또는 WAIT JOURNALNUM 명령을 실행하여 나중에 저널 완료(즉, 동기화)를 확인하고 기다릴 수 있습니다.

참고: 일부 경우, SHUTDOWN IMMEDIATE는 사용자 저널 레코드가 로그 관리자 버퍼가 아닌 외부 저장영역에 기록된 경우 저널 레코드 유실을 초래할 수 있습니다. 정상 종료가 정지되어 CICS 종료 지원 트랜잭션(CESD)이 정상 종료 중에 SHUTDOWN IMMEDIATE를 강제하는 경우에도 마찬가지입니다. 저널 레코드 유실 위험을 방지하려면 CICS WAIT JOURNALNUM 요청을 정기적으로, 프로그램을 종료하기 전에 실행하는 것이 좋습니다.

WAIT가 없으면 CICS는 데이터 버퍼가 가득 찰 때까지 또는 일부 다른 관련되지 않은 활동이 버퍼 강화를 요청할 때까지 로그 스트림에 데이터를 기록하지 않으므로 I/O 조작 수가 줄어듭니다. WAIT를 사용하면 CICS가 로그 구조 버퍼 크기를 정확하게 계산하기 더 어렵습니다. CF 로그 스트림의 경우, 이로 인해 결합 기능에서 저장영역을 비효율적으로 사용하게 됩니다.

저널 레코드를 주어진 저널의 CICS 버퍼 공간에 빌드하는 기본 프로세스는 다음 이벤트 중 하나가 발생할 때까지 계속됩니다.

- 시스템 로그의 경우:
 - 시스템이 무결성을 보장하고 향후 긴급 재시작을 허용하기 위해 필요로 할 때마다
 - 로그 스트림 버퍼가 채워짐
- 사용자 저널의 경우:
 - 로그 스트림 버퍼가 채워짐(또는 저널 버퍼가 채워질 때 저널이 SMF에 상주하는 경우)
 - 저널 레코드 출력에 대해 (임의 태스크에서) WAIT 옵션을 지정하는 요청이 수행됨
 - EXEC CICS SET JOURNALNAME 명령이 실행됨
 - EXEC CICS DISCARD JOURNALNAME 명령이 실행됨
 - 동일한 로그 스트림에 맵핑되는 다른 저널에 대해 발생하는 이전에 나열된 이벤트
 - 정상 종료 시
- 정방향 복구 로그의 경우:

- 로그 스트림 버퍼가 채워짐
- 동기점에서(첫 번째 단계)
- 파일 종료 시
- 자동 저널의 경우:
 - 로그 스트림 버퍼가 채워짐
 - 저널 레코드 출력에 대해 (임의 태스크에서) WAIT 옵션을 지정하는 요청이 수행됨
 - 파일 종료 시
- 로그의 로그(DFHLOG)의 경우:
 - 파일 OPEN 및 CLOSE 요청 시

이 중 하나가 발생하면, 비동기 요청으로 인해 연기된 출력을 포함하여 버퍼에 있는 모든 저널 레코드가 하나의 블록으로 로그 스트림에 기록됩니다.

저널 출력을 연기하여 얻을 수 있는 장점은 다음과 같습니다.

- 대기 줄을 줄여 트랜잭션 응답 시간이 향상됩니다.
- 호스트 시스템에서 물리적 I/O 요청 로드가 줄어듭니다.
- 로그 스트림에 더 적긴 하지만 더 큰 블록이 포함되므로 기본 저장영역을 보다 잘 활용할 수 있습니다.

그러나 프로그래밍이 더 복잡해져야만 이러한 장점을 얻을 수 있습니다. 저널 출력과의 동기화를 제어하도록 계획하고 프로그래밍하는 것이 필요합니다. 저널 레코드의 데이터 콘텐츠와 사용 방법에 따라 달라지는 추가 결정을 애플리케이션에서 내려야 합니다. 어떤 경우라도 저널 로드가 많은 경우에만 저널 출력 연기로 인한 모든 이점을 얻을 수 있습니다.

요청 시 사용 가능한 저널 버퍼 공간이 저널 레코드를 포함하기에 부족하면 NOJBUFSP 조건이 발생합니다. 이 조건에 대해 HANDLE CONDITION 명령이 활성화되어 있지 않으면 요청 태스크가 제어를 유실하고 현재 버퍼의 콘텐츠가 기록되며, 요청 태스크로 제어가 리턴되기 전에 비워진 결과 버퍼 공간에 저널 레코드가 빌드됩니다.

요청 태스크가 제어를 유실하지 않으려면(예를 들어, 다른 태스크가 제어를 가져오기 전에 일부 하우스키핑을 수행해야 하는 경우) HANDLE CONDITION 명령을 실행해야 합니다. NOJBUFSP 조건이 발생하면 요청에 대해 저널 레코드가 빌드되지 않으며, HANDLE CONDITION 명령에 제공된 위치에서 요청 프로그램에 제어가 직접 리턴됩니다. 요청 프로그램은 저널 출력 요청을 다시 실행하기 전에 필요한 하우스키핑을 수행할 수 있습니다.

저널 명령으로 인해 저널에 즉시 또는 연기된 출력이 발생할 수 있습니다. 일부 시스템 로그는 요청에 JOURNALNAME(DFHLOG)를 지정하여 다른 모든 레코드와 구별됨

니다. 사용자 저널 레코드는 일부 다른 JOURNALNAME 또는 JOURNALNUM을 사용하여 작성됩니다. 모든 레코드는 저널 유형 ID(JTYPEID)를 포함해야 합니다. 사용자 저널링이 시스템 로그에 수행될 경우, 최상위 비트 설정에 따라 저널 유형 ID는 웹 또는 긴급 재시작 시 글로벌 사용자 엑시트 XRCINPT에 이에 대한 표시를 제어하는 역할도 합니다. 레코드는 다음과 같이 로그의 역방향 스캔 중에 제공됩니다.

- 인플라이트 또는 인다우트(indoubt) 태스크에 대해서만(최상위 비트 해제)
- 스캔이 종료될 때까지 발생한 모든 레코드에 대해(최상위 비트 설정)

저널 레코드에 대한 자세한 정보는 저널 레코드를 참조하십시오. 복구 프로세스에 대한 자세한 정보는 *CICS* 복구 및 다시 시작 안내서의 *CICS* 긴급 재시작을 참조하십시오.

동기점 지정

CICS 태스크 비정상 종료 또는 *CICS* 시스템 장애가 발생할 경우 복구를 용이하게 하기 위해 시스템 프로그래머는 *CICS* 테이블 생성 중에 특정 자원(예: 파일)을 복구 기능으로 정의할 수 있습니다. 태스크가 비정상적으로 종료되면 이러한 자원이 태스크 시작 시에 있던 조건으로 복원되며 재실행할 수 있습니다.

태스크와 연관된 자원을 복원하는 프로세스를 백아웃이라고 합니다.

개별 태스크가 실패하면 DTB(Dynamic Transaction Backout) 프로그램이 백아웃을 수행합니다. *CICS* 시스템에 장애가 발생하면 긴급 재시작 프로세스의 일부로 백아웃이 수행됩니다. 애플리케이션 코딩에는 영향을 주지 않는 이러한 기능을 설명하는 *CICS* 복구 및 다시 시작 안내서의 *START=INITIA* 매개변수로 *CICS* 시작을 참조하십시오.

그러나 장기 실행 프로그램의 경우 오랜 기간 동안 누적되어 태스크 또는 시스템 장애가 발생할 경우 백아웃 기능성에 노출된 많은 변경사항을 갖는 것은 바람직하지 않습니다. SYNCPOINT 명령을 사용하여 프로그램을 작업 단위(UOW)로 알려진 논리적으로 별개의 섹션으로 분할하면 이러한 가능성을 방지할 수 있습니다. UOW의 끝을 동기화 점(동기점)이라고 합니다. 동기점에 대한 자세한 정보는 *CICS* 복구 및 다시 시작 안내서의 복구 및 재시작 기능을 참조하십시오.

장애가 동기점 이후에 발생하지만 태스크가 완료되기 이전에 발생하면 동기점 이후에 수행된 변경사항만 백아웃됩니다.

또는 SYNCPOINT 명령 대신 SAA 자원 복구 인터페이스를 사용할 수 있습니다. 이는 기존 *CICS* 자원 복구 서비스에 대한 대체 API를 제공합니다. 공통 API의 일관성이 도움이 된다고 보여지는 다중 SAA 플랫폼을 포함하는 네트워크에서 SAA 자원 복구 인터페이스를 사용할 수 있습니다. *CICS* 시스템에서 SAA 자원 복구 인터페이스는 EXEC CICS API와 동일한 기능을 제공합니다.

제한사항: 전체 SAA 자원 복구는 CICS 구현에서 지원되지 않는 일부 리턴 코드를 제 공합니다(자원 복구를 위한 SAA 공통 프로그래밍 인터페이스 참조서 매뉴얼의 CICS 부록 참조).

SAA 자원 복구 인터페이스는 다음 두 가지 호출 유형을 갖는 호출 인터페이스로 구현 됩니다.

SRRCMIT

확약(SYNCPOINT 명령과 동일)

SRRBACK

백아웃(SYNCPOINT ROLLBACK 명령과 동일)

SAA 자원 복구 인터페이스에 대한 추가 정보는 자원 복구를 위한 SAA 공통 프로그 램밍 인터페이스 참조서 매뉴얼을 참조하십시오.

UOW는 보호 자원과 관련해서는 물론 실행 플로우와 관련해서도 논리적으로 완전히 독립되어야 합니다. 일반적으로 UOW는 SEND 및 RECEIVE 명령을 통해 바운드된 전체 대화식 작업으로 구성됩니다. 찾아보기는 UOW의 한 예입니다. 따라서 ENDBR 명령이 동기점 앞에 와야 합니다.

DL/I 종료 호출을 동기점으로 간주하는 것 이외에 SYNCPOINT 명령을 실행하면 CICS가 DL/I 종료 호출을 실행합니다. 후속 UOW에서 DL/I PSB가 필요하면 프로그램 제어 블록(PCB) 호출 또는 SCHEDULE 명령을 사용하여 다시 스케줄해야 합니다.

분산 프로그램 링크(DPL)를 사용하면 동기점이 서버 프로그램에서 사용되도록 지정하고 클라이언트로 제어를 리턴하기 전에 서버 자원을 확약할 수 있습니다. LINK 명령에서 SYNCONRETURN 옵션을 사용하여 이를 수행합니다. SYNCONRETURN 옵션에 대한 프로그래밍 정보는 458 페이지의 『DPL(Distributed Program Link)의 예제』의 "서버 프로그램용 SYNCONRETURN 옵션"과 CICS 애플리케이션 프로그래밍 참조서를 참조하십시오.

SYNCPOINT 명령이 처리될 때 시작되지만 완료되지 않은 BMS 논리 메시지는 내재된 SEND PAGE 명령을 통해 완료되도록 강제 실행됩니다. 그러나 첫 번째 페이지가 완료되지 않은 논리 메시지는 유실되므로 여기에 의존해서는 안됩니다. 또는 SYNCPOINT 명령 전에 또는 트랜잭션 종료 전에 명시적 SEND PAGE 명령을 코드화해야 합니다.

트랜잭션을 재시작할 수 있는 트랜잭션에서 동기점이 실행될 경우 시스템 프로그래머에게 문의하십시오.

예외 조건 처리

애플리케이션 중 하나에서 EXEC CICS 명령을 처리할 때마다 CICS에서 자동으로 조건 또는 리턴 코드가 발생하여 사용자에게 발생한 상황에 대해 알립니다.

CICS EXEC 인터페이스 프로그램이 이 조건을 애플리케이션으로 다시 전달하도록 선택할 수 있습니다. 이 조건을 RESP 값이라고도 합니다. 이 조건은 명령에서 RESP 옵션을 사용하여 얻을 수 있기 때문입니다. 또는 실행 인터페이스 블록(EIB)에서 읽어 이 값을 얻을 수 있습니다. 이 조건은 일반적으로 NORMAL입니다.

비정상적인 상황이 발생하면 NORMAL 외의 조건인 예외 조건이 표시됩니다. 이 조건을 테스트하여 발생한 상황과 가능한 경우 이유도 확인할 수 있습니다.

여러 예외 조건에는 그에 연관된 추가 값(RESP2)이 있으며 이 값은 추가 정보를 제공합니다. 이 RESP2 값을 얻기 위해서 명령에서 RESP2 옵션(RESP 옵션에 추가하여)을 사용하거나 EIB에서 그 값을 읽을 수 있습니다.

NORMAL이 아니지만 일부 조건은 오류 상황을 표시하지 않습니다. 예를 들어, 파일을 찾아보는 중 READNEXT 명령의 ENDFILE 조건은 예상한 작동을 표시할 수 있습니다. 모든 가능한 조건 및 그러한 조건이 발생할 수 있는 명령에 대한 정보는 애플리케이션 개발 참조서의 내용을 참조하십시오.

이 섹션에서는 다음에 대해 설명합니다.

- 『기본 CICS 예외 처리』
- 424 페이지의 『인라인 코드로 예외 조건 처리』
- 428 페이지의 『기본 CICS 예외 처리 수정』

기본 CICS 예외 처리

COBOL, PL/I 및 어셈블러 언어 애플리케이션(그러나 AMODE(64) 어셈블러 언어 애플리케이션은 해당되지 않음)의 경우 별도로 지정하지 않는 한 CICS는 예외 조건이 발생할 때마다 내장된 예외 처리를 사용합니다. AMODE(64) 어셈블러 언어 애플리케이션과 C 또는 C++로 작성된 애플리케이션의 경우 예외 조건이 발생하면 CICS가 조치를 취하지 않습니다. 애플리케이션은 예외 조건을 처리해야 합니다.

예외 조건 처리에 대한 정보는 424 페이지의 『인라인 코드로 예외 조건 처리』의 내용을 참조하십시오.

CICS 기본 예외 처리에서 예외 조건이 발생하면 가장 자주 사용되는 조치는 이상 종료를 발생시키는 것입니다. 각 조건 및 각 명령의 특정 작동에 대한 세부사항은 애플리케이션 개발 참조서 및 시스템 프로그램 개발의 시스템 프로그래밍 명령 소개의 내용을 참조하십시오.

CICS 기본 예외 처리가 요구사항을 충족시키지 못하는 경우 다음과 같은 방식으로 다른 조치를 지정할 수 있습니다.

- NOHANDLE 옵션을 지정하여 특정 EXEC CICS 명령 호출에서 CICS 기본 예외 처리를 끕니다.
- 명령에서 RESP 옵션을 지정하여 CICS 기본 예외 처리를 끕니다. 이 옵션은 NOHANDLE 명령과 같은 방식으로 기본 CICS 예외 처리를 끕니다. 또한 명령이 리턴하는 조건의 값을 사용하여 RESP 옵션의 인수에 의해 이름 지정되는 변수를 갱신합니다. 세부사항은 『인라인 코드로 예외 조건 처리』의 내용을 참조하십시오.

기본 CICS 예외 처리를 끄는 경우 프로그램이 명령 호출에서 발생할 수 있는 모든 상황을 대처할 수 있도록 해야 합니다.

HANDLE ABEND, HANDLE CONDITION 및 IGNORE CONDITION 명령의 조합을 사용하여 기본 CICS 예외 처리를 수정할 수 있지만 이 방법은 더 이상 권장되지 않습니다. 세부사항은 428 페이지의 『기본 CICS 예외 처리 수정』의 내용을 참조하십시오.

인라인 코드로 예외 조건 처리

프로그램이 C 또는 C++로 작성되었거나 AMODE(64) 어셈블러 언어 애플리케이션인 경우 인라인 코드가 예외 조건을 처리하는 데 사용 가능한 유일한 기술입니다. 프로그램이 C 또는 C++로 작성되지 않고 AMODE(64) 어셈블러 언어 애플리케이션이 아닌 경우 예외 조건 처리에는 CICS가 기본 예외 처리를 수행하는 것을 막는 NOHANDLE 옵션 사용 또는 EXEC CICS 명령에서 RESP 옵션 지정이 포함됩니다.

이 태스크 정보

RESP 옵션을 사용하면 프로그램이 정정 조치를 취할 수 있도록 예외 조건의 값을 프로그램에 사용 가능하게 할 수 있습니다.

NOHANDLE 또는 RESP 옵션을 사용하는 경우 프로그램이 명령 실행 과정에서 발생하는 모든 조건에 대처할 수 있는지 확인하십시오. RESP 값은 프로그램이 수행할 작업을 결정할 수 있게 하는 데 사용 가능하고 사용해야 될 수도 있는 추가 정보는 실행 인터페이스 블록(EIB)에 포함됩니다. 특히 RESP2 값은 EIB 필드 중 하나에 포함됩니다. EIB에 대한 자세한 정보는 참조 -> 애플리케이션 개발에서 EIB 필드의 내용을 참조하십시오. 또는 프로그램이 명령에서 RESP2를 지정하는 경우 RESP2 값은 CICS에 의해 직접 리턴됩니다.

DFHRESP 내장 변환기 함수를 사용하면 RESP 값을 기호로 검사할 수 있기 때문에 RESP 값을 쉽게 테스트할 수 있습니다. 이 기술은 코드를 읽는 사람에게 의미 전달이 잘 안되는 2진 값을 검사하는 것보다 쉽습니다.

RESP 및 RESP2 옵션을 사용하는 방법

RESP의 인수는 사용자 정의 전자 2진 데이터 영역(긴 정수)입니다. 명령에서 리턴 시 이 인수는 발생했을 수도 있는 조건에 해당하는 값을 포함합니다. 일반적으로 그 값은 DFHRESP(NORMAL)입니다.

COBOL 및 PL/I에서 RESP 및 DFHRESP의 사용

다음 예제는 COBOL에서 RESP 옵션을 사용하는 EXEC CICS 호출을 보여줍니다. PL/I 예제도 비슷하지만 END-EXEC 대신 세미콜론(;)으로 종료됩니다.

```
EXEC CICS WRITEQ TS FROM(abc)
      QUEUE(qname)
      NOSUSPEND
      RESP(xxx)
      END-EXEC.
```

다음 코드는 RESP 값을 테스트하는 데 DFHRESP를 사용하는 예제입니다.

```
IF xxx=DFHRESP(NOSPACE) THEN ...
```

C 및 C++에서 RESP 및 DFHRESP 사용

다음 예제는 RESP 변수의 선언을 포함하여 RESP 옵션을 사용하는 C에서의 EXEC CICS 호출을 보여줍니다.

```
long response;
...
EXEC CICS WRITEQ TS FROM(abc)
      QUEUE(qname)
      NOSUSPEND
      RESP(response);
```

다음 코드는 RESP 값을 테스트하는 데 DFHRESP를 사용하는 예제입니다.

```
if (response == DFHRESP(NOSPACE))
{
...
}
```

어셈블러 언어에서 DFHRESP의 사용

다음 코드는 어셈블러 언어에서 RESP 값에 대한 테스트의 예제입니다.

```
CLC    xxx,DFHRESP(NOSPACE)
BE     ...
```

C에서 예외 처리의 예제

다음 예제는 BMS 맵을 수신하고 예외 조건에 대처하는 데 사용할 수 있는 일반적인 함수입니다.

```

int ReadAccountMap(char *mapname, void *map)
{
    long    response;
    int      ExitKey;
    EXEC CICS RECEIVE MAP(mapname)
           MAPSET("ACCOUNT")
           INTO(map)
           RESP(response);
    switch (response)
    {
    case DFHRESP(NORMAL):
        ExitKey = dfheiptr->eibaid;
        ModifyMap(map);
        break;
    case DFHRESP(MAPFAIL):
        ExitKey = dfheiptr->eibaid;
        break;
    default:
        ExitKey = DFHCLEAR;
        break;
    }
    return ExitKey;
}

```

그림 108. C에서 예외 처리의 예제

ReadAccountMap 함수에는 두 개의 인수가 있습니다.

1. *mapname*은 수신할 맵의 이름을 포함하는 변수입니다.
2. *map*은 맵이 작성될 메모리에 있는 영역의 주소입니다.

RESP 값은 응답으로 리턴됩니다. 응답의 선언은 자동 변수의 적절한 유형을 설정합니다.

EXEC CICS 문은 응답 변수가 보유 중인 조건의 값과 함께 변수 맵이 가리키는 메모리의 영역으로 판독될 맵핑 세트 ACCOUNT의 *mapname*에 의해 제공된 이름이 있는 맵을 요청합니다.

조건 처리는 if 문을 사용하여 수행할 수 있습니다. 그러나 가독성을 향상시키기 위해서는 여기에서처럼 복합 if ... else 문을 사용하는 대신에 switch 문을 사용하는 것이 낫습니다. 프로그램 실행 시간에 대한 영향은 대수롭지 않습니다.

두 개의 조건에 해당하는 특정 경우:

1. NORMAL의 조건은 일반적으로 예상되는 것입니다. 여기 예제에서 NORMAL 조건이 발견되면 함수는 사용자가 CICS로 돌아가기 위해 어떤 키를 눌렀는지를 알아내고 이 값이 ExitKey로 전달됩니다. 그러면 프로그램은 ModifyMap 함수에 의해 메모리에 보관된 맵을 일부 갱신하므로 이에 대해서는 더 이상 걱정할 필요가 없습니다.

2. 사용자가 화면에 갱신을 하지 않았음을 나타내는 MAPFAIL 조건 또한 지극히 정상이고 여기에서 처리합니다. 이 경우에는 프로그램이 ExitKey를 다시 갱신하지만 ModifyMap을 호출하지 않습니다.

이 예제에서는 다른 조건은 오류로 보류됩니다. 예제는 ExitKey를 사용자가 화면을 정리했다면 설정될 값과 동일한 값인 DFHCLEAR로 설정합니다. 그런 다음 이는 호출 프로그램으로 돌아갑니다. ReadAccountMap에서 리턴 코드를 확인하여 호출 프로그램은 맵이 갱신되지 않았고 몇몇 개선 조치가 필요함을 알게 됩니다.

COBOL에서 예외 처리의 예제

다음 예제는 BMS 맵을 수신하고 예외 조건에 대처하는 데 사용할 수 있는 일반적인 함수입니다.

```

03  RESPONSE                      PIC S9(8)  BINARY.
03  EXITKEY                       PIC X.

EXEC CICS RECEIVE MAP(MAPNAME)
      MAPSET('ACCOUNT')
      INTO(MAP)
      RESP(RESPONSE)
      END-EXEC.
IF (RESPONSE NOT = DFHRESP(NORMAL)) AND
   (RESPONSE NOT = DFHRESP(MAPFAIL))
   MOVE DFHCLEAR TO EXITKEY
ELSE
   MOVE EIBAID TO EXITKEY
   IF RESPONSE = DFHRESP(NORMAL)
     GO TO MODIFYMAP
   END-IF
END-IF.

MODIFYMAP.
```

그림 109. COBOL에서 예외 처리의 예제

MAPNAME은 수신할 맵의 이름을 포함하는 변수입니다.

RESP 값은 RESPONSE로 리턴됩니다. RESPONSE는 데이터 섹션에서 전자 2진 변수로서 선언됩니다.

EXEC CICS 문은 RESPONSE 변수가 보유하는 조건의 값과 함께 판독될 맵핑 세트 ACCOUNT의 MAPNAME이 제공하는 이름이 있는 맵을 요청합니다.

조건 처리는 IF ... 문을 사용하여 수행됩니다. 조건이 NORMAL이거나 MAPFAIL이면 프로그램은 사용자가 화면을 정리한 것처럼 행동합니다.

조건이 NORMAL이거나 MAPFAIL이면 프로그램은 사용자가 EXITKEY에서 화면을 종료하기 위해 누른 키의 값을 저장합니다. 또한 조건이 NORMAL이면 프로그램은 일부 추가 기능을 수행하기 위해 MODIFYMAP을 분기합니다.

기본 CICS 예외 처리 수정

애플리케이션에서 **HANDLE CONDITION**, **IGNORE CONDITION** 및 **HANDLE ABEND** 명령을 사용하여 CICS가 예외 및 이상 종료 처리하는 방법을 수정할 수 있습니다.

이 태스크 정보

이 정보는 COBOL, PL/I 또는 어셈블러 언어 애플리케이션(AMODE(64) 어셈블러 언어 애플리케이션은 해당되지 않음)에만 적용됩니다. 지원되는 다른 상위 레벨 언어에 대해서 이러한 명령은 지원되지 않습니다.

명령의 요약은 다음과 같습니다.

HANDLE CONDITION

조건이 발생되면 제어가 전달될 레이블을 지정합니다.

IGNORE CONDITION

조건이 발생하면 어떠한 조치도 취하지 않도록 지정합니다.

HANDLE ABEND

비정상 종료 처리에 대해 엑시트를 활성화 또는 취소하거나 재활성화합니다.

이상 종료는 CICS가 예외 조건 처리에 가장 많이 사용하는 방법입니다.

IGNORE CONDITION, **HANDLE ABEND** 및 **HANDLE CONDITION** 명령의 현재 영향은 **PUSH HANDLE** 명령을 사용하여 일시중단할 수 있고 **POP HANDLE** 명령을 사용하여 복원할 수 있습니다.

다음과 같은 방식으로 지정된 레이블에 제어를 전달할 수 있습니다.

- 조건이 예외 조건의 이름인 **HANDLE CONDITION** 조건(레이블) 명령을 사용합니다.
- **HANDLE CONDITION ERROR**(레이블) 명령을 사용합니다.

HANDLE CONDITION 명령은 CICS 코드를 이름별 조건으로 설정한 다음 해당 조건이 발생하면 이 코드를 사용하여 제어를 애플리케이션의 해당 섹션에 전달합니다. 활성 **HANDLE CONDITION** 명령에서 제어는 해당 특정 조건에 대해 지정한 레이블로 이동합니다.

여러 다른 명령에서 다른 이유로 같은 조건이 발생할 수 있습니다. 예를 들어, IOERR 조건은 파일 제어 조작, 간격 제어 조작 및 기타 작업 중에 발생할 수 있습니다. 따라서 조건이 발생한 이유를 조사하려면 먼저 특정 조건을 발생시킨 명령을 식별해야 합니다. 이는 새 CICS 애플리케이션에서 RESP 옵션을 사용하는 것이 좋은 이유입니다. 여러 조건에 대한 오류 처리를 설정하기 위해 한 개의 **HANDLE CONDITION** 명령만 필요하더라도 일부 경우에 CICS 명령이 코드 어딘가에서 실패했을 때 여러 **HANDLE CONDITION** 명령 중 현재 활성인 명령을 정확하게 식별하는 것이 어려울 수 있습니다.

이름을 지정하지 않은 조건이 발생할 경우 이 조치가 태스크를 이상 종료하는 것이 아니면 CICS가 기본 조치를 취합니다. 이 경우에 **ERROR** 조건이 발생합니다. 조건의 이름은 지정하고 레이블은 생략할 경우 해당 조건에 대한 **HANDLE CONDITION** 명령이 비활성화되고 이 조건이 발생하면 CICS가 기본 조치를 취합니다.

HANDLE CONDITION 명령에서 오류의 일반적인 소스는 모든 조건을 다루지는 않습니다. 익숙하지 않은 명령을 사용하는 경우 명령 설명을 확인하여 발생 가능한 예외 조건을 확인하십시오(애플리케이션 개발 참조서 참조). 모든 예외 조건에 대해 **HANDLE** 명령을 실행하는 경우에도 오류 처리 코드가 완료되지 않을 수 있습니다. 일부 경우 결과는 **RETURN** 명령을 실행하여 불완전하거나 정확하지 않은 데이터 변경사항이 커미트되도록 하는 오류 처리 루틴입니다.

좋은 접근법은 **HANDLE CONDITION** 명령을 사용하는 것이지만 특정 문제점을 피하는 명백한 방법이 없는 경우에는 시스템 기본 조치가 진행되도록 합니다.

대기를 거의 발생시키지 않는 조건(대기의 원인이 되는 조건의 예제는 431 페이지의 『CICS가 수행할 작업을 추적 관리하는 방법』 참조)인 오류 조건과 특수한 경우의 **SEND MAP** 명령 오버플로우 처리(**HANDLE CONDITION** 조건(레이블), 또는 **HANDLE CONDITION ERROR**(label) 명령이 애플리케이션에서 실행된 후에 **HANDLE CONDITION** 명령이 활성임) 간 차이점에 유의하십시오.

조건에 대해 활성인 **HANDLE CONDITION** 명령은 없지만 **ERROR**에 대해 활성인 명령은 있는 경우 조건이 대기가 아니라 오류이면 제어가 **ERROR**에 대한 레이블에 전달됩니다.

HANDLE CONDITION 명령을 사용하거나 이러한 명령을 사용하는 애플리케이션을 유지 보수하는 경우 명령을 오류 루틴에 포함시키지 마십시오. 포함시킬 경우 루프가 생길 수 있기 때문에 루틴에 대한 원래 분기를 제공한 동일한 조건이 발생하는 원인이 될 수 있습니다.

ERROR 조건 자체에서 루프가 생기지 않도록 특별히 주의하십시오. **ERROR** 조건에 대해 시스템 기본 조치로 임시로 되돌아가서 루프를 피할 수 있습니다. 레이블을 지정하지 않고 **HANDLE CONDITION ERROR** 명령을 코드화하여 이를 수행하십시오. 오류 처리 루틴 종료 시 적절한 레이블이 있는 **HANDLE CONDITION ERROR** 명령을 포함시켜 오류 조치를 복원할 수 있습니다. 이전 **HANDLE CONDITION** 상태를 아는 경우에 이를 명시적으로 수행할 수 있습니다. 코드의 여러 다른 위치에서 호출될 수 있는 일반 서브루틴에서 **PUSH HANDLE** 및 **POP HANDLE** 명령이 유용할 수 있습니다. 435 페이지의 『**PUSH HANDLE** 및 **POP HANDLE** 명령 사용』의 내용을 참조하십시오.

HANDLE CONDITION 명령 사용

HANDLE CONDITION 명령을 사용하여 조건이 발생하면 제어를 전달할 레이블을 지정합니다.

이 태스크 정보

이 명령을 사용할 때 조건의 이름을 포함시키고 연관된 조건이 발생되게 할 수 있는 명령 전에 **HANDLE CONDITION** 명령이 실행되도록 해야 합니다.

제한사항: 이 명령은 COBOL, PL/I, 어셈블러 언어 애플리케이션(AMODE(64) 어셈블러 언어 애플리케이션은 제외)으로만 지원됩니다. 지원되는 다른 모든 고급 언어로는 지원되지 않습니다.

같은 명령에 16개를 초과하는 조건을 포함시킬 수 없습니다. 추가 **HANDLE CONDITION** 명령에서 추가 조건을 지정해야 합니다. 같은 목록의 **ERROR** 조건을 사용하여 다른 모든 조건이 제어에 같은 레이블로 전달되게 하도록 지정할 수도 있습니다.

지정된 조건에 대한 **HANDLE CONDITION** 명령은 명령이 지정되는 프로그램에만 적용됩니다. 프로그램이 실행되는 동안 또는 다음 이벤트 중 하나가 발생할 때까지 이 명령은 활성 상태로 유지됩니다.

- 같은 조건에 대한 **IGNORE CONDITION** 명령이 충족됩니다. **HANDLE CONDITION** 명령을 덮어씁니다.
- 같은 조건에 대한 다른 **HANDLE CONDITION** 명령이 충족됩니다. 새 명령이 이전 명령을 덮어씁니다.

HANDLE CONDITION 명령이 명령의 **NOHANDLE** 또는 **RESP** 옵션에 의해 임시로 비활성화됩니다.

LINK 또는 **XCTL** 명령에 의해 다른 프로그램으로 제어가 전달되면 호출 프로그램에서 활성인 **HANDLE CONDITION** 명령이 비활성화됩니다. 낮은 논리 레벨의 프로그램에서 프로그램으로 제어가 리턴되면 제어가 전송되기 전에 더 높은 레벨의 프로그램에서 활성이었던 **HANDLE CONDITION** 명령이 재활성화되고 낮은 레벨 프로그램의 해당 명령은 비활성화됩니다(논리 레벨에 대한 정보는 366 페이지의 『애플리케이션 논리 레벨』 참조).

다음 예제는 **WRITE** 명령을 사용하여 데이터 세트에 레코드를 추가할 때 발생할 수 있는 조건을 처리하는 방식(예: **DUPREC** 및 **LENGERR**)을 보여줍니다. 예제에서 **DUPREC**는 특수한 경우로 처리됩니다. 표준 시스템 조치(즉, 태스크를 비정상적으로 종료)가 **LENGERR**에 대해 수행됩니다. 다른 모든 조건은 오류 루틴 **ERRHANDL**에 의해 처리됩니다.

```
EXEC CICS HANDLE CONDITION
      ERROR(ERRHANDL)
      DUPREC(DUPRTN) LENGERR
END-EXEC.
```

PL/I 애플리케이션에서 비활성 프로시저 또는 비활성 시작 블록에서 조건으로 인해 발생하는 레이블에 대한 분기는 예측하지 못한 결과를 생성합니다.

어셈블리 언어 애플리케이션에서 레이블에 대한 분기가 어떤 조건으로 인해 발생하면 조건의 원인이 된 명령이 실행된 위치에서 애플리케이션의 레지스터가 프로그램에서 해당 값으로 복원됩니다.

RESP 및 NOHANDLE 옵션

명령에서 RESP 또는 NOHANDLE 옵션을 사용하여 HANDLE CONDITION 명령의 영향을 임시로 비활성화할 수 있습니다.

이러한 옵션을 사용하는 방법은 424 페이지의 『인라인 코드로 예외 조건 처리』에 설명되어 있습니다. 이를 수행하면 해당 명령에 대한 모든 시스템 기본 조치를 사용할 수 없게 됩니다. 즉, 자체 『모두 발견』 오류 처리를 수행해야 합니다.

CICS가 수행할 작업을 추적 관리하는 방법

CICS에는 애플리케이션에서 HANDLE CONDITION 및 IGNORE CONDITION 명령이 참조하는 조건 테이블이 있습니다.

다음 명령 중 하나를 실행할 때마다 이 테이블의 기존 항목이 갱신되거나 해당 명령에서 이 조건이 처음으로 인용되는 경우 CICS가 새 항목을 작성합니다. 각 항목은 애플리케이션이 해당되는 세 가지 예외 처리 상태 중 하나를 표시하여 CICS에게 수행할 작업을 알립니다. 그 상태는 다음과 같습니다.

1. 프로그램이 계속 실행되게 함, CICS에서 프로그램의 실패한 명령 뒤 다음 지시사항으로 바로 전달된 제어를 사용합니다. 그러면 테스트에서 발생하는 상황을 확인할 수 있습니다(예: 명령 실행 후 CICS가 리턴하는 RESP 값). 이 테스트의 결과를 사용하여 다음에 수행할 작업을 결정할 수 있습니다. 세부사항은 424 페이지의 『인라인 코드로 예외 조건 처리』의 내용을 참조하십시오.

이는 권장되는 메소드이고 샘플 애플리케이션 안내서에서 참조된 『파일 A』 샘플 프로그램과 CICS 애플리케이션 디자인 및 프로그래밍의 COBOL 샘플 애플리케이션을 참조한 접근법입니다. 이는 모든 새 CICS 애플리케이션에 대해 권장되는 접근법입니다. 또한 구조화된 코드에 적합하고 과거에 CICS가 요구했던 내재된 GOTO에 대한 필요성을 제거합니다.

2. 이름 지정된 조건이 발생하는 경우 제어를 지정된 레이블에 전달. 이 상태를 처리하기 위해 코드에 루틴의 레이블 및 조건 둘 다의 이름을 지정하는 HANDLE CONDITION 명령 또는 HANDLE CONDITION ERROR 명령을 사용하여 이를 수행합니다. 세부사항은 429 페이지의 『HANDLE CONDITION 명령 사용』 및 432 페이지의 『HANDLE CONDITION ERROR 명령 사용』의 내용을 참조하십시오.
3. CICS 시스템 기본 조치를 수행, 대부분의 조건에서 이 상태는 태스크를 비정상적으로 종료하고 테스트 또는 조건 처리 과정에서 아무 것도 수행하지 않음을 의미합니다.

조건 ENQBUSY, NOJBUFSP, NOSTG, QBUSY, SESSBUSY 및 SYSBUSY의 경우 일반적인 기본값은 필요한 자원(예: 저장영역)이 사용 가능해지고 명령 처리를 재개할 때까지 태스크가 대기하도록 강제 실행하는 것입니다. NOSUSPEND 옵션을 사용하여 이 작동을 조건 무시로 변경할 수 있습니다. 조건 NOSPACE의 경우 일반적인 기본값은 WRITEQ TS 명령을 처리하는 경우 대기하고 WRITEQ TD, WRITE 또는 REWRITE 명령을 처리하는 경우 태스크를 이상 종료하는 것입니다. **WRITEQ TS** 명령을 NOSUSPEND 옵션을 사용하여 코드화하면 이 명령은 발생하는 모든 NOSPACE 조건을 무시합니다. 자세한 정보는 다음 항목을 참조하십시오. *CICS 애플리케이션 프로그래밍 참조서*.

CICS는 각 링크 레벨에 대한 이러한 조건 테이블을 보존합니다. 따라서 각 프로그램 레벨에는 자체 조건 처리를 제어하는 자체 HANDLE 상태 테이블이 있어야 합니다.

이 작동은 HANDLE CONDITION ERROR 및 IGNORE CONDITION으로 수정됩니다.

HANDLE CONDITION ERROR 명령 사용

다음 예제는 HANDLE CONDITION ERROR 명령을 사용하여 예상치 못한 오류를 트랩하는 방법을 보여줍니다.

이 태스크 정보

그림 110에서는 프로그램 ACCT01에서 사용되는 단 두 개의 HANDLE CONDITION 명령 중 첫 번째 명령을 보여줍니다.

```
PROCEDURE DIVISION.
*
*   INITIALIZE.
*   TRAP ANY UNEXPECTED ERRORS.
*   EXEC CICS HANDLE CONDITION
*       ERROR(OTHER-ERRORS)
*       END-EXEC.
*
```

그림 110. HANDLE CONDITION ERROR 명령을 사용하여 예상치 못한 오류 트랩

HANDLE CONDITION ERROR 명령은 NOHANDLE 또는 RESP를 지정하지 않는 명령에 대해 어떠한 조건이 발생하면 레이블 OTHER-ERRORS의 단락으로 제어를 전달합니다.

이 명령은 이 COBOL 프로그램의 프로시저 디비전에서 실행되는 첫 번째 명령입니다. 처리 중인 조건을 발생시킬 수 있는 CICS 명령이 처리되기 전에 HANDLE CONDITION 명령이 처리되어야 하기 때문입니다. 그러나 HANDLE CONDITION

명령을 처리할 때 프로그램은 그 영향을 알 수 없고 나중에 프로그램이 이름 지정된 조건 중 하나를 발생시키는 CICS 명령을 실행하는 경우에만 그러한 영향을 알 수 있습니다.

이 명령과 다른 ACCT 프로그램에서는 일반적으로 RESP 옵션을 사용합니다. RESP 옵션을 지정하는 모든 명령은 특정 조건에 대한 명시적 테스트 후에 『모두 발견』 테스트(IF RESPONSE NOT = DFHRESP(NORMAL) GO TO OTHER-ERRORS)에서 작성됩니다. 따라서 분명히 예상되는 예외가 아닌 예외가 각 프로그램에서 OTHER-ERRORS에 있는 단락에 대한 제어를 가집니다. 그러한 상대적으로 적은 수의 RESP가 없는 명령은 이 HANDLE CONDITION ERROR 명령으로 인해 그 결과가 NORMAL 외의 조건인 경우 같은 장소에 대한 제어를 가집니다.

IGNORE CONDITION 명령 사용

IGNORE CONDITION 명령을 사용하여 특정 조건이 발생하면 프로그램이 계속해서 실행되도록 지정할 수 있습니다(특정 조건이 발생하면 제어가 지정된 레이블로 전달되도록 지정하는 HANDLE CONDITION 명령과 반대임).

이 태스크 정보

제한사항: 이 명령은 COBOL, PL/I, 어셈블러 언어 애플리케이션(AMODE(64) 어셈블러 언어 애플리케이션은 제외)으로만 지원됩니다. 지원되는 다른 모든 고급 언어로는 지원되지 않습니다.

명령에서 잠재적으로 발생할 수 있는 한 개 이상의 조건을 무시하도록 IGNORE CONDITION 명령을 설정할 수 있습니다. IGNORE CONDITION 명령은 조건이 발생하면 어떠한 조치도 취하지 않고 제어가 이 명령 뒤의 명령어로 리턴되며 리턴 코드가 EIB에서 설정되는 것을 의미합니다. 다음 예제에서는 MAPFAIL 조건을 무시합니다.

```
EXEC CICS IGNORE CONDITION MAPFAIL
END-EXEC.
```

단일 EXEC CICS 명령이 처리될 때 여러 조건이 발생할 수 있습니다. 예를 들어, 파일 제어 명령이 올바르지 않거나 파일 제어 테이블에 정의되지 않은 파일에 적용될 수도 있습니다. CICS는 이러한 조건을 확인하고 (IGNORE CONDITION 명령에 의해) 무시되지 않는 첫 번째 조건을 애플리케이션에 다시 전달합니다. CICS는 한 번에 한 개의 예외 조건만 애플리케이션에 전달합니다.

지정된 조건에 대한 IGNORE CONDITION 명령은 이 명령을 삽입하는 프로그램에만 적용됩니다. 이 명령은 프로그램이 실행되는 동안 또는 나중에 HANDLE CONDITION 명령이 충족될 때까지(이 경우 IGNORE CONDITION 명령을 덮어쓰) 활성화 상태로 유지됩니다.

제공한 공간보다 길 수도 있는 레코드를 읽는 프로그램에 대해 IGNORE CONDITION 명령을 사용할 수 있지만 이를 오류로 간주하지 않고 어떠한 조치도 수행하지 않으려고 합니다. 따라서 READ 명령을 실행하기 전에 IGNORE CONDITION LENGERR을 코드화할 수 있습니다.

또한 IGNORE CONDITION ERROR 명령을 사용하여 오류로 간주되는 조건(레이블을 포함하고 현재 활성인 HANDLE CONDITION 명령이 없음)을 발견할 수 있습니다. 오류가 발생하면 제어가 다음 명령문으로 전달되고 프로그램은 EIB에서 리턴 코드를 확인해야 합니다. 오류로 간주되지 않는 조건의 예제는 431 페이지의 『CICS가 수행할 작업을 추적 관리하는 방법』의 내용을 참조하십시오.

또한 조건 무시에서 조건 처리 또는 시스템 기본 조치 사용으로 전환할 수 있습니다. 예를 들어, 다음을 코드화할 수 있습니다.

```
* MIXED ERROR PROCESSING
EXEC CICS IGNORE CONDITION LENGERR
END-EXEC.

EXEC CICS HANDLE CONDITION DUPREC(DUPRTN)
LENGERR
ERROR(ERRHANDL)
END-EXEC.
```

이 코드는 처음에 조건 LENGERR을 무시하기 때문에 프로그램에 LENGERR 조건이 발생하면 아무 조치도 발생하지 않고 애플리케이션이 처리를 계속합니다. 물론, LENGERR이 발생했다는 사실은 애플리케이션이 정상적으로 계속될 수 없고 문제점이 있다는 것을 의미합니다.

코드 뒷 부분에서 시스템 기본 조치에 대한 조건 LENGERR을 레이블 없이 HANDLE CONDITION 명령에서 이름 지정하여 명시적으로 설정할 수 있습니다. 이 명령이 실행되면 프로그램이 조건 LENGERR을 더 이상 무시하지 않고 이후에 이 조건이 다시 발생하면 그로 인해 시스템 기본 조치가 수행됩니다. 메소드 조합에 대한 요점은 사용자가 할 수 있다는 것과 각 조건이 개별적으로 처리된다는 것입니다.

같은 명령에서 16개를 초과하는 조건을 코드화할 수 없습니다. 추가 IGNORE CONDITION 명령에서 추가 조건을 지정해야 합니다.

HANDLE ABEND 명령 사용

HANDLE ABEND 명령은 애플리케이션에서 프로그램 레벨 이상 종료 엑시트를 활성화하거나 재활성화합니다. 이 명령을 사용하여 이전에 활성화된 엑시트도 취소할 수 있습니다.

특정 프로그래밍 언어에 대해 HANDLE ABEND 명령을 사용하여 이상 종료가 처리될 때 실행할 자체 코드를 제공할 수 있습니다. 이는 애플리케이션이 체계적인 방식으

로 비정상적인 상황에 대처하고 실행을 계속할 수 있다는 것을 의미합니다. 사용자 엑시트 프로그램을 제공하고 CICS에 의존하여 필요 시 해당 프로그램을 호출합니다.

이상 종료 처리 중 제어의 플로우는 447 페이지의 그림 111에 표시되어 있습니다.

제한사항

- **HANDLE ABEND** 명령은 Java 프로그램에 적용되지 않습니다.
- C 및 C++ 프로그램의 예외 조건은 이상 종료를 발생시키지 않기 때문에 이러한 프로그램에서는 이 방식으로 **HANDLE ABEND** 명령을 사용할 수 없습니다. 그러나 **HANDLE ABEND** 명령은 PROGRAM 옵션과 함께 사용될 때 C 및 C++에서 지원됩니다.
- DFHEIENT 및 DFHEIRET를 사용하지 않는 어셈블러 프로그램에서 **HANDLE ABEND LABEL**을 사용할 수 없습니다. Language Environment 스텝 CEESTART를 사용하는 어셈블러 프로그램은 **HANDLE ABEND PROGRAM** 또는 **CEEHDLR**과 같은 Language Environment 서비스를 사용해야 합니다.
- AMODE(64) 프로그램에서 **HANDLE ABEND LABEL**을 사용할 수 없습니다.

PUSH HANDLE 및 POP HANDLE 명령 사용

PUSH HANDLE 및 **POP HANDLE** 명령은 **HANDLE CONDITION**, **IGNORE CONDITION**, **HANDLE ABEND** 및 **HANDLE AID** 명령의 영향을 각각 일시중단하고 복원합니다.

PUSH HANDLE

HANDLE CONDITION, **IGNORE CONDITION**, **HANDLE ABEND** 및 **HANDLE AID** 명령의 현재 영향을 일시중단합니다.

POP HANDLE

HANDLE CONDITION, **IGNORE CONDITION**, **HANDLE ABEND** 및 **HANDLE AID** 명령을 이전 **PUSH HANDLE**이 호출되기 전의 상태로 복원합니다.

제한사항: 이러한 명령은 COBOL, PL/I 및 어셈블러 언어 애플리케이션(AMODE(64) 어셈블러 언어 애플리케이션은 해당되지 않음)에서만 지원됩니다. 이러한 명령은 지원되는 다른 모든 상위 레벨 언어에 대해 지원되지 않습니다.

CICS는 일치하는 **POP HANDLE** 명령에 의해 취소되지 않은 각 **PUSH HANDLE** 명령에 대한 조건 테이블도 보존합니다.

각 조건이 발생하면 CICS는 다음 테스트 순서를 사용합니다.

1. 명령에 **RESP** 또는 **NOHANDLE** 옵션이 있는 경우 제어가 애플리케이션의 다음 명령어로 리턴합니다. 그렇지 않으면 CICS가 조건 테이블을 스캔합니다.
2. 조건에 대한 항목이 있으면 이 항목이 조치를 판별합니다.
3. 항목이 없고 이 조건에 대한 기본 조치가 실행을 일시중단하는 것이면 조치는 다음과 같습니다.

- a. 명령에 NOSUSPEND 또는 NOQUEUE 옵션이 있는 경우 다음 명령어로 제어가 리턴됩니다.
 - b. 명령에 NOSUSPEND 또는 NOQUEUE 옵션이 없는 경우 태스크는 일시중단됩니다.
4. 항목이 없고 이 조건에 대한 기본 조치가 이상 종료인 경우 CICS는 ERROR 조건을 검색합니다.
- a. ERROR 조건을 찾은 경우 이 항목이 조치를 판별합니다.
 - b. ERROR를 찾을 수 없는 경우 태스크가 이상 종료됩니다. 이상 종료 처리를 선택할 수 있습니다.

참고: SEND MAP 명령에서 OVERFLOW 조건은 규칙에 대한 예외입니다.

ALLOCATE, ENQ, GETMAIN, WRITE JOURNALNAME, WRITE JOURNALNUM, READQ TD 및 WRITEQ TS 명령은 모두 기본 조치가 지정된 자원이 사용 가능해질 때까지 애플리케이션을 일시중단하는 것인 조건을 발생시킬 수 있습니다. 따라서 이러한 명령에서 NOSUSPEND 옵션을 사용하여 이러한 대기를 막고 애플리케이션의 다음 명령어로 즉시 리턴할 수 있습니다.

일부 조건은 다수의 비관련 명령을 실행하는 동안 발생할 수 있습니다. 모든 발생에 대해 같은 조치를 원하는 경우 프로그램 시작 시 단일 **HANDLE CONDITION** 명령을 코드화하십시오.

참고: RESP를 사용하는 것은 NOHANDLE를 의미하므로 **RECEIVE** 명령에서 RESP를 사용할 때에는 주의하십시오. 이 경우 **HANDLE CONDITION** 명령 외에도 **HANDLE AID** 명령을 덮어쓰기 때문입니다. 이는 기능 키 응답이 무시되고 ACCT 코드에서 먼저 기능 키 응답을 테스트하는 이유입니다. 961 페이지의 『HANDLE AID 명령 사용』의 내용을 참조하십시오.

SOAP 결함 메시지 구문 분석

EXEC CICS INVOKE WEBSERVICE 또는 **EXEC CICS INVOKE SERVICE** 명령을 사용하여 원격 웹 서비스를 호출하는 경우 RESP2 값이 6인 INVREQ 응답을 수신할 수 있는데, SOAP 결함 메시지가 리턴되었음을 나타냅니다. z/OS용 CICS Transaction Server(CICS TS) V4.1 이상에서는 애플리케이션의 SOAP 결함 메시지를 구문 분석할 수 있습니다.

문제점

애플리케이션의 SOAP 결함 메시지에 대해 자세히 알고 싶습니다. 예를 들어, 결함 메시지가 CICS에서 시작되었는지 또는 원격 애플리케이션에서 시작된 것인지 알고 싶습니다. 또는 결함 메시지의 일부 임베디드 데이터에 액세스하려고 합니다.

응답

CICS TS V3.1 및 CICS TS V3.2에서는 CICS가 리턴한 DFHWS-BODY 컨테이너를 읽어 SOAP 결합 메시지의 XML 표현에 액세스할 수 있습니다. 선택한 메커니즘을 사용하여 XML 데이터를 구문 분석할 수 있습니다. CICS TS V4.1 이상에서는 새 XML 구문 분석 API(Application Programming Interface) 명령을 사용하여 이 프로세스에서 도움을 받을 수 있습니다. 이 정보는 **TRANSFORM** 명령을 사용하여 SOAP V1.1 결합 메시지 구문 분석 방법을 설명합니다. 비슷한 접근방식을 사용하여 SOAP V1.2 결합 메시지를 처리할 수 있습니다.

DFHSC2LS를 사용하여 SOAP 결합에 대한 COBOL 바인딩 빌드

<http://schemas.xmlsoap.org/soap/envelope/> 위치에서 SOAP 엔벨로프에 대한 XML 스키마 사본을 다운로드하십시오. 예를 들어, 이 스키마를 UNIX 파일 시스템의 `/u/example/source/SOAP11.xsd` 위치에 저장하십시오. DFHSC2LS를 사용하여 XML 스키마를 처리하고 스키마에 대한 COBOL 바인딩 세트를 출력으로 작성하고 번들 디렉토리에 XSDBind 파일을 작성하십시오. 다음 예제와 비슷한 JCL을 사용할 수 있습니다.

```
//EXAMPLE EXEC DFHSC2LS,
//INPUT.SYSUT1 DD *
MAPPING-LEVEL=3.0
ELEMENTS=Body,Fault
SCHEMA=/u/example/source/SOAP11.xsd
LANG=COBOL
PDSLIB=//EXAMPLE.COBOL.LIBRARY
PDSMEM=SOAP11
XSDBIND=SOAP11.xsdbind
BUNDLE=/u/example/output/bundle/SOAP11
LOGFILE=/u/example/output/logfile.log
*/
```

DFHSC2LS는 다음과 같이 몇 개의 COBOL 언어 구조를 작성합니다.

- SOAP 엔벨로프 본문에 대한 바인딩

```
03 Body.
   06 Body-num                PIC S9(9) COMP-5 SYNC.
   06 Body-cont                PIC X(16).

01 SOAP1101-Body.
   03 Body-xml-cont            PIC X(16).
   03 Body-xmlns-cont          PIC X(16).
```

이 언어 구조는 SOAP 본문에 임의의 수의 XML 태그가 나타나도록 허용하기 위한 바인딩을 포함합니다. CICS는 Body-num 필드에 찾은 태그 수를 저장하고 Body-cont 필드가 이름 지정한 컨테이너에 데이터에 대한 정보를 저장합니다. 본문의 각 XML 태그에는 이와 연관된 두 개의 필드가 있는데 Body-xml-cont에 이름 지정된 컨테이너의 태그에 대한 XML을 제공하고 Body-xmlns-cont에 이름 지정된 컨테이너의 범위 내 XML 이름 공간 선언을 제공합니다.

- SOAP 엔벨로프 본문의 결함에 대한 바인딩

```

03 Fault.
    06 faultcode-length      PIC S9999 COMP-5 SYNC.
    06 faultcode             PIC X(255).
    06 faultstring-length    PIC S9999 COMP-5 SYNC.
    06 faultstring           PIC X(255).
    06 faultactor-num        PIC S9(9) COMP-5 SYNC.
    06 faultactor.
        09 faultactor2-length PIC S9999 COMP-5 SYNC.
        09 faultactor2       PIC X(255).
    06 detail3-num           PIC S9(9) COMP-5 SYNC.
    06 detail2.
        09 Xdetail-num        PIC S9(9) COMP-5 SYNC.
        09 Xdetail-cont       PIC X(16).

01 SOAP1102-Xdetail.
    03 detail-xml-cont       PIC X(16).
    03 detail-xmlns-cont    PIC X(16).
  
```

이 언어 구조는 단일 SOAP 결함 구문 분석을 허용하기 위한 바인딩을 포함합니다. SOAP 결함의 detail 섹션 내에서 찾은 임의의 수의 XML 태그를 매핑하기 위한 구조와 함께 faultcode, faultstring 및 faultactor 필드에 대한 액세스를 제공합니다.

CICS에 번들 설치

다음 예제와 같은 BUNDLE 정의를 작성하여 설치하십시오.

GROUP: EXAMPLE

DESCRIPTION: Bundle for mapping SOAP 1.1 SOAP Faults

BUNDLEDIR: /u/example/output/bundle/SOAP11

BUNDLEDIR은 DFHSC2LS의 BUNDLE 매개변수를 사용하여 지정한 위치를 가리킵니다. CICS가 사용하는 것과 다른 z/OS 이미지에서 DFHSC2LS를 실행하는 경우 대상 시스템에 번들 디렉토리를 복사해야 합니다. 이런 상황에서는 다른 디렉토리 경로를 사용하고 그에 따라 BUNDLEDIR 값을 설정하십시오. 임의의 번들 이름을 설정할 수 있습니다. SOAP11이 아니어도 됩니다.

번들이 CICS에 설치되면 SOAP11이라는 BUNDLE 자원과 SOAP11이라는 XMLTRANSFORM 자원도 갖게 됩니다. XMLTRANSFORM 이름은 DFHSC2LS의 XSDBIND 매개변수 값에서 파생됩니다.

SOAP 결함 메시지 예제

다음은 **EXEC CICS INVOKE WEBSERVICE** 명령 다음에 DFHWS-BODY 컨테이너에서 찾을 수 있는 SOAP 결함 메시지 예제입니다.


```

SOAP-ENV:Body>
  <SOAP-ENV:Fault xmlns="">
    <faultcode>SOAP-ENV:Server</faultcode>
    <faultstring>Conversion to SOAP failed</faultstring>
    <detail>
      <CICSFault xmlns="http://www.ibm.com/software/http/cics/WSFault">
        DFHPI1010 *** XML generation failed. A conversion error
        INVALID_PACKED_DEC) occurred when converting field 'example' for
        WEBSERVICE 'testWebservice'.
      </CICSFault>
    </detail>
  </SOAP-ENV:Fault>
</SOAP-ENV:Body>

```

이 예제는 변환 오류가 발생할 때 CICS가 작성하는 결함 메시지입니다. **TRANSFORM** 명령이 결함 메시지를 처리할 때 Body-num은 1로 설정되어 Body 태그에 단일 XML 태그가 있음을 나타냅니다. Body-cont는 컨테이너 이름으로 설정됩니다(예: DFHPICC-00000001).

두 개의 추가 컨테이너 이름이 DFHPICC-00000001 컨테이너 내에 배치됩니다(예: DFHPICC-00000002 및 DFHPICC-00000003).

컨테이너 DFHPICC-00000002는 Body 태그의 첫 번째 태그를 포함합니다. 예를 들어, 다음과 같습니다.

```

SOAP-ENV:Fault xmlns="">
  <faultcode>SOAP-ENV:Server</faultcode>
  <faultstring>Conversion to SOAP failed</faultstring>
  <detail>
    <CICSFault xmlns="http://www.ibm.com/software/http/cics/WSFault">
      DFHPI1010 *** XML generation failed. A conversion error
      INVALID_PACKED_DEC) occurred when converting field 'example' for
      WEBSERVICE 'testWebservice'.
    </CICSFault>
  </detail>
</SOAP-ENV:Fault>

```

컨테이너 DFHPICC-00000003은 범위 내 이름 공간 선언을 포함합니다. 예를 들어, 다음과 같습니다.

```

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "

```

DFHPICC-00000002 컨테이너가 두 번째 **EXEC CICS TRANSFORM** 명령을 통해 구문 분석되면 추가 출력이 작성됩니다. faultcode 및 faultcode-length 필드는 SOAP-ENV:Server 및 15로 설정됩니다. faultstring 및 faultstring-length 필드는 『Conversion to SOAP failed』 및 25로 설정됩니다. faultactor-num 필드는 0으로 설정됩니다. detail3-num 필드는 1로 설정되어 선택적 detail 태그가 결함에 존재함을 나타냅니다. detail2-num 필드는 1로 설정되어 선택적 detail 태그에 하나의 서브 태그가 있음을 나타냅니다. detail2-cont 필드는 컨테이너 이름으로 설정됩니다(예: DFHPICC-00000004).

컨테이너 DFHPICC-00000004는 두 개의 추가 컨테이너 이름을 포함합니다(예: DFHPICC-00000005 및 DFHPICC-00000006).

컨테이너 DFHPICC-00000005는 SOAP 결함의 detail 섹션에서 찾은 첫 번째 XML 태그를 포함합니다. 예를 들어, 다음과 같습니다.

```
CICSFault xmlns="http://www.ibm.com/software/htp/cics/WSFault ">
  DFHPI1010 *** XML generation failed. A conversion error
  (INVALID_PACKED_DEC) occurred when converting field 'example'
  for WEBSERVICE 'testWebservice'.
</CICSFault>
```

컨테이너 DFHPICC-00000006은 범위 내 이름 공간 선언을 포함합니다. 예를 들어, 다음과 같습니다.

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
```

애플리케이션 코드

SOAP 결함을 구문 분석하기 위한 애플리케이션을 구현하려면 다음 프로시저를 사용하십시오.

1. **TRANSFORM** 명령을 호출하여 DFHWS-BODY 컨테이너의 콘텐츠를 조회하십시오. 예를 들어, 다음과 같습니다.

```
EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
XMLCONTAINER('DFHWS-BODY') NSCONTAINER('DFHWS-XMLNS')
ELEMNAME(element-name) ELEMNAMELEN(element-name-len)
END-EXEC
```

2. element-name이 Body로 설정된 경우 컨테이너를 구문 분석하십시오. 그렇지 않은 경우에는 오류가 발생합니다. Body를 구문 분석하려면 다음 명령을 사용하십시오.

```
EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
XMLTRANSFORM('SOAP11') XMLCONTAINER('DFHWS-BODY')
NSCONTAINER('DFHWS-XMLNS') DATCONTAINER('PARSEDBODY')
END-EXEC
```

```
EXEC CICS GET CONTAINER('PARSEDBODY') SET(body-ptr) END-EXEC
```

3. 구문 분석된 데이터의 주소를 지정하십시오. 예를 들어, 다음과 같습니다.

```
SET ADDRESS OF Body TO body-ptr
```

4. Body-num을 확인하여 Body에 하나 이상의 항목이 있는지 확인하십시오. 항목이 있으면 세부사항을 나열하는 컨테이너를 읽으십시오. 예를 들어, 다음과 같습니다.

```
EXEC CICS GET CONTAINER(Body-cont) SET(body-cont-ptr) END-EXEC
```

```
SET ADDRESS OF SOAP1101-Body TO body-cont-ptr
```

5. **TRANSFORM** 명령을 다시 호출하여 Body의 첫 번째 태그를 조회하십시오.

```
EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
XMLCONTAINER(Body-xml-cont) NSCONTAINER(Body-xmlns-cont)
ELEMNAME(element-name) ELEMNAMELEN(element-name-len)
END-EXEC
```

6. element-name이 Fault로 설정된 경우 컨테이너를 구문 분석하십시오.

```
EXEC CICS TRANSFORM XMLTODATA CHANNEL(channel-name)
XMLTRANSFORM('SOAP11') XMLCONTAINER(Body-xml-cont)
NSCONTAINER(Body-xmlns-cont) DATCONTAINER('PARSEDFAULT') END-EXEC
```

```
EXEC CICS GET CONTAINER('PARSEDFAULT') SET(fault-ptr) END-EXEC
```

```
SET ADDRESS OF Fault TO fault-ptr
```

7. 이제 결합의 데이터를 조회할 수 있습니다. 예를 들어, faultstring 필드가 유용함을 알 수 있습니다. 결합의 detail 섹션에서 애플리케이션 특정 세부사항을 구문 분석하려면 DFHSC2LS를 사용하고 애플리케이션에서 **TRANSFORM** 명령을 추가로 실행하여 애플리케이션 특정 COBOL 바인딩을 추가로 빌드하십시오.

참고: 1단계와 2단계를 단일 **TRANSFORM** 명령으로 결합할 수 있으며 5단계와 6단계도 결합할 수 있습니다.

배수 효과 방지

자원 관리자에 대한 연결이 활성 상태가 아님을 알고 있으므로 애플리케이션이 자원 관리자에 대한 호출을 방지하지만 대신 오류 메시지를 표시하고 정상적으로 리턴하는 경우 CICS 리전에 추가 작업 경로를 지정하도록 워크로드 관리자를 속일 수 있습니다. 이 상황을 "배수 효과"라고 합니다.

애플리케이션이 이상 종료되지 않았으므로 워크로드 관리자는 자원 관리자와 관련된 작업에서 이 CICS 리전이 나타나는 응답 시간이 양호하다고 판단하므로 추가 작업을 라우팅합니다(배수 효과("storm drain")). 이 상황은 CICS에서 DB2, IMS, WebSphere MQ 및 VSAM RLS로의 연결을 사용하는 애플리케이션에서 발생할 수 있습니다.

"배수 효과"를 방지하려면 자원 관리자 어댑터가 z/OS® Workload Manager에 요청이 실패했음을 알려야 합니다. 그러면 CICSplex® SM 동적 라우팅 프로그램에서 이 정보를 사용할 수 있습니다(즉, DTRPGM=EYU9XLOP로 설정하여 실행할 때). (이 정보는 EXEC CICS START를 라우팅하기 위해 호출되는 DSRTPGM=xxxxxxx를 통해 지정된 분산 라우팅 프로그램에서 사용하지 않습니다.) CICSplex SM 동적 라우팅 프로그램에서는 이상 종료 확률 임계값을 사용하여 영향받는 리전에 작업이 라우팅되지 않도록 이 정보를 사용할 시기를 결정합니다. 기본적으로 이상 종료 확률 기능은 활성화 아니므로 워크로드 스펙의 ABENDCRIT 및 ABENDTHRESH 속성을 지정해야 합니다. 이상 종료 가능성 및 워크로드 관리의 내용을 참조하십시오.

활성화된 경우 영향받는 CICS 리전에 추가 작업이 라우팅되지 않도록 CICSplex® SM 동적 라우팅 프로그램에서 이상 종료 확률 기능을 사용할 수 있게 허용하도록 다음 자원의 자원 관리자 어댑터가 z/OS® Workload Manager에 알려야 합니다.

DB2

CICS와 DB2 사이의 연결에서 -923 SQL 리턴 코드가 리턴될 때 CICS DB2 첨부 기능은 요청에 실패했음을 z/OS Workload Manager에 알립니다. DB2CONN 정의는 STANDBYMODE=RECONNECT 및 CONNECTERROR=SQLCODE 속성으로 구성해야 합니다. 애플리케이션에서 먼저 EXEC CICS EXTRACT EXIT 또는 EXEC CICS INQUIRE EXITPROGRAM 명령을 사용하여 CICS가 DB2에 연결되어 있는지 테스트하지 않고 무조건 EXEC SQL 요청을 발행한 다음 SQLCODE(-923)를 테스트해야 합니다. -923 SQLCODE가 리턴된 경우에만 z/OS® Workload Manager에 알립니다.

IMS

CICS 및 IMS 사이의 연결에서 CICS 및 IMS DBCTL 사이의 인터페이스가 활성 상태가 아니므로 **CALLDLI** 요청을 발행하는 애플리케이션은 UIB에서 리턴 코드 08FF를 수신합니다. 이때 CICS-DBCTL 인터페이스는 z/OS Workload Manager에 요청에 실패했음을 알립니다. **EXEC DLI** 요청의 경우 일반적으로 **NODHABEND** 키워드를 사용하지 않는 한, 요청으로 인해 애플리케이션이 이상 종료됩니다. 이 상황에서도 배수 효과를 방지하기 위해 z/OS Workload Manager에 알립니다. .

WebSphere MQ

CICS와 WebSphere MQ 사이의 연결인 경우 CICS가 큐 관리자에 연결되지 않으면 CICS-WebSphere MQ 어댑터로 들어가기 전에 요청이 거부되므로 배수 효과를 방지할 수 없습니다.

CICS-WebSphere MQ 어댑터가 활성 상태일 때 후속 연결에 실패하고 다음 MQI 이 유 코드가 리턴되면 요청에 실패했음을 z/OS Workload Manager에 알립니다.

- mqrc_connection_broken
- mqrc_q_mgr_quiescing
- mqrc_connection_quiescing
- mqrc_connection_not_authorized
- mqrc_q_mgr_name_error
- mqrc_q_mgr_not_available
- mqrc_q_mgr_stopping
- mqrc_connection_stopping
- mqrc_adapter_not_available

VSAM RLS

CICS와 VSAM RLS 사이의 연결인 경우 CICS가 RLS 실패 또는 RLS를 사용할 수 없음을 알리는 응답을 수신하거나 이전 RLS에서 장애가 발생하면 보통 결과는 각각 AFCE, AFCS 또는 AFCT 이상 종료입니다. 이상 종료를 처리하는 경우 CICS 파일 제어는 요청에 실패했음을 z/OS Workload Manager에 알립니다.

비정상 종료 복구

CICS는 태스크의 비정상 종료 중에 제어를 수신할 수 있는 자체 엑시트(프로그램 또는 루틴)를 작성할 수 있도록 프로그램 레벨의 이상 종료 엑시트 기능을 제공합니다. 이런 이상 종료 엑시트로 수행되는 기능의 예로는 시작되었지만 제대로 완료되지 않은 프로그램 정리가 있습니다.

비정상 종료를 초래하는 몇 가지는 다음과 같습니다.

- 예를 들어, 다음 코드로 인한 사용자 요청:
EXEC CICS ABEND ABCODE(...)
- 올바르게 않은 사용자 요청으로 인한 CICS 요청. 예를 들어, 올바르게 않은 FREEMAIN 요청은 트랜잭션 이상 종료 코드 ASCF를 작성합니다.
- 프로그램 확인. 시스템 복구 프로그램(DFHSRP)이 구동되고 태스크는 코드 ASRA로 이상 종료됩니다.
- 운영 체제 이상 종료. DFHSRP 프로그램이 구동되고 태스크는 코드 ASRB로 이상 종료됩니다.
- 반복되는 태스크. DFHSRP 프로그램이 구동되고 태스크는 코드 AICA로 이상 종료됩니다.

참고: ASRB 또는 ASRA가 CICS 코드에서 발견되면, CICS는 HANDLE ABEND 엑시트를 호출하기 전에 덤프를 작성합니다.

문제점 해결에 대한 자세한 정보는 문제점 해결 및 지원의 내용을 참조하십시오. CICS, 해당 의미, 사용자 응답으로 시작되는 비정상 종료에 대한 트랜잭션 이상 종료 코드 정보는 의 내용을 참조하십시오.

HANDLE ABEND 명령은 사용자의 애플리케이션에서 프로그램 레벨의 이상 종료 엑시트를 활성화 또는 다시 활성화합니다. 이 명령을 사용하여 이전에 활성화한 엑시트를 취소할 수도 있습니다.

엑시트를 활성화할 때 다음 중 하나를 수행해야 합니다.

- PROGRAM 옵션을 사용하여 제어를 수신하는 프로그램 이름을 지정하십시오.

- COBOL 또는 어셈블러 애플리케이션(AMODE(64) 어셈블러 애플리케이션은 제외)에 대해서만 LABEL 옵션을 사용하여 비정상 종료 조건이 발생할 때 분기를 제어하는 루틴 레이블을 지정하십시오.

HANDLE ABEND LABEL 명령에는 C, C++, PL/I 또는 AMODE(64) 애플리케이션을 사용할 수 없습니다. PL/I에서 동등한 결과를 얻으려면 ON ERROR 블록을 사용하십시오.

HANDLE ABEND 명령은 동일한 논리 레벨에서 모든 애플리케이션의 이런 모든 선행 명령을 덮어씁니다. 트랜잭션의 각 애플리케이션은 자체적으로 고유한 이상 종료 엑시트를 포함할 수 있지만 각 논리 레벨에서 한 개의 이상 종료 엑시트만 활성화할 수 있습니다. (논리 레벨은 364 페이지의 『프로그램 제어』에서 설명됩니다.)

태스크가 비정상적으로 종료되면 CICS가 활성 이상 종료 엑시트를 검색하며 이 때 이상 종료가 발생한 애플리케이션의 논리 레벨부터 시작하여 더 높은 레벨로 진행합니다. 찾은 첫 번째 활성 이상 종료 엑시트(있는 경우)에 제어가 지정됩니다. 이 프로시저는 447 페이지의 그림 111에서 보여주며 후속 이상 종료 처리가 사용자 작성 이상 종료 엑시트로 판별되는 방법도 보여줍니다.

CICS가 이상 종료 엑시트를 찾지 못하면 제어를 비정상 조건 프로그램으로 전달하여 태스크를 비정상적으로 종료하도록 합니다. 이 프로그램은 사용자 대체 가능 프로그램 오류 프로그램인 DFHPEP를 호출합니다. DFHPEP 사용자 정의 방법에 대한 프로그래밍 정보는 시스템 프로그램 개발에서 프로그램 오류 프로그램 작성의 내용을 참조하십시오.

CICS는 엑시트 루틴이나 프로그램에 대한 항목에서 엑시트를 비활성화하여 이상 종료 엑시트에서 반복적인 이상 종료를 금지합니다. 조작을 다시 시도하려면 이상 종료 시에 제어에 있던 프로그램의 지점으로 분기하고 **HANDLE ABEND RESET** 명령을 실행하여 이상 종료 엑시트를 다시 활성화할 수 있습니다. 이 명령을 사용하여 **HANDLE ABEND CANCEL** 명령으로 이전에 취소되었던 이상 종료 엑시트(프로그램 발행의 논리 레벨에서)를 다시 활성화할 수도 있습니다. 435 페이지의 『PUSH HANDLE 및 POP HANDLE 명령 사용』에서 설명하는 대로 **HANDLE ABEND** 명령은 **PUSH HANDLE** 및 **POP HANDLE** 명령으로 일시중단할 수도 있습니다.

이상 종료가 처리되면 DTB(Dynamic Transaction Backout) 프로그램이 호출되지 않습니다. DTB(Dynamic Transaction Backout) 프로그램이 필요하면 내재적 또는 명시적 동기점으로 사용하거나 SYNCPOINT ROLLBACK을 실행 또는 **ABEND** 명령을 실행합니다.

IRC 연결 시스템에서 실행 중인 트랜잭션이 실패해서 이상 종료된 경우(예: AZI2) 동기점 처리는 백아웃 처리 중에 동일한 IRC 연결 사용이 시도되면 ASP1을 이상 종료할 수 있습니다.

HANDLE ABEND 명령은 ASPx 또는 APSJ 이상 종료 코드를 인터셉트할 수 없습니다.

이 섹션에서는 다음에 대해 설명합니다.

- 『프로그램 레벨 이상 종료 프로그램 또는 루틴 작성』
- 446 페이지의 『조작 재시도』
- 447 페이지의 『추적』
- 449 페이지의 『모니터링 애플리케이션 성능』
- 449 페이지의 『덤프』

프로그램 레벨 이상 종료 프로그램 또는 루틴 작성

프로그램 레벨 이상 종료 프로그램을 작성할 수 있습니다. 일부 프로그래밍 언어의 경우 프로그램 레벨 이상 종료 루틴을 작성할 수 있습니다.

프로그램 레벨 이상 종료 프로그램

RDO를 사용하거나 프로그램 자동 설치 엑시트를 사용하여 이상 종료 프로그램을 정의할 수 있습니다.

자동 설치 메소드를 사용하는 경우, **HANDLE ABEND** 시간에 프로그램 정의를 사용할 수 없습니다. 이것은 프로그램이 처음 호출될 때 다르게 기능함을 의미할 수 있습니다. **HANDLE ABEND**가 발행될 때 프로그램이 정의되지 않았고 프로그램 자동 설치가 활성화된 경우, 프로그램의 이름에 대한 보안 검사가 발생하는 유일한 검사입니다. 기타 검사는 이상 종료 프로그램이 호출될 때 발생합니다. 자동 설치가 실패하는 경우 태스크는 APCT를 이상 종료하고 제어가 다음 상위 레벨로 전달됩니다.

이상 종료 엑시트 프로그램은 임의의 지원되는 언어로 코딩할 수 있습니다.

이상 종료 엑시트 프로그램에 대한 입력 시에, 해당 언어로 코딩되는 모든 애플리케이션에 대해 정상적으로 지정된 것과는 다른 주소 지정 가능성을 가정할 수 없습니다.

프로그램 레벨 이상 종료 루틴

제한사항: 이 섹션은 **HANDLE ABEND LABEL** 명령이 지원되지 않기 때문에 C, C++, PL/I, AMODE(64) 애플리케이션에 적용되지 않습니다.

이상 종료 엑시트 루틴은 해당 프로그램과 동일한 언어로 코딩되어야 합니다.

이상 종료 엑시트 루틴의 경우, 주소 지정 모드 및 실행 키가 **HANDLE ABEND** 명령이 발행된 주소 지정 모드 및 실행 키로 설정됩니다.

이상 종료 엑시트 루틴에 대한 입력 시에 레지스터 값은 다음과 같습니다.

COBOL

레지스터가 복원되어 **HANDLE ABEND** 명령으로 제어가 리턴됩니다. 그런 다음 COBOL GOTO가 실행됩니다.

어셈블러 (**AMODE(64)** 어셈블러는 아님)

Reg 15

이상 종료 레이블.

Reg 0 - 14

마지막 CICS 서비스 요청 시의 내용.

처리의 종료

다음 방법 중 하나로 이상 종료 엑시트 루틴 또는 프로그램에서 처리를 종료할 수 있습니다. 이상 종료 루틴 및 프로그램이 CICS 내부 로직에 의해 호출될 때, 추가 처리는 더 많은 문제점을 유발하기 때문에 이상 종료로 종료해야 합니다.

- 태스크가 다음 상위 논리 레벨의 프로그램으로 전달되는 제어를 갖고 계속 실행함으로써 표시하려면 **RETURN** 명령을 사용하십시오. 그런 프로그램이 없는 경우, 태스크는 정상적으로 종료되고 모든 복구 가능 자원이 확보됩니다.
- 상위 논리 레벨의 프로그램에 대해 지정된 이상 종료 엑시트 또는 하나가 아닌 경우 비정상 종료 처리를 위한 비정상 조건 프로그램으로 제어가 전달되어 태스크가 비정상적으로 종료됨을 표시하려면 **ABEND** 명령을 사용하십시오.
- 조작을 재시도하기 위한 분기. 이 방법을 사용하여 조작을 재시도하고, 두 번째 실패가 발생하는 경우 원래 이상 종료 엑시트 루틴 또는 프로그램을 재진입하기 원할 때, 이상 종료 엑시트 루틴 또는 프로그램이 분기하기 전에 **HANDLE ABEND RESET** 명령을 실행해야 합니다. 이것은 CICS가 엑시트 루틴 또는 프로그램이 이상 종료 엑시트에 다시 진입하지 못하도록 엑시트 루틴을 사용 불가능하게 했기 때문입니다.

미해결 **RECEIVE** 명령의 제한시간 초과에 의해 유발되는 이상 종료의 경우, CICS가 **RECEIVE**를 취소할 수 있도록 CICS 이상 종료에 계속하도록 하는 것이 중요합니다.

조작 재시도

CICS 서비스 호출 중에 이상 종료 발생 시, 동일한 서비스에 대한 추가 요청을 발행하면 예기치 않은 결과가 유발될 수 있습니다. 포인터 및 작업 영역의 재초기화 때문에, 엑시트 루틴의 저장영역의 비우기가 완료되지 않았을 수 있습니다.

추가 I/O 조작을 시도하여 ATNI 또는 ATND 이상 종료로부터 복구하려고 시도하지 않아야 합니다. 이들 이상 종료 중 하나는 TERMERR 상태가 되는데, 모든 경우에 세션이 종료되어야 합니다. CICS가 RTIMOUT로부터 완전히 정리하지 않았고 무한대 대기 발생할 수 있으므로 AZCT 이상 종료 또는 AZIG 이상 종료로부터 복구할 때 터미널 제어 명령을 실행하려고 시도하지 않아야 합니다.

상호 시스템 통신이 사용되고 있는 경우, 원격 시스템의 이상 종료(종료)가 지정된 프로그램 또는 레이블에 대한 분기를 유발할 수 있지만, 원격 시스템의 동일한 자원을 사용하려는 후속 요청이 실패할 수 있습니다. 이상 종료(종료)가 원격 시스템에 대한 연결에서의 실패 때문에 발생하는 경우, 원격 시스템의 임의의 자원을 사용하려는 후속 요청이 실패할 수 있습니다.

이상 종료(종료)가 BMS 명령의 결과로 발생하는 경우, 제어 블록은 제어가 BMS 프로그램으로 리턴되기 전에 정리되지 않으며 명령이 재시도되는 경우 결과는 예측 불가능합니다.

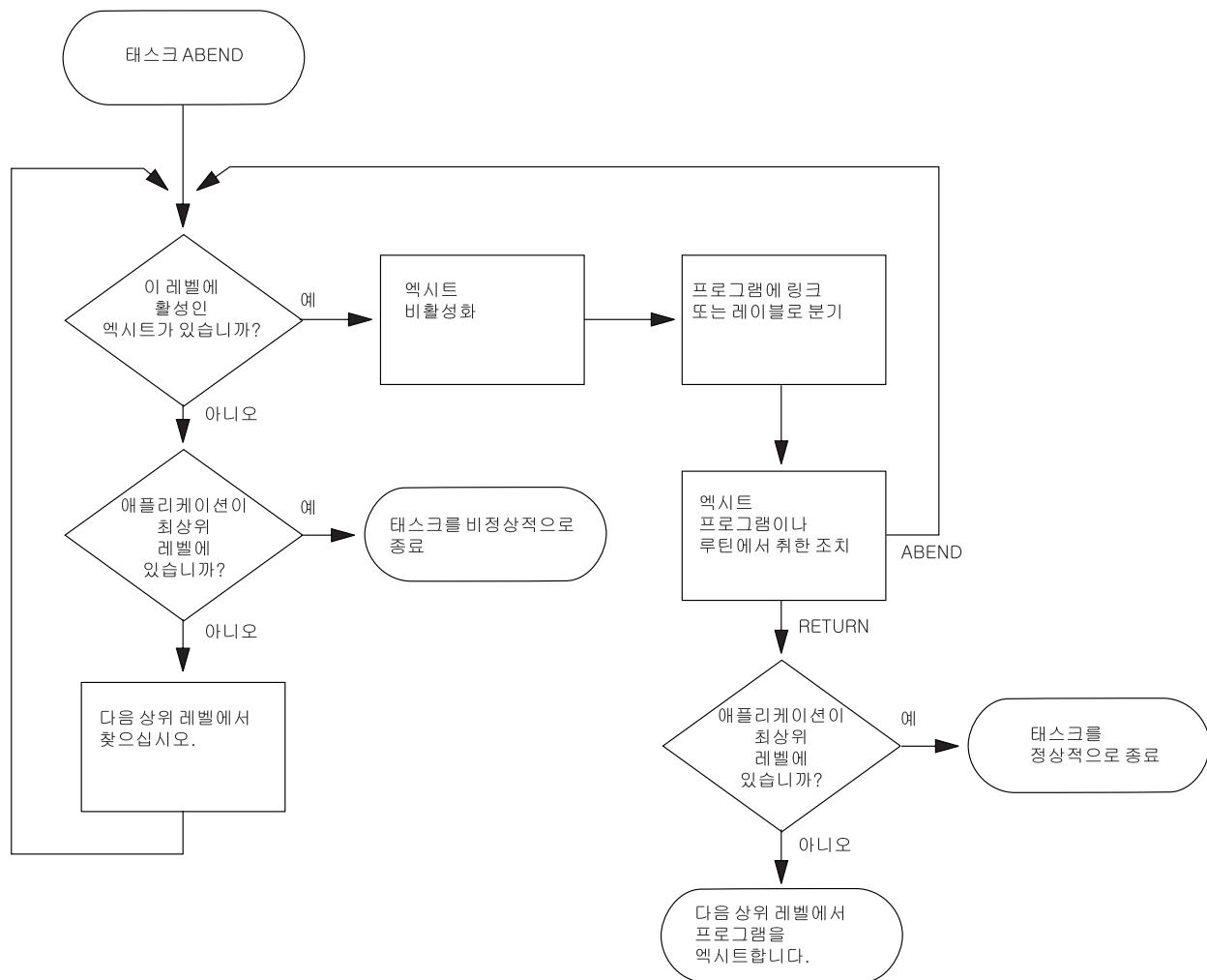


그림 111. ABEND 엑시트 처리

추적

CICS 추적은 애플리케이션 프로그래머, 시스템 프로그래머 및 IBM 필드 엔지니어를 위한 디버깅 지원 도구입니다. 추적 명령에 대한 응답으로 추적 항목을 생성합니다.

추적 항목은 현재 활성 상태인 추적 대상으로 전송할 수 있습니다. 대상은 다음과 같습니다.

- 내부 추적 테이블
- 보조 추적 데이터 세트
- 범용 추적 기능(GTF) 데이터 세트

추적 대상에 대한 정보는 *CICS 문제점 판별 안내서*에서 문제점 판별 시 추적 사용을 참조하십시오.

다음은 수행할 수 있습니다.

- 사용자 추적 시작점을 지정합니다(ENTER TRACENUM).
- SET TRACEDEST, SET TRACEFLAG 및 SET TRACETYPE 명령을 사용하여 CICS 내부 추적을 켜거나 끌 수 있습니다.

추적 시작점

추적 항목이 CICS 운영 중에 생성되는 지점은 네 개의 유형이 있습니다. 시스템 추적 시작점, 사용자 추적 시작점, 예외 추적 시작점 및 사용자 예외 추적 시작점입니다.

애플리케이션 코딩에는 영향을 주지 않는 이러한 기능을 설명하는 추적에 대한 자세한 정보는 *CICS 문제점 판별 안내서*에 있는 문제점 판별에서 추적 및 덤프 사용을 참조하십시오.

시스템 추적 시작점

다음은 추적 제어 요청이 작성되는 CICS 내의 위치입니다. 애플리케이션 프로그래머를 위한 가장 중요한 시스템 추적 시작점은 EXEC 인터페이스 프로그램을 위한 것입니다. 이들은 CICS 명령이 처리될 때마다 추적 테이블에 항목을 생성합니다.

두 개의 추적 항목이 생성됩니다. 첫 번째는 명령이 실행될 때이고, 두 번째는 CICS가 필수 기능을 수행했고 애플리케이션에 제어를 리턴하려고 할 때입니다. 이들 사이에서 다음 두 개의 추적 항목을 사용하면 애플리케이션 전체에서 제어의 플로우를 추적할 수 있고 실행 중에 예외 조건이 있는 경우 어떤 예외 조건이 발생했는지 확인할 수 있습니다. ABEND, RETURN, TRACEFLAG 및 XCTL 명령은 단일 항목만을 생성합니다.

사용자 추적 시작점

이는 애플리케이션 내에서 완전한 프로그램 디버깅을 허용하기 위해 포함할 수 있는 추가적인 위치입니다. 예를 들어, 루프가 입력된 횟수를 보여주는 카운터 값을 포함하는 프로그램 루프에 항목을 지정할 수 있습니다.

추적 항목은 ENTER TRACENUM 명령이 실행되는 곳마다 생성됩니다. 고유 ID가 지정될 수 있는 각 추적 항목 요청은 데이터를 추적 테이블에 배치하게 합니다.

예외 추적 시작점

이는 CICS가 예외 조건을 발견한 추가적인 위치입니다. 이는 CICS 코드의 특정 위치에서 생기고, 데이터는 원인에 대한 일부 정보를 제공할 수도 있는 영역에서 구합니다. 예외 추적 시작점은 연관된 『레벨』 속성이 없습니다. 추적 호출은 예외 조건이 발생할 때에만 작성됩니다.

사용자 예외 추적 시작점

이는 항상 내부 추적 테이블(내부 추적이 꺼졌더라도)에 작성되지만 활성화된 경우에만 다른 대상에 작성되는 추적 항목입니다. CICS 유틸리티 프로그램이 생성하는 형식화된 추적 출력으로 된 문자열 *EXCU로 식별할 수 있습니다. 사용자 예외 추적 시작점에 대한 일반 정보는 CICS 문제점 판별 안내서에 있는 사용자 예외 추적 시작점을 참조하십시오. 프로그래밍 정보는 CICS 사용자 정의 안내서에 있는 추적 항목 작성에 있습니다.

모니터링 애플리케이션 성능

CICS 모니터링은 애플리케이션 트랜잭션의 성능에 대한 정보를 제공합니다. 사용자 이벤트 모니터링 위치에 대해 MONITOR 명령을 사용하십시오.

다른 곳에 정의된 시스템에서 수집된 모니터링 데이터, CICS의 모니터링 위치(EMP) 외에도, 사용자 애플리케이션은 CICS 모니터링 레코드의 사용자 필드로 데이터를 컨트리뷰션할 수 있습니다. 사용자 정의 EMP를 호출하기 위해 MONITOR POINT 명령을 사용하여 이를 수행할 수 있습니다. 이 EMP 각각에서, 각 성능 모니터링 레코드의 사용자 소유 데이터에서 최고 16384바이트까지 추가하거나 변경할 수 있습니다. 해당되는 16384바이트에서, 다음 조합을 수반할 수 있습니다.

- 범위 0 - 256 카운터에서
- 범위 0 - 256 클럭에서
- 단일 8192바이트 문자열

예를 들어, 이 사용자 EMP를 사용하여 특정 이벤트가 발생하는 횟수를 계산하거나, 두 이벤트 사이의 간격 시간을 지정할 수 있습니다.

덤프

CICS 덤프는 디스크 또는 테이프에 있을 수 있는 순차 데이터 세트로, DUMP TRANSACTION 명령에 의해 덤프될 기본 저장영역의 영역을 지정할 수 있도록 합니다.

.

PERFORM DUMP 명령은 시스템 덤프를 요청할 수 있도록 허용합니다. 이 명령에 대한 프로그래밍 정보는 *CICS 시스템 프로그래밍* 참조서에 있는 **PERFORM DUMP**를 참조하십시오.

덤프 데이터 세트의 내용을 형식화하고, 시스템 덤프에 대한 대화식 문제점 제어 시스템(IPCS)이나 트랜잭션 덤프에 대한 CICS 덤프 유틸리티 프로그램(DFHDU700)을 사용하여 오프라인에서 인쇄할 수 있습니다. 이러한 문제점 사용에 대한 지시사항은 *CICS 운영 및 유틸리티 안내서*에서 덤프 유틸리티 프로그램을 참조하십시오.

한 번에 단 하나의 덤프 제어 명령만 처리됩니다. 다른 태스크가 트랜잭션 덤프를 사용하는 동안 추가 덤프 제어 명령을 실행하면, 덤프가 완료될 때까지 해당 명령과 연관된 태스크 내의 활동은 일시중단됩니다. 나머지 덤프 명령은 명령이 작성된 순서대로 처리됩니다. **DUMP TRANSACTION** 명령을 사용하면 EIB 및 TCA의 일부 필드(예: EIBFN 및 EIBRCODE)가 겹쳐쓰여집니다.

DUMP TRANSACTION 명령의 옵션은 다양한 조합으로 기본 저장영역의 다음 영역을 덤프할 수 있도록 합니다.

- 태스크 관련 저장영역: 요청하는 태스크에 관련되는 선택된 기본 저장영역. 애플리케이션을 테스트하고 디버깅하기 위해 정상적으로 이러한 영역의 덤프를 사용합니다. (관련된 태스크가 비정상적으로 종료되면 CICS는 자동으로 이 서비스를 제공합니다.)
- CICS 제어 테이블: CICS 시스템에서 FILE, PROGRAM, TRANSACTION 및 TERMINAL 자원 정의와 CICS 시스템 초기화 매개변수의 목록. 이러한 테이블의 덤프는 일반적으로 테스트의 기본이 설정되어야 하는 테스트에서 취해지는 첫 번째 덤프입니다. 후속 덤프는 보통 태스크 관련 저장영역 유형입니다.
- 이는 간혹 태스크 관련 저장영역 및 CICS 제어 테이블 둘 다의 덤프를 갖도록 하기 위해 태스크 실행 동안 적절합니다. 하나의 CICS 제어 테이블 덤프와 여러 태스크 관련 저장영역 덤프를 지정하는 것은 일반적으로 비슷한 수의 전체 덤프를 지정하는 것보다 더 효율적입니다. 그러나 CICS 제어 테이블은 기본적으로 정적 영역이므로, 과도하게 이 기능을 사용하면 안됩니다.
- 또한 세 개의 옵션 SEGMENTLIST, LENGTHLIST 및 NUMSEGMENTS와 함께 사용되는 **DUMP TRANSACTION** 명령은 일련의 태스크 관련 저장영역을 동시에 덤프할 수 있도록 허용합니다.

프로그램 저장영역은 **RELOAD(YES)** 속성으로 정의된 프로그램에 대해 덤프되지 않습니다.

또한 인쇄된 덤프의 끝에서, 주소에 의해 색인화된 CICS 핵심 모듈 및 활성 프로그램 목록도 가져옵니다.

QUERY SECURITY 명령

EXEC CICS QUERY SECURITY 명령을 사용하여 터미널 사용자가 외부 보안 관리자에 정의된 자원에 액세스할 수 있는지 여부를 판별할 수 있습니다.

QUERY SECURITY 명령을 RACF® 또는 동등한 외부 보안 관리자(ESM)와 함께 사용할 수 있습니다. 이 컨텍스트의 터미널 사용자는 **QUERY SECURITY** 명령을 포함하는 트랜잭션을 호출하는 사용자입니다.

QUERY SECURITY 명령에 대한 응답에서 CICS는 터미널 사용자의 보안 권한에 관한 정보를 리턴합니다. CICS는 외부 보안 관리자로부터 이 정보를 얻습니다. 사용자의 허용되는 액세스 권한에 따라서 여러 가지 방법으로 진행하도록 애플리케이션을 코딩할 수 있습니다.

CICS 자원 유형 이름에 의해 조회 중인 자원의 유형을 지정합니다. 예를 들어, 파일에 액세스하기 위한 사용자 권한을 조회하려는 경우 **RESTYPE('FILE')**을 지정할 수 있습니다. 유형 안에서 특정 파일을 식별하려면 **RESID** 매개변수를 지정합니다.

QUERY SECURITY 명령 사용

이 주제는 일반적인 **QUERY SECURITY** 명령의 사용을 다룹니다. 그 중 하나는 메뉴에서 트랜잭션 코드를 표시하기 전에 사용자가 특정 트랜잭션을 사용할 권한이 있는지 여부를 확인하는 것입니다.

레코드 또는 필드 레벨에서의 보안 보호

QUERY SECURITY의 또 다른 사용은 사용자가 레코드 또는 필드 레벨에서 데이터에 대한 액세스를 제어할 수 있게 하는 것입니다. 예를 들어 파일 자원에 대한 일반 CICS 자원 보안 검사는 파일 레벨에서만 작업합니다. 개별 레코드 또는 심지어 레코드 내의 필드에 대한 액세스를 제어하기 위해 **QUERY SECURITY**를 사용할 수 있습니다. 이를 위해서 보안 관리자는 보호하려는 레코드나 필드에 대해 적절한 액세스 권한을 갖고 자원 프로파일 이름을 정의해야 합니다. 이들 프로파일은 CICS 자원 클래스에서 *아니라* 관리자가 정의하는 사용자 자원 클래스에서 정의됩니다.

이들 클래스 및 자원을 조회하기 위해 **QUERY SECURITY** 명령은 **RESCLASS** 및 **RESID** 옵션을 사용합니다. (**RESCLASS**와 **RESTYPE**은 상호 배타적인 옵션입니다.) **QUERY SECURITY**가 리턴하는 CVDA 값을 사용하여 레코드나 필드에 액세스할 수 있습니다.

CICS 정의 자원 ID

SPCOMMAND 자원 유형을 제외한 모든 경우에 자원 ID는 사용자 정의됩니다. 그러나 **SPCOMMAND** 유형의 경우 ID는 CICS에 의해 수정됩니다. *CICS RACF* 보안

안내서에 있는 RESID 값이 SPCOMMAND 자원 유형에 대한 가능한 RESID 값을 상세히 설명합니다.

SEC 시스템 초기화 매개변수

SEC 시스템 초기화 매개변수의 설정이 QUERY SECURITY 명령에 의해 리턴되는 CVDA 값에 영향을 줍니다. SEC 시스템 초기화 매개변수는 참조 -> 시스템 정의에서 SEC 시스템 초기화 매개변수.

프로그래밍 힌트

- 트랜잭션은 QUERY SECURITY 명령을 사용하여 터미널 사용자가 액세스할 수 있는 자원의 목록을 준비하기 위해 많은 자원을 조회할 수 있습니다. 이 기술을 사용하면 트랜잭션이 액세스할 권한이 부여되지 않은 자원에 대한 각 조회에 대해 최고 4개의 자원 위반 메시지가 생성될 수 있습니다. 이들 메시지는 시스템 콘솔, CICS TD 큐, SMF 로그 데이터 세트에 나타납니다. 이들 메시지를 비활성화하려는 경우 QUERY SECURITY 명령에서 NOLOG를 코딩하십시오.
- 트랜잭션이 한 번의 실행에서 동일한 자원에 여러 번 액세스하는 경우, 트랜잭션 자원 정의에서 RESSEC(NO)와 함께 트랜잭션을 정의하여 성능을 개선할 수 있습니다. 그런 다음 단일 QUERY SECURITY 명령을 실행하고 리턴되는 CVDA 값에 따라서 자원에 대한 액세스를 허용하도록 트랜잭션을 코딩할 수 있습니다. 상세한 안내에 대해서는 CICS RACF 보안 안내서에 있는 QUERY SECURITY 명령을 사용한 보안 검사를 참조하십시오.

CICS 상호 통신

다른 CICS 시스템과 통신하는 애플리케이션을 작성할 때 고려해야 하는 영역에 대해 요약합니다.

CICS 상호 통신에 대한 자세한 정보는 시작하기에서 상호 통신 시작하기의 내용을 참조하십시오.

다음 기능 중 하나 이상을 사용하여 CICS 상호 통신 환경에서 애플리케이션을 실행할 수 있습니다.

트랜잭션 라우팅

트랜잭션 라우팅을 사용하면 임의 CICS 시스템의 터미널이 다른 CICS 시스템의 트랜잭션을 실행할 수 있습니다. 454 페이지의 『트랜잭션 라우팅』의 내용을 참조하십시오.

Function shipping

Function Shipping을 사용하면 애플리케이션이 다른 CICS 시스템의 자원에 액세스할 수 있습니다. 454 페이지의 『Function shipping』의 내용을 참조하십시오.

DPL(Distributed Program Link)

DPL을 사용하면 임의의 CICS 리전에서 실행 중인 애플리케이션이 원격 CICS 리전에서 실행 중인 다른 애플리케이션에 링크할 수 있습니다. 455 페이지의 『DPL(Distributed Program Link)』의 내용을 참조하십시오.

비동기 처리

비동기 처리를 사용하면 CICS 트랜잭션이 원격 시스템에서 다른 트랜잭션을 시작하고 선택적으로 데이터를 전달할 수 있습니다. 468 페이지의 『비동기 처리』의 내용을 참조하십시오.

DTP(Distributed Transaction Processing)

DTP를 사용하면 CICS 트랜잭션이 다른 시스템에서 실행 중인 트랜잭션과 통신할 수 있습니다. DTP에 대해 다음과 같은 두 개의 인터페이스를 사용할 수 있습니다. 명령 레벨 EXEC CICS 및 DTP용 SAA 인터페이스(공통 프로그래밍 인터페이스(CPI 통신)라고도 함). 469 페이지의 『DTP(Distributed Transaction Processing)』의 내용을 참조하십시오.

공통 프로그래밍 인터페이스 통신(CPI-C)

CPI-C는 APPC 연결에서 DTP를 제공하고 다중 시스템 플랫폼에서 사용할 수 있는 API를 정의합니다. 469 페이지의 『공통 프로그래밍 인터페이스 통신(CPI 통신)』의 내용을 참조하십시오.

외부 CICS 인터페이스(EXCI)

EXCI를 사용하면 MVS에서 실행 중인 비CICS 프로그램이 세션을 CICS 시스템에 할당하고 세션을 열어 해당 세션에서 DPL 요청을 발행할 수 있습니다. CICS는 외부 CICS 인터페이스를 사용하는 애플리케이션에서 MVS 자원 복구 서비스(RRS)를 지원합니다. 470 페이지의 『외부 CICS 인터페이스(EXCI)』의 내용을 참조하십시오.

CICS FEPI(Front End Programming Interface)에 관련된 통신에 대한 자세한 내용은 제품 개요에서 FEPI 소개의 내용을 참조하십시오.

디자인 고려사항

애플리케이션이 이전에 나열된 기능 중 둘 이상을 사용하는 경우 각각에 대해 디자인 고려사항을 허용해야 합니다. 또한, 프로그램이 분산 트랜잭션 처리에 둘 이상의 상호 시스템 세션을 사용하는 경우, 해당 세션 유형에 대한 규칙에 따라 각 세션을 제어해야 합니다.

프로그래밍 언어

일반적으로 COBOL, C, C++, PL/I 또는 어셈블러 언어를 사용하여 CICS 상호 통신 기능을 사용하는 애플리케이션을 작성할 수 있습니다. 그렇지만 EXEC CICS API를

사용하여 APPC 맵핑되지 않은 대화를 보유하는 DTP 애플리케이션에 대해서는 C, C++ 또는 어셈블리 언어만 사용할 수 있습니다.

트랜잭션 라우팅

다른 CICS 시스템이 소유하는 터미널에서 호출될 수 있거나, 트랜잭션 초기 동안 다른 CICS 시스템이 소유하는 터미널을 획득할 수 있는 트랜잭션은 트랜잭션 라우팅 환경에서 실행할 수 있어야 합니다.

일반적으로, 로컬 환경에서 사용되는 것과 같은 트랜잭션을 디자인하고 코딩할 수 있습니다. 그러나 트랜잭션을 실행할 터미널, BMS(Basic Mapping Support) 및 의사 대화식 트랜잭션에 관련되는 몇 가지 제한사항이 있습니다. 트랜잭션에서 사용되는 모든 프로그램, 테이블 및 맵은 트랜잭션을 소유하는 시스템에 상주해야 합니다. (필요한 만큼 많은 시스템에서 중복할 수 있습니다.)

일부 CICS 트랜잭션은 서로 관련됩니다. 예를 들어, CWA에 대한 공통 액세스나 GETMAIN 명령을 사용하여 획득된 공유 저장영역을 통해 관련됩니다. 이러한 경우, 시스템 프로그래머는 이 트랜잭션이 동일한 CICS 시스템으로 라우팅되는지 확인해야 합니다. 동적 트랜잭션 라우팅을 수행할 수 있는 기능에 불리하게 영향을 줄 수 있는 트랜잭션간 연관관계를 작성할 수 있는 기술은 가능하면 피해야 합니다.

이러한 명령을 실행하는 프로그램의 잠재된 문제점을 쉽게 식별하기 위해 CICS 상호 종속 분석기를 사용할 수 있습니다. 이 유틸리티에 대한 자세한 정보는 CICS Interdependency Analyzer for z/OS의 내용을 참조하십시오. 트랜잭션 연관관계에 대한 자세한 정보는 379 페이지의 『연관관계』의 내용을 참조하십시오.

트랜잭션을 처리하기 위한 요청이 하나의 CICS 시스템에서 다른 CICS 시스템으로 전송될 때, 트랜잭션 ID는 로컬 이름에서 원격 이름으로 변환될 수 있습니다. 그러나 트랜잭션 소유 시스템에서 터미널 소유 시스템으로 전송될 때 RETURN 명령에 지정된 트랜잭션 ID는 변환되지 않습니다.

Function shipping

로컬 시스템에서와 거의 같은 방법으로 원격 시스템에서 자원에 액세스하기 위해 프로그램을 코딩합니다.

다음을 사용할 수 있습니다.

DL/I 호출(EXEC DLI 명령)

원격 CICS 시스템과 연관된 데이터에 액세스합니다.

파일 제어 명령

원격 시스템의 파일에 액세스합니다. TOKEN 키워드를 포함하는 요청은 function-ship될 수 없습니다.

임시 저장영역 명령

원격 시스템의 임시 저장영역 큐로부터 데이터에 액세스합니다.

트랜지언트 데이터 명령

원격 시스템의 트랜지언트 데이터 큐에 액세스합니다.

세 개의 추가 예외 조건은 원격 자원에 대해 발생할 수 있습니다. 이 상태는 원격 시스템이 사용 불가능한 경우에(SYSDERR), 요청이 올바르지 않은 경우(ISCINVREQ) 또는 미리 트랜잭션이 이상 종료하는 경우(IPIC에 대한 AIPM 이상 종료, ISC 연결에 대한 ATNI, MRO에 대한 AZI6) 발생합니다.

DPL(Distributed Program Link)

DPL(Distributed Program Link) 함수를 사용하면 CICS 프로그램(클라이언트 프로그램)이 원격 CICS 리전에서 다른 CICS 프로그램(서버 프로그램)을 호출할 수 있습니다.

DPL(Distributed Program Link)을 사용하도록 애플리케이션을 디자인할 수 있는 몇 가지 이유가 있습니다. 이들 중 일부는 다음과 같습니다.

- 데이터 액세스 및 처리와 같이, 애플리케이션 비즈니스 로직으로부터 일반 사용자 인터페이스(예: BMS 화면 처리)를 구분하고, 애플리케이션의 일부가 호스트에서 워크스테이션으로 더 즉각적으로 이식될 수 있도록 하기 위해.
- 실행 중인 프로그램이 액세스하는 자원에 더 근접하여 성능 이점을 얻어서, 반복되는 Function Shipping 요청에 대한 필요성을 줄이기 위해.
- 많은 경우에, DTP(Distributed Transaction Processing) 애플리케이션을 작성하기 위한 간단한 대안을 제공하기 위해.

애플리케이션이 링크 중인 프로그램이 원격임을 지정할 수 있는 몇 가지 방법이 있습니다.

1. LINK 명령에서 원격 시스템 이름을 지정하여
2. 설치된 프로그램 자원 정의에서 원격 시스템 이름을 지정하여
3. 동적 라우팅 프로그램을 사용하는 원격 시스템 이름을 지정하여(설치된 프로그램 정의에 DYNAMIC(YES)이 지정되거나 설치된 프로그램 정의가 없는 경우)
4. XPCREQ 글로벌 사용자 엑시트에서 원격 시스템 이름을 지정하여

DPL(Distributed Program Link)의 기본 플로우 는 CICS DPL(Distributed Program Link)에서 설명됩니다. 456 페이지의 그림 112에 설명된 다음 용어는 DPL(Distributed Program Link)에 대한 설명에서 사용됩니다.

클라이언트 리전

다른 CICS 리전에서 프로그램에 대한 링크를 발행하는 애플리케이션을 실행 중인 CICS 리전.

서버 리전

클라이언트 리전이 링크 요청을 전달(ship)할 CICS 리전.

클라이언트 프로그램

원격 링크 요청을 발행하는 애플리케이션.

서버 프로그램

서버 리전에서 실행되는, 링크 요청에 지정된 애플리케이션.

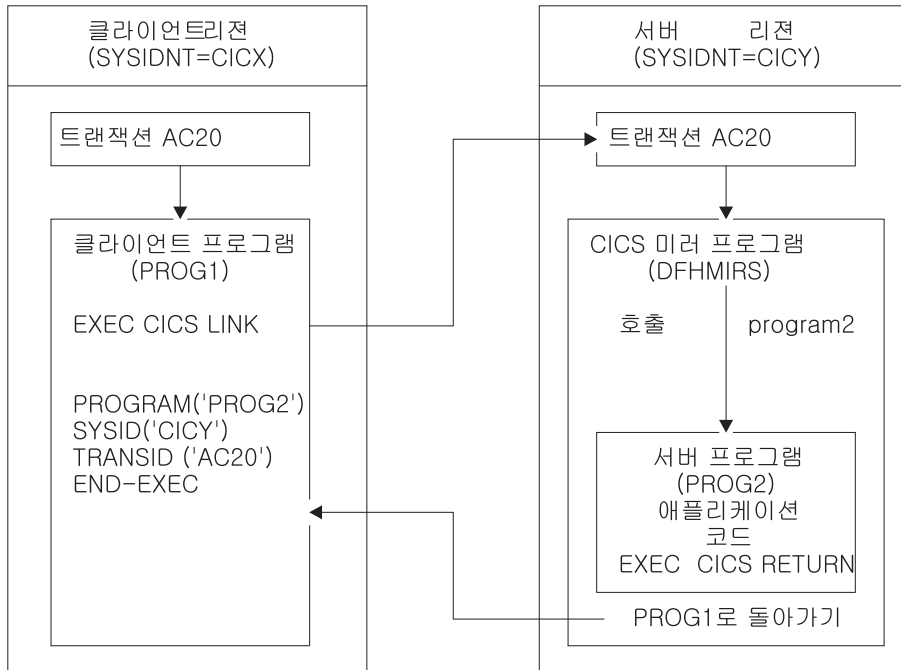


그림 112. DPL(Distributed Program Link)의 그림

DPL(Distributed Program Link) 기능 사용

다음 테이블을 사용하여 DPL(Distributed Program Link) 기능의 옵션에 대해 이해합니다.

다음을 지정할 수 있습니다.

- 원격 시스템 이름(서버 리전).
- 서버 리전에서 다른 이름으로 인식되는 경우 서버 프로그램 이름.
- 링크된 프로그램을 로컬로 실행하려고 하지만 이를 테스트하기 위해 API(Application Programming Interface)의 DPL(Distributed Program Link) 서브세트로 제한하는 경우. (서버 프로그램은 원격으로 실행되는 경우 전체 CICS API를 사용할 수 없으며 제한사항은 467 페이지의 표 28에 나열됩니다.)
- 서버 프로그램이 클라이언트에서 동기점을 독립적으로 가져오는 경우.

- 프로그램이 서버 리전에서 실행되도록 하려는 트랜잭션 이름.
- 전달 중인 COMMAREA의 데이터 길이.

서버 프로그램은 자체적으로 DPL(Distributed Program Link)을 발행하고 링크하는 프로그램에 대해 클라이언트 프로그램으로 수행됩니다.

표 26에 표시되는 옵션은 LINK 명령에서 사용되며 표 27에 표시되는 옵션은 DPL(Distributed Program Link) 기능을 지원하기 위해 PROGRAM 자원 정의에서 사용됩니다.

표 26. DPL을 지원하기 위한 LINK 명령의 옵션

키워드	설명
DATALENGTH	애플리케이션이 서버 프로그램으로 전송 중인 저장영역 (COMMAREA 시작에서)의 인접 영역 길이를 지정합니다.
LENGTH	COMMAREA의 길이(바이트)인 반자 2진 값을 지정합니다. LENGTH 옵션에 대해 올바른 값(0이 아님)을 지정하십시오. 그 이유는 0의 길이가 DPL 요청에서 사용되면 예상치 못한 결과가 초래되어 결국 프로그램이 실패할 수 있기 때문입니다.
SYSID	클라이언트 리전이 프로그램 링크 요청을 전송하는 서버 리전의 연결 이름을 지정합니다. 참고: LINK 명령에 지정된 원격 SYSID는 프로그램 자원 정의 또는 동적 라우팅 프로그램에서 리턴된 sysid에 지정되는 REMOTESYSTEM 이름을 덮어씁니다.
SYNCONRETURN	서버 리전이 서버 프로그램이 성공적으로 완료될 때 동기점을 사용하도록 지정합니다. 참고: 이 옵션은 LINK 명령에 고유하며 프로그램 자원 정의에는 지정할 수 없습니다.
TRANSID	서버 리전이 서버 프로그램 실행용으로 접속하는 트랜잭션 이름을 지정합니다. 참고: LINK 명령에 지정된 TRANSID는 프로그램 자원 정의에 지정되는 모든 TRANSID를 덮어씁니다.

참고: LINK 명령의 전체 구문을 포함하는 프로그래밍 정보는 CICS 애플리케이션 프로그래밍 참조서에 있지만 DPL(Distributed Program Link)에 대해서는 INPUTMSG 또는 INPUTMSGLEN 옵션을 지정할 수 없음에 주의하십시오.

표 27. DPL을 지원하는 PROGRAM 자원 정의의 옵션

키워드	설명
REMOTESYSTEM	클라이언트 리전이 프로그램 링크 요청을 전송하는 서버 리전(SYSID)의 연결 이름을 지정합니다.
REMOTENAME	프로그램이 서버 리전에서 인식되는 이름을 지정합니다(로컬 이름과 다른 경우).

표 27. DPL을 지원하는 PROGRAM 자원 정의의 옵션 (계속)

키워드	설명
DYNAMIC	프로그램 링크 요청이 동적으로 라우팅되는지 여부를 지정합니다. DPL 요청에 대한 동적 라우팅에 대한 자세한 정보는 CICS 상호 통신 안내서를 참조하십시오.
EXECUTIONSET	프로그램이 CICS API의 DPL(Distributed Program Link) 서브세트로 제한되는지 여부를 지정합니다. 참고: 이 옵션은 프로그램 정의에 고유하며 LINK 명령에는 지정할 수 없습니다.
TRANSID	서버 리전이 서버 프로그램 실행용으로 접속하는 트랜잭션 이름을 지정합니다.

DPL(Distributed Program Link)의 예제

DPL(Distributed Program Link) 명령의 COBOL 예제는 명령의 구문을 표시합니다.

중요사항: LINK 명령의 SYSID 옵션이 원격 리전의 이름을 지정하면, 프로그램 정의에 지정되었거나 동적 라우팅 프로그램에 의해 리턴된 REMOTESYSTEM, REMOTENAME 또는 TRANSID 속성은 아무런 영향이 없습니다.

```
EXEC CICS LINK PROGRAM('DPLPROG')
      COMMAREA(DPLPRO-Data-Area)
      LENGTH(24000)
      DATALENGTH(100)
      SYSID('CICR')
      TRANSID('AC20')
      SYNCONRETURN
```

1
2
2
2
3
4
5

그림 113. DPL(Distributed Program Link)의 COBOL 예제

1. 서버 프로그램의 프로그램 이름.

프로그램은 클라이언트와 서버 리전에 서로 다른 이름이 있을 수 있습니다. LINK 명령에 지정하는 이름은 SYSID 옵션을 지정하는지 여부에 달려 있습니다.

LINK 명령의 SYSID 옵션에 원격 리전의 이름을 지정하면 CICS는 클라이언트 리전에 있는 프로그램 자원 정의의 REMOTENAME 속성이나 동적 라우팅 프로그램이 리턴하는 프로그램 이름에 대한 참조 없이 링크 요청을 서버 리전으로 보냅니다. 이 상황에서 LINK 명령에 지정하는 PROGRAM 이름은 프로그램이 서버 리전에 알려진 이름이어야 합니다.

LINK 명령에 SYSID 옵션을 지정하지 않거나 로컬 클라이언트 리전의 이름을 지정하는 경우에는, LINK 명령에 지정하는 PROGRAM 이름은 프로그램이 클라이언트 리전에 알려진 이름이어야 합니다. CICS는 클라이언트 리전에서 프로그램 자원 정의를 찾아봅니다. 설치된 프로그램 정의의 REMOTESYSTEM 옵션은 원격 리전의 이름을 지정한다고 가정할 때 원격 리전에서 서버 프로그램의 이름은 다음에서 구합니다.

- a. 프로그램 정의의 REMOTENAME 속성
- b. REMOTENAME이 지정되지 않은 경우 LINK 명령의 PROGRAM 옵션.

프로그램 정의가 DYNAMIC(YES)을 지정하거나, 설치된 프로그램 정의가 없으면, 동적 라우팅 프로그램이 호출되고 서버 프로그램의 이름을 승인하거나 변경할 수 있습니다.

2. 통신 데이터 영역(COMMAREA).

성능을 개선하려면 LINK 명령에 DATALENGTH 옵션을 지정할 수 있습니다. 이 옵션을 사용하여 클라이언트 리전이 서버 프로그램에 전달하려는 COMMAREA 데이터의 양을 지정할 수 있습니다. 일반적으로 큰 COMMAREA가 서버 프로그램이 클라이언트 프로그램에 리턴하려는 데이터를 보유해야 할 때 이 옵션을 사용하지만 이 예제에서처럼 소량의 데이터만이 클라이언트 프로그램에 의해 서버 프로그램으로 전송되어야 합니다.

둘 이상의 서버 프로그램이 동일한 COMMAREA를 클라이언트 프로그램에 다시 전달하기 전에 이를 갱신하면 DATALENGTH 옵션을 사용하여 COMMAREA의 길이를 지정하십시오. 서버 프로그램이 XCTL 명령을 사용하여 COMMAREA를 다음 서버 프로그램으로 전달하는 경우 이에 대한 동일한 길이와 주소를 지정하는지 확인하십시오. 그러면 원본 COMMAREA가 클라이언트 프로그램으로 돌아갑니다. 다른 길이나 주소가 지정되면 호출되는 프로그램은 원본 COMMAREA가 아니라 사본 COMMAREA를 받게 되므로 원본 COMMAREA는 클라이언트 프로그램으로 돌아가지 않습니다. 367 페이지의 『COMMAREA』에는 COMMAREA를 사용하여 데이터를 다른 프로그램으로 전달하는 방법에 대한 더 많은 정보가 있습니다.

LENGTH 옵션에 올바른 값(0이 아님)을 지정했는지 확인하십시오. 0 길이는 DPL 요청에 사용되면 예측할 수 없는 결과를 내고 프로그램 장애를 일으킬 수 있기 때문입니다.

COMMAREA를 사용할 때 링크되는 프로그램은 태스크의 EIB에 있는 EIBCALEN 필드와 프로그램이 예상하는 것과 일치하는지 확인해야 합니다. 자세한 정보는 367 페이지의 『COMMAREA』의 내용을 참조하십시오.

3. 원격 시스템 ID(SYSID).

애플리케이션 리전이 다음 중 하나를 사용하여 프로그램 링크 요청을 전달할 서버 리전의 4문자 이름을 지정할 수 있습니다.

- LINK 명령의 SYSID 옵션
- 프로그램 자원 정의의 REMOTESYSTEM 옵션
- 동적 라우팅 프로그램.

우선순위 규칙은 다음과 같습니다.

- a. EXEC CICS LINK 명령의 SYSID 옵션이 원격 CICS 리전을 지정하면 CICS는 요청을 원격 리전으로 전달합니다.

프로그램 정의가 DYNAMIC(YES)을 지정하거나 프로그램 정의가 없으면(동적 라우팅 프로그램이 알림 용도로만 호출됨) 요청을 경로 재지정할 수 없습니다.

- b. SYSID 옵션이 지정되지 않았거나 로컬 CICS 리전과 동일한 이름을 지정하는 경우

- 1) 프로그램 정의가 DYNAMIC(YES)을 지정하거나 설치된 프로그램 정의가 없는 경우 동적 라우팅 프로그램이 호출되고 요청을 라우팅할 수 있습니다.

프로그램 정의의 REMOTESYSTEM 옵션은 지정된 경우, 동적 라우팅 프로그램에 전달되는 기본 서버 리전의 이름을 지정합니다.

참고: REMOTESYSTEM 옵션이 원격 리전의 이름을 지정하면 동적 라우팅 프로그램은 요청을 로컬로 라우팅할 수 없습니다.

- 2) 프로그램 정의가 DYNAMIC(NO)을 지정하면, CICS는 요청을 REMOTESYSTEM 옵션에 이름 지정된 원격 시스템으로 전달합니다. REMOTESYSTEM이 지정되지 않으면 CICS는 프로그램을 로컬로 실행합니다.

사용자가 지정하는 이름은 서버 리전과의 연결을 정의하는 클라이언트 리전에 설치된 연결 정의의 이름입니다. (CICS는 테이블 검색에서 연결 이름을 사용하여 서버 리전의 네트이름(z/OS Communications Server APPLID)을 구합니다.) 사용자가 지정하는 서버 리전의 이름은 클라이언트 리전의 이름일 수 있습니다. 이 경우 프로그램은 로컬로 실행됩니다.

서버 리전이 요청된 프로그램(이 예제에서는 DPLPROG)을 로드하거나 실행할 수 없는 경우 CICS는 링크 요청에 대한 응답으로 PGMIDERR 조건을 클라이언트 프로그램으로 되돌립니다. EIBRESP2 값은 서버 리전에서 오류가 발견된 DPL(Distributed Program Link) 요청으로 인해 링크를 통해 리턴되지 않음을 유의하십시오. 클라이언트 리전에서 발견된 오류로 인해 EIBRESP2 값이 리턴됩니다.

XPCREQ 글로벌 사용자 엑시트 프로그램에서 서버 리전의 이름을 지정하거나 수정할 수도 있습니다. XPCREQ 글로벌 사용자 엑시트 위치에 대한 프로그래밍 정보는 CICS 사용자 정의 안내서에서 Enabling for specific invocation types 의 내용을 참조하십시오.

4. 첨부할 원격 트랜잭션(TRANSID).

TRANSID 옵션은 LINK 명령과 프로그램 자원 정의 둘 모두에서 사용 가능합니다. 이를 사용하면 서버 리전에 서버 프로그램이 실행되는 미리 태스크를 첨부할 때 사용할 트랜잭션 ID를 알려줄 수 있습니다. TRANSID 옵션을 지정하는 경우에는 서버 리전에 트랜잭션을 정의하고 이를 제공된 미리 프로그램 DFHMIRS와 연관시

켜야 합니다. 이 옵션을 사용하면 성능과 미세 조정을 위해 트랜잭션 정의에 사용자 고유의 속성을 지정할 수 있습니다. 예를 들어, 태스크 우선순위와 트랜잭션 클래스 속성을 변경할 수 있습니다.

설치된 프로그램 정의가 DYNAMIC(YES)을 지정하거나 설치된 프로그램 정의가 없으면 동적 라우팅 프로그램이 호출되고, LINK 명령의 SYSID 옵션이 원격 리전을 이름 지정하지 않는 경우 TRANSID 속성의 값을 변경할 수 있습니다.

우선순위의 순서는 다음과 같습니다.

- a. LINK 명령의 SYSID 옵션이 원격 리전을 지정한 경우 LINK에 제공된 TRANSID
- b. 동적 라우팅 프로그램이 제공하는 TRANSID
- c. LINK 명령에 제공된 TRANSID
- d. 프로그램 정의의 TRANSID 속성
- e. 미리 TRANSID, CSMI.

클라이언트 프로그램의 트랜잭션 ID를 서버 프로그램의 트랜잭션 ID로서 지정하는 것이 좋습니다. 이를 통해 사용자가 수집하는 통계 및 모니터링 데이터는 동일한 트랜잭션 하에서 올바르게 상호 연관될 수 있습니다.

분산 링크 프로그램 요청에 사용된 트랜잭션 ID는 다음과 같이 서버 프로그램에 전달됩니다.

- 분산 링크 프로그램 요청에 사용자 고유의 트랜잭션 ID를 지정하면 이는 EIB의 EIBTRNID 필드에서 서버 프로그램에 전달됩니다.
- EIBTRNID는 DPL API 또는 서버 자원 정의에 지정된 대로 TRANSID 값으로 설정됩니다. 그렇지 않으면 이는 기본적으로 클라이언트의 트랜잭션 코드로 설정되고, 이는 클라이언트의 EIBTRNID에 있는 것과 동일한 값입니다.

5. 서버 프로그램의 SYNCONRETURN 옵션.

SYNCONRETURN 옵션을 지정하면 이는 서버의 자원이 제어를 클라이언트로 리턴하기 직전에 별도의 논리 작업 단위로 확약됨을 의미합니다. 즉, 서버가 클라이언트에 제어를 리턴하기 직전에 서버에 암시적 동기점이 발행됩니다. 462 페이지의 그림 114에서는 SYNCONRETURN 옵션과 함께 DPL(Distributed Program Link)를 사용하는 예제를 제공합니다. SYNCONRETURN 옵션은 클라이언트 프로그램이 복구 가능한 자원을 갱신하지 않을 때 사용하기 위한 것입니다. 예를 들어, 화면 처리를 수행할 때입니다. 그러나 클라이언트에 복구 가능한 자원이 있는 경우에는 이 시점에서는 확약되지 않습니다. 이들은 클라이언트 자체가 동기점에 도달하거나 클라이언트 태스크 종료 시에 암시적 동기점에 도달할 때 확약됩니다. 클라이언트와 서버 프로그램이 이 용도로 올바르게 설계되었는지와 데이터 무결성을 해치지 않는지 확인해야 합니다. 예를 들어, 클라이언트 프로그램이 데이터를 서버에 전달했고, 그 결과 서

버가 서버 리전이 소유하는 데이터베이스를 갱신하는 경우에는 독립적인 동기점만을 지정해도 안전하고 클라이언트 프로그램에서 발생하는 일에 대해 종속성이 없다면 독립적인 동기점만을 지정하십시오. 이 옵션은 EXECUTIONSET (DPLSUBSET)가 지정되지 않는 한 서버 프로그램이 클라이언트 리전에서 로컬로 실행되는 경우에는 아무런 효과가 없습니다. 이 경우 로컬 링크를 관리하는 동기점 규칙이 적용됩니다.

SYNCONRETURN 옵션 없이 클라이언트는 명시적 명령 또는 태스크 종료 시 암시적 동기점을 사용하여 클라이언트와 서버 자원 둘 모두에 대해 논리 작업 단위를 확약합니다. 따라서, 이 경우 서버 자원은 클라이언트 자원이 확약될 때 동일한 동기점의 일부로서 확약됩니다. 463 페이지의 그림 115에서는 SYNCONRETURN 옵션 없이 DPL(Distributed Program Link)을 사용하는 예제를 보여줍니다.

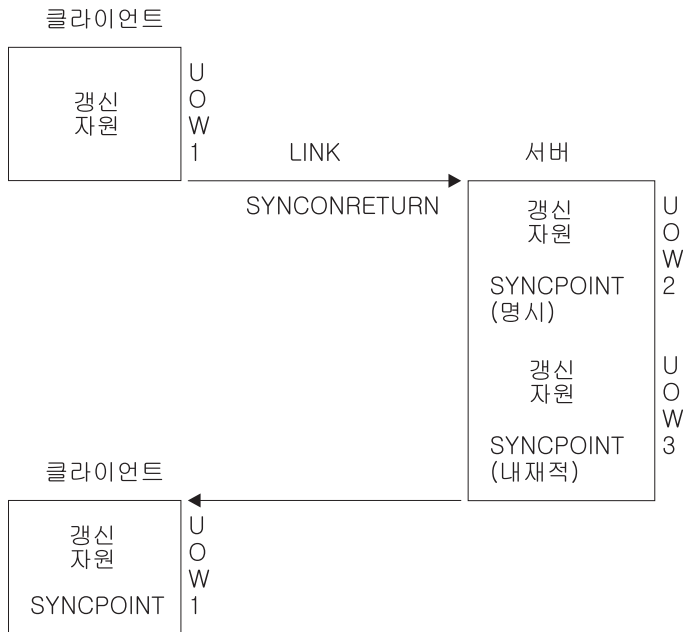


그림 114. SYNCONRETURN 옵션과 함께 DPL(Distributed Program Link) 사용

참고: 여기에는 클라이언트용으로 하나와 서버용으로 두 개, 총 세 개의 논리적 작업 단위(LUW)가 포함됩니다. 클라이언트 자원은 서버와 별개로 확약됩니다.

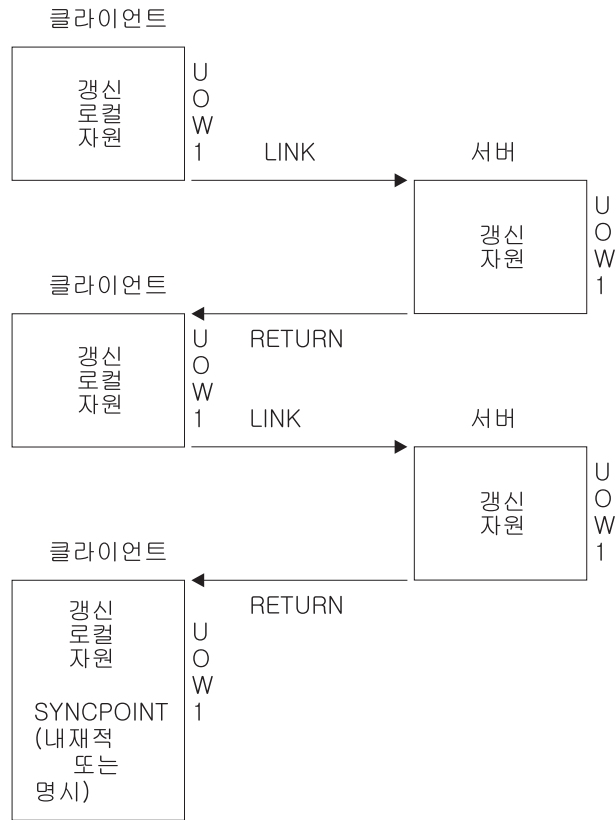


그림 115. SYNCONRETURN 옵션 없이 DPL(Distributed Program Link) 사용

참고: 암시적 또는 명시적 동기점은 모든 클라이언트와 서버 자원을 확약되게 합니다. 클라이언트와 서버 자원 둘 모두가 확약되는 시기를 관별할 책임은 클라이언트에게 있기 때문에 단 하나의 논리적 작업 단위(LUW)만이 있습니다.

클라이언트에 HANDLE ABEND 명령이 있는 경우를 고려해야 합니다. 클라이언트가 서버에서 이상 종료로 다루고 있을 때에는 클라이언트는 서버가 이상 종료되는 시기를 제어합니다. 이는 또한 LINK 명령에 SYNCONRETURN 옵션이 지정된 경우에도 참입니다. 이 경우에는 클라이언트는 최소한의 정리를 수행한 후에 이상 종료를 실행하는 것이 좋습니다. 이로 인해 클라이언트 논리적 작업 단위(LUW) 및 서버 원인 둘 모두가 취소됩니다.

DPL(Distributed Program Link)의 프로그래밍 고려사항

DPL(Distributed Program Link)을 사용하는 애플리케이션을 작성할 때에는 다음 요소를 고려하십시오.

동일한 클라이언트 태스크로부터 다중 DPL(Distributed Program Link) 실행

클라이언트 태스크는 SYNCONRETURN 옵션이 지정되지 않는 한 단일 클라이언트 작업 단위에서 둘 이상의 트랜잭션 코드를 사용하여 단일 CICS 서버 리전에 DPL(Dis-

tributed Program Link)을 요청할 수 없습니다. 클라이언트 태스크는 동일 또는 기본 트랜잭션 코드를 사용하여 하나의 CICS 서버 시스템에 여러 DPL(Distributed Program Link)을 실행할 수 있습니다.

클라이언트와 서버 프로그램 간의 자원 공유

서버 프로그램은 클라이언트에서 태스크의 수명 저장영역에 대한 액세스(예: TWA)가 없습니다. 또한 파일 요청이 전달 중이지 않는 한 클라이언트 프로그램이 사용 중인 자원에 대한 액세스를 반드시 가지고 있을 필요가 없습니다(예: 파일).

DPL 및 Function Shipping을 동일 CICS 시스템으로 혼합

Function Shipping과 DPL을 동일한 클라이언트 태스크에서 동일 CICS 시스템에 혼합할 때에는 특별히 주의하십시오. 몇몇 고려사항은 다음과 같습니다.

- 클라이언트 태스크는 Function Ship 요청을 할 수 없고 SYNCONRETURN 옵션과 함께 DPL(Distributed Program Link)를 동일한 세션에서 사용할 수 있습니다(동일한 논리적 작업 단위(LUW) 또는 지정된 시스템 초기화 매개변수 MROFSE=YES 또는 UOW 또는 TASK로 설정된 IPCONN MIRRORLIFE). DPL(Distributed Program Link)은 INVREQ 응답으로 실패합니다. 이 경우 EIBRESP2는 14로 설정됩니다.
- 클라이언트 태스크는 Function Ship 요청을 할 수 없고 TRANSID 옵션과 함께 DPL(Distributed Program Link)를 동일한 클라이언트 논리적 작업 단위(LUW)에서 사용할 수 있습니다. DPL(Distributed Program Link)은 INVREQ 응답으로 실패합니다. 이 경우 EIBRESP2는 15로 설정됩니다.
- SYNCONRETURN 옵션과 함께 DPL 요청을 뒤따르는 모든 Function-Shipped 요청은 서버 논리적 작업 단위(LUW)와는 별개의 논리적 작업 단위(LUW)에서 실행됩니다.
- TRANSID 옵션과 함께 동일한 서버 리전까지 DPL 요청을 뒤따르는 실행 중인 모든 Function-Shipped 요청은 기본 미리 트랜잭션 코드 하에서가 아니라 TRANSID 옵션에 지정된 트랜잭션 코드 하에서 실행됩니다. Function-Shipped 요청은 클라이언트가 확약할 때 전체 클라이언트 논리적 작업 단위(LUW)의 일부로서 확약됩니다.
- SYNCONRETURN 또는 TRANSID 옵션 없이 DPL 요청 전이나 후에 실행 중인 모든 Function-Shipped 요청은 클라이언트가 확약할 때 전체 클라이언트 논리적 작업 단위(LUW)의 일부로서 확약됩니다.

Function Shipping에 대한 자세한 정보는 시작하기의 CICS Function Shipping의 내용을 참조하십시오.

동일 CICS 시스템에 DPL 및 DTP 혼합

동일 애플리케이션에서 DPL과 DTP 둘 모두를 사용할 때, 특히 서버 프로그램에서 DTP를 사용할 때에는 주의해야 합니다. 예를 들어, SYNCONRETURN 옵션을 사용하지 않는 경우에는 동기점에 DPL 서버 프로그램을 필요로 하는 DTP 파트너에서 동기점을 사용

하는 것을 피해야 합니다.



그림 116. DPL과 DTP 혼합의 예제

프로그램을 DPL(Distributed Program Link) 서브세트로 제한

프로그램이 DPL(Distributed Program Link)의 결과로서 실행될 경우 DPL(Distributed Program Link) 서브세트를 호출한 전체 CICS API의 서브세트로 프로그램이 제한됩니다. 서버 프로그램에서 금지된 명령은 467 페이지의 표 28에 요약되어 있습니다.

로컬 LINK 명령이 호출하는 프로그램을 EXECUTIONSET(DPLSUBSET) 옵션이 있는 이 서브세트로 제한하고 싶음을 프로그램 자원 정의에서만 지정할 수 있습니다. 그러면 금지된 명령을 사용하면 애플리케이션이 분배 환경에서 사용되기 전에 발견될 수 있습니다.

서버 프로그램이 로컬에서 실행 중이면 다음 고려사항이 적용됩니다.

- 서버 프로그램에 EXECUTIONSET(DPLSUBSET)가 지정된 경우에는 SYNCONRETURN 옵션은 제어를 클라이언트 프로그램에 되돌리기 전에 로컬 서버 프로그램에서 암시적 동기점을 사용하게 합니다. 이 경우에는 서버 프로그램이 로컬에서 실행 중이므로 클라이언트와 서버 자원 둘 모두가 확약됩니다. 그러나 SYNCONRETURN은 클라이언트에 복구 가능한 자원이 없을 때 사용하기 위한 것입니다.
- EXECUTIONSET(FULLAPI)가 서버 프로그램에 지정되면 SYNCONRETURN 옵션은 무시됩니다.
- TRANSID 및 DATALENGTH 옵션은 로컬 링크를 처리할 때 무시되지만 인수의 형식은 확인됩니다. 예를 들어, TRANSID 인수는 모두 공백일 수 없습니다.

프로그램이 호출된 방법 결정

ASSIGN 명령의 STARTCODE 옵션에 리턴된 2바이트 값은 DPL(Distributed Program Link) 함수의 지원으로 확장되어 서버 프로그램이 DPL(Distributed Program Link) 서브세트로 제한되어 있음을 찾아낼 수 있게 합니다. EXEC CICS 명령에 대한 프로그래밍 정보는 참조 -> 애플리케이션 개발에서 CICS API 명령의 내용을 참조하십시오.

ASSIGN 명령으로 사용자 관련 정보에 액세스

서버 프로그램에서 ASSIGN 명령의 USERID 및 OPID 키워드와 함께 리턴되는 값은 클라이언트 CICS 리전과 서버 CICS 리전 사이에서 사용 중인 연결에 ATTACHSEC 옵션

션이 정의된 방법에 따라 다릅니다. 예를 들어, 서버 프로그램이 클라이언트 프로그램과 동일한 USERID 및 OPID 값에 액세스할 수 있고 ATTACHSEC 옵션이 판별하는 다른 값에 액세스할 수 있도록 시스템이 정의될 수 있습니다.

ATTACHSEC(LOCAL)이 지정되면 OPID 및 USERID 매개변수가 해당하는 사용자 ID는 다음에 표시된 순서대로 다음 중 하나입니다.

1. SESSIONS 자원 정의의 **USERID** 매개변수(사전 설정된 보안)에 지정된 사용자(있는 경우)
2. 연결 자원 정의의 **SECURITYNAME** 매개변수에 지정된 사용자 ID(존재하고, 세션에 사전 설정된 보안 사용자 ID가 정의되지 않은 경우).
3. 서버 리전의 **DFLTUSER** 시스템 초기화 매개변수에 지정된 사용자 ID(세션이나 연결 정의가 사용자 ID를 지정하지 않는 경우).

LOCAL이 아닌 값이 ATTACHSEC에 지정된 경우에는 사인온한 사용자 ID는 클라이언트 리전으로부터 접속 요청에 수신한 ID입니다.

자세한 정보는 보안에서 상호 통신 보안의 내용을 참조하십시오.

또 다른 보안 관련 고려사항은 **ASSIGN** 명령의 CMDSEC 및 RESSEC 옵션의 사용과 관계가 있습니다. 서버 리전에 미리 트랜잭션에 대한 트랜잭션 정의의 속성이 있습니다. 이는 동일 TRANSID가 사용되었더라도 클라이언트 리전에서 정의와는 다를 수 있습니다.

LINK 명령의 예외 조건:

클라이언트와 서버 프로그램으로 돌아가는 DPL을 지원하여 도입되는 오류 조건이 있습니다.

클라이언트 프로그램으로 돌아간 예외 조건

클라이언트 프로그램으로 돌아간 조건 코드는 서버 프로그램에서 『알려지지 않은 원격 시스템』 또는 『확약 실패』 등과 같은 이벤트를 설명합니다. LINK 명령에서 INVREQ 및 LENGERR 조건을 발생시키기 위해 EIBRESP2 값으로 식별되는 서로 다른 이유가 있습니다. ROLLEDBACK, SYSIDERR 및 TERMERR 조건 또한 발생할 수 있습니다. 이러한 명령에 대한 프로그래밍 정보는 참조 -> 애플리케이션 개발에서 CICS API 명령의 내용을 참조하십시오.

원격 리전에서 미리 트랜잭션이 실패하면 DPL 요청을 실행한 애플리케이션은 다음 둘 모두가 참인 경우에만 미러의 이상 종료를 다루고 자체의 로컬 자원을 확약할 수 있습니다.

1. 애플리케이션은 미러의 실패로 인해 초래된 이상 종료를 명백하게 처리하고 다음 중 하나를 수행합니다.

- 암시적 동기점을 정상 트랜잭션 종료로 사용합니다.
 - 또는 명시적 동기점 요청을 실행합니다.
2. 원격 미리 트랜잭션은 애플리케이션의 작업 단위의 범위 내에서 어떤 복구 가능한 작업도 수행하지 않았습니다. 즉, 미치는 SYNCONRETURN을 사용하여 DPL(Distributed Program Link) 요청만을 위해 호출되었습니다.

다른 모든 경우, 즉, 애플리케이션이 이상 종료를 처리하지 않거나 미러가 복구 가능한 작업을 수행하는 경우(예: 심지어 복구 불가능한 파일에 대한 파일 갱신) CICS는 트랜잭션을 취소하도록 강제합니다.

PGMIDERR 조건은 로컬 프로그램 정의가 프로그램이 원격임을 지정하는 경우 HANDLE ABEND PROGRAM, LOAD, RELEASE 및 XCTL 명령에서 발생합니다. 이 예외는 EIBRESP2 값 9로 규정됩니다.

서버 프로그램으로 돌아간 예외 조건

INVREQ 조건은 금지된 API 명령의 사용을 다룹니다. 표 28에 요약된 금지된 명령 중 하나를 실행하는 경우 EIBRESP2 값 200으로 규정된 INVREQ가 서버 프로그램에 리턴됩니다. 서버 프로그램이 INVREQ 조건을 다루지 않으면 기본 조치는 서버 프로그램이 이상 종료 코드 ADPL을 사용하여 실행되고 있는 미리 트랜잭션을 이상 종료하는 것입니다.

DPL 관련 예외 조건에 대한 프로그래밍 정보는 참조 -> 애플리케이션 개발에서 LINK의 내용을 참조하십시오.

표 28. DPL에 의해 호출된 프로그램에서 금지된 API 명령

명령	옵션
ASSIGN	ALTSCRNHT ALTSCRNWD APLKYBD APLTEXT BTRANS COLOR DEFSCRNHT DEFSCRNWD DELIMITER DESTCOUNT DESTID DESTIDLENG DS3270 DSSCS EWASUPP EXTDS FACILITY FCI GCHARS GCODES GMMI HIGHLIGHT INPARTN KATAKANA LDCMNEM LDCNUM MAPCOLUMN MAPHEIGHT MAPLINE MAPWIDTH MSRCONTROL NATLANGINUSE NEXTTRANSID NUMTAB OPCLASS OPSECURITY OUTLINE PAGENUM PARTNPAGE PARTNS PARTNSET PS QNAME SCRNHT SCRNWD SIGDATA SOSI STATIONID TCTUALENG TELLERID TERMCODE TERMPRIORITY TEXTKYBD TEXTPRINT UNATTEND USERNAME USERPRIORITY VALIDATION
CONNECT PROCESS	모두
CONVERSE	모두
EXTRACT ATTRIBUTES	모두
EXTRACT PROCESS	모두
FREE	모두
HANDLE AID	모두

표 28. DPL에 의해 호출된 프로그램에서 금지된 API 명령 (계속)

명령	옵션
ISSUE	ABEND CONFIRMATION ERROR PREPARE SIGNAL PRINT ABORT ADD END ERASE NOTE QUERY RECEIVE REPLACE SEND WAIT
LINK	INPUTMSG INPUTMSGLEN
PURGE MESSAGE	모두
RECEIVE	모두
RETURN	INPUTMSG INPUTMSGLEN
ROUTE	모두
SEND	CONTROL MAP PARTNSET TEXT TEXT(MAPPED) TEXT(NOEDIT) PAGE
SIGNOFF	모두
SIGNON	모두
START	TERMIN, 여기서 해당 값은 상호 시스템 세션의 ID입니다. (즉, 실행 태스크의 프린시פל 기능은 터미널이 아니라 세션입니다.)
START CHANNEL	TERMIN, 여기서 해당 값은 상호 시스템 세션의 ID입니다. (즉, 실행 태스크의 프린시פל 기능은 터미널이 아니라 세션입니다.)
SYNCPPOINT	SYNCONRETURN이 LINK에 지정한 경우 서버 리전에서 실행될 수 있습니다.
SYNCPPOINT ROLLBACK	SYNCONRETURN이 LINK에 지정한 경우 서버 리전에서 실행될 수 있습니다.
WAIT TERMINAL	모두
XCTL	INPUTMSG INPUTMSGLEN

다음 명령 또한 제한되지만 SYNCONRETURN이 LINK에 지정된 경우 서버 리전에서 사용될 수 있습니다.

- CPIRR COMMIT
- CPIRR BACK
- EXEC DLI TERM
- CALL DLI TERM

명령에 특정 옵션만이 금지되어 있고, 이들이 표시되어 있습니다. 나열된 모든 APPC 명령은 프린시פל 기능을 참조할 때에만 금지됩니다. 이들 중 하나인 CONNECT PROCESS 명령은 비DPL 환경에서 프린시פל 기능을 참조하는 경우에도 오류를 일으킵니다. 이는 CONNECT PROCESS 명령이 서버 프로그램에서 해당 프린시פל 기능을 참조하는 경우 발생한 예외 조건이 DPL 오류를 나타내기 때문에 여기에 포함되어 있습니다.

비동기 처리

원격으로 초기화된 트랜잭션의 응답은 트랜잭션을 초기화한 태스크로 반드시 리턴되지 않아도 됩니다. 이러한 이유로 처리가 비동기로 언급됩니다.

비동기 처리는 원격 요청이 처리되는 동안 로컬 자원을 묶어야 하거나 묶으려고 할 때 유용합니다. 예를 들어, 원격 데이터베이스에 대한 온라인 조회를 사용하여, 터미널 운영자는 첫 번째 조회에 대한 응답을 기다리지 않고 조회 입력을 계속할 수 있습니다.

논리 트랜잭션과 같이 START 명령을 사용하여 원격 시스템에서 트랜잭션을 시작할 수 있습니다. RETRIEVE 명령을 사용하여, 논리 트랜잭션에서와 같이, 원격으로 실행되는 START, CANCEL, SEND 또는 RECEIVE 명령의 결과로 태스크에 대해 저장된 데이터를 검색할 수 있습니다.

DTP(Distributed Transaction Processing)

DTP의 주요 장점은 두 트랜잭션이 한 세션의 독점 제어를 가질 수 있고 『반대』일 수도 있다는 것입니다. DTP는 원격 자원을 원격으로 처리해야 하거나 시스템 사이에 데이터를 전송해야 하는 경우에 유용합니다.

DTP를 사용하여, 유연하고 효율적인 애플리케이션을 디자인할 수 있습니다. DTP는 EXEC CICS 또는 CPI 통신과 함께 사용할 수 있습니다. CICS 상호 통신 기능을 사용하는 EXEC CICS API 및 애플리케이션을 사용하여 LU 유형 6.2 맵핑되지 않은 대화를 보유하는 DTP 애플리케이션에서 C, C++ 및 어셈블러 언어를 사용할 수 있습니다.

DTP는 CICS 및 비CICS 플랫폼 둘 다를 포함하여, 다양한 파트너와 함께 사용할 수 있습니다(APPC를 지원하는 한). DTP에 대한 추가 정보는 제품 개요에서 분배 트랜잭션 처리 개요 및 시작하기에서 상호 통신 시작하기의 내용을 참조하십시오.

공통 프로그래밍 인터페이스 통신(CPI 통신)

CPI 통신은 대체 API를 기존 CICS APPC 지원에 제공합니다. CPI 통신은 APPC 연결에 대해 DTP를 제공하고, COBOL, C, C++, PL/I 및 어셈블러 언어에서 사용될 수 있습니다.

CPI 통신은 공통 API의 일관성이 이점인 여러 시스템 플랫폼을 포함하는 APPC 네트워크에서 사용할 수 있는 API를 정의합니다.

공통 프로그래밍 인터페이스 통신(CPI 통신)은 시스템 애플리케이션 구조(SAA) 공통 프로그래밍 인터페이스(CPI)의 통신 요소입니다.

CPI 통신 인터페이스는 APPC API를 제공하는 시스템에서 애플리케이션과 대화할 수 있습니다. 여기에는 CICS 플랫폼의 애플리케이션이 포함됩니다. 대화의 한쪽 끝에서 EXEC CICS APPC API 명령을 사용하고 다른 끝에서 CPI 통신 명령을 사용할 수 있습니다.

CPI 통신에서는 특정 정보(부가 정보)가 파트너 프로그램과의 대화를 시작해야 합니다. 부가 정보의 CICS 구현은 시스템 프로그래머가 유지보수 책임을 가지고 있는 파트너 자원을 사용하여 이뤄집니다.

애플리케이션에서 CPI 통신 인터페이스로의 호출은 CICS CPI 통신 스텝(DFHCPLC)을 사용하여 애플리케이션을 링크 편집하여 해결됩니다. 이를 수행하는 방법에 대한 정보는 166 페이지의 『CICS 제공 인터페이스 모듈 포함』의 내용을 참조하십시오.

CPI 통신 API는 일반 호출 인터페이스로 정의됩니다. 인터페이스는 공통 프로그래밍 인터페이스 통신 참조서에 설명되어 있습니다.

외부 CICS 인터페이스(EXCI)

외부 CICS 인터페이스는 MVS에서 실행 중인 비CICS 프로그램(클라이언트 프로그램)이 CICS 리전에서 실행 중인 프로그램(서버 프로그램)을 호출하고 통신 영역의 데이터를 전달하고 수신할 수 있도록 하는 API(Application Programming Interface)입니다. CICS 프로그램은 다른 CICS 프로그램에 의해 링크된 것처럼 호출됩니다.

이 프로그래밍 인터페이스는 사용자가 CICS 시스템에 세션(파이프)을 할당하여 열고 세션을 통해 DPL(Distributed Program Link)(DPL) 요청을 전달할 수 있도록 합니다. CICS 리전간 통신(IRC)은 이러한 요청을 지원하며 각 파이프는 하나의 MRO 세션에 매핑됩니다.

EXCI에 대한 프로그래밍 정보에 대해서는 애플리케이션 개발에서 외부 CICS 인터페이스 소개의 내용을 참조하십시오.

외부 CICS 인터페이스를 사용하는 클라이언트 프로그램은 서로에 대한 지식이 없거나 간섭하지 않고 동일한 MVS 주소 공백에서 모두 공존하는 다양한 사용자(동일한 TCB 나 별도의 TCB 아래에 있는)에 대해 열 세션을 조작할 수 있습니다.

외부 CICS 인터페이스는 두 가지 양식의 프로그래밍 인터페이스를 제공합니다.

- EXCI CALL 인터페이스는 다음을 수행할 수 있는 6개의 명령으로 구성됩니다.
 - MVS 아래에서 실행 중인 비CICS 프로그램에서 CICS 시스템으로 세션을 할당하여 엽니다.
 - 비CICS 프로그램에서 이러한 세션에 대한 DPL 요청을 발행합니다.
 - DPL 요청 완료 시 세션을 닫고 할당을 해제합니다.
- EXEC CICS 인터페이스는 다음을 제공합니다.
 - EXCI CALL 인터페이스에 대한 6개의 명령 모두를 하나의 호출로 수행하는 단일 콤포지트 명령(LINK PROGRAM)

명령은 CICS 명령 레벨 API(Application Programming Interface)의 DPL(Distributed Program Link) 명령과 같은 양식을 사용합니다.

CICS는 외부 CICS 인터페이스를 사용하는 애플리케이션에서 MVS 자원 복구 서비스 (RRS)를 지원합니다. 이는 다음을 의미합니다.

- CICS 서버 프로그램이 복구 가능 자원을 변경하는 작업 단위는 이제 EXCI 클라이언트 프로그램과 연관된 MVS 복구 단위의 파트가 될 수 있습니다.
- EXCI 클라이언트가 복구 단위를 확약 또는 백아웃할 것을 결정할 때까지, 서버 프로그램이 클라이언트에 제어를 리턴하거나 여러 EXCI DPL 호출을 통해 계속할 경우 CICS 서버 작업 단위가 확약될 수 있습니다.

제 5 부 데이터 맵핑

애플리케이션 데이터를 XML로, XML을 애플리케이션 데이터로 변환하기 위한 맵핑 작성.

제 19 장 애플리케이션 데이터와 XML 맵핑 및 변환

애플리케이션 2진 데이터를 XML로(부터) 변환하는 애플리케이션을 쓸 수 있습니다. CICS는 많은 수의 고급 언어를 지원하며 런타임 처리 중에 데이터가 변환되는 방식을 맵핑할 XML 지원을 제공합니다. CICS는 웹 서비스 지원의 일부로, 애플리케이션 데이터를 SOAP 메시지에 XML로 맵핑하는 데 동일한 기술을 사용합니다.

시작하기 전에

XML 지원을 실행하고 CICS가 데이터 또는 XML을 변환할 때 선택적으로 유효성 검사를 수행하려면 Java를 설치했어야 합니다.

이 태스크 정보

애플리케이션 데이터를 XML로(부터) 변환시키는 이 접근법의 장점은 CICS가 XML 구문 분석기에 제공된 기능보다 더 많은 역할을 한다는 점입니다. CICS는 XML을 해석하고 애플리케이션 데이터의 레코드 기반 변환을 수행할 수 있습니다. 따라서 이 접근법을 사용하면 XML에 대해 작업하는 애플리케이션을 보다 간편하고 빠르게 작성할 수 있습니다.

CICS XML 지원은 애플리케이션 2진 데이터를 XML로 변환하거나 XML을 애플리케이션 2진 데이터로 변환하기 위한 필수 아티팩트를 작성하는 데 도움을 주도록 제공된 유틸리티입니다. XML 지원은 번들 디렉토리나 z/OS UNIX의 지정된 또 다른 위치에 아티팩트를 작성할 수 있습니다.

프로시저

1. XML 지원을 사용하여 맵핑을 작성하십시오.
2. CICS에 자원을 작성하여 맵핑을 사용 가능하게 하십시오.
3. **TRANSFORM** API 명령을 사용할 애플리케이션을 작성하거나 갱신하십시오. 애플리케이션은 채널 기반 인터페이스를 사용해야 합니다.
4. 애플리케이션을 실행하여 변환이 의도한 대로 작동하는지 테스트하십시오. CICS가 데이터를 제대로 변환시키는지 확인하려면 유효성 검사를 켜 수 있습니다.

다음에 수행할 작업

이 단계는 다음 주제에서 더 자세히 설명합니다.

CICS XML 지원

CICS XML 지원은 XML을 고급 언어 구조로(부터) 변환하도록 돕는 일괄처리 유틸리티 세트입니다. 지원은 최소한의 프로그래밍 노력으로 XML 처리를 수행하는 신속한 애플리케이션 전개를 지원합니다.

XML 지원을 CICS에 사용하면 XML을 구문 분석하거나 구성하기 위해 써야 하는 코드 양이 줄어듭니다. CICS는 XML 단편과 애플리케이션의 데이터 구조 간에 데이터를 변환시킵니다.

XML 지원은 단순 언어 구조 또는 기존 XML 스키마의 언어 구조에서 XML 스키마를 작성하며 COBOL, C/C++, PL/I를 지원합니다. 또한 CICS가 런타임에 XML 데이터를 2진 애플리케이션 데이터로 또는 그 반대로 자동 변환시키기 위해 사용하는 메타데이터를 생성합니다. 메타데이터는 XML 바인딩에 정의되고 z/OS UNIX에 저장됩니다. XML 바인딩의 스키마는 z/OS UNIX의 /usr/lpp/cicsts/cicsts52/schemas/xmltransform/ 디렉토리에 있습니다.

CICS XML 지원은 두 개의 유틸리티 프로그램으로 이루어집니다.

DFHLS2SC

이 유틸리티는 언어 구조에서 XML 스키마와 바인딩을 생성합니다.

DFHSC2LS

이 유틸리티는 애플리케이션에 사용할 수 있는 XML 바인딩과 언어 구조를 생성합니다. WSDL 문서나 XML 스키마를 입력으로 사용할 수 있습니다.

두 프로그램을 모두 실행하는 JCL 프로시저는 *hlq.XDFHINST* 라이브러리에 있으며, 여기서 *hlq*는 CICS 설치의 상위 레벨 규정자입니다.

관련 태스크:

563 페이지의 『언어 구조에서 맵핑 생성』

애플리케이션 데이터에서 XML을 작성(또는 반대로)하기 위해서는 런타임에 CICS가 데이터와 XML을 변환할 방식을 설명하는 맵핑을 작성합니다. 어느 애플리케이션 데이터 레코드에서나 시작할 수 있습니다. 예를 들어, COMMAREA, VSAM 파일, 임시 저장영역 큐 또는 DB2 레코드로 시작할 수 있습니다.

566 페이지의 『XML 스키마에서 맵핑 생성』

기존 XML 스키마를 따르는 XML에서 애플리케이션 데이터를(또는 반대로) 작성하기 위해서는 CICS가 런타임에 데이터를 변환할 방식을 설명하는 맵핑을 작성합니다. XML 스키마나 WSDL 문서로부터 시작할 수 있습니다.

관련 정보:

전개 오류 진단

DFHLS2SC: 고급 언어 대 XML 스키마 변환

DFHLS2SC 카탈로그된 프로시저는 고급 언어 구조로부터 XML 스키마와 XML 바인딩 파일을 생성합니다. XML을 구문 분석하거나 작성할 수 있는 CICS 프로그램을 작성하려 할 때 DFHLS2SC를 사용하십시오. DFHLS2SC의 작업 제어문, 기호 매개변수, 입력 매개변수, 설명이 나열됩니다.

DFHLS2SC의 작업 제어문

JOB 작업을 시작합니다.

EXEC

프로시저 이름(DFHLS2SC)을 지정합니다.

INPUT.SYSUT1 DD

입력을 지정합니다. 입력 매개변수는 일반적으로 입력 스트림에 지정됩니다. 하지만 데이터 세트나 파티션된 데이터 세트의 멤버에 정의할 수 있습니다.

기호 매개변수

다음 기호 매개변수가 DFHLS2SC에 정의됩니다.

JAVADIR=*path*

DFHLS2SC에 사용되는 Java 디렉토리의 이름을 지정합니다. 이 매개변수의 값은 전체 경로 이름 */usr/lpp/path*를 제공하는 */usr/lpp/*에 추가됩니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **JAVADIR** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

PATHPREFIX=*prefix*

다른 매개변수에 사용된 z/OS UNIX 디렉토리를 확장하는 선택적 접두부를 지정합니다. 기본값은 빈 문자열입니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **JAVADIR** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

SERVICE=*value*

IBM 지원 센터에서 지시한 경우에만 이 매개변수를 사용하십시오.

TMPDIR=*tmpdir*

DFHLS2SC가 임시 작업공간으로 사용하는 z/OS UNIX의 디렉토리 위치를 지정합니다. 작업을 실행하는 사용자 ID에 이 디렉토리에 대한 읽기 및 쓰기 권한이 있어야 합니다.

기본값은 */tmp*입니다.

TMPFILE=*tmpprefix*

DFHLS2SC가 임시 작업공간 파일의 이름을 구성하는 데 사용하는 접두부를 지정합니다.

기본값은 SC2WS입니다.

USSDIR=path

z/OS UNIX 파일 시스템에서 CICS TS 디렉토리의 이름을 지정합니다. 이 매개 변수의 값은 전체 경로 이름 `/usr/lpp/cicsts/path`를 제공하는 `/usr/lpp/cicsts/`에 추가됩니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **USSDIR** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

임시 작업공간

DFHLS2SC는 런타임 시에 다음의 세 임시 파일을 작성합니다.

```
tmpdir/tmpprefix.in
tmpdir/tmpprefix.out
tmpdir/tmpprefix.err
```

여기서,

`tmpdir`은 **TMPDIR** 매개변수에 지정된 값입니다.

`tmpprefix`는 **TMPFILE** 매개변수에 지정된 값입니다.

파일의 기본 이름(**TMPDIR** 및 **TMPFILE**이 지정되지 않았을 때)은 다음과 같습니다.

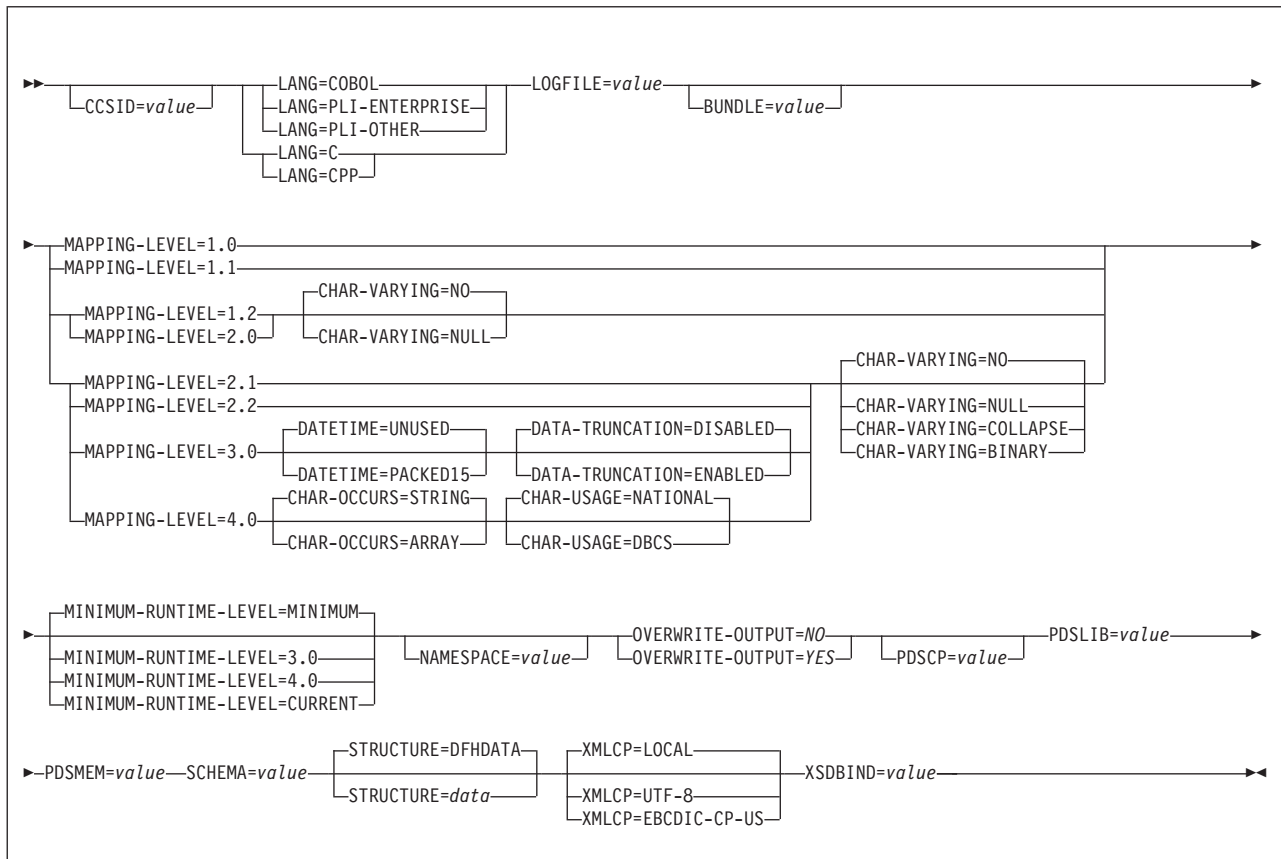
```
/tmp/LS2SC.in
/tmp/LS2SC.out
/tmp/LS2SC.err
```

중요사항: DFHLS2SC는 z/OS UNIX 파일이나 데이터 세트 멤버에 대한 액세스를 잠그지 않습니다. 따라서 둘 이상의 DFHLS2SC 인스턴스가 동시에 실행하며 동일한 임시 작업공간 파일을 사용하는 경우에는, 한 작업이 작업공간 파일을 사용하는 동안 또 다른 작업이 이 파일을 겹쳐쓰게 되어 예측 불가능한 장애가 발생할 수 있습니다.

이러한 이유로, 이 같은 상황을 피하는 작동 프로시저와 이름 지정 규칙을 고안하는 것이 좋습니다. 예를 들어, 시스템 기호 매개변수 **SYSUID**를 사용하여 개별 사용자 고유의 작업공간 파일 이름을 생성할 수 있습니다.

이들 임시 파일은 작업 종료 전에 삭제됩니다.

DFHLS2SC의 입력 매개변수



매개변수 사용

- 입력 매개변수를 어느 순서로든 지정할 수 있습니다.
- 각 매개변수는 새 행에서 시작해야 합니다.
- 매개변수(및 연속 문자, 사용하는 경우)가 72열을 초과하면 안됩니다. 73 - 80열은 공백을 포함해야 합니다.
- 매개변수가 단일 행에 맞추기에 너무 긴 경우에는 행의 끝에 별표(*) 문자를 사용하여 매개변수가 다음 행에서 계속됨을 표시하십시오. 별표 이전의 모든 항목(공백을 포함하여)이 매개변수의 일부로 간주됩니다. 예를 들어, 다음과 같습니다.

```
XSDBIND=/path/xsdbinddir*  
/app1
```

은 다음과 동등합니다.

```
XSDBIND=/path/xsdbinddir/app1
```

- 행의 첫 번째 문자 위치에 있는 # 문자는 주석 문자입니다. 해당 행은 무시됩니다.
- 행의 마지막 문자 위치에 있는 쉼표는 선택적인 행 분리문자이며 무시됩니다.

매개변수 설명

BUNDLE=*value*

z/OS UNIX의 번들 디렉토리 경로 및 이름을 지정합니다. 이 값을 지정하면 XML 지원이 XSD 바인딩을 포함한 번들을 생성합니다. 이 매개변수의 경로 정보는 **XSDBIND** 매개변수의 경로 정보를 덮어씁니다.

선택적으로 디렉토리 이름이 아닌 아카이브 파일을 지정할 수 있습니다. XML 지원은 .zip 및 .jar 아카이브를 지원합니다. 하지만 BUNDLE 자원을 설치하려 시도하기 전에 아카이브의 압축을 풀어야 합니다.

이 매개변수를 지정하지 않으면 CICS는 XML 스키마와 바인딩을 **XSDBIND** 매개변수에 지정된 위치에 둡니다.

CCSID=*value*

런타임 시에 애플리케이션 데이터 구조에서 문자 데이터를 인코딩하는 데 사용되는 CCSID를 지정합니다. 이 매개변수의 값이 **LOCALCCSID** 시스템 초기화 매개변수의 값을 덮어씁니다. *value* 값은 Java 및 z/OS 변환 서비스에서 지원되는 EBCDIC CCSID여야 합니다. 이 매개변수를 지정하지 않는 경우, 애플리케이션 데이터 구조는 시스템 초기화 매개변수에 지정된 CCSID를 사용하여 인코딩됩니다.

어느 맵핑 레벨에서나 이 매개변수를 사용할 수 있습니다.

CHAR-VARYING={NO**|**NULL**|**COLLAPSE**|**BINARY**}**

맵핑 레벨이 1.2 이상일 때 언어 구조의 문자 필드가 맵핑되는 방법을 지정합니다. COBOL의 문자 필드는 X 유형의 Picture 절입니다(예를 들어, PIC(X) 10). C/C++의 문자 필드는 문자 배열입니다. 이 매개변수는 Enterprise 및 기타 PL/I 언어 구조에 적용되지 않습니다. 다음 옵션을 선택할 수 있습니다.

NO 문자 필드가 xsd:string으로 맵핑되고 고정 길이 필드로 처리됩니다. 데이터의 최대 길이는 필드의 길이와 동일합니다. NO가 맵핑 레벨 2.0 이하에서 COBOL 및 PL/I에 대한 **CHAR-VARYING** 매개변수의 기본값입니다.

NULL

문자 필드가 xsd:string으로 맵핑되고 널(null) 종료 문자열로 처리됩니다. CICS는 XML 스키마로부터 변환할 때 종료 널 문자를 추가합니다. 문자열의 최대 길이는 언어 구조에 표시된 길이보다 한 문자 적은 길이로 계산됩니다. NULL은 C/C++에 대한 **CHAR-VARYING** 매개변수의 기본값입니다.

COLLAPSE

문자 필드가 xsd:string으로 맵핑됩니다. 이 필드의 후미 공백은 XML 스키마에 포함되지 않습니다. COLLAPSE는 맵핑 레벨 2.1 이상에서 COBOL 및 PL/I에 대한 **CHAR-VARYING** 매개변수의 기본 *value*입니다.

BINARY

문자 필드가 `xsd:base64binary` 데이터 유형으로 맵핑되고 고정 길이 필드로 처리됩니다. **CHAR-VARYING** 매개변수의 **BINARY** *value*는 맵핑 레벨 1.2부터만 사용할 수 있습니다.

CHAR-OCCURS={STRING|ARRAY}

맵핑 레벨이 4.0 이상일 때 언어 구조의 문자 배열이 맵핑되는 방법을 지정합니다.
예: `PIC X OCCURS 20`. 이 매개변수는 COBOL 언어에 의해서만 사용됩니다.

ARRAY

문자 배열이 XML 배열에 맵핑됩니다. 이것은 모든 문자가 개별 XML 요소로서 맵핑됨을 의미합니다. 이것은 또한 맵핑 레벨 3.0 이하에서의 작동입니다.

STRING

문자 배열이 XML 문자열로 맵핑됩니다. 이것은 전체 COBOL 배열이 단일 XML 요소로서 맵핑됨을 의미합니다.

CHAR-USAGE={NATIONAL|DBCS}

COBOL에서는, 국가 데이터 유형, PIC N을 UTF-16 또는 DBCS 데이터에 사용할 수 있습니다. 이 설정은 NSYMBOL 컴파일러 옵션으로 제어합니다. 지원의 **CHAR-USAGE** 매개변수를 NSYMBOL 컴파일러 옵션과 동일한 값으로 설정해서 데이터가 적합하게 처리되도록 해야 합니다. UTF-16을 사용하는 경우 이는 일반적으로 `CHAR-USAGE=NATIONAL`로 설정됩니다.

DBCS `PIC(n)` 필드의 데이터가 UTF-16 인코딩 데이터로서 취급됩니다.

NATIONAL

`PIC (n)` 필드의 데이터가 DBCS 인코딩 데이터로 취급됩니다.

DATA-TRUNCATION={DISABLED|ENABLED}

가변 길이 데이터가 고정 길이 필드 구조에 용인되는지 여부를 지정합니다.

DISABLED

데이터가 CICS가 기대하고 있는 고정 길이보다 작은 경우, CICS는 잘린 데이터를 거부하고 오류 메시지를 발행합니다.

ENABLED

데이터가 CICS가 기대하고 있는 고정 길이보다 작은 경우, CICS가 잘린 데이터를 용인하고 누락된 데이터를 널(null)값으로 처리합니다.

DATETIME={UNUSED|PACKED15}

고급 언어 구조의 잠재적 **ABSTIME** 필드가 시간소인으로 맵핑되는지 여부를 지정합니다.

PACKED15

길이 15(8바이트)의 팩형 10진수 필드가 CICS ABSTIME 필드로 처리되고 시간 소인으로 맵핑됩니다.

UNUSED

길이 15(8바이트)의 팩형 10진수 필드가 시간 소인으로 처리되지 않습니다.

맵핑 레벨 3.0에서 이 매개변수를 설정할 수 있습니다.

LANG=COBOL

고급 언어 구조의 프로그래밍 언어가 COBOL임을 지정합니다.

LANG=PLI-ENTERPRISE

고급 언어 구조의 프로그래밍 언어가 Enterprise PL/I임을 지정합니다.

LANG=PLI-OTHER

고급 언어 구조의 프로그래밍 언어가 Enterprise PL/I 이외의 PL/I 레벨임을 지정합니다.

LANG=C

고급 언어 구조의 프로그래밍 언어가 C임을 지정합니다.

LANG=CPP

고급 언어 구조의 프로그래밍 언어가 C++임을 지정합니다.

LOGFILE=*value*

DFHLS2SC가 활동 로그 및 추적 정보를 쓰는 파일의 완전한 z/OS UNIX 이름입니다. DFHLS2SC는 아직 존재하지 않을 경우 디렉토리 구조가 아닌 파일을 작성합니다.

일반적으로는 이 파일을 사용하지 않지만 DFHLS2SC에 관한 문제가 발생할 경우 IBM 서비스 조직이 이 파일을 사용하도록 요청할 수 있습니다.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}

XML 바인딩 및 언어 구조를 생성할 때 사용할 지원의 맵핑 레벨을 지정합니다. 사용 가능한 최신 맵핑 레벨을 사용하는 것이 좋습니다. Atom 피드에 대한 XML 바인딩을 작성 중이면 맵핑 레벨 3.0을 사용해야 합니다.

MINIMUM-RUNTIME-LEVEL={MINIMUM|3.0|4.0|CURRENT}

XML 바인딩을 전개할 수 있는 최소 CICS 런타임 환경을 지정합니다. 지정한 기타 매개변수와 일치하지 않는 레벨을 선택할 경우 오류 메시지가 수신됩니다. 선택 가능한 옵션은 다음과 같습니다.

MINIMUM

CICS의 가능한 최저 런타임 레벨이 지정한 매개변수에 기반하여 자동으로 할당됩니다.

3.0 CICS XML 지원을 사용하고 고급 데이터 맵핑을 이용하려면 런타임 레벨 3.0 이상을 지정하십시오.

4.0 생성된 웹 서비스 바인딩 파일이 CICS TS 5.2 리전 이상에 전개됩니다. 이 런타임 레벨을 사용하면 **MAPPING-LEVEL** 매개변수에 대해 4.0 이하의 맵핑 레벨을 사용할 수 있습니다. 이 레벨에서 임의의 선택적 매개변수를 사용할 수 있습니다.

CURRENT

런타임 환경이 바인딩 파일을 생성하는 데 사용한 환경으로 설정된 CICS 리전으로 생성된 바인딩 파일을 전개하려면 이 런타임 레벨을 사용하십시오.

NAMESPACE=value

생성된 XML 스키마에 사용할 CICS의 이름 공간을 지정합니다. Atom 피드의 경우 CICS는 Atom 피드와 Atom 이름 공간에 이 이름 공간을 제공합니다.

이 매개변수를 지정하지 않을 경우 CICS가 이름 공간을 자동으로 생성합니다.

OVERWRITE-OUTPUT=NO|YES

파일 시스템의 기존 CICS BUNDLE을 겹쳐쓸 수 있는지 여부를 제어합니다.

NO 기존 BUNDLE을 바꾸지 않습니다. 기존 BUNDLE이 있으면 DFHLS2SC는 오류 메시지 DFHPI9689E를 발행하고 종료됩니다.

YES 기존 BUNDLE을 바꿉니다. 기존 BUNDLE이 있으면 DFHPI9683W 메시지가 발행되어 파일을 바꿨음을 알립니다.

PDSCP=value

파티션된 데이터 세트 멤버에 사용되는 코드 페이지를 지정하며, 여기서 *value*는 CCSID 번호 또는 Java 코드 페이지 번호입니다. 이 매개변수를 지정하지 않으면 z/OS UNIX System Services 코드 페이지가 사용됩니다. 예를 들어, PDSCP=037을 지정할 수 있습니다.

PDSLIB=value

처리될 고급 언어 데이터 구조를 포함하는 파티션된 데이터 세트의 이름을 지정합니다.

제한사항: 파티션된 데이터 세트의 레코드는 80바이트의 고정 길이를 가져야 합니다.

PDSMEM=value

처리될 고급 언어 데이터 구조를 포함하는 파티션된 데이터 세트 멤버의 이름을 지정합니다.

SCHEMA=*value*

XML 스키마를 쓰는 파일의 완전한 z/OS UNIX 이름입니다. XML 스키마는 WSDL 2.0 스펙에 따릅니다. DFHLS2SC는 아직 존재하지 않을 경우 디렉토리 구조가 아닌 파일을 작성합니다.

STRUCTURE={DFHDATA|*data***}**

C 및 C++의 최상위 레벨 데이터 구조 이름입니다. 기본값은 DFHDATA입니다.

XMLCP={LOCAL|UTF-8|EBCDIC-CP-US}

XML 바인딩을 생성하는데 사용되는 코드 페이지를 지정합니다.

LOCAL

이 값이 기본값입니다. XML이 로컬 코드 페이지를 사용하여 생성되고 XML 스키마에 인코딩 태그가 생성되지 않음을 지정합니다.

UTF-8

XML이 UTF-8 코드 페이지를 사용하여 생성됨을 지정합니다. XML 스키마에 인코딩 태그가 생성됩니다. 이 옵션을 지정하면 다른 플랫폼 간에 XML 스키마를 복사할 때 인코딩이 올바른 상태를 유지하도록 해야 합니다.

EBCDIC-CP-US

XML이 US EBCDIC 코드 페이지를 사용하여 생성됨을 지정합니다. XML 스키마에 인코딩 태그가 생성됩니다.

XSDBIND=*value*

XSD 바인딩의 완전한 z/OS UNIX 이름입니다. DFHLS2SC는 아직 존재하지 않을 경우 디렉토리 구조가 아닌 파일을 작성합니다. BUNDLE 매개변수가 지정된 경우에는 경로를 제외시키십시오. 파일 확장자는 .xsdbind입니다.

관련 태스크:

563 페이지의 『언어 구조에서 맵핑 생성』

애플리케이션 데이터에서 XML을 작성(또는 반대로)하기 위해서는 런타임에 CICS가 데이터와 XML을 변환할 방식을 설명하는 맵핑을 작성합니다. 어느 애플리케이션 데이터 레코드에서나 시작할 수 있습니다. 예를 들어, COMMAREA, VSAM 파일, 임시 저장영역 큐 또는 DB2 레코드로 시작할 수 있습니다.

DFHSC2LS: XML 스키마 대 고급 언어 변환

DFHSC2LS 카탈로그된 프로시저는 XML 스키마나 WSDL 문서로부터 고급 언어 데이터 구조 및 XML 바인딩을 생성합니다. XML을 구문 분석하거나 작성할 수 있는 CICS 프로그램을 작성하려 할 때 DFHSC2LS를 사용하십시오. DFHSC2LS의 작업 제어문, 기호 매개변수, 입력 매개변수, 설명이 나열됩니다.

DFHSC2LS의 작업 제어문

JOB 작업을 시작합니다.

EXEC

프로시저 이름(DFHSC2LS)를 지정합니다.

INPUT.SYSUT1 DD

입력을 지정합니다. 입력 매개변수는 일반적으로 입력 스트림에 지정됩니다. 하지만 데이터 세트나 파티션된 데이터 세트의 멤버에 정의할 수 있습니다.

기호 매개변수

다음 기호 매개변수가 DFHSC2LS에 정의됩니다.

JAVADIR=*path*

DFHSC2LS에 사용되는 Java 디렉토리의 이름을 지정합니다. 이 매개변수의 값은 전체 경로 이름 */usr/lpp/path*를 제공하는 */usr/lpp/*에 추가됩니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **JAVADIR** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

PATHPREFIX=*prefix*

다른 매개변수에 사용된 z/OS UNIX 디렉토리를 확장하는 선택적 접두부를 지정합니다. 기본값은 빈 문자열입니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **JAVADIR** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

TMPDIR=*tmpdir*

DFHLS2SC가 임시 작업공간으로 사용하는 z/OS UNIX의 디렉토리 위치를 지정합니다. 작업을 실행하는 사용자 ID에 이 디렉토리에 대한 읽기 및 쓰기 권한이 있어야 합니다.

기본값은 */tmp*입니다.

TMPFILE=*tmpprefix*

DFHSC2LS가 임시 작업공간 파일의 이름을 구성하는 데 사용하는 접두부를 지정합니다.

기본값은 SC2LS입니다.

USSDIR=*path*

z/OS UNIX 파일 시스템에서 CICS TS 디렉토리의 이름을 지정합니다. 이 매개변수의 값은 전체 경로 이름 */usr/lpp/cicsts/path*를 제공하는 */usr/lpp/cicsts/*에 추가됩니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **USSDIR** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

SERVICE=*value*

IBM 지원 센터에서 지시한 경우에만 이 매개변수를 사용하십시오.

임시 작업공간

DFHSC2LS는 런타임 시에 다음의 세 임시 파일을 작성합니다.

```
tmpdir/tmpprefix.in
tmpdir/tmpprefix.out
tmpdir/tmpprefix.err
```

여기서,

*tmpdir*은 **TMPDIR** 매개변수에 지정된 값입니다.

*tmpprefix*는 **TMPFILE** 매개변수에 지정된 값입니다.

파일의 기본 이름(**TMPDIR** 및 **TMPFILE**이 지정되지 않았을 때)은 다음과 같습니다.

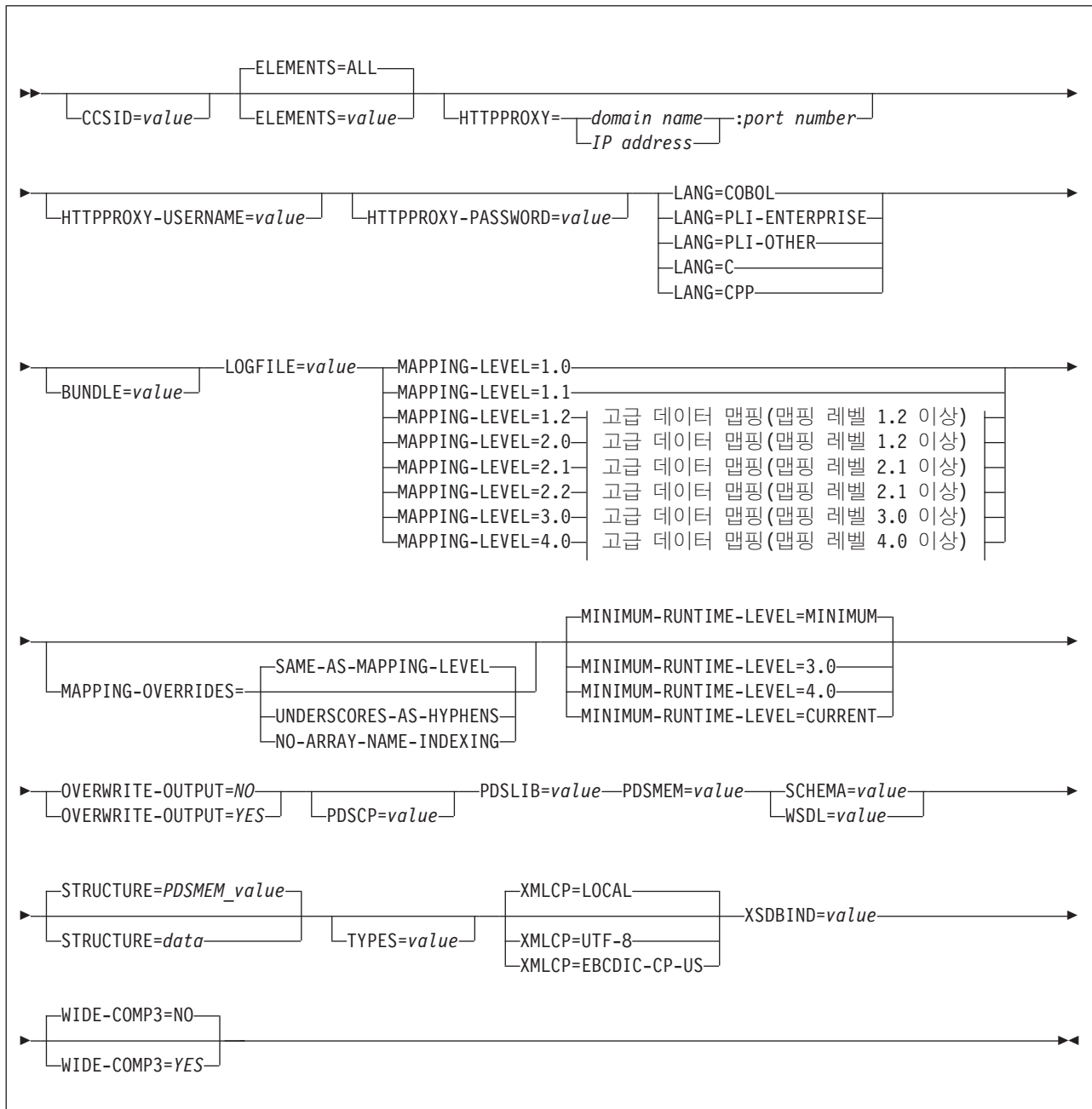
```
/tmp/SC2LS.in
/tmp/SC2LS.out
/tmp/SC2LS.err
```

중요사항: DFHSC2LS는 z/OS UNIX 파일이나 데이터 세트 멤버에 대한 액세스를 잠그지 않습니다. 따라서 둘 이상의 DFHSC2LS 인스턴스가 동시에 실행하며 동일한 임시 작업공간 파일을 사용하는 경우에는, 한 작업이 작업공간 파일을 사용하는 동안 또 다른 작업이 이 파일을 겹쳐쓰게 되어 예측 불가능한 장애가 발생할 수 있습니다.

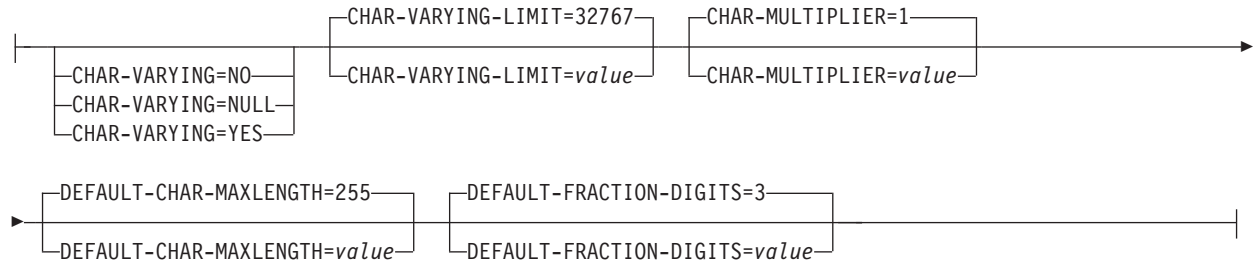
이러한 이유로, 이 같은 상황을 피하는 작동 프로시저와 이름 지정 규칙을 고안하는 것이 좋습니다. 예를 들어, 시스템 기호 매개변수 **SYSUID**를 사용하여 개별 사용자 고유의 작업 공간 파일 이름을 생성할 수 있습니다.

이들 임시 파일은 작업 종료 전에 삭제됩니다.

DFHSC2LS의 입력 매개변수



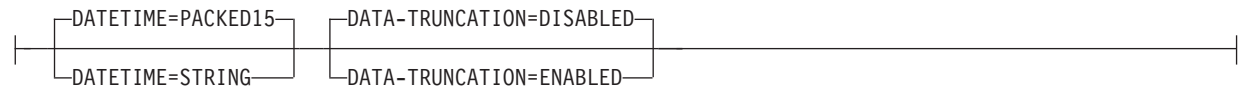
고급 데이터 맵핑(맵핑 레벨 1.2 이상):



고급 데이터 맵핑(맵핑 레벨 2.1 이상):



고급 데이터 맵핑(맵핑 레벨 3.0):



매개변수 사용

- 입력 매개변수를 어느 순서로든 지정할 수 있습니다.
- 각 매개변수는 새 줄에서 시작해야 합니다.
- 매개변수(및 연속 문자, 사용하는 경우)가 72열을 초과하면 안됩니다. 73 - 80열은 공백을 포함해야 합니다.
- 매개변수가 단일 행에 맞추기에 너무 긴 경우에는 행의 끝에 별표(*) 문자를 사용하여 매개변수가 다음 행에서 계속됨을 표시하십시오. 별표 이전의 모든 항목(공백을 포함하여)이 매개변수의 일부로 간주됩니다. 예를 들어, 다음과 같습니다.

```
XSDBIND=xsdbindir*  
/app1
```

은 다음과 동등합니다.

```
XSDBIND=xsdbindir/app1
```

- 행의 첫 번째 문자 위치에 있는 # 문자는 주석 문자입니다. 해당 행은 무시됩니다.

매개변수 설명

BUNDLE=*value*

z/OS UNIX의 번들 디렉토리 경로 및 이름을 지정합니다. 이 값을 지정하면 XML 지원은 번들 디렉토리에 XML 바이딩을 생성하고 번들 Manifest를 작성합니다. 이

매개변수의 경로 정보는 **XSDBIND** 매개변수의 경로 정보를 덮어씁니다. 이 매개변수를 지정하지 않을 경우 CICS는 XML 바인딩을 지정된 위치에 둡니다.

CCSID=value

런타임 시에 애플리케이션 데이터 구조에서 문자 데이터를 인코딩하는 데 사용되는 CCSID를 지정합니다. 이 매개변수의 값이 **LOCALCCSID** 시스템 초기화 매개변수의 값을 덮어씁니다. *value*는 Java 및 z/OS 변환 서비스가 지원하는 EBCDIC CCSID 여야 합니다(z/OS Unicode Services User's Guide and Reference 참조). 이 매개변수를 지정하지 않는 경우, 애플리케이션 데이터 구조는 시스템 초기화 매개변수에 지정된 CCSID를 사용하여 인코딩됩니다.

어느 맵핑 레벨에서나 이 매개변수를 사용할 수 있습니다. 그러나 생성된 파일을 CICS TS 3.1 리전에 전개하려는 경우, APAR PK23547을 적용하여 웹 서비스 바인딩 파일을 설치하기 위한 코드의 최소 런타임 레벨을 달성해야 합니다.

CHAR-MULTIPLIER={1|value}

맵핑 레벨이 1.2 이상일 때 각 문자에 대해 허용할 바이트 수를 지정합니다. 이 매개변수의 *value*는 1 - 2,147,483,647 범위에 있는 양의 정수일 수 있습니다. 모든 비슷자 문자 기반 맵핑은 이 승수에 적용됩니다. 2진, 숫자, 존 및 팩형 10진수 필드는 이 승수가 적용되지 않습니다.

예를 들어 런타임 시에 모든 2바이트 문자 주위에 잠재적인 SO(shift-out) 및 SI(shift-in) 문자를 위한 공간을 허용하기 위해 3의 승수를 선택할 수 있는 DBCS 문자를 사용할 계획인 경우 이 매개변수가 유용할 수 있습니다.

CCSID=1200(UTF-16 표시)을 설정할 때, **CHAR-MULTIPLIER**에 대해 유일하게 유효한 값은 2 또는 4입니다. UTF-16을 사용할 때 기본값은 2입니다. 애플리케이션 데이터가 1 UTF-16 인코딩 단위가 필요한 문자를 포함할 것으로 예상할 때는 **CHAR-MULTIPLIER=2**를 사용하십시오. 애플리케이션 데이터가 2 UTF-16 인코딩 단위가 필요한 문자를 포함할 것으로 예상할 때는 **CHAR-MULTIPLIER=4**를 사용하십시오.

참고: **CHAR-MULTIPLIER**를 1로 설정해도 DBCS 문자 사용을 배제하지 않으며, 2로 설정해도 UTF-16 대리 쌍의 사용을 배제하지 않습니다. 그러나 와이드 문자가 일상적으로 사용되는 경우 일부 올바른 값이 할당된 필드에 맞지 않습니다. 더 큰 **CHAR-MULTIPLIER** 값이 사용되는 경우, XML에서 유효한 것보다 더 많은 문자를 할당된 필드에 저장할 수 있습니다. 적합한 범위 제한사항을 준수하도록 주의해야 합니다.

CHAR-VARYING={NO|NULL|YES}

맵핑 레벨이 1.2 이상일 때 가변 길이 문자 데이터가 맵핑되는 방법을 지정합니다. 가변 길이 2진 데이터 유형은 항상 컨테이너 또는 변동 구조로 맵핑됩니다. 이 매개변수를 지정하지 않을 경우 기본 맵핑은 지정한 언어에 따라 다릅니다. 다음 옵션을 선택할 수 있습니다.

NO 가변 길이 문자 데이터가 고정 길이 문자열로 맵핑됩니다.

NULL

가변 길이 문자 데이터가 널(null) 종료 문자열로 맵핑됩니다.

YES 가변 길이 문자 데이터가 PL/I에서 CHAR VARYING 데이터 유형으로 맵핑됩니다. COBOL, C, C++ 언어에서 가변 길이 문자 데이터는 두 개의 관련 요소, 데이터 길이 및 데이터로 이루어진 동등한 표현으로 맵핑됩니다.

CHAR-VARYING-LIMIT={32767|value}

맵핑 레벨이 1.2 이상일 때 언어 구조로 맵핑되는 가변 길이 문자 데이터와 2진 데이터의 최대 크기를 지정합니다. 문자나 2진 데이터가 이 매개변수에 지정된 값보다 크면 컨테이너로 맵핑되고 생성된 언어 구조에 컨테이너 이름이 사용됩니다. *value*의 가능한 범위는 0부터 기본값 32 767바이트까지입니다.

DATA-TRUNCATION={DISABLED|ENABLED}

가변 길이 데이터가 고정 길이 필드 구조에 용인되는지 여부를 지정합니다.

DISABLED

데이터가 CICS가 기대하고 있는 고정 길이보다 작은 경우, CICS는 잘린 데이터를 거부하고 오류 메시지를 발행합니다.

ENABLED

데이터가 CICS가 기대하고 있는 고정 길이보다 작은 경우, CICS가 잘린 데이터를 용인하고 누락된 데이터를 널(null)값으로 처리합니다.

DATETIME={PACKED15|STRING}

xsd:dateTime 필드가 CICS ABSTIME 데이터 형식이나 텍스트로 맵핑됨을 지정합니다.

PACKED15

xsd:dateTime 필드가 CICS ABSTIME 형식으로 맵핑됩니다.

STRING

xsd:dateTime 필드가 텍스트로 맵핑됩니다. 이 맵핑은 이전의 모든 맵핑 레벨과 동일합니다.

맵핑 레벨 3.0에서 이 매개변수를 사용할 수 있습니다.

DEFAULT-CHAR-MAXLENGTH={255|value}

맵핑 레벨이 1.2 이상일 때 XML 스키마 문서나 WSDL 문서에 길이가 암시되지 않은 맵핑에 대한 문자에서 문자 데이터의 기본 배열 길이를 지정합니다. 이 매개변수의 가능한 *value*는 1 - 2 147 483 647 범위의 양의 정수입니다.

DEFAULT-FRACTION-DIGITS=3|value

XML 10진수 스키마 유형에서 사용할 기본 분수 자릿수를 지정합니다. 기본값은

3입니다. COBOL의 경우 매개변수 **WIDE-COMP3**이 사용되면 올바른 범위는 0-17 또는 0-30입니다. C 또는 PLI의 경우 유효한 범위는 0 - 30입니다.

ELEMENTS={ALL|value}

사용할 글로벌 요소 로컬 이름의 목록을 정의합니다. 기본값 ALL은 모든 글로벌 요소가 사용됨을 나타냅니다.

HTTPPROXY={domain name|IP address}:port number

XML 스키마나 WSDL 문서에 인터넷에 있는 다른 XML 스키마 또는 WSDL 파일에 대한 참조가 포함되어 있고 DFHSC2LS를 실행 중인 시스템이 프록시 서버를 사용하여 인터넷에 액세스하는 경우에는 도메인 이름이나 IP 주소 및 프록시 서버의 포트 번호를 지정하십시오. 예를 들어, 다음과 같습니다.

HTTPPROXY=proxy.example.com:8080

다른 경우에는 이 매개변수가 필요하지 않습니다.

HTTPPROXY-USERNAME=value

DFHSC2LS를 실행 중인 시스템이 HTTP 프록시 서버를 사용하여 인터넷에 액세스하고 HTTP 프록시 서버가 기본 인증을 사용하는 경우 **HTTPPROXY-PASSWORD**에 사용할 HTTP 프록시 사용자 이름을 지정합니다. **HTTPPROXY**도 지정하는 경우에만 이 매개변수를 사용할 수 있습니다.

HTTPPROXY-PASSWORD=value

DFHSC2LS를 실행 중인 시스템이 HTTP 프록시 서버를 사용하여 인터넷에 액세스하고 HTTP 프록시 서버가 기본 인증을 사용하는 경우 **HTTPPROXY-USERNAME**에 사용할 HTTP 프록시 암호를 지정합니다. **HTTPPROXY**도 지정하는 경우에만 이 매개변수를 사용할 수 있습니다.

INLINE-MAXOCCURS-LIMIT={1|value}

XML 속성의 maxOccurs 속성을 기반으로 하여 인라인 변동 반복 내용이 사용되는지 여부를 지정합니다.

INLINE-MAXOCCURS-LIMIT 매개변수는 맵핑 레벨 1.2부터만 사용할 수 있습니다. **INLINE-MAXOCCURS-LIMIT**의 value는 0 - 32 767의 범위의 양의 정수일 수 있습니다. 0 값은 인라인 맵핑을 사용하지 않음을 나타냅니다. 1 값은 선택적 요소가 인라인 맵핑되도록 합니다. maxOccurs 속성의 value가 **INLINE-MAXOCCURS-LIMIT**의 value보다 크면 컨테이너 기반 맵핑이 사용되고, 그렇지 않은 경우에는 인라인 맵핑이 사용됩니다.

가변적으로 반복되는 목록을 인라인 맵핑하도록 결정한 경우에는 반복 데이터의 단일 항목 길이를 고려하십시오. 길이가 긴 인스턴스가 거의 없는 경우 컨테이너 기반 맵핑이 바람직합니다. 짧은 길이의 인스턴스가 많으면 인라인 맵핑이 더 좋습니다.

LANG=COBOL

고급 언어 구조의 프로그래밍 언어가 COBOL임을 지정합니다.

LANG=PLI-ENTERPRISE

고급 언어 구조의 프로그래밍 언어가 Enterprise PL/I임을 지정합니다.

LANG=PLI-OTHER

고급 언어 구조의 프로그래밍 언어가 Enterprise PL/I 이외의 PL/I 레벨임을 지정합니다.

LANG=C

고급 언어 구조의 프로그래밍 언어가 C임을 지정합니다.

LANG=CPP

고급 언어 구조의 프로그래밍 언어가 C++임을 지정합니다.

LOGFILE=value

DFHLS2JS가 활동 로그 및 추적 정보를 쓰는 파일의 완전한 z/OS UNIX 이름입니다. DFHSC2LS는 아직 존재하지 않을 경우 디렉토리 구조가 아닌 파일을 작성합니다.

일반적으로는 이 파일을 사용하지 않지만 DFHSC2LS에 관한 문제가 발생할 경우 IBM 서비스 조직이 이 파일을 사용하도록 요청할 수 있습니다.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}

XML 바인딩 및 언어 구조를 생성할 때 사용할 지원의 맵핑 레벨을 지정합니다. 사용 가능한 최신 맵핑 레벨을 사용하는 것이 좋습니다. DFHSC2LS의 경우 맵핑 레벨 3.0 이상을 사용할 것을 권장합니다.

3.0 xsd:dateTime 데이터 유형은 CICS ASKTIME 형식으로 맵핑됩니다.

4.0 UTF-16을 사용하려는 경우 CICS TS 5.2 리전에서는 이 맵핑 레벨을 사용하십시오.

MAPPING-OVERRIDES={SAME-AS-MAPPING-LEVEL|UNDERSCORES-AS-HYPHENS|NO-ARRAY-NAME-INDEXING}

지정된 맵핑 레벨에 대한 기본 맵핑 작동을 덮어쓰는지 여부를 지정합니다. 기본값은 SAME-AS-MAPPING-LEVEL입니다.

UNDERSCORES-AS-HYPHENS

COBOL 전용입니다. XML 문서의 밑줄을 문자 X 대신 하이픈으로 변환합니다. 이 옵션은 생성된 COBOL 언어 구조를 보다 쉽게 읽을 수 있도록 개선합니다. 필드 이름 충돌이 발생하는 경우 필드를 고유하게 하기 위해 필드에 번호가 지정됩니다. 자세한 정보는 511 페이지의 『XML 스키마 대 COBOL 맵핑』의 내용을 참조하십시오.

NO-ARRAY-NAME-INDEXING

Enterprise PL/I 전용입니다. 배열에 속한 필드 이름은 상위 레벨 구조의 범위에서만 고유해야 합니다.

MINIMUM-RUNTIME-LEVEL={MINIMUM|3.0|CURRENT}

XML 바인딩을 전개할 수 있는 최소 CICS 런타임 환경을 지정합니다. 지정한 기타 매개변수와 일치하지 않는 레벨을 선택할 경우 오류 메시지가 수신됩니다. 선택 가능한 옵션은 다음과 같습니다.

MINIMUM

CICS의 가능한 최저 런타임 레벨이 지정한 매개변수에 기반하여 자동으로 할당됩니다.

3.0 CICS XML 지원을 사용하고 고급 데이터 맵핑을 이용하려면 런타임 레벨 3.0 이상을 지정하십시오.

4.0 생성된 웹 서비스 바인딩 파일이 CICS TS 5.2 리전 이상에 전개됩니다. 이 런타임 레벨을 사용하면 **MAPPING-LEVEL** 매개변수에 대해 4.0 이하의 맵핑 레벨을 사용할 수 있습니다. 이 레벨에서 임의의 선택적 매개변수를 사용할 수 있습니다.

CURRENT

XML 바인딩을 생성하는 데 사용한 리전과 런타임 환경이 동일한 CICS 리전으로 생성된 XML 바인딩을 전개하려면 이 런타임 레벨을 사용하십시오.

OVERWRITE-OUTPUT=NO|YES

파일 시스템의 기존 CICS 번들을 겹쳐쓸 수 있는지 여부를 제어합니다.

NO 기존 BUNDLE을 바꾸지 않습니다. 기존 BUNDLE이 있으면 DFHLS2SC는 오류 메시지 DFHPI9689E를 발행하고 종료됩니다.

YES 기존 BUNDLE을 바꿉니다. 기존 BUNDLE이 있으면 DFHPI9683W 메시지가 발행되어 파일을 바꿨음을 알립니다.

PDSCP=value

파티션된 데이터 세트 멤버에 사용되는 코드 페이지를 지정하며, 여기서 *value*는 CCSID 번호 또는 Java 코드 페이지 번호입니다. 이 매개변수를 지정하지 않으면 z/OS UNIX System Services 코드 페이지가 사용됩니다. 예를 들어, PDSCP=037을 지정할 수 있습니다.

PDSLIB=value

생성된 고급 언어를 포함하는 파티션된 데이터 세트의 이름을 지정합니다.

PDSMEM=value

DFHSC2LS가 고급 언어 구조를 포함할 파티션된 데이터 세트 멤버의 이름을 생성하기 위해 사용하는 1-6자의 접두부를 지정합니다.

DFHSC2LS는 각 조작의 파티션된 데이터 세트 멤버를 생성합니다. 접두부에 두 자리 숫자를 추가해서 멤버 이름을 생성합니다.

SCHEMA=value

XML 스키마를 읽는 파일의 완전한 z/OS UNIX 이름입니다. DFHSC2LS는 아직 존재하지 않을 경우 디렉토리 구조가 아닌 파일을 작성합니다.

DFHSC2LS에 대한 입력으로 XML 스키마 또는 WSDL 문서를 사용할 수 있습니다. 입력의 출처를 표시하려면 이 매개변수와 **WSDL** 매개변수 모두가 아닌 둘 중 하나를 지정해야 합니다.

STRUCTURE={PDSMEM_value|data}

C 및 C++의 최상위 레벨 데이터 구조 이름입니다. 기본값은 **PDSMEM** 매개변수의 값입니다.

TYPES=value

사용할 글로벌 유형 로컬 이름의 목록을 정의합니다. **ALL value**는 모든 글로벌 유형이 사용됨을 나타냅니다. 기본적으로 글로벌 유형은 사용되지 않습니다.

WIDE-COMP3=NO|YES

COBOL 전용입니다. COBOL 언어 구조에서 팩형 10진수 가변 길이의 최대 크기를 제어합니다.

NO DFHWS2LS가 COBOL 언어 구조 유형 COMP-3를 생성할 때 팩형 10진수 가변 길이를 18로 제한합니다. 팩형 10진수 크기가 18보다 크면 DFHPI9022W 메시지가 발행되어 지정된 유형이 총 18자리 숫자로 제한되고 있음을 표시합니다.

YES DFHWS2LS가 COBOL 언어 구조 유형 COMP-3를 생성할 때 최대 크기 31을 지원합니다.

WSDL=value

WSDL 문서의 완전한 z/OS UNIX 이름입니다.

DFHSC2LS에 대한 입력으로 XML 스키마 또는 WSDL 문서를 사용할 수 있습니다. 입력의 출처를 표시하려면 이 매개변수와 **SCHEMA** 매개변수 모두가 아닌 둘 중 하나를 지정해야 합니다.

XMLCP={LOCAL|UTF-8|EBCDIC-CP-US}

XML 바인딩을 생성하는데 사용되는 코드 페이지를 지정합니다.

LOCAL

이 값이 기본값입니다. XML이 로컬 코드 페이지를 사용하여 생성되고 XML 스키마에 인코딩 태그가 생성되지 않음을 지정합니다.

UTF-8

XML이 UTF-8 코드 페이지를 사용하여 생성됨을 지정합니다. XML 스키마에 인코딩 태그가 생성됩니다. 이 옵션을 지정하면 다른 플랫폼 간에 XML 스키마를 복사할 때 인코딩이 올바른 상태를 유지하도록 해야 합니다.

EBCDIC-CP-US

XML이 US EBCDIC 코드 페이지를 사용하여 생성됨을 지정합니다. XML 스키마에 인코딩 태그가 생성됩니다.

XSDBIND=*value*

XML 바인딩의 완전한 z/OS UNIX 이름입니다. DFHSC2LS는 아직 존재하지 않을 경우 디렉토리 구조가 아닌 파일을 작성합니다. 파일 확장자는 .xsdbind입니다.

CICS 지원의 맵핑 레벨

맵핑은 언어 구조와 XML 스키마 간에 정보가 변환되는 방식을 지정하는 규칙 세트입니다. 사용 가능한 가장 정교한 맵핑을 이용하려면 CICS 지원의 **MAPPING-LEVEL** 매개변수를 최신 레벨로 설정해야 합니다.

각 맵핑 레벨은 이전 맵핑의 기능을 상속하며 최고 레벨의 맵핑은 사용 가능한 최상의 기능을 제공합니다. 최고 맵핑 레벨은 런타임에 데이터 변환에 대한 더 많은 제어를 제공하며 특정 데이터 유형과 XML 요소에 대한 지원의 제한사항을 제거합니다.

이전에 해당 레벨에서 사용한 애플리케이션을 재전개하려는 경우에는 **MAPPING-LEVEL** 매개변수를 이전 레벨로 설정할 수 있습니다.

맵핑 레벨 4.0

맵핑 레벨 4.0은 CICS TS 5.2 리전과 호환 가능합니다.

맵핑 레벨 4.0 이상에서, DFHLS2SC 및 DFHLS2WS는 COBOL OCCURS DEPENDING ON 절을 지원하고 COBOL 문자 배열을 XML 문자열로 맵핑하도록 지원합니다. CICS 지원에서 **CHAR-OCCURS** 매개변수를 사용하여 이 작동을 설정할 수 있습니다.

- **DATA-TRUNCATION=ENABLED** 매개변수를 지정해야 합니다.
- 복합 OCCURS DEPENDING ON은 지원되지 않습니다. 이 제한사항은 OCCURS DEPENDING ON이 구조의 마지막 필드에 대해서만 지원된다는 의미입니다.
- CICS는 규정된 이름('OF' 키워드 사용)을 OCCURS DEPENDING ON 절의 대상으로 지원하지 않습니다(예를 들어, FIELD1 OF STRUCTURE1).
- CICS는 UNBOUNDED 키워드를 지원하지 않습니다. 애플리케이션이 예상하는 테이블의 최대 크기를 지정해야 합니다.

맵핑 레벨 4.0 이상에서, CICS 웹 서비스는 UTF-16 유니코드를 사용하여 인코딩한 애플리케이션 데이터의 변환을 지원합니다.

- LS2WS 또는 LS2SC를 사용하면 UTF-16에 대한 언어별 데이터 유형을 사용하여 이 작동을 사용 가능하게 할 수 있습니다.

- WS2LS 또는 SC2LS를 사용하면 CCSID=1200을 설정해서 이 작동을 사용 가능하게 할 수 있습니다.
- CICS는 단일 유니코드 코드 페이지, 『IBM Private Use Area의 UTF-16BE』(CCSID 1200)만 지원합니다.
- UTF-8을 사용하여 인코딩한 애플리케이션 데이터의 변환은 지원되지 않습니다.

참고: DFHLS2WS 및 DFHLS2SC는 COBOL GROUP USAGE NATIONAL 절을 지원하지 않습니다.

맵핑 레벨 3.0 이상

맵핑 레벨 3.0은 CICS TS 4.1 리전 이상과 호환 가능합니다.

이 맵핑 레벨은 다음 지원을 제공합니다.

- DFHSC2LS 및 DFHWS2LS는 xsd:dateTime 데이터 유형을 CICS ASKTIME 형식으로 맵핑합니다.
- DFHLS2WS는 한 컨테이너 뿐이 아닌 많은 컨테이너를 사용하는 애플리케이션에서 WSDL 문서와 웹 서비스 바인딩을 생성할 수 있습니다.
- 고정 길이 데이터 구조로 설명된 잘린 데이터를 허용합니다. CICS 지원에서 **DATA-TRUNCATION** 매개변수를 사용하여 이 작동을 설정할 수 있습니다.

맵핑 레벨 2.2 이상

맵핑 레벨 2.2는 APAR PK69738이 적용된 CICS TS 3.2 리전 이상과 호환 가능합니다.

맵핑 레벨 2.2 이상에서 DFHSC2LS 및 DFHWS2LS는 다음 XML 맵핑을 지원합니다.

- 요소의 고정 값
- 대체 그룹
- 추상 데이터 유형
- XML 스키마 <sequence> 요소는 <choice> 요소 내에 중첩될 수 있음

DFHSC2LS 및 DFHWS2LS는 다음 XML 맵핑에 대한 개선된 지원을 제공합니다.

- 추상 요소
- XML 스키마 <choice> 요소

맵핑 레벨 2.1 이상

맵핑 레벨 2.1은 APAR PK59794가 적용된 CICS TS 3.2 리전 이상과 호환 가능합니다.

이 맵핑 레벨은 새 **INLINE-MAXOCCURS-LIMIT** 매개변수 및 **CHAR-VARYING** 매개변수의 새 값으로 변수 내용을 처리하는 방식에 대한 더 많은 제어를 제공합니다.

맵핑 레벨 2.1 이상에서, DFHSC2LS 및 DFHWS2LS는 XML 맵핑에 대한 다음과 같은 개선된 지원과 새 지원을 제공합니다.

- XML 스키마 <any> 요소
- xsd:anyType 유형
- 추상 요소 허용
- **INLINE-MAXOCCURS-LIMIT** 매개변수
- minOccurs 속성

INLINE-MAXOCCURS-LIMIT 매개변수는 가변적으로 반복되는 목록이 인라인 맵핑되는지 여부를 지정합니다. 가변적 반복 내용의 인라인 맵핑에 대한 자세한 정보는 요소의 변동 배열을 참조하십시오.

minOccurs 속성에 대한 지원이 XML 스키마 <sequence>, <choice>, <all> 요소에 대해 개선되었습니다. minOccurs="0"인 경우 CICS 지원은 minOccurs="0" 속성도 모든 하위 요소의 속성인 것처럼 이 요소를 처리합니다.

맵핑 레벨 2.1 이상에서 DFHLS2SC 및 DFHLS2WS는 다음 XML 맵핑을 지원합니다.

- COBOL 및 PL/I의 FILLER 필드는 무시됨
- **CHAR-VARYING** 매개변수의 COLLAPSE 값
- **CHAR-VARYING** 매개변수의 BINARY 값

COBOL 및 PL/I의 FILLER 필드는 무시됩니다. 생성된 XML 스키마에 표시되지 않고 런타임에 데이터 구조에 해당 값이 남습니다.

COLLAPSE는 CICS가 텍스트 필드의 후미 공백을 무시하도록 합니다.

BINARY는 2진 필드에 대한 지원을 제공합니다. 이 값은 COBOL을 XML 스키마로 변환할 때 유용합니다. 이 옵션은 SBCS 문자 배열에만 사용 가능하며 xsd:string 필드가 아닌 고정 길이 xsd:base64Binary 필드로 배열을 맵핑할 수 있게 합니다.

맵핑 레벨 1.2 이상

맵핑 레벨 1.2는 CICS TS 3.1 리전 이상과 호환 가능합니다.

일괄처리 도구의 이 추가 매개변수를 통해 런타임에 문자 및 2진 데이터가 변환되는 방식에 대한 더 많은 제어가 사용 가능합니다.

- **CHAR-VARYING**
- **CHAR-VARYING-LIMIT**

- **CHAR-MULTIPLIER**
- **DEFAULT-CHAR-MAXLENGTH**

DFHSC2LS 또는 DFHWS2LS에서 **CHAR-MULTIPLIER** 매개변수를 사용하도록 결정한 경우 이 매개변수의 값을 사용하여 문자 데이터에 필요한 공간의 크기를 계산한 후 다음 규칙이 적용됩니다.

- DFHSC2LS 및 DFHWS2LS는 다음 매핑을 제공합니다.
 - 최대 길이가 32 767바이트를 초과한 가변 길이 문자 데이터 유형은 컨테이너로 매핑됩니다. **CHAR-VARYING-LIMIT** 매개변수를 사용하여 하한을 설정할 수 있습니다. 컨테이너의 이름을 저장하기 위해 16바이트 필드가 언어 구조에 작성됩니다. 런타임에 문자 데이터는 컨테이너에 저장되고 컨테이너 이름은 언어 구조에 놓입니다.
 - 최대 길이가 32 768바이트 미만인 가변 길이 문자 데이터 유형은 C/C++ 및 Enterprise PL/I를 제외한 모든 언어에서 VARYING 구조로 매핑됩니다. C/C++에서는, 이 데이터 유형이 널(null) 종료 문자열로 매핑되고 Enterprise PL/I에서는 이 데이터 유형이 VARYINGZ 구조로 매핑됩니다. **CHAR-VARYING** 매개변수를 사용하여 가변 길이 문자 데이터가 매핑되는 방식을 선택할 수 있습니다.
 - 최대 길이가 32 768바이트 미만인 가변 길이 2진 데이터는 모든 언어에서 VARYING 구조로 매핑합니다. 최대 길이가 32 768바이트 이상인 경우에는 데이터가 컨테이너로 매핑됩니다. 컨테이너의 이름을 저장하기 위해 16바이트 필드가 언어 구조에 작성됩니다. 런타임에 2진 데이터는 컨테이너에 저장되고 컨테이너 이름은 언어 구조에 놓입니다.

XML 스키마에 길이가 데이터 유형과 연관되지 않은 문자 데이터 유형이 있는 경우 DFHWS2LS 또는 DFHSC2LS에서 **DEFAULT-CHAR-MAXLENGTH** 매개변수를 사용하여 기본 길이를 지정할 수 있습니다.

DFHLS2SC 및 DFHLS2WS는 다음 매핑을 제공합니다.

- 문자 필드는 xsd:string 데이터 유형으로 매핑하며 런타임에 고정 길이 필드나 널(null) 종료 문자열로 처리될 수 있습니다. **CHAR-VARYING** 매개변수를 사용하여 PL/I를 제외한 모든 언어에서 런타임에 가변 길이 문자 데이터가 처리되는 방식을 선택할 수 있습니다.
- Base64Binary 데이터 유형은 데이터의 최대 길이가 32 767바이트를 초과할 때에나 길이가 정의되지 않았을 때 컨테이너로 매핑됩니다. 데이터 길이가 32 767 이하인 경우에는 base64Binary 데이터 유형이 모든 언어에서 VARYING 구조로 매핑됩니다.

매핑 레벨 1.1 이상

매핑 레벨 1.1은 CICS TS 3.1 리전 이상과 호환 가능합니다.

이 맵핑 레벨은 특히 XML 스키마에 다른 값으로 정의된 maxLength 및 minLength 속성이 있는 가변 길이의 데이터를 맵핑할 때 XML 문자 및 2진 데이터 유형의 개선된 맵핑을 제공합니다. 데이터는 다음 방식으로 처리됩니다.

- 16MB를 초과하는 고정 길이의 문자 및 2진 데이터 유형은 PL/I를 제외한 모든 언어에서 컨테이너로 맵핑됩니다. PL/I에서는, 32 767바이트를 초과한 고정 길이 문자 및 2진 데이터 유형이 컨테이너로 맵핑됩니다. 컨테이너의 이름을 저장하기 위해 16 바이트 필드가 언어 구조에 작성됩니다. 런타임에 고정 길이 데이터는 컨테이너에 저장되고 컨테이너 이름은 언어 구조에 놓입니다.

컨테이너는 가변 길이이므로 컨테이너로 맵핑되는 고정 길이 데이터는 XML 스키마 또는 웹 서비스 설명에 지정된 고정 길이에 일치시키기 위해 공백 또는 널(null)로 채우지 않거나 자릅니다. 데이터의 길이가 중요한 경우에는 길이를 확인할 애플리케이션을 쓰거나 CICS 리전에서 유효성 검사를 켤 수 있습니다. SOAP 및 XML 유효성 검사 모두 성능에 상당한 영향을 미칩니다.

- XML 스키마 <list> 및 <union> 데이터 유형은 문자 필드로 맵핑됩니다.
- 스키마 정의 XML 속성은 무시되지 않고 맵핑됩니다. 최대 255개의 속성이 각 XML 요소마다 허용됩니다. 자세한 정보는 XML 속성에 대한 지원을 참조하십시오.
- xsi:nil 속성이 지원됩니다. 자세한 정보는 XML 속성에 대한 지원을 참조하십시오.

맵핑 레벨 1.1만

맵핑 레벨 1.1은 CICS TS 3.1 리전 이상과 호환 가능합니다.

이 맵핑 레벨은 특히 XML 스키마에 다른 값으로 정의된 maxLength 및 minLength 속성이 있는 가변 길이의 데이터를 맵핑할 때 XML 문자 및 2진 데이터 유형의 개선된 맵핑을 제공합니다. 데이터는 다음 방식으로 처리됩니다.

- 가변 길이 2진 데이터 유형은 컨테이너로 맵핑됩니다. 컨테이너의 이름을 저장하기 위해 16바이트 필드가 언어 구조에 작성됩니다. 런타임에 2진 데이터는 컨테이너에 저장되고 컨테이너 이름은 언어 구조에 놓입니다.
- 최대 길이가 32 767바이트를 초과한 가변 길이 문자 데이터 유형은 컨테이너로 맵핑됩니다. 컨테이너의 이름을 저장하기 위해 16바이트 필드가 언어 구조에 작성됩니다. 런타임에 문자 데이터는 컨테이너에 저장되고 컨테이너 이름은 언어 구조에 놓입니다.
- 16MB 미만의 고정 길이 문자 및 2진 데이터 유형은 PL/I를 제외한 모든 언어에서 고정 길이 필드로 맵핑됩니다. PL/I에서는, 32 767 이하의 고정 길이 문자 및 2진 데이터 유형이 고정 길이 필드로 맵핑됩니다.
- CICS는 base64Binary 형식이 아닌 hexBinary 형식의 데이터를 인코딩 및 디코딩합니다. XML 스키마의 Base64Binary 데이터 유형은 언어 구조의 필드로 맵핑됩니다. 필드의 크기는 $4 \times (\text{ceil}(z/3))$ 공식을 사용하여 계산됩니다. 여기서,

- z 는 XML 스키마의 데이터 유형 길이입니다.
- $\text{ceil}(x)$ 는 x 이상의 최소 정수입니다.

z 길이가 24 566바이트를 초과하는 경우 결과적인 언어 구조는 컴파일에 실패합니다. 24 566바이트를 초과하는 base64Binary 데이터가 있으면 맵핑 레벨 1.2를 사용해야 합니다. 맵핑 레벨 1.2에서는 언어 구조의 필드를 사용하는 대신 base64Binary 데이터를 컨테이너로 맵핑할 수 있습니다.

맵핑 레벨 1.0만

맵핑 레벨 1.0은 CICS TS 3.1 리전 이상과 호환 가능합니다.

이후의 맵핑 레벨에서 수정된 다음 제한사항에 유의하십시오.

- DFHSC2LS 및 DFHWS2LS는 XML 스키마의 문자 및 2진 데이터 유형을 언어 구조의 고정 길이 필드로 맵핑합니다. 다음 부분 XML 스키마를 보십시오.

```
<xsd:element name="example">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="33000"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

이 부분 XML 스키마는 다음 예제와 유사한 COBOL 언어 구조에 표시됩니다.

```
15 example      PIC X(33000)
```

- CICS는 base64Binary 형식이 아닌 hexBinary 형식의 데이터를 인코딩 및 디코딩합니다. DFHSC2LS 및 DFHWS2LS는 Base64Binary 데이터를 고정 길이 문자 필드로 맵핑하며 이 내용은 애플리케이션을 통해 인코딩 또는 디코딩해야 합니다.
- DFHSC2LS 및 DFHWS2LS는 처리 중 XML 속성을 무시합니다.
- DFHLS2SC 및 DFHLS2WS는 언어 구조의 문자 및 2진 필드를 고정 길이 필드로 해석하고 이 필드를 maxLength 속성이 있는 XML 요소로 맵핑합니다. 런타임에 사용 가능한 데이터가 충분하지 않은 경우 언어 구조의 필드를 공백 또는 널(null)로 채웁니다.

관련 참조:

543 페이지의 『XML 속성에 대한 지원』

XML 스키마는 XML에 필요하거나 허용된 속성을 지정할 수 있습니다. CICS 지원 유틸리티 DFHWS2LS 및 DFHSC2LS는 기본적으로 XML 속성을 무시합니다.

XML 스키마에 정의된 XML 속성을 처리하려면 **MAPPING-LEVEL** 매개변수의 값이 1.1 이상이어야 합니다.

502 페이지의 『CICS 지원 사용 시의 데이터 맵핑 제한사항』

CICS는 WSDL 버전 1.1 또는 2.0에 맞춘 WSDL 문서 또는 XML 스키마와 고급 언어 구조 간의 양방향 데이터 맵핑을 지원하지만, 일정 제한사항이 있습니다. 이 제한사항은 DFHWS2LS 및 DFHSC2LS 도구에만 적용되며 맵핑 레벨에 따라 다릅니다.

고급 언어 및 XML 스키마 맵핑

CICS 지원을 사용하여 고급 언어 구조와 XML 스키마 또는 WSDL 문서 간의 맵핑을 생성합니다. CICS 지원은 고급 언어 데이터 구조로부터 XML 스키마 또는 WSDL 문서를 생성하기도 합니다(반대의 경우도 마찬가지).

유틸리티 프로그램 DFHSC2LS 및 DFHLS2SC를 총괄하여 CICS XML 지원이라 합니다. 유틸리티 프로그램 DFHWS2LS 및 DFHLS2WS를 총괄하여 CICS 웹 서비스 지원이라 합니다.

- DFHLS2SC 및 DFHLS2WS는 고급 언어 구조를 각각 XML 스키마 및 WSDL 문서로 맵핑합니다.
- DFHSC2LS 및 DFHWS2LS는 XML 스키마와 WSDL 문서를 고급 언어 구조로 맵핑합니다.

두 개의 맵핑은 대칭적이지 않습니다.

- DFHLS2SC 또는 DFHLS2WS로 언어 데이터 구조를 처리한 후 결과적인 XML 스키마나 WSDL 문서를 보완 유틸리티 프로그램(각각 DFHSC2LS 또는 DFHWS2LS)으로 처리하는 경우 최종 데이터 구조가 원래와 같을 것이라 예상하지 마십시오. 하지만 최종 데이터 구조는 논리적으로 원래 구조와 동등합니다.
- DFHSC2LS 또는 DFHWS2LS로 XML 스키마나 WSDL 문서를 처리한 후 결과적인 언어 구조를 보완 유틸리티 프로그램(각각 DFHLS2SC 또는 DFHLS2WS)으로 처리하는 경우 최종 XML 스키마나 WSDL 문서의 XML 스키마가 원래와 같을 것이라 예상하지 마십시오.
- 일부 경우 DFHSC2LS 및 DFHWS2LS는 DFHLS2SC 및 DFHLS2WS에서 지원되지 않는 언어 구조를 생성합니다.

CICS가 지원하는 언어 컴파일러에 구현된 대로, 언어 규칙에 따라 DFHLS2SC 및 DFHLS2WS로 처리한 언어 구조를 코딩해야 합니다.

관련 태스크:

563 페이지의 『언어 구조에서 맵핑 생성』

애플리케이션 데이터에서 XML을 작성(또는 반대로)하기 위해서는 런타임에 CICS가 데이터와 XML을 변환할 방식을 설명하는 맵핑을 작성합니다. 어느 애플리케이션 데이터 레코드에서나 시작할 수 있습니다. 예를 들어, COMMAREA, VSAM 파일, 임시 저장영역 큐 또는 DB2 레코드로 시작할 수 있습니다.

CICS 자원 사용 시의 데이터 맵핑 제한사항

CICS는 WSDL 버전 1.1 또는 2.0에 맞춘 WSDL 문서 또는 XML 스키마와 고급 언어 구조 간의 양방향 데이터 맵핑을 지원하지만, 일정 제한사항이 있습니다. 이 제한사항은 DFHWS2LS 및 DFHSC2LS 도구에만 적용되며 맵핑 레벨에 따라 다릅니다.

모든 맵핑 레벨의 제한사항

- 리터럴 인코딩을 사용하는 SOAP 바인딩만 지원됩니다. 따라서 use 속성을 literal 값으로 설정해야 합니다. use="encoded"는 지원되지 않습니다.
- 데이터 유형 정의는 XML 스키마 정의 언어(XSD)를 사용하여 인코딩해야 합니다. 스키마에서 SOAP 메시지에 사용된 데이터 유형을 명시적으로 선언해야 합니다.
- 웹 서비스 설명의 일부 키워드 길이가 제한됩니다. 예를 들어, 조작, 바인딩, 파트 이름은 255자로 제한됩니다. 일부 경우에는 최대 조작 이름 길이가 약간 더 짧을 수도 있습니다.
- 웹 서비스 설명에 정의된 SOAP 결함은 무시됩니다. 서비스 제공자 애플리케이션이 SOAP 결함 메시지를 보내게 하려면 **EXEC CICS SOAPFAULT** 명령을 사용하십시오.
- DFHWS2LS 및 DFHSC2LS는 특정 범위에서 단일 <xsd:any> 요소만 지원합니다. 예를 들어, 다음 스키마 단편은 지원되지 않습니다.

```
<xsd:sequence>
  <xsd:any/>
  <xsd:any/>
</xsd:sequence>
```

여기서, <xsd:any>는 필요에 따라 minOccurs 및 maxOccurs를 지정할 수 있습니다. 예를 들어, 다음 스키마 단편이 지원됩니다.

```
<xsd:sequence>
  <xsd:any minOccurs="2" maxOccurs="2"/>
</xsd:sequence>
```

- 순환 참조는 지원되지 않습니다. 예를 들어, 유형 A에 유형 B가 포함되어 있고 유형 B에 유형 A가 포함된 경우입니다.
- <xsd:choice>, <xsd:sequence>, <xsd:group> 또는 <xsd:all> 요소와 같은 그룹 요소에서는 반복이 지원되지 않습니다. 예를 들어, 다음 스키마 단편은 지원되지 않습니다.

```
<xsd:choice maxOccurs="2">
  <xsd:element name="name1" type="string"/>
</xsd:choice>
```

맵핑 레벨 2.1 이상에서는 예외로, maxOccurs="1" 및 minOccurs="0"이 이 요소에서 지원됩니다.

- DFHSC2LS 및 DFHWS2LS는 xsi:type 속성 또는 대체 그룹의 XML 스키마의 선언된 데이터 유형 및 요소에서 파생된 SOAP 메시지의 데이터 유형과 요소를 지원하지 않습니다. 맵핑 레벨 2.2 이상에서 상위 요소나 유형이 추상으로 정의된 경우에는 예외입니다.
- <xsd:choice> 요소 내의 임베디드 <xsd:sequence> 및 <xsd:group> 요소는 맵핑 레벨 2.2 이전에 지원되지 않습니다. <xsd:choice> 내의 임베디드 <xsd:choice> 및 <xsd:all> 요소는 항상 지원되지 않습니다.

맵핑 레벨 1.1 이상의 개선된 지원

맵핑 레벨이 1.1 이상일 때 DFHWS2LS는 다음 XML 요소 및 요소 유형에 대한 지원을 제공합니다.

- <xsd:list> 요소.
- <xsd:union> 요소.
- xsd:anySimpleType 유형.
- <xsd:attribute> 요소. 맵핑 레벨 1.0에서 이 요소가 무시됩니다.

맵핑 레벨 2.1 이상의 개선된 지원

맵핑 레벨이 2.1 이상일 때 DFHWS2LS는 다음 XML 요소 및 요소 속성을 지원합니다.

- <xsd:any> 요소.
- xsd:anyType 유형.
- 추상 요소. 이전 맵핑 레벨에서는, 추상 요소가 상속 계층의 비터미널 유형으로만 지원되었습니다.
- <xsd:all>, <xsd:choice>, <xsd:sequence> 요소의 maxOccurs 및 minOccurs 속성, maxOccurs="1" 및 minOccurs="0"일 때에만.
- COBOL의 "FILLER" 필드 및 PL/I의 "*" 필드는 비활성화됩니다. 생성된 WSDL에 필드가 표시되지 않고 런타임에 데이터 구조에 해당 값이 남습니다.

맵핑 레벨 2.2 이상의 개선된 지원

맵핑 레벨이 2.2 이상일 때 DFHSC2LS 및 DFHWS2LS는 <xsd:choice> 요소에 최대 250개의 옵션을 지원하는 등, <xsd:choice> 요소에 대한 개선된 지원을 제공합니다. <xsd:choice> 지원에 대한 자세한 정보는 549 페이지의 『<xsd:choice>에 대한 지원』의 내용을 참조하십시오.

맵핑 레벨 2.2 이상에서 CICS 지원은 다음 XML 맵핑을 지원합니다.

- 대체 그룹
- 요소의 고정 값

- 추상 데이터 유형

<xsd:choice> 요소 내의 임베디드 <xsd:sequence> 및 <xsd:group> 요소는 맵핑 레벨 2.2 이상에서 지원됩니다. 예를 들어, 다음 스키마 단편이 지원됩니다.

```
<xsd:choice>
  <xsd:element name="name1" type="string"/>
  <xsd:sequence/>
</xsd:choice>
```

SOAP 메시지의 상위 요소나 유형이 추상으로 정의된 경우 DFHSC2LS 및 DFHWS2LS는 XML 스키마의 선언된 데이터 유형 및 요소에서 파생된 데이터 유형과 요소를 지원합니다.

맵핑 레벨 3.0 이상의 개선된 지원

맵핑 레벨이 3.0 이상일 때 CICS 지원은 다음 맵핑 개선사항을 지원합니다.

- DFHSC2LS 및 DFHWS2LS는 xsd:dateTime 데이터 유형을 CICS ASKTIME 형식으로 맵핑합니다.
- DFHLS2WS는 한 컨테이너 뿐이 아닌 많은 컨테이너를 사용하는 애플리케이션에서 WSDL 문서와 웹 서비스 바인딩을 생성할 수 있습니다.
- 고정 길이 데이터 구조로 설명된 잘린 데이터를 허용합니다. CICS 지원에서 **DATA-TRUNCATION** 매개변수를 사용하여 이 작동을 설정할 수 있습니다.

맵핑 레벨 4.0 이상의 개선된 지원

맵핑 레벨이 4.0 이상일 때 CICS 지원은 다음 맵핑 개선사항을 지원합니다.

맵핑 레벨 4.0 이상에서, DFHLS2SC 및 DFHLS2WS는 COBOL OCCURS DEPENDING ON 절을 지원하고 COBOL 문자 배열을 XML 문자열로 맵핑하도록 지원합니다. CICS 지원에서 **CHAR-OCCURS** 매개변수를 사용하여 이 작동을 설정할 수 있습니다.

- **DATA-TRUNCATION=ENABLED** 매개변수를 지정해야 합니다.
- 복합 OCCURS DEPENDING ON은 지원되지 않습니다. 이 제한사항은 OCCURS DEPENDING ON이 구조의 마지막 필드에 대해서만 지원된다는 의미입니다.
- CICS는 규정된 이름('OF' 키워드 사용)을 OCCURS DEPENDING ON 절의 대상으로 지원하지 않습니다(예를 들어, FIELD1 OF STRUCTURE1).
- CICS는 UNBOUNDED 키워드를 지원하지 않습니다. 애플리케이션이 예상하는 테이블의 최대 크기를 지정해야 합니다.

맵핑 레벨 4.0 이상에서, CICS 웹 서비스는 UTF-16 유니코드를 사용하여 인코딩한 애플리케이션 데이터의 변환을 지원합니다.

- DFHLS2WS 또는 DFHLS2SC를 사용하면 UTF-16에 대한 언어별 데이터 유형을 사용하여 이 작동을 사용 가능하게 할 수 있습니다.
- DFHWS2LS 또는 DFHSC2LS를 사용하면 CCSID=1200을 설정해서 이 작동을 사용 가능하게 할 수 있습니다.
- CICS는 단일 유니코드 코드 페이지, 『IBM Private Use Area의 UTF-16BE』(CCSID 1200)만 지원합니다.
- UTF-8을 사용하여 인코딩한 애플리케이션 데이터의 변환은 지원되지 않습니다.

참고: DFHLS2WS 및 DFHLS2SC는 COBOL GROUP USAGE NATIONAL 절을 지원하지 않습니다.

관련 참조:

495 페이지의 『CICS 지원의 �핑 레벨』

�핑은 언어 구조와 XML 스키마 간에 정보가 변환되는 방식을 지정하는 규칙 세트입니다. 사용 가능한 가장 정교한 �핑을 이용하려면 CICS 지원의 **MAPPING-LEVEL** 매개변수를 최신 레벨로 설정해야 합니다.

COBOL 대 XML 스키마 �핑

DFHLS2SC 및 DFHLS2WS 유틸리티 프로그램은 COBOL 데이터 구조와 XML 스키마 정의 간의 �핑을 지원합니다.

다음 규칙에 따라 COBOL 이름이 XML 이름으로 변환됩니다.

1. 중복 이름은 하나 이상의 숫자를 추가해서 고유 이름이 되게 합니다.

예를 들어 year의 두 인스턴스는 year 및 year1이 됩니다.

2. 하이픈은 밑줄 문자로 바꿉니다. 연속 하이픈 문자열은 연속 밑줄로 바꿉니다.

예를 들어 current-user--id는 current_user_id가 됩니다.

3. 하이픈으로 구분하며 대문자만 포함된 이름의 세그먼트는 소문자로 변환합니다.

예를 들어 CA-REQUEST-ID는 ca_request_id가 됩니다.

4. 숫자로 시작하는 이름은 이름 앞에 밑줄 문자가 추가됩니다.

예를 들어 9A-REQUEST-ID는 _9a_request_id가 됩니다.

CICS는 다음 테이블에 따라 COBOL 데이터 설명 요소를 스키마 요소로 맵핑합니다. 테이블에 표시되지 않은 COBOL 데이터 설명 요소는 DFHLS2SC 또는 DFHLS2WS에서 지원되지 않습니다. 다음 제한사항도 적용됩니다.

- 레벨 번호가 66 및 77인 데이터 설명 항목은 지원되지 않습니다. 레벨 번호가 88인 데이터 설명 항목은 무시됩니다.
- 데이터 설명 항목에 대한 다음 절은 지원되지 않습니다.

REDEFINES

RENAMES. 즉 레벨 66

DATE FORMAT

- 데이터 설명 항목에 대한 다음 절은 무시됩니다.

BLANK WHEN ZERO

JUSTIFIED

VALUE

- SIGN 절 SIGN TRAILING은 지원됩니다. SIGN 절 SIGN LEADING은 DFHLS2SC 또는 DFHLS2WS의 지정된 맵핑 레벨이 1.2 이상일 때에만 지원됩니다.
- SEPARATE CHARACTER는 SIGN TRAILING 및 SIGN LEADING 절 모두, 맵핑 레벨 1.2 이상에서 지원됩니다.
- USAGE 절의 다음 구문은 지원되지 않습니다.

OBJECT REFERENCE

POINTER

FUNCTION-POINTER

PROCEDURE-POINTER

- USAGE 절의 다음 구문은 맵핑 레벨 1.2 이상에서 지원됩니다.

COMPUTATIONAL-1

COMPUTATIONAL-2

- DISPLAY 및 COMPUTATIONAL-5 데이터 설명 항목에 지원되는 유일한 PICTURE 문자는 9, S, Z입니다.
- PACKED-DECIMAL 데이터 설명 항목에 지원되는 PICTURE 문자는 9, S, V, Z입니다.
- 편집한 숫자 데이터 설명 항목에 지원되는 유일한 PICTURE 문자는 9 및 Z입니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정된 경우 문자 배열은 xsd:string으로 맵핑되고 널(null) 종료 문자열로 처리됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 BINARY로 설정되면 문자 배열은 xsd:base64Binary로 맵핑되며 2진 데이터로 처리됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 COLLAPSE로 설정되면 문자열의 후미 공백이 무시됩니다.
- OCCURS DEPENDING ON 절은 맵핑 레벨 4.0 이상에서 지원됩니다. 복합 OCCURS DEPENDING ON은 지원되지 않습니다. 이는 OCCURS DEPENDING ON이 구조의 마지막 필드에 대해서만 지원된다는 의미입니다.
- OCCURS INDEXED BY 절은 어느 맵핑 레벨에서나 지원됩니다.

COBOL 데이터 설명	스키마 simpleType
PIC X(<i>n</i>) PIC A(<i>n</i>) PIC G(<i>n</i>) DISPLAY-1 PIC N(<i>n</i>)	<pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="<i>n</i>"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre>
PIC S9 DISPLAY PIC S99 DISPLAY PIC S999 DISPLAY PIC S9999 DISPLAY	<pre><xsd:simpleType> <xsd:restriction base="xsd:short"> <xsd:minInclusive value="-<i>n</i>"/> <xsd:maxInclusive value="<i>n</i>"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>n</i>은 문자 '9'의 패턴으로 표현 가능한 최대값입니다.
PIC S9(<i>z</i>) DISPLAY 여기서, $5 \leq z \leq 9$	<pre><xsd:simpleType> <xsd:restriction base="xsd:int"> <xsd:minInclusive value="-<i>n</i>"/> <xsd:maxInclusive value="<i>n</i>"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>n</i>은 문자 '9'의 패턴으로 표현 가능한 최대값입니다.
PIC S9(<i>z</i>) DISPLAY 여기서, $9 < z$	<pre><xsd:simpleType> <xsd:restriction base="xsd:long"> <xsd:minInclusive value="-<i>n</i>"/> <xsd:maxInclusive value="<i>n</i>"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>n</i>은 문자 '9'의 패턴으로 표현 가능한 최대값입니다.
PIC 9 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="<i>n</i>"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>n</i>은 문자 '9'의 패턴으로 표현 가능한 최대값입니다.
PIC 9(<i>z</i>) DISPLAY 여기서, $5 \leq z \leq 9$	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="<i>n</i>"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>n</i>은 문자 '9'의 패턴으로 표현 가능한 최대값입니다.
PIC 9(<i>z</i>) DISPLAY 여기서, $9 < z$	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="<i>n</i>"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>n</i>은 문자 '9'의 패턴으로 표현 가능한 최대값입니다.

COBOL 데이터 설명	스키마 simpleType
PIC S9(n) COMP PIC S9(n) COMP-4 PIC S9(n) COMP-5 PIC S9(n) BINARY 여기서 $n \leq 4$.	<pre><xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType></pre>
PIC S9(n) COMP PIC S9(n) COMP-4 PIC S9(n) COMP-5 PIC S9(n) BINARY 여기서, $5 \leq n \leq 9$ 입니다.	<pre><xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType></pre>
PIC S9(n) COMP PIC S9(n) COMP-4 PIC S9(n) COMP-5 PIC S9(n) BINARY 여기서, $9 < n$ 입니다.	<pre><xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType></pre>
PIC 9(n) COMP PIC 9(n) COMP-4 PIC 9(n) COMP-5 PIC 9(n) BINARY 여기서 $n \leq 4$.	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType></pre>
PIC 9(n) COMP PIC 9(n) COMP-4 PIC 9(n) COMP-5 PIC 9(n) BINARY 여기서, $5 \leq n \leq 9$ 입니다.	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType></pre>
PIC 9(n) COMP PIC 9(n) COMP-4 PIC 9(n) COMP-5 PIC 9(n) BINARY 여기서, $9 < n$ 입니다.	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType></pre>
PIC S9(m)V9(n) COMP-3	<pre><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="p"/> <xsd:fractionDigits value="n"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, $p = m + n$입니다.
PIC 9(m)V9(n) COMP-3	<pre><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="p"/> <xsd:fractionDigits value="n"/> <xsd:minInclusive value="0"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, $p = m + n$입니다.

COBOL 데이터 설명	스키마 simpleType
PIC S9(<i>m</i>) COMP-3 DATETIME=PACKED15일 때 맵핑 레벨 3.0에서 지원됨	<pre><xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType></pre> 시간 소인 형식은 CICS ABSTIME입니다.
PIC S9(<i>m</i>)V9(<i>n</i>) DISPLAY 맵핑 레벨 1.2 이상에서 지원됨	<pre><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="p"/> <xsd:fractionDigits value="n"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, $p = m + n$입니다.
COMP-1 맵핑 레벨 1.2 이상에서 지원됨 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 COMP-1 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.	<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>
COMP-2 맵핑 레벨 1.2 이상에서 지원됨 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:double 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 COMP-2 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.	<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>
<i>data description</i> OCCURS <i>n</i> TIMES	<pre><xsd:element name="field-name" minOccurs="n" maxOccurs="n"> ... </xsd:element></pre> 요소의 내용은 사용된 데이터 유형에 따라 다릅니다.
<i>data description</i> OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDENT ON <i>t</i> 맵핑 레벨 4.0에서 지원됨	<pre><xsd:element name="field-name" minOccurs="n" maxOccurs="m"> ... </xsd:element></pre>

COBOL 데이터 설명	스키마 simpleType
PIC X OCCURS <i>n</i> TIMES PIC A OCCURS <i>n</i> TIMES PIC G DISPLAY-1 OCCURS <i>n</i> TIMES PIC N OCCURS <i>n</i> TIMES	CHAR-OCCURS=STRING일 때: <pre><xsd:element name="field-name" > <xsd:simpleType> <xsd:restriction base="s:string"> <xsd:maxLength value="n"> </xsd:restriction> </xsd:simpleType> </xsd:element></pre> 문자열입니다. CHAR-OCCURS=ARRAY일 때: <pre><xsd:element name="field-name" minOccurs="n" maxOccurs="n"> <xsd:simpleType> <xsd:restriction base="s:string"> <xsd:maxLength value="1"> </xsd:restriction> </xsd:simpleType> </xsd:element></pre> 단일 문자 배열입니다.
PIC X OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDENT ON <i>t</i> PIC A OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDENT ON <i>t</i> PIC G DISPLAY-1 OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDENT ON <i>t</i> PIC N OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDENT ON <i>t</i>	CHAR-OCCURS=STRING일 때: <pre><xsd:element name="field-name" > <xsd:simpleType> <xsd:restriction base="s:string"> <xsd:maxLength value="m"> <xsd:minLength value="n"> </xsd:restriction> </xsd:simpleType> </xsd:element></pre>
PIC N(<i>n</i>) USAGE NATIONAL CHAR-USAGE=NATIONAL 일 때: PIC N(<i>n</i>)	<pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre> 런타임에 CICS는 애플리케이션 데이터 구조 필드를 UTF-16 데이터로 채웁니다.

관련 참조:

511 페이지의 『XML 스키마 대 COBOL 맵핑』

DFHSC2LS 및 DFHWS2LS 유틸리티 프로그램은 XML 스키마 정의와 COBOL 데이터 구조 간의 맵핑을 지원합니다.

555 페이지의 『COBOL의 가변적 반복 내용 처리 방법』

COBOL에서는, 데이터의 각 인스턴스를 처리하는 포인터 산술을 사용하여 가변적으로 반복되는 내용을 처리할 수 없습니다. 기타 프로그래밍 언어에는 이 제한사항이 없습니다. 다음 예제는 웹 서비스 애플리케이션에 대해 가변적으로 반복되는 내용을 COBOL로 처리하는 방법을 보여줍니다.

XML 스키마 대 COBOL 맵핑

DFHSC2LS 및 DFHWS2LS 유틸리티 프로그램은 XML 스키마 정의와 COBOL 데이터 구조 간의 맵핑을 지원합니다.

CICS 지원은 다음 규칙을 사용하여 스키마 요소 이름에서 COBOL 변수의 올바른 고유 이름을 생성합니다.

1. COBOL 예약어에는 'X' 접두부가 붙습니다.

예를 들어, DISPLAY는 XDISPLAY가 됩니다.

2. A-Z, a-z, 0-9 또는 하이픈 이외의 문자는 'X'로 바꿉니다.

예를 들어, monthly_total은 monthlyXtotal이 됩니다. **MAPPING-OVERRIDES** 매개변수를 사용하여 기타 문자의 처리 방식을 변경할 수 있습니다. 예를 들어, UNDERSCORES-AS-HYPHENS 값을 설정하면 XML의 밑줄이 X 대신 하이픈으로 변환됩니다. 따라서 monthly_total이 monthly-total이 됩니다.

3. 마지막 문자가 하이픈이면 'X'로 바꿉니다.

예를 들어, ca-request-는 ca-requestX가 됩니다.

4. 변수에 변동 카디널리티가 있다고 스키마에 지정되어 있고(즉, minOccurs 및 maxOccurs가 xsd:element에 다른 값으로 지정됨), 스키마 요소 이름이 23자를 초과하면 이 이름은 해당 길이에서 잘립니다.

변수에 고정 카디널리티가 있다고 스키마에 지정되고 스키마 요소 이름이 28자를 초과하는 경우 이름이 해당 길이에서 잘립니다.

5. 동일한 범위의 중복된 이름은 이름의 두 번째 및 후속 인스턴스에 하나 또는 두 개의 숫자를 추가해서 이름을 고유하게 합니다.

예를 들어, year의 세 가지 인스턴스는 year, year1, year2가 됩니다.

6. 다섯 개의 문자는 -cont 또는 -num 문자열에 예약되어 있으며, 이 문자열은 변수에 변동 카디널리티가 있다고 스키마에 지정될 때 즉, minOccurs 및 maxOccurs가 다른 값으로 지정될 때 사용됩니다.

자세한 정보는 538 페이지의 『변동 요소 배열』의 내용을 참조하십시오.

7. 속성의 경우에는 이전 규칙이 요소 이름에 적용됩니다. attr- 접두부가 요소 이름에 추가되고 -value 또는 -exist가 그 뒤를 따릅니다. 총 길이가 28자를 초과할 경우에는 요소 이름이 잘립니다. 자세한 정보는 543 페이지의 『XML 속성에 대한 지원』의 내용을 참조하십시오.

nil 가능 속성은 특수 규칙이 있습니다. attr- 접두부가 추가되지만 nil-도 요소 이름의 처음에 추가됩니다. 요소 이름 다음에는 -value가 이어집니다. 총 길이가 28자를 초과할 경우에는 요소 이름이 잘립니다.

결과적인 이름의 총 길이는 30자 이하입니다.

DFHSC2LS 및 DFHWS2LS는 다음 테이블에 따라 지정된 맵핑 레벨을 사용하여 스키마 유형을 COBOL 데이터 설명 요소로 맵핑합니다. 다음 사항에 유의하십시오.

- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정되면 가변 길이 문자 데이터는 널(null) 종료 문자열로 맵핑되고 널 종료 문자에 대한 여분의 문자가 할당됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 YES로 설정된 경우에는 가변 길이 문자 데이터가 관련된 두 개의 요소, 길이 필드 및 데이터 필드로 맵핑됩니다. 예를 들어, 다음과 같습니다.

```
<xsd:simpleType name="VariableStringType">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="1"/>
    <xsd:maxLength value="10000"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="textString" type="tns:VariableStringType"/>
```

은 다음으로 맵핑됩니다.

```
15 textString-length PIC S9999 COMP-5 SYNC
15 textString          PIC X(10000)
```

스키마 단순 유형	COBOL 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 2.0 이하: 지원되지 않음 맵핑 레벨 2.1: 지원됨
<pre><xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.0: 지원되지 않음 맵핑 레벨 1.1 이상: PIC X(255)

스키마 단순 유형	COBOL 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:type" <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>type</i>은 다음 중 하나입니다. - string - normalizedString - token - Name - NMTOKEN - language - NCName - ID - IDREF - ENTITY - hexBinary	모든 맵핑 레벨: PIC X(z)
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>type</i>은 다음 중 하나입니다. - string - normalizedString - token - Name - NMTOKEN - language - NCName - ID - IDREF - ENTITY	맵핑 레벨 4.0 이상에서 CCSID=1200일 때: PIC N(z) USAGE NATIONAL

스키마 단순 유형	COBOL 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:type" </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>type</i>은 다음 중 하나입니다. duration date time gDay gMonth gYear gMonthDay gYearMonth	모든 맵핑 레벨: PIC X(32)
<pre><xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.2 이하: PIC X(32) 맵핑 레벨 2.0 이상: PIC X(40) 맵핑 레벨 3.0 이상 PIC S9(15) COMP-3 형식은 CICS ABSTIME입니다.
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>type</i>은 다음 중 하나입니다. byte unsignedByte	모든 맵핑 레벨: PIC X DISPLAY
<pre><xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: PIC S9999 COMP-5 SYNC 또는 PIC S9999 DISPLAY
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: PIC 9999 COMP-5 SYNC 또는 PIC 9999 DISPLAY
<pre><xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: PIC S9(18) COMP-3

스키마 단순 유형	COBOL 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: PIC S9(9) COMP-5 SYNC 또는 PIC S9(9) DISPLAY
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: PIC 9(9) COMP-5 SYNC 또는 PIC 9(9) DISPLAY
<pre><xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: PIC S9(18) COMP-5 SYNC 또는 PIC S9(18) DISPLAY
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: PIC 9(18) COMP-5 SYNC 또는 PIC 9(18) DISPLAY
<pre><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="m"> <xsd:fractionDigits value="n"> </xsd:fractionDigits> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: PIC 9(p)V9(n) COMP-3 여기서, $p = m - n$입니다.
<pre><xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: PIC X DISPLAY x'00' 값은 false를 의미하고 x'01'은 true를 의미합니다.
<pre><xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType></pre>	맵핑 레벨 1.0: 지원되지 않음 맵핑 레벨 1.1 이상: PIC X(255)
<pre><xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType></pre>	맵핑 레벨 1.0: 지원되지 않음 맵핑 레벨 1.1 이상: PIC X(255)

스키마 단순 유형	COBOL 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType> <xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> </xsd:restriction> </xsd:simpleType></pre> 여기서 길이는 정의되지 않습니다.	맵핑 레벨 1.0: 지원되지 않음 맵핑 레벨 1.1: PIC X(y) 여기서, $y = 4 \times (\text{ceil}(z/3))$입니다. $\text{ceil}(x)$는 x 이상의 최소 정수입니다. 맵핑 레벨 1.2 이상: PIC X(z) 여기서, 길이는 고정 길이입니다. PIC X(16) 여기서, 길이는 정의되지 않습니다. 필드는 2진 데이터를 저장하는 컨테이너의 16바이트 이름을 보유합니다.
<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.1 이하 PIC X(32) 맵핑 레벨 1.2 이상: COMP-1 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 COMP-1 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.
<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.1 이하 PIC X(32) 맵핑 레벨 1.2 이상: COMP-2 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:double 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 COMP-2 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.

테이블에 표시된 일부 스키마 유형은 minInclusive 및 maxInclusive 패킷에 지정된 값에 따라(있는 경우), COMP-5 SYNC 또는 DISPLAY의 COBOL 형식으로 맵핑됩니다.

- 부호 있는 유형(short, int, long)의 경우 다음이 지정되면 DISPLAY가 사용됩니다.

```
<xsd:minInclusive value="-a"/>
<xsd:maxInclusive value="a"/>
```

여기서, *a*는 '9' 문자열입니다.

- 부호 없는 유형(unsignedShort, unsignedInt, unsignedLong)의 경우에는 다음이 지정되면 DISPLAY가 사용됩니다.

```
<xsd:minInclusive value="0"/>
<xsd:maxInclusive value="a"/>
```

여기서, *a*는 '9' 문자열입니다.

다른 모든 값이 지정되거나 값이 지정되지 않은 경우에는 COMP-5 SYNC가 사용됩니다.

관련 참조:

505 페이지의 『COBOL 대 XML 스키마 맵핑』

DFHLS2SC 및 DFHLS2WS 유틸리티 프로그램은 COBOL 데이터 구조와 XML 스키마 정의 간의 맵핑을 지원합니다.

555 페이지의 『COBOL의 가변적 반복 내용 처리 방법』

COBOL에서는, 데이터의 각 인스턴스를 처리하는 포인터 산술을 사용하여 가변적으로 반복되는 내용을 처리할 수 없습니다. 기타 프로그래밍 언어에는 이 제한사항이 없습니다. 다음 예제는 웹 서비스 애플리케이션에 대해 가변적으로 반복되는 내용을 COBOL로 처리하는 방법을 보여줍니다.

546 페이지의 『<xsd:any> 및 xsd:anyType에 대한 지원』

DFHWS2LS 및 DFHSC2LS는 XML 스키마의 <xsd:any> 및 xsd:anyType 사용을 지원합니다. <xsd:any> XML 스키마 요소를 사용하여 내용이 정의되지 않은 XML 문서의 섹션을 설명할 수 있습니다. xsd:anyType은 모든 단순 및 복합 데이터 유형이 파생되는 기본 데이터 유형으로, 데이터 내용에 대한 제한사항이나 제한조건이 없습니다.

C 및 C++ 대 XML 스키마 맵핑

DFHLS2SC 및 DFHLS2WS 유틸리티 프로그램은 C 및 C++ 데이터 유형과 XML 스키마 정의 간의 맵핑을 지원합니다.

C 및 C++ 이름은 다음 규칙에 따라 XML 이름으로 변환됩니다.

- XML 요소 이름에 유효하지 않은 문자는 'X'로 바꿉니다.

예를 들어, `monthly-total`은 `monthlyXtotal`이 됩니다.

- 중복 이름은 하나 이상의 숫자를 추가해서 고유 이름이 되게 합니다.

예를 들어 `year`의 두 인스턴스는 `year` 및 `year1`이 됩니다.

DFHLS2SC 및 DFHLS2WS는 다음 테이블에 따라 C 및 C++ 데이터 유형을 스키마 요소로 맵핑합니다. 테이블에 표시되지 않은 C 및 C++ 유형은 DFHLS2SC 또는 DFHLS2WS에서 지원되지 않습니다. 구조에 대한 `_Packed` 규정자가 지원됩니다. 다음과 같은 제한사항이 적용됩니다.

- 헤더 파일은 최상위 레벨 `struct` 인스턴스를 포함해야 합니다.
- 자체를 멤버로 포함하는 구조 유형은 선언할 수 없습니다.
- 다음 C 및 C++ 데이터 유형은 지원되지 않습니다.

```
decimal
long double
wchar_t(C++만 해당)
```

- 헤더 파일에 다음 문자가 표시되는 경우 무시됩니다.

저장영역 클래스 지정자:

```
auto
register
static
extern
mutable
```

규정자

```
const
volatile
_Export(C++만 해당)
```

함수 지정자

```
inline(C++만 해당)
virtual(C++만 해당)
```

초기값

- 헤더 파일에 다음 항목이 포함되면 안됩니다.

공용체

클래스 선언

Enumeration 데이터 유형

포인터 유형 변수

템플릿 선언

사전 정의된 매크로. 즉, 이름이 두 개의 밑줄 문자(`_`)로 시작하고 끝나는 매크로

행 연속 시퀀스(줄 바꾸기 문자 바로 앞에 있는 \ 기호)

프로토타입 함수 선언자

프리프로세서 지시문

비트 필드

`__cdecl`(또는 `_cdecl`) 키워드(C++만 해당)

- 애플리케이션 프로그래머는 `int`가 4바이트로 맵핑되도록 32비트 컴파일러를 사용해야 합니다.
- 다음 C++ 예약 키워드는 지원되지 않습니다.

`explicit`

`using`

`namespace`

`typename`

`typeid`

- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정된 경우 문자 배열은 `xsd:string`으로 맵핑되고 널(null) 종료 문자열로 처리됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 BINARY로 설정되면 문자 배열은 `xsd:base64Binary`로 맵핑되며 2진 데이터로 처리됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 COLLAPSE로 설정된 경우에는 문자열에 대해 `<xsd:whiteSpace value="collapse"/>`가 생성됩니다.

C 및 C++ 데이터 유형	스키마 simpleType
<code>char[z]</code>	<pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType></pre>
<code>char16_t[n]</code>	맵핑 레벨 4.0 이상: <pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre> 런타임에 CICS는 애플리케이션 데이터 구조 필드를 UTF-16 데이터로 채웁니다.
<code>char[8]</code> DATETIME=PACKED15일 때 맵핑 레벨 3.0 이상에서 지원됨	<pre><xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType></pre> 시간 소인 형식은 CICS ABSTIME입니다.

C 및 C++ 데이터 유형	스키마 simpleType
char	<pre><xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType></pre>
unsigned char	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType></pre>
short	<pre><xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType></pre>
unsigned short	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType></pre>
int long	<pre><xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType></pre>
unsigned int unsigned long	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType></pre>
long long	<pre><xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType></pre>
unsigned long long	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType></pre>
bool (C++만 해당)	<pre><xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType></pre>
float 맵핑 레벨 1.2 이상에서 지원됨	<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre> 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 float 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.

C 및 C++ 데이터 유형	스키마 simpleType
double 맵핑 레벨 1.2 이상에서 지원됨	<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre> 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:double 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 double 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.

관련 참조:

『XML 스키마 대 C 및 C++ 맵핑』

DFHSC2LS 및 DFHWS2LS 유틸리티 프로그램은 각 웹 서비스 설명에 포함된 XML 스키마 정의와 C 및 C++ 데이터 유형 간의 맵핑을 지원합니다.

XML 스키마 대 C 및 C++ 맵핑

DFHSC2LS 및 DFHWS2LS 유틸리티 프로그램은 각 웹 서비스 설명에 포함된 XML 스키마 정의와 C 및 C++ 데이터 유형 간의 맵핑을 지원합니다.

CICS 지원은 다음 규칙을 사용하여 스키마 요소 이름에서 C 및 C++ 변수의 올바른 고유 이름을 생성합니다.

1. A - Z, a - z, 0 - 9 또는 _ 이외의 문자는 'X'로 바꿉니다.

예를 들어, monthly-total은 monthlyXtotal이 됩니다.

2. 첫 번째 문자가 영문자가 아니면 선행 'X'로 바꿉니다.

예를 들어, _monthlysummary는 Xmonthlysummary가 됩니다.

3. 스키마 요소 이름이 50자를 초과하면 해당 길이에서 잘립니다.
4. 동일한 범위의 중복 이름은 하나 이상의 숫자를 추가해서 고유 이름이 되게 합니다.

예를 들어 year의 두 인스턴스는 year 및 year1이 됩니다.

5. 다섯 개의 문자는 _cont 또는 _num 문자열에 예약되어 있으며, 이 문자열은 변수에 변동 카디널리티가 있다고 스키마에 지정될 때 즉, minOccurs 및 maxOccurs가 xsd:element에 지정될 때 사용됩니다.

자세한 정보는 538 페이지의 『변동 요소 배열』의 내용을 참조하십시오.

6. 속성의 경우에는 이전 규칙이 요소 이름에 적용됩니다. attr_ 접두부가 요소 이름에 추가되고 _value 또는 _exist가 그 뒤를 따릅니다. 총 길이가 28자를 초과할 경우에는 요소 이름이 잘립니다.

nil 가능 속성은 특수 규칙이 있습니다. attr_ 접두부가 추가되지만 nil_도 요소 이름의 처음에 추가됩니다. 요소 이름 다음에는 _value가 이어집니다. 총 길이가 28자를 초과할 경우에는 요소 이름이 잘립니다.

결과적인 이름의 총 길이는 57자 이하입니다.

DFHSC2LS 및 DFHWS2LS는 다음 테이블에 따라 스키마 유형을 C 및 C++ 데이터 유형으로 매핑합니다. 다음 규칙도 적용됩니다.

- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정되면 가변 길이 문자 데이터는 널(null) 종료 문자열로 매핑되고 널 종료 문자에 대한 여분의 문자가 할당됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 YES로 설정된 경우에는 가변 길이 문자 데이터가 관련된 두 개의 요소, 길이 필드 및 데이터 필드로 매핑됩니다.

스키마 simpleType	C 및 C++ 데이터 유형
<pre><xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType></pre>	매핑 레벨 2.0 이하: 지원되지 않음 매핑 레벨 2.1 이상: 지원됨
<pre><xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType></pre>	매핑 레벨 1.0: 지원되지 않음 매핑 레벨 1.1 이상: char[255]

스키마 simpleType	C 및 C++ 데이터 유형
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>type</i>은 다음 중 하나입니다. - string - normalizedString - token - Name - NMTOKEN - language - NCName - ID - IDREF - ENTITY - hexBinary	모든 맵핑 레벨: char[z]
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>type</i>은 다음 중 하나입니다. - string - normalizedString - token - Name - NMTOKEN - language - NCName - ID - IDREF - ENTITY	맵핑 레벨 4.0 이상에서 CCSID=1200일 때: char16_t[z]

스키마 simpleType	C 및 C++ 데이터 유형
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>type</i>은 다음 중 하나입니다. - duration - date - decimal - time - gDay - gMonth - gYear - gMonthDay - gYearMonth	모든 맵핑 레벨: char[32]
<pre><xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.2 이하: char[32] 맵핑 레벨 2.0 이상: char[40] 맵핑 레벨 3.0 이상 char[8] 시간 소인 형식은 CICS ABSTIME입니다.
<pre><xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: signed char
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: char
<pre><xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: short
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: unsigned short
<pre><xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: char[33]
<pre><xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: int

스키마 simpleType	C 및 C++ 데이터 유형
<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType></code>	모든 맵핑 레벨: unsigned int
<code><xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType></code>	모든 맵핑 레벨: long long
<code><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType></code>	모든 맵핑 레벨: unsigned long long
<code><xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType></code>	모든 맵핑 레벨: bool(C++만 해당) short(C만 해당)
<code><xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType></code>	맵핑 레벨 1.0: 지원되지 않음 맵핑 레벨 1.1 이상: char[255]
<code><xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType></code>	맵핑 레벨 1.0: 지원되지 않음 맵핑 레벨 1.1 이상: char[255]
<code><xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType> <xsd:simpleType> <xsd:restriction base="xsd:base64binary"> </xsd:restriction> </xsd:simpleType></code> 여기서 길이는 정의되지 않음	맵핑 레벨 1.1 이하 char[y] 여기서 $y = 4 \times (\text{ceil}(z/3))$ 입니다. $\text{ceil}(x)$ 은 x 이상의 최소 정수입니다. 맵핑 레벨 1.2 이상: char[z] 여기서, 길이는 고정 길이입니다. char[16] 은 길이가 정의되지 않았을 때 2진 데이터를 저장하는 컨테이너의 이름입니다.

스키마 simpleType	C 및 C++ 데이터 유형
<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.1 이하 char[32] 맵핑 레벨 1.2 이상: float(*) 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 float 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.
<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.0 이하: char[32] 맵핑 레벨 1.2 이상: double(*) 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:double 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 double 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.

관련 참조:

517 페이지의 『C 및 C++ 대 XML 스키마 맵핑』

DFHLS2SC 및 DFHLS2WS 유틸리티 프로그램은 C 및 C++ 데이터 유형과 XML 스키마 정의 간의 맵핑을 지원합니다.

546 페이지의 『<xsd:any> 및 xsd:anyType에 대한 지원』

DFHWS2LS 및 DFHSC2LS는 XML 스키마의 <xsd:any> 및 xsd:anyType 사용을 지원합니다. <xsd:any> XML 스키마 요소를 사용하여 내용이 정의되지 않은 XML 문서의 섹션을 설명할 수 있습니다. xsd:anyType은 모든 단순 및 복합 데이터 유형이 파생되는 기본 데이터 유형으로, 데이터 내용에 대한 제한사항이나 제한조건이 없습니다.

PL/I 대 XML 스키마 매핑

DFHLS2SC 및 DFHLS2WS 유틸리티 프로그램은 PL/I 데이터 구조와 XML 스키마 정의 간의 매핑을 지원합니다. Enterprise PL/I 컴파일러와 이전 PL/I 컴파일러가 다르므로 두 개의 언어 옵션인 PLI-ENTERPRISE 및 PLI-OTHER가 지원됩니다.

PL/I 이름은 다음 규칙에 따라 XML 이름으로 변환됩니다.

1. XML 요소 이름에 유효하지 않은 문자는 'x'로 바꿉니다.

예를 들어, `monthly$total`은 `monthlyxtotal`이 됩니다.

2. 중복 이름은 하나 이상의 숫자를 추가해서 고유 이름이 되게 합니다.

예를 들어 `year`의 두 인스턴스는 `year` 및 `year1`이 됩니다.

DFHLS2SC 및 DFHLS2WS는 다음 테이블에 따라 PL/I 데이터 유형을 스키마 요소로 매핑합니다. 테이블에 표시되지 않은 PL/I 유형은 DFHLS2SC 또는 DFHLS2WS에서 지원되지 않습니다. 다음 제한사항도 적용됩니다.

- COMPLEX 속성이 있는 데이터 항목은 지원되지 않습니다.
- FLOAT 속성이 있는 데이터 항목은 매핑 레벨 1.2 이상에서 지원됩니다. Enterprise PL/I FLOAT IEEE는 지원되지 않습니다.
- VARYING 및 VARYINGZ 순수 DBCS 문자열은 매핑 레벨 1.2 이상에서 지원됩니다.
- DECIMAL(p,q)로 지정된 데이터 항목은 $p \geq q$ 일 때에만 지원됩니다.
- BINARY(p,q)로 지정된 데이터 항목은 $q = 0$ 일 때에만 지원됩니다.
- 데이터 항목에 PRECISION 속성이 지정된 경우 무시됩니다.
- PICTURE 문자열은 지원되지 않습니다.
- ORDINAL 데이터 항목은 FIXED BINARY(7) 데이터 유형으로 처리됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정된 경우 문자 배열은 `xsd:string`으로 매핑되고 널(null) 종료 문자열로 처리되며, 이 매핑은 Enterprise PL/I에 적용되지 않습니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 BINARY로 설정되면 문자 배열은 `xsd:base64Binary`로 매핑되며 2진 데이터로 처리됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 COLLAPSE로 설정된 경우에는 문자열에 대해 `<xsd:whiteSpace value="collapse"/>`가 생성됩니다.

DFHLS2SC 및 DFHLS2WS는 PL/I의 채우기 알고리즘을 완전히 구현하지 않으므로 데이터 구조에서 명시적으로 채우기 바이트를 선언해야 합니다. DFHLS2SC 및 DFHLS2WS는 채우기 바이트가 누락된 사실을 발견하면 메시지를 발행합니다. 각 최

상위 레벨 구조는 2배 단어 경계에서 시작하고 구조의 각 바이트는 올바른 경계로 맵핑되어야 합니다. 다음 코드 단편을 고려하십시오.

```
3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);
```

이 예제에서,

- FIELD1은 1바이트 길이며 모든 경계에 맞출 수 있습니다.
- FIELD2는 4바이트 길이며 전자 경계에 맞춰야 합니다.
- FIELD3은 8바이트 길이며 2배 단어 경계에 맞춰야 합니다.

Enterprise PL/I 컴파일러는 다음과 같은 순서로 필드를 맞춥니다.

1. FIELD3은 경계 요구사항이 가장 강력하므로 먼저 맞춥니다.
2. FIELD2는 FIELD3 바로 앞의 전자 경계에 맞춥니다.
3. FIELD1은 FIELD3 바로 앞의 바이트 경계에 맞춥니다.

마지막으로 전체 구조가 전자 경계에 맞춰지도록 컴파일러는 FIELD1 바로 앞에 채우기 3바이트를 삽입합니다.

DFHLS2WS가 해당 채우기 바이트를 삽입하지 않으므로 DFHLS2WS가 구조를 처리하기 전에 직접 이를 명시적으로 선언해야 합니다. 예를 들어, 다음과 같습니다.

```
3 PAD1    FIXED BINARY(7),
3 PAD2    FIXED BINARY(7),
3 PAD3    FIXED BINARY(7),
3 FIELD1  FIXED BINARY(7),
3 FIELD2  FIXED BINARY(31),
3 FIELD3  FIXED BINARY(63);
```

또는 모든 필드를 맞추지 않음으로 선언하도록 구조를 변경하고 이 구조를 사용하는 애플리케이션을 다시 컴파일할 수도 있습니다. PL/I 구조 메모리 맞추기 요구사항에 대한 자세한 정보는 *Enterprise PL/I Language Reference*를 참조하십시오.

PL/I 데이터 설명	스키마
FIXED BINARY (<i>n</i>) 여기서 $n \leq 7$	<pre><xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType></pre>
FIXED BINARY (<i>n</i>) 여기서, $8 \leq n \leq 15$	<pre><xsd:simpleType> <xsd:restriction base="xsd:short"/> </xsd:simpleType></pre>
FIXED BINARY (<i>n</i>) 여기서, $16 \leq n \leq 31$	<pre><xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType></pre>
FIXED BINARY (<i>n</i>) 여기서, $32 \leq n \leq 63$ 제한사항: Enterprise PL/I만 해당	<pre><xsd:simpleType> <xsd:restriction base="xsd:long"/> </xsd:simpleType></pre>

PL/I 데이터 설명	스키마
UNSIGNED FIXED BINARY(n) 여기서 $n \leq 8$ 제한사항: Enterprise PL/I만 해당	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"/> </xsd:simpleType></pre>
UNSIGNED FIXED BINARY(n) 여기서, $9 \leq n \leq 16$ 제한사항: Enterprise PL/I만 해당	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"/> </xsd:simpleType></pre>
UNSIGNED FIXED BINARY(n) 여기서, $17 \leq n \leq 32$ 제한사항: Enterprise PL/I만 해당	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"/> </xsd:simpleType></pre>
UNSIGNED FIXED BINARY(n) 여기서, $33 \leq n \leq 64$ 제한사항: Enterprise PL/I만 해당	<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"/> </xsd:simpleType></pre>
FIXED DECIMAL(n, m)	<pre><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="n"/> <xsd:fractionDigits value="m"/> </xsd:restriction> </xsd:simpleType></pre>
FIXED DECIMAL(15) DATETIME=PACKED15일 때 맵핑 레벨 3.0 이상에서 지원됨	<pre><xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType></pre> 시간 소인 형식은 CICS ABSTIME입니다.
BIT(n) 여기서, n 은 8의 배수이며 기타 값은 지원되 지 않습니다.	<pre><xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="m"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, $m = n/8$
CHARACTER(n) VARYING 및 VARYINGZ는 맵핑 레벨 1.2 이 상에서도 지원됩니다. 제한사항: VARYINGZ는 Enterprise PL/I에서 만 지원됩니다.	<pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre>
GRAPHIC(n) VARYING 및 VARYINGZ는 맵핑 레벨 1.2 이 상에서도 지원됩니다. 제한사항: VARYINGZ는 Enterprise PL/I에서 만 지원됩니다.	맵핑 레벨 1.0 및 1.1에서, 여기서 $m = 2*n$: <pre><xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="m"/> </xsd:restriction> </xsd:simpleType></pre> 맵핑 레벨 1.2 이상에서: <pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre>

PL/I 데이터 설명	스키마
WIDECCHAR(<i>n</i>) 제한사항: Enterprise PL/I만 해당	맵핑 레벨 1.0 및 1.1에서, 여기서 $m = 2 * n$: <pre><xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="m"/> </xsd:restriction> </xsd:simpleType></pre> 맵핑 레벨 1.2 이상에서: <pre><xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="n"/> </xsd:restriction> </xsd:simpleType></pre> 맵핑 레벨 4.0 이상에서, CICS는 애플리케이션 데이터 구조 필드를 UTF-16 데이터로 채웁니다. <pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre>
ORDINAL 제한사항: Enterprise PL/I만 해당	<pre><xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType></pre>
BINARY FLOAT(<i>n</i>) 여기서, $n \leq 21$ 맵핑 레벨 1.2 이상에서 지원됩니다.	<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>
BINARY FLOAT(<i>n</i>) 여기서, $21 < n \leq 53$ 53보다 큰 값은 지원되지 않습니다. 맵핑 레벨 1.2 이상에서 지원됩니다.	<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>
DECIMAL FLOAT(<i>n</i>) 여기서, $n \leq 6$ 맵핑 레벨 1.2 이상에서 지원됩니다. 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 DECIMAL FLOAT 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.	<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>

PL/I 데이터 설명	스키마
DECIMAL FLOAT(<i>n</i>) 여기서, $6 < n \leq 16$ 16보다 큰 값은 지원되지 않습니다. 맵핑 레벨 1.2 이상에서 지원됩니다. 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:double 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 DECIMAL FLOAT 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.	<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>

관련 참조:

『XML 스키마 대 PL/I 맵핑』

DFHSC2LS 및 DFHWS2LS 유틸리티 프로그램은 XML 스키마 정의와 PL/I 데이터 구조 간의 맵핑을 지원합니다. Enterprise PL/I 컴파일러와 이전 PL/I 컴파일러가 다르므로 두 개의 언어 옵션인 PLI-ENTERPRISE 및 PLI-OTHER가 지원됩니다.

XML 스키마 대 PL/I 맵핑

DFHSC2LS 및 DFHWS2LS 유틸리티 프로그램은 XML 스키마 정의와 PL/I 데이터 구조 간의 맵핑을 지원합니다. Enterprise PL/I 컴파일러와 이전 PL/I 컴파일러가 다르므로 두 개의 언어 옵션인 PLI-ENTERPRISE 및 PLI-OTHER가 지원됩니다.

CICS 지원은 다음 규칙을 사용하여 스키마 요소 이름에서 PL/I 변수의 올바른 고유 이름을 생성합니다.

1. A-Z, a-z, 0-9, @, # 또는 \$ 이외의 문자는 'X'로 바꿉니다.

예를 들어, monthly-total은 monthlyXtotal이 됩니다.

2. 변수에 변동 카디널리티가 있다고 스키마에 지정되어 있고(즉, minOccurs 및 maxOccurs 속성이 xsd:element에 다른 값으로 지정됨), 스키마 요소 이름이 24자를 초과하면 이 이름은 해당 길이에서 잘립니다.

변수에 고정 카디널리티가 있다고 스키마에 지정되고 스키마 요소 이름이 29자를 초과하는 경우 이름이 해당 길이에서 잘립니다.

3. 동일한 범위의 중복된 이름은 이름의 두 번째 및 후속 인스턴스에 하나 이상의 숫자를 추가해서 이름을 고유하게 합니다.

예를 들어, year의 세 가지 인스턴스는 year, year1, year2가 됩니다.

4. 다섯 개의 문자는 `_cont` 또는 `_num` 문자열에 예약되어 있으며, 이 문자열은 변수에 변동 카디널리티가 있다고 스키마에 지정될 때 즉, `minOccurs` 및 `maxOccurs` 속성이 다른 값으로 지정될 때 사용됩니다.

자세한 정보는 538 페이지의 『변동 요소 배열』의 내용을 참조하십시오.

5. 속성의 경우에는 이전 규칙이 요소 이름에 적용됩니다. `attr-` 접두부가 요소 이름에 추가되고 `-value` 또는 `-exist`가 그 뒤를 따릅니다. 총 길이가 28자를 초과할 경우에는 요소 이름이 잘립니다. 자세한 정보는 543 페이지의 『XML 속성에 대한 지원』의 내용을 참조하십시오.

`nil` 가능 속성은 특수 규칙이 있습니다. `attr-` 접두부가 추가되지만 `nil-`도 요소 이름의 처음에 추가됩니다. 요소 이름 다음에는 `-value`가 이어집니다. 총 길이가 28자를 초과할 경우에는 요소 이름이 잘립니다.

결과적인 이름의 총 길이는 31자 이하입니다.

DFHSC2LS 및 DFHWS2LS는 다음 테이블에 따라 스키마 유형을 PL/I 데이터 유형으로 매핑합니다. 다음 사항도 유의하십시오.

- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정되면 가변 길이 문자 데이터는 널(null) 종료 문자열로 매핑되고 널 종료자에 대한 여분의 문자가 할당됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 지정되지 않으면 기본적으로 가변 길이 문자 데이터는 Enterprise PL/I의 경우 **VARYINGZ** 데이터 유형으로 그리고 기타 PL/I의 경우에는 **VARYING** 데이터 유형으로 매핑됩니다.
- 가변 길이 2진 데이터는 32 768바이트 미만인 경우 **VARYING** 데이터 유형으로 그리고 32 768바이트를 초과하는 경우에는 컨테이너로 매핑됩니다.

스키마	PL/I 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType></pre>	매핑 레벨 2.0 이하: 지원되지 않음 매핑 레벨 2.1 이상: 지원됨
<pre><xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType></pre>	매핑 레벨 1.1 이상:CHAR(255)

스키마	PL/I 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> <xsd:maxLength value="z"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>type</i>은 다음 중 하나입니다. - string - normalizedString - token - Name - NMTOKEN - language - NCName - ID - IDREF - ENTITY	모든 맵핑 레벨: CHARACTER(z)
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>type</i>은 다음 중 하나입니다. - string - normalizedString - token - Name - NMTOKEN - language - NCName - ID - IDREF - ENTITY	맵핑 레벨 4.0 이상에서 CCSID=1200일 때: WIDECHAR(z)

스키마	PL/I 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:type"> </xsd:restriction> </xsd:simpleType></pre> 여기서, <i>type</i>은 다음 중 하나입니다. duration date time gDay gMonth gYear gMonthDay gYearMonth	모든 맵핑 레벨:CHAR(32)
<pre><xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.2 이하: CHAR(32) 맵핑 레벨 2.0 이상: CHAR(40) 맵핑 레벨 3.0 이상 FIXED DECIMAL(15) 시간 소인 형식은 CICS ABSTIME입니다.
<pre><xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="y"/> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.1 이하 BIT(<i>z</i>) 여기서, $z = 8 \times y$ 및 $z < 4095$바이트입니다. CHAR(<i>z</i>) 여기서, $z = 8 \times y$ 및 $z > 4095$바이트입니다. 맵핑 레벨 1.2 이상: CHAR(<i>y</i>)
<pre><xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: Enterprise PL/I SIGNED FIXED BINARY (7) 기타 PL/I FIXED BINARY (7)
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: Enterprise PL/I UNSIGNED FIXED BINARY (8) 기타 PL/I FIXED BINARY (8)

스키마	PL/I 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: Enterprise PL/I SIGNED FIXED BINARY (15) 기타 PL/I FIXED BINARY (15)
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: Enterprise PL/I UNSIGNED FIXED BINARY (16) 기타 PL/I FIXED BINARY (16)
<pre><xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: Enterprise PL/I FIXED DECIMAL(31,0) 기타 PL/I FIXED DECIMAL(15,0)
<pre><xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: Enterprise PL/I SIGNED FIXED BINARY (31) 기타 PL/I FIXED BINARY (31)
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: Enterprise PL/I UNSIGNED FIXED BINARY(32) 기타 PL/I BIT(64)
<pre><xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.1 이하 Enterprise PL/I SIGNED FIXED BINARY(63) 맵핑 레벨 1.2 이상: Enterprise PL/I CHAR(y) 여기서, y는 16MB 미만의 고정 길이입니다. 모든 맵핑 레벨: 기타 PL/I BIT(64)

스키마	PL/I 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.1 이하 Enterprise PL/I UNSIGNED FIXED BINARY(64) 맵핑 레벨 1.2 이상: Enterprise PL/I CHAR(<i>y</i>) 여기서, <i>y</i>는 16MB 미만의 고정 길이입니다. 모든 맵핑 레벨: 기타 PL/I BIT(64)
<pre><xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.1 이하 Enterprise PL/I SIGNED FIXED BINARY (7) 기타 PL/I FIXED BINARY (7) 맵핑 레벨 1.2 이상: Enterprise PL/I BIT(7) BIT(1) 기타 PL/I BIT(7) BIT(1) 여기서, BIT(7)은 맞추기를 위해 제공되고 BIT(1)은 부울 맵핑 값을 포함합니다.
<pre><xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="n"/> <xsd:fractionDigits value="m"/> </xsd:restriction> </xsd:simpleType></pre>	모든 맵핑 레벨: FIXED DECIMAL(<i>n</i> , <i>m</i>)
<pre><xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType></pre>	모든 맵핑 레벨: CHAR(255)
<pre><xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType></pre>	모든 맵핑 레벨: CHAR(255)

스키마	PL/I 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="y"/> </xsd:restriction> </xsd:simpleType></pre> 여기서 길이는 정의되지 않음	맵핑 레벨 1.0: 지원되지 않음 맵핑 레벨 1.1: CHAR(<i>z</i>) 여기서, $z = 4 \times (\text{ceil}(y/3))$입니다. $\text{ceil}(x)$은 x 이상의 최소 정수입니다. 맵핑 레벨 1.2 이상: CHAR(<i>y</i>) 여기서, 길이는 고정 길이입니다. CHAR(16) 여기서, 길이는 정의되지 않습니다. 필드는 2진 데이터를 저장하는 컨테이너의 16바이트 이름을 보유합니다.
<pre><xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.0 및 1.1: CHAR(32) 맵핑 레벨 1.2 이상: Enterprise PL/I DECIMAL FLOAT(6) HEXADEC 기타 PL/I DECIMAL FLOAT(6) 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 DECIMAL FLOAT 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.

스키마	PL/I 데이터 설명
<pre><xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpleType></pre>	맵핑 레벨 1.0 및 1.1: CHAR(32) 맵핑 레벨 1.2 이상: Enterprise PL/I DECIMAL FLOAT(16) HEXADEC 기타 PL/I DECIMAL FLOAT(16) 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 XML에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 xsd:double 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 DECIMAL FLOAT 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.

관련 참조:

527 페이지의 『PL/I 대 XML 스키마 맵핑』

DFHLS2SC 및 DFHLS2WS 유틸리티 프로그램은 PL/I 데이터 구조와 XML 스키마 정의 간의 맵핑을 지원합니다. Enterprise PL/I 컴파일러와 이전 PL/I 컴파일러가 다르므로 두 개의 언어 옵션인 PLI-ENTERPRISE 및 PLI-OTHER가 지원됩니다.

546 페이지의 『<xsd:any> 및 xsd:anyType에 대한 지원』

DFHWS2LS 및 DFHSC2LS는 XML 스키마의 <xsd:any> 및 xsd:anyType 사용을 지원합니다. <xsd:any> XML 스키마 요소를 사용하여 내용이 정의되지 않은 XML 문서의 섹션을 설명할 수 있습니다. xsd:anyType은 모든 단순 및 복합 데이터 유형이 파생되는 기본 데이터 유형으로, 데이터 내용에 대한 제한사항이나 제한조건이 없습니다.

변동 요소 배열

XML은 가변적인 수의 요소가 있는 배열을 포함할 수 있습니다. 일반적으로 가변적인 수의 요소가 있는 WSDL 문서 및 XML 스키마는 단일 고급 언어 데이터 구조로 효과적으로 맵핑되지 않습니다. CICS는 컨테이너 기반 맵핑이나 인라인 맵핑을 사용하여 XML에서 가변적인 수의 요소를 처리합니다.

가변적인 수의 요소가 있는 배열은 요소 선언에 minOccurs 및 maxOccurs 속성을 사용하여 XML 스키마에 표현됩니다.

- minOccurs 속성은 요소가 발생할 수 있는 최소 횟수를 지정합니다. 가능한 값은 0 또는 양의 정수입니다.

- `maxOccurs` 속성은 요소가 발생할 수 있는 최대 횟수를 지정합니다. 가능한 값은 `minOccurs` 속성 값 이상의 양의 정수입니다. 요소가 발생할 수 있는 횟수에 상한을 적용하지 않음을 나타내는 `unbounded` 값도 사용할 수 있습니다.
- 두 속성 모두 기본값은 1입니다.

이 예제는 선택적인 8바이트 문자열을 보여줍니다. 애플리케이션 XML 또는 SOAP 메시지에 결코 발생하지 않거나 한 번 발생할 수 있습니다.

```
<xsd:element name="component" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

다음 예제는 적어도 한 번은 발생해야 하는 8바이트 문자열을 보여줍니다.

```
<xsd:element name="component" minOccurs="1" maxOccurs="unbounded">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

일반적으로 가변적인 수의 요소가 있는 WSDL 문서는 단일 고급 언어 데이터 구조로 효과적으로 맵핑되지 않습니다. 따라서 이러한 경우를 처리하기 위해 CICS는 일련의 컨테이너에 있는 애플리케이션으로 전달되는 일련의 연결된 데이터 구조를 사용합니다. 이 구조는 애플리케이션의 입력 및 출력으로 사용됩니다.

- CICS는 XML을 애플리케이션 데이터로 변환할 때 애플리케이션 데이터로 이 구조를 채우며 애플리케이션은 이 데이터를 읽습니다.
- CICS는 애플리케이션 데이터를 XML로 변환할 때 구조에서 애플리케이션이 채운 애플리케이션 데이터를 읽습니다.

이 데이터 구조의 형식은 일련의 예제에 가장 잘 설명되어 있습니다. XML은 SOAP 메시지가 애플리케이션에서 나올 수 있습니다. 다음 예제는 단순 8바이트 필드의 배열을 사용합니다. 하지만 모델은 복합 데이터 유형 배열 및 기타 배열을 포함한 데이터 유형 배열을 지원합니다.

고정된 수의 요소

첫 번째 예제는 정확히 세 번 발생하는 요소를 보여줍니다.

```
<xsd:element name="component" minOccurs="3" maxOccurs="3">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
```

```

        <xsd:length value="8"/>
    </xsd:restriction>
</xsd:simpleType>
</xsd:element>

```

이 예제에서는, 요소가 발생하는 횟수가 미리 알려져 있어서 단순 COBOL 선언에서 고정 길이 배열로(또는 다른 언어의 동등한 표시로) 이를 표현할 수 있습니다.

```
05 component PIC X(8) OCCURS 3 TIMES
```

맵핑 레벨 2 이하에서 가변적인 수의 요소

이 예제는 1 - 5번 발생할 수 있는 필수 요소를 보여줍니다.

```

<xsd:element name="component" minOccurs="1" maxOccurs="5">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

```

기본 데이터 구조는 두 필드의 선언을 포함합니다. CICS가 XML을 2진 데이터로 변환할 때 첫 번째 필드 component-num은 XML에 요소가 표시되는 횟수를 포함하고, 두 번째 필드 component-cont는 컨테이너의 이름을 포함합니다.

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

두 번째 데이터 구조는 요소 선언 자체를 포함합니다.

```
01 DFHWS-component
  02 component PIC X(8)
```

component-num의 값(1 - 5 범위의 값을 포함할)을 검토해서 요소가 발생하는 횟수를 파악해야 합니다. 요소 내용은 component-cont의 이름 지정된 컨테이너에 있고, 컨테이너는 요소의 배열을 보유합니다. 각 요소는 DFHWS-component 데이터 구조를 통해 맵핑됩니다.

minOccurs="0" 및 maxOccurs="1"인 경우 요소는 선택적입니다. 애플리케이션에서 데이터 구조를 처리하려면 component-num의 값을 검토해야 합니다.

- 이 값이 0인 경우 메시지에 구성요소 요소가 없고 component-cont의 내용이 정의되지 않습니다.
- 1인 경우에는 구성요소 요소가 component-cont의 이름 지정된 컨테이너에 있습니다.

컨테이너의 내용은 DFHWS-component 데이터 구조를 통해 맵핑됩니다.

참고: SOAP 메시지가 단일 반복 요소로 이루어진 경우 DFHWS2LS는 두 가지 언어 구조를 생성합니다. 기본 언어 구조는 배열의 요소 수와 요소 배열을 보유한 컨테이너의 이름을 포함합니다. 두 번째 언어 구조는 반복 요소의 단일 인스턴스를 맵핑합니다.

맵핑 레벨 2.1 이상에서 가변적인 수의 요소

맵핑 레벨 2.1 이상에서는, CICS 지원의 **INLINE-MAXOCCURS-LIMIT** 매개변수를 사용할 수 있습니다. **INLINE-MAXOCCURS-LIMIT** 매개변수는 가변적인 수의 요소가 처리되는 방식을 지정합니다. 가변적인 수의 요소에 대한 맵핑 옵션은 540 페이지의 『맵핑 레벨 2 이하에서 가변적인 수의 요소』에 설명된 컨테이너 기반 맵핑 또는 인라인 맵핑입니다. 이 매개변수의 가능한 *value*는 0 - 32767 범위의 양의 정수입니다.

- **INLINE-MAXOCCURS-LIMIT**의 기본값은 선택적 요소를 인라인 맵핑하는 1입니다.
- **INLINE-MAXOCCURS-LIMIT** 매개변수의 0 값은 인라인 맵핑을 차단합니다.
- maxOccurs가 **INLINE-MAXOCCURS-LIMIT**의 값 이하인 경우에는 인라인 맵핑이 사용됩니다.
- maxOccurs가 **INLINE-MAXOCCURS-LIMIT**의 값보다 크면 컨테이너 기반 맵핑이 사용됩니다.

가변적인 수의 요소를 인라인 맵핑하면 위의 고정 발생 예제 및 카운터의 경우처럼, 두 배열이 모두 생성됩니다. component-num 필드는 존재하는 요소의 인스턴스 수를 표시하며 이는 배열을 통해 지정됩니다. 540 페이지의 『맵핑 레벨 2 이하에서 가변적인 수의 요소』에 표시된 예제의 경우 **INLINE-MAXOCCURS-LIMIT**가 5 이하이면 생성된 데이터 구조는 다음과 같습니다.

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

첫 번째 필드, component-num은 이전 섹션의 컨테이너 기반 맵핑 예제의 출력과 동일합니다. 두 번째 필드는 생성 가능한 최대 수의 요소를 충분히 포함할 수 있는 길이 5의 배열을 포함합니다.

인라인 맵핑은 모든 데이터를 현재 컨테이너에 저장하기 때문에, 요소의 발생 수와 데이터가 위치한 컨테이너의 이름을 저장하는 컨테이너 기반 맵핑과 차이가 있습니다. 데이터를 현재 컨테이너에 저장하면 일반적으로 성능이 개선되며 인라인 맵핑이 더 바람직하게 됩니다.

중첩 변동 배열

복잡한 WSDL 문서 및 XML 스키마는 가변적으로 반복되는 요소를 포함할 수 있으며, 이 요소는 차례로 가변적으로 반복되는 요소를 포함합니다. 이 경우 설명된 구조는 예제에 설명된 두 레벨을 넘어서 확장됩니다.

다음 예제는 <component1>이라는 필수 요소에 중첩된 <component2>라는 선택적 요소를 보여주며, 여기서 필수 요소는 1- 5회 발생할 수 있습니다.

```
<xsd:element name="component1" minOccurs="1" maxOccurs="5">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="component2" minOccurs="0" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:length value="8"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

최상위 레벨 데이터 구조는 이전 예제와 정확히 동일합니다.

```
05 component1-num PIC S9(9) COMP-5
05 component1-cont PIC X(16)
```

하지만 두 번째 데이터 구조는 다음 요소를 포함합니다.

```
01 DFHWS-component1
  02 component2-num PIC S9(9) COMP-5
  02 component2-cont PIC X(16)
```

세 번째 레벨 구조는 다음 요소를 포함합니다.

```
01 DFHWS-component2
  02 component2 PIC X(8)
```

가장 바깥쪽 요소 <component1>의 발생 수는 component1-num에 있습니다.

component1-cont의 이름 지정된 컨테이너는 두 번째 데이터 구조 DFHWS-component1의 인스턴스 수가 있는 배열을 포함합니다.

component2-cont의 각 인스턴스는 상이한 컨테이너의 이름을 지정하며, 각각 세 번째 레벨 구조 DFHWS-component2를 통해 맵핑되는 데이터 구조를 포함합니다.

이 구조를 설명하려면 예제에 일치하는 XML의 단편을 고려하십시오.

```
<component1><component2>string1</component2></component1>
<component1><component2>string2</component2></component1>
<component1></component1>
```

<component1>은 세 번 발생합니다. 처음 두 개는 각각 <component2>의 인스턴스를 포함하지만 세 번째 인스턴스는 그렇지 않습니다.

최상위 레벨 데이터 구조에서 component1-num은 값 3을 포함합니다. component1-cont의 이름 지정된 컨테이너에는 DFHWS-component1의 세 가지 인스턴스가 있습니다.

1. 첫 번째에서, component2-num은 값이 1이고 component2-cont의 이름 지정된 컨테이너가 *string1*을 보유합니다.
2. 두 번째에서, component2-num은 값이 1이고 component2-cont의 이름 지정된 컨테이너가 *string2*를 보유합니다.
3. 세 번째에서, component2-num은 값이 0이고 component2-cont의 내용이 정의되지 않습니다.

이 인스턴스에서 완전한 데이터 구조는 모두 네 개의 컨테이너로 표현됩니다.

- 컨테이너 DFHWS-DATA의 루트 데이터 구조
- component1-cont의 이름 지정된 컨테이너
- component2-cont의 처음 두 인스턴스에 이름 지정된 두 개의 컨테이너

선택적 구조 및 **xsd:choice**

DFHWS2LS 및 DFHSC2LS는 맵핑 레벨 2.1 이상에서만 **<xsd:sequence>**, **<xsd:choice>**, **<xsd:all>** 요소에 maxOccurs 및 minOccurs를 사용하도록 지원합니다. 여기서 minOccurs 및 maxOccurs 속성은 minOccurs="0" 및 maxOccurs="1"로 설정됩니다.

지원은 이 요소를 각 하위 요소가 선택적인 것처럼 처리하는 맵핑을 생성합니다. 이 요소로 애플리케이션을 구현할 때에는 애플리케이션이 옵션의 올바르지 않은 조합을 생성하지 않는지 확인하십시오. 각 요소마다 생성된 언어 구조에 자체 count 필드가 있으며, 이 필드를 모두 "0" 또는 "1"로 설정해야 합니다. **<xsd:choice>** 요소의 경우를 제외하면, 다른 값 조합은 올바르지 않습니다.

<xsd:choice> 요소는 요소의 옵션 중 하나만을 사용할 수 있음을 나타냅니다. 이는 모든 맵핑 레벨에서 지원됩니다. 지원은 **<xsd:choice>**의 각 옵션을 minOccurs="0" 및 maxOccurs="1"인 **<xsd:sequence>** 요소에 있는 것처럼 처리합니다. **<xsd:choice>** 요소를 사용하여 애플리케이션을 구현할 때에는 애플리케이션이 옵션의 올바르지 않은 조합을 생성하지 않는지 확인하도록 주의하십시오. 각 요소마다 생성된 언어 구조에 자체 count 필드가 있으며, 이 중에서 정확히 한 필드는 '1'로 설정하고 다른 필드는 모두 '0'으로 설정해야 합니다. **<xsd:choice>** 요소 자체가 선택적인(이 경우 모든 필드를 '0'으로 설정하는 것이 올바름) 경우를 제외하면 다른 모든 값 조합은 올바르지 않습니다.

XML 속성에 대한 지원

XML 스키마는 XML에 필요하거나 허용된 속성을 지정할 수 있습니다. CICS 지원 유틸리티 DFHWS2LS 및 DFHSC2LS는 기본적으로 XML 속성을 무시합니다. XML 스키마에 정의된 XML 속성을 처리하려면 **MAPPING-LEVEL** 매개변수의 값이 1.1 이상이여야 합니다.

선택적 속성

속성은 선택적이거나 필수적이며 애플리케이션의 XML 또는 SOAP 메시지의 요소와 연관시킬 수 있습니다. 스키마에 정의된 모든 선택적 속성의 경우 다음 두 개의 필드가 해당 언어 구조에 생성됩니다.

1. 존재 플래그. 이 필드는 Boolean 데이터 유형으로 처리되며 일반적으로 1바이트 길이입니다.
2. 값. 이 필드는 동등 유형의 XML 요소와 동일한 방식으로 매핑됩니다. 예를 들어, NMTOKEN 유형의 속성은 NMTOKEN 유형의 XML 요소와 동일한 방식으로 매핑됩니다.

속성 존재 및 값 필드는 생성된 언어 구조에서 연관된 요소의 필드 이전에 표시됩니다. 인스턴스 문서에 표시되는 예기치 않은 속성은 무시됩니다.

예를 들어, 다음 스키마 속성 정의를 고려하십시오.

```
<xsd:attribute name="age" type="xsd:short" use="optional" />
```

이 선택적 속성은 다음 COBOL 구조로 매핑됩니다.

```
05 attr-age-exist PIC X DISPLAY
05 attr-age-value PIC S9999 COMP-5 SYNC
```

선택적 속성의 런타임 처리

선택적 속성에 대한 다음 런타임 처리가 발생합니다.

- 속성이 존재하면 존재 플래그가 설정되고 값이 매핑됩니다.
- 속성이 존재하지 않으면 존재 플래그가 설정되지 않습니다.
- 속성이 기본값이 있고 존재할 경우 값이 매핑됩니다.
- 속성이 기본값이 있고 존재하지 않으면 기본값이 매핑됩니다.

기본값이 있는 선택적 속성은 필수 속성으로 간주됩니다.

CICS가 데이터를 XML로 변환할 때에는 다음 런타임 처리가 발생합니다.

- 존재 플래그가 설정된 경우 속성이 변환되어 XML에 포함됩니다.
- 존재 플래그가 설정되지 않았으면 속성이 XML에 포함되지 않습니다.

필수 속성 및 런타임 처리

필요한 모든 속성은 해당 언어 구조에 값 필드만 생성됩니다.

속성이 XML에 존재하면 값이 매핑됩니다. 속성이 없는 경우에는 다음 처리가 발생합니다.

- 애플리케이션이 웹 서비스 제공자인 경우 CICS는 클라이언트 SOAP 메시지의 오류를 나타내는 SOAP 결함 메시지를 생성합니다.

- 애플리케이션이 웹 서비스 요청자이면 CICS는 메시지를 발행하고 RESP2 코드 13의 변환 오류 응답을 애플리케이션에 리턴합니다.
- 애플리케이션이 **TRANSFORM XMLTODATA** 명령을 사용 중인 경우에는 CICS가 메시지를 발행하고 RESP2 코드 3의 올바른지 않은 요청 응답을 애플리케이션에 리턴합니다.

CICS가 COMMAREA 또는 컨테이너의 내용을 바탕으로 SOAP 메시지를 생성할 때 속성이 변환되어 메시지에 포함됩니다. 애플리케이션이 **TRANSFORM DATATOXML** 명령을 사용할 때 CICS도 속성을 변환하여 XML에 포함시킵니다.

nil 가능 속성

nil 가능 속성은 XML 스키마의 `xsd:element`에 표시될 수 있는 특수 속성입니다. `xsi:nil` 속성이 XML의 요소에 유효함을 지정합니다. 요소에 `xsi:nil` 속성이 지정될 경우 이는 요소가 존재하지만 값이 없어서 연관된 내용도 없음을 나타냅니다.

XML 스키마가 nil 가능 속성을 `true`로 정의한 경우 이 속성은 부울 값이 있는 필수 속성으로 매핑됩니다.

CICS가 SOAP 메시지를 수신하거나 `xsi:nil` 속성을 포함한 애플리케이션에 대한 XML을 변환해야 하는 경우 속성의 값은 `true` 또는 `false`입니다. 값이 `true`이면 애플리케이션은 `xsi:nil` 속성 범위의 중첩된 요소나 요소의 값을 무시해야 합니다.

CICS가 `xsi:nil` 속성의 값이 `true`인 컨테이너나 COMMAREA의 내용을 바탕으로 XML 또는 SOAP 메시지를 생성하는 경우 다음 처리가 발생합니다.

- `xsi:nil` 속성이 XML 또는 SOAP 메시지로 생성됩니다.
- 연관된 요소의 값이 무시됩니다.
- 요소 내의 중첩 요소가 무시됩니다.

SOAP 메시지 예제

WSDL 문서의 일부일 수 있는 다음 예제 XML 스키마를 고려하십시오.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="root" nillable="true">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element nillable="true" name="num" type="xsd:int" maxOccurs="3" minOccurs="3"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

다음은 이 스키마에 맞춘 부분 SOAP 메시지 예제입니다.

```
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <num xsi:nil="true"/>
  <num>15</num>
  <num xsi:nil="true"/>
</root>
```

COBOL에서 이 SOAP 메시지는 다음 요소로 맵핑합니다.

```
05    root
10    attr-nil-root-value    PIC X DISPLAY
10    num                    OCCURS 3
15    num1                   PIC S9(9) COMP-5 SYNC
15    attr-nil-num-value    PIC X DISPLAY
10    filler                 PIC X(3)
```

관련 참조:

495 페이지의 『CICS 지원의 맵핑 레벨』

맵핑은 언어 구조와 XML 스키마 간에 정보가 변환되는 방식을 지정하는 규칙 세트입니다. 사용 가능한 가장 정교한 맵핑을 이용하려면 CICS 지원의 **MAPPING-LEVEL** 매개변수를 최신 레벨로 설정해야 합니다.

<xsd:any> 및 *xsd:anyType*에 대한 지원

DFHWS2LS 및 DFHSC2LS는 XML 스키마의 <xsd:any> 및 *xsd:anyType* 사용을 지원합니다. <xsd:any> XML 스키마 요소를 사용하여 내용이 정의되지 않은 XML 문서의 섹션을 설명할 수 있습니다. *xsd:anyType*은 모든 단순 및 복합 데이터 유형이 파생되는 기본 데이터 유형으로, 데이터 내용에 대한 제한사항이나 제한조건이 없습니다.

<xsd:any> 및 *xsd:anyType*을 CICS 지원으로 사용하려면 다음 매개변수를 설정하십시오.

- **MAPPING-LEVEL** 매개변수를 2.1 이상으로 설정하십시오.
- 웹 서비스 제공자 애플리케이션의 경우에는 **PGMINT** 매개변수를 CHANNEL로 설정하십시오.

<xsd:any> 예제

다음 예제는 <xsd:any> 요소를 사용하여 "Customer" 태그의 "Surname" 태그 다음에서 일부 구조화되지 않은 선택적 XML 내용을 설명합니다.

```
<xsd:element name="Customer">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Title" type="xsd:string"/>
      <xsd:element name="FirstName" type="xsd:string"/>
      <xsd:element name="Surname" type="xsd:string"/>
      <xsd:any minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

이 XML 스키마에 맞는 예제 SOAP 메시지는 다음과 같습니다.

```
<xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Customer xmlns="http://www.example.org/anyExample">
      <Title xmlns="">Mr</Title>
      <FirstName xmlns="">John</FirstName>
      <Surname xmlns="">Smith</Surname>
      <ExtraInformation xmlns="http://www.example.org/ExtraInformation">
        <!-- This 'ExtraInformation' tag is associated with the optional xsd:any from the XML schema.
              It can contain any well formed XML. -->
        <ExampleField1>one</ExampleField1>
        <ExampleField2>two</ExampleField2>
      </ExtraInformation>
    </Customer>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

이 SOAP 메시지가 CICS로 전송되면 CICS는 Customer-xml-cont 컨테이너를 다음 XML 데이터로 채웁니다.

```
<ExtraInformation xmlns="http://www.example.org/ExtraInformation">
  <!-- This 'ExtraInformation' tag is associated with the optional xsd:any from the XML schema.
        It can contain any well formed XML. -->
  <ExampleField1>one</ExampleField1>
  <ExampleField2>two</ExampleField2>
</ExtraInformation>
```

CICS는 Customer-xmlns-cont 컨테이너도 범위에 있는 다음 XML 이름 공간 선언으로 채우며, 이 선언은 공백으로 구분됩니다.

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns="http://www.example.org/anyExample"
```

xsd:anyType 예제

*xsd:anyType*은 모든 단순 및 복합 데이터 유형이 파생되는 기본 데이터 유형입니다. 이 유형은 데이터 내용을 제한하지 않습니다. 데이터 유형을 지정하지 않을 경우 기본적으로 *xsd:anyType*이 사용됩니다. 예를 들어, 다음 두 개의 XML 단편은 동일합니다.

```
<xsd:element name="Name" type="xsd:anyType"/>
<xsd:element name="Name"/>
```

생성된 언어 구조

<xsd:any> 또는 *xsd:anyType*의 생성된 언어 구조는 COBOL에서 다음 양식을 사용하고 다른 언어에서도 동등한 양식을 취합니다.

elementName-xml-cont PIC X(16)

원시 XML을 보유한 컨테이너의 이름입니다. CICS는 수신 SOAP 메시지를 처리할 때 <xsd:any> 또는 *xsd:anyType*이 정의하는 SOAP 메시지의 서브세트

를 이 컨테이너에 둡니다. 애플리케이션은 XML 데이터를 기본적인 방식으로만 처리할 수 있습니다. 애플리케이션은 XML을 생성하고 이 컨테이너를 채운 후 컨테이너 이름을 제공해야 합니다.

이 컨테이너는 텍스트 모드로 채워야 합니다. CICS는 이 컨테이너를 채울 때 웹 서비스에 사용하도록 정의된 동일한 EBCDIC 변형을 사용하여 채웁니다. 대상 EBCDIC 코드 페이지에 존재하지 않는 문자는 대체 문자로 대체됩니다(애플리케이션이 컨테이너를 UTF-8로 읽더라도).

elementName+xmlns-cont PIC X(16)

범위 내의 이름 공간 접두부 선언을 보유한 컨테이너의 이름입니다. 이 컨테이너의 내용은 SOAP 엔벨로프 태그의 서브세트만 포함하기 보다는 범위 내의 관련된 모든 이름 공간 선언을 포함한다는 점을 제외하면, DFHWS-XMLNS 컨테이너의 내용과 유사합니다.

이 컨테이너는 텍스트 모드로 채워야 합니다. CICS는 이 컨테이너를 채울 때 웹 서비스에 사용하도록 정의된 동일한 EBCDIC 변형을 사용하여 채웁니다. 대상 EBCDIC 코드 페이지에 존재하지 않는 문자는 대체 문자로 대체됩니다(애플리케이션이 컨테이너를 UTF-8로 읽더라도).

이 컨테이너는 CICS로 전송된 SOAP 메시지를 처리할 때에만 사용됩니다. 애플리케이션이 출력 SOAP 메시지가 생성될 때 컨테이너에 이름 공간 선언을 제공하려 시도할 경우에는 CICS가 이 컨테이너와 내용을 무시합니다. CICS는 이름 공간 선언에 대해, 애플리케이션이 제공한 XML을 전체적으로 자체 포함시킬 것을 요구합니다.

<xsd:any> 요소를 포함한 XML 요소의 이름은 <xsd:any> 요소의 생성된 변수 이름에 포함됩니다. <xsd:any> 예제에서, <xsd:any> 요소는 <xsd:element name="Customer"> 요소 내에 중첩되며 <xsd:any> 요소의 생성된 변수 이름은 Customer-xml-cont PIC X(16) 및 Customer-xmlns-cont PIC X(16)입니다.

xsd:anyType 유형의 경우에는 직접 XML 요소 이름이 사용됩니다. 위의 xsd:anyType 예제에서 변수 이름은 Name-xml-cont PIC X(16) 및 Name-xmlns-cont PIC X(16)입니다.

관련 태스크:

573 페이지의 『<xsd:any> 데이터 유형 처리』

하나 이상의 <xsd:any> 데이터 유형을 포함한 XML 스키마에 대해 작업 중인 경우 XML 지원은 데이터 유형을 CICS 컨테이너 쌍으로 매핑할 수 있습니다. 컨테이너의 XML을 구문 분석할 애플리케이션을 쓸 수 있습니다.

관련 참조:

511 페이지의 『XML 스키마 대 COBOL 매핑』

DFHSC2LS 및 DFHWS2LS 유틸리티 프로그램은 XML 스키마 정의와 COBOL 데이터 구조 간의 매핑을 지원합니다.

521 페이지의 『XML 스키마 대 C 및 C++ 맵핑』

DFHSC2LS 및 DFHWS2LS 유틸리티 프로그램은 각 웹 서비스 설명에 포함된 XML 스키마 정의와 C 및 C++ 데이터 유형 간의 맵핑을 지원합니다.

531 페이지의 『XML 스키마 대 PL/I 맵핑』

DFHSC2LS 및 DFHWS2LS 유틸리티 프로그램은 XML 스키마 정의와 PL/I 데이터 구조 간의 맵핑을 지원합니다. Enterprise PL/I 컴파일러와 이전 PL/I 컴파일러가 다르므로 두 개의 언어 옵션인 PLI-ENTERPRISE 및 PLI-OTHER가 지원됩니다.

<xsd:choice>에 대한 지원

<xsd:choice> 요소는 요소의 옵션을 하나만 사용할 수 있음을 표시합니다. CICS 지원은 다양한 맵핑 레벨에서 <xsd:choice> 요소에 대한 다양한 수준의 지원을 제공합니다.

맵핑 레벨 2.2 이상의 <xsd:choice>에 대한 지원

맵핑 레벨 2.2 이상에서, DFHWS2LS 및 DFHSC2LS는 <xsd:choice> 요소에 대한 개선된 지원을 제공합니다. 지원은 <xsd:choice> 요소와 연관된 값을 저장하는 새 컨테이너를 생성합니다. 지원은 새 컨테이너의 이름과 여분의 필드를 포함한 언어 구조를 생성합니다.

fieldname-enum

<xsd:choice> 요소에 사용할 옵션을 표시하는 차별화된 필드입니다.

fieldname-cont

사용할 옵션을 저장하는 컨테이너의 이름입니다. 옵션의 값을 맵핑하기 위해 추가 언어 구조가 생성됩니다.

다음 XML 스키마 단편은 <xsd:choice> 요소를 포함합니다.

```
<xsd:element name="choiceExample">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="option1" type="xsd:string" />
      <xsd:element name="option2" type="xsd:int" />
      <xsd:element name="option3" type="xsd:short" maxOccurs="2" minOccurs="2" />
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

이 XML 스키마 단편이 맵핑 레벨 2.2 이상에서 처리되면 지원은 다음 COBOL 언어 구조를 생성합니다.

```
03 choiceExample.
  06 choiceExample-enum          PIC X DISPLAY.
    88 empty                     VALUE X'00'.
    88 option1                   VALUE X'01'.
    88 option2                   VALUE X'02'.
    88 option3                   VALUE X'03'.
  06 choiceExample-cont          PIC X(16).
```

```

01 Example-option1.
   03 option1-length          PIC S9999 COMP-5 SYNC.
   03 option1                 PIC X(255).

01 Example-option2.
   03 option2                 PIC S9(9) COMP-5 SYNC.

01 Example-option3.
   03 option3 OCCURS 2        PIC S9999 COMP-5 SYNC.

```

맵핑 레벨 2.2 이상의 <xsd:choice>에 대한 제한사항

DFHSC2LS 및 DFHWS2LS는 중첩 <xsd:choice> 요소를 지원하지 않습니다. 예를 들어, 다음 XML은 지원되지 않습니다.

```

<xsd:choice>
  <xsd:element name ="name1" type="string"/>
  <xsd:choice>
    <xsd:element name ="name2a" type="string"/>
    <xsd:element name ="name2b" type="string"/>
  </xsd:choice>
</xsd:choice>

```

DFHSC2LS 및 DFHWS2LS는 반복되는 <xsd:choice> 요소를 지원하지 않습니다. 예를 들어, 다음 XML은 지원되지 않습니다.

```

<xsd:choice maxOccurs="2">
  <xsd:element name ="name1" type="string"/>
</xsd:choice>

```

DFHSC2LS 및 DFHWS2LS는 <xsd:choice> 요소에 최대 255개의 옵션을 지원합니다.

맵핑 레벨 2.1 이하의 <xsd:choice>에 대한 지원

맵핑 레벨 2.1 이하에서는, DFHWS2LS가 <xsd:choice> 요소에 대한 제한된 지원을 제공합니다. DFHWS2LS는 <xsd:choice> 요소의 각 옵션을 기껏해야 한 번 발생할 수 있는 <xsd:sequence> 요소인 것처럼 처리합니다.

<xsd:choice> 요소에 옵션을 하나만 사용할 수 있으므로 일반적으로 올바른 옵션 조합만 생성하는 <xsd:choice> 요소를 사용하여 애플리케이션을 구현할 때 주의하십시오. 각 요소마다 생성된 언어 구조에 자체 count 필드가 있으며, 이 중에서 정확히 한 필드는 1로 설정되고 다른 필드는 모두 0으로 설정되어야 합니다. <xsd:choice> 자체가 선택적인(이 경우 모든 필드를 0으로 설정하는 것이 올바른) 경우를 제외하면, 다른 모든 값 조합은 올바르지 않습니다.

관련 참조:

546 페이지의 『<xsd:any> 및 xsd:anyType에 대한 지원』

DFHWS2LS 및 DFHSC2LS는 XML 스키마의 <xsd:any> 및 xsd:anyType 사용을 지원합니다. <xsd:any> XML 스키마 요소를 사용하여 내용이 정의되지 않은 XML 문서의 섹션을 설명할 수 있습니다. xsd:anyType은 모든 단순 및 복합 데이터 유형이 파생되는 기본 데이터 유형으로, 데이터 내용에 대한 제한사항이나 제한조건이 없습니다.

553 페이지의 『추상 요소 및 추상 데이터 유형에 대한 지원』

CICS 지원은 맵핑 레벨 2.2 이상의 추상 요소 및 추상 데이터 유형에 대한 지원을 제공합니다. CICS 지원은 대체 그룹과 유사한 방식으로 추상 요소 및 추상 데이터 유형을 맵핑합니다.

552 페이지의 『대체 그룹에 대한 지원』

대체 그룹을 사용하여 교환 가능한 XML 요소 그룹을 정의할 수 있습니다. CICS 지원은 맵핑 레벨 2.2 이상에서 대체 그룹에 대한 지원을 제공합니다.

<xsd:sequence>에 대한 지원

CICS 지원의 <xsd:sequence> 요소 사용이 지원되지만 많은 제한사항이 있습니다.

<xsd:sequence> 요소에 대한 minOccurs 및 maxOccurs 속성의 사용은 지원되지 않습니다. 이 규칙의 예외는 minOccurs="0" and maxOccurs="1" 또는 minOccurs="1" 및 maxOccurs="1"일 때입니다.

CICS 지원은 동일한 <xsd:sequence> 요소에 이름이 같은 두 개의 요소를 사용하도록 허용하지 않습니다. 예를 들어, {a, b, c, a} 순서는 CICS 지원에서 거부됩니다. 이 제한사항을 피하려면 {a maxOccurs="2", b, c}로 순서를 바꾸십시오.

CICS 지원은 <xsd:any> 요소 하나만 또는 <xsd:any>로 간주되는 한 요소를 동일한 <xsd:sequence> 요소에 허용합니다. <xsd:any> 요소로 간주되는 요소는 대체 그룹이 정의되지 않은 추상 요소를 포함하고 있습니다.

XML을 구문 분석할 때 CICS는 <xsd:sequence> 요소를 <xsd:all> 요소처럼 처리합니다. 이는 CICS가 순서의 항목 순서가 올바른지 확인하지 않는다는 의미입니다. 예를 들어, 스키마가 {a, b, c} 순서를 정의하는 경우 CICS는 a, b 및 c를 어느 순서로든 포함한 XML을 용인합니다. 하지만 CICS가 XML을 생성할 때 <xsd:sequence>의 항목 순서가 항상 보존됩니다.

CICS는 누락된 XML 데이터를 자동으로 발견하지 않습니다. 예를 들어, <xsd:sequence>의 요소가 필수로 정의(minOccurs="1" maxOccurs="1")되었지만 XML 문서에 발생하지 않으면 CICS는 런타임 유효성 검사가 사용되는 경우에만 이 문제를 보고합니다.

대체 그룹에 대한 지원

대체 그룹을 사용하여 교환 가능한 XML 요소 그룹을 정의할 수 있습니다. CICS 지원은 맵핑 레벨 2.2 이상에서 대체 그룹에 대한 지원을 제공합니다.

맵핑 레벨 2.2 이상에서 DFHSC2LS 및 DFHWS2LS는 <xsd:choice> 요소에 사용된 것과 유사한 맵핑을 사용하여 대체 그룹을 지원합니다. 지원은 열거 필드 및 새 컨테이너 이름을 언어 구조에 생성합니다.

다음 XML 스키마 단편은 각 요소를 replacementOption1 또는 replacementOption2와 바꿀 수 있는 두 개의 subGroupParent 요소를 포함합니다.

```
<xsd:element name="subGroupExample" >
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="subGroupParent" maxOccurs="2" minOccurs="2" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="subGroupParent" type="xsd:anySimpleType" />
<xsd:element name="replacementOption1" type="xsd:int" substitutionGroup="subGroupParent" />
<xsd:element name="replacementOption2" type="xsd:short" substitutionGroup="subGroupParent" />
```

이 XML 단편을 지원으로 처리하면 다음 COBOL 언어 구조가 생성됩니다.

```
03 subGroupExample.
  06 subGroupParent OCCURS2.
    09 subGroupExample-enum PIC X DISPLAY.
      88 empty VALUE X '00'.
      88 replacementOption1 VALUE X '01'.
      88 replacementOption2 VALUE X '02'.
      88 subGroupParent VALUE X '03'.
    09 subGroupExample-cont PIC X (16).

01 Example-replacementOption1.
  03 replacementOption1 PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
  03 replacementOption2 PIC S9999 COMP-5 SYNC.

01 Example-subGroupParent.
  03 subGroupParent-length PIC S9999 COMP-5 SYNC.
  03 subGroupParent PIC X(255).
```

대체 그룹에 대한 자세한 정보는 *W3C XML Schema Part 1: Structures Second Edition specification*: http://www.w3.org/TR/xmlschema-1/#Elements_Equivalence_Class를 참조하십시오.

관련 참조:

546 페이지의 『<xsd:any> 및 xsd:anyType에 대한 지원』

DFHWS2LS 및 DFHSC2LS는 XML 스키마의 <xsd:any> 및 xsd:anyType 사용을 지원합니다. <xsd:any> XML 스키마 요소를 사용하여 내용이 정의되지 않은 XML 문서의 섹션을 설명할 수 있습니다. xsd:anyType은 모든 단순 및 복합 데이터 유형이 파생되는 기본 데이터 유형으로, 데이터 내용에 대한 제한사항이나 제한조건이 없습니다.

549 페이지의 『<xsd:choice>에 대한 지원』

<xsd:choice> 요소는 요소의 옵션을 하나만 사용할 수 있음을 표시합니다. CICS 지원은 다양한 매핑 레벨에서 <xsd:choice> 요소에 대한 다양한 수준의 지원을 제공합니다.

『추상 요소 및 추상 데이터 유형에 대한 지원』

CICS 지원은 매핑 레벨 2.2 이상의 추상 요소 및 추상 데이터 유형에 대한 지원을 제공합니다. CICS 지원은 대체 그룹과 유사한 방식으로 추상 요소 및 추상 데이터 유형을 매핑합니다.

추상 요소 및 추상 데이터 유형에 대한 지원

CICS 지원은 매핑 레벨 2.2 이상의 추상 요소 및 추상 데이터 유형에 대한 지원을 제공합니다. CICS 지원은 대체 그룹과 유사한 방식으로 추상 요소 및 추상 데이터 유형을 매핑합니다.

매핑 레벨 2.2 이상의 추상 요소에 대한 지원

매핑 레벨 2.2 이상에서는, 추상 요소가 그룹의 올바른 멤버가 아니라는 점을 제외하면 DFHSC2LS 및 DFHWS2LS가 대체 그룹과 거의 동일한 방식으로 추상 요소를 다룹니다. 대체 요소가 없을 경우 추상 요소가 <xsd:any> 요소로 간주되며 매핑 레벨 2.1의 <xsd:any> 요소와 동일한 매핑을 사용합니다.

다음 XML 스키마 단편은 추상 요소 대신에 사용할 수 있는 두 개의 옵션을 지정합니다. 추상 요소 자체는 올바른 옵션이 아닙니다.

```
<xsd:element name="abstractElementExample" >
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="abstractElementParent" maxOccurs="2" minOccurs="2" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="abstractElementParent" type="xsd:anySimpleType" abstract="true" />
<xsd:element name="replacementOption1" type="xsd:int" substitutionGroup="abstractElementParent" />
<xsd:element name="replacementOption2" type="xsd:short" substitutionGroup="abstractElementParent" />
```

이 XML 단편을 지원으로 처리하면 다음 COBOL 언어 구조가 생성됩니다.

```
03 abstractElementExample.
  06 abstractElementParent OCCURS 2.
    09 abstractElementExample-enum PIC X DISPLAY.
      88 empty VALUE X '00'.
```

```

      88 replacementOption1          VALUE X '01'.
      88 replacementOption2          VALUE X '02'.
09 abstractElementExample-cont PIC X (16).

```

```

01 Example-replacementOption1.
   03 replacementOption1              PIC S9(9) COMP-5 SYNC.

```

```

01 Example-replacementOption2.
   03 replacementOption2              PIC S9999 COMP-5 SYNC.

```

추상 요소에 대한 자세한 정보는 *W3C XML Schema Part 0: Primer Second Edition* 스펙: <http://www.w3.org/TR/xmlschema-0/#SubsGroups>의 내용을 참조하십시오.

매핑 레벨 2.2 이상의 추상 데이터 유형에 대한 지원

매핑 레벨 2.2 이상에서는, DFHSC2LS 및 DFHWS2LS가 추상 데이터 유형을 대체 그룹으로 다룹니다. 지원은 열거 필드 및 새 컨테이너 이름을 언어 구조에 생성합니다.

다음 XML 스키마 단편은 추상 유형 대신에 사용할 수 있는 두 가지 대안을 지정합니다.

```

<xsd:element name="AbstractDataTypeExample" type="abstractDataType" />

<xsd:complexType name="abstractDataType" abstract="true">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string" />
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option1">
  <xsd:simpleContent>
    <xsd:restriction base="abstractDataType">
      <xsd:length value="5" />
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option2">
  <xsd:simpleContent>
    <xsd:restriction base="abstractDataType">
      <xsd:length value="10" />
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>

```

이 XML 단편을 지원으로 처리하면 다음 COBOL 언어 구조가 생성됩니다.

```

03 AbstractDataTypeExamp-enum PIC X DISPLAY.
   88 empty                    VALUE X'00'.
   88 option1                  VALUE X'01'.
   88 option2                  VALUE X'02'.
03 AbstractDataTypeExamp-cont PIC X(16).

```

언어 구조가 별도의 카피북으로 생성됩니다. option1의 생성된 언어 구조가 하나의 카피북으로 생성됩니다.

03 option1 PIC X(5).

option2의 언어 구조는 상이한 카피북으로 생성됩니다.

03 option2 PIC X(10).

추상 데이터 유형에 대한 자세한 정보는 *W3C XML Schema Part 0: Primer Second Edition* 스펙: <http://www.w3.org/TR/xmlschema-0/#SubsGroups>의 내용을 참조하십시오.

관련 참조:

546 페이지의 『<xsd:any> 및 xsd:anyType에 대한 지원』

DFHWS2LS 및 DFHSC2LS는 XML 스키마의 <xsd:any> 및 xsd:anyType 사용을 지원합니다. <xsd:any> XML 스키마 요소를 사용하여 내용이 정의되지 않은 XML 문서의 섹션을 설명할 수 있습니다. xsd:anyType은 모든 단순 및 복합 데이터 유형이 파생되는 기본 데이터 유형으로, 데이터 내용에 대한 제한사항이나 제한조건이 없습니다.

549 페이지의 『<xsd:choice>에 대한 지원』

<xsd:choice> 요소는 요소의 옵션을 하나만 사용할 수 있음을 표시합니다. CICS 지원은 다양한 매핑 레벨에서 <xsd:choice> 요소에 대한 다양한 수준의 지원을 제공합니다.

552 페이지의 『대체 그룹에 대한 지원』

대체 그룹을 사용하여 교환 가능한 XML 요소 그룹을 정의할 수 있습니다. CICS 지원은 매핑 레벨 2.2 이상에서 대체 그룹에 대한 지원을 제공합니다.

COBOL의 가변적 반복 내용 처리 방법

COBOL에서는, 데이터의 각 인스턴스를 처리하는 포인터 산술을 사용하여 가변적으로 반복되는 내용을 처리할 수 없습니다. 기타 프로그래밍 언어에는 이 제한사항이 없습니다. 다음 예제는 웹 서비스 애플리케이션에 대해 가변적으로 반복되는 내용을 COBOL로 처리하는 방법을 보여줍니다.

이 기술은 **TRANSFORM** API 명령을 사용하여 XML을 애플리케이션 데이터로 변환하는 방식에도 적용됩니다. 다음 예제 WSDL 문서는 가변적인 횟수로 반복되는 8자 문자열로 이루어진 애플리케이션 데이터가 있는 웹 서비스를 나타냅니다.

```
<?xml version="1.0"?>
<definitions name="ExampleWSDL"
  targetNamespace="http://www.example.org/variablyRepeatingData/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.example.org/variablyRepeatingData/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <types>
    <xsd:schema targetNamespace="http://www.example.org/variablyRepeatingData/">
      <xsd:element name="applicationData">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="component" minOccurs="1" maxOccurs="unbounded">
```

```

        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:length value="8"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
</types>

<message name="exampleMessage">
  <part element="tns:applicationData" name="messagePart"/>
</message>

<portType name="examplePortType">
  <operation name="exampleOperation">
    <input message="tns:exampleMessage"/>
    <output message="tns:exampleMessage"/>
  </operation>
</portType>

<binding name="exampleBinding" type="tns:examplePortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="exampleOperation">
    <soap:operation soapAction=""/>
    <input><soap:body parts="messagePart" encodingStyle="" use="literal"/></input>
    <output><soap:body parts="messagePart" encodingStyle="" use="literal"/></output>
  </operation>
</binding>
</definitions>

```

이 WSDL 문서를 DFHWS2LS를 통해 처리하면 다음 COBOL 언어 구조가 생성됩니다.

```
03 applicationData.
```

```

06 component-num          PIC S9(9) COMP-5 SYNC.
06 component-cont         PIC X(16).
```

```
01 DFHWS-component.
```

```
03 component              PIC X(8).
```

8자 component 필드는 DFHWS-component라는 별도의 구조에 정의됨에 유의하십시오. 기본 데이터 구조는 applicationData이며 두 개의 필드, component-num 및 component-cont를 포함합니다. component-num 필드는 표시되는 component 데이터의 인스턴스 수를 표시하고, component-cont 필드는 component 필드의 연결된 목록을 보유한 컨테이너의 이름을 표시합니다.

다음 COBOL 코드는 가변적으로 반복되는 데이터 목록을 처리할 한 가지 방법을 보여줍니다. 연계 섹션 배열을 사용하여 데이터의 후속 인스턴스를 처리하며, 각 인스턴스는 DISPLAY 문을 사용하여 표시됩니다.

```

IDENTIFICATION DIVISION.
PROGRAM-ID.    EXVARY.

ENVIRONMENT DIVISION.
```

DATA DIVISION.
WORKING-STORAGE SECTION.

```
* working storage variables
01 APP-DATA-PTR          USAGE IS POINTER.
01 APP-DATA-LENGTH      PIC S9(8) COMP.
01 COMPONENT-PTR        USAGE IS POINTER.
01 COMPONENT-DATA-LENGTH PIC S9(8) COMP.
01 COMPONENT-COUNT      PIC S9(8) COMP-4 VALUE 0.
01 COMPONENT-LENGTH     PIC S9(8) COMP.
```

LINKAGE SECTION.

```
* a large linkage section array
01 BIG-ARRAY PIC X(659999).

* application data structures produced by DFHWS2LS
* this is normally referenced with a COPY statement
01 DFHWS2LS-data.
   03 applicationData.
      06 component-num PIC S9(9) COMP-5 SYNC.
      06 component-cont PIC X(16).

01 DFHWS-component.
   03 component      PIC X(8).
```

PROCEDURE DIVISION USING DFHEIBLK.
A-CONTROL SECTION.
A010-CONTROL.

```
* Get the DFHWS-DATA container
EXEC CICS GET CONTAINER('DFHWS-DATA')
      SET(APP-DATA-PTR)
      FLENGTH(APP-DATA-LENGTH)
END-EXEC
SET ADDRESS OF DFHWS2LS-data TO APP-DATA-PTR

* Get the recurring component data
EXEC CICS GET CONTAINER(component-cont)
      SET(COMPONENT-PTR)
      FLENGTH(COMPONENT-DATA-LENGTH)
END-EXEC

* Point the component structure at the first instance of the data
SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* Store the length of a single component
MOVE LENGTH OF DFHWS-component TO COMPONENT-LENGTH

* process each instance of component data in turn
PERFORM WITH TEST AFTER
      UNTIL COMPONENT-COUNT = component-num

* display the current instance of the data
DISPLAY 'component value is: ' component
```



```

* address the next instance of the component data
  SET ADDRESS OF BIG-ARRAY TO ADDRESS OF DFHWS-component
  SET ADDRESS OF DFHWS-component
    TO ADDRESS OF BIG-ARRAY (COMPONENT-LENGTH + 1:1)
  ADD 1 TO COMPONENT-COUNT

* end the loop
  END-PERFORM.

* Point the component structure back at the first instance of
* of the data, for any further processing we may want to perform
  SET ADDRESS OF DFHWS-component TO COMPONENT-Ptr

* return to CICS.

  EXEC CICS
    RETURN
  END-EXEC

  GOBACK.

```

위의 코드는 가변적으로 반복되는 내용을 처리하기 위한 일반 솔루션을 제공합니다. BIG-ARRAY 배열은 각 구성요소의 시작을 차례로 이동시키며 데이터의 시작 부분에 고정되지 않습니다. 그런 다음 구성요소 데이터 구조가 다음 구성요소의 첫 번째 바이트 지점으로 이동합니다. COMPONENT-Ptr는 필요에 따라 구성요소 데이터의 시작 위치를 복구하는 데 사용할 수 있습니다.

다음은 WSDL 문서에 맞춘 예제 SOAP 메시지입니다.

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <applicationData xmlns="http://www.example.org/variablyRepeatingData/">
      <component xmlns="">VALUE1</component>
      <component xmlns="">VALUE2</component>
      <component xmlns="">VALUE3</component>
    </applicationData>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

다음은 SOAP 메시지를 처리할 때 COBOL 프로그램이 생성하는 출력입니다.

```

CPIH 20080115103151 component value is: VALUE1
CPIH 20080115103151 component value is: VALUE2
CPIH 20080115103151 component value is: VALUE3

```

관련 참조:

505 페이지의 『COBOL 대 XML 스키마 맵핑』

DFHLS2SC 및 DFHLS2WS 유틸리티 프로그램은 COBOL 데이터 구조와 XML 스키마 정의 간의 맵핑을 지원합니다.

511 페이지의 『XML 스키마 대 COBOL 맵핑』

DFHSC2LS 및 DFHWS2LS 유틸리티 프로그램은 XML 스키마 정의와 COBOL 데이터 구조 간의 맵핑을 지원합니다.

가변 길이 값과 공백에 대한 지원

CICS 지원의 설정을 사용하고 XML 스키마에 패킷을 직접 추가해서 가변 길이 값과 공백을 처리하는 방식을 사용자 정의할 수 있습니다.

일반적으로 CICS XML 지원 및 CICS 웹 서비스 지원은 데이터 문자열을 고정 길이 문자 배열로 매핑하며, 이 배열은 공백이나 널(null)로 채워야 합니다. 가변 길이 값을 고정 길이 데이터 배열로 매핑하는 방식은 비능률적이며 저장영역이 낭비됩니다. 데이터의 길이가 가변적이면 매핑의 처리 방식을 사용자 정의하는 것이 좋습니다.

언어 구조에서 XML 스키마나 WSDL 문서로 변환하는 경우 `whiteSpace` 및 `maxLength` 패킷을 XML 스키마에 지정하고 **CHAR-VARYING-LIMIT** 매개변수를 지원에 설정하는 것이 좋습니다.

XML 스키마나 WSDL 문서에서 언어 구조로 변환하는 경우에는 **CHAR-VARYING** 매개변수의 적합한 값을 지원에 설정하는 것이 좋습니다.

참고: 널 문자('x00')는 XML 문서에 올바르지 않습니다. CICS가 구문 분석한 애플리케이션의 널 문자는 문자열의 끝과 값이 잘렸음을 나타내기 위해 표시되는 것입니다. CICS는 애플리케이션 데이터를 생성할 때 **CHAR-VARYING** 매개변수의 값에 따라 생성합니다. 예를 들어, **CHAR-VARYING=NULL** 옵션이 지정되면 CICS가 생성한 가변 길이 문자열은 널 문자로 끝납니다.

XML에서 언어 구조로 가변 길이 값 매핑

XML 스키마에 패킷을 사용하거나 CICS 지원에 특정 매개변수를 지정해서 XML 스키마 또는 WSDL 문서와 언어 구조 간의 매핑이 처리되는 방식을 사용자 정의하십시오.

XML 데이터 유형은 패킷을 사용하여 제한할 수 있습니다. 길이 패킷(`length`, `maxLength`, `minLength`) 및 `whiteSpace` 패킷을 사용하여 XML의 가변 길이 데이터가 처리되는 방식을 사용자 정의하십시오.

length

데이터가 고정 길이임을 지정하는 데 사용됩니다.

maxLength

데이터 유형의 최대 길이를 지정하는 데 사용됩니다. 문자열 기반 데이터 유형에 이 값이 설정되지 않을 경우 최대 길이가 무제한입니다.

minLength

데이터 유형의 최소 길이를 지정하는 데 사용됩니다. 문자열 기반 데이터 유형에 이 값이 설정되지 않을 경우 최소 길이는 0입니다.

whiteSpace

데이터 값을 에워싼 공백의 처리 방식을 지정하는 데 사용됩니다. 공백은 공백, 탭, 줄 바꾸기를 포함합니다. whiteSpace 패킷은 preserve, replace 또는 collapse로 설정할 수 있습니다.

- preserve 값은 데이터 값의 공백을 유지합니다.
- replace 값은 탭이나 줄 바꾸기를 해당하는 수의 공백으로 바꾼다는 의미입니다.
- collapse 값은 선행, 후미, 임베디드 공백을 제거하고 모든 탭, 줄 바꾸기, 연속 공백을 단일 공백 문자로 바꾼다는 의미입니다.

whiteSpace 패킷이 설정되지 않은 경우 공백이 보존됩니다.

XML 스키마 패킷에 대한 자세한 정보는 W3C 권장 스키마 *XML Schema Part 2: Datatypes Second Edition* <http://www.w3.org/TR/xmlschema-2/#facets>를 참조하십시오.

CICS 지원, DFHSC2LS 및 DFHWS2LS의 다음 매개변수를 사용하여 가변 길이 데이터가 XML 스키마에서 언어 구조로 맵핑되는 방식을 변경할 수 있습니다. 이 매개변수는 맵핑 레벨 1.2 이상에서 사용 가능합니다.

DEFAULT-CHAR-MAXLENGTH

XML 스키마나 WSDL 문서에 길이가 암시되지 않은 맵핑에 대한 문자에서 문자 데이터의 기본 배열 길이를 지정합니다. 이 매개변수의 가능한 값은 1 - 2 147 483 647 범위의 양의 정수입니다.

하지만 maxLength 패킷으로 DFHSC2LS 또는 DFHWS2LS가 사용할 최대 문자 길이를 XML 스키마나 WSDL 문서에 직접 지정할 것을 권장합니다. XML 스키마나 WSDL 문서에 직접 최대 길이를 지정하면 모든 문자열 기반 데이터 유형에 글로벌 기본값이 적용된 점과 연관된 문제를 피할 수 있습니다.

CHAR-VARYING-LIMIT

언어 구조로 맵핑되는 가변 길이 문자 데이터의 최대 크기를 지정합니다. 문자 데이터가 이 매개변수에 지정된 값보다 크면 컨테이너로 맵핑되고 생성된 언어 구조에 컨테이너 이름이 사용됩니다. 가능한 값 범위는 0에서 기본값 32 767 바이트까지입니다.

CHAR-VARYING

가변 길이 문자 데이터가 맵핑되는 방법을 지정합니다. 이 매개변수를 지정하지 않을 경우 기본 맵핑은 지정한 언어에 따라 다릅니다. 다음 옵션을 선택할 수 있습니다.

- **CHAR-VARYING=NO**는 가변 길이 문자 데이터가 고정 길이 문자열로 맵핑됨을 지정합니다.

- **CHAR-VARYING=NULL**은 가변 길이 문자 데이터가 널(null) 종료 문자열로 매핑됨을 지정합니다.
- **CHAR-VARYING=YES**는 가변 길이 문자 데이터가 PL/I에서 CHAR VARYING 데이터 유형으로 매핑됨을 지정합니다. COBOL, C, C++ 언어에서는 가변 길이 문자 데이터가 두 개의 관련 요소, 데이터 길이 및 데이터로 이루어진 동등한 표현으로 매핑됩니다.

일반적으로 **CHAR-VARYING=YES**를 설정하면 최상의 성능이 보장됩니다.

언어 구조에서 XML로의 가변 길이 값 매핑

언어 구조와 XML 스키마 또는 WSDL 문서 간의 매핑이 처리되는 방식을 사용자 정의할 수 있습니다. 문자 배열이 생성되는 방식을 변경하려면 DFHLS2SC 또는 DFHLS2WS에서 **CHAR-VARYING** 매개변수를 COLLAPSE 또는 NULL로 설정하십시오.

CHAR-VARYING=NULL 옵션을 설정하면 CICS가 XML을 생성할 때 각 문자 배열의 끝에 널 문자를 추가합니다.

CHAR-VARYING=COLLAPSE 옵션을 설정하면 CICS가 XML을 생성할 때 각 문자 배열의 끝에서 자동으로 후미 공백을 제거합니다. 이 옵션은 매핑 레벨 2.1 이상에서만 사용 가능하며 **CHAR-VARYING=COLLAPSE**는 C 및 C++ 이외의 다른 모든 언어에서 매핑 레벨 2.1 이상의 기본값입니다. XML이 구문 분석될 때 모든 선행, 후미, 임베디드 공백이 제거됩니다.

자세한 정보는 Support for white space and variable length values in CICS web services(Technote)를 참조하십시오.

애플리케이션 데이터의 UTF-16에 대한 지원

CICS 웹 서비스는 UTF-16 인코딩 애플리케이션 데이터를 XML 또는 JSON으로 그리고 XML 또는 JSON을 UTF-16 인코딩 애플리케이션 데이터로 변환하도록 지원합니다. 여러 언어로 데이터를 저장하고 처리해야 하는 경우 UTF-16을 사용하십시오.

CICS SOAP 및 JSON 웹 서비스는 UTF-16 인코딩 애플리케이션 데이터를 XML 또는 JSON으로 그리고 XML 또는 JSON을 UTF-16 인코딩 애플리케이션 데이터로 변환하도록 지원합니다. 유니코드는 시스템이 데이터를 효율적으로 처리할 수 있는 가변 너비의 인코딩 체계입니다.

UTF-16은 각 문자가 2 또는 4바이트로 표현되는 유니코드의 가변 너비 인코딩입니다. CICS 웹 서비스는 IBM Private Use Area의 UTF-16 BE(Big Endian)인 CCSID 1200 데이터를 지원합니다. 이 작동은 모든 지원 언어의 UTF-16 지원과 일치합니다.

UTF-16은 매핑 레벨 4.0 이상에서 지원됩니다. 지원의 매핑 설정을 사용하여 애플리케이션 데이터의 변환 방식을 사용자 정의할 수 있습니다. XML 매핑 레벨에 대한 자

세한 정보는 애플리케이션 개발에서 CICS 지원자에 대한 맵핑 레벨의 내용을 참조하십시오. JSON 맵핑 레벨에 대한 자세한 정보는 애플리케이션 개발에서 CICS JSON 지원자에 대한 맵핑 레벨의 내용을 참조하십시오.

참고: UTF-16은 처리 시간이 더 길고 저장영역은 덜 필요해서 EBCDIC 인코딩보다 효율적입니다. 또한 인코딩 유형을 혼합하면 여분의 런타임 처리가 발생합니다.

XML 또는 JSON 스키마에서 언어 구조로의 UTF-16 맵핑

UTF-16에 대한 지원은 웹 서비스 작성 방식에 따라 다릅니다. XML 또는 JSON 스키마의 언어 구조로의 맵핑은 하향식 맵핑이라고도 하며 다음 특징이 있습니다. UTF-16을 사용할 경우 모든 텍스트 필드가 UTF-16 필드로 맵핑되는 반면, COBOL의 숫자 표시 데이터 유형은 EBCDIC으로 맵핑됩니다. UTF-16을 사용하려면 DFHJS2LS, DFHSC2LS 또는 DFHWS2LS의 CCSID 매개변수를 1200으로 설정하십시오.

예를 들어, 다음 XML 스키마 단편이 WSDL에 있으면:

```
<xsd:element name="myString" nillable="false">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="20"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

DFHWS2LS 지원은 COBOL 언어 구조에 다음 필드를 생성합니다.

```
myString PIC N(20) USAGE NATIONAL
```

웹 서비스 지원의 CHAR-MULTIPLIER 매개변수를 사용하여 지원이 생성하는 필드의 길이를 지정할 수 있습니다.

CHAR-MULTIPLIER

UTF-16을 사용할 때 **CHAR-MULTIPLIER** 매개변수의 유일하게 올바른 값은 2 또는 4이며, 2는 기본값입니다.

CHAR-MULTIPLIER=2, 스키마가 maxlength x 문자열을 설명하고 PIC N(x)를 생성합니다. **CHAR-MULTIPLIER=2**를 설정해도 UTF-16 문자열에 대리 쌍을 사용할 수 있지만 필드에 맞는 문자 수에는 영향이 미칩니다.

CHAR-MULTIPLIER=4는 PIC N(2x)를 생성합니다. **CHAR-MULTIPLIER=4**인 경우 단일 인코딩 단위로 표현할 수 없는 문자가 문자열에 포함되면 런타임의 값이 채워집니다.

언어 구조에서 XML 또는 JSON 스키마로의 UTF-16 맵핑

언어 구조에서 XML 또는 JSON 스키마로의 맵핑은 상향식 맵핑이라고도 하며 하향식 맵핑과 다르게 관리됩니다. UTF-16 문자열이 언어 구조에 선언되면 CICS는 데이터를

UTF-16 인코딩으로 해석하고 그렇지 않은 경우에는 데이터가 EBCDIC 인코딩이라 가정합니다. DFHLS2JS, DFHLS2SC 또는 DFHLS2WS의 CCSID 매개변수는 애플리케이션 데이터 내에 EBCDIC 텍스트의 인코딩을 표시하며, 이를 UTF-16을 표시하도록 설정해선 안됩니다.

UTF-16 문자로 해석되는 데이터 유형은 다음과 같습니다. COBOL의 PIC N(*n*), PL/I의 WIDECHAR(*n*), C 및 C++의 char16_t[*n*].

웹 서비스 지원의 CHAR-USAGE 매개변수를 사용하여 데이터 유형을 지정할 수 있습니다.

CHAR-USAGE

COBOL에서는, 국가 데이터 유형, PIC N을 UTF-16 또는 DBCS 데이터에 사용할 수 있습니다. 이 설정은 NSYMBOL 컴파일러 옵션으로 제어합니다. 지원의 **CHAR-USAGE** 매개변수를 NSYMBOL 컴파일러 옵션과 동일한 값으로 설정해서 데이터가 적합하게 처리되도록 해야 합니다. UTF-16을 사용하는 경우 이는 일반적으로 CHAR-USAGE=NATIONAL로 설정됩니다.

UTF-16 및 DBCS 데이터를 동일한 카피북에 포함한 국가 데이터 유형을 혼합하려는 경우에는 개별 필드에서 USAGE NATIONAL 또는 USAGE DISPLAY-1 규정자를 사용할 수 있습니다.

참고: DFHLS2WS, DFHLS2SC, DFHLS2JS는 COBOL GROUP USAGE NATIONAL 절을 지원하지 않습니다.

언어 구조에서 맵핑 생성

애플리케이션 데이터에서 XML을 작성(또는 반대로)하기 위해서는 런타임에 CICS가 데이터와 XML을 변환할 방식을 설명하는 맵핑을 작성합니다. 어느 애플리케이션 데이터 레코드에서나 시작할 수 있습니다. 예를 들어, COMMAREA, VSAM 파일, 임시 저장영역 큐 또는 DB2 레코드로 시작할 수 있습니다.

시작하기 전에

맵핑을 작성하기 전에 다음 전제조건이 완료되었는지 확인해야 합니다.

- 파티션된 데이터 세트에서 애플리케이션 레코드를 설명하는 언어 구조가 있어야 합니다. 언어 구조는 CICS XML 지원에서 지원되는 어느 고급 언어(COBOL, PL/I, C, C++)로든 쓸 수 있습니다. 이 맵핑이 Atom 피드에 사용되며 애플리케이션 레코드에 Atom 항목에 대한 메타데이터를 제공하는 필드를 사용 중인 경우에는(예를 들어, 작성자 이름) 이 필드가 언어 구조에 중첩되지 않는지 확인해야 합니다. Atom 항목의 내용을 제공하는 한 필드 내에 다른 필드를 중첩시키는 구조가 가능합니다.
- z/OS UNIX를 사용하기 위해 DFHLS2SC를 실행하는 사용자 ID를 구성해야 합니다.

- 사용자 ID에 언어 구조에 액세스할 읽기 권한과 z/OS UNIX의 해당 디렉토리에 출력을 둘 수 있는 쓰기 권한이 있어야 합니다.
- ID가 Java를 실행할 수 있도록 사용자 ID에 충분한 저장영역을 할당해야 합니다. 지원되는 어느 Java 버전이나 사용할 수 있습니다. 기본적으로 DFHLS2SC는 **JAVADIR** 매개변수에 지정된 Java 버전을 사용합니다.

이 태스크 정보

CICS XML 지원을 사용하여 애플리케이션 레코드에 대한 데이터 매핑을 작성하십시오. CICS XML 지원이 지원하는 각 고급 언어마다 몇 가지 유형의 데이터가 제한되거나 지원되지 않습니다. CICS XML 지원은 언어 구조에서 식별한 지원되지 않는 항목에 대한 오류 메시지를 발행합니다. CICS XML 지원에 대한 참조 정보에 각 고급 언어에 적용되는 제한사항이 나열되어 있습니다.

프로시저

1. DFHLS2SC 일괄처리 작업을 실행하십시오. DFHLS2SC에는 특정 코드 페이지 또는 이름 공간 선택처럼 요구사항을 충족시키기 위해 선택하는 선택적 매개변수가 있습니다. 최소한 다음 매개변수를 사용하십시오.
 - a. **LANG** 매개변수에 언어 구조의 고급 언어를 지정하십시오.
 - b. 번들에서 데이터 매핑을 전개 중인 경우 **BUNDLE** 매개변수에 번들 자원의 이름을 지정하십시오. Atom 피드에 대한 XML 바인딩을 작성 중이면 이 매개변수를 지정하지 마십시오.
 - c. **MAPPING-LEVEL** 매개변수에 매핑 레벨을 지정하십시오. Atom 피드에 대한 XML 바인딩을 작성 중인 경우 매핑 레벨 3.0 이상을 사용해야 합니다. 기타 상황의 경우 어느 매핑 레벨이나 사용할 수 있지만 고급 매핑 옵션을 얻으려면 최신 매핑 레벨을 사용하십시오.
 - d. 옵션: Atom 피드에 대한 XML 바인딩을 작성 중이며 애플리케이션 데이터 레코드에 CICS ABSTIME 형식의 시간 소인이 포함된 경우에는 선택적 매개변수 **DATETIME=PACKED15**를 지정하여 이 필드를 시간 소인으로서 매핑하십시오.
 - e. **PDSMEM** 및 **PDSCP** 매개변수에 애플리케이션 레코드를 설명하는 언어 구조의 코드 페이지 및 위치를 지정하십시오.
 - f. **SCHEMA** 매개변수에 스키마 파일의 이름과 위치를 지정하십시오. 파일 확장자는 .xsd입니다. 번들을 작성 중이면 위치를 지정하지 마십시오. DFHLS2SC는 파일이 아직 존재하지 않을 경우 디렉토리 구조가 아닌 XML 스키마를 작성합니다.
 - g. **XSDBIND** 매개변수에 XML 바인딩의 이름과 위치를 지정하십시오. 파일 확장자는 .xsdbind입니다. 번들을 작성 중이면 위치를 지정하지 마십시오. DFHLS2SC는 파일이 아직 존재하지 않을 경우 디렉토리 구조가 아닌 XML 바인딩을 작성합니다.

팁: 유효성 검사를 사용하려면 XML 바인딩과 스키마를 동일한 디렉토리 구조에 두십시오. 유효성 검사는 개발 또는 테스트 환경에서 애플리케이션을 테스트할 때 유용할 수 있습니다. 번들을 작성 중인 경우 CICS가 파일을 동일한 디렉토리에 둡니다.

BUNDLE 매개변수를 지정하면 일괄처리 작업이 z/OS UNIX에 번들 디렉토리 구조를 작성합니다. 번들 디렉토리에는 번들 Manifest를 포함한 META-INF 서브디렉토리가 있습니다. 일괄처리 작업은 **SCHEMA** 및 **XSDBIND** 매개변수에 지정한 파일 이름을 사용하여 번들에 XML 스키마와 XML 바인딩을 작성하기도 합니다. **BUNDLE** 매개변수를 지정하지 않을 경우 일괄처리 작업은 지정된 위치에만 XML 스키마와 XML 바인딩을 작성합니다.

2. **BUNDLE** 자원이나 이 XML 바인딩을 지정하는 **ATOMSERVICE** 자원을 설치하십시오. **BUNDLE** 및 **ATOMSERVICE** 자원은 XML 스키마 및 바인딩 파일의 위치를 정의하는 **XMLTRANSFORM** 자원을 동적으로 작성합니다.

결과

언어 구조로부터 맵핑을 생성할 때에는 XML 변환이 하나만 가능합니다.

예

다음 예제는 최소한의 매개변수 세트가 지정된 DFHLS2SC를 보여줍니다.

```
//LS2SC JOB 'accounting information',name,MSGCLASS=A
// SET QT='''
//JAVAPROG EXEC DFHLS2SC,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/xsdbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
XSDBIND=example.xsdbind
SCHEMA=example.xsd
/*
```

다음에 수행할 작업

애플리케이션 데이터를 XML로(그리고 반대로) 변환시키는 애플리케이션을 쓰십시오. 두 변환에 모두 동일한 맵핑을 사용할 수 있습니다. Atom 피드에 대한 XML 바인딩을 작성한 경우에는 Atom 피드를 설정하는 단계를 계속하십시오.

관련 개념:

476 페이지의 『CICS XML 지원』

CICS XML 지원은 XML을 고급 언어 구조로부터 변환하도록 돕는 일괄처리 유틸리티 세트입니다. 지원은 최소한의 프로그래밍 노력으로 XML 처리를 수행하는 신속

한 애플리케이션 전개를 지원합니다.

관련 태스크:

568 페이지의 『애플리케이션 데이터를 XML로 변환』

애플리케이션 데이터를 XML로 변환할 애플리케이션을 쓸 수 있습니다.

569 페이지의 『XML을 애플리케이션 데이터로 변환』

XML을 애플리케이션 데이터로 변환하는 애플리케이션을 쓸 수 있습니다. XML을 변환하기 전에 조회할 수도 있습니다.

관련 참조:

477 페이지의 『DFHLS2SC: 고급 언어 대 XML 스키마 변환』

DFHLS2SC 카탈로그된 프로시저는 고급 언어 구조로부터 XML 스키마와 XML 바인딩 파일을 생성합니다. XML을 구문 분석하거나 작성할 수 있는 CICS 프로그램을 작성하려 할 때 DFHLS2SC를 사용하십시오. DFHLS2SC의 작업 제어문, 기호 매개변수, 입력 매개변수, 설명이 나열됩니다.

501 페이지의 『고급 언어 및 XML 스키마 맵핑』

CICS 지원을 사용하여 고급 언어 구조와 XML 스키마 또는 WSDL 문서 간의 맵핑을 생성합니다. CICS 지원은 고급 언어 데이터 구조로부터 XML 스키마 또는 WSDL 문서를 생성하기도 합니다(반대의 경우도 마찬가지).

XML 스키마에서 맵핑 생성

기존 XML 스키마를 따르는 XML에서 애플리케이션 데이터를(또는 반대로) 작성하기 위해서는 CICS가 런타임에 데이터를 변환할 방식을 설명하는 맵핑을 작성합니다. XML 스키마나 WSDL 문서로부터 시작할 수 있습니다.

시작하기 전에

올바른 XML 스키마나 WSDL 문서가 있어야 합니다. 맵핑을 작성하기 전에 다음 전제조건이 완료되었는지 확인해야 합니다.

- 올바른 XML 스키마나 WSDL 문서가 있어야 합니다.
- UNIX System Services를 사용하기 위해 DFHSC2LS를 실행하는 사용자 ID를 구성해야 합니다.
- 사용자 ID에 XML 스키마 또는 WSDL 문서에 액세스할 읽기 권한과 z/OS UNIX의 해당 디렉토리에 출력을 둘 수 있는 쓰기 권한이 있어야 합니다.
- 사용자 ID에 Java를 실행하기에 충분히 큰 저장영역이 할당되어 있어야 합니다. 지원되는 어느 Java 버전이나 사용할 수 있습니다. 기본적으로 DFHWS2LS는 **JAVADIR** 매개변수에 지정된 Java 버전을 사용합니다.

이 태스크 정보

CICS XML 지원을 사용하여 XML 스키마에 대한 데이터 맵핑을 작성하십시오.

프로시저

1. DFHSC2LS 일괄처리 작업을 실행하십시오. DFHSC2LS에는 특정 코드 페이지 선택이나 가변 길이 문자 데이터의 처리 방식 지정과 같은 요구사항을 충족시키기 위해 선택할 수 있는 선택적 매개변수가 있습니다. 최소한 다음 매개변수를 사용하십시오.
 - a. **WSDL** 또는 **SCHEMA** 매개변수에 입력 파일의 위치를 지정하십시오. WSDL 문서나 XML 스키마를 사용할 수 있습니다. 입력 파일에 인터넷의 다른 스키마 또는 문서에 대한 참조가 포함되어 있고 시스템이 프록시 서버를 사용하는 경우에는, 프록시 서버의 포트 번호와 IP 주소 또는 도메인 이름을 지정하십시오.
 - b. **LANG** 매개변수에 생성하려는 고급 언어를 지정하십시오. XML 지원은 COBOL, C, C++, PLI 언어를 지원합니다.
 - c. 번들에서 데이터 매핑을 전개 중인 경우 **BUNDLE** 매개변수에 번들의 이름과 위치를 지정하십시오.

XML 지원은 XML 바인딩에 지원되는 변환의 라이브러리를 작성합니다. 입력 파일의 각 글로벌 요소마다 지원이 별도의 변환을 작성합니다.

BUNDLE 매개변수를 지정하면 일괄처리 작업이 z/OS UNIX에 번들 디렉토리 구조를 작성합니다. 번들 디렉토리에는 번들 Manifest를 포함한 META-INF 서브디렉토리가 있습니다. 또한 일괄처리 작업은 번들 디렉토리에 XML 바인딩을 작성하고 언어 구조를 지정된 위치에 배치합니다. XML 지원은 입력 파일의 사본을 번들 디렉토리에 배치하기도 합니다. **BUNDLE** 매개변수를 지정하지 않을 경우 일괄처리 작업은 지정된 위치에만 언어 구조와 XML 바인딩을 작성합니다.

2. **BUNDLE** 자원을 설치하십시오. **BUNDLE** 자원은 XML 스키마나 WSDL 문서, XML 바인딩, 언어 구조의 위치를 정의하는 XMLTRANSFORM 자원을 동적으로 작성합니다.

결과

XML 스키마에서 매핑을 생성할 때 CICS가 스키마에 있는 모든 글로벌 요소에 대한 언어 구조를 생성합니다.

예

다음 예제는 최소한의 매개변수 세트가 지정된 DFHSC2LS를 보여줍니다.

```
//SC2LS JOB 'accounting information',name,MSGCLASS=A
// SET QT='''
//JAVAPROG EXEC DFHSC2LS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/xsdbind/example.log
MAPPING-LEVEL=3.0
```

```

PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
XSDBIND=example.xsdbind
SCHEMA=example.xsd
/*

```

다음에 수행할 작업

애플리케이션 데이터를 XML로(또는 반대로) 변환시키는 애플리케이션을 쓰십시오. 두 변환에 모두 동일한 맵핑을 사용할 수 있습니다.

관련 개념:

476 페이지의 『CICS XML 지원』

CICS XML 지원은 XML을 고급 언어 구조로(부터) 변환하도록 돕는 일괄처리 유틸리티 세트입니다. 지원은 최소한의 프로그래밍 노력으로 XML 처리를 수행하는 신속한 애플리케이션 전개를 지원합니다.

관련 태스크:

『애플리케이션 데이터를 XML로 변환』

애플리케이션 데이터를 XML로 변환할 애플리케이션을 쓸 수 있습니다.

569 페이지의 『XML을 애플리케이션 데이터로 변환』

XML을 애플리케이션 데이터로 변환하는 애플리케이션을 쓸 수 있습니다. XML을 변환하기 전에 조회할 수도 있습니다.

애플리케이션 데이터를 XML로 변환

애플리케이션 데이터를 XML로 변환할 애플리케이션을 쓸 수 있습니다.

시작하기 전에

XML 바인딩 및 XML 스키마를 정의하는 XMLTRANSFORM 자원을 사용으로 설정했어야 합니다.

이 태스크 정보

XML 지원은 XML 바인딩에 맵핑을 생성합니다. 언어 구조로 시작하고 DFHLS2SC를 사용한 경우 XML로의 변환이 하나만 가능합니다. XML 스키마로 시작한 경우에는 많은 XML 대 언어 구조 변환이 있을 수 있으므로 애플리케이션이 생성할 XML 요소를 선택해야 합니다.

프로시저

1. 애플리케이션은 채널을 작성하고 언어 구조와 연관된 데이터를 이 채널의 비트 모드 컨테이너에 두어야 합니다.
2. **TRANSFORM DATATOXML** API 명령을 사용하여 데이터를 XML로 변환하십시오.

```
EXEC CICS TRANSFORM DATATOXML
      XMLTRANSFORM('MyXmlTransformName')
      CHANNEL('MyChannelName')
      DATCONTAINER('SourceContainerName')
      XMLCONTAINER('TargetContainerName')
```

XMLTRANSFORM 자원이 언어 구조로부터의 단일 변환만 지원하면 명령에 변환 유형을 지정하지 않아도 됩니다. 많은 변환이 가능할 경우 다음 옵션을 애플리케이션에 추가하십시오.

```
ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

이 추가 옵션은 애플리케이션 데이터를 변환하여 출력 컨테이너에 배치할 XML 요소를 표시합니다.

3. 애플리케이션을 설치하십시오.

결과

애플리케이션이 **TRANSFORM DATATOXML** 명령을 실행할 때 CICS는 XMLTRANSFORM 자원을 확인하여 XML 바인딩에서 매핑을 찾고 채널의 컨테이너를 사용하여 애플리케이션 2진 데이터를 XML로 변환시킵니다. XML은 애플리케이션이 XMLCONTAINER 옵션에 대해서 지정한 컨테이너에 위치합니다. XML은 XMLTRANSFORM 자원에 정의된 XML 스키마를 따릅니다.

다음에 수행할 작업

동일한 매핑을 사용하여 XML을 애플리케이션 데이터로 변환할 수도 있습니다. 세부사항은 『XML을 애플리케이션 데이터로 변환』의 내용을 참조하십시오.

관련 태스크:

563 페이지의 『언어 구조에서 매핑 생성』

애플리케이션 데이터에서 XML을 작성(또는 반대로)하기 위해서는 런타임에 CICS가 데이터와 XML을 변환할 방식을 설명하는 매핑을 작성합니다. 어느 애플리케이션 데이터 레코드에서나 시작할 수 있습니다. 예를 들어, COMMAREA, VSAM 파일, 임시 저장영역 큐 또는 DB2 레코드로 시작할 수 있습니다.

566 페이지의 『XML 스키마에서 매핑 생성』

기존 XML 스키마를 따르는 XML에서 애플리케이션 데이터를(또는 반대로) 작성하기 위해서는 CICS가 런타임에 데이터를 변환할 방식을 설명하는 매핑을 작성합니다.

XML 스키마나 WSDL 문서로부터 시작할 수 있습니다.

XML을 애플리케이션 데이터로 변환

XML을 애플리케이션 데이터로 변환하는 애플리케이션을 쓸 수 있습니다. XML을 변환하기 전에 조회할 수도 있습니다.

시작하기 전에

XML 바인딩 및 XML 스키마를 정의하는 XMLTRANSFORM 자원을 사용으로 설정했어야 합니다.

이 태스크 정보

CICS XML 지원은 XML 바인딩에 맵핑을 생성합니다. 언어 구조로 시작하고 DFHLS2SC를 사용한 경우 XML로부터의 변환이 하나만 가능합니다. XML 스키마로 시작한 경우에는 많은 XML 대 언어 구조 변환이 있을 수 있으므로 애플리케이션은 입력으로 사용할 언어 구조를 선택해야 합니다.

프로시저

1. 채널을 작성하고 이 채널의 텍스트 모드 컨테이너에 XML을 두십시오. 애플리케이션이 XML을 BIT 모드 컨테이너에 넣으면 CICS는 텍스트 데이터의 인코딩을 판별하려 시도하지만 컨테이너를 처리하는 데 시간이 더 오래 걸리고 인코딩이 올바르지 않을 수도 있습니다.
2. **TRANSFORM XMLTODATA** API 명령을 사용하십시오. XML에 대한 변환이 하나만 가능한 경우 다음 명령을 사용할 수 있습니다.

```
EXEC CICS TRANSFORM XMLTODATA
      XMLTRANSFORM('MyXmlTransformName')
      CHANNEL('MyChannelName')
      XMLCONTAINER('SourceContainerName')
      DATCONTAINER('TargetContainerName')
```

둘 이상의 변환이 가능하면 다음 추가 옵션을 사용하십시오.

```
EXEC CICS TRANSFORM XMLTODATA
      XMLTRANSFORM('MyXmlTransformName')
      CHANNEL('MyChannelName')
      XMLCONTAINER('SourceContainerName')
      DATCONTAINER('TargetContainerName')
      ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

두 번째 예제에서, CICS는 ELEMNAME 옵션에 있는 요소의 이름을 리턴합니다. 그러면 애플리케이션은 요소의 이름을 사용하여 대상 컨테이너의 내용을 해석하기 위해 사용할 언어 구조의 라이브러리를 판별할 수 있습니다.

3. 애플리케이션을 설치하십시오.

결과

애플리케이션이 **TRANSFORM XMLTODATA** 명령을 실행할 때 CICS는 XMLTRANSFORM 자원의 세부사항을 사용하여 채널의 컨테이너를 통해 XML을 애플리케이션 2진 데이터로 변환합니다.

예

다음에 수행할 작업

동일한 맵핑을 사용하여 애플리케이션 데이터를 XML로 변환할 수도 있습니다. 세부사항은 568 페이지의 『애플리케이션 데이터를 XML로 변환』의 내용을 참조하십시오.

관련 태스크:

563 페이지의 『언어 구조에서 맵핑 생성』

애플리케이션 데이터에서 XML을 작성(또는 반대로)하기 위해서는 런타임에 CICS가 데이터와 XML을 변환할 방식을 설명하는 맵핑을 작성합니다. 어느 애플리케이션 데이터 레코드에서나 시작할 수 있습니다. 예를 들어, COMMAREA, VSAM 파일, 임시 저장영역 큐 또는 DB2 레코드로 시작할 수 있습니다.

566 페이지의 『XML 스키마에서 맵핑 생성』

기존 XML 스키마를 따르는 XML에서 애플리케이션 데이터를(또는 반대로) 작성하기 위해서는 CICS가 런타임에 데이터를 변환할 방식을 설명하는 맵핑을 작성합니다.

XML 스키마나 WSDL 문서로부터 시작할 수 있습니다.

애플리케이션에서 XML 조회

XML을 애플리케이션 데이터로 변환하기 전에 XML의 단편을 조회할 애플리케이션을 쓸 수 있습니다.

이 태스크 정보

애플리케이션이 여러 다른 유형의 XML을 처리할 경우 XML을 조회하여 XML을 애플리케이션 데이터로 변환하기 위해 사용할 XMLTRANSFORM 자원을 판별하고자 할 수 있습니다. 이 명령은 XML에 <xsd:any> 요소가 포함된 경우에도 유용할 수 있습니다.

프로시저

1. 애플리케이션에서 **TRANSFORM XMLTODATA** API 명령을 사용하여 XML을 조회하십시오.

```
EXEC CICS TRANSFORM XMLTODATA
      CHANNEL('MyChannelName')
      XMLCONTAINER('SourceContainerName')
      ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

XML을 보유한 컨테이너의 이름과 채널의 이름을 지정해야 합니다. XML을 조회할 XMLTRANSFORM 자원을 지정할 필요는 없습니다. 위의 예제는 첫 번째 XML 요소의 이름과 XML 요소의 길이를 조회합니다. 첫 번째 XML 요소의 유형, 유형의 길이, 유형의 이름 공간도 조회할 수 있습니다.

2. 옵션: 애플리케이션이 XML 요소의 이름 공간을 필요로 하는 경우에는 CICS가 ELEMNS 값을 쓸 수 있는 데이터 영역을 제공하십시오.
3. 옵션: XML을 조회한 후 XML을 애플리케이션 데이터로 변환하기 위해 사용할 XMLTRANSFORM 자원을 판별하는 애플리케이션 로직을 쓸 수 있습니다.
4. CICS에서 애플리케이션을 설치하십시오.

결과

CICS는 지정된 컨테이너를 읽고 XML 요소에 대한 정보를 애플리케이션에 리턴합니다.

데이터 유형으로 XML 처리

XML 스키마에 글로벌 데이터 유형이 포함되어 있고 이 중에서 하나 이상이 XML에 참조되는 경우 이 글로벌 데이터 유형을 지원할 메타데이터를 생성한 후 애플리케이션에서 XML을 구문 분석하거나 변환할 수 있습니다.

시작하기 전에

올바른 메타데이터를 생성하려면 TYPES=ALL 매개변수를 지정해서 DFHSC2LS를 실행해야 합니다.

이 태스크 정보

프로시저

1. 데이터 유형을 통해 애플리케이션 데이터를 XML로 변환하려면 **TRANSFORM DATATOXML** 명령을 사용하십시오.

```
EXEC CICS TRANSFORM DATATOXML
      XMLTRANSFORM('MyXmlTransformName')
      CHANNEL('MyChannelName')
      DATCONTAINER('SourceContainerName')
      XMLCONTAINER('TargetContainerName')
      ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
      ELEMNS(elementNamespace) ELEMNSLEN(elementNamespaceLength)
      TYPENAME(typeName) TYPENAMELEN(typeNameLen)
      TYPENS(typeNamespace) TYPENSLEN(typeNamespaceLen)
```

*MyXmlTransformName*은 XML 바인딩과 스키마를 지정하는 XMLTRANSFORM 자원의 32자 이름이고, *MyChannelName*은 입력 및 출력 컨테이너가 있는 채널의 16자 이름이고, *SourceContainerName*은 애플리케이션 데이터를 보유한 입력 컨테이너의 16자 이름이며, *TargetContainerName*은 CICS가 XML로 채우는 출력 컨테이너의 16자 이름입니다. XML 요소 이름, 이름 공간, 데이터 유형, 변환할 데이터 유형의 이름 공간도 지정해야 합니다.

2. XML을 데이터 유형으로 구문 분석하려면 **TRANSFORM XMLTODATA** 명령을 사용하십시오. 명령에 지정하는 옵션은 XML에 `xsi:type` 속성이 있는지 여부에 따라 다릅니다.

- XML에 `xsi:type` 속성이 사용된 경우 애플리케이션에 다음 명령을 지정하십시오.

```
EXEC CICS TRANSFORM XMLTODATA
      XMLTRANSFORM('MyXmlTransformName')
      CHANNEL('MyChannelName')
      XMLCONTAINER('SourceContainerName')
      DATCONTAINER('TargetContainerName')
      ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
      TYPENAME(typeName) TYPENAMELEN(typeNameLength)
```

애플리케이션에 데이터 유형의 이름 공간도 필요한 경우에는 **TYPENS** 옵션을 API 명령에 추가하십시오. CICS는 **TYPENAME**의 `xsi:type` 속성에서 데이터 유형을 리턴합니다.

- XML에 `xsi:type` 속성이 사용되지 않으면 애플리케이션은 글로벌 데이터 유형의 이름 공간과 로컬 이름을 지정할 수 있습니다. 애플리케이션에 다음 명령을 사용하십시오.

```
EXEC CICS TRANSFORM XMLTODATA
      XMLTRANSFORM('MyXmlTransformName')
      CHANNEL('MyChannelName')
      XMLCONTAINER('SourceContainerName')
      DATCONTAINER('TargetContainerName')
      TYPENAME(typeName) TYPENAMELEN(typeNameLength)
      TYPENS(TypeNamespace) TYPENSLEN(typeNamespaceLen)
```

애플리케이션은 이름과 이름 공간이 아닌 유형 이름 및 이름 공간을 지정해야 합니다. 이 명령은 애플리케이션이 CICS에 사용할 데이터 유형을 알린다고 표시합니다. CICS는 XML을 생성할 때 항상 `xsi:type` 속성을 추가합니다.

3. CICS에서 애플리케이션을 설치하십시오.

다음에 수행할 작업

애플리케이션이 예상대로 XML을 생성하고 구문 분석하는지 테스트하십시오.

<xsd:any> 데이터 유형 처리

하나 이상의 `<xsd:any>` 데이터 유형을 포함한 XML 스키마에 대해 작업 중인 경우 XML 지원은 데이터 유형을 CICS 컨테이너 쌍으로 맵핑할 수 있습니다. 컨테이너의 XML을 구문 분석할 애플리케이션을 쓸 수 있습니다.

시작하기 전에

맵핑 레벨 2.1 이상에서 DFHSC2LS 또는 DFHWS2LS를 사용하여 XML 스키마를 맵핑해야 합니다.

이 태스크 정보

CICS는 데이터를 XML로 변환할 때 <xsd:any> 데이터 유형과 연관된 XML을 첫 번째 컨테이너에 넣고 범위 내의 이름 공간 접두부 선언은 두 번째 컨테이너에 넣습니다.

프로시저

1. XML 데이터를 구문 분석하려면 애플리케이션에서 **TRANSFORM XMLTODATA** 명령을 사용하십시오.

```
EXEC CICS TRANSFORM XMLTODATA
      XMLTRANSFORM('MyXmlTransformName')
      CHANNEL('MyChannelName')
      XMLCONTAINER('SourceContainerName')
      DATCONTAINER('TargetContainerName')
      NSCONTAINER('NamespacesContainerName')
      ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

*MyXmlTransformName*은 XML 바인딩과 스키마를 지정하는 XMLTRANSFORM 자원의 32자 이름이고, *MyChannelName*은 입력 및 출력 컨테이너가 있는 채널의 16자 이름이고, *SourceContainerName*은 XML을 보유한 입력 컨테이너의 16자 이름이고, *TargetContainerName*은 CICS가 애플리케이션 데이터로 채우는 출력 컨테이너의 16자 이름이며, *NamespacesContainerName*은 CICS가 이름 공간 접두부 선언으로 채우는 컨테이너의 16자 이름입니다. ELEMNAME 및 ELEMNAMELEN 옵션의 초기값을 제공하십시오. CICS는 ELEMNAME 및 ELEMNAMELEN 옵션에 XML 요소 및 길이를 리턴합니다.

2. CICS에서 애플리케이션을 설치하십시오.

다음에 수행할 작업

애플리케이션이 XML을 제대로 구문 분석하는지 테스트하십시오.

관련 참조:

546 페이지의 『<xsd:any> 및 xsd:anyType에 대한 지원』

DFHWS2LS 및 DFHSC2LS는 XML 스키마의 <xsd:any> 및 xsd:anyType 사용을 지원합니다. <xsd:any> XML 스키마 요소를 사용하여 내용이 정의되지 않은 XML 문서의 섹션을 설명할 수 있습니다. xsd:anyType은 모든 단순 및 복합 데이터 유형이 파생되는 기본 데이터 유형으로, 데이터 내용에 대한 제한사항이나 제한조건이 없습니다.

XML 변환 유효성 검사

CICS XML 지원을 사용하여 애플리케이션 데이터를 XML로 변환하는 경우 런타임에 발생하는 변환의 유효성을 검사해서 이 변환이 XML 바인딩에 포함된 스키마를 따르는지 확인하도록 지정할 수 있습니다. XML에서 2진 데이터 또는 2진 데이터에서 XML로의 변환에 대한 유효성 검사를 수행할 수 있습니다.

시작하기 전에

CICS 애플리케이션의 개발 및 테스트 중 전체 유효성 검사는 XML의 문제를 발견하도록 지원합니다. 하지만 XML의 전체 유효성 검사는 상당한 오버헤드를 수반하므로 완전히 테스트한 프로덕션 애플리케이션에서 XML의 유효성을 검사하는 것은 바람직하지 않습니다.

CICS는 Java 프로그램을 사용하여 스키마에 대해 XML의 유효성을 검사합니다. 따라서 유효성 검사를 수행하려면 CICS 리전에서 Java 지원을 사용 가능하게 설정해야 합니다.

프로시저

1. CICS 리전에 JVM 서버를 설정하십시오. XML 유효성 검사기 클래스는 Liberty 프로파일이 아닌 OSGi 프레임워크 또는 Axis2에서 실행할 수 있습니다. CICS는 OSGi 프레임워크를 사용하는 JVM 서버를 빠르게 설정하기 위한 샘플을 제공합니다.
 - a. 샘플 JVM 서버 DFHJVMS를 DFH\$OSGI 그룹에 설치하거나 자신만의 JVM 서버를 작성하십시오. 자세한 정보는 구성에서 JVM 서버 설정의 내용을 참조하십시오.
 - b. 자신의 JVM 서버를 작성한 경우 그룹 DFHPIVAL에서 DFHPIVAL 프로그램 정의를 수정하여 JVMSERVER 자원의 이름을 참조하십시오. DFHPIVAL 정의는 잠겨 있지 않으며 편집할 수 있습니다. 기본적으로 정의는 DFHJVMS를 참조합니다.
2. XML 바인딩과 스키마가 z/OS UNIX의 동일한 위치에 있는지 확인하십시오. XMLTRANSFORM 자원은 이 파일을 CICS에 정의합니다. **INQUIRE XMLTRANSFORM** 명령을 사용하여 각 파일의 위치를 확인할 수 있습니다.
3. 애플리케이션에 대한 유효성 검사를 켜십시오. CICS Explorer에서 XMLTRANSFORM 자원을 열고 속성 목록에서 유효성 검사 상태 필드를 편집하십시오. CEMT 또는 SPI를 사용할 수도 있습니다.

결과

시스템 로그를 검토해서 XML 변환이 올바른지 확인하십시오. DFHML0508 메시지는 XML의 유효성 검사에 성공했음을 나타내고 DFHML0507 메시지는 유효성 검사에 실패했음을 나타냅니다.

다음에 수행할 작업

애플리케이션에 대한 XML 유효성 검사가 더 이상 필요하지 않을 때 XMLTRANSFORM 자원을 갱신하여 끄십시오.

제 20 장 애플리케이션 데이터와 JSON 맵핑 및 변환

애플리케이션 2진 데이터를 JSON(JavaScript Object Notation)으로(그리고 반대로) 변환하는 애플리케이션을 작성할 수 있습니다. CICS는 많은 수의 고급 언어를 지원하며 런타임 처리 중에 데이터가 변환되는 방식을 맵핑할 JSON 지원을 제공합니다. CICS는 웹 서비스 지원의 일부로, 애플리케이션 데이터를 JSON 메시지로 맵핑하는 데 동일한 기술을 사용합니다.

시작하기 전에

JSON 지원을 실행하려면 Java가 설치되어 있어야 합니다. CICS를 사용하여 내부적으로 또는 JVM 서버를 사용하여 변환할 수 있습니다. 변환을 위해 Java를 사용하는 경우, 애플리케이션 데이터 및 JSON을 변환하려면 Axis2 JVM 서버가 설치되어 있어야 합니다. 자세한 정보는 Axis2에 대한 JVM 서버 구성을 참조하십시오.

이 태스크 정보

애플리케이션 데이터를 JSON으로(부터) 변환시키는 이 접근법의 장점은 CICS가 JSON 구문 분석기에 제공된 기능보다 더 많은 역할을 한다는 점입니다. CICS는 JSON을 해석하고 애플리케이션 데이터의 레코드 기반 변환을 수행할 수 있습니다. 따라서 이 접근법을 사용하면 JSON에 대해 작업하는 애플리케이션을 보다 간편하고 빠르게 작성할 수 있습니다.

CICS JSON 지원은 애플리케이션 2진 데이터를 JSON으로 변환하거나 JSON을 애플리케이션 2진 데이터로 변환하기 위한 필수 맵핑 아티팩트를 작성하는 데 도움을 주도록 제공된 유틸리티입니다. JSON 지원은 번들 디렉토리에 아티팩트를 작성합니다.

프로시저

1. JSON 지원을 사용하여 번들을 작성하십시오.
2. CICS에 번들을 설치하여 맵핑을 사용 가능하게 하십시오.
3. LINK PROGRAM API 명령을 사용하여 CICS 제공 링크 가능 인터페이스 DFHJSON에 링크할 애플리케이션을 작성 또는 갱신하십시오. 애플리케이션은 채널 기반 인터페이스를 사용해야 합니다.
4. 애플리케이션을 실행하여 변환이 의도한 대로 작동하는지 테스트하십시오.

결과

애플리케이션 데이터가 JSON으로 변환되거나 JSON이 애플리케이션 데이터로 변환됩니다.

다음에 수행할 작업

577 페이지의 1 - 577 페이지의 4 단계는 다음 주제에서 보다 자세히 설명합니다.

CICS JSON 지원

The CICS JSON 지원은 JSON을 애플리케이션 데이터로(부터) 변환하도록 돕는 일괄 처리 유틸리티 세트입니다. 지원은 최소한의 프로그래밍 노력으로 JSON 처리를 수행하는 신속한 애플리케이션 전개를 지원합니다.

CICS 지원을 사용하여 고급 언어 구조와 JSON 스키마 간의 맵핑을 생성하십시오. CICS 지원은 고급 언어 데이터 구조로부터 JSON 스키마를 생성하기도 합니다(반대의 경우도 마찬가지).

JSON 지원을 CICS에 사용하면 JSON을 구문 분석하거나 구성하기 위해 써야 하는 코드 양이 줄어듭니다. CICS는 JSON 단편과 애플리케이션의 데이터 구조 간에 데이터를 변환시킵니다.

JSON 지원은 단순 언어 구조 또는 기존 XML 스키마의 언어 구조에서 JSON 스키마를 작성하며 COBOL, C/C++, PL/I를 지원합니다. 또한 CICS가 런타임에 XML 데이터를 2진 애플리케이션 데이터로 또는 그 반대로 자동 변환시키기 위해 사용하는 메타데이터를 생성합니다. 메타데이터는 XML 바인딩에 정의되고 z/OS UNIX에 저장됩니다. XML 바인딩의 스키마는 z/OS UNIX의 /usr/lpp/cicsts/cicsts52/schemas/xmltransform/ 디렉토리에 있습니다.

유틸리티 프로그램 DFHJS2LS 및 DFHLS2JS를 총괄하여 CICS JSON 지원이라 합니다.

DFHLS2JS

DFHLS2JS는 고급 언어 구조를 JSON 스키마로 맵핑합니다.

DFHJS2LS

DFHJS2LS는 JSON 스키마를 고급 언어 구조로 맵핑합니다.

두 프로그램을 모두 실행하는 JCL 프로시저는 *hlq.XDFHINST* 라이브러리에 있으며, 여기서 *hlq*는 CICS 설치의 상위 레벨 규정자입니다.

두 개의 맵핑은 대칭적이지 않습니다.

- DFHLS2JS로 언어 데이터 구조를 처리한 후 결과적인 JSON 스키마를 DFHJS2LS로 처리하는 경우 최종 데이터 구조가 원래와 같을 것이라 예상하지 마십시오.
- JSON 스키마를 DFHJS2LS로 처리한 후 결과적인 언어 구조를 DFHLS2JS로 처리하는 경우 최종 JSON 스키마가 원래와 같을 것이라 예상하지 마십시오.
- 일부 경우 DFHJS2LS는 DFHLS2JS에서 지원되지 않는 언어 구조를 생성합니다.

CICS가 지원하는 언어 컴파일러에 구현된 대로, 언어 규칙에 따라 DFHLS2JS로 처리한 고급 언어 구조를 코딩해야 합니다.

DFHLS2JS: 링크 가능한 인터페이스의 고급 언어 대 JSON 스키마 변환

DFHLS2JS 카탈로그된 프로시저는 고급 언어 구조로부터 JSON 스키마와 JSON 바인딩 파일을 생성합니다. JSON을 구문 분석하거나 작성할 수 있는 CICS 프로그램을 작성하려 할 때 DFHLS2JS를 사용하십시오. 이 프로시저의 사용을 돕기 위해 작업 제어문, 기호 매개변수, DFHLS2JS의 입력 매개변수, DFHLS2JS를 사용하는 예제 작업이 제공됩니다.

DFHLS2JS JCL 프로시저는 *HLQ.XDFHINST* 데이터 세트에 설치되는데, *HLQ*는 CICS가 설치된 상위 레벨 규정자입니다.

DFHLS2JS 또는 DFHJS2LS 프로시저에 대한 관련 사용 모드는 사용자 요구사항에 따라 다릅니다.

- DFHLS2JS: 링크 가능 인터페이스에 대한 고급 언어 대 JSON 스키마 변환
- DFHJS2LS: 링크 가능 인터페이스에 대한 JSON 스키마 대 고급 언어 변환
- DFHLS2JS: 요청-응답 서비스를 위한 JSON 스키마 변환을 위한 고급 언어
- DFHJS2LS: 요청-응답 서비스에 대한 고급 언어 변환에 대한 JSON 스키마
- DFHJS2LS: RESTful 서비스를 위한 고급 언어 변환에 대한 JSON 스키마

DFHLS2JS의 작업 제어문

JOB 작업을 시작합니다.

EXEC

프로시저 이름(DFHLS2JS)를 지정합니다.

INPUT.SYSUT1 DD

입력을 지정합니다. 입력 매개변수는 입력 스트림에 지정됩니다. 하지만 데이터 세트나 파티션된 데이터 세트의 멤버에 정의할 수도 있습니다.

기호 매개변수

다음 기호 매개변수가 DFHLS2JS에 정의됩니다.

JAVADIR=*path*

DFHLS2JS가 사용하는 Java 디렉토리의 이름을 지정합니다. 이 매개변수의 값은 전체 경로 이름 */usr/lpp/path*를 제공하는 */usr/lpp/*에 추가됩니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **JAVADIR** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

PATHPREF=prefix

다른 매개변수에 사용된 z/OS UNIX 디렉토리를 확장하는 선택적 접두부를 지정합니다. 기본값은 빈 문자열입니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **PATHPREF** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

SERVICE=value

IBM 지원 센터에서 지시한 경우에만 이 매개변수를 사용하십시오.

TMPDIR=tmpdir

DFHLS2JS가 임시 작업공간으로 사용하는 z/OS UNIX의 디렉토리 위치를 지정합니다. 작업을 실행하는 사용자 ID에 이 디렉토리에 대한 읽기 및 쓰기 권한이 있어야 합니다.

기본값은 /tmp입니다.

TMPFILE=tmpprefix

DFHLS2JS가 임시 작업공간 파일의 이름을 구성하는 데 사용하는 접두부를 지정합니다.

기본값은 LS2JS입니다.

USSDIR=path

z/OS UNIX 파일 시스템에서 CICS TS 디렉토리의 이름을 지정합니다. 이 매개변수의 값은 전체 경로 이름 /usr/lpp/cicsts/path를 제공하는 /usr/lpp/cicsts/에 추가됩니다. 기본값이 사용되는 경우 이것은 '.'(마침표)로 지정되어야 합니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **USSDIR** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

임시 작업공간

DFHLS2JS는 런타임 시에 다음의 세 임시 파일을 작성합니다.

tmpdir/tmpprefix.in

tmpdir/tmpprefix.out

tmpdir/tmpprefix.err

여기서,

*tmpdir*은 **TMPDIR** 매개변수에 지정된 값입니다.

*tmpprefix*는 **TMPFILE** 매개변수에 지정된 값입니다.

파일의 기본 이름(**TMPDIR** 및 **TMPFILE**이 지정되지 않았을 때)은 다음과 같습니다.

/tmp/LS2JS.in

/tmp/LS2JS.out

/tmp/LS2JS.err

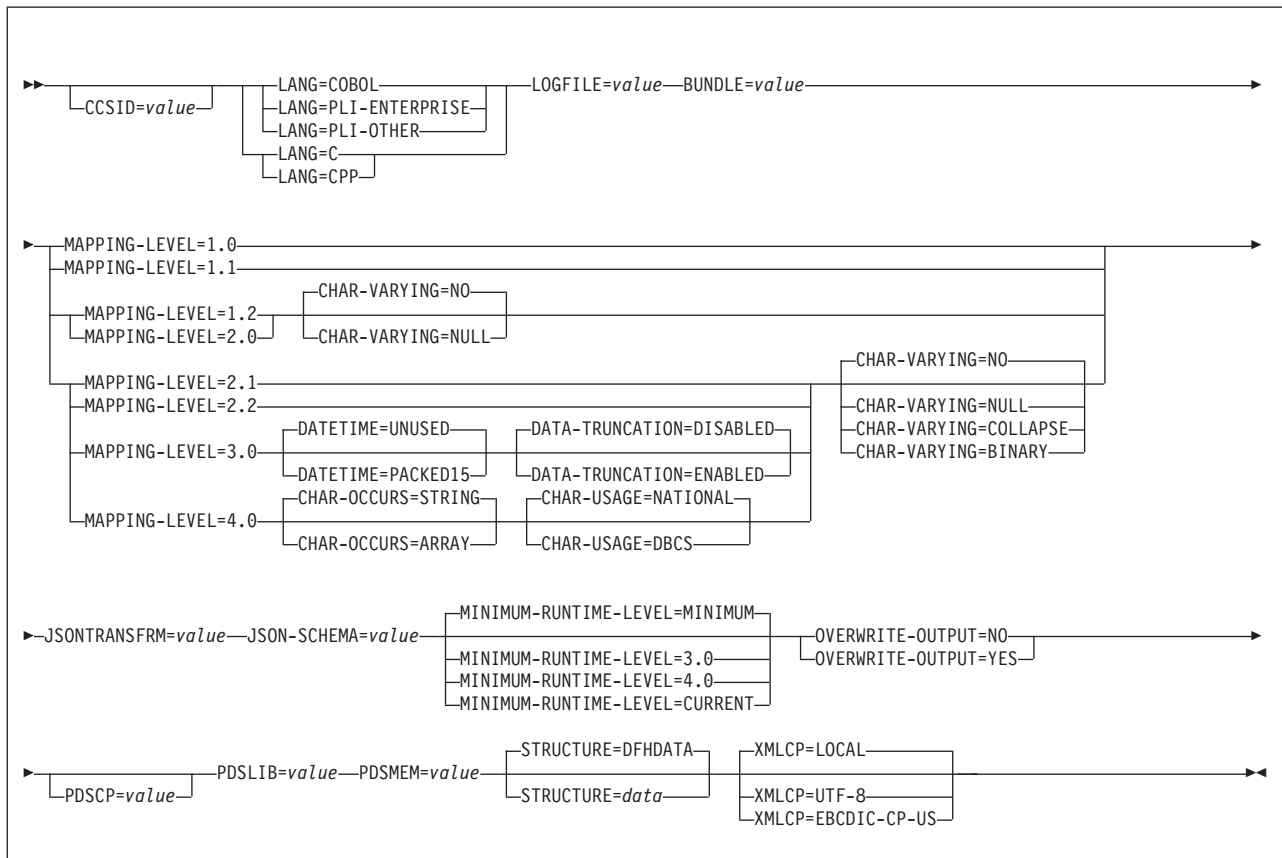
중요사항: DFHLS2JS는 z/OS UNIX 파일이나 데이터 세트 멤버에 대한 액세스를 잡지 않습니다. 따라서 둘 이상의 DFHLS2JS 인스턴스가 동시에 실행하며 동일한 임시 작업공간 파일을 사용하는 경우에는, 한 작업이 작업공간 파일을 사용하는 동안 또 다른 작업이 이 파일을 겹쳐쓰게 되어 예측 불가능한 장애가 발생할 수 있습니다.

이 같은 상황을 피하는 작동 프로시저와 이름 지정 규칙을 고안하십시오. 예를 들어, 시스템 기호 매개변수 **SYSUID**를 사용하여 개별 사용자 고유의 작업공간 파일 이름을 생성할 수 있습니다.

이들 임시 파일은 작업 종료 전에 삭제됩니다.

DFHLS2JS의 입력 매개변수

다음 구문 다이어그램은 사용 가능한 입력 매개변수를 보여줍니다.



매개변수 사용

매개변수는 다음 규칙을 준수해야 합니다.

- 입력 매개변수를 어느 순서로든 지정할 수 있습니다.
- 각 매개변수는 새 줄에서 시작해야 합니다.

- 매개변수(및 연속 문자, 사용하는 경우)가 72열을 초과하면 안됩니다. 73 - 80열은 공백을 포함해야 합니다.
- 매개변수가 단일 행에 맞추기에 너무 긴 경우에는 행의 끝에 별표(*) 문자를 사용하여 매개변수가 다음 행에서 계속됨을 표시하십시오. 별표 이전의 모든 항목(공백을 포함하여)이 매개변수의 일부로 간주됩니다.
- 행의 첫 번째 문자 위치에 있는 번호 부호(#) 문자는 주석 문자입니다. 해당 행은 무시됩니다.
- 행의 마지막 문자 위치에 있는 쉼표는 선택적인 행 분리문자이며 무시됩니다.

매개변수 설명

BUNDLE=*value*

z/OS UNIX의 번들 디렉토리 경로 및 이름을 지정합니다. 이 값을 지정하면 JSON 지원이 JSON 바인딩을 포함한 번들을 생성합니다. 이 매개변수의 경로 정보는 **JSONTRANSFRM** 매개변수의 경로 정보를 덮어씁니다.

선택적으로 디렉토리 이름이 아닌 아카이브 파일을 지정할 수 있습니다. JSON 지원은 .zip 및 .jar 아카이브를 지원합니다. 하지만 BUNDLE 자원을 설치하려 시도하기 전에 아카이브의 압축을 풀어야 합니다.

CCSID=*value*

런타임 시에 애플리케이션 데이터 구조에서 문자 데이터를 인코딩하는 데 사용되는 CCSID를 지정합니다. 이 매개변수의 값이 **LOCALCCSID** 시스템 초기화 매개변수의 값을 덮어씁니다. *value* 값은 Java 및 z/OS 변환 서비스에서 지원되는 EBCDIC CCSID여야 합니다. 이 매개변수를 지정하지 않는 경우, 애플리케이션 데이터 구조는 시스템 초기화 매개변수에 지정된 CCSID를 사용하여 인코딩됩니다.

어느 맵핑 레벨에서나 이 매개변수를 사용할 수 있습니다.

CHAR-VARYING={NO**|****NULL****|****COLLAPSE****|****BINARY****}**

맵핑 레벨이 1.2 이상일 때 언어 구조의 문자 필드가 맵핑되는 방법을 지정합니다. COBOL의 문자 필드는 X 유형의 Picture 절입니다(예를 들어, PIC(X) 10). C/C++의 문자 필드는 문자 배열입니다. 이 매개변수는 Enterprise 및 기타 PL/I 언어 구조에 적용되지 않습니다. 다음 옵션을 선택할 수 있습니다.

NO 문자 필드가 JSON 문자열로 맵핑되고 고정 길이 필드로 처리됩니다. 데이터의 최대 길이는 필드의 길이와 동일합니다. NO가 맵핑 레벨 2.0 이하에서 COBOL 및 PL/I에 대한 **CHAR-VARYING** 매개변수의 기본값입니다.

NULL

문자 필드가 JSON 문자열로 맵핑되고 널(null) 종료 문자열로 처리됩니다. CICS는 JSON 스키마로부터 변환할 때 종료 널 문자를 추가합니다. 문자열의 최대 길이는 언어 구조에 표시된 길이보다 한 문자 적은 길이로 계산됩니다. NULL은 C/C++에 대한 **CHAR-VARYING** 매개변수의 기본값입니다.

COLLAPSE

문자 필드가 JSON 문자열로 맵핑됩니다. 이 필드의 후미 공백은 JSON 스키마에 포함되지 않습니다. COLLAPSE는 맵핑 레벨 2.1 이상에서 COBOL 및 PL/I에 대한 **CHAR-VARYING** 매개변수의 기본값입니다.

BINARY

문자 필드가 base64 인코딩 데이터를 포함한 JSON 문자열로 맵핑되고 고정 길이 필드로 처리됩니다. **CHAR-VARYING** 매개변수의 BINARY 값은 맵핑 레벨 2.1 이상에서만 사용 가능합니다.

CHAR-OCCURS={STRING|ARRAY}

맵핑 레벨이 4.0 이상일 때 언어 구조의 문자 배열이 맵핑되는 방법을 지정합니다.

예: PIC X OCCURS 20. 이 매개변수는 COBOL 언어에서만 사용됩니다.

ARRAY

문자 배열이 JSON 배열에 맵핑됩니다. 이것은 모든 문자가 개별 JSON 요소로서 맵핑됨을 의미합니다. 이것은 또한 맵핑 레벨 3.0 이하에서의 작동입니다.

STRING

문자 배열이 JSON 문자열로 맵핑됩니다. 이것은 전체 COBOL 배열이 단일 JSON 요소로서 맵핑됨을 의미합니다.

CHAR-USAGE={NATIONAL|DBCS}

COBOL에서는, 국가 데이터 유형, PIC N을 UTF-16 또는 DBCS 데이터에 사용할 수 있습니다. 이 설정은 NSYMBOL 컴파일러 옵션으로 제어합니다. 지원의 **CHAR-USAGE** 매개변수를 NSYMBOL 컴파일러 옵션과 동일한 값으로 설정해서 데이터가 적합하게 처리되도록 해야 합니다. UTF-16을 사용하는 경우 이는 일반적으로 CHAR-USAGE=NATIONAL로 설정됩니다.

DBCS PIC(*n*) 필드의 데이터가 UTF-16 인코딩 데이터로서 취급됩니다.

NATIONAL

PIC (*n*) 필드의 데이터가 DBCS 인코딩 데이터로 취급됩니다.

DATA-TRUNCATION={DISABLED|ENABLED}

가변 길이 데이터가 고정 길이 필드 구조에 용인되는지 여부를 지정합니다.

DISABLED

데이터가 CICS가 기대하고 있는 고정 길이보다 작은 경우, CICS는 잘린 데이터를 거부하고 오류 메시지를 발행합니다.

ENABLED

데이터가 CICS가 기대하고 있는 고정 길이보다 작은 경우, CICS는 잘린 데이터를 용인하고 누락된 데이터를 널(null)값으로 처리합니다.

DATETIME={UNUSED|PACKED15}

CICS ABSTIME 값을 포함하여 고급 언어 구조의 JSON 날짜-시간 특성이 시간 소인으로 매핑되는지 여부를 지정합니다.

PACKED15

JSON 날짜-시간 특성이 시간 소인으로 매핑됩니다.

UNUSED

JSON 날짜-시간 특성이 시간 소인으로 매핑되지 않습니다. 이 매핑이 기본값입니다.

매핑 레벨 3.0에서 이 매개변수를 설정할 수 있습니다.

JSON-SCHEMA=value

JSON 스키마를 쓰는 파일의 완전한 z/OS UNIX 이름입니다.

JSONTRANSFRM=value

이 매개변수는 LINKable 모드에 필수이지만 요청-응답 모드에는 올바르지 않습니다. CICS에서 **JSONTRANSFRM** 번들 자원에 사용되는 이름을 표시합니다.

LANG=COBOL

고급 언어 구조의 프로그래밍 언어가 COBOL임을 지정합니다.

LANG=PLI-ENTERPRISE

고급 언어 구조의 프로그래밍 언어가 Enterprise PL/I임을 지정합니다.

LANG=PLI-OTHER

고급 언어 구조의 프로그래밍 언어가 Enterprise PL/I 이외의 PL/I 레벨임을 지정합니다.

LANG=C

고급 언어 구조의 프로그래밍 언어가 C임을 지정합니다.

LANG=CPP

고급 언어 구조의 프로그래밍 언어가 C++임을 지정합니다.

LOGFILE=value

DFHLS2JS가 활동 로그 및 추적 정보를 쓰는 파일의 완전한 z/OS UNIX 이름입니다. 존재하지 않을 경우 DFHLS2JS는 디렉토리 구조가 아닌 파일을 작성합니다.

DFHLS2JS에 관한 문제가 발생할 경우 IBM 서비스 조직이 이 매개변수를 사용하도록 요청할 수 있습니다.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}

JSON 바인딩 및 언어 구조를 생성할 때 사용할 지원의 매핑 레벨을 지정합니다. 매핑 레벨 3.0 이상을 사용해야 합니다.

MINIMUM-RUNTIME-LEVEL={MINIMUM|3.0|4.0|CURRENT}

JSON 바인딩을 전개할 수 있는 최소 CICS 런타임 환경을 지정합니다. 지정한 기타 매개변수와 일치하지 않는 레벨을 선택할 경우 오류 메시지가 수신됩니다. 선택 가능한 옵션은 다음과 같습니다.

MINIMUM

CICS의 가능한 최저 런타임 레벨이 지정한 매개변수에 기반하여 자동으로 할당됩니다.

3.0 CICS JSON 지원을 사용하고 고급 데이터 매핑을 이용하려면 런타임 레벨 3.0 이상을 지정하십시오.

4.0 생성된 웹 서비스 바인딩 파일이 CICS TS 5.2 리전 이상에 전개됩니다. 이 런타임 레벨을 사용하면 **MAPPING-LEVEL** 매개변수에 대해 4.0 이하의 매핑 레벨을 사용할 수 있습니다. 이 레벨에서 임의의 선택적 매개변수를 사용할 수 있습니다.

CURRENT

런타임 환경이 바인딩 파일을 생성하는 데 사용한 환경으로 설정된 CICS 리전으로 생성된 바인딩 파일을 전개하려면 이 런타임 레벨을 사용하십시오.

OVERWRITE-OUTPUT={NO|YES}

파일 시스템의 기존 CICS 번들을 겹쳐쓸 수 있는지 여부를 제어합니다.

NO 기존 BUNDLE을 바꾸지 않습니다. 기존 BUNDLE이 있는 경우 DFHLS2JS는 오류 메시지 DFHPI9689E를 발행하고 종료됩니다.

YES 기존 BUNDLE을 바꿉니다. 기존 BUNDLE이 있으면 DFHPI9683W 메시지가 발행되어 파일을 바꿨음을 알립니다.

PDSCP=value

파티션된 데이터 세트 멤버에 사용되는 코드 페이지를 지정하며, 여기서 *value*는 CCSID 번호 또는 Java 코드 페이지 번호입니다. 이 매개변수를 지정하지 않으면 z/OS UNIX System Services 코드 페이지가 사용됩니다. 예를 들어, PDSCP=037을 지정할 수 있습니다.

PDSLIB=value

처리될 고급 언어 데이터 구조를 포함하는 파티션된 데이터 세트의 이름을 지정합니다.

제한사항: 파티션된 데이터 세트의 레코드는 80바이트의 고정 길이를 가져야 합니다.

PDSMEM=value

처리될 고급 언어 데이터 구조를 포함하는 파티션된 데이터 세트 멤버의 이름을 지정합니다.

STRUCTURE={DFHDATA|data}

C 및 C++의 최상위 레벨 데이터 구조 이름입니다. 기본값은 DFHDATA입니다.

```
//LS2JS JOB 'accounting information',name,MSGCLASS=A
// SET QT='''
//JAVAPROG EXEC DFHLS2JS,
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSON-SCHEMA=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/example.log
MAPPING-LEVEL=4.0
JSONTRANSFRM=EXAMPLE
BUNDLE=/u/exampleapp/bundles/exampleBundle
/*
```

DFHJS2LS: 링크 가능한 인터페이스의 JSON 스키마 대 고급 언어 변환

DFHJS2LS 카탈로그된 프로시저는 JSON 스키마에서 고급 언어 데이터 구조와 JSON 바인딩을 생성합니다. JSON을 구문 분석하거나 작성할 수 있는 CICS 프로그램을 작성하려 할 때 DFHJS2LS를 사용하십시오. DFHJS2LS의 작업 제어문, 기호 매개변수, 입력 매개변수, 설명이 나열됩니다.

DFHJS2LS JCL 프로시저는 *HLQ.XDFHINST* 데이터 세트에 설치되는데, *HLQ*는 CICS가 설치된 상위 레벨 규정자입니다.

DFHLS2JS 또는 DFHJS2LS 프로시저에 대한 관련 사용 모드는 사용자 요구사항에 따라 다릅니다.

- DFHLS2JS: 링크 가능 인터페이스에 대한 고급 언어 대 JSON 스키마 변환
- DFHJS2LS: 링크 가능 인터페이스에 대한 JSON 스키마 대 고급 언어 변환
- DFHLS2JS: 요청-응답 서비스를 위한 JSON 스키마 변환을 위한 고급 언어
- DFHJS2LS: 요청-응답 서비스에 대한 고급 언어 변환에 대한 JSON 스키마
- DFHJS2LS: RESTful 서비스를 위한 고급 언어 변환에 대한 JSON 스키마

DFHJS2LS의 작업 제어문

JOB 작업을 시작합니다.

EXEC

프로시저 이름(DFHJS2LS)를 지정합니다.

INPUT.SYSUT1 DD

입력을 지정합니다. 입력 매개변수는 입력 스트림에 지정됩니다. 하지만 데이터 세트나 파티션된 데이터 세트의 멤버에 정의할 수도 있습니다.

기호 매개변수

다음 기호 매개변수가 DFHJS2LS에 정의됩니다.

JAVADIR=*path*

DFHJS2LS가 사용하는 Java 디렉토리의 이름을 지정합니다. 이 매개변수의 값은 전체 경로 이름 */usr/lpp/path*를 제공하는 */usr/lpp/*에 추가됩니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **JAVADIR** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

PATHPREFIX=*prefix*

다른 매개변수에 사용된 z/OS UNIX 디렉토리를 확장하는 선택적 접두부를 지정합니다. 기본값은 빈 문자열입니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **PATHPREFIX** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

SERVICE=*value*

IBM 지원 센터에서 지시한 경우에만 이 매개변수를 사용하십시오.

TMPDIR=*tmpdir*

DFHJS2LS가 임시 작업공간으로 사용하는 z/OS UNIX의 디렉토리 위치를 지정합니다. 작업을 실행하는 사용자 ID에 이 디렉토리에 대한 읽기 및 쓰기 권한이 있어야 합니다.

기본값은 */tmp*입니다.

TMPPFILE=*tmpprefix*

DFHJS2LS가 임시 작업공간 파일의 이름을 구성하는 데 사용하는 접두부를 지정합니다.

기본값은 JS2LS입니다.

USSDIR=*path*

UNIX System Services 파일 시스템에서 CICS TS 디렉토리의 이름을 지정합니다. 이 매개변수의 값이 */usr/lpp/cicsts/*에 추가되어 */usr/lpp/cicsts/path*의 전체 경로 이름을 생성합니다. 기본값이 사용되는 경우 이것은 '.'(마침표)로 지정되어야 합니다.

일반적으로 이 매개변수를 지정하지 않습니다. 기본값은 **USSDIR** 매개변수에서 CICS 설치 작업(DFHISTAR)에 제공된 값입니다.

임시 작업공간

DFHJS2LS는 런타임 시에 다음의 세 임시 파일을 작성합니다.

tmpdir/tmpprefix.in

tmpdir/tmpprefix.out

tmpdir/tmpprefix.err

여기서,

*tmpdir*은 **TMPDIR** 매개변수에 지정된 값입니다.

*tmpprefix*는 **TMPFILE** 매개변수에 지정된 값입니다.

파일의 기본 이름(**TMPDIR** 및 **TMPFILE**이 지정되지 않았을 때)은 다음과 같습니다.

/tmp/JS2LS.in

/tmp/JS2LS.out

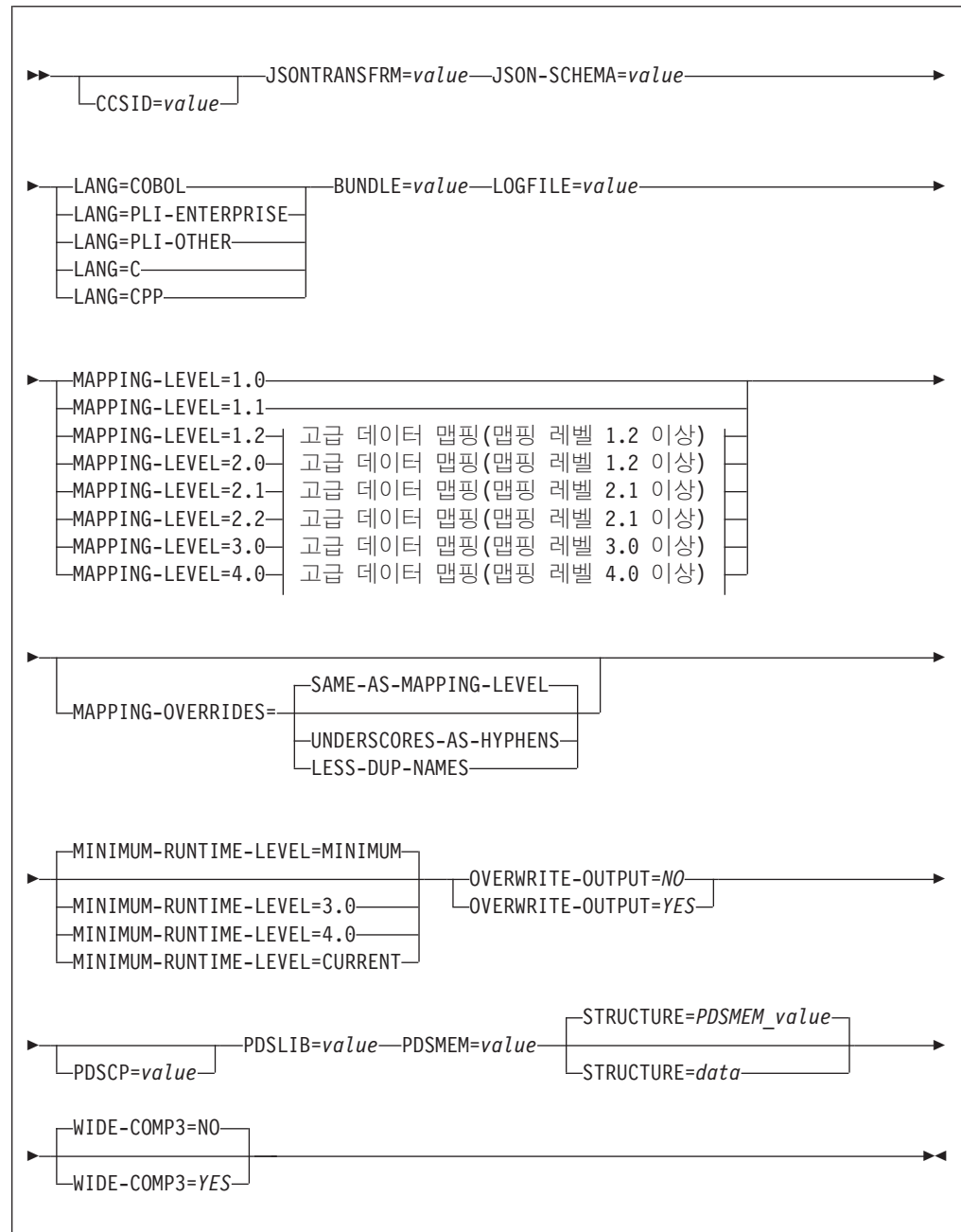
/tmp/JS2LS.err

중요사항: DFHJS2LS는 z/OS UNIX 파일이나 데이터 세트 멤버에 대한 액세스를 잠그지 않습니다. 따라서 둘 이상의 DFHJS2LS 인스턴스가 동시에 실행하며 동일한 임시 작업공간 파일을 사용하는 경우에는, 한 작업이 작업공간 파일을 사용하는 동안 또 다른 작업이 이 파일을 겹쳐쓰게 되어 예측 불가능한 장애가 발생할 수 있습니다.

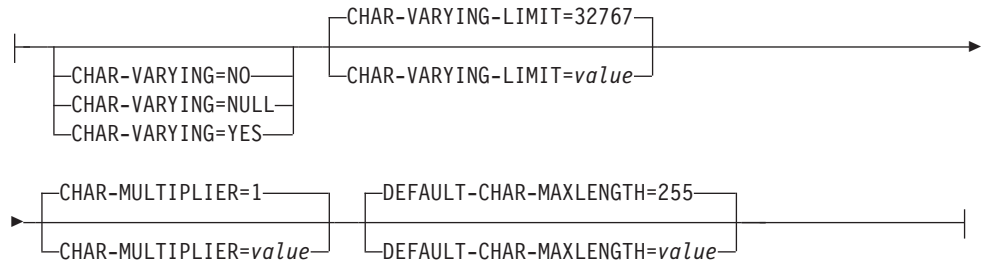
이러한 이유로, 이 같은 상황을 피하는 작동 프로시저와 이름 지정 규칙을 고안하는 것이 좋습니다. 예를 들어, 시스템 기호 매개변수 **SYSUID**를 사용하여 개별 사용자 고유의 작업 공간 파일 이름을 생성할 수 있습니다.

이들 임시 파일은 작업 종료 전에 삭제됩니다.

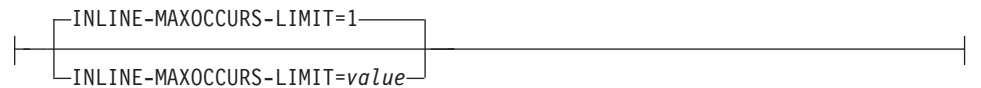
DFHJS2LS의 입력 매개변수



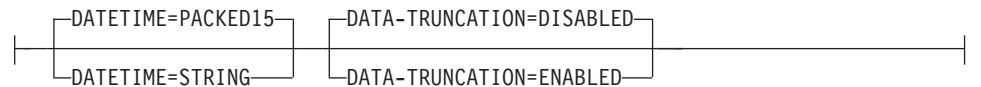
고급 데이터 맵핑(맵핑 레벨 1.2 이상):



고급 데이터 맵핑(맵핑 레벨 2.1 이상):



고급 데이터 맵핑(맵핑 레벨 3.0):



매개변수 사용

- 입력 매개변수를 어느 순서로든 지정할 수 있습니다.
- 각 매개변수는 새 줄에서 시작해야 합니다.
- 매개변수(및 연속 문자, 사용하는 경우)가 72열을 초과하면 안됩니다. 73 - 80열은 공백을 포함해야 합니다.
- 매개변수가 단일 행에 맞추기에 너무 긴 경우에는 행의 끝에 별표(*) 문자를 사용하여 매개변수가 다음 행에서 계속됨을 표시하십시오. 별표 이전의 모든 항목(공백을 포함하여)이 매개변수의 일부로 간주됩니다.
- 행의 첫 번째 문자 위치에 있는 # 문자는 주석 문자입니다. 해당 행은 무시됩니다.
- 행의 마지막 문자 위치에 있는 쉼표는 선택적인 행 분리문자이며 무시됩니다.

매개변수 설명

BUNDLE=value

z/OS UNIX의 번들 디렉토리 경로 및 이름을 지정합니다. 이 값을 지정하면 JSON 지원은 번들 디렉토리에 JSON 바인딩을 생성하고 번들 Manifest를 작성합니다. 이 매개변수의 경로 정보는 **JSONTRANSFRM** 매개변수의 경로 정보를 덮어씁니다.

CCSID=value

런타임 시에 애플리케이션 데이터 구조에서 문자 데이터를 인코딩하는 데 사용되는

CCSID를 지정합니다. 이 매개변수의 값이 **LOCALCCSID** 시스템 초기화 매개변수의 값을 덮어씁니다. *value* 값은 Java 및 z/OS 변환 서비스에서 지원되는 EBCDIC CCSID여야 합니다. 이 매개변수를 지정하지 않는 경우, 애플리케이션 데이터 구조는 시스템 초기화 매개변수에 지정된 CCSID를 사용하여 인코딩됩니다.

어느 맵핑 레벨에서나 이 매개변수를 사용할 수 있습니다.

CHAR-MULTIPLIER={1|*value*}

맵핑 레벨이 1.2 이상일 때 각 문자에 대해 허용할 바이트 수를 지정합니다. 이 매개변수의 *value*는 1 - 2,147,483,647 범위에 있는 양의 정수일 수 있습니다. 모든 비숫자 문자 기반 맵핑은 이 승수에 적용됩니다. 2진, 숫자, 존 및 팩형 10진수 필드는 이 승수가 적용되지 않습니다.

예를 들어 런타임 시에 모든 2바이트 문자 주위에 잠재적인 SO(shift-out) 및 SI(shift-in) 문자를 위한 공간을 허용하기 위해 3의 승수를 선택할 수 있는 DBCS 문자를 사용할 계획인 경우 이 매개변수가 유용할 수 있습니다.

CCSID=1200(UTF-16 표시)을 설정할 때, **CHAR-MULTIPLIER**에 대해 유일하게 유효한 값은 2 또는 4입니다. UTF-16을 사용할 때 기본값은 2입니다. 애플리케이션 데이터가 1 UTF-16 인코딩 단위가 필요한 문자를 포함할 것으로 예상할 때는 **CHAR-MULTIPLIER=2**를 사용하십시오. 애플리케이션 데이터가 2 UTF-16 인코딩 단위가 필요한 문자를 포함할 것으로 예상할 때는 **CHAR-MULTIPLIER=4**를 사용하십시오.

참고: **CHAR-MULTIPLIER**를 1로 설정해도 DBCS 문자 사용을 배제하지 않으며, 2로 설정해도 UTF-16 대리 쌍의 사용을 배제하지 않습니다. 그러나 와이드 문자가 일상적으로 사용되는 경우 일부 올바른 값이 할당된 필드에 맞지 않습니다. 더 큰 **CHAR-MULTIPLIER** 값이 사용되는 경우, XML에서 유효한 것보다 더 많은 문자를 할당된 필드에 저장할 수 있습니다. 적합한 범위 제한사항을 준수하도록 주의해야 합니다.

CHAR-VARYING={NO|NULL|YES}

맵핑 레벨이 1.2 이상일 때 가변 길이 문자 데이터가 맵핑되는 방법을 지정합니다. 가변 길이 2진 데이터 유형은 항상 컨테이너 또는 변동 구조로 맵핑됩니다. 이 매개변수를 지정하지 않을 경우 기본 맵핑은 지정한 언어에 따라 다릅니다. 다음 옵션을 선택할 수 있습니다.

NO 가변 길이 문자 데이터가 고정 길이 문자열로 맵핑됩니다.

NULL

가변 길이 문자 데이터가 널(null) 종료 문자열로 맵핑됩니다.

YES 가변 길이 문자 데이터가 PL/I에서 CHAR VARYING 데이터 유형으로 맵핑됩니다. COBOL, C, C++ 언어에서 가변 길이 문자 데이터는 두 개의 관련 요소, 데이터 길이 및 데이터로 이루어진 동등한 표현으로 맵핑됩니다.

CHAR-VARYING-LIMIT={32767|value}

맵핑 레벨이 1.2 이상일 때 언어 구조로 맵핑되는 가변 길이 문자 데이터와 2진 데이터의 최대 크기를 지정합니다. 문자나 2진 데이터가 이 매개변수에 지정된 값보다 크면 컨테이너로 맵핑되고 생성된 언어 구조에 컨테이너 이름이 사용됩니다. *value*의 가능한 범위는 0부터 기본값 32 767바이트까지입니다.

DATA-TRUNCATION={DISABLED|ENABLED}

가변 길이 데이터가 고정 길이 필드 구조에 용인되는지 여부를 지정합니다.

DISABLED

데이터가 CICS가 기대하고 있는 고정 길이보다 작은 경우, CICS는 잘린 데이터를 거부하고 오류 메시지를 발행합니다.

ENABLED

데이터가 CICS가 기대하고 있는 고정 길이보다 작은 경우, CICS는 잘린 데이터를 용인하고 누락된 데이터를 널(null)값으로 처리합니다.

DATETIME={PACKED15|STRING}

JSON 날짜-시간 특성이 CICS ABSTIME 데이터 형식이나 텍스트로 맵핑됨을 지정합니다.

PACKED15

JSON 날짜-시간 특성 필드가 CICS ABSTIME 형식으로 맵핑됩니다.

STRING

JSON 날짜-시간 특성이 텍스트로 맵핑됩니다. 이 맵핑은 이전의 모든 맵핑 레벨과 동일합니다.

맵핑 레벨 3.0에서 이 매개변수를 사용할 수 있습니다.

DEFAULT-CHAR-MAXLENGTH={255|value}

맵핑 레벨이 1.2 이상일 때 JSON 스키마 문서에 길이가 암시되지 않은 맵핑에 대한 문자에서 문자 데이터의 기본 배열 길이를 지정합니다. 이 매개변수의 가능한 *value*는 1 - 2 147 483 647 범위의 양의 정수입니다.

INLINE-MAXOCCURS-LIMIT={1|value}

JSON 스키마 키워드의 *maxItems* 속성을 기반으로 하여 인라인 변수 반복 내용이 사용되는지 여부를 지정합니다.

INLINE-MAXOCCURS-LIMIT 매개변수는 맵핑 레벨 1.2부터만 사용할 수 있습니다.

INLINE-MAXOCCURS-LIMIT의 *value*는 0 - 32 767의 범위의 양의 정수일 수 있습니다. 0 값은 인라인 맵핑을 사용하지 않음을 나타냅니다. 1 값은 선택적 요소가 인라인 맵핑되도록 합니다. *maxItems* 속성의 *value*가 **INLINE-MAXOCCURS-LIMIT**의 *value*보다 크면 컨테이너 기반 맵핑이 사용되고, 그렇지 않은 경우에는 인라인 맵핑이 사용됩니다.

가변적으로 반복되는 목록을 인라인 맵핑하도록 결정한 경우에는 반복 데이터의 단일 항목 길이를 고려하십시오. 길이가 긴 인스턴스가 거의 없는 경우 컨테이너 기반 맵핑이 바람직합니다. 짧은 길이의 인스턴스가 많으면 인라인 맵핑이 더 좋습니다.

JSONTRANSFRM=value

이 매개변수는 LINKable 모드에 필수이지만 요청 응답 및 RESTful 모드에는 유효하지 않습니다. CICS에서 **JSONTRANSFRM** 번들 자원에 사용되는 이름을 표시합니다.

JSON-SCHEMA=value

JSON 스키마를 읽는 파일의 완전한 z/OS UNIX 이름입니다. 아직 존재하지 않을 경우 DFHJS2LS는 디렉토리 구조가 아닌 파일을 작성합니다.

LANG=COBOL

고급 언어 구조의 프로그래밍 언어가 COBOL임을 지정합니다.

LANG=PLI-ENTERPRISE

고급 언어 구조의 프로그래밍 언어가 Enterprise PL/I임을 지정합니다.

LANG=PLI-OTHER

고급 언어 구조의 프로그래밍 언어가 Enterprise PL/I 이외의 PL/I 레벨임을 지정합니다.

LANG=C

고급 언어 구조의 프로그래밍 언어가 C임을 지정합니다.

LANG=CPP

고급 언어 구조의 프로그래밍 언어가 C++임을 지정합니다.

LOGFILE=value

DFHJS2LS가 활동 로그 및 추적 정보를 쓰는 파일의 완전한 z/OS UNIX 이름입니다. 아직 존재하지 않을 경우 DFHJS2LS는 디렉토리 구조가 아닌 파일을 작성합니다.

일반적으로는 이 파일을 사용하지 않지만 DFHJS2LS에 관한 문제가 발생할 경우 IBM 서비스 조직이 이 파일을 사용하도록 요청할 수 있습니다.

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0|4.0}

JSON 바인딩 및 언어 구조를 생성할 때 사용할 지원의 맵핑 레벨을 지정합니다. 사용 가능한 최신 맵핑 레벨을 사용해야 합니다. DFHJS2LS의 경우 맵핑 레벨 3.0 이상을 사용해야 합니다.

3.0 DFHJS2LS에 사용 가능한 최소 맵핑 레벨입니다.

4.0 UTF-16을 사용하려는 경우 CICS TS 5.2 리전에서는 이 맵핑 레벨을 사용하십시오.

MAPPING-OVERRIDES={SAME-AS-MAPPING-LEVEL|UNDERScores-AS-HYPHENS|LESS-DUP-NAMES}

언어 구조를 생성할 때 지정된 맵핑 레벨에 대한 기본 작동이 덮어써지는지 여부를 지정합니다.

LESS-DUP-NAMES

이 매개변수는 필드에 대한 직접 참조가 가능하도록 이름의 끝에 `_value` 를 갖는 비구조적 구조 필드 이름을 생성합니다. 예를 들어, 다음 PLI 언어 구조에서 `MAPPING-OVERRIDES=LESS-DUP-NAMES`가 지정될 때 레벨 12 필드 `streetName`은 `_value`가 접미부로 추가됩니다.

```
09 streetName,
   12 streetName                CHAR(255) VARYING
   UNALIGNED,
   12 filler                    BIT (7),
   12 attr_nil_streetName_value BIT (1),
```

결과 구조는 다음과 같습니다.

```
09 streetName,
   12 streetName_value          CHAR(255) VARYING
   UNALIGNED,
   12 filler                    BIT (7),
   12 attr_nil_streetName_value BIT (1),
```

SAME-AS-MAPPING-LEVEL

이 매개변수는 맵핑 레벨과 동일한 스타일로 언어 구조를 생성합니다. 이것이 기본값입니다.

UNDERScores-AS-HYPHENS

COBOL 전용입니다. 이 매개변수는 JSON 스키마에 있는 모든 밑줄을 문자 `X`가 아니라 하이픈으로 변환하여 생성된 COBOL 구조의 가독성을 개선합니다. 필드 이름 충돌이 발생하는 경우 필드를 고유하게 하기 위해 필드에 번호가 지정됩니다. 자세한 정보는 개발 애플리케이션에서 COBOL에 대한 JSON 스키마 맵핑의 내용을 참조하십시오.

MINIMUM-RUNTIME-LEVEL={MINIMUM|3.0|4.0|CURRENT}

JSON 바인딩을 전개할 수 있는 최소 CICS 런타임 환경을 지정합니다. 지정한 기타 매개변수와 일치하지 않는 레벨을 선택할 경우 오류 메시지가 수신됩니다. 선택 가능한 옵션은 다음과 같습니다.

MINIMUM

CICS의 가능한 최저 런타임 레벨이 지정한 매개변수에 기반하여 자동으로 할당됩니다.

3.0 CICS JSON 지원을 사용하고 고급 데이터 맵핑을 이용하려면 런타임 레벨 3.0 이상을 지정하십시오.

4.0 생성된 웹 서비스 바인딩 파일이 CICS TS 5.2 리전 이상에 전개됩니다.

이 런타임 레벨을 사용하면 **MAPPING-LEVEL** 매개변수에 대해 4.0 이하의 맵핑 레벨을 사용할 수 있습니다. 이 레벨에서 임의의 선택적 매개변수를 사용할 수 있습니다.

CURRENT

JSON 바인딩을 생성하는 데 사용한 리전과 런타임 환경이 동일한 CICS 리전으로 생성된 JSON 바인딩을 전개하려면 이 런타임 레벨을 사용하십시오.

OVERWRITE-OUTPUT={NO|YES}

파일 시스템의 기존 CICS 번들을 겹쳐쓸 수 있는지 여부를 제어합니다.

NO 기존 BUNDLE을 바꾸지 않습니다. 기존 BUNDLE이 있는 경우 DFHJS2LS는 오류 메시지 DFHPI9689E를 발행하고 종료됩니다.

YES 기존 BUNDLE을 바꿉니다. 기존 BUNDLE이 있으면 DFHPI9683W 메시지가 발행되어 파일을 바꿨음을 알립니다.

PDSCP=*value*

파티션된 데이터 세트 멤버에 사용되는 코드 페이지를 지정하며, 여기서 *value*는 CCSID 번호 또는 Java 코드 페이지 번호입니다. 이 매개변수를 지정하지 않으면 z/OS UNIX System Services 코드 페이지가 사용됩니다. 예를 들어, PDSCP=037을 지정할 수 있습니다.

PDSLIB=*value*

생성된 고급 언어를 포함하는 파티션된 데이터 세트의 이름을 지정합니다.

PDSMEM=*value*

DFHJS2LS가 고급 언어 구조를 포함할 파티션된 데이터 세트 멤버의 이름을 생성하기 위해 사용하는 1-6자의 접두부를 지정합니다.

DFHJS2LS는 각 조작의 파티션된 데이터 세트 멤버를 생성합니다. 접두부에 두 자리 숫자를 추가해서 멤버 이름을 생성합니다.

STRUCTURE={PDSMEM_*value*|data}

C 및 C++의 최상위 레벨 데이터 구조 이름입니다. 기본값은 PDSMEM 매개변수의 값입니다.

WIDE-COMP3={NO|YES}

COBOL 전용입니다. COBOL 언어 구조에서 팩형 10진수 가변 길이의 최대 크기를 제어합니다.

NO DFHJS2LS가 COBOL 언어 구조 유형 COMP-3를 생성할 때 팩형 10진수 가변 길이를 18로 제한합니다. 팩형 10진수 크기가 18보다 크면 DFHPI9022W 메시지가 발행되어 지정된 유형이 총 18자리 숫자로 제한되고 있음을 표시합니다.

YES DFHJS2LS가 COBOL 언어 구조 유형 COMP-3를 생성할 때 최대 크기 31을 지원합니다.

```
//JS2LS JOB 'accounting information',name,MSGCLASS=A
// SET QT='''
//JAVAPROG EXEC DFHJS2LS,
// TMPFILE=&QT.&SYSUID.&QT
/INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSON-SCHEMA=example.json
LANG=COBOL
LOGFILE=/u/exampleapp/example.log
MAPPING-LEVEL=4.0
CHAR-VARYING=NULL
JSONTRANSFRM=EXAMPLE
BUNDLE=/u/exampleapp/bundles/exampleBundle
/*
```

CICS JSON 지원의 �핑 레벨

�핑은 언어 구조와 JSON 스키마 간에 정보가 변환되는 방식을 지정하는 규칙 세트입니다. 사용 가능한 가장 정교한 �핑을 이용하려면 CICS 지원의 **MAPPING-LEVEL** 매개변수를 최신 레벨로 설정하는 것이 좋습니다.

각 �핑 레벨은 이전 �핑의 기능을 상속하며 최고 레벨의 �핑은 사용 가능한 최상의 기능을 제공합니다. 최고 �핑 레벨은 런타임에 데이터 변환에 대한 더 많은 제어를 제공하며 특정 데이터 유형과 JSON 특성에 대한 지원의 제한사항을 제거합니다.

이전에 해당 레벨에서 사용한 애플리케이션을 재전개하려는 경우에는 **MAPPING-LEVEL** 매개변수를 이전 레벨로 설정할 수 있습니다.

모든 �핑 레벨의 제한사항

- JSON 스키마에 사용된 데이터 유형을 명시적으로 선언해야 합니다.
- "\$ref" 및 "definition"과 같은 참조는 JSON 스키마 내에서 지원되지 않습니다. 예를 들어, 다음과 같습니다.

```
"definitions": {
  "schema1": {
    "id": "schema1",
    "type": "integer"
  },
  "schema2": {
    "type": "array",
    "items": { "$ref": "schema1" }
  }
}
```

�핑 레벨 4.0

이 �핑 레벨은 다음 지원을 제공합니다.

맵핑 레벨 4.0 이상에서, DFHLS2JS는 COBOL OCCURS DEPENDING ON 절을 지원하고 COBOL 문자 배열을 JSON 스키마로 맵핑하도록 지원합니다. CICS 지원에서 **CHAR-OCCURS** 매개변수를 사용하여 이 작동을 설정할 수 있습니다.

- **DATA-TRUNCATION=ENABLED** 매개변수를 지정해야 합니다.
- 복합 OCCURS DEPENDING ON은 지원되지 않습니다. 이 제한사항은 OCCURS DEPENDING ON이 구조의 마지막 필드에 대해서만 지원된다는 의미입니다.
- CICS는 규정된 이름('OF' 키워드 사용)을 OCCURS DEPENDING ON 절의 대상으로 지원하지 않습니다(예를 들어, FIELD1 OF STRUCTURE1).
- CICS는 UNBOUNDED 키워드를 지원하지 않습니다. 애플리케이션이 예상하는 테이블의 최대 크기에 대한 정수 한계를 지정해야 합니다.

맵핑 레벨 4.0 이상에서, JSON 웹 서비스는 UTF-16 유니코드를 사용하여 인코딩한 애플리케이션 데이터의 변환을 지원합니다.

- LS2JS를 사용하면 UTF-16에 대한 언어별 데이터 유형을 사용하여 이 작동을 사용 가능하게 할 수 있습니다.
- JS2LS를 사용하면 CCSID=1200을 설정해서 이 작동을 사용 가능하게 할 수 있습니다.
- CICS는 단일 유니코드 코드 페이지, 『IBM Private Use Area의 UTF-16BE』(CCSID 1200)만 지원합니다.
- UTF-8을 사용하여 인터딩한 애플리케이션 데이터의 변환은 지원되지 않습니다.

참고: DFHLS2JS는 COBOL GROUP USAGE NATIONAL 절을 지원하지 않습니다.

맵핑 레벨 3.0 이상

이 맵핑 레벨은 다음 지원을 제공합니다.

- DFHJS2LS는 date-time 데이터 유형을 CICS ASKTIME 형식으로 맵핑합니다.
- DFHLS2JS는 한 컨테이너 뿐이 아닌 많은 컨테이너를 사용하는 애플리케이션에서 JSON 스키마와 웹 서비스 바인딩을 생성할 수 있습니다.
- 고정 길이 데이터 구조로 설명된 잘린 데이터를 허용합니다. CICS 지원에서 **DATA-TRUNCATION** 매개변수를 사용하여 이 작동을 설정할 수 있습니다.

맵핑 레벨 2.2 이상

이 옵션은 SOAP 웹 서비스와의 호환성을 위해 유지됩니다. JSON에서의 사용은 권장하지 않습니다.

맵핑 레벨 2.1 이상

이 옵션은 SOAP 웹 서비스와의 호환성을 위해 유지됩니다. JSON에서의 사용은 권장하지 않습니다.

이 맵핑 레벨은 새 **INLINE-MAXOCCURS-LIMIT** 매개변수 및 **CHAR-VARYING** 매개변수의 새 값으로 변수 내용을 처리하는 방식에 대한 더 많은 제어를 제공합니다.

맵핑 레벨 2.1 이상에서 DFHJS2LS는 배열에 대한 다음 새 지원과 개선된 지원을 제공합니다.

- **INLINE-MAXOCCURS-LIMIT** 매개변수
- minItems 특성

INLINE-MAXOCCURS-LIMIT 매개변수는 가변적으로 반복되는 목록이 인라인 맵핑되는지 여부를 지정합니다. 가변적 반복 내용의 인라인 맵핑에 대한 자세한 정보는 634 페이지의 『DFHJS2LS의 변동 요소 배열』의 내용을 참조하십시오.

맵핑 레벨 2.1 이상에서 DFHLS2JS는 다음 JSON 맵핑을 지원합니다.

- COBOL 및 PL/I의 FILLER 필드는 무시됨
- **CHAR-VARYING** 매개변수의 COLLAPSE 값
- **CHAR-VARYING** 매개변수의 BINARY 값

COBOL 및 PL/I의 FILLER 필드는 무시됩니다. 생성된 JSON 스키마에 표시되지 않고 런타임에 데이터 구조에 해당 값이 남습니다.

COLLAPSE는 CICS가 텍스트 필드의 후미 공백을 무시하도록 합니다.

BINARY는 2진 필드에 대한 지원을 제공합니다. 이 값은 COBOL을 JSON 스키마로 변환할 때 유용합니다. 이 옵션은 SBCS 문자 배열에만 사용 가능하며 일반 문자열이 아닌 base64 인코딩 데이터를 포함한 고정 길이 JSON 문자열로 배열을 맵핑할 수 있게 합니다.

맵핑 레벨 1.2 이상

이 옵션은 SOAP 웹 서비스와의 호환성을 위해 유지됩니다. JSON에서의 사용은 권장하지 않습니다.

일괄처리 도구의 이 추가 매개변수를 통해 런타임에 문자 및 2진 데이터가 변환되는 방식에 대한 더 많은 제어가 사용 가능합니다.

- **CHAR-VARYING**
- **CHAR-VARYING-LIMIT**
- **CHAR-MULTIPLIER**
- **DEFAULT-CHAR-MAXLENGTH**

DFHJS2LS에서 **CHAR-MULTIPLIER** 매개변수를 사용하도록 결정한 경우에는 이 매개변수의 값을 사용하여 문자 데이터에 필요한 공간의 크기를 계산한 후 다음 규칙이 적용됨에 유의하십시오.

- DFHJS2LS는 다음 매핑을 제공합니다.
 - 최대 길이가 32 767바이트를 초과한 가변 길이 문자 데이터 유형은 컨테이너로 매핑됩니다. **CHAR-VARYING-LIMIT** 매개변수를 사용하여 하한을 설정할 수 있습니다. 컨테이너의 이름을 저장하기 위해 16바이트 필드가 언어 구조에 작성됩니다. 런타임에 문자 데이터는 컨테이너에 저장되고 컨테이너 이름은 언어 구조에 놓입니다.
 - 최대 길이가 32 768바이트 미만인 가변 길이 문자 데이터 유형은 C/C++ 및 Enterprise PL/I를 제외한 모든 언어에서 **VARYING** 구조로 매핑됩니다. C/C++에서는, 이 데이터 유형이 널(null) 종료 문자열로 매핑되고 Enterprise PL/I에서는 이 데이터 유형이 **VARYINGZ** 구조로 매핑됩니다. **CHAR-VARYING** 매개변수를 사용하여 가변 길이 문자 데이터가 매핑되는 방식을 선택할 수 있습니다.
 - 최대 길이가 32 768바이트 미만인 가변 길이 2진 데이터는 모든 언어에서 **VARYING** 구조로 매핑합니다. 최대 길이가 32 768바이트 이상인 경우에는 데이터가 컨테이너로 매핑됩니다. 컨테이너의 이름을 저장하기 위해 16바이트 필드가 언어 구조에 작성됩니다. 런타임에 2진 데이터는 컨테이너에 저장되고 컨테이너 이름은 언어 구조에 놓입니다.

JSON 스키마에 길이가 데이터 유형과 연관되지 않은 문자 데이터 유형이 있는 경우 DFHJS2LS에서 **DEFAULT-CHAR-MAXLENGTH** 매개변수를 사용하여 기본 길이를 지정할 수 있습니다.

DFHLS2JS는 다음 매핑을 제공합니다.

- 문자 필드는 string 데이터 유형으로 매핑되며 런타임에 고정 길이 필드나 널(null) 종료 문자열로 처리될 수 있습니다. **CHAR-VARYING** 매개변수를 사용하여 PL/I를 제외한 모든 언어에서 런타임에 가변 길이 문자 데이터가 처리되는 방식을 선택할 수 있습니다.
- Base64Binary 데이터 유형은 데이터의 최대 길이가 32 767바이트를 초과할 때에나 길이가 정의되지 않았을 때 컨테이너로 매핑됩니다. 데이터 길이가 32 767 이하인 경우에는 base64Binary 데이터 유형이 모든 언어에서 **VARYING** 구조로 매핑됩니다.

매핑 레벨 1.1 이상

이 옵션은 SOAP 웹 서비스와의 호환성을 위해 유지됩니다. JSON에서의 사용은 권장하지 않습니다.

이 매핑 레벨은 특히 JSON 스키마에 다른 값으로 정의된 maxLength 및 minLength 특성이 있는 가변 길이의 데이터를 매핑할 때 JSON 문자 및 2진 데이터 유형의 개선된 매핑을 제공합니다. 데이터는 다음과 같이 처리됩니다.

- 16MB를 초과하는 고정 길이의 문자 및 2진 데이터 유형은 PL/I를 제외한 모든 언어에서 컨테이너로 매핑됩니다. PL/I에서는, 32 767바이트를 초과한 고정 길이 문자

및 2진 데이터 유형이 컨테이너로 맵핑됩니다. 컨테이너의 이름을 저장하기 위해 16 바이트 필드가 언어 구조에 작성됩니다. 런타임에 고정 길이 데이터는 컨테이너에 저장되고 컨테이너 이름은 언어 구조에 놓입니다.

컨테이너는 가변 길이이므로 컨테이너로 맵핑되는 고정 길이 데이터는 JSON 스키마 또는 웹 서비스 설명에 지정된 고정 길이에 일치시키기 위해 공백 또는 널(null)로 채우지 않거나 자릅니다. 데이터의 길이가 중요한 경우에는 길이를 확인할 애플리케이션을 쓰거나 CICS 리전에서 유효성 검사를 켤 수 있습니다. 유효성 검사는 성능에 상당한 영향을 미칩니다.

맵핑 레벨 1.1만

이 옵션은 SOAP 웹 서비스와의 호환성을 위해 유지됩니다. JSON에서의 사용은 권장하지 않습니다.

이 맵핑 레벨은 특히 JSON 스키마에 다른 값으로 정의된 `maxLength` 및 `minLength` 특성이 있는 가변 길이의 데이터를 맵핑할 때 JSON 문자 및 2진 데이터 유형의 개선된 맵핑을 제공합니다. 데이터는 다음 방식으로 처리됩니다.

- 가변 길이 2진 데이터 유형은 컨테이너로 맵핑됩니다. 컨테이너의 이름을 저장하기 위해 16바이트 필드가 언어 구조에 작성됩니다. 런타임에 2진 데이터는 컨테이너에 저장되고 컨테이너 이름은 언어 구조에 놓입니다.
- 최대 길이가 32 767바이트를 초과한 가변 길이 문자 데이터 유형은 컨테이너로 맵핑됩니다. 컨테이너의 이름을 저장하기 위해 16바이트 필드가 언어 구조에 작성됩니다. 런타임에 문자 데이터는 컨테이너에 저장되고 컨테이너 이름은 언어 구조에 놓입니다.
- 16MB 미만의 고정 길이 문자 및 2진 데이터 유형은 PL/I를 제외한 모든 언어에서 고정 길이 필드로 맵핑됩니다. PL/I에서는, 32 767 이하의 고정 길이 문자 및 2진 데이터 유형이 고정 길이 필드로 맵핑됩니다.
- CICS는 `base64Binary` 형식이 아닌 `hexBinary` 형식의 데이터를 인코딩 및 디코딩합니다. JSON 스키마의 `Base64Binary` 데이터 유형은 언어 구조의 필드로 맵핑됩니다. 필드의 크기는 $4 \times (\text{ceil}(z/3))$ 공식을 사용하여 계산됩니다. 여기서,
 - z 는 JSON 스키마의 데이터 유형 길이입니다.
 - $\text{ceil}(x)$ 는 x 이상의 최소 정수입니다.

z 길이가 24 566바이트를 초과하는 경우 결과적인 언어 구조는 컴파일에 실패합니다. 24 566바이트를 초과하는 `base64Binary` 데이터가 있으면 맵핑 레벨 1.2를 사용하는 것이 좋습니다. 맵핑 레벨 1.2에서는 언어 구조의 필드를 사용하는 대신 `base64Binary` 데이터를 컨테이너로 맵핑할 수 있습니다.

맵핑 레벨 1.0만

이 옵션은 SOAP 웹 서비스와의 호환성을 위해 유지됩니다. JSON에서의 사용은 권장하지 않습니다.

이후의 맵핑 레벨에서 수정된 다음 제한사항에 유의하십시오.

- DFHJS2LS는 JSON 스키마의 문자 및 2진 데이터 유형을 언어 구조의 고정 길이 필드로 맵핑합니다. 다음의 부분 JSON 스키마를 보십시오.

```
"example":{
  "type":"string",
  "maxLength":33000
```

이 부분 JSON 스키마는 다음과 같은 COBOL 언어 구조에 표시됩니다.

```
15 example      PIC X(33000)
```

- CICS는 base64Binary 형식이 아닌 hexBinary 형식의 데이터를 인코딩 및 디코딩합니다. DFHJS2LS는 Base64Binary 데이터를 고정 길이 문자 필드로 맵핑하며, 이 내용은 애플리케이션을 통해 인코딩 또는 디코딩해야 합니다.
- DFHLS2JS는 언어 구조의 문자 및 2진 필드를 고정 길이 필드로 해석하고 이 필드를 maxLength 특성이 있는 JSON 문자열로 맵핑합니다. 런타임에 사용 가능한 데이터가 충분하지 않은 경우 언어 구조의 필드를 공백 또는 널(null)로 채웁니다.

고급 언어 및 JSON 스키마 맵핑

CICS JSON 지원을 사용하여 고급 언어 구조와 JSON 스키마 간의 맵핑을 생성합니다. CICS JSON 지원은 고급 언어 데이터 구조로부터 JSON 스키마를 생성하기도 합니다(반대의 경우도 마찬가지).

유틸리티 프로그램 DFHJS2LS 및 DFHLS2JS를 총괄하여 CICS JSON 지원이라 합니다.

- DFHLS2JS는 고급 언어 구조를 JSON 스키마로 맵핑합니다.
- DFHJS2LS는 JSON 스키마를 고급 언어 구조로 맵핑합니다.

두 개의 맵핑은 대칭적이지 않습니다.

- DFHLS2JS로 언어 데이터 구조를 처리한 후 결과적인 JSON 스키마를 DFHJS2LS로 처리하는 경우 최종 데이터 구조가 원래와 같을 것이라 예상하지 마십시오.
- JSON 스키마를 DFHJS2LS로 처리한 후 결과적인 언어 구조를 DFHLS2JS로 처리하는 경우 최종 JSON 스키마가 원래와 같을 것이라 예상하지 마십시오.
- 일부 경우 DFHJS2LS는 DFHLS2JS에서 지원되지 않는 언어 구조를 생성합니다.

CICS가 지원하는 언어 컴파일러에 구현된 대로, 언어 규칙에 따라 DFHLS2JS로 처리한 고급 언어 구조를 코딩해야 합니다.

COBOL 대 JSON 스키마 맵핑

DFHLS2JS 유틸리티 프로그램은 COBOL 데이터 구조와 JSON 스키마 정의 간의 맵핑을 지원합니다.

COBOL 이름은 다음 규칙에 따라 JSON 이름으로 변환됩니다.

- 중복 이름은 하나 이상의 숫자를 추가해서 고유 이름이 되게 합니다.

예를 들어 `year`의 두 인스턴스는 `year` 및 `year1`이 됩니다.

- 하이픈은 밑줄로 바꿉니다. 연속 하이픈 문자열은 연속 밑줄로 바꿉니다.

예를 들어 `current-user--id`는 `current_user__id`가 됩니다.

- 하이픈으로 구분하며 대문자만 포함된 이름의 세그먼트는 소문자로 변환합니다.

예를 들어 `CA-REQUEST-ID`는 `ca_request_id`가 됩니다.

- 숫자로 시작하는 이름은 이름 앞에 밑줄이 추가됩니다.

예를 들어 `9A-REQUEST-ID`는 `_9a_request_id`가 됩니다.

CICS는 다음 테이블에 따라 COBOL 데이터 설명 요소를 스키마 요소로 맵핑합니다. 테이블에 표시되지 않은 COBOL 데이터 설명 요소는 DFHLS2JS에서 지원되지 않습니다. 다음 제한사항도 적용됩니다.

- 레벨 번호가 66 및 77인 데이터 설명 항목은 지원되지 않습니다. 레벨 번호가 88인 데이터 설명 항목은 무시됩니다.
- 데이터 설명 항목에 대한 다음 절은 지원되지 않습니다.

REDEFINES

RENAMES; 즉 레벨 66

DATE FORMAT

- 데이터 설명 항목에 대한 다음 절은 무시됩니다.

BLANK WHEN ZERO

JUSTIFIED

VALUE

- SIGN 절 SIGN TRAILING은 지원됩니다. SIGN 절 SIGN LEADING은 DFHLS2JS의 지정된 맵핑 레벨이 1.2 이상일 때에만 지원됩니다.
- SEPARATE CHARACTER는 SIGN TRAILING 및 SIGN LEADING 절 모두, 맵핑 레벨 1.2 이상에서 지원됩니다.
- USAGE 절의 다음 구문은 지원되지 않습니다.

OBJECT REFERENCE

POINTER

FUNCTION-POINTER

PROCEDURE-POINTER

- USAGE 절의 다음 구문은 맵핑 레벨 1.2 이상에서 지원됩니다.

COMPUTATIONAL-1

COMPUTATIONAL-2

- DISPLAY 및 COMPUTATIONAL-5 데이터 설명 항목에 지원되는 유일한 PICTURE 문자는 9, S, Z입니다.
- PACKED-DECIMAL 데이터 설명 항목에 지원되는 PICTURE 문자는 9, S, V, Z입니다.
- 편집한 숫자 데이터 설명 항목에 지원되는 유일한 PICTURE 문자는 9 및 Z입니다.
- MAPPING-LEVEL 매개변수가 1.2 이상으로 설정되고 CHAR-VARYING 매개변수가 NULL로 설정된 경우 문자 배열은 string으로 맵핑되고 널(null) 종료 문자 열로 처리됩니다.
- MAPPING-LEVEL 매개변수가 1.2 이상으로 설정되고 CHAR-VARYING 매개변수가 BINARY로 설정된 경우에는 문자 배열이 string으로 맵핑되고 2진 데이터로 처리됩니다.
- MAPPING-LEVEL 매개변수가 1.2 이상으로 설정되고 CHAR-VARYING 매개변수가 COLLAPSE로 설정되면 문자열의 후미 공백이 무시됩니다.
- OCCURS DEPENDING ON 절은 맵핑 레벨 4.0 이상에서 지원됩니다. 복합 OCCURS DEPENDING ON은 지원되지 않습니다. 이는 OCCURS DEPENDING ON이 구조의 마지막 필드에 대해서만 지원된다는 의미입니다.
- OCCURS INDEXED BY 절은 어느 맵핑 레벨에서나 지원됩니다.

COBOL 데이터 설명	JSON 스키마 정의
PIC X(<i>n</i>) PIC A(<i>n</i>) PIC G(<i>n</i>) DISPLAY-1 PIC N(<i>n</i>)	"type":"string", "maxLength": <i>n</i>
PIC S9 DISPLAY PIC S99 DISPLAY PIC S999 DISPLAY PIC S9999 DISPLAY PIC S9(<i>n</i>) DISPLAY PIC S9(<i>n</i>) COMP PIC S9(<i>n</i>) COMP-4 PIC S9(<i>n</i>) COMP-5 PIC S9(<i>n</i>) BINARY	"type":"integer", "minimum":- (<i>n</i> + 1), "maximum": <i>n</i> 여기서, <i>n</i> 은 '9'자리 패턴으로 표현 가능한 최대값입니다.

COBOL 데이터 설명	JSON 스키마 정의
PIC 9 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY PIC 9(<i>n</i>) DISPLAY PIC 9(<i>n</i>) COMP PIC 9(<i>n</i>) COMP-4 PIC 9(<i>n</i>) COMP-5 PIC 9(<i>n</i>) BINARY	<pre>"type": "integer", "minimum": 0, "maximum": <i>n</i></pre> 여기서, <i>n</i>은 '9'자리 패턴으로 표현 가능한 최대값입니다.
PIC S9(<i>m</i>)V9(<i>n</i>) COMP-3	<pre>"type": "number", "description": "decimal", "minimum": <i>x</i>, "maximum": <i>y</i>, "multipleOf": <i>z</i></pre> 여기서, <i>x</i>는 '9'자리 패턴으로 표현 가능한 최소값이고, <i>y</i>는 '9'자리 패턴으로 표현 가능한 최대값이며, <i>z</i>는 사용 가능한 최소 단위 = 1 / 10^{<i>n</i>}입니다.
PIC 9(<i>m</i>)V9(<i>n</i>) COMP-3	<pre>"type": "number", "description": "decimal", "minimum": 0, "maximum": <i>y</i>, "multipleOf": <i>z</i></pre> 여기서, <i>y</i>는 '9'자리 패턴으로 표현 가능한 최대값이며, <i>z</i>는 사용 가능한 최소 단위 = 1 / 10^{<i>n</i>}입니다.
PIC S9(15) COMP-3 DATETIME=PACKED15일 때 맵핑 레벨 3.0에서 지원됨	<pre>"type": "string", "format": "date-time"</pre> 시간 소인 형식은 RFC3339를 통해 정의됩니다.
PIC S9(<i>m</i>)V9(<i>n</i>) DISPLAY 맵핑 레벨 1.2 이상에서 지원됨	<pre>"type": "number", "description": "decimal", "minimum": <i>x</i>, "maximum": <i>y</i>, "multipleOf": <i>z</i></pre> 여기서, <i>x</i>는 '9'자리 패턴으로 표현 가능한 최소값이고, <i>y</i>는 '9'자리 패턴으로 표현 가능한 최대값이며, <i>z</i>는 사용 가능한 최소 단위 = 1 / 10^{<i>n</i>}입니다.

COBOL 데이터 설명	JSON 스키마 정의
COMP-1 맵핑 레벨 1.2 이상에서 지원됨 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요 요소인 경우에는 COMP-1 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.	<pre>"type":"number", "description":"float"</pre>
COMP-2 맵핑 레벨 1.2 이상에서 지원됨 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 double 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요 요소인 경우에는 COMP-2 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오. .	<pre>"type":"number", "description":"double"</pre>
<i>data description OCCURS n TIMES</i>	기본요소의 경우: <pre>"type":"array" "maxItems":n "minItems":n "items":{ "type":"object", "properties":{ name:{ data description JSON } } "required":[name] }</pre> 구조의 경우: <pre>"type":"array" "maxItems":n "minItems":n "items":{ data description JSON }</pre> 여기서, <i>data description JSON</i>은 COBOL <i>data description</i>의 JSON 스키마이고 <i>name</i>은 COBOL <i>data description</i>의 이름입니다.

COBOL 데이터 설명	JSON 스키마 정의
<i>data description</i> OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDENT ON <i>t</i> 맵핑 레벨 4.0에서 지원됨	<pre>"field-name":{ "type":"array", "maxItems":<i>m</i> "minItems":<i>n</i> "items":{ ... } }</pre> 배열 항목의 내용은 사용한 데이터 유형에 따라 다릅니다.
PIC X OCCURS <i>n</i> TIMES PIC A OCCURS <i>n</i> TIMES PIC G DISPLAY-1 OCCURS <i>n</i> TIMES PIC N OCCURS <i>n</i> TIMES	CHAR-OCCURS=STRING일 때: <pre>"field-name":{ "type":"string", "maxLength":<i>n</i> }</pre> 문자열입니다. CHAR-OCCURS=ARRAY일 때: <pre>"field-name":{ "maxItems":<i>m</i>, "minItems":<i>n</i>, "items":{ "type":"object", "properties":{ "field-name":{ "type":"string", "maxLength":1 } }, "required":["field-name"] } }</pre> 단일 문자 배열입니다.
PIC X OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDENT ON <i>t</i> PIC A OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDENT ON <i>t</i> PIC G DISPLAY-1 OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDENT ON <i>t</i> PIC N OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDENT ON <i>t</i>	CHAR-OCCURS=STRING일 때: <pre>"field-name":{ "type":"string", "maxLength":<i>m</i> "minLength":<i>n</i> }</pre>
PIC N(<i>n</i>) USAGE NATIONAL CHAR-USAGE=NATIONAL일 때: PIC N(<i>n</i>)	맵핑 레벨 4.0 이상: <pre>"type":"string", "maxLength":<i>n</i></pre> 런타임에 CICS는 애플리케이션 데이터 구조 필드를 UTF-16 데이터로 채웁니다.

관련 참조:

『JSON 스키마 대 COBOL 맵핑』

DFHJS2LS 유틸리티 프로그램은 JSON 스키마와 COBOL 데이터 구조 간의 맵핑을 지원합니다.

JSON 스키마 대 COBOL 맵핑

DFHJS2LS 유틸리티 프로그램은 JSON 스키마와 COBOL 데이터 구조 간의 맵핑을 지원합니다.

CICS 지원은 다음 규칙을 사용하여 스키마 요소 이름에서 COBOL 변수의 올바른 고유 필드 이름을 생성합니다.

1. COBOL 예약어에는 'X' 접두부가 붙습니다.

예를 들어, DISPLAY는 XDISPLAY가 됩니다.

2. A - Z, a - z, 0 - 9 또는 하이픈 이외의 문자는 'X'로 바꿉니다.

예를 들어, monthly_total은 monthlyXtotal이 됩니다.

3. 마지막 문자가 하이픈이면 'X'로 바꿉니다.

예를 들어, ca-request-는 ca-requestX가 됩니다.

4. 동일한 범위의 중복된 이름은 이름의 두 번째 및 후속 인스턴스에 하나 또는 두 개의 숫자를 추가해서 이름을 고유하게 합니다.

예를 들어, year의 세 가지 인스턴스는 year, year1, year2가 됩니다.

5. JSON 스키마는 "type" 값이 "array"이고 "minItems" 및 "maxItems" 키워드가 생략되거나 값이 다른 경우 변수의 변동 카디널리티가 있음을 지정합니다. 변수의 변동 카디널리티가 있다고 스키마에 지정되면 "_cont" 및 "_num" 접미부가 붙은 필드 이름이 작성됩니다.

자세한 정보는 634 페이지의 『DFHJS2LS의 변동 요소 배열』의 내용을 참조하십시오.

6. JSON 스키마는 엔클로징 JSON 스키마 "object" 유형과 연관된 "required" 키워드 배열에 표시되지 않는 경우 변수가 선택적이라고 지정합니다. 선택적 필드의 경우에는 요소 이름에 _num 접미부가 추가된 추가 필드가 생성됩니다. 런타임에 이는 JSON 데이터에 값이 없음을 나타내는 0 및 JSON 데이터에 값이 있으면 0 이외의 값입니다.
7. 필드 이름은 28자로 제한됩니다. 접두부 및 접미부를 포함하여 생성된 이름이 이 길이를 초과하는 경우 요소 이름이 잘립니다.

DFHJS2LS는 다음 테이블에 따라 지정된 맵핑 레벨을 사용하여 스키마 유형을 COBOL 데이터 설명 요소로 맵핑합니다. 다음 사항에 유의하십시오.

- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정되면 가변 길이 문자 데이터는 널(null) 종료 문자열로 매핑되고 널 종료자에 대한 여분의 문자가 할당됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 YES로 설정된 경우에는 가변 길이 문자 데이터가 관련된 두 개의 요소, 길이 필드 및 데이터 필드로 매핑됩니다. 예를 들어, 다음과 같습니다.

```
"textString": {
  "type": "string",
  "maxLength": 10000,
  "minLength": 1
}
```

은 다음으로 매핑됩니다.

```
15 textString-length    PIC S9999 COMP-5 SYNC
15 textString           PIC X(10000)
```

JSON 스키마 키워드	COBOL 데이터 설명
다음 모두: "type": "null" "type": [] "enum": [] "allOf" "anyOf" "noneOf" "not" "definitions"	지원되지 않음
"\$schema": "http://json-schema.org/draft-04/schema#"	이 키워드는 무시되지만 드래프트 04 JSON 스키마 스펙과 호환 가능할 것이라 가정됩니다.
"title": "same text" "description": "more text"	이 키워드는 무시됩니다.
"format": "<predefined values>"	"format" 키워드는 생성된 구조나 런타임 값을 수정하는 데 사용됩니다. "format"의 지원되는 사용에 대해서는 이 테이블 후반부의 정보를 참조하십시오.
"type": "array", "items": {<JSON Sub-schema>}, "additionalItems": false, "maxItems": m, "minItems": n	현재 지원되는 JSON 배열의 유일한 양식은 반복된 수의 동일한 유형 값입니다. <JSON Sub-schema>는 지원되는 "type"을 정의해야 합니다. 하지만 "type"이 "array"이면 안됩니다. 이는 생성되는 언어 구조에 대한 제한사항입니다. "additionalItems"는 false로 가정되며 다른 값은 지원되지 않습니다. "minItems" 및 "maxItems"가 모두 있고 동일하면 배열이 고정 카디널리티로 처리되고 그렇지 않은 경우에는 변동 카디널리티로 처리됩니다. 634 페이지의 『DFHJS2LS의 변동 요소 배열』의 내용을 참조하십시오.
"type": "array", "uniqueItems": true	"uniqueItems"는 JSON 배열에 지원되지 않습니다.

JSON 스키마 키워드	COBOL 데이터 설명
<pre>"type": "object", "additionalProperties": false, "properties": { ["<element name>": {<JSON Sub-schema> } [,]]* } "required": [["<element name>" [,]]*]</pre>	현재 지원되는 JSON 오브젝트의 유일한 양식은 이름 요소의 고정 세트입니다. 이는 요소 이름을 사용하는 구조(또는 하위 구조)를 생성합니다. "additionalProperties"는 false로 가정되며 다른 값은 지원되지 않습니다. "properties" 오브젝트의 요소는 "required" 배열에 있지 않거나 "required" 배열이 없으면 "optional"로 간주됩니다. "optional" 요소에는 0 - X의 가변 서수가 제공되며, 여기서 X는 1 또는 해당 항목이 배열로 정의된 배열의 최대 항목 수입니다. 634 페이지의 『DFHJS2LS의 변동 요소 배열』의 내용을 참조하십시오.
<pre>"type": "object", "maxProperties": m, "minProperties": n, "patternProperties": {}, "dependencies":</pre>	이 키워드는 JSON 오브젝트에 지원되지 않습니다.
<pre>"type": "string" "maxLength": m "pattern": "regular expression", "minLength": l</pre>	PIC X(z) 여기서, z 값은 m에 기초하지만 CHAR-VARYING 매개변수의 설정에 종속됩니다. m은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다. "pattern" 및 "minLength" 제한사항은 언어 구조에 주석으로만 전달됩니다.
<pre>"type": "string" "maxLength": m</pre>	맵핑 레벨 4.0 이상에서 CCSID=1200일 때: PIC N(z) USAGE NATIONAL 여기서, z 값은 m에 기초하지만 CHAR-VARYING 매개변수의 설정에 종속됩니다. m은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다.
<pre>"*name*": { "type": "string", "format": "date-time" }</pre>	PIC S9(15) COMP-3 DATETIME=PACKED15일 때 모두 지원됨 "maxLength" 및 "minLength"는 이 형식에 지원되지 않음에 유의하십시오.

JSON 스키마 키워드	COBOL 데이터 설명
<pre>"*name*":{ "type":"string", "format":"uri" }</pre>	PIC X(<i>m</i>) 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다. 맵핑 레벨 4.0 이상에서 CCSID=1200일 때: PIC N(<i>m</i>) USAGE NATIONAL 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다.
<pre>"*name*":{ "type":"string", "format":"base64Binary" }</pre>	PIC X(<i>m</i>) 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 합니다.
<pre>"*name*":{ "type":"string", "format":"hexBinary" }</pre>	PIC X(<i>m</i>) 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 합니다.
<pre>"*name*":{ "type":"string", "format":"<predefined>" }</pre>	PIC X(<i>m</i>) 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리되고 <predefined>는 <i>email</i>, <i>hostname</i>, <i>ipv4</i> 또는 <i>ipv6</i> 중 하나입니다. 관련 "pattern"이 사용되고 주석에 전달됩니다. 맵핑 레벨 4.0 이상에서 CCSID=1200일 때: PIC N(<i>m</i>) USAGE NATIONAL 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리되고 <predefined>는 <i>email</i>, <i>hostname</i>, <i>ipv4</i> 또는 <i>ipv6</i> 중 하나입니다. 관련 "pattern"이 사용되고 주석에 전달됩니다.
<pre>"type":"boolean"</pre>	PIC X DISPLAY x'00' 값은 false를 의미하고 x'01'은 true를 의미합니다.
<pre>"type": "integer", "maxExclusive": true, "minExclusive": true, "multipleOf": n</pre>	"maxExclusive" 및 "minExclusive" 제한사항은 언어 구조에 주석으로만 전달됩니다. "multipleOf"는 무시됩니다.

JSON 스키마 키워드	COBOL 데이터 설명
"type"="integer", minimum=0, maximum=255	맵핑 레벨 3.0 이하: PIC X DISPLAY 맵핑 레벨 4.0 이상: PIC 9(z) COMP-5 SYNC 또는 PIC 9(z) DISPLAY 여기서, $10^{(z-1)} < m \leq 10^z$
"type": "integer", minimum:-128, maximum:127	맵핑 레벨 3.0 이하: PIC X DISPLAY 맵핑 레벨 4.0 이상: PIC S9(z) COMP-5 SYNC 또는 PIC S9(z) DISPLAY 여기서, $10^{(z-1)} < m \leq 10^z$
"type": "integer", minimum:0, maximum;m	PIC 9(z) COMP-5 SYNC 또는 PIC 9(z) DISPLAY 여기서, $10^{(z-1)} < m \leq 10^z$
"type": "integer", minimum:-m, maximum:m-1	PIC S9(z) COMP-5 SYNC 또는 PIC S9(z) DISPLAY 여기서, $10^{(z-1)} < m \leq 10^z$
"type": "number", "maximum": m, "minimum": n, "maxExclusive": true, "minExclusive": true, "multipleOf": n	"maximum", "minimum", "maxExclusive", "minExclusive" 제한사항은 언어 구조에 주석 으로만 전달됩니다. "multipleOf"는 무시됩니다.
"type": "number" "format": "decimal"	PIC 9(p)V9(n) COMP-3 여기서, p 및 n은 기본값입니다.

JSON 스키마 키워드	COBOL 데이터 설명
<pre>"type": "number" "format": "float"</pre>	맵핑 레벨 1.1 이하: • PIC X(32) 맵핑 레벨 1.2 이상: • COMP-1 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요 요소인 경우에는 COMP-1 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.
<pre>"type": "number" "format": "double"</pre>	맵핑 레벨 1.1 이하: • PIC X(32) 맵핑 레벨 1.2 이상: • COMP-2 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 double 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요 요소인 경우에는 COMP-2 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오. .

참고: CICS는 부호 있는 long($2^{63} - 1$)의 최대값보다 큰 정수 값을 따옴표(')로 묶지 않으면 변환할 수 없습니다.

참고: 숫자 유형에 대한 스키마에 지정된 최소 및 최대값은 COBOL 데이터 유형으로 맵핑하는 용도로만 사용됩니다. 런타임에 이 값에 대해 데이터의 유효성을 검사하지 않습니다.

테이블에 표시된 일부 스키마 유형은 minimum 및 maximum 키워드에 지정된 값에 따라(있는 경우) COMP-5 SYNC 또는 DISPLAY의 COBOL 형식으로 맵핑됩니다.

- 부호 있는 유형(short, int, long)의 경우 다음이 지정되면 DISPLAY가 사용됩니다.

```
"maximum":a
"minimum":-a
```

여기서, *a*는 '9' 문자열입니다.

- 부호 없는 유형(unsignedShort, unsignedInt, unsignedLong)의 경우에는 다음이 지정되면 DISPLAY가 사용됩니다.

```
"maximum":a
"minimum":0
```

여기서, *a*는 '9' 문자열입니다.

다른 모든 값이 지정되거나 값이 지정되지 않은 경우에는 COMP-5 SYNC가 사용됩니다.

관련 참조:

602 페이지의 『COBOL 대 JSON 스키마 맵핑』

DFHLS2JS 유틸리티 프로그램은 COBOL 데이터 구조와 JSON 스키마 정의 간의 맵핑을 지원합니다.

C 및 C++ 대 JSON 스키마 맵핑

DFHLS2JS 유틸리티 프로그램은 C 및 C++ 데이터 유형과 JSON 스키마 정의 간의 맵핑을 지원합니다.

C 및 C++ 이름은 다음 규칙에 따라 JSON 이름으로 변환됩니다.

- JSON 특성 이름에 유효하지 않은 문자는 'X'로 바꿉니다.

예를 들어, monthly-total은 monthlyXtotal이 됩니다.

- 중복 이름은 하나 이상의 숫자를 추가해서 고유 이름이 되게 합니다.

예를 들어 year의 두 인스턴스는 year 및 year1이 됩니다.

DFHLS2JS는 다음 테이블에 따라 C 및 C++ 데이터 유형을 스키마 요소로 맵핑합니다. 테이블에 표시되지 않은 C 및 C++ 유형은 DFHLS2JS에서 지원되지 않습니다. 구조에 대한 _Packed 규정자가 지원됩니다. 다음과 같은 제한사항이 적용됩니다.

- 헤더 파일은 최상위 레벨 struct 인스턴스를 포함해야 합니다.
- 자체를 멤버로 포함하는 구조 유형은 선언할 수 없습니다.
- 다음 C 및 C++ 데이터 유형은 지원되지 않습니다.

```
decimal
long double
```


wchar_t(C++만 해당)

- 헤더 파일에 다음이 표시되는 경우 무시됩니다.

저장영역 클래스 지정자:

auto
register
static
extern
mutable

규정자

const
volatile
_Export(C++만 해당)

함수 지정자

inline(C++만 해당)
virtual(C++만 해당)

초기값

- 헤더 파일에 다음 항목이 포함되면 안됩니다.

공용체

클래스 선언

Enumeration 데이터 유형

포인터 유형 변수

템플릿 선언

사전 정의된 매크로. 즉, 이름이 두 개의 밑줄 문자(____)로 시작하고 끝나는 매크로

행 연속 시퀀스(줄 바꾸기 문자 바로 앞에 있는 \ 기호)

프로토타입 함수 선언자

프리프로세서 지시문

비트 필드

__cdecl(또는 _cdecl) 키워드(C++만 해당)

- 애플리케이션 프로그래머는 int가 4바이트로 맵핑되도록 32비트 컴파일러를 사용해야 합니다.
- 다음 C++ 예약 키워드는 지원되지 않습니다.

explicit
using
namespace
typename

typeid

- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정된 경우 문자 배열은 string으로 맵핑되고 널(null) 종료 문자열로 처리됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 BINARY로 설정되면 문자 배열은 xsd:base64Binary로 맵핑되며 2진 데이터로 처리됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 COLLAPSE로 설정된 경우에는 문자열에 대해 <xs:whiteSpace value="collapse"/>가 생성됩니다.

C 및 C++ 데이터 유형	스키마 simpleType
char[z]	"type":"string" "maxLength":z
char16_t[n]	맵핑 레벨 4.0 이상: "type":"string" "maxLength":n 런타임에 CICS는 애플리케이션 데이터 구조 필드를 UTF-16 데이터로 채웁니다.
char[8] DATETIME=PACKED15일 때 맵핑 레벨 3.0 이상에서 지원됨	"type":"string" "format":"date-time" 시간 소인 형식은 RFC3339를 통해 정의됩니다.
char short int long long long	"type":"integer", "minimum":- (n + 1), "maximum":n 여기서, n은 기본요소로 표현할 수 있는 최대값입니다.
unsigned char unsigned short unsigned int unsigned long unsigned long long	"type":"integer", "minimum":0, "maximum":n 여기서, n은 기본요소로 표현할 수 있는 최대값입니다.
bool (C++만 해당)	"type":"boolean"
float 맵핑 레벨 1.2 이상에서 지원됩니다.	"type":"number", "description":"float" 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 float 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.

C 및 C++ 데이터 유형	스키마 simpleType
double 맵핑 레벨 1.2 이상에서 지원됩니다.	<pre>"type": "number", "description": "double"</pre> 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 double 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 double 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.
type name[n]	기본요소의 경우: <pre>"type": "array" "maxItems": n "minItems": n "items": { "type": "object", "properties": { name: { type JSON } } "required": [name] }</pre> struct의 경우: <pre>"type": "array" "maxItems": n "minItems": n "items": { type JSON }</pre> 여기서, type JSON은 C 또는 C++ 유형의 JSON 스키마 표현입니다.

관련 참조:

『JSON 스키마 대 C 및 C++ 맵핑』

DFHJS2LS 유틸리티 프로그램은 JSON 스키마와 C 및 C++ 데이터 유형 간의 맵핑을 지원합니다.

JSON 스키마 대 C 및 C++ 맵핑

DFHJS2LS 유틸리티 프로그램은 JSON 스키마와 C 및 C++ 데이터 유형 간의 맵핑을 지원합니다.

CICS 지원은 다음 규칙을 사용하여 스키마 요소 이름에서 C 및 C++ 변수의 올바른 고유 필드 이름을 생성합니다.

1. A - Z, a - z, 0 - 9 또는 _ 이외의 문자는 'X'로 바꿉니다.

예를 들어, monthly-total은 monthlyXtotal이 됩니다.

2. 첫 번째 문자가 영문자가 아니면 선행 'X'로 바꿉니다.

예를 들어, `_monthlysummary`는 `Xmonthlysummary`가 됩니다.

3. 동일한 범위의 중복된 이름은 이름의 두 번째 및 후속 인스턴스에 하나 또는 두 개의 숫자를 추가해서 이름을 고유하게 합니다.

예를 들어, `year`의 세 가지 인스턴스는 `year`, `year1`, `year2`가 됩니다.

4. JSON 스키마는 `"type"` 값이 `"array"`이고 `"minItems"` 및 `"maxItems"` 키워드가 생략되거나 값이 다른 경우 변수의 변동 카디널리티가 있음을 지정합니다. 변수의 변동 카디널리티가 있다고 스키마에 지정되면 `"_cont"` 및 `"_num"` 접미부가 붙은 필드 이름이 작성됩니다.

자세한 정보는 634 페이지의 『DFHJS2LS의 변동 요소 배열』의 내용을 참조하십시오.

5. JSON 스키마는 엔클로징 JSON 스키마 `"object"` 유형과 연관된 `"required"` 키워드 배열에 표시되지 않는 경우 변수가 선택적이라고 지정합니다. 선택적 필드의 경우에는 요소 이름에 `_num` 접미부가 추가된 추가 필드가 생성됩니다. 런타임에 이는 JSON 데이터에 값이 없음을 나타내는 0 및 JSON 데이터에 값이 있으면 0 이외의 값입니다.
6. 필드 이름은 50자로 제한됩니다. 접두부 및 접미부를 포함하여 생성된 이름이 이 길이를 초과하는 경우 요소 이름이 잘립니다.

DFHJS2LS는 다음 테이블에 따라 JSON 스키마 유형 값을 C 및 C++ 데이터 유형으로 매핑합니다. 다음 규칙도 적용됩니다.

- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정되면 가변 길이 문자 데이터는 널(null) 종료 문자열로 매핑되고 널 종료자에 대한 여분의 문자가 할당됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 YES로 설정된 경우에는 가변 길이 문자 데이터가 관련된 두 개의 요소, 길이 필드 및 데이터 필드로 매핑됩니다.

JSON 스키마 키워드	C 및 C++ 데이터 유형
다음 모두: <code>"type": "null"</code> <code>"type": []</code> <code>"enum": []</code> <code>"allOf"</code> <code>"anyOf"</code> <code>"noneOf"</code> <code>"not"</code> <code>"definitions"</code>	지원되지 않음
<code>"\$schema": "http://json-schema.org/draft-04/schema#"</code>	이 키워드는 무시되지만 드래프트 04 JSON 스키마 스펙과 호환 가능할 것이라 가정됩니다.

JSON 스키마 키워드	C 및 C++ 데이터 유형
"title": "same text" "description": "more text"	이 키워드는 무시됩니다.
"format": "<predefined values>"	"format" 키워드는 생성된 구조나 런타임 값을 수정하는 데 사용됩니다. "format"의 지원되는 사용에 대해서는 다음 정보를 참조하십시오.
"type": "array", "items": {<JSON Sub-schema>}, "additionalItems": false, "maxItems": m, "minItems": n	현재 지원되는 JSON 배열의 유일한 양식은 반복된 수의 동일한 유형 값입니다. <JSON Sub-schema>는 지원되는 "type"을 정의해야 합니다. 하지만 "type"이 "array"이면 안됩니다. 이는 생성되는 언어 구조에 대한 제한사항입니다. "additionalItems"는 false로 가정되며 다른 값은 지원되지 않습니다. "minItems" 및 "maxItems"가 모두 있고 동일하면 배열이 고정 카디널리티로 처리되고 그렇지 않은 경우에는 변동 카디널리티로 처리됩니다. 634 페이지의 『DFHJS2LS의 변동 요소 배열』의 내용을 참조하십시오.
"type": "array", "uniqueItems": true	"uniqueItems"는 JSON 배열에 지원되지 않습니다. <JSON Sub-schema>는 지원되는 "type"을 정의해야 합니다. 하지만 "type"이 "array"이면 안됩니다. 이는 생성되는 언어 구조에 대한 제한사항입니다.
"type": "object", "additionalProperties": false, "properties": { ["<element name>": {<JSON Sub-schema> } [,]]* } "required": [["<element name>" [,]]*]	현재 지원되는 JSON 오브젝트의 유일한 양식은 이름 요소의 고정 세트입니다. 이는 요소 이름을 사용하여 구조(또는 하위 구조)를 생성합니다. "additionalProperties"는 false로 가정되며 다른 값은 지원되지 않습니다. "properties" 오브젝트의 요소는 "required" 배열에 있지 않거나 "required" 배열이 없으면 "optional"로 간주됩니다. "optional" 요소에는 0 - X의 가변 서수가 제공되며, 여기서 X는 1 또는 해당 항목이 배열로 정의된 배열의 최대 항목 수입니다. 634 페이지의 『DFHJS2LS의 변동 요소 배열』의 내용을 참조하십시오.
"type": "object", "maxProperties": m, "minProperties": n, "patternProperties": {}, "dependencies":	이 키워드는 JSON 오브젝트에 지원되지 않습니다.

JSON 스키마 키워드	C 및 C++ 데이터 유형
<pre>"type": "string" "maxLength": m "pattern": "regular expression", "minLength": l</pre>	char[z] 여기서, z 값은 m에 기초하지만 CHAR-VARYING 매개변수의 설정에 종속됩니다. m은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다. "pattern" 및 "minLength" 제한사항은 언어 구조에 주석으로만 전달됩니다.
<pre>"type": "string" "maxLength": m</pre>	맵핑 레벨 4.0 이상에서 CCSID=1200일 때: char16_t[z] 여기서, z 값은 m에 기초하지만 CHAR-VARYING 매개변수의 설정에 종속됩니다. m은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다.
<pre>"*name*": { "type": "string", "format": "date-time" }</pre>	char[8] DATETIME=PACKED15일 때 모두 지원됨
<pre>"*name*": { "type": "string", "format": "uri" }</pre>	char[m] 여기서, m은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다. 맵핑 레벨 4.0 이상에서 CCSID=1200일 때: char16_t[m] 여기서, m은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다.
<pre>"*name*": { "type": "string", "format": "base64Binary" }</pre>	char[m] 여기서, m은 "maxLength" 키워드를 기반으로 합니다.
<pre>"*name*": { "type": "string", "format": "hexBinary" }</pre>	char[m] 여기서, m은 "maxLength" 키워드를 기반으로 합니다.

JSON 스키마 키워드	C 및 C++ 데이터 유형
<pre>"*name*":{ "type":"string", "format":"<predefined>" }</pre>	char[m] 여기서, m은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리되고 <predefined>는 email, hostname, ipv4 또는 ipv6 중 하나입니다. 관련 "pattern"이 사용되고 주석에 전달됩니다. 맵핑 레벨 4.0 이상에서 CCSID=1200일 때: char16_t[m] 여기서, m은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리되고 <predefined>는 email, hostname, ipv4 또는 ipv6 중 하나입니다. 관련 "pattern"이 사용되고 주석에 전달됩니다.
"type": "boolean"	bool(C++만 해당) short(C만 해당)
<pre>"type": "integer", "maxExclusive": true, "minExclusive": true, "multipleOf": n</pre>	"maxExclusive" 및 "minExclusive" 제한사항은 언어 구조에 주석으로만 전달됩니다. "multipleOf"는 무시됩니다.
<pre>"type": "integer", minimum: -128, maximum: 127</pre>	signed char
<pre>"type": "integer", minimum: 0, maximum: 255</pre>	unsigned char
<pre>"type": "integer", minimum: -32768, maximum: 32767</pre>	short
<pre>"type": "integer", minimum: 0, maximum: 65535</pre>	unsigned short
<pre>"type": "integer", minimum: -2147483648, maximum: 2147483647</pre>	int
<pre>"type": "integer", minimum: 0, maximum: 4294967295</pre>	unsigned int
<pre>"type": "integer", minimum: -9223372036854775808, maximum: 9223372036854775807</pre>	long long
<pre>"type": "integer", minimum: 0, maximum: 18446744073709551615</pre>	unsigned long long
<pre>"type": "number", "maximum": m, "minimum": n, "maxExclusive": true, "minExclusive": true, "multipleOf": n</pre>	"maximum", "minimum", "maxExclusive", "minExclusive" 제한사항은 언어 구조에 주석으로만 전달됩니다. "multipleOf"는 무시됩니다.

JSON 스키마 키워드	C 및 C++ 데이터 유형
"type": "number" "format": "float"	맵핑 레벨 1.1 이하: • char[32] 맵핑 레벨 1.2 이상: • float(*) 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 float 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.
"type": "number" "format": "double"	맵핑 레벨 1.0 이하: • char[32] 맵핑 레벨 1.2 이상: • double(*) 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 double 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정밀도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 double 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.

참고: CICS는 부호 있는 long($2^{63} - 1$)의 최대값보다 큰 정수 값을 따옴표(')로 묶지 않으면 변환할 수 없습니다.

참고: 숫자 유형에 대한 스키마에 지정된 최소 및 최대값은 C 또는 C++ 데이터 유형으로 맵핑하는 용도로만 사용됩니다. 런타임에 이 값에 대해 데이터의 유효성을 검사하지 않습니다.

관련 참조:

613 페이지의 『C 및 C++ 대 JSON 스키마 맵핑』

DFHLS2JS 유틸리티 프로그램은 C 및 C++ 데이터 유형과 JSON 스키마 정의 간의 맵핑을 지원합니다.

PL/I 대 JSON 스키마 매핑

DFHLS2JS 유틸리티 프로그램은 PL/I 데이터 구조와 JSON 스키마 정의 간의 매핑을 지원합니다. Enterprise PL/I 컴파일러와 이전 PL/I 컴파일러가 다르므로 두 개의 언어 옵션인 PLI-ENTERPRISE 및 PLI-OTHER가 지원됩니다.

PL/I 이름은 다음 규칙에 따라 JSON 이름으로 변환됩니다.

1. JSON 특성 이름에 유효하지 않은 문자는 'x'로 바꿉니다.

예를 들어, `monthly$total`은 `monthlyxtotal`이 됩니다.

2. 중복 이름은 하나 이상의 숫자를 추가해서 고유 이름이 되게 합니다.

예를 들어 `year`의 두 인스턴스는 `year` 및 `year1`이 됩니다.

DFHLS2JS는 다음 테이블에 따라 PL/I 데이터 유형을 스키마 요소로 매핑합니다. 테이블에 표시되지 않은 PL/I 유형은 DFHLS2JS에서 지원되지 않습니다. 다음 제한사항도 적용됩니다.

- COMPLEX 속성이 있는 데이터 항목은 지원되지 않습니다.
- FLOAT 속성이 있는 데이터 항목은 매핑 레벨 1.2 이상에서 지원됩니다. Enterprise PL/I FLOAT IEEE는 지원되지 않습니다.
- VARYING 및 VARYINGZ 순수 DBCS 문자열은 매핑 레벨 1.2 이상에서 지원됩니다.
- DECIMAL(p,q)로 지정된 데이터 항목은 $p \geq q$ 일 때에만 지원됩니다.
- BINARY(p,q)로 지정된 데이터 항목은 $q = 0$ 일 때에만 지원됩니다.
- 데이터 항목에 PRECISION 속성이 지정된 경우 무시됩니다.
- PICTURE 문자열은 지원되지 않습니다.
- ORDINAL 데이터 항목은 FIXED BINARY(7) 데이터 유형으로 처리됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정된 경우 문자 배열은 string으로 매핑되고 널(null) 종료 문자열로 처리되며, 이 매핑은 Enterprise PL/I에 적용되지 않습니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 BINARY로 설정되면 문자 배열은 string으로 매핑되고 2진 데이터로 처리됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 COLLAPSE로 설정된 경우에는 선행 및 후미 공백이 제거되고 다수의 공백이 단일 공백으로 바뀝니다.

DFHLS2JS는 PL/I의 채우기 알고리즘을 완전히 구현하지 않으므로 데이터 구조에서 명시적으로 채우기 바이트를 선언해야 합니다. DFHLS2JS는 채우기 바이트가 누락된 사

실을 발견할 경우 메시지를 발행합니다. 각 최상위 레벨 구조는 2배 단어 경계에서 시작하고 구조의 각 바이트는 올바른 경계로 맵핑되어야 합니다. 다음 코드 단편을 고려하십시오.

```
3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);
```

이 예제에서,

- FIELD1은 1바이트 길이며 모든 경계에 맞출 수 있습니다.
- FIELD2는 4바이트 길이며 전자 경계에 맞춰야 합니다.
- FIELD3은 8바이트 길이며 2배 단어 경계에 맞춰야 합니다.

Enterprise PL/I 컴파일러는 다음과 같은 순서로 필드를 맞춥니다.

1. FIELD3은 경계 요구사항이 가장 강력하므로 먼저 맞춥니다.
2. FIELD2는 FIELD3 바로 앞의 전자 경계에 맞춥니다.
3. FIELD1은 FIELD3 바로 앞의 바이트 경계에 맞춥니다.

마지막으로 전체 구조가 전자 경계에 맞춰지도록 컴파일러는 FIELD1 바로 앞에 채우기 3바이트를 삽입합니다.

DFHLS2JS가 해당 채우기 바이트를 삽입하지 않으므로 DFHLS2JS가 구조를 처리하기 전에 직접 이를 명시적으로 선언해야 합니다. 예를 들어, 다음과 같습니다.

```
3 PAD1    FIXED BINARY(7),
3 PAD2    FIXED BINARY(7),
3 PAD3    FIXED BINARY(7),
3 FIELD1  FIXED BINARY(7),
3 FIELD2  FIXED BINARY(31),
3 FIELD3  FIXED BINARY(63);
```

또는 모든 필드를 맞추지 않음으로 선언하도록 구조를 변경하고 이 구조를 사용하는 애플리케이션을 다시 컴파일할 수도 있습니다. PL/I 구조 메모리 맞추기 요구사항에 대한 자세한 정보는 *Enterprise PL/I Language Reference*를 참조하십시오.

PL/I 데이터 설명	JSON 스키마 정의
FIXED BINARY (<i>n</i>)	<pre>"type":"integer", "minimum":- (n + 1), "maximum":n</pre> 여기서, <i>n</i>은 기본요소로 표시할 수 있는 최대값입니다.
UNSIGNED FIXED BINARY(<i>n</i>) 제한사항: Enterprise PL/I만 해당	<pre>"type":"integer", "minimum":0, "maximum":n</pre> 여기서, <i>n</i>은 기본요소로 표시할 수 있는 최대값입니다.

PL/I 데이터 설명	JSON 스키마 정의
FIXED DECIMAL(n,m)	<pre>"type":"number", "description":"decimal", "multipleOf":x, "maximum":y, "minimum":-z</pre> 여기서, x는 사용 가능한 최소 단위 = $1 / 10^m$이고, y는 n 및 m 조합으로 표현할 수 있는 최대값이며, z는 n 및 m 조합으로 표시할 수 있는 최대값입니다.
FIXED DECIMAL(15) DATETIME=PACKED 15일 때 맵핑 레벨 3.0 이상에서 지원됨	<pre>"type":"string", "format":"date-time"</pre> 시간 소인 형식은 RFC3339를 통해 정의됩니다.
BIT(n) 여기서, n 은 8의 배수이며 기타 값은 지원되지 않습니다.	<pre>"type":"string" "maxLength":m</pre> 여기서, $m = n/8$
CHARACTER(n) VARYING 및 VARYINGZ는 맵핑 레벨 1.2 이상에서도 지원됩니다. 제한사항: VARYINGZ는 Enterprise PL/I에서만 지원됩니다.	<pre>"type":"string" "maxLength":n</pre>
GRAPHIC(n) VARYING 및 VARYINGZ는 맵핑 레벨 1.2 이상에서도 지원됩니다. 제한사항: VARYINGZ는 Enterprise PL/I에서만 지원됩니다.	맵핑 레벨 1.0 및 1.1에서, 여기서 $m = 2*n$: <pre>"type":"string" "maxLength":m</pre> 맵핑 레벨 1.2 이상에서: <pre>"type":"string" "maxLength":n</pre>
WIDECHAR(n) 제한사항: Enterprise PL/I만 해당	맵핑 레벨 1.0 및 1.1에서, 여기서 $m = 2*n$: <pre>"type":"string" "maxLength":m</pre> 맵핑 레벨 1.2 이상에서: <pre>"type":"string" "maxLength":n</pre> 맵핑 레벨 4.0 이상에서, CICS는 애플리케이션 데이터 구조 필드를 UTF-16 데이터로 채웁니다. <pre><xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="n"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType></pre>
ORDINAL 제한사항: Enterprise PL/I만 해당	<pre>"type":"integer", "minimum":0, "maximum":255</pre>

PL/I 데이터 설명	JSON 스키마 정의
BINARY FLOAT(n) 여기서, $n \leq 21$ 맵핑 레벨 1.2 이상에서 지원됩니다. 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정확도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 BINARY FLOAT 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.	<pre>"type":"number", "description":"float"</pre>
BINARY FLOAT(n) 여기서, $21 < n \leq 53$ 53보다 큰 값은 지원되지 않습니다. 맵핑 레벨 1.2 이상에서 지원됩니다. 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정확도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 BINARY FLOAT 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.	<pre>"type":"number", "description":"double"</pre>
DECIMAL FLOAT(n) 여기서, $n \leq 6$ 맵핑 레벨 1.2 이상에서 지원됩니다. 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정확도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 DECIMAL FLOAT 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.	<pre>"type":"number", "description":"float"</pre>

PL/I 데이터 설명	JSON 스키마 정의
DECIMAL FLOAT(<i>n</i>) 여기서, $6 < n \leq 16$ 16보다 큰 값은 지원되지 않습니다. 맵핑 레벨 1.2 이상에서 지원됩니다. 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정확도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 DECIMAL FLOAT 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.	<pre>"type": "number", "description": "double"</pre>
<i>name (n) data description</i>	기본요소의 경우: <pre>"type": "array" "maxItems": n "minItems": n "items": { "type": "object", "properties": { name: { data description JSON } } "required": [name] }</pre> 데이터 선언의 경우: <pre>"type": "array" "maxItems": n "minItems": n "items": { data description JSON }</pre> 여기서, <i>data description JSON</i>은 PL/I 데이터 설명의 JSON 스키마 표현입니다.

관련 참조:

627 페이지의 『JSON 스키마 대 PL/I 맵핑』

DFHJS2LS 유틸리티 프로그램은 JSON 스키마와 PL/I 데이터 구조 간의 맵핑을 지원합니다. Enterprise PL/I 컴파일러와 이전 PL/I 컴파일러가 다르므로 두 개의 언어 옵션인 PLI-ENTERPRISE 및 PLI-OTHER가 지원됩니다.

JSON 스키마 대 PL/I 맵핑

DFHJS2LS 유틸리티 프로그램은 JSON 스키마와 PL/I 데이터 구조 간의 맵핑을 지원합니다. Enterprise PL/I 컴파일러와 이전 PL/I 컴파일러가 다르므로 두 개의 언어 옵션인 PLI-ENTERPRISE 및 PLI-OTHER가 지원됩니다.

CICS 지원은 다음 규칙을 사용하여 스키마 요소 이름에서 PL/I 변수의 올바른 고유 이름을 생성합니다.

1. A-Z, a-z, 0-9, @, # 또는 \$ 이외의 문자는 'X'로 바꿉니다.

예를 들어, `monthly_total`은 `monthlyXtotal`이 됩니다.

2. 동일한 범위의 중복된 이름은 이름의 두 번째 및 후속 인스턴스에 하나 또는 두 개의 숫자를 추가해서 이름을 고유하게 합니다.

예를 들어, `year`의 세 가지 인스턴스는 `year`, `year1`, `year2`가 됩니다.

3. JSON 스키마는 "type" 값이 "array"이고 "minItems" 및 "maxItems" 키워드가 생략되거나 값이 다른 경우 변수의 변동 카디널리티가 있음을 지정합니다. 변수의 변동 카디널리티가 있다고 스키마에 지정되면 "_cont" 및 "_num" 접미부가 붙은 필드 이름이 작성됩니다.

자세한 정보는 634 페이지의 『DFHJS2LS의 변동 요소 배열』의 내용을 참조하십시오.

4. JSON 스키마는 엔클로징 JSON 스키마 "object" 유형과 연관된 "required" 키워드 배열에 표시되지 않는 경우 변수가 선택적이라고 지정합니다. 선택적 필드의 경우에는 요소 이름에 _num 접미부가 추가된 추가 필드가 생성됩니다. 런타임에 이는 JSON 데이터에 값이 없음을 나타내는 0 및 JSON 데이터에 값이 있으면 0 이외의 값입니다.

5. 필드 이름은 31자로 제한됩니다. 접두부 및 접미부를 포함하여 생성된 이름이 이 길이를 초과하는 경우 요소 이름이 잘립니다.

결과적인 이름의 총 길이는 31자 이하입니다.

DFHJS2LS는 다음 테이블에 따라 스키마 유형 값을 PL/I 데이터 유형으로 맵핑합니다. 다음 사항도 유의하십시오.

- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 NULL로 설정되면 가변 길이 문자 데이터는 널(null) 종료 문자열로 맵핑되고 널 종료자에 대한 여분의 문자가 할당됩니다.
- **MAPPING-LEVEL** 매개변수가 1.2 이상으로 설정되고 **CHAR-VARYING** 매개변수가 지정되지 않으면 기본적으로 가변 길이 문자 데이터는 Enterprise PL/I의 경우 **VARYINGZ** 데이터 유형으로 그리고 기타 PL/I의 경우에는 **VARYING** 데이터 유형으로 맵핑됩니다.

- 가변 길이 2진 데이터는 32 768바이트 미만인 경우 VARYING 데이터 유형으로 그리고 32 768바이트를 초과하는 경우에는 컨테이너로 맵핑됩니다.

JSON 스키마 키워드	PL/I 데이터 설명
다음 모두: "type": "null" "type": [] "enum": [] "allOf" "anyOf" "noneOf" "not" "definitions"	지원되지 않음
"\$schema": "http://json-schema.org/draft-04/schema#"	이 키워드는 무시되지만 드래프트 04 JSON 스키마 스펙과 호환 가능할 것이라 가정됩니다.
"title": "same text" "description": "more text"	이 키워드는 무시됩니다.
"format": "<predefined values>"	"format" 키워드는 생성된 구조나 런타임 값을 수정하는 데 사용됩니다. "format"의 지원되는 사용에 대해서는 다음 정보를 참조하십시오.
"type": "array", "items": {<JSON Sub-schema>}, "additionalItems": false, "maxItems": m, "minItems": n	현재 지원되는 JSON 배열의 유일한 양식은 반복된 수의 동일한 유형 값입니다. <JSON Sub-schema>는 지원되는 "type"을 정의해야 합니다. 하지만 "type"이 "array"이면 안됩니다. 이는 생성되는 언어 구조에 대한 제한사항입니다. "additionalItems"는 false로 지정되며 다른 값은 지원되지 않습니다. "minItems" 및 "maxItems"가 모두 있고 동일하면 배열이 고정 카디널리티로 처리되고 그렇지 않은 경우에는 변동 카디널리티로 처리됩니다. 634 페이지의 『DFHJS2LS의 변동 요소 배열』의 내용을 참조하십시오.
"type": "array", "uniqueItems": true	"uniqueItems"는 JSON 배열에 지원되지 않습니다. <JSON Sub-schema>는 지원되는 "type"을 정의해야 합니다. 하지만 "type"이 "array"이면 안됩니다. 이는 생성되는 언어 구조에 대한 제한사항입니다.

JSON 스키마 키워드	PL/I 데이터 설명
<pre>"type": "object", "additionalProperties": false, "properties": { ["<element name>": {<JSON Sub-schema> } [,]]* } "required": [["<element name>" [,]]*]</pre>	현재 지원되는 JSON 오브젝트의 유일한 양식은 이름 요소의 고정 세트입니다. 이는 요소 이름을 사용하여 구조(또는 하위 구조)를 생성합니다. "additionalProperties"는 false로 가정되며 다른 값은 지원되지 않습니다. "properties" 오브젝트의 요소는 "required" 배열에 있지 않거나 "required" 배열이 없으면 "optional"로 간주됩니다. "optional" 요소에는 0 - X의 가변 서수가 제공되며, 여기서 X는 1 또는 해당 항목이 배열로 정의된 배열의 최대 항목 수입니다. 634 페이지의 『DFHJS2LS의 변동 요소 배열』의 내용을 참조하십시오.
<pre>"type": "object", "maxProperties": m, "minProperties": n, "patternProperties": {}, "dependencies":</pre>	이 키워드는 JSON 오브젝트에 지원되지 않습니다.
<pre>"type": "string" "maxLength": m "pattern": "regular expression", "minLength": l</pre>	char[z] 여기서, z 값은 m에 기초하지만 CHAR-VARYING 매개변수의 설정에 종속됩니다. m은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다. "pattern" 및 "minLength" 제한사항은 언어 구조에 주석으로만 전달됩니다.
<pre>"type": "string" "maxLength": m</pre>	맵핑 레벨 4.0 이상에서 CCSID=1200일 때: WIDECHAR(z) 여기서, z 값은 m에 기초하지만 CHAR-VARYING 매개변수의 설정에 종속됩니다. m은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다.
<pre>"*name*": { "type": "string", "format": "date-time" }</pre>	FIXED DECIMAL (15,0) DATETIME=PACKED15일 때 모두 지원됨

JSON 스키마 키워드	PL/I 데이터 설명
<pre>"*name*":{ "type":"string", "format":"uri" }</pre>	CHAR(<i>m</i>) 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다. 맵핑 레벨 4.0 이상에서 CCSID=1200일 때: WIDECHAR(<i>m</i>) 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리됩니다.
<pre>"*name*":{ "type":"string", "format":"base64Binary" }</pre>	CHAR(<i>m</i>) 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 합니다.
<pre>"*name*":{ "type":"string", "format":"hexBinary" }</pre>	CHAR(<i>m</i>) 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 합니다.
<pre>"*name*":{ "type":"string", "format":"<predefined>" }</pre>	CHAR(<i>m</i>) 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리되고 <predefined>는 <i>email</i>, <i>hostname</i>, <i>ipv4</i> 또는 <i>ipv6</i> 중 하나입니다. 관련 "pattern"이 주석에 전달됩니다. 맵핑 레벨 4.0 이상에서 CCSID=1200일 때: WIDECHAR(<i>m</i>) 여기서, <i>m</i>은 "maxLength" 키워드를 기반으로 하며 고정 길이 문자열로 처리되고 <predefined>는 <i>email</i>, <i>hostname</i>, <i>ipv4</i> 또는 <i>ipv6</i> 중 하나입니다. 관련 "pattern"이 사용되고 주석에 전달됩니다.

JSON 스키마 키워드	PL/I 데이터 설명
"type": "boolean"	맵핑 레벨 1.1 이하: Enterprise PL/I SIGNED FIXED BINARY (7) 기타 PL/I FIXED BINARY (7) 맵핑 레벨 1.2 이상: Enterprise PL/I BIT(7) BIT(1) 기타 PL/I BIT(7) BIT(1) 여기서, BIT(7)은 맞추기를 위해 제공되고 BIT(1)은 부울 맵핑 값을 포함합니다.
"type": "integer", "maxExclusive": true, "minExclusive": true, "multipleOf": n	"maxExclusive" 및 "minExclusive" 제한사항은 언어 구조에 주석으로만 전달됩니다. "multipleOf"는 무시됩니다.
"type": "integer", minimum: -128, maximum: 127	Enterprise PL/I SIGNED FIXED BINARY (7) 기타 PL/I FIXED BINARY (7)
"type": "integer", minimum: 0, maximum: 255	Enterprise PL/I UNSIGNED FIXED BINARY (8) 기타 PL/I FIXED BINARY (8)
"type": "integer", minimum: -32768, maximum: 32767	Enterprise PL/I SIGNED FIXED BINARY (15) 기타 PL/I FIXED BINARY (15)
"type": "integer", minimum: 0, maximum: 65535	Enterprise PL/I UNSIGNED FIXED BINARY (16) 기타 PL/I FIXED BINARY (16)

JSON 스키마 키워드	PL/I 데이터 설명
"type": "integer", minimum: -2147483648, maximum: 2147483647	Enterprise PL/I SIGNED FIXED BINARY (31) 기타 PL/I FIXED BINARY (31)
"type": "integer", minimum: 0, maximum: 4294967295	맵핑 레벨 1.1 이하: Enterprise PL/I UNSIGNED FIXED BINARY(32) 맵핑 레벨 1.2 이상: Enterprise PL/I CHAR(y) 여기서, y는 16MB 미만의 고정 길이 입니다. 모든 맵핑 레벨: 기타 PL/I BIT(64)
"type": "integer", minimum: -9223372036854775808, maximum: 9223372036854775807	맵핑 레벨 1.1 이하: Enterprise PL/I SIGNED FIXED BINARY(63) 맵핑 레벨 1.2 이상: Enterprise PL/I CHAR(y) 여기서, y는 16MB 미만의 고정 길이 입니다. 모든 맵핑 레벨: 기타 PL/I BIT(64)
"type": "integer", minimum: 0, maximum: 18446744073709551615	맵핑 레벨 1.1 이하: Enterprise PL/I UNSIGNED FIXED BINARY(64) 맵핑 레벨 1.2 이상: Enterprise PL/I CHAR(y) 여기서, y는 16MB 미만의 고정 길이 입니다. 기타 PL/I BIT(64)

JSON 스키마 키워드	PL/I 데이터 설명
"type": "number" "description": "decimal"	FIXED DECIMAL(<i>n,m</i>)
"type": "number" "description": "float"	맵핑 레벨 1.0 및 1.1: CHAR(32) 맵핑 레벨 1.2 이상: Enterprise PL/I DECIMAL FLOAT(6) HEXADEC 기타 PL/I DECIMAL FLOAT(6) 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정확도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 DECIMAL FLOAT 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.
"type": "number" "description": "double"	맵핑 레벨 1.0 및 1.1: CHAR(32) 맵핑 레벨 1.2 이상: Enterprise PL/I DECIMAL FLOAT(16) HEXADEC 기타 PL/I DECIMAL FLOAT(16) 참고: IBM 16진 부동 소수점(HFP) 데이터 표현은 JSON에 사용된 IEEE-754-1985 표현과 정확히 일치하지 않습니다. 일부 값이 한 표현에서 다른 표현으로 정확히 변환되지 않을 수도 있습니다. 극도로 크거나 작은 일부 값은 float 데이터 유형에 유효하지 않을 수 있습니다. 몇 가지 값은 HFP 표현으로(부터) 변환할 때 정확도가 손실될 수 있습니다. 변환의 정밀도가 중요한 경우에는 DECIMAL FLOAT 데이터 유형을 고정 정밀도 대안으로 바꿀 것을 고려하십시오.

참고: CICS는 부호 있는 long($2^{63} - 1$)의 최소값보다 큰 정수 값을 따옴표(')로 묶지 않으면 변환할 수 없습니다.

참고: 숫자 유형에 대한 스키마에 지정된 최소 및 최대값은 PL/I 데이터 유형으로 맵핑하는 용도로만 사용됩니다. 런타임에 이 값에 대해 데이터의 유효성을 검사하지 않습니다.

관련 참조:

622 페이지의 『PL/I 대 JSON 스키마 맵핑』

DFHLS2JS 유틸리티 프로그램은 PL/I 데이터 구조와 JSON 스키마 정의 간의 맵핑을 지원합니다. Enterprise PL/I 컴파일러와 이전 PL/I 컴파일러가 다르므로 두 개의 언어 옵션인 PLI-ENTERPRISE 및 PLI-OTHER가 제공됩니다.

DFHJS2LS의 변동 요소 배열

JSON은 가변적인 수의 요소 배열을 포함할 수 있습니다. 일반적으로 가변적인 수의 요소가 있는 JSON 스키마는 단일 고급 언어 데이터 구조로 효과적으로 맵핑되지 않습니다. CICS는 컨테이너 기반 맵핑이나 인라인 맵핑을 사용하여 JSON 데이터의 가변적인 수의 요소를 처리합니다.

가변적인 수의 요소가 있는 배열은 "type" 값이 "array"인 스키마에서 minItems 및 maxItems 키워드를 사용하여 JSON 스키마에 표현됩니다.

- minItems 키워드는 요소가 발생할 수 있는 최소 횟수를 지정합니다. 가능한 값은 0 또는 양의 정수입니다. 기본값은 0입니다.
- maxItems 키워드는 요소가 발생할 수 있는 최대 횟수를 지정합니다. 가능한 값은 minItems 값 이상의 양의 정수입니다.
- maxItems 키워드가 누락된 경우 이는 배열이 무제한임을 의미합니다.

선택적 필드는 변동 배열 "maxItems":1로 표시할 수 있습니다. 예를 들어, "component"라는 선택적 8바이트 문자열이 있습니다.

```
"properties":{
  "component": {
    "type":"array",
    "maxItems":1,
    "items": {
      "type": "string",
      "maxLength": 8
    }
  }
},
"required": ["component"]
```

"required" 키워드 값에 필드 이름을 포함하지 않으면 동일한 효과를 생성할 수 있습니다.

```
"properties":{
  "component": {
    "type": "string",
    "maxLength": 8
  }
}
```


일반적으로 가변적인 수의 요소가 있는 JSON 스키마는 단일 고급 언어 데이터 구조로 효과적으로 맵핑되지 않습니다. 이러한 경우를 처리하기 위해 CICS는 일련의 컨테이너에 있는 애플리케이션으로 전달되는 일련의 연결된 데이터 구조를 사용합니다. 이 구조는 애플리케이션의 입력 및 출력으로 사용됩니다.

- CICS는 JSON 데이터를 애플리케이션 데이터로 변환할 때 애플리케이션 데이터로 이 구조를 채우며 애플리케이션은 이 데이터를 읽습니다.
- CICS는 애플리케이션 데이터를 JSON 데이터로 변환할 때 구조에서 애플리케이션이 채운 애플리케이션 데이터를 읽습니다.

다음 예제는 이 데이터 구조의 형식을 보여줍니다. 다음 예제는 단순 8바이트 필드의 배열을 사용합니다. 하지만 모델은 복합 데이터 유형 배열 및 기타 배열을 포함한 데이터 유형 배열을 지원합니다.

예제 1. 고정된 수의 요소

이 예제는 정확히 세 번 발생하는 요소를 보여줍니다.

```
"properties":{
  "component": {
    "type": "array",
    "maxItems": 3,
    "minItems": 3,
    "items": {
      "type": "string",
      "maxLength": 8
    }
  }
},
"required": ["component"]
```

이 예제에서는, 요소가 발생하는 횟수가 미리 알려져 있어서 단순 COBOL 선언에서 고정 길이 배열로 또는 다른 언어의 동등한 표시로 이를 표현할 수 있습니다.

```
05 component PIC X(8) OCCURS 3 TIMES
```

예제 2. 맵핑 레벨 2 이하에서 가변적인 수의 요소

이 예제는 1 - 5번 발생할 수 있는 필수 요소를 보여줍니다.

```
"properties":{
  "component": {
    "type": "array",
    "maxItems": 5,
    "minItems": 1,
    "items": {
      "type": "string",
      "maxLength": 8
    }
  }
},
"required": ["component"]
```

기본 데이터 구조는 두 필드의 선언을 포함합니다. CICS가 JSON 데이터를 2진 데이터로 변환할 때 첫 번째 필드 `component-num`은 JSON 데이터에 요소가 표시되는 횟수를 포함하고, 두 번째 필드 `component-cont`는 컨테이너의 이름을 포함합니다.

```
05 component-num PIC S9(9) COMP-5
```

```
05 component-cont PIC X(16)
```

두 번째 데이터 구조는 요소 선언 자체를 포함합니다.

```
01 DFHJS-component
```

```
02 component PIC X(8)
```

`component-num`의 값(1 - 5 범위의 값을 포함할)을 검토해서 요소가 발생하는 횟수를 파악해야 합니다. 요소 내용은 `component-cont`의 이름 지정된 컨테이너에 있고, 컨테이너는 요소의 배열을 보유합니다. 각 요소는 DFHJS-component 데이터 구조를 통해 매핑됩니다.

`minItems="0"`이거나 누락된 경우 및 `maxItems="1"`인 경우 요소가 선택적입니다. 애플리케이션에서 데이터 구조를 처리하려면 `component-num`의 값을 검토해야 합니다.

- 이 값이 0인 경우 메시지에 구성요소 요소가 없고 `component-cont`의 내용이 정의되지 않습니다.
- 1인 경우에는 구성요소 요소가 `component-cont`의 이름 지정된 컨테이너에 있습니다.

컨테이너의 내용은 DFHJS-component 데이터 구조를 통해 매핑됩니다.

참고: JSON 데이터가 단일 반복 요소로 이루어진 경우 DFHJS2LS는 두 가지 언어 구조를 생성합니다. 기본 언어 구조는 배열의 요소 수와 요소 배열을 보유한 컨테이너의 이름을 포함합니다. 두 번째 언어 구조는 반복 요소의 단일 인스턴스를 매핑합니다.

예제 3. 매핑 레벨 2.1 이상에서 가변적인 수의 요소

매핑 레벨 2.1 이상에서는, CICS 지원의 **INLINE-MAXOCCURS-LIMIT** 매개변수를 사용할 수 있습니다. **INLINE-MAXOCCURS-LIMIT** 매개변수는 가변적인 수의 요소가 처리되는 방식을 지정합니다. 가변적인 수의 요소에 대한 매핑 옵션은 635 페이지의 『예제 2. 매핑 레벨 2 이하에서 가변적인 수의 요소』에 설명된 컨테이너 기반 매핑 또는 인라인 매핑입니다. 이 매개변수의 가능한 값은 0 - 32767 범위의 양의 정수입니다.

- **INLINE-MAXOCCURS-LIMIT**의 기본값은 선택적 요소를 인라인 매핑하는 1입니다.
- **INLINE-MAXOCCURS-LIMIT** 매개변수의 0 값은 인라인 매핑을 차단합니다.
- `maxItems`가 **INLINE-MAXOCCURS-LIMIT**의 값 이하인 경우에는 인라인 매핑이 사용됩니다.
- `maxItems`가 **INLINE-MAXOCCURS-LIMIT**의 값보다 크면 컨테이너 기반 매핑이 사용됩니다.

가변적인 수의 요소를 인라인 맵핑하면 위의 고정 발생 예제 및 카운터의 경우처럼, 두 배열이 모두 생성됩니다. `component-num` 필드는 존재하는 요소의 인스턴스 수를 표시 하며 이는 배열을 통해 지정됩니다. 635 페이지의 『예제 2. 맵핑 레벨 2 이하에서 가 변적인 수의 요소』에 표시된 예제의 경우 **INLINE-MAXOCCURS-LIMIT**가 5 이하이면 생 성된 데이터 구조는 다음과 같습니다.

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

첫 번째 필드, `component-num`은 이전 섹션의 컨테이너 기반 맵핑 예제의 출력과 동일 합니다. 두 번째 필드는 생성 가능한 최대 수의 요소를 충분히 포함할 수 있는 길이 5 의 배열을 포함합니다.

인라인 맵핑은 모든 데이터를 현재 컨테이너에 저장하기 때문에, 요소의 발생 수와 데 이터가 위치한 컨테이너의 이름을 저장하는 컨테이너 기반 맵핑과 차이가 있습니다. 데 이터를 현재 컨테이너에 저장하면 일반적으로 성능이 개선되며 인라인 맵핑이 더 바람 작하게 됩니다.

예제 4. 중첩 변동 배열

복합 JSON 스키마는 가변적으로 반복되는 요소를 포함할 수 있으며, 이 요소는 차례 로 가변적으로 반복되는 요소를 포함합니다. 이 경우 설명된 구조는 예제에 설명된 두 레벨을 넘어서 확장됩니다.

다음 예제는 "component1"이라는 필수 요소에 중첩된 "component2"라는 선택적 요 소를 보여주며, 여기서 필수 요소는 1 - 5회 발생할 수 있습니다.

```
"properties":{
  "component1": {
    "type": "array",
    "maxItems": 5,
    "minItems": 1,
    "items": {
      "type": "object",
      "properties":{
        "component2":{
          "type": "string",
          "maxLength": 8
        }
      },
      "required": ["component2"]
    },
    "required": ["component1"]
  },
  "required": ["component1"]
}
```

최상위 레벨 데이터 구조는 이전 예제와 정확히 동일합니다.

```
05 component1-num PIC S9(9) COMP-5
05 component1-cont PIC X(16)
```

하지만 두 번째 데이터 구조는 다음 요소를 포함합니다.

```

01 DFHJS-component1
  02 component2-num PIC S9(9) COMP-5
  02 component2-cont PIC X(16)

```

세 번째 레벨 구조는 다음 요소를 포함합니다.

```

01 DFHJS-component2
  02 component2 PIC X(8)

```

가장 바깥쪽 요소 "component1"의 발생 수는 component1-num에 있습니다.

component1-cont의 이름 지정된 컨테이너는 두 번째 데이터 구조 DFHJS-component1의 인스턴스 수가 있는 배열을 포함합니다.

component2-cont의 각 인스턴스는 상이한 컨테이너의 이름을 지정하며, 각각 세 번째 레벨 구조 DFHJS-component2를 통해 맵핑되는 데이터 구조를 포함합니다.

이 구조를 설명하려면 예제에 일치하는 JSON의 단편을 고려하십시오.

```

{"component1":
  [
    {
      "component2": "string1"
    },
    {
      "component2": "string2"
    },
  ]
}

```

"component1"은 세 번 발생합니다. 처음 두 개는 "component2"의 인스턴스를 포함하지만 세 번째 인스턴스는 그렇지 않습니다.

최상위 레벨 데이터 구조에서 component1-num은 값 3을 포함합니다. component1-cont의 이름 지정된 컨테이너에는 DFHJS-component1의 세 가지 인스턴스가 있습니다.

1. 첫 번째에서, component2-num은 값이 1이고 component2-cont의 이름 지정된 컨테이너가 *string1*을 보유합니다.
2. 두 번째에서, component2-num은 값이 1이고 component2-cont의 이름 지정된 컨테이너가 *string2*를 보유합니다.
3. 세 번째에서, component2-num은 값이 0이고 component2-cont의 내용이 정의되지 않습니다.

이 인스턴스에서 완전한 데이터 구조는 모두 네 개의 컨테이너로 표현됩니다.

- 컨테이너 DFHJS-DATA의 루트 데이터 구조
- component1-cont의 이름 지정된 컨테이너
- component2-cont의 처음 두 인스턴스에 이름 지정된 두 개의 컨테이너

선택적 구조 및 *required* 키워드

데이터 구조는 JSON 스키마 "type", "object"로 정의됩니다. 스키마는 "properties" 키워드가 제공하는 오브젝트를 사용하여 파일 이름을 개별 유형에 관련시킵니다. 이 필드가 JSON 스키마에 설명된 JSON 데이터의 일부여야 한다는 요구사항은 "required" 키워드가 제공하는 배열을 통해 제어됩니다. 이 배열은 JSON 데이터에 존재해야 하는 모든 필드 이름을 나열합니다. 따라서 선택적 필드는 배열에 부재하는 것으로 표현하거나 비어 있는 배열이 허용되지 않으면 "required" 키워드의 부재로 표현합니다. 이 경우에는 모든 필드가 선택적입니다.

선택적 필드는 0 또는 1 요소의 변동 배열로 처리됩니다. 이는 요소 이름에 "-num" 접미부가 추가된 추가 필드를 더합니다. 총 길이가 28자를 초과할 경우에는 요소 이름이 잘립니다. 런타임에 이는 JSON 데이터에 값이 존재함을 나타내는 0 이외의 값과 그렇지 않음을 나타내는 0 값입니다.

다음 예제는 "required-structure"라는 필수 필드와 "optional-structure"라는 다른 선택적 필드의 두 가지 필드를 보여줍니다.

```
{
  "type": "object",
  "properties": {
    "required-structure": {
      "type": "string",
      "maxLength": 8
    },
    "optional-structure": {
      "type": "string",
      "maxLength": 8
    }
  },
  "required": [
    "required-structure"
  ]
}
```

생성된 Cobol 구조는 두 필드를 모두 보여주지만 두 번째 필드 앞에는 요소의 정수 개수인 "optional-structure-num"이 있습니다. 0은 하나도 없음을 표시하고 1은 요소가 존재함을 나타냅니다. "optional-structure"에 올바른 데이터가 포함되었는지 여부를 표시하는 값이 설정됩니다.

```
03 OutputData.
    06 required-structure          PIC X(8).
    06 optional-structure-num      PIC S9(9) COMP-5 SYNC.
    06 optional-structure          PIC X(8).
```

애플리케이션 데이터의 UTF-16에 대한 지원

CICS 웹 서비스는 UTF-16 인코딩 애플리케이션 데이터를 XML 또는 JSON으로 그리고 XML 또는 JSON을 UTF-16 인코딩 애플리케이션 데이터로 변환하도록 지원합니다. 여러 언어로 데이터를 저장하고 처리해야 하는 경우 UTF-16을 사용하십시오.

CICS SOAP 및 JSON 웹 서비스는 UTF-16 인코딩 애플리케이션 데이터를 XML 또는 JSON으로 그리고 XML 또는 JSON을 UTF-16 인코딩 애플리케이션 데이터로 변환하도록 지원합니다. 유니코드는 시스템이 데이터를 효율적으로 처리할 수 있는 가변 너비의 인코딩 체계입니다.

UTF-16은 각 문자가 2 또는 4바이트로 표현되는 유니코드의 가변 너비 인코딩입니다. CICS 웹 서비스는 IBM Private Use Area의 UTF-16 BE(Big Endian)인 CCSID 1200 데이터를 지원합니다. 이 작동은 모든 지원 언어의 UTF-16 지원과 일치합니다.

UTF-16은 맵핑 레벨 4.0 이상에서 지원됩니다. 지원의 맵핑 설정을 사용하여 애플리케이션 데이터의 변환 방식을 사용자 정의할 수 있습니다. XML 맵핑 레벨에 대한 자세한 정보는 애플리케이션 개발에서 CICS 지원자에 대한 맵핑 레벨의 내용을 참조하십시오. JSON 맵핑 레벨에 대한 자세한 정보는 애플리케이션 개발에서 CICS JSON 지원자에 대한 맵핑 레벨의 내용을 참조하십시오.

참고: UTF-16은 처리 시간이 더 길고 저장영역은 덜 필요해서 EBCDIC 인코딩보다 효율적입니다. 또한 인코딩 유형을 혼합하면 여분의 런타임 처리가 발생합니다.

XML 또는 JSON 스키마에서 언어 구조로의 UTF-16 맵핑

UTF-16에 대한 지원은 웹 서비스 작성 방식에 따라 다릅니다. XML 또는 JSON 스키마의 언어 구조로의 맵핑은 하향식 맵핑이라고도 하며 다음 특징이 있습니다. UTF-16을 사용할 경우 모든 텍스트 필드가 UTF-16 필드로 맵핑되는 반면, COBOL의 숫자 표시 데이터 유형은 EBCDIC으로 맵핑됩니다. UTF-16을 사용하려면 DFHJS2LS, DFHSC2LS 또는 DFHWS2LS의 CCSID 매개변수를 1200으로 설정하십시오.

예를 들어, 다음 XML 스키마 단편이 WSDL에 있으면:

```
<xsd:element name="myString" nillable="false">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="20"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

DFHWS2LS 지원은 COBOL 언어 구조에 다음 필드를 생성합니다.

```
myString PIC N(20) USAGE NATIONAL
```

웹 서비스 지원의 CHAR-MULTIPLIER 매개변수를 사용하여 지원이 생성하는 필드의 길이를 지정할 수 있습니다.

CHAR-MULTIPLIER

UTF-16을 사용할 때 **CHAR-MULTIPLIER** 매개변수의 유일하게 올바른 값은 2 또는 4이며, 2는 기본값입니다.

CHAR-MULTIPLIER=2, 스키마가 maxlength x 문자열을 설명하고 PIC N(x)를 생성합니다. **CHAR-MULTIPLIER=2**를 설정해도 UTF-16 문자열에 대리 쌍을 사용할 수 있지만 필드에 맞는 문자 수에는 영향이 미칩니다.

CHAR-MULTIPLIER=4는 PIC N(2x)를 생성합니다. **CHAR-MULTIPLIER=4**인 경우 단일 인코딩 단위로 표현할 수 없는 문자가 문자열에 포함되면 런타임의 값이 채워집니다.

언어 구조에서 XML 또는 JSON 스키마로의 UTF-16 맵핑

언어 구조에서 XML 또는 JSON 스키마로의 맵핑은 상향식 맵핑이라고도 하며 하향식 맵핑과 다르게 관리됩니다. UTF-16 문자열이 언어 구조에 선언되면 CICS는 데이터를 UTF-16 인코딩으로 해석하고 그렇지 않은 경우에는 데이터가 EBCDIC 인코딩이라 가정합니다. DFHLS2JS, DFHLS2SC 또는 DFHLS2WS의 CCSID 매개변수는 애플리케이션 데이터 내에 EBCDIC 텍스트의 인코딩을 표시하며, 이를 UTF-16을 표시하도록 설정해선 안됩니다.

UTF-16 문자로 해석되는 데이터 유형은 다음과 같습니다. COBOL의 PIC N(n), PL/I의 WIDECHAR(n), C 및 C++의 char16_t[n].

웹 서비스 지원의 CHAR-USAGE 매개변수를 사용하여 데이터 유형을 지정할 수 있습니다.

CHAR-USAGE

COBOL에서는, 국가 데이터 유형, PIC N을 UTF-16 또는 DBCS 데이터에 사용할 수 있습니다. 이 설정은 NSYMBOL 컴파일러 옵션으로 제어합니다. 지원의 **CHAR-USAGE** 매개변수를 NSYMBOL 컴파일러 옵션과 동일한 값으로 설정해서 데이터가 적합하게 처리되도록 해야 합니다. UTF-16을 사용하는 경우 이는 일반적으로 CHAR-USAGE=NATIONAL로 설정됩니다.

UTF-16 및 DBCS 데이터를 동일한 카피북에 포함한 국가 데이터 유형을 혼합하려는 경우에는 개별 필드에서 USAGE NATIONAL 또는 USAGE DISPLAY-1 규정자를 사용할 수 있습니다.

참고: DFHLS2WS, DFHLS2SC, DFHLS2JS는 COBOL GROUP USAGE NATIONAL 절을 지원하지 않습니다.

언어 구조에서 맵핑 생성

애플리케이션 데이터에서 JSON(JavaScript Object Notation)을 작성하거나 JSON에서 애플리케이션 데이터를 작성하기 위해서는 CICS가 런타임에 데이터와 JSON을 변환할 방식을 설명하는 맵핑을 작성합니다. 애플리케이션 데이터 레코드에서 시작할 수 있습니다. 예를 들어, COMMAREA, VSAM 파일, 임시 저장영역 큐 또는 IBM DB2 레코드로 시작할 수 있습니다.

시작하기 전에

맵핑을 작성하기 전에 다음 전제조건이 충족되는지 확인해야 합니다.

- 파티션된 데이터 세트에서 애플리케이션 레코드를 설명하는 언어 구조가 있어야 합니다. 언어 구조는 CICS JSON 지원에서 지원되는 어느 고급 언어(COBOL, PL/I, C, C++)로도 쓸 수 있습니다.
- z/OS UNIX를 사용하기 위해 DFHLS2JS를 실행하는 사용자 ID를 구성해야 합니다.
- 사용자 ID에 언어 구조에 액세스할 읽기 권한과 z/OS UNIX의 해당 디렉토리에 출력을 둘 수 있는 쓰기 권한이 있어야 합니다.
- ID가 Java를 실행할 수 있도록 사용자 ID에 충분한 저장영역을 할당해야 합니다. 지원되는 Java 버전을 사용할 수 있습니다.

이 태스크 정보

CICS JSON 지원을 사용하여 애플리케이션 레코드에 대한 데이터 맵핑을 작성하십시오. CICS JSON 지원은 CICS 번들을 작성하고 언어 구조에서 식별한 지원되지 않는 항목에 대한 오류 메시지를 발행합니다. CICS JSON 지원의 참조 정보에 각 고급 언어에 적용되는 제한사항이 나열되어 있습니다.

프로시저

DFHLS2JS 일괄처리 작업을 실행하십시오. DFHLS2JS에는 특정 코드 페이지 선택처럼 요구사항을 충족시키기 위해 선택하는 선택적 매개변수가 있습니다. 최소한 다음 매개변수를 사용하십시오.

- **LANG** 매개변수에 언어 구조의 고급 언어를 지정하십시오.
- **BUNDLE** 매개변수에 번들 자원의 이름과 위치를 지정하십시오.
- **MAPPING-LEVEL** 매개변수에 맵핑 레벨을 지정하십시오. 어느 맵핑 레벨이나 사용할 수 있지만 고급 맵핑 옵션을 얻으려면 최신 맵핑 레벨을 사용하십시오.
- **PDSMEM** 및 **PDSCP** 매개변수에 애플리케이션 레코드를 설명하는 언어 구조의 코드 페이지 및 위치를 지정하십시오.
- **JSON-SCHEMA** 매개변수에 .json JSON 스키마 파일의 이름과 위치를 지정하십시오. 파일이 존재하지 않을 경우 DFHLS2JS는 디렉토리 구조가 아닌 JSON 스키마를 작성합니다.
- **JSONTRANSFRM** 번들 자원에 사용되는 이름을 지정하십시오. 이 이름은 애플리케이션이 JSON 맵핑을 식별하는 용도로 사용됩니다.

일괄처리 작업은 z/OS UNIX에 번들 디렉토리 구조를 작성합니다. 번들 디렉토리는 번들 Manifest를 포함한 META-INF 서브디렉토리가 있습니다. 일괄처리 작업은

JSONTRANSFRM 및 **JSON-SCHEMA** 매개변수에 지정한 파일 이름을 사용하여 번들에 JSON 스키마와 JSON 바인딩을 작성하기도 합니다.

이 JSON 바인딩을 지정하는 **BUNDLE** 자원을 설치하십시오. **JSONTRANSFRM** 번들 자원은 JSON 스키마 및 바인딩 파일의 위치를 정의하는 **JSONTRANSFRM** 자원을 동적으로 작성합니다. CICS 탐색기 번들 및 번들 파트 운영 뷰를 사용하여 설치된 **BUNDLE** 자원의 상태를 확인할 수 있습니다.

결과

JSON 변환을 하나 포함한 CICS 번들이 생성됩니다.

다음 예제는 최소한의 매개변수 세트가 지정된 **DFHLS2JS**를 보여줍니다.

```
//LS2JS JOB 'accounting information',name,MSGCLASS=A
// SET QT=''
//JCLLIB JCLLIB ORDER=FPHLQ.SDFHMOBI
//JAVAPROG EXEC DFHLS2JS,
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/jsbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
JSONTRANSFRM=example.jsbind
JSON-SCHEMA=/u/exampleapp/example.json
/*
```

다음에 수행할 작업

애플리케이션 데이터를 JSON으로(그리고 반대로) 변환시키는 애플리케이션을 쓰십시오. 두 변환에 모두 동일한 맵핑을 사용할 수 있습니다.

관련 참조:

579 페이지의 『**DFHLS2JS: 링크 가능한 인터페이스의 고급 언어 대 JSON 스키마 변환**』

DFHLS2JS 카탈로그된 프로시저는 고급 언어 구조로부터 JSON 스키마와 JSON 바인딩 파일을 생성합니다. JSON을 구문 분석하거나 작성할 수 있는 CICS 프로그램을 작성하려 할 때 **DFHLS2JS**를 사용하십시오. 이 프로시저의 사용을 돕기 위해 작업 제어문, 기호 매개변수, **DFHLS2JS**의 입력 매개변수, **DFHLS2JS**를 사용하는 예제 작업이 제공됩니다.

관련 정보:

 **DFHLS2JS: 구성에서 요청-응답 서비스를 위한 JSON 스키마 변환에 대한 고급 언어**

JSON 스키마에서 맵핑 생성

JSON(JavaScript Object Notation)에서 애플리케이션 데이터를 작성하거나 애플리케이션 데이터에서 JSON을 작성하기 위해서는 CICS가 런타임에 데이터와 JSON을 변환할 방식을 설명하는 맵핑을 작성합니다. JSON 스키마로부터 시작할 수 있습니다. 언어 구조와 맵핑이 생성되고 나면 언어 구조를 사용하여 CICS 애플리케이션을 개발하고 JSON을 애플리케이션으로(그리고 반대로) 변환할 수 있습니다.

시작하기 전에

맵핑을 작성하기 전에 다음 전제조건이 충족되는지 확인해야 합니다.

- JSON 레코드를 설명하는 JSON 스키마가 있어야 합니다.
- z/OS UNIX를 사용하기 위해 DFHJS2LS를 실행하는 사용자 ID를 구성해야 합니다.
- 사용자 ID에 JSON 스키마에 액세스할 읽기 권한과 z/OS UNIX의 해당 디렉토리에 출력을 둘 수 있는 쓰기 권한이 있어야 합니다.
- ID가 Java를 실행할 수 있도록 사용자 ID에 충분한 저장영역을 할당해야 합니다. 지원되는 Java 버전을 사용할 수 있습니다.

이 태스크 정보

CICS JSON 지원을 사용하여 애플리케이션 레코드에 대한 데이터 맵핑을 작성하십시오. CICS JSON 지원은 CICS 번들을 작성하고 언어 구조에서 식별한 지원되지 않는 항목에 대한 오류 메시지를 발행합니다. CICS JSON 지원의 참조 정보에 각 고급 언어에 적용되는 제한사항이 나열되어 있습니다. 자세한 정보는 586 페이지의 『DFHJS2LS: 링크 가능한 인터페이스의 JSON 스키마 대 고급 언어 변환』의 내용을 참조하십시오.

프로시저

DFHJS2LS 일괄처리 작업을 실행하십시오. DFHJS2LS에는 특정 코드 페이지 선택처럼 요구사항을 충족시키기 위해 선택하는 선택적 매개변수가 있습니다. 최소한 다음 매개변수를 사용하십시오.

- **BUNDLE** 매개변수에 번들 자원의 이름과 위치를 지정하십시오.
- **JSON-SCHEMA** 매개변수에 JSON 스키마 파일의 이름과 위치를 지정하십시오.
- **MAPPING-LEVEL** 매개변수에 맵핑 레벨을 지정하십시오. 어느 맵핑 레벨이나 사용할 수 있지만 고급 맵핑 옵션을 얻으려면 최신 맵핑 레벨을 사용하십시오.
- **LANG** 매개변수에 생성된 언어 구조의 고급 언어를 지정하십시오.
- **PDSMEM** 및 **PDSCP** 매개변수에 애플리케이션 레코드를 설명하는 언어 구조의 코드 페이지 및 위치를 지정하십시오. DFHJS2LS는 디렉토리 구조가 아닌 언어 구조를 작성합니다.

- CICS에서 JSONTRANSFRM 번들 자원에 사용되는 이름을 지정하십시오. 이 이름은 애플리케이션이 JSON 맵핑을 식별하는 용도로 사용됩니다.

일괄처리 작업은 z/OS UNIX에 번들 디렉토리 구조를 작성합니다. 번들 디렉토리에 번들 Manifest를 포함한 META-INF 서브디렉토리가 있습니다. 일괄처리 작업은 **JSONTRANSFRM** 및 **JSON-SCHEMA** 매개변수에 지정한 파일 이름을 사용하여 번들 디렉토리에 JSON 스키마를 복사하고 JSON 바인딩을 작성하기도 합니다. 또한 일괄처리 작업은 **PDSMEM** 및 **PDSLIB** 매개변수에 지정된 위치에 언어 구조를 작성합니다.

이 JSON 바인딩을 지정하는 **BUNDLE** 자원을 설치하십시오. **JSONTRANSFRM** 번들 자원은 JSON 스키마 및 바인딩 파일의 위치를 정의하는 **JSONTRANSFRM** 자원을 동적으로 작성합니다. 이 번들 자원은 CICS 탐색기를 사용하여 설치된 번들의 내용을 볼 때 CICS에 표시됩니다. 이 자원은 일반 CICS 자원이 아니며 CEMT 또는 CPSM WUI를 사용할 때 표시되지 않습니다.

결과

JSON 변환을 하나 포함한 CICS 번들이 생성됩니다. 언어 구조가 생성됩니다.

다음 예제는 최소한의 매개변수 세트가 지정된 DFHJS2LS를 보여줍니다.

```
//JS2LS JOB 'accounting information',name,MSGCLASS=A
// SET QT='''
//JCLLIB JCLLIB ORDER=FPHLQ.SDFHMOBI
//JAVAPROG EXEC DFHJS2LS,
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test2
LOGFILE=/u/exampleapp/jsbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=/u/exampleapp
PDSMEM=CPYBK2
JSONTRANSFRM=example.jsbind
JSON-SCHEMA=/u/exampleapp/example.json
/*
```

다음에 수행할 작업

애플리케이션 데이터를 JSON으로(그리고 반대로) 변환시키는 애플리케이션을 쓰십시오. 두 변환에 모두 동일한 맵핑을 사용할 수 있습니다.

애플리케이션 데이터를 JSON으로 변환

애플리케이션 데이터를 JSON으로 변환하는 애플리케이션을 쓸 수 있습니다.

시작하기 전에

JSON 바인딩 및 JSON 스키마를 정의하는 사용 가능한 XMLTRANSFORM 번들 자원이 있어야 합니다. Java를 사용하여 변환하려는 경우, Axis2 JVM 서버가 이미 실행 중이어야 합니다.

이 태스크 정보

변환을 수행할 CICS 제공 프로그램 DFHJSON으로 링크하는 애플리케이션을 작성 또는 갱신하십시오.

프로시저

1. 애플리케이션은 채널을 작성하고(예를 들어, *MyChannelName*) 다음 컨테이너를 채널에 넣어야 합니다.

- DFHJSON-DATA
- DFHJSON-TRANSFRM
- DFHJSON-JVMSEVR(선택적)

이 컨테이너에 대한 자세한 정보는 참조 > 애플리케이션 개발에서 JSON 변환기 링크 가능한 인터페이스 컨테이너의 내용을 참조하십시오.

2. 참조 -> 애플리케이션 개발에서 LINK 명령을 사용하여 데이터를 JSON으로 변환하십시오.

```
EXEC CICS LINK PROGRAM('DFHJSON') CHANNEL('MyChannelName')
```

3. 컨테이너 DFHJSON-ERROR를 확보해서 변환 중에 오류가 발생했는지 확인하십시오.
4. 컨테이너 DFHJSON-JJSON을 확보하여 애플리케이션에서 JSON을 이용하십시오.
5. 애플리케이션을 설치하십시오.

결과

애플리케이션이 LINK 명령을 실행할 때 CICS는 JSONTRANSFRM 번들 자원을 확인하여 JSON 바인딩에서 맵핑을 찾고 채널의 컨테이너를 사용하여 애플리케이션 2진 데이터를 JSON으로 변환합니다. JSON은 리턴 시 DFHJSON-JJSON 컨테이너에 배치됩니다. JSON은 JSONTRANSFRM 번들 자원에 정의된 JSON 스키마를 따릅니다.

다음에 수행할 작업

동일한 맵핑을 사용하여 JSON을 애플리케이션 데이터로 변환할 수도 있습니다. 자세한 정보는 647 페이지의 『JSON을 애플리케이션 데이터로 변환』의 내용을 참조하십시오.

JSON을 애플리케이션 데이터로 변환

JSON을 애플리케이션 데이터로 변환하는 애플리케이션을 쓸 수 있습니다.

시작하기 전에

JSON 바인딩 및 JSON 스키마를 정의하는 사용 가능한 XMLTRANSFORM 자원이 있어야 합니다. Java를 사용하여 변환하려는 경우, Axis2 JVM 서버가 이미 실행 중이어야 합니다.

이 태스크 정보

변환을 수행할 CICS 제공 프로그램 DFHJSON으로 링크하는 애플리케이션을 작성 또는 갱신하십시오.

프로시저

1. 채널을 작성하고(예를 들어, *MyChannelName*) 다음 컨테이너를 채널에 넣으십시오.

- DFHJSON-JSON
- DFHJSON-TRANSFRM
- DFHJSON-JVMSERVER

이 컨테이너에 대한 자세한 정보는 참조 > 애플리케이션 개발에서 JSON 변환기 링크 가능한 인터페이스 컨테이너의 내용을 참조하십시오.

2. EXEC CICS LINK PROGRAM API 명령을 사용하여 데이터를 JSON으로 변환하십시오.

```
EXEC CICS LINK PROGRAM('DFHJSON') CHANNEL('MyChannelName')
```

3. 애플리케이션을 설치하십시오.

결과

애플리케이션이 **LINK PROGRAM** 명령을 실행할 때 CICS는 JSONTRANSFRM 자원을 확인하여 JSON 바인딩에서 맵핑을 찾고 채널의 컨테이너를 사용하여 JSON을 애플리케이션 데이터로 변환합니다. 애플리케이션 데이터는 리턴 시 DFHJSON-DATA 비트 컨테이너에 배치됩니다.

다음에 수행할 작업

동일한 맵핑을 사용하여 애플리케이션 데이터를 JSON으로 변환할 수도 있습니다. 자세한 정보는 645 페이지의 『애플리케이션 데이터를 JSON으로 변환』의 내용을 참조하십시오.

JSON 웹 서비스 클라이언트 애플리케이션 작성

링크 가능한 인터페이스를 사용하여 JSON을 변환한 후 WEB 명령을 사용해서 원격 서비스 제공자에게 전송시키는 방식으로 RESTful 웹 서비스를 호출할 애플리케이션을 쓸 수 있습니다.

시작하기 전에

링크 가능 인터페이스를 사용하여 애플리케이션 데이터 및 JSON 변환에 설명된 링크 가능 인터페이스를 사용한 JSON 변환 및 HTTP 클라이언트로서 CICS를 통한 HTTP 요청에 설명된 EXEC CICS WEB API에 익숙해야 합니다.

이 태스크 정보

CICS 애플리케이션의 일부로 다른 시스템에서 호스팅되는 RESTful 웹 서비스를 호출하고자 할 수 있습니다. 이를 위해서는 먼저 원격 서비스와 교환할 데이터를 설명해야 합니다. 그런 다음 서비스에 요청 데이터를 보내고 응답 데이터를 받기 위해 EXEC CICS WEB API를 사용하여 HTTP 프로토콜로 웹 서비스와 통신하는 애플리케이션을 쓸 수 있습니다. 링크 가능한 인터페이스를 사용하여 애플리케이션 데이터를 JSON으로 변환해서 요청의 일부로 사용하고, JSON 응답을 애플리케이션 데이터로 변환시킬 수 있습니다. 일부 서비스는 요청과 응답 모두에 대한 페이로드를 지원하지 않을 수 있습니다.

프로시저

1. 원격 서비스에 인터페이스를 정의하십시오.
 - a. 원격 서비스가 존재하는 경우 요청 및 응답 페이로드를 설명하는 JSON 스키마가 사용 가능한지 확인하십시오. 사용 가능하지 않으면 이 스키마를 작성해야 합니다. 그런 다음 JSON 지원을 사용하여 언어 구조로의 맵핑을 생성하십시오. 자세한 정보는 644 페이지의 『JSON 스키마에서 맵핑 생성』의 내용을 참조하십시오.
 - b. 원격 서비스가 아직 존재하지 않으며 애플리케이션의 데이터 구조를 인터페이스의 기반으로 하려는 경우 JSON 지원을 사용하여 JSON 스키마를 생성하십시오. 그런 다음 JSON 스키마를 원격 서비스 애플리케이션 개발자에게 전달하십시오. 자세한 정보는 641 페이지의 『언어 구조에서 맵핑 생성』의 내용을 참조하십시오.
2. JSON 지원이 생성하는 번들의 BUNDLE 자원을 정의하고 번들을 CICS에 설치하십시오.
3. 원격 서비스 엔드포인트에 대한 URIMAP 자원을 정의하고 설치하십시오. 자세한 정보는 참조 -> 시스템 정의에서 URIMAP 자원의 내용을 참조하십시오.
4. 다음과 같이 원격 서비스를 호출할 애플리케이션을 작성하거나 갱신하십시오.

- a. 원격 서비스에 요청에 대한 JSON 페이로드가 필요한 경우(예를 들어, HTTP 메소드가 POST 또는 PUT일 때) 링크 가능한 인터페이스를 사용하여 애플리케이션 데이터를 JSON으로 변환시키십시오. 자세한 정보는 645 페이지의 『애플리케이션 데이터를 JSON으로 변환』의 내용을 참조하십시오.
- b. **EXEC CICS WEB OPEN** 명령을 사용하여 원격 서비스가 호스팅되는 서버에 대한 연결을 여십시오. 자세한 정보는 참조 -> 애플리케이션 개발에서 **WEB OPEN**의 내용을 참조하십시오.
- c. 서비스 요구사항에 따라 JSON이 제공되고 있음을 나타내는 데이터 유형 헤더 **application/JSON**을 지정하기 위해 **EXEC CICS WEB WRITE HTTPHEADER** 명령을 코딩하려 할 수 있습니다. 자세한 정보는 참조 -> 애플리케이션 개발에서 **WEB WRITE HTTPHEADER**의 내용을 참조하십시오.
- d. **EXEC CICS WEB CONVERSE** 명령을 코딩하여 원격 서비스에 요청을 보내고 응답을 수신하십시오. 필요에 따라 조회 문자열 또는 요청 본문을 지정하십시오 (DFHJSON-JSON 컨테이너에서). 서비스의 응답을 예상하면 DFHJSON-JSON 컨테이너를 지정하여 응답 JSON을 수신하십시오. 자세한 정보는 참조 -> 애플리케이션 개발에서 **WEB CONVERSE**의 내용을 참조하십시오.
- e. 추가 요청이 예상되지 않으면 **EXEC CICS WEB CLOSE** 명령을 코딩하여 연결을 닫으십시오. 자세한 정보는 참조 -> 애플리케이션 개발에서 **WEB CLOSE**의 내용을 참조하십시오.
- f. **EXEC CICS WEB CONVERSE** 명령에 필요한 HTTP 응답 코드가 리턴되는지 확인하고 오류가 발생한 경우 적절한 조치를 수행하십시오. 예를 들어, 요청을 다시 시도하거나 사용자에게 오류를 리턴하십시오.
- g. 원격 서비스로부터 응답 본문이 예상되면 링크 가능한 인터페이스를 사용하여 JSON을 애플리케이션 데이터로 변환시키십시오. 자세한 정보는 645 페이지의 『애플리케이션 데이터를 JSON으로 변환』의 내용을 참조하십시오.

결과

JSON 페이로드의 RESTful 웹 서비스를 호출할 수 있는 애플리케이션을 작성했습니다.

제 6 부 비즈니스 서비스 및 번들

CICS 애플리케이션에서 서비스 지향 구조에 참여하는 비즈니스 서비스 작성 및 번들로 전개.

제 21 장 CICS 애플리케이션에서 비즈니스 서비스 작성

비즈니스 서비스인 애플리케이션은 서비스 지향 구조(SOA)에 참여할 수 있습니다. 비즈니스 서비스는 기술 구현보다는 비즈니스 프로세스 및 모델에 맞춰진 서비스입니다. CICS에 제공된 지원을 사용하여 기존 및 새 CICS 애플리케이션을 비즈니스 서비스의 일부로 노출시킬 수 있습니다.

CICS 애플리케이션에서 다음 두 가지 유형의 서비스를 작성할 수 있습니다.

채널 기반 서비스

이 서비스는 SCA(Service Component Architecture) 지원을 사용하여 애플리케이션을 서비스 구성요소의 일부로 노출시킵니다. 이 서비스를 위한 인터페이스는 채널입니다. 채널 기반 서비스는 **INVOKE SERVICE** API 명령을 사용하여 기타 CICS 애플리케이션을 통해서만 호출할 수 있습니다.

XML 기반 서비스

이 서비스는 일반적으로 XML을 사용하여 다른 애플리케이션과 인터페이스하고 바인딩을 사용하여 데이터를 변환하는 웹 서비스 제공자 또는 요청자 애플리케이션입니다. SCA를 사용하여 웹 서비스 애플리케이션을 구성요소로 설명할 수도 있습니다. 이 서비스를 위한 인터페이스는 데이터 변환을 위한 바인딩이 있는 XML입니다. XML 기반 서비스는 **INVOKE SERVICE** API 명령을 사용하여 CICS 애플리케이션으로 또는 외부 클라이언트로 호출할 수 있습니다.

XML 기반 서비스는 **TRANSFORM** API 명령을 사용하여 XML으로(부터) 애플리케이션을 맵핑하는 애플리케이션일 수도 있습니다. XML 지원은 언어 구조나 XML 스키마를 사용하여 XML 바인딩을 생성하고 번들도 작성합니다.

두 가지 유형의 서비스 모두 CICS의 파이프라인 지원을 사용합니다.

SCA(Service Component Architecture)

SCA(Service Component Architecture)는 서비스 지향 구조(SOA)를 사용하여 애플리케이션과 시스템을 빌드하기 위한 프로그래밍 모델을 설명하는 스펙 세트입니다. SCA는 서비스 구현을 위한 이전 접근법을 확장 및 보완하고 웹 서비스와 같은 개방형 표준을 빌드합니다.

스펙은 컴포지트 애플리케이션을 작성하는 방법을 설명합니다. 컴포지트 애플리케이션은 새 애플리케이션의 비즈니스 로직을 함께 구현하는 하나 이상의 구성요소를 결합해서 작성됩니다. 구성요소는 비즈니스 로직과 구성 정보를 구현하는 하나의 애플리케이션으로 이루어집니다. 애플리케이션 개발자는 다른 구성의 동일한 애플리케이션을 사용하여 여

러 다른 구성요소를 형성합니다. 구성요소는 다른 구성요소에 서비스를 제공하고 서비스 지향 인터페이스를 사용하여 다른 서비스가 제공하는 기능을 이용합니다.

애플리케이션 개발자는 구성요소를 함께 어셈블하여 특정 비즈니스 요구사항에 대한 솔루션을 작성할 수 있습니다. 컴포지트 애플리케이션은 비즈니스 애플리케이션을 위해 특별히 작성된 새 구성요소와 다른 애플리케이션으로부터 재사용되는 기존 구성요소를 모두 포함할 수 있습니다.

CICS는 서비스 구성요소를 어셈블하여 컴포지트를 형성하는 방법을 설명하는 *SCA 어셈블리 모델 1.0* 스펙을 지원합니다. 컴포지트는 SCA의 전개 단위로, SCDL이라는 XML 언어로 설명됩니다. 컴포지트는 구성요소, 서비스, 참조, 특성 선언, 이러한 요소 간의 연결을 설명하는 와이어링을 포함할 수 있습니다. 컴포지트를 다른 컴포지트와 함께 구성요소에 사용하여 여러 하위 레벨 서비스 세트를 통해 상위 레벨 서비스가 내부적으로 구현되는 계층 구축을 허용할 수도 있습니다.

구성요소의 구조

단순한 유형의 구성요소에는 하나의 서비스와 하나의 참조가 있습니다. 서비스는 하나 이상의 조작을 포함할 수 있는 구성요소의 주소 지정 가능한 인터페이스입니다. 참조는 또 다른 구성요소가 제공하는 서비스에 대한 종속성입니다. 구성요소의 바인딩을 서비스 및 참조 모두에 정의할 수 있습니다.

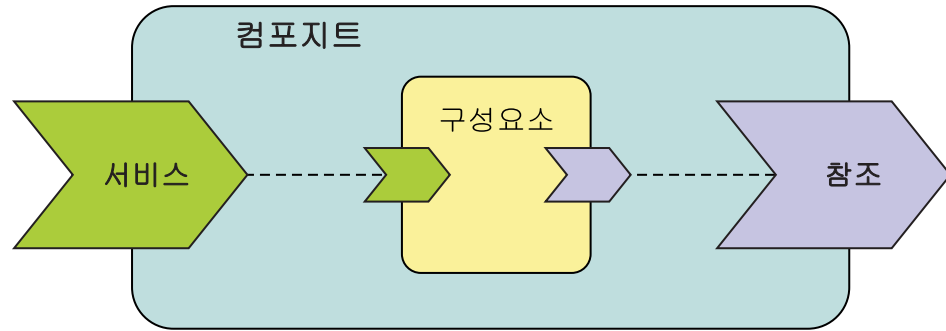
- 구성요소 서비스는 바인딩을 사용하여 다른 내부 구성요소가 구성요소를 호출하기 위해 사용해야 하는 액세스 메커니즘을 설명합니다.
- 구성요소 참조는 바인딩을 사용하여 다른 구성요소를 호출하는 데 사용되는 액세스 메커니즘을 설명합니다.

구성요소 서비스와 참조는 내부적이며 구성요소 대 구성요소 통신에만 사용됩니다. 외부 인터페이스를 작성하려면 구성요소를 컴포지트 안에 전개해야 합니다. 매우 단순한 컴포지트에는 외부 서비스 하나와 외부 참조가 하나 있습니다.

- 컴포지트 서비스는 바인딩을 사용하여 외부 클라이언트가 서비스를 호출하기 위해 사용해야 하는 액세스 메커니즘을 설명합니다.
- 컴포지트 참조는 바인딩을 사용하여 다른 서비스를 호출하는 데 사용되는 액세스 메커니즘을 설명합니다.

컴포지트 서비스 및 참조는 구성요소 자체의 일부가 아니라 구성요소와 다른 외부 클라이언트 또는 서비스 사이의 경계입니다.

다음 다이어그램은 내부 및 외부 서비스와 참조가 모두 있는 단순 구성요소를 포함한 컴포지트를 보여줍니다.



SCA 컴포지트 및 와이어링

애플리케이션 개발자는 서비스와 참조를 함께 와이어링해서 SCA 컴포지트를 작성합니다. 와이어는 구성요소에서 대상으로 제어 및 데이터를 전달하는 커넥터입니다.

z Systems용 Rational Developer를 사용하여 다음 두 가지 방법으로 컴포지트의 구성요소를 함께 와이어링할 수 있습니다.

승격 컴포지트의 구성요소 중 하나에 정의된 구성요소 서비스를 승격시켜서 컴포지트 서비스를 와이어링할 수 있습니다. 마찬가지로 컴포지트 참조는 구성요소 참조를 승격시킬 수 있습니다. 컴포지트 서비스를 또 다른 컴포지트 서비스로 승격시키거나 컴포지트 참조를 컴포지트 참조로 승격시킬 수는 없습니다. 서비스나 참조에 바인딩을 추가하는 경우 승격된 컴포지트 서비스 또는 참조의 바인딩 속성이 구성요소 서비스 또는 참조의 속성보다 우선합니다.

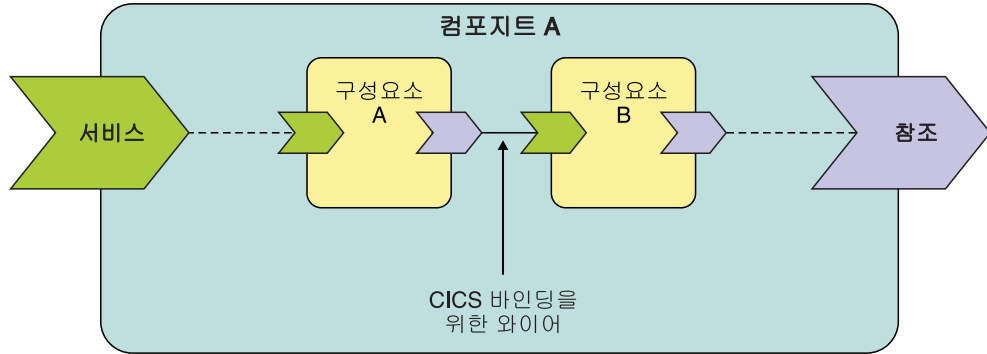
외부 서비스에 다른 유형의 바인딩이 사용되는 경우에는 내부 서비스를 승격시킬 수 없습니다. 예를 들어, 외부 서비스에 CICS 바인딩이 있으면 내부 서비스에 웹 서비스 바인딩이 있을 수 없습니다.

대상 참조에 대상 속성을 지정해서 컴포지트의 두 가지 구성요소를 함께 연결하여 구성요소 참조를 다른 구성요소 서비스에 직접 와이어링할 수 있습니다. 이 와이어링은 컴포지트의 내부 연결입니다. 마찬가지로 컴포지트 참조에 대상 속성을 지정해서 두 개의 컴포지트를 함께 와이어링할 수 있습니다. 이 와이어링은 복합 컴포지트의 일반 연결입니다.

단순 컴포지트

한 컴포지트는 서로 와이어링된 둘 이상의 구성요소를 포함할 수 있습니다. 다음 예제에서, 컴포지트 A에는 두 개의 구성요소가 있습니다. 컴포지트 A에는 구성요소 B에 서비스에 대한 종속성이 있습니다. 각 구성요소에는 서비스 및 참조가 있습니다. 구성요소 A의 참조에 정의된 바인딩과 구성요소 B의 서비스가 호환 가능해야 합니다. 컴포지트 A는 컴포지트 A가 제공하는 서비스를 필요로 하는 애플리케이션으로부터 하위 레벨 세부사항을 숨겨서 두 개의 구성요소를 캡슐화합니다.

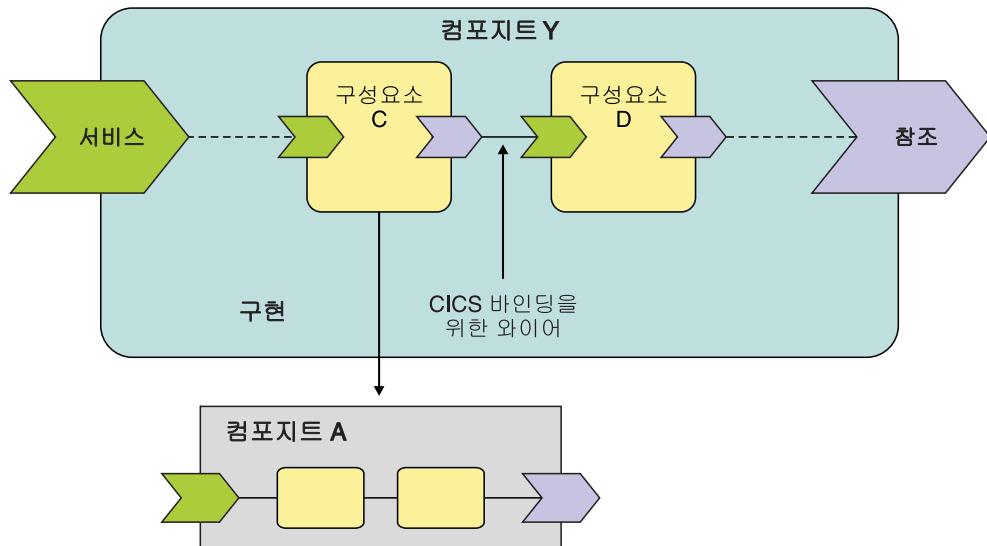
컴포지트의 외부 서비스 및 참조는 구성요소 A의 서비스와 구성요소 B의 참조를 승격시켜서 형성합니다. 구성요소 A와 B 간의 와이어가 대상이며 각 인터페이스의 바인딩은 동일합니다. 이 예제에서 구성요소는 CICS 바인딩을 사용하고 있습니다.



복합 컴포지트

구성요소의 구현 세부사항을 캡슐화하는 모델은 계층적 컴포지트 애플리케이션을 작성하는 매우 유연한 방식입니다. 다음 예제에서, 컴포지트 Y에는 두 개의 구성요소가 있습니다. 하지만 이 예제에서 구성요소 C의 구현은 두 개의 구성요소가 내부에 있는 컴포지트 자체입니다. 컴포지트 Y가 제공하는 서비스를 호출하려는 클라이언트가 컴포지트 C에 있는 구성요소에 대한 지식이 필요하지 않습니다. 즉, 기본 구현 세부사항이 클라이언트에 영향을 미치지 않고 변경될 수 있다는 의미입니다.

컴포지트 Y의 외부 서비스 및 참조는 구성요소 C의 서비스와 구성요소 D의 참조를 승격시켜서 형성합니다. 구성요소 C와 D 간의 와이어가 대상이며 각 인터페이스의 바인딩은 동일합니다. 이 예제에서 구성요소는 CICS 바인딩을 사용하고 있습니다.



컴포지트 작성 및 전개 우수 사례

구성요소를 컴포지트 애플리케이션으로 어셈블링하기 위한 모델은 매우 유연합니다. 애플리케이션 컴포지트를 여러 CICS 리전에서 재사용하려면 애플리케이션을 개발 및 테스트를 거쳐 프로덕션으로 이동시키고 있거나 프로덕션 환경에서 애플리케이션을 복제하고 있는지 여부와 무관하게, 애플리케이션 로직을 바인딩에서 분리시키는 것이 좋습니다.

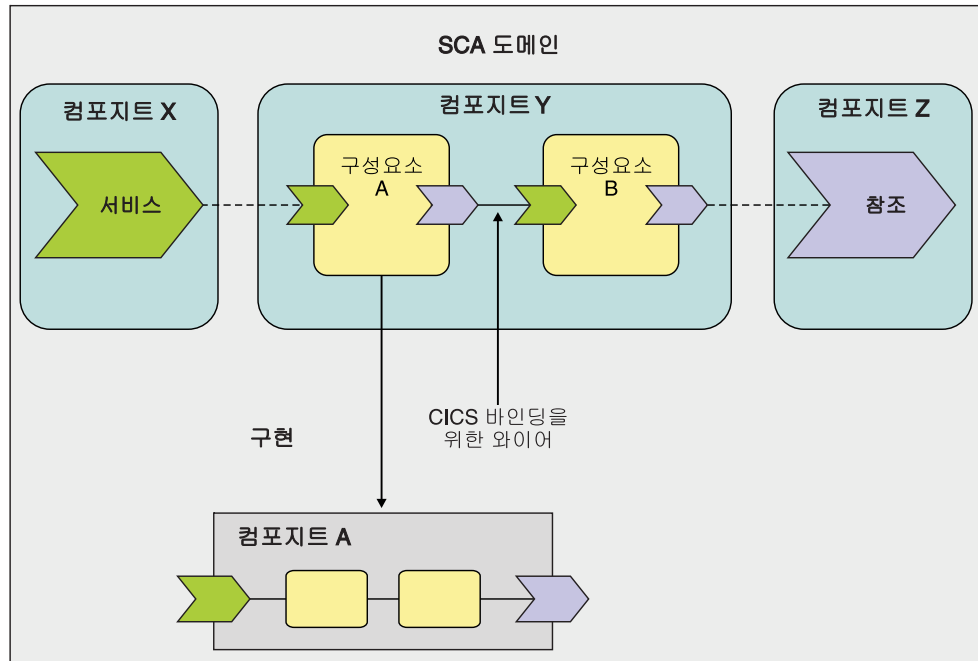
컴포지트 서비스 및 참조에서 구현 로직 분리

동일한 애플리케이션 로직을 여러 다른 CICS 리전에서 보다 쉽게 재사용하려면 컴포지트 서비스 및 참조를 애플리케이션의 구현 로직에서 분리시켜서 별도의 번들로 전개하십시오. 컴포지트 서비스와 참조는 특정 시스템 자원 세트나 트랜잭션 ID와 같은 CICS 리전 고유의 정보를 포함할 수 있습니다. 이 방법을 사용하여 애플리케이션 컴포지트를 재전개할 필요 없이 컴포지트 서비스와 참조를 변경 또는 갱신할 수 있습니다.

번들에 대한 SCA 도메인 정의

번들은 CICS에 BUNDLE 자원으로 표시됩니다. CICS에 전개된 모든 번들은 값이 비어 있어도 기본적으로 SCA 도메인이 동일합니다. BUNDLE 자원에는 시스템 프로그래머가 SCA 도메인을 표현할 절대 이름을 설정하는 데 사용할 수 있는 BASESCOPE라는 선택적 속성이 있습니다. 서비스와 참조를 컴포지트 애플리케이션에서 분할하여 별도의 번들로 전개하는 권장 모델을 따르는 경우, 번들이 관련되어 있음을 표시하기 위해 각 BUNDLE 자원에 동일한 BASESCOPE 값이 사용되도록 요청할 수 있습니다.

BASESCOPE 속성에 여러 다른 SCA 도메인을 지정해서 동일한 번들을 CICS 리전에 여러 번 전개할 수도 있습니다. CICS는 SCA 도메인과 컴포지트를 함께 사용하여 런타임 처리 중에 서비스를 식별합니다. 서비스 범위는 요청을 처리 중인 태스크에 사용 가능합니다. BASESCOPE 값을 고유 URI로 지정하는 것이 좋습니다.



채널 기반 서비스 작성

채널 기반 서비스는 z Systems용 Rational Developer와 같은 도구를 사용하여 함께 어셈블되며 구성요소로 설명되는 CICS 애플리케이션입니다. 이 서비스는 **INVOKE SERVICE** API 명령을 사용하며 채널의 컨테이너에서 2진 데이터를 전달하는 기타 CICS 애플리케이션에만 사용 가능합니다.

이 태스크 정보

SCA에서 채널 기반 서비스로 표시하려는 애플리케이션은 비즈니스 로직의 구현입니다. 구성요소 서비스는 애플리케이션에 인터페이스를 WSDL로 설명합니다. 구성요소에는 조작 및 바인딩을 포함할 수 있는 인터페이스가 하나 있습니다. 채널 기반 서비스의 경우 CICS 바인딩이 채널을 설명합니다. 이 바인딩을 선택하는 경우 z Systems용 Rational Developer가 `binding.cics` 섹션을 SCDL에 추가합니다. 애플리케이션이 컨테이너의 채널 및 2진 데이터를 예상하기 때문에 채널 기반 서비스는 데이터 매핑이 없습니다.

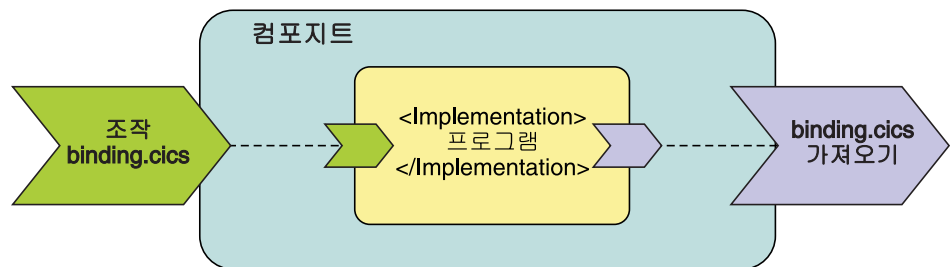
프로시저

1. z Systems용 Rational Developer를 사용하여 컴포지트를 작성하십시오.
 - a. 애플리케이션 이름을 구성요소 구현으로 지정하십시오.
 - b. 도구에서 direct 매핑 모드를 선택해서 CICS 바인딩을 적합한 서비스 또는 참조에 추가하십시오.
 - 컴포지트 서비스에 연결된 CICS 바인딩에 URI를 지정하는 경우 이 URI는 서비스의 외부 이름을 제공합니다. CICS에서 이 URI는 상대 경로 이름입니다.

다. 예를 들어, 서비스를 `http://myhost:port/myService`로 노출할 경우 바인딩의 URI는 `myService`입니다. 호스트와 포트가 컴포지트 애플리케이션의 제어를 벗어나기 때문입니다.

- 컴포지트 참조에 연결된 CICS 바인딩에 URI를 지정하는 경우에는 URI가 대상 서비스에 완전합니다(예: `http://myhost:port/myService` 또는 `cics://PROGRAM/prog1?user=user1`).

c. 옵션: 컴포지트 참조의 가져오기를 정의하십시오. 가져오기는 구성요소나 컴포지트의 종속성을 정의합니다.



CICS에서 컴포지트를 작성하고 전개하는 권장 방법에 대한 세부사항은 657 페이지의 『컴포지트 작성 및 전개 우수 사례』의 내용을 참조하십시오.

2. 컴포지트를 CICS에 번들로 전개하십시오. z Systems용 Rational Developer는 번들 Manifest를 생성하고 SCDL 및 기타 아티팩트를 패키징합니다. Manifest는 CICS에서 BUNDLE 자원을 설치하기 위해 필요한 모든 자원과 메타데이터를 설명합니다. BUNDLE 자원은 CICS 리전의 컴포지트를 나타냅니다. Manifest는 컴포지트를 SCACOMPOSITE 자원 유형으로 정의하고 상대 경로를 사용하여 SCDL의 위치를 참조합니다.
3. BUNDLE 자원을 작성하고 설치하십시오. 선택적으로 SCA 도메인을 BUNDLE 자원 정의에 설정할 수 있습니다. SCA 도메인은 일반적으로 비즈니스 기능 영역을 제공하는 서비스 세트를 나타냅니다. CICS는 SCA 도메인과 서비스 이름을 결합하여 서비스를 식별하므로 다른 SCA 도메인을 사용하여 동일한 번들을 설치할 수 있습니다. SCA 도메인 추가 방법의 세부사항과 예제는 *CICS* 자원 정의 안내서의 내용을 참조하십시오.

설치에 성공하려면 번들의 모든 전제조건이 BUNDLE 자원의 CICS 리전에서 사용 가능한지 확인해야 합니다.

4. **INVOKE SERVICE** API 명령을 사용하여 채널 기반 서비스를 호출하는 애플리케이션을 작성하십시오.

```
EXEC CICS INVOKE SERVICE('servicename')
          CHANNEL(channel)
          OPERATION(operation)
```

*servicename*은 서비스의 외부 이름이고, *channel*은 채널의 16바이트 이름이며, *operation*은 임의의 값입니다. *operation*이 명령의 필수 옵션이지만 채널 기반 서비스에는 값이 사용되지 않습니다.

결과

애플리케이션이 채널 기반 서비스를 호출할 때 CICS는 서비스의 이름을 분석하고 **EXEC CICS LINK** 명령을 실행하여 컴포지트에 정의한 애플리케이션에 지정된 채널과 콘텐츠를 전달합니다. 특정 요청자 파이프라인을 바인딩에 지정한 경우 CICS는 요청자 파이프라인을 통해 요청을 실행합니다. 요청자 파이프라인이 바인딩에 지정되어 있지 않으면 CICS는 요청의 요청자 파이프라인을 동적으로 작성합니다.

다음에 수행할 작업

서비스가 예상대로 작동하는지 테스트하고 유효성을 검사할 수 있습니다. IBM CICS Explorer를 사용하여 BUNDLE 자원과 내용을 볼 수 있습니다. 모든 자원을 함께 관리할 BUNDLE 자원을 사용 및 사용 안함으로 설정할 수도 있습니다.

XML 기반 서비스 작성

XML 기반 서비스는 XML을 사용하여 다른 애플리케이션과 인터페이스하고 바인딩을 사용하여 데이터를 변환하는 웹 서비스 제공자 또는 요청자 애플리케이션입니다. XML 기반 서비스는 **INVOKE SERVICE API** 명령을 사용하는 CICS 애플리케이션이나 외부 네트워크에 있는 비즈니스 서비스에 사용 가능합니다.

이 태스크 정보

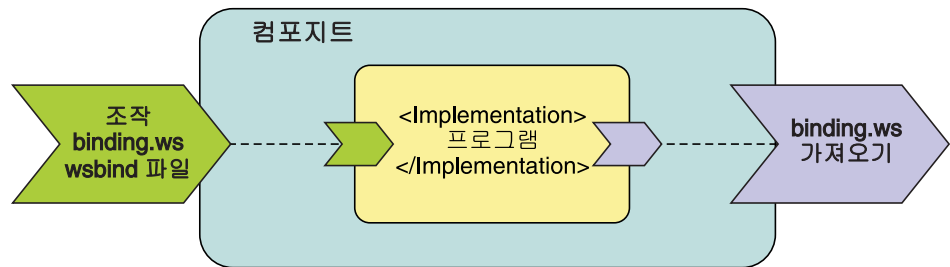
CICS의 웹 서비스 지원을 사용하여 웹 서비스를 작성하거나 z Systems용 Rational Developer를 사용할 수 있습니다. z Systems용 Rational Developer를 사용하면 웹 서비스로부터 SCA 구성요소를 작성할 수도 있습니다. 웹 서비스로부터 구성요소를 작성하는 것의 장점은 다음과 같습니다.

- 기존구성요소를 쉽게 재사용하여 z Systems용 Rational Developer로 새 컴포지트 애플리케이션을 빠르게 개발할 수 있습니다.
- SCDL을 사용하여 애플리케이션을 변경할 필요 없이 쉽게 변경하는 메타데이터로 그리고 애플리케이션으로부터 구성 정보를 이동시켜서 웹 서비스를 설명할 수 있습니다. 예를 들어, 다른 트랜잭션 및 사용자 ID로 웹 서비스를 실행하려는 경우 웹 서비스 바인딩 파일을 재생성할 필요 없이 SCDL을 변경할 수 있습니다.

SCA에서, 웹 서비스로 표시하려는 애플리케이션은 비즈니스 로직의 구현입니다. 애플리케이션은 구성요소의 <Implementation> 요소에 정의됩니다. 구성요소 서비스는 애플리케이션에 인터페이스를 WSDL로 설명합니다. 구성요소에는 많은 조작과 바인딩을 포함할 수 있는 인터페이스가 하나 있습니다.

프로시저

1. z Systems용 Rational Developer를 사용하여 컴포지트를 작성하십시오.
 - a. 애플리케이션 이름을 구성요소 구현으로 지정하십시오.
 - b. CICS 외부에서 서비스를 사용하려면 해당 서비스 또는 참조에 웹 서비스 바인딩을 추가하십시오. CICS 애플리케이션에만 서비스를 사용하려는 경우에는 해당 서비스 또는 참조에 CICS 바인딩을 추가하고 도구에서 mapped 맵핑 모드를 선택하십시오. 도구는 SCDL로 바인딩을 설명합니다. SCDL은 XML을 해당 고급 언어로 변환시키는 데이터 맵핑도 포함합니다.
 - c. 컴포지트 참조의 가져오기를 정의하십시오. 가져오기는 CICS 리전에서 충족시켜야 하는 컴포지트의 종속성을 정의합니다.



CICS에서 컴포지트를 작성하고 전개하는 권장 방법에 대한 세부사항은 657 페이지의 『컴포지트 작성 및 전개 우수 사례』의 내용을 참조하십시오.

2. 컴포지트를 CICS에 번들로 전개하십시오. z Systems용 Rational Developer는 번들 Manifest를 생성하고 SCDL 및 기타 아티팩트를 패키징합니다. Manifest는 CICS에서 BUNDLE 자원을 설치하기 위해 필요한 모든 자원과 메타데이터를 설명합니다. BUNDLE 자원은 CICS 리전의 컴포지트를 나타냅니다. Manifest는 컴포지트를 SCACOMPOSITE 자원 유형으로 정의하고 상대 경로를 사용하여 SCDL의 위치를 참조합니다.
3. BUNDLE 자원을 작성하고 설치하십시오. 선택적으로 SCA 도메인을 BUNDLE 자원 정의에 설정할 수 있습니다. SCA 도메인은 일반적으로 비즈니스 기능 영역을 제공하는 서비스 세트를 나타냅니다. CICS는 SCA 도메인과 서비스 이름을 결합하여 서비스를 식별하므로 다른 SCA 도메인을 사용하여 동일한 번들을 설치할 수 있습니다. SCA 도메인 추가 방법의 세부사항과 예제는 CICS 자원 정의 안내서의 내용을 참조하십시오.

설치에 성공하려면 번들의 모든 전제조건이 BUNDLE 자원의 CICS 리전에서 사용 가능한지 확인해야 합니다. BUNDLE 자원이 설치되면 CICS는 Manifest와 SCDL의 정보를 사용하여 WEBSERVICE 및 URIMAP 자원을 작성합니다. CICS는 참조의 가져오기 섹션에 정의된 자원이 CICS 리전에 있는지 확인하기도 합니다. 가져오기는 애플리케이션의 전제조건을 정의합니다.

결과

웹 서비스가 CICS에 설치되었습니다.

다음에 수행할 작업

웹 서비스가 예상대로 작동하는지 테스트하고 유효성을 검사할 수 있습니다. IBM CICS Explorer를 사용하여 BUNDLE 자원과 내용을 볼 수 있습니다. 모든 자원을 함께 관리할 BUNDLE 자원을 사용 및 사용 안함으로 설정할 수도 있습니다.

CICS 서비스 처리

애플리케이션은 서비스를 호출할 때 예를 들어, 웹 서비스 요청자가 SOAP 메시지를 전송하거나 CICS 애플리케이션이 **INVOKE SERVICE** API 명령을 사용할 때 CICS는 파이프라인을 사용하여 요청을 처리합니다.

CICS는 서비스 요청자 및 서비스 제공자 파이프라인에서 XML 기반 서비스를 처리합니다. 각 파이프라인 유형마다 네트워크로 요청을 전송할 수 있는 전송 핸들러가 있습니다. CICS 애플리케이션은 외부 클라이언트에 대한 웹 서비스 제공자 또는 웹 서비스 요청자일 수 있으며 HTTP, HTTPS, JMS 또는 WebSphere MQ 프로토콜을 사용하여 요청을 전송하고 수신할 수 있습니다.

웹 서비스가 둘 다 CICS 애플리케이션이면 서비스 요청자는 **INVOKE SERVICE** 명령과 `cics://`로 시작하는 URI를 사용하여 파이프라인 처리를 최적화할 수 있습니다. 2진 데이터를 XML로 변환하는 데이터 변환이 일상적으로 발생하고, 파이프라인이 메시지 핸들러를 런 스루하며, 파이프라인 전송 핸들러가 다른 프로그램으로 링크하거나 지정된 URI 유형에 따라 파이프라인을 시작합니다.

예를 들어, `cics://SERVICE/service?targetServiceUri=targetServiceUri` URI를 사용하여 네트워크 사용 오버헤드 없이 제공자 파이프라인을 통해 요청을 실행할 수 있습니다. CICS는 `service` 및 `targetServiceUri` 값을 사용하여 URIMAP으로 요청을 해결합니다.

이 URI는 요청자와 제공자 애플리케이션이 다른 언어로 쓰여졌을 때나 다른 맵핑 레벨을 사용하고 상이한 2진 데이터를 예상할 때 유용할 수 있습니다.

CICS는 **EXEC CICS LINK PROGRAM**을 사용하여 서비스에 대한 SCDL에 정의된 프로그램으로 채널 및 컨테이너를 전달해서 채널 기반 서비스에 대한 호출을 처리합니다. 호출 애플리케이션도 `cics://`로 시작하는 URI를 제공하는 경우 CICS는 요청자 파이프라인에서 추가 처리를 수행할 수 있습니다. 바인딩이 요청자 파이프라인을 지정하지 않는 경우에는 CICS가 동적으로 요청에 대한 파이프라인을 작성합니다.

URI 형식

URI가 `cics://`로 시작하는 경우 파이프라인 전송 핸들러는 프로그램에 링크하거나, 또 다른 요청자 파이프라인을 시작하거나, 제공자 파이프라인을 시작할 수 있습니다. 세 가지 유형의 URI가 있습니다.

cics://PROGRAM/program

이 URI 유형의 파이프라인 처리는 동일한 CICS 리전에 있는 웹 서비스 요청자 및 제공자에 발생하는 로컬 최적화와 유사합니다. *program*은 요청자 파이프라인 처리가 완료될 때 전송 핸들러가 링크하는 프로그램의 이름을 정의합니다. 전송 핸들러는 **EXEC CICS LINK PROGRAM** 명령을 사용하여 채널과 컨테이너를 특정 프로그램으로 전달합니다. URI에 추가 옵션이 표시되면 CICS는 COMMAREA를 프로그램에 전달할 수도 있습니다.

cics://SERVICE/service?targetServiceUri=targetServiceUri

이 유형의 파이프라인 처리는 요청을 네트워크로 전송하지 않음으로써 요청을 최적화합니다. 하지만 요청이 요청자와 제공자 파이프라인 모두를 통해 실행합니다. *service* 값은 특정 프로그램이 아닌 서비스의 이름을 정의합니다. *targetServiceUri* 값은 URIMAP 자원이 분석하는 서비스의 경로를 정의합니다. 전송 핸들러는 URIMAP을 사용하여 동일한 CICS 리전의 올바른 제공자 파이프라인으로 요청을 전송합니다.

cics://PIPELINE/pipeline?targetServiceUri=targetServiceUri

pipeline 값은 요청자 파이프라인의 이름을 정의하며 전송 핸들러는 지정된 요청자 파이프라인을 시작하기 전에 *targetServiceUri*의 값을 DFHWS-URI에 넣습니다. 이 URI 유형은 여러 다른 메시지 핸들러 세트에서 요청이 처리될 수 있도록 많은 요청자 파이프라인을 함께 체인으로 연결할 수 있습니다.

또한 파이프라인의 메시지 핸들러가 파이프라인 처리를 제어하기 위해 애플리케이션이 제공한 URI를 덮어쓸 수도 있습니다.

이 URI의 지정 가능한 매개변수에 대한 세부사항은 *CICS 웹 서비스 안내서*의 내용을 참조하십시오.

SCA 문제점 해결

CICS에서 SCA 컴포지트 애플리케이션을 구현할 때의 문제점은 전개 프로세스 중이나 런타임에 CICS가 서비스 요청을 처리 중일 때 발생할 수 있습니다.

이 태스크 정보

CICS는 오류가 발생하면 CICS 로그에 DFHPI 또는 DFHRL 접두부가 붙은 메시지를 발행합니다. 일반적으로 메시지에는 문제점을 해결하기 위해 수행할 수 있는 추가 조치를 제공하는 사용자 응답이 포함되어 있습니다.

프로시저

- BUNDLE 자원이 사용 안함 상태로 설치되며 사용자가 이를 사용으로 설정할 수 없습니다. 가장 가능한 원인은 동적으로 작성된 자원이 UNUSABLE 상태로 설치되었기 때문입니다.
 1. IBM CICS Explorer를 사용하여 CICS가 번들에 대해 작성한 자원을 보십시오.
 2. UNUSABLE 상태인 자원을 판별하고 문제점을 수정하십시오. 로그의 추가 DFHPI 또는 DFHRL 메시지에 자원이 이 상태로 설치된 이유가 표시되어 있을 수 있습니다.
 3. 문제를 해결한 후 BUNDLE 자원을 버리고 다시 설치하십시오.
- BUNDLE 자원이 사용 안함 상태로 설치되고 DFHPI2005 오류 메시지가 수신됩니다. 이 메시지를 수신할 경우 컴포지트 애플리케이션에 바인딩 호환성 문제가 있습니다.
 1. z Systems용 Rational Developer에서 컴포지트를 편집하여 컴포지트 서비스 또는 참조의 바인딩을 변경하십시오. 서비스의 바인딩이 모두 CICS 바인딩 또는 웹 서비스 바인딩인지 확인해야 합니다.
 2. CICS에 번들을 다시 전개하십시오.
 3. 사용 안함 상태의 BUNDLE 자원을 버리고 다시 설치하십시오.
- 애플리케이션이 서비스를 호출할 수 없으며 CICS가 DFHPI 오류 메시지를 발행합니다. URI가 컴포지트 서비스에 올바른지 확인하십시오.

제 22 장 전개를 위한 애플리케이션 패키징

애플리케이션을 개발한 후에는 플랫폼에 전개할 수 있도록 CICS Explorer를 사용하여 패키징할 수 있습니다. CICS Explorer를 사용하면 애플리케이션 종속성과 애플리케이션을 구성하는 자원을 정의할 수 있습니다.

이 태스크 정보

애플리케이션을 패키징하려면 각 애플리케이션 구성요소에 대한 CICS 번들과 이를 함께 그룹화하고 라이프사이클 관리를 제공하는 애플리케이션 번들을 작성하십시오. 각 CICS 번들은 애플리케이션의 구성요소에 대한 자원을 포함하며, 애플리케이션 전개 프로세스 동안 CICS 리전에 설치됩니다. 애플리케이션 번들은 이름과 버전 정보를 포함하여 애플리케이션을 설명하는 관리 번들 유형입니다. 이 관리 번들은 애플리케이션에 대한 종속성과 자원을 포함하는 CICS 번들을 참조합니다. 애플리케이션 자원은 모두 함께 설치되고 관리됩니다.

CICS 번들에서 애플리케이션에 대한 애플리케이션 시작점을 선언하십시오. 애플리케이션 시작점은 애플리케이션에 대한 액세스 지점인 자원을 식별하고 플랫폼에 전개된 여러 버전의 애플리케이션에 대한 사용자의 액세스를 제어합니다. 애플리케이션 시작점을 통해 애플리케이션에 액세스하면 애플리케이션이 전개된 CICS 리전이 애플리케이션 관리 및 모니터링을 위한 애플리케이션 컨텍스트 데이터를 생성합니다.

또한 애플리케이션 바인딩을 작성하여 대상 플랫폼에 애플리케이션 전개 방법을 정의하십시오. 애플리케이션 바인딩은 애플리케이션에 대한 각 CICS 번들을 대상 플랫폼에 있는 하나 이상의 각 CICS 리전 유형에 맵핑합니다.

다중 버전화를 사용하여 동일한 플랫폼 인스턴스에 다중 버전의 애플리케이션을 동시에 설치하고 서비스 중단 없이 사용자가 새 버전을 사용하게 할 수 있습니다. 다중 버전화 및 다중 버전화가 가능하도록 애플리케이션을 구조화할 경우의 장점에 대한 자세한 정보는 플랫폼에 전개된 애플리케이션에 대한 다중 버전화를 참조하십시오.

애플리케이션을 위해 설계된 다른 자원 및 가져온 자원과의 조합으로 개인용 자원으로 지원되는 CICS 자원을 사용하는 플랫폼에 전개된 애플리케이션은 다중 버전화에 적합합니다. 다음 자원은 다중 버전화 애플리케이션의 일부로 지원됩니다.

- 애플리케이션의 일부인 CICS 번들에서 정의된 PROGRAM 자원
- 애플리케이션의 일부인 CICS 번들에서 정의된 LIBRARY 자원
- 애플리케이션의 일부인 CICS 번들에서 정의된 PACKAGESET 자원
- 정책
- 애플리케이션 시작점의 명령문

- 애플리케이션에 대한 종속 항목 또는 가져오기로 정의되는 모든 자원

개인용 자원으로 지원되는 CICS 자원에 대한 자세한 내용은 구성에서 애플리케이션 버전을 위한 개인용 자원의 내용을 참조하십시오.

애플리케이션의 서로 다른 버전 사이의 자원 이름 충돌을 피하기 위해 자원을 적절하게 관리하는 경우 기타 자원이 다중 버전화된 애플리케이션에 포함될 수 있습니다.

플랫폼에 애플리케이션을 전개할 준비가 되면 애플리케이션 번들과 애플리케이션 바인딩을 zFS의 플랫폼 홈 디렉토리로 내보내십시오. 플랫폼 홈 디렉토리의 애플리케이션 번들을 가리키는 CICSplex SM APPLDEF 자원 정의인 애플리케이션 정의도 작성하십시오.

CICS Explorer의 CICS 클라우드 퍼스펙티브에서 프로젝트 탐색기 보기를 사용하여 전개할 애플리케이션을 패키징할 수 있습니다. 다음 단계 각각에 대한 상세 정보는 CICS Explorer 제품 문서의 애플리케이션에 대한 작업의 내용을 참조하십시오.

팁: 애플리케이션과 애플리케이션 바인딩에 대한 작업을 하기 전에 대상 플랫폼의 플랫폼 프로젝트가 CICS Explorer의 로컬 작업공간에 있는지 확인하십시오. 애플리케이션 프로젝트와 애플리케이션 바인딩 프로젝트의 유효성을 검증하려면 CICS Explorer에 플랫폼 프로젝트가 필요합니다.

프로시저

1. 각 애플리케이션 구성요소에 대해 CICS 번들 프로젝트를 작성하고, CICS 번들에 적절한 자원을 정의하며 구성요소가 필요로 하는 다른 자원에 대한 종속성을 선언하십시오. CICS 번들은 자원을 함께 그룹화하고 단일 엔티티로 버전 제어되고 관리되므로 동일한 CICS 번들에 개별적으로 갱신 및 관리해야 하는 자원을 배치하지 마십시오. CICS 번들에 정의할 수 있는 자원, CICS 번들에 자원을 정의하고 관리하는 경우 자원의 작동 및 특성 변경에 대한 정보는 CICS 번들 정의를 참조하십시오.
2. CICS 번들 프로젝트에서 애플리케이션에 액세스하기 위한 해당 태스크 호출에 PROGRAM 및 TRANSACTION 자원에 대한 애플리케이션 시작점을 추가하거나 요청자가 애플리케이션에 액세스할 수 있도록 URI를 제공하는 URIMAP 자원을 추가하십시오. 애플리케이션 시작점으로 선언된 자원은 환경의 자원 유형에 고유 이름이 있어야 합니다. 애플리케이션 시작점에 대한 자세한 정보는 제품 개요의 애플리케이션 시작점의 내용을 참조하십시오.
3. CICS 애플리케이션의 이름과 버전을 지정하는 CICS 애플리케이션 프로젝트를 작성하십시오. CICS 번들에 대한 참조를 추가하여 애플리케이션 번들에 포함시키십시오. 애플리케이션 번들을 작성한 후에는 이를 편집하여 CICS 번들을 추가하거나 제거할 수 있습니다.
4. CICS 애플리케이션 바인딩 프로젝트를 작성하여 애플리케이션에 대한 각 CICS 번들이 대상 플랫폼의 CICS 리전 유형에 전개되는 방법을 정의하십시오. 애플리케이션

션 바인딩을 작성한 후에는 이를 편집하여 CICS 번들 전개 방법을 변경할 수 있습니다. 애플리케이션 바인딩 및 예제에 대한 자세한 정보는 『애플리케이션을 플랫폼으로 바인드하는 방법』의 내용을 참조하십시오.

5. 애플리케이션 번들과 애플리케이션 바인딩을 애플리케이션을 실행할 플랫폼에 대해 z/OS UNIX의 플랫폼 홈 디렉토리로 내보내고, 애플리케이션 정의를 작성하여 애플리케이션의 위치를 식별하십시오. 애플리케이션 정의를 내보내기 프로세스의 일부로 작성할 수 있습니다. 애플리케이션을 내보내고 설치할 준비가 되지 않았으면 플랫폼에 나중에 전개할 수 있도록 소스 저장소에 애플리케이션을 저장하고 나중에 애플리케이션 정의를 작성할 수 있습니다.

결과

애플리케이션 번들, 애플리케이션 바인딩, 연관된 CICS 번들을 대상 플랫폼의 홈 디렉토리에서 사용할 수 있으며 플랫폼의 CICS 리전에 설치할 준비가 되었습니다.

다음에 수행할 작업

애플리케이션 정의를 사용하여 CICS에 애플리케이션을 설치한 후 애플리케이션을 사용할 수 있게 하십시오. 애플리케이션을 사용 가능하게 하면 호출자가 애플리케이션 시작점 및 시작점이 애플리케이션에 대해 제어하는 자원을 사용할 수 있게 됩니다. 세부 사항은 132 페이지의 『플랫폼에 애플리케이션 설치』의 내용을 참조하십시오.

애플리케이션을 플랫폼으로 바인드하는 방법

애플리케이션을 전개하려면 애플리케이션을 대상 플랫폼으로 바인드하거나 맵핑해야 합니다. 애플리케이션의 각 CICS 번들을 플랫폼의 하나 이상의 CICS 리전 유형으로 바인드해야 합니다. 애플리케이션 바인딩은 CICS 번들이 대상 플랫폼에 제대로 전개되도록 합니다.

CICS 탐색기 애플리케이션 바인딩 프로젝트 마법사로 애플리케이션 바인딩을 작성합니다. 마법사는 관리 번들 유형인 애플리케이션 바인딩 번들을 출력으로 생성합니다. 마법사는 바인드 디렉토리의 세부사항으로 애플리케이션 정의를 갱신하기도 합니다. 마법사를 사용한 애플리케이션 바인딩 작성 및 전개에 대한 지시사항은 CICS Explorer 제품 문서의 CICS 애플리케이션 바인딩 프로젝트 작성의 내용을 참조하십시오.

단순 CICS 토폴로지는 세 개의 CICS 번들을 단일 CICS 리전에 설치해야 할 수 있습니다. 보다 정교한 토폴로지의 경우에는 하나의 CICS 번들을 플랫폼의 한 리전 유형에 전개하고 다른 두 개의 CICS 번들을 모두 플랫폼의 각기 다른 리전 유형에 전개해야 할 수 있습니다. 애플리케이션 번들이 아닌 애플리케이션 바인딩이 CICS 번들의 전개를 제어하므로 애플리케이션 자체에 전개 정보를 포함할 필요는 없습니다. CICS 번들을 리전 유형으로 맵핑하는 애플리케이션 바인딩은 애플리케이션 번들 설치가 필요합니다.

애플리케이션 바인딩은 여러 다른 CICS 자원이 상이한 플랫폼과 리전 유형에 전개되는 보다 복잡한 시나리오도 지원합니다. 예를 들어, 다른 URI를 포함한 URIMAP 자원을 상이한 플랫폼과 리전 유형에 전개할 수 있습니다. 이 유형의 사용자 정의는 추가 CICS 번들을 애플리케이션 바인딩에 포함시켜서 수행할 수 있습니다. 각 번들(이 경우, 다른 URIMAP 자원을 포함한)이 애플리케이션 바인딩에 지정되지만 애플리케이션 자체는 수정이 필요하지 않습니다. 이 유형의 애플리케이션 바인딩은 선택적입니다.

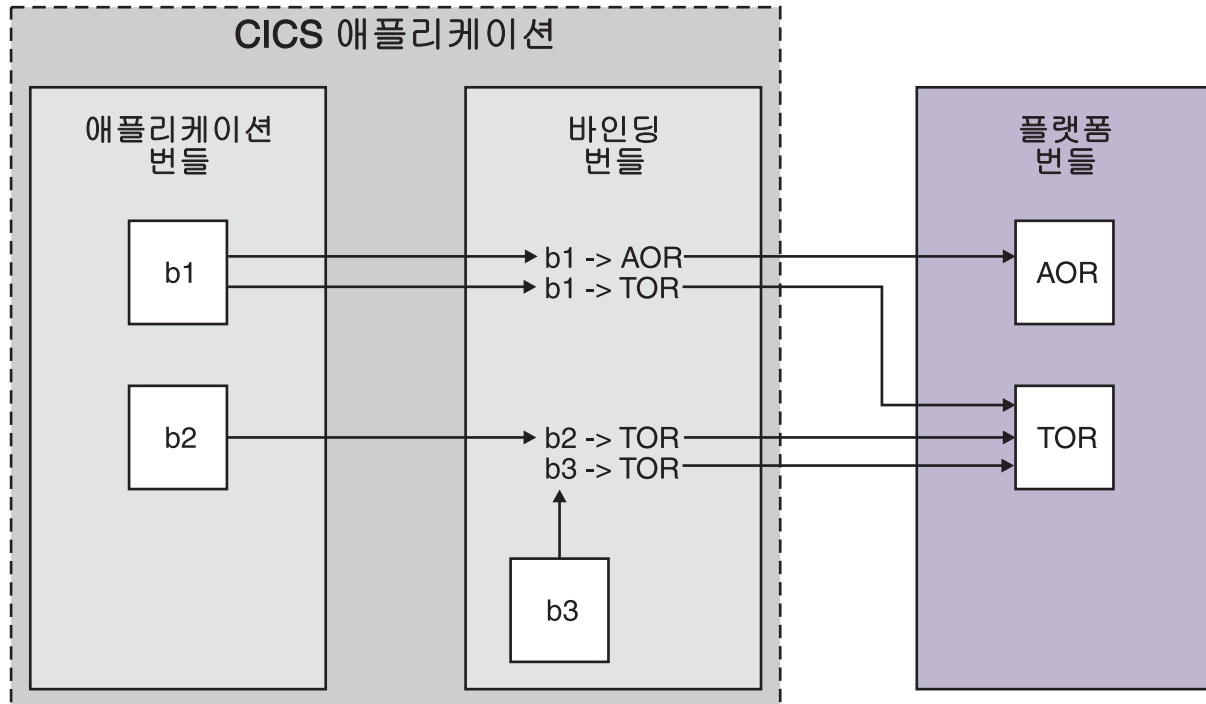


그림 117. 플랫폼으로의 애플리케이션 바인딩 예제

- 애플리케이션, 바인딩, 플랫폼 구성요소는 CICS 탐색기에 각각 자체 프로젝트가 있는 다른 유형의 관리 번들입니다.
- **b1, b2, b3** 구성요소는 CICS 번들입니다.
- **APPLDEF** 구성요소는 CICSplex SM의 애플리케이션 정의입니다. APPLDEF의 애플리케이션 디렉토리는 UNIX 파일 시스템의 애플리케이션 위치를 표시합니다. APPLDEF 바인딩 디렉토리는 UNIX 파일 시스템의 바인딩의 위치를 표시합니다. 애플리케이션 정의가 설치될 때 CICSplex SM에서 두 관리 번들이 모두 처리됩니다.

CICS Explorer 애플리케이션 바인딩 프로젝트 마법사를 사용하여 애플리케이션 바인딩을 작성할 때에는 다음 XML 파일이 출력으로 생성됩니다.

- **bundles.xml**은 이 애플리케이션 바인딩으로 전개할 CICS 번들을 나열합니다.
- **appbinding.xml**은 애플리케이션 바인딩을 특정 애플리케이션 및 특정 플랫폼과 연관시킵니다.

- `deployment.xml`은 애플리케이션과 애플리케이션 바인딩 모두에서 CICS 번들을 나열하고 플랫폼의 리전 유형과 연관시킵니다.

애플리케이션 바인딩을 작성하고 난 후 CICS 탐색기에서 애플리케이션 바인딩 편집기를 사용하여 애플리케이션 및 애플리케이션 바인딩 모두에서 CICS 번들이 플랫폼의 리전 유형으로 전개되는 방식을 변경할 수 있습니다. 애플리케이션 바인딩으로 전개되는 CICS 번들을 추가 또는 제거할 수도 있습니다.

애플리케이션의 CICS 번들을 변경할 때에는 애플리케이션 바인딩 편집기를 사용하여 애플리케이션의 새 버전을 지정하도록 애플리케이션 바인딩을 변경하고, 애플리케이션 바인딩의 버전 번호를 애플리케이션 버전에 일치하게 갱신하기도 합니다. 기존 버전의 애플리케이션 바인딩을 새 버전의 애플리케이션 번들과 함께 사용할 수는 없습니다. 애플리케이션의 CICS 번들을 갱신할 때마다 애플리케이션 번들과 애플리케이션 바인딩의 버전을 갱신해야 합니다.

제 7 부 파일 제어

CICS 파일 제어 기능 이해 및 작동.

제 23 장 파일 제어 이해

CICS 데이터 관리 서비스는 전통적으로 CICS 파일 제어로 알려져 있습니다. CICS 파일 제어는 VSAM(Virtual Storage Access Method) 또는 기본 직접 액세스 메소드(BDAM) 중 하나가 관리하는 데이터 세트에 대한 액세스 권한을 제공합니다.

CICS 파일 제어를 사용하면 VSAM 및 BDAM 데이터 세트의 데이터를 읽기, 갱신, 추가 및 찾아보고 VSAM 데이터 세트에서 데이터를 삭제할 수 있습니다. 또한 파일 제어를 사용하여 CICS 공유 데이터 테이블 및 결합 기능 데이터 테이블에 액세스할 수 있습니다.

CICS 애플리케이션은 개별 레코드의 양식으로 데이터를 읽고 씁니다. 각 읽기 또는 쓰기 요청은 CICS 명령에 의해 이루어집니다.

레코드에 액세스하기 위해 애플리케이션은 레코드 및 레코드를 보유하는 데이터 세트를 둘 다 식별해야 합니다. 또한 레코드를 읽어들이거나 레코드를 쓸 저장영역도 지정해야 합니다.

VSAM 데이터 세트: KSDS, ESDS, RRDS

CICS는 KSDS(KEY-SEQUENCED DATA SET), ESDS(Entry-Sequenced Data Set), RRDS(Relative Record Data Set)(고정 및 변수 레코드 길이 둘 다)의 데이터 세트 유형에 대한 액세스를 지원합니다.

VSAM 데이터 세트는 직접 액세스 저장영역 디바이스(DASD) 보조 저장영역에 보유됩니다. VSAM은 데이터 세트 저장영역을 제어 영역(CA)으로 분할하는데, CA는 제어 간격(CI)으로 다시 분할됩니다. CI는 가상 및 보조 저장영역 사이의 데이터 전송 단위입니다. 각 CI는 고정 크기이며, 일반적으로 많은 레코드를 포함합니다. KSDS 또는 ESDS는 둘 이상의 CI에 확장하는 레코드를 가질 수 있습니다. 이들을 스펠 레코드라고 부릅니다.

KSDS(Key-Sequenced Data Set)

KEY-SEQUENCED DATA SET는 키에 의해 식별되는 각 레코드를 갖습니다. (각 레코드의 키는 레코드 내에서 사전 정의된 위치에 있는 필드입니다.) 각 키는 데이터 세트에서 고유해야 합니다.

데이터 세트가 초기에 데이터와 함께 로드될 때나 새 레코드가 추가될 때, 레코드의 논리적 순서는 키 필드의 조합 순서에 따라 다릅니다. 이것은 또한 데이터 세트를 찾아볼 때 레코드를 검색하는 순서를 고정합니다.

KSDS에서 레코드의 물리적 위치를 찾기 위해, VSAM은 색인을 작성하고 유지보수합니다. 이것은 각 레코드의 키를 데이터 세트에서 레코드의 상대 위치와 관련시킵니다. 레코드를 추가 또는 삭제할 때 이 색인이 그에 따라서 갱신됩니다.

DFSMS/MVS 1.4 이상의 릴리스에서, 데이터 세트는 저장영역 클래스에서 확장 형식 및 확장 주소 지정 가능성으로 정의될 때 크기가 4GB보다 클 수 있습니다. CICS는 RL 및 비RLS 모드 모두에서 이러한 확장 속성을 사용하여 정의되는 KSDS 데이터 세트를 지원합니다.

ESDS(Entry-Sequenced Data Set)

ESDS(Entry-Sequenced Data Set)는 각 레코드가 상대 바이트 주소(RBA)에 의해 식별되는 데이터 세트입니다.

레코드는 처음 데이터 세트에 로드된 순서로 ESDS에 보유됩니다. ESDS에 추가되는 새 레코드는 항상 데이터 세트의 마지막 레코드 뒤로 갑니다. 레코드를 삭제하거나 그 길이를 변경할 수 없습니다. 레코드가 ESDS에 저장된 후, 해당 RBA는 상수로 남습니다. 찾아볼 때 레코드는 데이터 세트에 추가된 순서대로 검색됩니다.

표준 RBA는 부호 없는 32비트 숫자입니다. 32비트 RBA의 사용은 표준 ESDS가 4GB 이상의 데이터를 포함할 수 없음을 의미합니다. 그러나 64비트 확장 상대 바이트 주소(XRBAs)를 지원하고 따라서 4GB 한계로 제한되지 않는 다른 ESDS가 있습니다. 이 유형의 ESDS를 『확장 형식, 확장 주소 지정 ESDS 데이터 세트』라고 합니다. 줄여서, 이를 『확장 주소 지정 ESDS』 또는 『확장 ESDS』라고 부릅니다. z/OS용 CICS TS, 버전 3.2에서부터, CICS는 64비트 XRBA 및 확장 ESDS 데이터 세트를 지원합니다.

RRDS(Relative Record Data Set)

상대 레코드 데이터 세트는 상대 레코드 번호(RRN)에 의해 식별되는 레코드를 갖습니다. 데이터 세트의 첫 번째 레코드는 RRN 1이고, 두 번째는 RRN 2, 등등입니다.

RRDS의 레코드는 고정 또는 가변 길이 레코드일 수 있으며, VSAM이 데이터를 처리하는 방법은 데이터 세트가 고정 또는 가변 RRDS인지 여부에 따라 다릅니다. 고정 RRDS는 VSAM에 사전 정의된 고정 길이 슬롯이며, 여기에 레코드가 저장됩니다. 고정 RRDS의 레코드 길이는 항상 슬롯의 크기와 동일합니다. VSAM은 데이터 세트의 시작부터 바이트 오프셋을 계산하기 위해 슬롯 크기에 RRN(파일 제어 요청에서 제공함)에 곱하여 고정 RRDS에서 레코드를 찾습니다.

가변 RRDS는 데이터 세트에 대한 최대값까지의 모든 길이의 레코드를 승인할 수 있습니다. 가변 RRDS에서 VSAM은 색인을 갖는 레코드를 찾습니다.

고정 RRDS는 일반적으로 더 좋은 성능을 제공합니다. 가변 RRDS는 더 큰 기능을 제공합니다.

DFSMS/MVS 1.5 이상의 릴리스에서, 데이터 세트는 저장영역 클래스에서 확장 형식 및 확장 주소 지정 가능성으로 정의될 때 크기가 4GB보다 클 수 있습니다. 4GB 경계를 넘어서 상주하는 레코드에 액세스하기 위해 4바이트 RRN 필드에서 지정될 수 있는 RRN을 사용하는 경우 CICS는 확장 RRDS 또는 VRRDS 데이터 세트에 대한 액세스를 지원합니다.

비어 있는 데이터 세트

비어 있는 데이터 세트는 아직 기록된 어떤 레코드도 없는 데이터 세트입니다. VSAM은 비RLS 액세스 모드에서 열린 비어 있는 데이터 세트에 대해 여러 가지 제한사항을 부과합니다. 그러나, CICS는 사용자에게 이러한 모든 제한사항을 숨겨 액세스 모드와 상관 없이 데이터를 포함하는 데이터 세트와 동일한 방법으로 빈 데이터 세트를 사용할 수 있습니다.

VSAM 대체 색인(alternate index)

가끔 다른 방법으로 동일한 레코드 세트에 액세스하기 원합니다. 예를 들어 직원 번호를 키로 갖는 개인용 데이터 세트에 레코드를 가질 수 있습니다. Smiths가 몇 명이든 간에, 각 Smiths는 고유한 직원 번호를 갖습니다. 이것을 기본 키로 생각하십시오.

데이터 세트로부터 전화 디렉토리를 생성 중인 경우, 전화번호가 아닌 이름으로 사람들을 나열하기 원할 수 있습니다. 이전에 설명되는 기본 키 대신 2차(대체) 키로 데이터 세트의 레코드를 식별할 수 있습니다. 그러므로 기본 키는 직원 번호이고, 직원 이름은 대체 키입니다. 대체 키는 레코드 내의 고정 길이 및 고정 위치의 KSDS 필드의 기본 키와 같습니다. 기본 파일당 임의 숫자의 대체 키를 가질 수 있으며, 기본 또는 1차 키와는 달리 대체 키는 고유할 필요가 없습니다.

개인 예제를 계속하기 위해, 직원의 부서 코드가 추가 대체 키로 정의될 수 있습니다.

VSAM은 KSDS 및 ESDS(그러나 RRDS 또는 확장 ESDS가 아님) 데이터 세트가 대체 키를 가질 수 있게 합니다. 데이터 세트가 작성될 때, 하나의 2차 또는 대체 색인이 레코드의 각 대체 키에 대해 빌드되고 기본 또는 1차 키와 관련됩니다. 대체 키를 사용하여 레코드에 액세스하려면 추가 VSAM 오브젝트인 대체 색인 경로를 정의해야 합니다. 그런 다음 경로는 레코드가 대체 키를 사용하여 액세스되는 KSDS인 것처럼 작동합니다.

경로를 통해서 레코드를 갱신할 때, 대응하는 대체 색인이 변경을 반영하도록 갱신됩니다. 그러나 기본을 통해 또는 다른 경로를 통해 레코드를 직접 갱신하는 경우, 대체 색인은 기본 데이터 세트의 업그레이드 세트에 속하기 위해 VSAM(작성될 때)에 정의된 경우에만 갱신됩니다. 대부분의 애플리케이션의 경우 대체 색인이 업그레이드 세트에 있기 원할 수 있습니다.

CICS 애플리케이션은 액세스 중인 파일이 경로인지 또는 기본인지 여부를 무시합니다. 실행 중인 CICS 시스템에서, 각 액세스 경로가 CICS에 정의되는 경우 기본 및 정의된 모든 경로를 통해 단일 기본 데이터 세트에 액세스할 수 있습니다.

또한 CICS 애플리케이션 프로그램이 경로가 아니라 대체 색인으로 직접 정의된 파일에 액세스할 수 있습니다. 이 때문에 파일 데이터가 아닌 색인 데이터가 애플리케이션으로 리턴됩니다. 이 조작은 레코드 레벨 공유(RLS) 모드에서 열린 파일에는 지원되지 않습니다.

RLS 모드에서 파일에 액세스

레코드 레벨 공유(RLS)는 DFSMS 버전 1 릴리스 3 및 이후 릴리스에서 제공되며, 많은 CICS 리전에서 실행 중인 많은 애플리케이션 사이에 전체 갱신 기능을 사용하여 VSAM 데이터를 공유할 수 있도록 하는 VSAM 기능입니다.

RLS에서, VSAM 데이터 세트를 공유하는 CICS 리전은 MVS 병렬 sysplex 내에서 하나 이상의 MVS 이미지에 상주할 수 있습니다. 데이터 세트가 CICS 리전 및 일괄 처리 작업 사이에 공유될 때 RLS는 이점을 제공합니다.

RLS 모드에서 파일을 여는 경우 제어 간격 레벨 대신 레코드 레벨에서 잠금이 발생하므로, 교착상태 위험이 감소합니다.

CICS는 다음 유형의 VSAM 데이터 세트에 대한 레코드 레벨 공유(RLS) 액세스를 지원합니다.

- 키 순서 지정 데이터 세트(KSDS). KSDS를 사용 중인 경우, 파일에 액세스하기 위해 상대 바이트 주소(RBA)를 사용할 수 없습니다.
- 항목 순서 지정 데이터 세트(ESDS). 항목 순서 지정 데이터 세트(ESDS)에 대해 RLS 액세스 모드가 허용되어도, 이 모드는 권장되지 않습니다. 레코드를 추가할 때 데이터 세트의 가용성과 성능에 부정적인 영향을 줄 수 있기 때문입니다. (CICS 성능 안내서에 있는 VSAM 레코드 레벨 공유의 성능 측면을 참조하십시오.)
- 고정 및 가변 길이 레코드 둘 다에 대한 RRDS(Relative Record Data Set).

참고: RLS 모드를 지정하는 파일에 대해 SET FILE EMPTY 명령을 실행하는 경우, 요청은 승인되지만 파일이 RLS 모드에서 열릴 때마다 항상 무시됩니다. 파일을 닫고 RLS 이외 모드로 전환하는 경우, 데이터 세트는 비어 있도록 재설정됩니다(해당되는 IDCAMS 정의에서 재사용 가능하도록 정의된 경우).

대부분의 데이터 세트 유형은 VSAM 레코드 레벨 공유에 참여할 수 있으며 대부분의 CICS 애플리케이션은 이 액세스 모드에서 이점을 얻을 수 있습니다. 그러나, 일부 애플리케이션에 영향을 줄 수 있는 제한사항이 있습니다. 다음 유형의 파일, 데이터 세트 또는 액세스 방법은 RLS 모드에서 지원되지 않습니다.

- KSDS에 대한 RBA 액세스

- 키 범위 데이터 세트
- 임시 데이터 세트
- IMBED 속성이 있는 VSAM 클러스터
- 대체 색인의 직접 열기
- 클러스터의 개별 구성요소 열기
- 카탈로그 또는 VVDS 데이터 세트에 대한 액세스
- CICS에서 유지보수되는 데이터 테이블
- Hiperbatch

VSAM 레코드 식별

키로, 상대 바이트 주소(RBA) 또는 확장된 상대 바이트 주소(XRBA)로, 또는 상대 레코드 번호(RRN)로 데이터 세트에서 레코드를 식별할 수 있습니다.

CICS 파일 제어 명령의 RIDFLD(레코드 식별 필드) 옵션은 액세스하는 파일의 유형과 액세스 방법에 적절한 레코드 ID를 포함하는 필드를 식별합니다. 레코드에 할 수 있는 대부분의 작업(읽기, 추가, 삭제 또는 찾아보기 시작)의 경우 갱신할 레코드를 처음 읽을 때를 제외하고는 RIDFLD 옵션을 지정하여 레코드를 식별합니다. 그러나 ENDBR, REWRITE 및 UNLOCK 명령에 대해 RIDFLD가 전혀 없습니다.

- 『키』
- 678 페이지의 『상대 레코드 번호(RRN), 상대 바이트 주소(RBA) 및 확장 상대 바이트 주소(XRBA)』

키

키를 사용하는 경우 전체 키 또는 일반(부분) 키를 지정할 수 있습니다. 그러나 레코드를 KSDS에 쓰거나 대체 색인 경로에 의해 레코드를 쓰는 경우, 명령의 RIDFLD 옵션에 전체 키를 지정해야 합니다.

일반 키를 사용할 때 KEYLENGTH 옵션에서 해당 길이를 지정하고 명령에 GENERIC 옵션을 지정해야 합니다. 일반 키는 전체 키 길이와 동일한 키 길이를 가질 수 없습니다. 전체 키보다 더 짧게 정의해야 합니다.

또한 특정 명령에서는 전체 및 일반 키 모두에 대해 GTEQ 옵션을 지정할 수도 있습니다. 그러면 명령은 일치하는 키를 찾을 수 없는 경우 다음으로 높은 키를 갖는 레코드를 찾거나 해당 레코드에 적용됩니다. 대체 색인 경로를 통해서 데이터 세트에 액세스할 때, 식별되는 레코드는 일치하는 레코드를 찾을 수 없을 때 다음으로 높은 대체 키를 갖는 레코드입니다.

일반 키를 사용할 때도 항상 길이에서 전체 키의 길이와 같은 레코드 식별 필드에 대해 저장영역을 사용하십시오. 찾아보기 조작 중에, 레코드를 검색한 후 CICS는 레코드

식별 영역을 검색된 레코드의 실제 ID에 복사합니다. CICS는 사용자가 명령에서 일반 키를 지정했을 때도 애플리케이션으로 전체 키를 리턴합니다. 예를 들어, KSDS를 통한 일반 찾아보기는 각 READNEXT 및 READPREV 명령에서 전체 키를 애플리케이션으로 리턴합니다.

상대 레코드 번호(RRN), 상대 바이트 주소(RBA) 및 확장 상대 바이트 주소(XRBA)

데이터 세트에 액세스하는 대부분의 명령에서 RRN, RBA, XRBA 옵션을 사용할 수 있습니다. 사실상, 레코드 식별 필드(RIDFLD)의 형식을 정의합니다.

RRN, RBA 또는 XRBA를 지정하지 않는 한, RIDFLD 옵션이 KSDS(또는 대체 색인을 통해 KSDS 또는 ESDS)에 사용될 키를 보유해야 합니다.

RRN

RRN은 레코드 식별 필드가 액세스될 레코드의 상대 레코드 번호를 포함하도록 지정합니다. 데이터 세트의 첫 번째 레코드가 번호 1입니다. RRDS를 참조하고 RIDFLD 옵션을 지정하는 모든 파일 제어 명령도 RRN 옵션을 지정해야 합니다.

RBA와 XRBA

RBA는 레코드 식별 필드가 액세스될 레코드의 상대 바이트 주소를 포함하도록 지정합니다. 상대 바이트 주소는 ESDS를 액세스하는 데 사용되며, RLS 액세스 모드에서 열리지 않은 KSDS에 액세스하는 데도 사용될 수 있습니다. ESDS 기본을 참조하고 RIDFLD 옵션을 지정하는 모든 파일 제어 명령도 RBA 옵션을 지정해야 합니다.

참고: KSDS가 이 방식으로 액세스되는 경우, 레코드의 RBA는 동일한 데이터 세트에 레코드를 추가하거나 레코드를 삭제하는 다른 트랜잭션의 결과로서 트랜잭션 중에 변할 수 있습니다.

RBA는 부호 없는 32비트 숫자입니다. 32비트 RBA의 사용은 표준 ESDS가 4GB 이상의 데이터를 포함할 수 없음을 의미합니다. 그러나 64비트 확장 상대 바이트 주소(XRBA)를 지원하고 따라서 4GB 한계로 제한되지 않는 다른 ESDS가 있습니다. 이 유형의 ESDS를 확장 ESDS라고 부릅니다. z/OS용 CICS TS, 버전 3.2에서부터, CICS는 64비트 XRBA 및 확장 ESDS 데이터 세트를 지원합니다.

일반적으로 확장 ESDS에 액세스하기 위해 프로그램은 64비트 XRBA를 지원합니다. 관련 명령의 RBA 키워드를 XRBA 키워드로 대체하고 키에 사용되는 영역의 길이를 변경하여 32비트 RBA를 사용하는 기존 프로그램을 64비트 XRBA를 사용하도록 변환할 수 있습니다. 또한 일부 상황에서는 32비트 RBA를 전달하지만 사용하지는 않는 경우 확장 ESDS에 액세스하기 위한 변경 없이 이전 프로그램을 재사용할 수 있습니다. 679 페이지의 『ESDS에 대한 확장된 주소 지정으로 업그레이드』는 표준 ESDS를 확장 주소 지정 ESDS로 업그레이드하는 방법 및 기존 프로그램을 새로운 형식과 함께

업그레이드 또는 재사용하는 방법을 설명합니다.

관련 정보:

『ESDS에 대한 확장된 주소 지정으로 업그레이드』

ESDS(Entry-Sequenced Data Set)를 사용하려면 데이터 세트를 업그레이드하고 32비트 상대 바이트 주소 지정(RBA)을 사용하는 기존 CICS 애플리케이션을 64비트 확장 상대 바이트 주소 지정(XRBA)으로 변환해야 합니다.

ESDS에 대한 확장된 주소 지정으로 업그레이드

ESDS(Entry-Sequenced Data Set)를 사용하려면 데이터 세트를 업그레이드하고 32비트 상대 바이트 주소 지정(RBA)을 사용하는 기존 CICS 애플리케이션을 64비트 확장 상대 바이트 주소 지정(XRBA)으로 변환해야 합니다.

표준 ESDS를 확장된 주소 지정 ESDS로 업그레이드

참고: 확장된 주소 지정을 사용하기 위해 표준 ESDS를 업그레이드하기 전에 데이터 세트가 정방향 복구를 사용하도록 정의되는 경우 정방향 복구 제품을 확장된 주소 지정 ESDS에 대해 작성된 새 로그 레코드를 읽을 수 있는 제품으로 업그레이드해야 합니다. CICS VR을 사용하는 경우 z/OS용 CICS VSAM Recovery V4.2 이상을 사용해야 합니다.

기존 표준 ESDS를 확장된 주소 지정 ESDS로 변환하려면 다음과 같이 데이터 세트를 다시 작성해야 합니다.

1. 계속해서 기존 데이터 세트의 내용을 사용하려면 해당 내용의 사본을 만드십시오. AMS REPRO 함수를 사용하여 이를 수행할 수 있습니다.
2. 기존 데이터 세트를 삭제하십시오.
3. 새 데이터 세트를 작성하십시오. 기존 데이터 세트의 AMS 정의를 기반으로 새 데이터 세트의 AMS를 정의할 수 있습니다. 유일한 필수 변경은 새 데이터 세트 정의의 **DATACLAS** 매개변수가 확장된 형식 및 확장된 주소 지정 둘 다 지정하는 SMS 데이터 클래스의 이름을 지정해야 한다는 것입니다.

SMS 데이터 클래스를 정의하는 방법은 *DFSMS 저장영역 관리* 참조 매뉴얼에 설명되어 있습니다.

4. 필요한 경우 이전에 만든 사본에서 데이터 세트의 내용을 복원하십시오.

프로그램을 32비트 RBA에서 64비트 XRBA로 업그레이드

32비트 RBA를 사용하는 기존 프로그램을 64비트 확장 상대 바이트 주소 지정(XRBA)으로 변환하려면 다음을 수행해야 합니다.

1. 다음 명령 모두에서 RBA 키워드를 XRBA 키워드로 바꾸십시오.
 - EXEC CICS READ

- EXEC CICS READNEXT
 - EXEC CICS READPREV
 - EXEC CICS RESETBR
 - EXEC CICS STARTBR
 - EXEC CICS WRITE
2. 모든 키에 사용된 모든 4바이트 영역을 8바이트 영역으로 바꾸십시오. "RBA"를 "XRBA"로 변경하지만 키 영역의 길이는 변경하지 않는 경우:
- a. STARTBR 및 READ 명령에서 CICS는 4바이트 RBA를 8바이트 XRBA의 상반부로 처리합니다. 대부분의 경우 이때 매우 큰 XRBA 수가 생성됩니다. 프로그램이 즉시 "RBA에 레코드 없음" 응답을 수신하기 때문에 이 오류를 빠르게 추적할 수 있습니다.
 - b. WRITE 명령은 좀 더 민감한 오류를 생성할 수 있습니다. 이 명령은 키 영역 바로 뒤 4바이트를 겹쳐쓰는 8바이트 XRBA를 리턴합니다.

RBA 미감지 프로그램을 사용하여 확장된 ESDS 데이터 세트에 액세스

RBA를 사용하지 않는 기존 32비트 RBA 프로그램을 재사용하여 64비트 확장 ESDS에 액세스할 수 있습니다.

예를 들어, 레코드가 먼저 순차적으로 쓰여지고 이후에 처음부터 순차적으로 검색되는 공통 애플리케이션 유형이 있습니다. RBA는 CICS와 프로그램 사이에서 전달되지만 프로그램은 그러한 RBA를 이용하지 않습니다. 프로그램은 다음 레코드만 읽고 씁니다. 이러한 프로그램을 『RBA 미감지』라고 합니다. 이름 지정된 RBA에서 레코드를 직접 읽거나 갱신하는 프로그램과 같은 기타 프로그램은 『RBA 감지』라고 합니다.

기존 32비트 RBA 미감지 프로그램은 변경 없이 64비트 확장된 ESDS에 액세스할 수 있습니다. RLS 및 비RLS 모드 둘 다 지원됩니다.

32비트 RBA 감지 프로그램은 데이터 세트가 4GB 미만의 데이터를 포함하더라도 64비트 확장 ESDS에 액세스할 수 없습니다.

백레벨 AOR을 z/OS용 CICS TS, 버전 5.3 FOR에 연결

이 시나리오에서 이전 스타일 32비트 RBA 프로그램은 z/OS용 CICS TS, 버전 5.3 파일 소유 리전(FOR)의 파일에 액세스를 시도합니다. 이 액세스는 다음과 같은 경우에 작동합니다.

- FOR의 대상 파일이 기존 ESDS에서 확장된 주소 지정 ESDS로 변환되지 않았습다.
- 대상 파일이 확장된 주소 지정 ESDS로 변환되었지만 프로그램이 RBA 미감지입니다.

대상 파일이 확장된 주소 지정 ESDS로 변환된 경우 AOR에서 실행되는 32비트 RBA 감지 프로그램이 이 파일에 액세스할 수 없습니다. 프로그램에서 ILLOGIC 응답을 수신합니다.

z/OS용 CICS TS, 버전 5.3 AOR을 백레벨 FOR에 연결

이 시나리오에서 새 스타일 64비트 XRBA 프로그램은 백레벨 파일 소유 리전의 파일에 액세스를 시도합니다.

대상 리전은 32비트 RBA만 지원하기 때문에 64비트 XRBA는 이해하지 못합니다. 프로그램에서 ILLOGIC 응답을 수신합니다.

복구 가능한 파일에서 VSAM 레코드 잠금

잠금은 파일이 레코드 레벨 공유(RLS) 모드에서 액세스하는 경우 VSAM에 의해, 그렇지 않은 경우 CICS에 의해 획득됩니다. 잠금은 트랜잭션이 동기점 요청을 발행하거나 종료할 때까지(동기점이 자동으로 수행될 때) 변경을 수행하는 트랜잭션 대신 보유됩니다.

복구 가능한 파일의 레코드가 수정될 때마다 획득된 레코드 잠금의 측면에서의 트랜잭션 교착 상태 방지는 695 페이지의 『트랜잭션 교착상태』에 설명되어 있습니다. VSAM 복구 가능 파일 처리에 대해, 다음에 유의하십시오.

- 수정 또는 삭제에 대해 VSAM 레코드가 확보될 때마다, CICS 파일 제어(또는 RLS의 경우에 VSAM)는 큐에 넣기 인수로서 1차 레코드 ID를 사용하여 ENQUEUE 요청으로 레코드를 잠급니다.

레코드가 경로 방식으로 수정되면, 큐에 넣기는 인수로서 기본 키 또는 기본 RBA를 사용합니다. 따라서 CICS는 한 번에 단 하나의 트랜잭션이 해당 요청을 수행하도록 허용합니다. 다른 트랜잭션은 첫 번째 트랜잭션이 동기점에 도달할 때까지 기다려야 합니다.

- READ UPDATE 및 REWRITE 관련 명령의 경우 READ UPDATE 명령이 실행되는 즉시 레코드 잠금이 획득됩니다.

READ UPDATE 명령이 앞에 있지 않은 DELETE 명령이나, WRITE 명령의 경우 VSAM 명령이 실행될 때 레코드 잠금이 획득됩니다.

WRITE MASSINSERT 명령(일련의 WRITE 명령으로 구성되는)의 경우, 각각의 개별 WRITE 명령이 수행될 때 별도의 레코드 잠금이 획득됩니다. 마찬가지로, DELETE GENERIC 명령의 경우 삭제된 각 레코드는 요청을 발행하는 트랜잭션 대신 별도의 잠금을 획득합니다.

갱신 잠금 및 삭제 잠금(바RLS 모드만)

이전에 언급된 레코드 잠금은 레코드가 갱신(수정)될 때마다 획득되므로 갱신 잠금이라고 합니다. 복구 가능한 KSDS나 KSDS를 통한 복구 가능한 경로에 대해 DELETE, WRITE 또는 WRITE MASSINSERT 명령이 수행될 때마다, 파일 제어에 의해 추가 잠금 유형(삭제 잠금)이 획득될 수도 있습니다. 따라서 삭제 조작은 삭제되는 레코드에 대해 두 개의 개별 잠금을 획득할 수 있습니다.

개별적인 삭제 잠금은 파일 제어가 해당되는 쓰기 조작을 수행하는 방식 때문에 필요합니다. KSDS 또는 RRDS에 대해 WRITE MASSINSERT 명령을 실행하기 전에, 파일 제어는 새 레코드가 이동할 비어 있는 범위를 찾아서 잠급니다. 비어있는 범위는 데이터 세트에서 다음 기존 레코드를 식별하고 해당되는 삭제 잠금을 획득하여 잠깁니다.

비어있는 범위는 레코드를 범위에 추가하여 다른 요청을 동시에 중지하도록 잠깁니다. 또한 비어 있는 범위의 끝을 정의하는 레코드는 추가 조작 동안 제거할 수 없습니다. 다른 트랜잭션이 레코드를 비어 있는 범위에 추가하거나 범위 끝에서 레코드를 삭제하기 위한 요청을 발행하는 경우, 삭제 잠금은 WRITE 또는 WRITE MASSINSERT 명령이 완료될 때까지 트랜잭션이 대기하도록 합니다. 그러나 삭제 잠금으로 보유된 레코드는 다른 CI에 있는 경우 쓰기 조작 동안 다른 트랜잭션에 의해 갱신될 수 있습니다.

갱신 잠금과 달리, 삭제 잠금은 단지 삭제 또는 쓰기 조작, 또는 UNLOCK 명령에 의해 종료된 일련의 WRITE MASSINSERT 명령 순서의 지속 기간 동안만 보유됩니다. 두 개 이상의 비어 있는 범위로 파일에 레코드를 추가하는 WRITE MASSINSERT 명령은 비어 있는 새 범위로 이동하는 것처럼 이전 삭제 잠금을 해제합니다.

CICS 큐에 넣기 메커니즘은 갱신 및 삭제에만 해당되므로, 다음 동기점 이전에는 읽기 요청이 충족되지 않도록 합니다. 이러한 상황에서 READ 명령의 무결성은 보증되지 않습니다.

RLS 레코드 레벨 잠금

RLS 모드에서 열린 파일은 동시에 많은 CICS 리전에서 액세스할 수 있습니다. 이는 개별적인 CICS 리전이 레코드 잠금을 제어하려고 하는 것이 비실용적임을 의미하므로, VSAM은 MVS 결합 기능의 잠금 지원 메커니즘을 사용하여 단일의 중심 잠금 구조를 유지합니다.

이 중심 잠금 구조는 레코드 레벨에서 sysplex 전반의 잠금을 제공합니다. 제어 간격(CI) 잠금은 사용되지 않습니다. 이는 범위가 단일 CICS 리전으로 제한되는 비RLS 모드의 파일에 대한, 그리고 CI 잠금 또는 CICS ENQ인 잠금과 반대입니다.

RLS 내의 레코드 잠금은 이름 지정된 CICS 리전 내에서 이름 지정된 UOW에서 소유됩니다. 잠금 소유자 이름은 CICS 리전의 APPLID에 UOW ID를 추가한 것입니다.

예를 들어, CICS가 잠금을 작성하는 요청을 작성할 때, CICS는 VSAM에 UOW ID를 전달합니다. 이로서 VSAM은 UOW ID, 레코드 키 및 CICS 리전 이름을 사용하여 잠금 이름을 빌드할 수 있습니다.

CICS는 VSAM 인터페이스를 사용하여 UOW 완료 시 모든 잠금을 해제합니다.

둘 이상의 요청에서 동일한 자원에 대한 독점 잠금을 요구하는 경우, 자원이 사용 가능하도록 해제되고 잠금을 부여할 수 있을 때까지 VSAM은 두 번째 및 후속 요청을 큐에 넣습니다. 그러나 VSAM은 보유된 잠금에 의해 잠겨진 자원에 대한 요청을 큐에 넣지 않습니다(685 페이지의 『잠금에 대한 활성화 및 보유 상태』 참조).

참고: RLS 액세스 모드에서 열린 파일에 대한 MASSINSERT 조작의 경우, CICS는 각각의 개별 WRITE 명령이 실행될 때 별도의 갱신 잠금을 획득합니다. 비RLS 모드 운영(681 페이지의 『복구 가능한 파일에서 VSAM 레코드 잠금』 아래에 설명된)과는 달리, CICS는 갱신 잠금 외에 별도의 삭제 잠금을 획득하지 않습니다. 비RLS 모드에는 열린 파일의 MASSINSERT 함수에 대해 잠금을 배열하는 것과 동일한 것이 없습니다.

독점 잠금 및 공유 잠금

VSAM은 RLS 모드에서 액세스되는 파일에 대해 두 유형의 잠금을 지원합니다. 두 유형은 독점 및 공유 잠금입니다.

독점 잠금은 활성화되거나 유지될 수 있습니다. 공유 잠금은 활성화만 가능합니다(685 페이지의 『잠금에 대한 활성화 및 보유 상태』 참조). RLS 모드에는 삭제 잠금이 없습니다.

독점 잠금

독점 잠금은 복구 가능 및 복구 불가능 파일 자원에 대해 갱신을 보호합니다. 한 번에 하나의 트랜잭션이 이 잠금을 소유할 수 있습니다. 독점 잠금을 요구하는 트랜잭션은 다른 태스크가 현재 요청된 자원에 대해 독점 잠금 또는 공유 잠금을 소유하는 경우 기다려야 합니다.

공유 잠금

공유 잠금은 읽기 무결성을 지원합니다. 이 잠금은 레코드가 읽기 전용 요청 동안 갱신되는 프로세스에 없도록 합니다. 공유 잠금은 또한 레코드가 읽혀질 때와 다음 동기점 사이에서 레코드의 갱신을 방지하기 위해 사용할 수도 있습니다.

자원에 대한 공유 잠금은 동시에 여러 태스크에서 소유할 수 있습니다. 그러나 여러 태스크가 공유 잠금을 소유할 수 있더라도, 태스크가 잠금을 기다리도록 강요할 수 있는 상황이 있습니다.

- 공유 잠금에 대한 요청은 다른 태스크가 현재 자원에 대해 독점 잠금을 소유하는 경우 기다려야 합니다.

- 독점 잠금에 대한 요청은 다른 태스크가 현재 해당 자원에 공유 잠금을 소유하는 경우 기다려야 합니다.
- 공유 잠금에 대한 새 요청은 이미 공유 잠금을 보유하는 자원에 대해 다른 태스크가 독점 잠금을 대기 중인 경우 기다려야 합니다.

잠금 기간

반복되는 읽기 요청, 복구 가능 및 복구 불가능한 데이터 세트에 대한 공유 잠금은 다음 동기점까지 유지됩니다.

복구 불가능한 데이터 세트에서 레코드에 대한 독점 잠금은 요청의 기간 동안만 보존됩니다. 즉, 이들은 요청 시작 시에 획득되고 완료 시에 해제됩니다. 예를 들어, WRITE 요청에 의해 획득된 잠금은 WRITE 요청이 완료될 때 해제되고 READ UPDATE 요청에 의해 획득된 잠금은 뒤따르는 REWRITE 또는 DELETE 요청이 완료되는 즉시 해제됩니다. 예외적으로, 순차 요청에 의해 획득된 잠금은 즉각적인 조작의 완료 이후까지 지속될 수 있습니다. 순차 요청은 MASSINSERT 옵션을 지정하고 갱신 요청을 찾아보는 WRITE 명령입니다. MASSINSERT 옵션과 함께 WRITE 명령을 사용하여 획득한 잠금은 항상 해당하는 UNLOCK 명령이 완료될 때 해제되지만 WRITE MASSINSERT 순서에서 이전의 요청에 의해 해제될 수도 있습니다. 잠금을 해제하게 하는 순서의 정확한 요청은 예측할 수 없습니다. 마찬가지로, READNEXT UPDATE 명령에 의해 획득된 잠금은 다음 DELETE 또는 REWRITE 명령이 완료된 후에도 여전히 존재할 수 있습니다. 이 잠금은 후속 ENDBR 명령이 완료될 때 해제되도록 보장할 수 있지만 찾아보기 순서에서 즉각적인 요청을 사용하여 해제될 수도 있습니다.

복구 가능한 데이터 세트를 갱신하기 위해 요청이 작성되면 연관된 독점 잠금은 다음 동기점까지 유지되어야 합니다. 이는 자원은 요청을 확약하거나 백아웃하기 위한 의사결정을 내릴 때까지 자원이 보호된 채로 남아 있게 해줍니다. CICS가 실패하면 VSAM은 CICS가 재시작될 때까지 잠금을 보유합니다.

Task 1	Task 2
CICS: READ(filea) UPDATE KEY(99)	
VSAM: grants exclusive lock - key 99	
	CICS: READ(filea) KEY(99)
	with integrity
	VSAM: Queues request for shared lock
CICS: REWRITE(filea) KEY(99)	
VSAM: holds exclusive lock until syncpoint	
CICS: task completes and takes syncpoint	
VSAM: frees exclusive lock	
	VSAM grants shared lock to task 2

그림 118. 복구 가능한 데이터 세트에서 CICS 태스크 간의 잠금 경합에 대한 설명

잠금에 대한 활성 및 보유 상태

VSAM RLS는 잠금에 대해 **활성** 및 **보유** 상태를 지원합니다. 이 두 가지 잠금 유형 사이의 차이는, 활성 잠금을 가지고 있는 자원에 대한 요청은 자원이 사용 가능하게 될 때까지 큐에 넣어지지만 보유 잠금을 가지고 있는 자원에 대한 요청은 즉시 실패한다는 것입니다.

활성 상태는 독점 및 공유 잠금 둘 다에 적용 가능합니다. 그러나 복구 가능 자원에 대해 독점 잠금만 활성에서 보유로 해당 상태가 변경될 수 있습니다. 이러한 상태의 중요한 특성은 태스크가 잠금에 대해 기다려야 하는지 여부를 상태가 판별하는 것입니다.

- 두 경우를 제외하고, 요청된 자원에 대해 이미 활성 잠금이 있는 경우 잠금에 대한 요청이 대기 상태가 됩니다.
 1. 현재 활성 잠금이 공유 잠금이기도 하고 대기 중인 독점 잠금 요청이 없는 경우 공유 잠금에 대한 요청이 기다리지 않아도 됩니다.
 2. NOSUSPEND를 지정하는 갱신 요청은 활성 잠금이 존재하는 경우 잠금에 대해 기다리지 않습니다. 이러한 경우에, CICS는 『레코드가 사용 중』임을 표시하는 예외 상태를 리턴합니다.
- 요청된 자원에 대해 보유 잠금이 있는 경우 잠금에 대한 요청은 LOCKED 상태로 즉시 거부됩니다.

잠금이 처음 획득될 때, 활성 잠금입니다. 이 잠금은 해제되거나(잠금 유형에 따라 잠금의 지속 기간), UOW가 임시로 실패하도록 해서 비정상적으로 긴 시간 동안 잠금이 보유되도록 하는 이벤트가 발생하는 경우 보유 잠금으로 변환됩니다. 잠금이 보유 상태가 될 수 있도록 하는 이벤트의 유형은 다음과 같습니다.

- CICS 시스템, VSAM 서버 또는 전체 MVS 시스템의 실패
- 백아웃 실패 상태에 들어가는 작업 단위
- 좌표계나 좌표계에 대한 링크 중 하나의 실패로 인해 의심이 가는 분산 작업 단위

UOW가 실패하는 경우, VSAM은 복구 가능한 데이터 세트에 대해 실패한 UOW가 소유한 독점 레코드 잠금을 계속 보유합니다. 새 요청이 실패한 UOW가 소유한 잠금에 대해 기다리지 않도록 하려면, VSAM은 실패한 UOW가 소유한 활성 잠금을 보유 잠금으로 변환합니다. 보유 잠금은 UOW가 완료될 때까지 잠긴 레코드를 위한 데이터 무결성이 유지되도록 합니다.

독점적인 복구 가능한 잠금은 CICS가 재시작되고 복구를 수행할 때까지 데이터 무결성이 유지되도록 CICS 실패의 이벤트에서 보유 잠금으로 변환됩니다.

VSAM 데이터 공유 서버(SMSVSAM)가 실패하는 경우 독점적인 복구 가능 잠금 역시 보유 잠금으로 변환됩니다. (변환은 Sysplex에서 다른 서버에 의해, 또는 모든 서버가 실패한 경우 재시작할 첫 번째 서버에 의해 수행됩니다.) 이는 UOW가 보유된 RLS 잠금을 보유하기 위해 자체가 실패하도록 하지 않음을 의미합니다.

실패한 CICS 리전에 의해 소유된 공유 잠금은 버려지므로, 활성 공유 잠금은 결코 보유될 수 없습니다. 마찬가지로, 활성 상태의 독점적인 복구 가능 잠금은 버려집니다. 독점적이고 복구 가능한 자원에 적용되는 잠금만 보유 상태가 될 수 있습니다.

BDAM 데이터 세트

CICS는 키 지정 및 키 지정되지 않은 BDAM 데이터 세트에 대한 액세스를 지원합니다. BDAM 지원은 DASD 디바이스의 레코드의 물리적 네이처를 사용합니다.

BDAM 데이터 세트는 다음 형식을 갖는 비블록화 레코드로 구성됩니다.

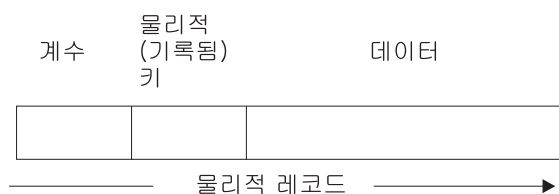


그림 119. BDAM 데이터 세트의 비블록화 레코드의 형식

키 지정 BDAM 파일은 BDAM 레코드를 식별하는 물리적 키를 갖고 있습니다. 계수 영역에는 물리적 키 길이, 물리적 데이터 길이, 레코드의 데이터 위치가 들어 있습니다.

CICS는 BDAM 데이터 세트 위에 추가 구조를 정의할 수 있어서 블록화된 데이터 세트의 개념을 도입합니다.



그림 120. 블록화된 데이터 세트

물리적 레코드의 데이터 부분은 논리 레코드를 포함하는 블록으로 보입니다. CICS는 물리적 레코드의 데이터 파트에서 논리 레코드의 검색을 지원합니다. CICS는 또한 비블록화 레코드(구조가 물리적 레코드당 하나의 논리 레코드의 원래 BDAM 개념으로 돌아감)를 지원합니다.

CICS에서 BDAM 파일의 물리적 레코드에서 데이터를 검색하기 위해, 레코드 식별 필드(RIDFLD)가 물리적 레코드가 검색되어야 하는 방법을 지정하기 위해 정의되어야 합니다. 이것은 물리적 키, 상대 주소 또는 절대 주소에 의해 완료될 수 있습니다.

데이터 세트가 CICS에 블록화되는 것으로 정의되는 경우, 블록 내의 개별 레코드가 키 및 상대 레코드에 의해 두 가지 주소 지정 모드로 검색(블록화 해제)될 수 있습니다.

키에 의한 블록화 해제는 논리 레코드(즉, 논리 레코드에 들어있는 키)를 사용하여 블록에서 필요한 레코드를 식별합니다. 상대 레코드에 의한 블록화 해제는 0에 상대적으로 검색될 레코드의 블록에 있는 레코드 번호를 사용합니다.

CICS BDAM 파일에 액세스할 때 사용되는 RIDFLD 옵션의 서브필드에서 블록화 해제에 사용되는 키 또는 상대 레코드 번호를 지정합니다. CICS BDAM 파일에 대한 주소 지정 모드는 RELTYPE 키워드를 사용하여 FCT에서 설정됩니다.

레코드 식별 및 BDAM 레코드 액세스에 대한 자세한 내용은 673 페이지의 제 23 장 『파일 제어 이해』의 내용을 참조하십시오.

BDAM 레코드 식별

BDAM 데이터 세트에서 블록 참조, 물리적 키(키 지정 데이터 세트) 또는 블록 분리 인수(블록화된 데이터 세트)로 레코드를 식별합니다.

CICS 파일 제어 명령의 RIDFLD(레코드 식별 필드) 옵션은 액세스하는 파일의 유형과 액세스 방법에 적절한 레코드 ID를 포함하는 필드를 식별합니다. 레코드에 할 수 있는 대부분의 작업(읽기, 추가, 삭제 또는 찾아보기 시작)의 경우 갱신할 레코드를 처음 읽을 때를 제외하고는 RIDFLD 옵션을 지정하여 레코드를 식별합니다. (그러나, ENDBR, REWRITE 및 UNLOCK 명령에는 RIDFLD가 없습니다.)

BDAM 레코드의 경우 RIDFLD 옵션에서 레코드 식별에는 블록 참조, 물리적 키 및 블록 참조 인수의 서브필드가 있습니다. 이러한 서브필드가 사용될 때는 이전에 제공된 순서에 있어야 합니다.

참고: EDF를 사용할 때에는 위의 세 필드의 첫 번째(블록 참조 서브필드)만이 표시됩니다.

블록 참조 서브필드

이는 다음 중 하나입니다.

- 상대 블록 주소: 상대 블록 0(RELTYPE=BLK)에서 시작하는 3바이트 2진.
- 상대 트랙 및 레코드(16진 형식): 2바이트 TT, 1바이트 R(RELTYPE=HEX).

2바이트 TT는 상대 트랙 1에서 시작합니다. 1바이트 R은 상대 레코드 1에서 시작합니다.

- 상대 트랙 및 레코드(구역이 있는 10진 형식): 6바이트 TTTTTT, 2바이트 RR(RELTYPE=DEC).
- 실제(절대) 주소: 8바이트 MBBCCHHR(생략된 RELTYPE 피연산자).

시스템 프로그래머는 데이터 세트를 정의하는 DFHFCT TYPE=FILE 시스템 매크로의 RELTYPE 피연산자에서 사용 중인 블록 참조의 유형을 지정해야 합니다.

물리적 키 서브필드

데이터 세트가 레코드된 키를 포함하도록 정의된 경우에만 이것이 필요합니다. 사용되는 경우에는 블록 참조를 곧바로 뒤따라야 합니다. 길이는 데이터 세트를 정의하는 DFHFCT TYPE=FILE 시스템 매크로의 BLKKEYL 피연산자에 지정된 길이와 일치해야 합니다.

블록 해제 인수 서브필드

특정 레코드를 블록에서 검색하고 싶은 경우에만 이것이 필요합니다. 사용되는 경우에는 물리적 키(있는 경우) 또는 블록 참조를 곧바로 뒤따라야 합니다. 이를 생략하면 전체 블록을 검색합니다.

블록 해제 인수는 키이거나 상대 레코드 번호일 수 있습니다. 키이면, READ 또는 STARTBR 명령에 DEBKEY 옵션을 지정하고 길이가 DFHFCT TYPE=FILE 시스템 매크로의 KEYLEN 피연산자에 지정된 것과 일치하는지 확인하십시오. 상대 레코드 번호인 경우에는 READ 또는 STARTBR 명령에 DEBREC 옵션을 지정하십시오. 이는 1바이트 2진수(첫 번째 레코드=0)입니다.

689 페이지의 그림 121에서는 BDAM 레코드 ID의 다음 가능한 양식의 예제를 표시합니다. 예제에서는 4개 바이트의 물리적 키와 3개 바이트의 블록 해제 키를 가정합니다.

- 상대 블록에 의해 검색되고 상대 레코드에 의해 블록 해제하기 위해 상대 레코드 번호가 뒤따르는 상대 블록 번호
- 상대 블록에 의한 검색 및 키에 의한 블록 해제를 위해 키가 뒤따르는 상대 블록 번호
- 상대 트랙과 레코드 및 키에 의한 검색 및 키에 의한 블록 해제를 위해 물리적 키와 블록 해제 키가 뒤따르는 TTR
- 실제 주소에 의한 검색과 상대 레코드에 의한 블록 해제를 위해 상대 레코드 번호가 뒤따르는 MBBCCHHR
- 구역이 있는 10진 상대 트랙 및 레코드와 키에 의한 검색 및 키에 의한 블록 해제를 위해 물리적 키와 블록 해제 키가 뒤따르는 TTTTTRR.
- 상대 트랙과 레코드에 의한 검색 및 키에 의한 블록 해제를 위해 물리적 키가 뒤따르는 TR

바이트 숫자																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15								
RELBLK#			N													상대 블록별 검색; 상대 레코드에 의한 블록 해제							
RELBLK#			KEY													상대 블록별 검색; 키에 의한 블록 해제							
T			T		R		PH-KEY				KEY						상대 트랙별 검색 및 레코드 및 키; 키에 의한 블록 해제						
M			B		B		C		C		H		H		R		N		실제 주소별 검색; 상대 레코드에 의한 블록 해제				
T			T		T		T		T		T		R		R		PH-KEY			KEY		구역 10진수별 검색 상대 트랙 및 레코드 및 키; 키에 의한 블록 해제	
T			T		R		KEY										상대 트랙별 검색 및 레코드; 키에 의해 블록 해제						

그림 121. BDAM 레코드 식별의 예제

CICS 공유 데이터 테이블

CICS 파일 제어 명령은 공유 데이터 테이블에 액세스할 수 있습니다. 공유 데이터 테이블은 16MB 라인 이상의 가상 저장영역에서 보유되는 테이블에 들어있는 데이터 레코드에 대한 빠른 액세스 구성, 유지보수 및 얻기 방법을 제공합니다. 각 공유 데이터 테이블은 소스 데이터 세트로 알려진 VSAM KSDS와 연관됩니다.

공유 데이터 테이블에 대한 자세한 정보는 공유 데이터 테이블의 공유 데이터 테이블의 내용을 참조하십시오.

테이블은 FILE 자원을 사용하여 정의됩니다. 테이블이 열릴 때, CICS는 테이블의 대응하는 소스 데이터 세트에서 데이터를 추출하고 이를 16MB 라인 이상의 가상 저장 영역에 로드하여 테이블을 빌드합니다.

CICS는 다음과 같은 두 유형의 공유 데이터 테이블을 지원합니다.

CICS 유지보수 테이블(CMT)

이 유형의 데이터 테이블은 CICS에 의해 소스 데이터 세트와의 동기화 상태를 유지합니다. 데이터 테이블에 대한 모든 변경사항은 소스 데이터 세트에서 반영됩니다. 비슷하게 소스 데이터 세트에 대한 모든 변경사항은 데이터 테이블에 반영됩니다.

CICS 유지보수 데이터 테이블의 소스는 RLS 액세스 모드에서 열린 파일일 수 없습니다.

사용자 유지보수 테이블(UMT)

이 유형의 데이터 테이블은 로드된 후에 해당 소스 데이터 세트에서 분리됩니다. 테이블에 대한 변경사항은 자동으로 소스 데이터 세트에 반영되지 않습니다.

VSAM KSDS 데이터 세트에 적합한 전체 파일 제어 API가 CICS 유지보수 데이터 테이블에 지원됩니다. 데이터 테이블에 대한 참조가 충족할 수 없는 요청은 소스 데이터 세트에 액세스하기 위한 VSAM에 대한 호출이 됩니다. 복구 가능하도록 정의된 테이블은 전체 무결성을 갖고 지원됩니다.

파일 제어 API의 서브세트는 사용자 유지보수 테이블에 대해 지원됩니다. 복구 가능으로 정의된 테이블은 DTB(Dynamic Transaction Backout)에 참여하지만 재시작 시에 복구되지 않습니다.

결합 기능 데이터 테이블

CICS 파일 제어 명령은 결합 기능 데이터 테이블(CFDTs)에 액세스할 수 있습니다. 결합 기능 데이터 테이블은 파일 소유 리전과 VSAM RLS 지원이 필요없이 파일 데이터 공유 메소드를 제공합니다.

CICS 결합 기능 데이터 테이블 지원은 갱신 무결성으로 시스플렉스 전체에 걸쳐 작동하는 데이터의 빠른 공유를 제공하도록 설계되었습니다. 데이터는 결합 기능에서 여러 면에서 공유된 사용자 유지보수 데이터 테이블과 유사한 테이블에 보관됩니다. 결합 기능 데이터 테이블은 VSAM 소스 데이터 세트의 초기 로드가 선택사항이라는 중요한 점에서 UMT와는 다릅니다. LOAD(NO)를 지정하고 데이터를 사용자 애플리케이션에서 직접 작성하여 테이블을 로드할 수 있습니다. 데이터를 저장하고 검색하는 데 사용되는 API는 사용자 유지보수된 데이터 테이블에 사용되는 파일 제어 API를 기반으로 합니다. CFDT에 대한 읽기 액세스와 쓰기 액세스는 유사한 성능이 있으며 이 테이블 양식을 비공식 공유 데이터에 유용하게 만들어줍니다. 비공식 공유 데이터는 다음과 같은 특징이 있습니다.

- 자연적으로 상대적으로 단기간인 데이터(애플리케이션이 실행 중일 때 작성되거나 처음에 외부 소스로부터 로드됨)
- 일반적으로 크지 않는 데이터 양
- 빨리 액세스될 필요가 있는 데이터
- 사용자 애플리케이션이 이에 대한 부수적 손실을 용인할 수 있는 데이터
- 일반적으로 갱신 무결성을 요구하는 데이터.

일반적인 용도는 시스플렉스 전체에서 CICS 리전 사이에 스크래치패드 데이터 공유 또는 변경사항이 영구적으로 저장될 필요가 없는 파일 공유를 포함할 것입니다. 애플리케이션이 비공식 공유 데이터를 사용하는 데에는 여러 가지 방법이 있고, 이들 대부분은 결합 기능 데이터 테이블을 사용하여 구현됩니다. 결합 기능 데이터 테이블은 데이터를

서로 다른 테이블로 그룹화하는 데 유용하며 여기서 항목을 키를 사용하여 식별하고 검색할 수 있습니다. 예를 들어, 주문 처리 애플리케이션이 사용할 수 있도록 다음 번 무료 주문 번호를 유지보수하기 위해 결합 기능 데이터 테이블에서 레코드를 사용할 수 있습니다. 다른 예제는 다음과 같습니다.

- 전화 번호 또는 다수의 도난당한 신용 카드의 검색 테이블
- 고객 목록으로부터의 고객의 서브세트와 같은, 몇 개의 항목으로 구성되는 작업 데이터
- 애플리케이션 사용자에게 고유하거나 애플리케이션이 실행되고 있는 터미널과 관련되는 정보
- 추가 처리를 위해 더 큰 파일이나 데이터베이스로부터 추출된 데이터.

결합 기능 데이터 테이블을 사용하면 비공식 데이터에 대한 다양한 액세스 유형이 허용됩니다. 읽기 전용, 단일 업데이트, 다중 업데이트, 순차 액세스, 임의 액세스, 임의 삽입 및 삭제입니다.

여러 용도를 위해, 이는 범위가 글로벌이므로 결합 기능 데이터 테이블은 CICS CWA(Common Work Area) 등과 같은 자원에 대해 중요한 장점을 제공할 수 있습니다.

애플리케이션에는 CFDT는 시스플렉스 전체 사용자 유지 데이터 테이블처럼 나타납니다. CFDT는 UMT와 같은 API의 서브세트를 사용하여 액세스됩니다. (즉, MASSINSERT와 RBA 옵션을 제외하고 전체 파일 제어 API입니다.) 그러나 CFDT는 16바이트의 최대 키 길이로 제한됩니다.

사용자 유지보수 데이터 테이블과의 다음 비교를 참조하십시오.

- CFDT에 대한 갱신은 UMT에 대한 갱신과 마찬가지로 기본 VSAM 데이터 세트에 반영되지 않습니다(테이블이 처음에 여기에서 로드된 경우). 갱신은 CFDT에만 작성됩니다.
- CFDT는 CFDT를 참조하는 마지막 파일이 닫힌 경우라도 테이블이 처음에 결합 기능 데이터 테이블에서 작성되고 결합 기능에 남아 있는 경우에 한 번만 로드됩니다 (반면, UMT는 소유 리전이 종료될 때마다 삭제됨). CFDT 서버 DELETE TABLE 명령을 사용하여 CFDT에서 먼저 테이블을 삭제해야만 원본 소스 데이터에서 CFDT의 다시 로드를 강제 실행할 수 있습니다. 삭제 작업 후에 CFDT에 대해 열린 첫 번째 파일은 서버가 테이블을 다시 로드하게 합니다.

참고: 결합 기능 데이터 테이블 풀은 결합 기능 목록 구조로서 정의되고 둘 이상의 데이터 테이블을 보관할 수 있습니다. 결합 기능 데이터 테이블의 목록 구조 작성에 대한 자세한 정보는 CICS 시스템 정의 안내서를 참조하십시오.

- 로드 과정에 있는 UMT의 액세스 규칙은 테이블로부터(레코드가 이미 로드된 경우) 또는 소스 데이터 세트로부터 만족할 모든 직접 읽기 요청을 허용하지만 LOADING

조건이 있는 모든 갱신 요청 또는 부정확한 읽기 또는 찾아보기 요청을 거부합니다. CFDT의 경우 로드 중에 모든 요청이 허용되지만 요청은 이미 로드된 키 범위 내에 있는 레코드의 경우에만 성공합니다.

결합 기능 데이터 테이블 모델

두 개의 결합 기능 데이터 테이블의 모델이 있습니다.

경합 모델

이는 최적의 성능을 제공하지만 데이터가 갱신 읽기 요청을 실행한 후에 변경된 상황을 처리하기 위해 작성된 프로그램을 필요로 합니다. 새 CHANGED 응답은 REWRITE 또는 DELETE 명령에서 발생할 수 있습니다. 이는 기존 NOTFND 응답의 경우 새로운 용도이고, 이는 프로그램이 갱신 읽기 요청을 실행한 후에 레코드가 삭제된 애플리케이션을 표시하기 위해 리턴될 수 있습니다.

참고: REWRITE 또는 DELETE에서 CHANGED 또는 NOTFND 예외를 수신할 수 없는 것이 확실한 경우에는 결합 모델을 사용하여 기존 프로그램 사용하는 것이 가능할 수도 있습니다. 이것의 예제는 애플리케이션이 프로그램의 사용자와 관련된 레코드에서만 작동하므로 다른 사용자가 동일 레코드를 갱신할 수 없는 경우일 수 있습니다.

잠금 모델

이 모델은 파일 제어 API의 UMT 서브세트를 준수하는 기존 프로그램과의 API-호환 가능합니다. 잠금 모델은 다음과 같을 수 있습니다.

복구 불가능

복구 불가능한 CFDT에 대한 갱신의 경우 잠금은 동기점까지 지속되지 않고(파일 제어 요청의 완료 시에 릴리스됨) 갱신은 작업 단위가 실패하는 경우에 백아웃되지 않습니다.

복구 가능

CFDT는 작업 단위 실패, CICS 리전 실패, CFDT 서버 실패 및 MVS 실패 시에 복구 가능합니다. 실패 당시에 인플라이트인 작업 단위가 작성한 갱신은 백아웃됩니다.

복구 가능한 잠금 모델은 인다우트 및 백아웃 실패를 지원합니다. 갱신을 CFDT에 백아웃할 때 작업 단위가 실패하거나 동기점 처리 중에 인다우트에 실패하는 경우에는 잠금은 보존된 잠금으로 변환되고 작업 단위가 처리 지연됩니다.

CFDT는 앞으로 복구 가능할 수 없습니다. CFDT는 이것이 상주하는 CF 구조의 손실로부터 살아남지 않습니다.

파일 응답 정의에 각 테이블에 원하는 갱신 모델을 지정하여 서로 다른 테이블은 서로 다른 모델을 사용하게 할 수 있습니다.

데이터 공유 기술

데이터 공유를 위해 사용할 수 있는 CICS 기술이 테이블 형식으로 비교되어 결합 기능 데이터 테이블 사용을 고려할 시기를 보여줍니다.

표 29. 스크래치패드 데이터 공유 기술

제한조건과 요인	단일 리전	단일 MVS	Sysplex
더 이상 권장하지 않는 기술 (너무 제한적임)	TWA	—	—
각 트랜잭션에 대한 단일 영역 에 대해 권장하는 방법	COMMAREA 또는 채널	COMMAREA 또는 채널	COMMAREA 또는 채널
기존 애플리케이션이 임시 저장영역(TS) 큐를 사용함	로컬 TS 큐	원격 TS 큐	공유 TS 큐
기존 프로그램이 UMT Random 삽입을 사용하고 저장되는 데이터의 다중 유형을 삭제함	UMT	원격 UMT	CFDT(경합 모델)

표 29에서는, 트랜잭션 단계 사이에서 스크래치패드 데이터에 대해 서로 다른 기술이 고려되는데, 한 번에 하나의 태스크만이 데이터에 액세스 중이지만 데이터는 한 리전의 태스크에서 다른 리전의 태스크로 전달될 수 있습니다. '원격 UMT'는 필요할 때(즉, 갱신 프로세스를 위해) 제공되는 함수 또는 비갱신 프로세스의 경우 SDT 교차 메모리 공유에 의해 AOR로부터 액세스되는 공유 사용자 유지보수 데이터 테이블을 의미합니다. 테이블은 Parallel Sysplex® 내에서 결합 기능 데이터 테이블이 데이터의 랜덤 삽입 및 삭제 및 다중 유형의 데이터가 저장되어야 하는 경우에 최상의 솔루션임을 보여줍니다. 이런 제한조건이 없으면, 공유 TS 큐가 애플리케이션이 이미 임시 저장영역을 사용 중인 경우 더 적합한 선택사항입니다.

표 30. 데이터의 큐 공유를 위한 기술

제한조건과 요인	단일 리전	단일 MVS	Sysplex
선행 시 읽기 전용, 후미 시 쓰기 전용 트리거링 필수	로컬 트랜지언트 데이터(TD)	원격 TD	원격 TD
항목의 프로세스 일괄처리	TS 큐 또는 UMT	원격 TS 또는 원격 UMT	공유 TS 또는 CFDT
처리 후 각 항목 삭제. 랜덤 삽입 및 삭제 필수.	UMT	원격 UMT	CFDT

693 페이지의 표 30에 데이터 큐 공유를 위한 여러 가지 기술이 표시되는데, 정보가 다른 애플리케이션 또는 태스크에 의해 동일한 순서로 처리되기 위해 특정 순서로 저장됩니다. CICS 트랜지언트 데이터 및 임시 저장영역 큐 기능이 대부분의 경우에 권장되며, 데이터 테이블이 순차 데이터 처리를 위해 더 적합한 솔루션을 제공하는 몇 가지 인스턴스가 있습니다.

표 31. 제어 레코드 공유 기술

제한조건과 요인	단일 리전	단일 MVS	Sysplex
더 이상 권장되지 않는 기술	CWA	MVS CSA	—
단일 갱신 리전, 단일 레코드	TS 큐 또는 UMT	원격 TS 큐 또는 UMT	공유 TS 큐 또는 CFDT(경합 모델)
다중 갱신 리전 또는 다중 레코드	UMT	원격 UMT	CFDT

표 31에는 제어 레코드 관리를 위한 여러 가지 기술이 표시됩니다. 이것은 중앙 제어 레코드가 정보를 모든 트랜잭션에 사용 가능하게 만드는 데 사용되는 것을 설명합니다. 예를 들어, 이것은 다음의 사용하지 않은 주문 번호나 고객 번호를 포함하여 프로그램이 키 지정 파일이나 데이터베이스에서 새 레코드를 쉽게 작성할 수 있게 합니다. (이 유형의 애플리케이션의 경우 역시 sysplex 전체 기능인 이름 지정된 카운터 기능을 고려하십시오. 자세한 내용은 851 페이지의 제 43 장 『이름 지정된 카운터 서버』의 내용을 참조하십시오.)

테이블은 MVS 이미지 내에서 단일 레코드에 대한 모든 갱신을 만드는 단일 리전이 있는 경우 어떤 Function Shipping 오버헤드도 없이 UMT를 사용할 수 있음을 보여줍니다.

제어 레코드를 갱신하는 다중 리전이 있거나 갱신할 둘 이상의 제어 레코드가 있는 경우, 결합 기능 데이터 테이블이 Parallel Sysplex 환경에서 유일한 솔루션이며 단일 MVS 내에서 UMT에 대한 갱신을 Function Shipping하는 것보다 더 효과적일 수 있습니다.

표 32. 키 지정 데이터 공유 기술

제한조건과 요인	단일 리전	단일 MVS	Sysplex
읽기 전용 또는 드물게 갱신됨	UMT	UMT	복제 UMT
단일 갱신 리전	UMT	UMT	복제 UMT 또는 CFDT
다중 갱신 리전 복구 가능(백아웃 전용)	UMT	원격 UMT 또는 CFDT	CFDT

표 32에 키 지정 데이터 공유를 위한 여러 가지 기술이 표시됩니다. 이것은 일반 키 지정 파일과 비슷한 구조를 갖는 데이터를 사용하지만 정보가 영구적으로 저장될 필요가 없고 성능 이점이 관련 데이터를 저장하기 위해 기본 저장영역 또는 결합 기능 자원의 사용을 정당화하기에 충분한 애플리케이션을 포함합니다.

이 데이터는 파일 제어 API를 사용하여 가정 적합하게 액세스되는데, Parallel Sysplex 내에서 솔루션은 다음을 사용하는 것임을 의미합니다.

- 최고 성능이 필요하고 액세스가 읽기 전용이거나 갱신이 드물고 단일 리전에서 이들 을 작성하고 다른 리전의 복제 UMT를 새로 고치기 위해 배열할 수 있는 복제된 사용자 유지보수 데이터 테이블
- 결합 기능 데이터 테이블.

UMT에 대한 복구 지원은 실패 후의 트랜잭션 백아웃으로 제한됨을 주의하십시오. 결합 기능 데이터 테이블의 경우 CICS 및 CFDT 서버 장애 및 인다우트 실패에 대한 복구도 제공됩니다.

트랜잭션 교착상태

트랜잭션 교착상태가 일어나지 않도록 애플리케이션을 디자인합니다. 트랜잭션은 파일 제어 명령 실행 중에 자원을 독점적으로 제어해야 합니다.

두 트랜잭션 모두(예를 들어, A와 B) 이미 다른 트랜잭션에서 보유 중인 일부 자원을 독점적으로 사용해야 하는 경우(예를 들어, 데이터 세트의 특정 레코드) 교착상태가 발생합니다. 트랜잭션 A는 자원이 사용 가능해지길 기다립니다. 그렇지만 트랜잭션 B가 이를 해제할 위치에 있지 않아서 이로 인해 결국 A에서 보유 중인 일부 자원을 대기하게 되면 결국 둘은 모두 교착상태가 되고 교착상태를 중지할 수 있는 유일한 방법은 트랜잭션 중 하나를 취소해서 해당 자원을 해제하는 방법입니다.

VSAM 및 BDAM 데이터 세트 모두에 대해 수정 중인 모든 레코드가 수정 시작 시점(예를 들어, READ UPDATE 명령이 실행되어 레코드 제어 확보)에서 종료 시점(예를 들어, REWRITE 명령이 레코드 변경을 완료)까지 CICS의 독점 제어 하에 보유됩니다.

RLS 모드로 액세스한 VSAM 파일의 경우 이 프로세스 중에 개별 레코드만 잠금됩니다. 비RLS 모드로 액세스하는 VSAM 파일의 경우, VSAM이 레코드 제어가 필요한 명령을 수신하면 레코드를 포함하는 전체 제어 간격을 잠금합니다. 그러면 CICS는 필요한 레코드에서 큐에 넣기를 확보하고 제어 간격을 해제하여 레코드만 잠금됩니다. 제어 간격 잠금은 각 명령 후에 해제되며 개별 레코드만 수정 프로세스 전체에 대해 잠금됩니다. (제어 간격 잠금이 해제되는 방법에 대한 자세한 정보는 CICS 복구 및 다시 시작 안내서에서 잠금을 참조하십시오.)

z/OS용 CICS Transaction Server, 버전 2 릴리스 2 이전 릴리스에서, 액세스 방법은 수정이 시작될 시점부터 종료될 시점까지 명령 사이의 전체 제어 간격에서 해당 잠금도 보유했었습니다. 이제는 더 이상 그러지 않습니다.

수정 프로세스 중에 CICS가 레코드를 독점 제어할 뿐만 아니라 트랜잭션이 복구 가능 파일을 수정할 때도 추가적인 잠금 기간이 있습니다. 이 상황에서 CICS(또는 파일이

RLS 모드로 액세스되는 경우 VSAM)는 변경을 수행한 요청이 완료된 후에도 트랜잭션에 대해 해당 레코드를 잠급니다. 트랜잭션은 동일한 레코드에 계속 액세스하고 수정할 수 있습니다. 다른 트랜잭션은 트랜잭션이 종료 또는 동기점 요청을 발행하여 잠금을 해제할 때까지 기다려야 합니다. 자세한 정보는 421 페이지의 『동기점 지정』의 내용을 참조하십시오.

교착상태 발생 여부는 다른 동시 트랜잭션의 자원 확보 및 해제 시기에 따라 달라집니다. 애플리케이션은 교착상태를 초래하는 상황이 발생하기 전까지는 얼마 동안 사용할 수 있습니다. 초기 애플리케이션 디자인 단계에서 교착상태의 가능성을 인지하고 허용하는 것이 중요합니다.

다음은 복구 가능 데이터 세트에서 찾은 다른 유형의 교착상태 예제입니다.

- 동시에 실행되는 두 개의 트랜잭션은 한 개의 복구 가능 파일 내에서 다음과 같이 레코드를 수정합니다.

Transaction 1

```
READ UPDATE record 1
UNLOCK      record 1

WRITE      record 2
```

Transaction 2

```
DELETE      record 2

READ UPDATE record 1
REWRITE     record 1
```

Transaction 1은 record 1에 대한 레코드 잠금을 확보했습니다(UNLOCK 명령으로 READ UPDATE 명령을 완료한 경우도 포함). Transaction 2는 유사하게 record 2에 대한 레코드 잠금을 확보했습니다. 그러면 트랜잭션은 각각이 서로 보유한 CICS 잠금을 확보하려고 하기 때문에 교착상태가 됩니다. CICS 잠금은 동기점까지 해제되지 않습니다.

- 동시에 실행 중인 두 개의 트랜잭션이 다음과 같이 두 개의 복구 가능 파일을 수정 중입니다.

Transaction 1

```
READ UPDATE file 1, record 1
REWRITE     file 1, record 1

READ UPDATE file 2, record 2
REWRITE     file 2, record 2
```

Transaction 2

```
READ UPDATE file 2, record 2
REWRITE     file 2, record 2

READ UPDATE file 1, record 1
REWRITE     file 1, record 1
```

여기에서 레코드 잠금은 다른 파일뿐만 아니라 다른 레코드에서도 확보되었으며 교착상태는 첫 번째 예제와 유사합니다.

비RLS 모드로 액세스한 VSAM 파일의 경우, CICS가 교착상태를 발견하고 교착상태가 되는 트랜잭션은 다른 트랜잭션으로 교착상태가 되는 경우 이상 종료 코드 AFCE로 이상 종료되거나 자체적으로 교착상태가 되면 이상 종료 코드 AFCE로 이상 종료됩니다.

VSAM 발견 교착(RLS만 해당)

파일에 RLS 모드에서 액세스한 상태에서, 두 개의 서로 다른 CICS 리전 간에 교착이 발생할 수 있으며 서로 다른 MVS 이미지 하에서 실행 중일 수 있습니다. 이 경우 CICS는 교착 발견과 해결을 수행할 수 없으므로 VSAM에 의해 수행됩니다.

VSAM인 RLS 교착 조건을 발견하면 이는 교착 예외 조건을 CICS에 리턴하여 CICS 파일 제어가 AFCE 이상 종료 코드로 트랜잭션을 이상 종료하도록 합니다. CICS는 또한 교착 체인의 멤버를 식별하는 메시지와 추적 항목을 작성합니다.

그러나 VSAM은 또 다른 자원 관리자가 관련된 교차 자원 교착(예: RLS 및 DB2 자원의 사용에서 발생하는 교착)을 발견할 수 없습니다. 교차 자원 교착은 제한시간이 만료되고 대기 요청이 제한시간 초과될 때 VSAM이 해결합니다. 이 상황에서 VSAM은 제한시간이 교차 자원 교착으로 인한 것인지 또는 RLS 잠금을 획득하고 이를 해제하지 않는 또 다른 트랜잭션에 의해 초래되었는지 여부를 판별할 수 없습니다. 제한시간이 발생하면 CICS는 추적 항목과 메시지를 쓰고 제한시간 트랜잭션이 대기 중인 잠금의 홀더를 식별합니다.

RLS 모드에서 발행된 모든 파일 제어 요청에는 연관된 제한시간 값이 있습니다. 이 제한시간 값은 DTIMEOUT이 트랜잭션 동안 활성인 경우 DTIMEOUT에 의해 정의되고 DTIMEOUT이 활성이 아닌 경우에는 시스템 초기화 테이블로부터 FTIMEOUT에 의해 정의됩니다.

교착을 피하기 위한 규칙

교착을 피하려면 다음 규칙을 사용하십시오.

- 여러 자원을 갱신(수정)하는 모든 애플리케이션은 동일한 순서로 수행해야 합니다. 예를 들어, 트랜잭션이 데이터 세트에서 둘 이상의 레코드를 갱신하는 중이라면 이를 오름차순 키 순서로 수행할 수 있습니다. 둘 이상의 파일에 액세스 중인 트랜잭션은 항상 동일한 사전 정의된 파일 순서로 수행해야 합니다.

데이터 세트에 대체 색인이 있는 경우에는, 대체 키를 사용하여 몇몇 갱신을 수행하는 트랜잭션과 기본 키를 사용하여 몇몇 갱신을 수행하는 트랜잭션을 혼합할 때 주의하십시오. 갱신을 수행하는 트랜잭션은 항상 오름차순 키 순서로 레코드에 액세스한다고 가정하십시오. 그런 다음 기본 키를 사용하여 모든 갱신을 수행하는 트랜잭션은 기본 키를 사용하여 모든 갱신을 수행하는 다른 트랜잭션과 교착하지 않습니다. 마찬가지로, 대체 키를 사용하여 모든 갱신을 수행하는 트랜잭션은 대체 키를 사용하여 모든 갱신을 수행하는 다른 트랜잭션과 교착하지 않습니다. 기본 키를 사용하

여 모든 갱신을 수행하는 트랜잭션은 대체 키를 사용하여 모든 갱신을 수행하는 트랜잭션과 교착할 수 있습니다. 이는 잠겨 있는 키가 항상 기본 키이기 때문입니다. 결과적으로, 대체 키를 사용하여 갱신을 수행하는 트랜잭션은 기본 키를 사용하여 갱신을 수행하는 트랜잭션과 다른 순서로 잠금을 획득할 수 있습니다.

- READ UPDATE 명령을 실행하는 애플리케이션은 REWRITE, RIDFLD 없는 DELETE 또는 UNLOCK 명령을 사용하여 이를 수행하여 파일에 다른 작업을 수행하기 전에 위치를 해제하거나 각 갱신 요청의 두 파트 모두에 TOKEN 옵션을 포함해야 합니다.
- WRITE MASSINSERT 명령의 순서는 UNLOCK 명령으로 종료하여 위치를 해제해야 합니다. 파일에 대한 다른 작업은 UNLOCK 명령이 실행되기 전에는 수행해서는 안 됩니다.
- 애플리케이션은 파일에 READ UPDATE, WRITE 또는 RIDFLD가 있는 DELETE 명령을 실행하기 전에 ENDBR 명령을 사용하여(VSAM 잠금 해제) 파일에서 모든 찾아보기를 종료해야 합니다.

파일 제어 조작

CICS 파일 제어에는 VSAM 및 BDAM 데이터 세트에서 데이터를 읽고 갱신하고 추가 및 찾아보고 VSAM 데이터 세트에서 데이터를 삭제할 수 있는 옵션이 있습니다. 파일 제어를 사용하여 CICS 공유 데이터 테이블 및 결합 기능 데이터 테이블에 액세스할 수도 있습니다.

VSAM 자원을 대기한 후에 CICS 처리

CICS 파일 제어는 자원에 대해 대기하는 태스크(예: 제어 간격의 VSAM 독점 제어) 중 자원이 사용 가능해지면 다시 디스패치할 태스크를 구성하지 않습니다. VSAM이 제어 간격의 독점 제어를 해제하면 CICS 디스패처가 태스크 우선순위 및 우선순위 유효 기간과 같은 요소에 따라 디스패치할 다음 태스크를 선택합니다. 자원에서 일시중단된 순서대로 태스크가 사용 가능한 상태로 다시 디스패치되는지는 보장되지 않습니다.

CICS 명령을 사용하여 레코드 읽기

READ 명령을 사용하여 직접 읽기, 순차 읽기 및 순차 생략 읽기의 애플리케이션이 레코드를 읽기 위한 3가지 방법이 있습니다. 레코드를 읽기 위한 이런 모든 방법은 VSAM 및 BDAM 데이터 세트 모두에서 사용될 수 있습니다.

이 태스크 정보

파일은 고정 길이 또는 가변 길이 레코드 중 하나를 포함하는 것으로 파일 정의에서 정의될 수 있습니다. 다음 경우에는 고정 길이 레코드만 정의하십시오.

- VSAM 데이터 세트의 정의(엑세스 방법 정의를 사용한)는 최대 레코드 크기와 동일한 평균 레코드 크기를 지정합니다.

- 및 데이터 세트의 모든 레코드는 해당 길이입니다.

직접 읽기 및 찾아보기의 경우 파일에 고정 길이 레코드가 있는 경우 및 애플리케이션이 레코드를 읽어들이 영역을 제공하는 경우, 해당 영역은 정의된 길이여야 합니다. 파일에 가변 길이 레코드가 있는 경우, 명령은 또한 레코드를 보유하기 위해 제공되는 영역의 길이(일반적으로 파일에 있는 레코드의 최대 길이)도 지정해야 합니다.

고정 길이 레코드 및 (사용자가 SET 옵션을 사용할 때) CICS에 의해 제공되는 저장 영역으로 검색되는 레코드의 경우 LENGTH 인수를 지정할 필요가 없습니다. 그러나 LENGTH 인수가 옵션이지만 CICS가 읽어지는 레코드가 사용 가능한 데이터 영역에 너무 길지 않은지 확인하게 만들기 때문에 INTO 옵션을 사용할 때 LENGTH 인수를 지정할 것을 권장합니다. LENGTH를 지정하는 경우 CICS는 LENGTH 필드를 사용하여 검색된 레코드의 실제 길이를 리턴합니다.

직접 읽기(READ 명령 사용)

파일의 레코드는 레코드를 지정하는 RIDFLD(레코드 식별 필드) 옵션과 함께 READ 명령을 사용하여 읽을 수 있습니다. 레코드 식별 필드의 내용과 길이를 설명하는 추가 옵션을 사용할 수 있습니다.

이 태스크 정보

READ 명령을 사용할 때,

프로시저

- 원하는 레코드를 지정하고 레코드 식별 필드의 내용을 설명하기 위한 추가 옵션을 추가하려면 RIDFLD(레코드 식별 필드) 옵션을 사용하십시오. 레코드를 식별하는 정확한 방법은 데이터 세트의 유형에 따라 다릅니다.
 1. KSDS의 경우 전체 키를 지정하여 원하는 레코드를 고유하게 식별할 수 있거나, 일반 키를 지정하고 그의 키가 해당 요구사항을 만족하는 첫 번째(가장 낮은 키) 레코드를 검색할 수 있습니다.
 - GENERIC 옵션은 키의 일부에서만 일치 필요함을 표시합니다. GENERIC 옵션을 지정할 때, 왼쪽부터 시작하여 일치해야 하는 키의 위치 수를 말하기 위해 KEYLENGTH 옵션도 지정해야 합니다. READ 명령의 경우, CICS는 첫 번째 KEYLENGTH 옵션 문자만 사용합니다.
 - GTEQ(크거나 같음) 옵션은 그의 키가 사용자가 지정한 키『보다 크거나 같은』 첫 번째 레코드를 원함을 표시합니다. 전체 또는 일반 키와 함께 GTEQ를 사용할 수 있습니다.
 - GTEQ 옵션의 반대는 EQUAL 옵션(기본값)이며, 키가 사용자가 지정한 키의 해당 부분(전체 또는 일반)과 정확하게 일치하는 레코드만 원함을 의미합니다.

KSDS가 대체 색인과 대체 색인 경로를 갖는 경우, 대체 색인에서 설정한 대체 키를 지정하여 파일에서 레코드를 검색할 수 있습니다(3 단계 참조).

2. 표준(확장되지 않은) ESDS의 경우, 그의 상대 바이트 주소(RBA)를 지정하여 레코드를 식별할 수 있습니다.

CICS에 RBA가 사용 중임을 알려려면 RBA 옵션을 추가하십시오. ESDS에서 레코드의 RBA는 변할 수 없기 때문에, 애플리케이션은 액세스하려는 레코드에 대응하는 RBA의 값을 계속 추적할 수 있습니다. RBA는 항상 레코드의 시작을 가리켜야 합니다. KSDS에 대해 사용할 수 있는 GENERIC 또는 GTEQ 옵션의 등가는 없습니다.

3. 확장 ESDS의 경우, 확장 상대 바이트 주소(XRBA)를 지정하여 레코드를 식별할 수 있습니다.

CICS에 XRBA가 사용 중임을 알려려면 XRBA 옵션을 추가하십시오. ESDS에서 레코드의 XRBA는 변할 수 없기 때문에, 애플리케이션은 액세스하려는 레코드에 대응하는 XRBA의 값을 계속 추적할 수 있습니다. XRBA는 항상 레코드의 시작을 가리켜야 합니다. KSDS에 대해 사용할 수 있는 GENERIC 또는 GTEQ 옵션의 등가는 없습니다.

4. KSDS 또는 대체 색인을 갖는 표준 ESDS(확장 ESDS는 표준 색인을 가질 수 없음)의 경우, 대체 색인에서 설정한 대체 키를 지정하여 파일에서 레코드를 검색할 수 있습니다. 대체 키를 사용할 때,

- GENERIC 옵션과 GTEQ 옵션이 기본 키를 사용한 KSDS에서 읽기의 경우와 동일한 방법으로 KSDS 및 ESDS 레코드 모두에 대해 사용될 수 있습니다.
- 대체 키가 고유하지 않은 경우, 해당 키를 갖는 파일의 첫 번째 레코드가 읽어지고 DUPKEY 상태를 얻습니다. 동일한 대체 키를 갖는 다른 레코드를 검색하려면 이 지점에서 찾아보기 조작을 시작해야 합니다.
- 일치하는 레코드가 없는 경우, 식별되는 레코드는 다음으로 높은 대체 키를 갖는 레코드입니다.

5. RRDS의 경우, 상대 레코드 번호(RRN)를 지정하여 레코드를 식별하십시오. CICS에 RRN이 사용 중임을 알려려면 RRN 옵션을 추가하십시오. 애플리케이션은 원하는 레코드의 RRN 값을 알아야 합니다. KSDS에 대해 사용할 수 있는 GENERIC 또는 GTEQ 옵션의 등가는 없으며, 대체 키를 사용할 수 없습니다.

- 필요한 경우 KEYLENGTH 옵션을 사용하여 RIDFLD 옵션에서 지정한 키의 길이를 명시하십시오.

1. 레코드 식별 필드에 대해 RBA 또는 RRN을 지정하고 RBA 또는 RRN 옵션을 지정하는 경우, KEYLENGTH 옵션은 필수가 아닙니다.

2. GENERIC 옵션을 지정하는 경우 KEYLENGTH 옵션은 필수입니다. VSAM 정의에서 지정된 키 길이보다는 작아야 하는 일반 키의 길이를 지정하십시오.
3. 원격 파일을 읽는 중이고 SYSID 옵션을 지정하는 경우, 키 길이는 다음 방법 중 하나로 식별할 수 있습니다.
 - 파일 정의에서 지정됩니다.
 - KEYLENGTH 옵션을 사용하여 애플리케이션에 의해 지정됩니다.
 - 키가 4문자보다 길고 키 길이가 파일 정의에서 또는 애플리케이션에 의해 지정되지 않는 경우 4로 기본 설정됩니다.

이것은 WRITE 명령에 대한 상황과는 다른데, 이 경우 RBA 또는 RRN을 사용한 경우가 아니면 KEYLENGTH 옵션이 원격 파일을 위해 필수입니다.

4. 다른 상황에서는 KEYLENGTH 옵션을 지정하거나 생략할 수 있습니다. 데이터 세트에 대해 정의된 길이와는 다른 KEYLENGTH를 지정하고 조작이 일반이 아닌 경우 INVREQ 상태가 발생합니다.
- 레코드가 애플리케이션에 의해 제공되는 기본 저장영역의 영역으로 읽어지는지(READ INTO), 아니면 파일 제어에 의해 획득되는 기본 저장영역 CICS SET 저장영역의 영역으로 읽어지는지(READ SET) 여부를 명시하려면 INTO 또는 SET 옵션을 사용하십시오. 후자의 경우, CICS SET 저장영역에 있는 데이터의 주소가 프로그램으로 리턴됩니다. CICS SET 저장영역은 일반적으로 다음 동기점이나 태스크의 종료 시까지, 또는 동일한 파일에 대한 다음 READ 시까지 어느 것이든 먼저 오는 상황까지 유효하게 남아 있습니다.
 - 필요한 경우 LENGTH 매개변수가 설정되었는지 확인하십시오.
 1. SET 옵션을 사용한 경우, LENGTH 옵션을 지정할 필요가 없습니다. 그러나 레코드 길이가 VSAM에 원래 정의된 것과 일치하는지 여부를 확인하려는 경우 이를 지정할 수 있습니다. 지정된 데이터 영역은 레코드가 검색된 후에 실제 레코드 길이로 설정됩니다.
 2. 고정 길이 레코드의 경우, LENGTH 옵션을 지정할 필요가 없지만 지정할 것을 권고합니다. 옵션을 지정하는 경우, CICS는 레코드가 사용 가능한 데이터 영역에 대해 너무 길지 않은지 확인합니다. 사용자가 지정하는 길이가 파일이 작성된 시점에서 지정된 레코드 길이와 정확하게 일치해야 합니다.
 3. 가변 길이 레코드를 읽기 위한 INTO 옵션을 사용한 경우, 길이 매개변수를 제공해야 합니다. 그러나 LENGTH 옵션이 항상 명령에서 명시적으로 지정될 필요는 없습니다.
 - 원격 시스템의 파일의 경우, LENGTH 매개변수를 여기에서 설정할 필요는 없지만 파일 자원 정의에서 설정되어야 합니다.
 - 어셈블러 언어나 PL/I를 사용 중인 경우, LENGTH를 명시적으로 지정하는 대신 어셈블러 언어의 길이 속성 참조나 PL/I의 STG 및 CSTG를 사용하여 기본 설정할 수 있습니다. C에서는 LENGTH를 명시적으로 지정해야 합니다.

가변 길이 파일을 읽을 때 LENGTH 옵션을 명시적으로 지정하지 않는 경우 항상 애플리케이션이 허용하는 가장 긴 레코드를 지정하십시오. 이것은 데이터 세트가 작성될 때 최대 레코드 크기로 정의된 값에 해당해야 합니다. 레코드가 지정된 길이를 초과하는 경우, LENGERR 조건이 발생하고 레코드는 해당 길이로 잘립니다. 레코드가 검색된 후, 레코드에서 지정된 데이터 영역이(자르기가 발생하기 전에) 실제 레코드 길이로 설정됩니다.

- 파일이 RLS 모드에서 열리는 경우 UNCOMMITTED, CONSISTENT, REPEATABLE 및 NOSUSPEND 옵션을 사용하여 읽기 무결성을 제어하십시오.
 1. UNCOMMITTED 옵션을 지정하는 경우, 읽기 무결성이 없으며 공유 잠금이 읽기 요청에 사용되지 않습니다. 682 페이지의 『RLS 레코드 레벨 잠금』이 공유 및 독점 잠금에 대해 설명합니다. 이것이 기본값이며 파일 제어가 비RLS 모드에서 열린 파일에 대해 작업하는 방법입니다.
 2. CONSISTENT 옵션을 지정하는 경우, 레코드가 다른 태스크에 의해 갱신되고 있는 중이면 레코드 읽기 요청이 큐 처리됩니다. 읽기는 갱신이 완료될 때만 완료되며, 갱신 작업 단위(UOW)가 독점 잠금을 포기합니다. 421 페이지의 『동기점 지정』에서 UOW 및 동기점에 대해 설명합니다.
 3. REPEATABLE 옵션을 지정하는 경우, 읽기 요청의 처리는 리더가 동기점까지 그의 읽기 잠금을 보유한다는 점을 제외하면 일치 읽기 요청의 경우와 동일합니다. 이것은 복구 가능 및 복구 불가능한 파일 모두에 적용됩니다. 이것은 UOW가 추가 읽기 요청을 작성하는 동안에는 UOW의 레코드 읽기를 수정할 수 없도록 보장합니다. 특히 일련의 관련 읽기 요청을 발행하고 마지막 레코드를 읽기 전에는 어떤 레코드도 수정되지 않도록 보장하려는 경우에 유용합니다.
 4. CONSISTENT 또는 REPEATABLE을 지정하는 경우, NOSUSPEND 옵션도 지정하여 레코드가 활성 잠금을 사용하여 VSAM에 의해 잠기는 경우 요청이 대기하지 않도록 할 수 있습니다. 685 페이지의 『잠금에 대한 활성 및 보유 상태』에서 활성 잠금에 대해 설명합니다.

어떤 옵션도 지정하지 않는 경우, 파일 자원 정의의 값이 사용됩니다.

참고: 애플리케이션이 '시간이 경과된(stale)' 데이터를 용인하지 않을 때만 읽기 무결성을 지정하십시오. 이것은 RLS가 잠금을 사용하여 읽기 무결성을 지원하며, 결과적으로 애플리케이션이 읽기 무결성을 지원하지 않는 CICS 릴리스에서는 발생하지 않는 교착상태를 만날 수 있습니다. 이것은 특히 파일 자원 정의에서 읽기 무결성을 정의하는 경우에 중요합니다. 이들 파일을 참조하는 애플리케이션이 읽기 무결성을 지원하지 않고 읽기 전용 파일 액세스에 대한 교착 상태를 다루도록 설계되지 않은 CICS 릴리스에 대해 작성되었을 수 있습니다.

순차 읽기(찾아보기)

파일에서 레코드를 찾아보려면 **STARTBR**, **READNEXT**, **READPREV**, **RESETBR** 명령을 사용하십시오. 앞이나 뒤쪽으로 찾아볼 수 있으며, 찾아보기가 진행 중인 동안 찾아보기

의 위치나 특성을 변경할 수 있습니다. 찾아보기를 종료하려면 **ENDBR**, **SYNCPOINT**, **SYNCPOINT ROLLBACK** 명령을 사용하십시오.

이 태스크 정보

레코드를 찾아볼 때 일반적으로 **READ** 명령(699 페이지의 『직접 읽기(READ 명령 사용)』 참조)을 사용한 직접 읽기의 경우와 동일한 방식으로 레코드를 식별하고 읽습니다. 레코드 식별 필드(RIDFLD 옵션)를 지정하고, 읽기 레코드의 대상을 선택합니다. 파일이 RLS 모드에서 열리는 경우 UNCOMMITTED, CONSISTENT, REPEATABLE 및 NOSUSPEND 옵션을 사용하여 읽기 무결성을 제어할 수 있습니다. 이 주제에서 찾아보기에 대한 몇 가지 추가 특수 사례가 다루어집니다.

찾아보기 및 직접 읽기에 대해 동일한 유형의 키가 사용됩니다.

- KSDS의 경우 전체 키, 일반 키 또는 대체 키를 사용할 수 있습니다.
- 표준(확장되지 않은) ESDS의 경우, RBA나 대체 키를 사용할 수 있습니다.
- 확장 ESDS의 경우, 확장 RBA(XRBA)를 사용할 수 있습니다.

참고: 일부 환경에서는 32비트 RBA를 사용하여 확장(64비트) ESDS에 액세스할 수 있습니다. 678 페이지의 『상대 레코드 번호(RRN), 상대 바이트 주소(RBA) 및 확장 상대 바이트 주소(XRBA)』의 내용을 참조하십시오.

- RRDS의 경우 RRN이 사용됩니다.

찾아보기를 위해 대체 키를 사용하는 경우 레코드는 대체 키 순서로 검색됩니다. 대체 키가 고유하지 않은 경우, 중복 키의 마지막 발생을 제외한 각 검색 조작에 대해 DUPKEY 상태가 발생합니다. 예를 들어, 동일한 대체 키를 갖는 3개의 레코드가 있는 경우, 처음 둘에서 DUPKEY가 발생하지만 세 번째는 발생하지 않습니다. 중복 대체 키를 갖는 레코드가 리턴되는 순서는 레코드가 데이터 세트에 기록된 순서입니다. 이것은 **READNEXT** 또는 **READPREV** 명령을 사용 중이든 간에 참입니다. 이 때문에 동일한 대체 키를 갖는 레코드를 반대 순서로 검색할 수 없습니다.

CICS는 트랜잭션이 동일한 파일에 대해 동시에 둘 이상의 찾아보기를 수행하도록 허용합니다. 각 찾아보기 명령에 REQID 옵션을 포함하여 찾아보기 조작을 구별합니다.

이 주제의 지시사항은 찾아보기에 대한 일반 원칙과, VSAM 데이터 세트의 여러 유형에 대한 특정 정보를 다룹니다. BDAM 데이터 세트 찾아보기에 관한 특정 정보는 706 페이지의 『BDAM 데이터 세트로부터 레코드 찾아보기』의 내용을 참조하십시오.

프로시저

- 찾아보기를 시작하려면 **STARTBR** 명령을 사용하십시오. **STARTBR** 명령은 찾아보기의 시작 위치만 식별합니다. 즉 레코드를 검색하지 않습니다. 직접 읽기의 경우와 동일

한 방식으로 특정 레코드를 식별하고, RIDFLD 옵션을 사용하여 레코드 ID를 지정하십시오. 699 페이지의 『직접 읽기(READ 명령 사용)』의 내용을 참조하고, 다음 고려사항도 주의하십시오.

1. 파일 시작에 찾아보기를 위치시키려면 KSDS 또는 ESDS의 경우 16진 0의 RIDFLD를 지정하십시오. 표준 ESDS의 경우 RBA 옵션도 지정하십시오. 확장 ESDS의 경우 XRBA 옵션을 지정하십시오.
 2. 파일의 시작에 찾아보기를 위치시키려면 RRDS의 경우 RIDFLD 옵션과 RRN 옵션을 사용하여 1의 RRN을 지정하십시오.
 3. KSDS의 경우에만 파일의 시작에 찾아보기를 위치시키기 위한 대체 방법으로 GENERIC, GTEQ 및 KEYLENGTH(0) 옵션을 지정할 수 있습니다. 해당 값이 사용되지 않는 경우에도 RIDFLD 키워드가 필요하며, 명령이 완료한 후 CICS는 1의 일반 키 길이를 사용 중입니다. KEYLENGTH(0) 옵션을 갖는 찾아보기 명령은 항상 KEYLENGTH(1) 및 2진 0의 1바이트의 부분 키가 지정된 것처럼 취급됩니다.
 4. 역방향 찾아보기를 준비하여 찾아보기를 파일의 마지막 레코드에 위치시키려면 X'FF' 문자의 RIDFLD를 지정하십시오. 이것은 KSDS, ESDS 또는 RRDS의 찾아보기 시작에 적용됩니다. 표준 ESDS의 경우 RBA 옵션을 지정하십시오. 확장 ESDS의 경우 XRBA 옵션을 지정하십시오. RRDS의 경우 RRN 옵션을 지정하십시오.
 5. KSDS의 찾아보기 시작의 경우, RIDFLD 옵션을 사용하여 일반 키를 지정할 수 있습니다. 그러나, 일반 키를 사용하는 경우 파일을 통해 앞쪽으로만 찾아볼 수 있으며 뒤쪽으로는 불가능합니다.
 6. KSDS의 찾아보기 시작의 경우 EQUAL(같은 키) 및 GTEQ(크거나 같은 키) 옵션을 사용할 수 있으며, 이들은 **READ** 명령에서와 같은 의미를 갖습니다. 그러나 GTEQ 옵션이 STARTBR 명령의 기본값인 반면, EQUAL이 **READ** 명령의 기본값입니다. **STARTBR** 명령은 지정된 키 또는 해당 키가 존재하지 않는 경우 첫 번째 키에 위치하기 원한다고 가정합니다.
 7. RRDS의 찾아보기 시작의 경우 GTEQ(크거나 같은 키) 옵션이 STARTBR 명령의 기본값입니다. 이 옵션이 적용될 때 지정된 RRN이 존재하지 않으면 포인터는 더 큰 키를 갖는 첫 번째 레코드로 설정됩니다. GTEQ 옵션은 RRDS의 경우 직접 **READ** 명령에 효과가 없으며, **READ** 명령이 존재하지 않는 RRN을 지정하는 경우 NOTFND 상태가 리턴됨을 주의하십시오.
- 레코드를 통해 찾아보려면 **READNEXT** 명령을 사용하십시오. **READNEXT** 명령은 **STARTBR** 명령에 의해 지정되는 시작점으로부터 순차적으로 레코드를 읽습니다. 직접 읽기와 유사한 방식으로 동작합니다. 699 페이지의 『직접 읽기(READ 명령 사용)』의 내용을 참조하고, 다음 고려사항도 주의하십시오.
 1. 검색되는 각 레코드의 ID를 리턴하지만 찾아보기를 다시 위치시키기를 원하지 않는 한 필드를 설정하지 않는 방법을 CICS에 제공하려면 RIDFLD 옵션을 포함하십시오. 각 **READNEXT** 명령의 완료 시에, CICS가 검색한 레코드의 전체 키를 리

턴합니다. **STARTBR** 명령을 더 짧은 일반 키와 함께 사용하는 경우에도 반드시 전체 키와 같은 길이의 필드를 제공하십시오.

2. 찾아보기를 다시 위치시키기 위해 **RIDFLD** 옵션을 사용하여 키를 제공하려는 경우 **KEYLENGTH** 옵션만 필요합니다. 그렇지 않으면, **STARTBR** 명령에서 또는 찾아보기의 마지막 재배치 동안 설정된 대로 현재 키 길이가 사용됩니다.
 3. **SYSID** 옵션이 지정된 원격 파일의 경우, **RIDFLD** 옵션이 키를 파일 제어로 전달 중인 경우 **KEYLENGTH** 옵션을 지정해야 합니다. 원격 파일을 찾아보고 있는 경우, **READNEXT** 또는 **READPREV** 명령에 대해 **KEYLENGTH** 옵션이 필수입니다.
 4. 직접 읽기의 경우와 같이 애플리케이션이 제공하는 영역으로(**INTO** 옵션을 사용할 때) 또는 **CICS**가 제공하는 저장영역으로(**SET** 옵션을 사용할 때) 레코드를 읽을 수 있습니다. 후자의 경우에 **SET** 포인터로 주소 지정되는 **CICS** 저장영역은 찾아보기의 다음 조작까지 또는 찾아보기 종료, 동기점 또는 태스크 종료 중 가장 빠른 것이 발생할 때까지 계속 유효합니다.
- 파일에서 위쪽으로 찾아보려면 **READNEXT** 명령 대신 **READPREV** 명령을 사용하십시오. **READPREV** 명령은 **READNEXT** 명령과 유사하지만, 레코드를 현재 위치부터 순차적으로 위쪽으로 읽는 점이 예외입니다. 언제든지 한 방향에서 다른 방향으로 전환할 수 있습니다. 다음 고려사항을 참고하십시오.
 1. 한 방향에서 다른 방향으로 전환할 때, 위치를 바꾸지 않으면 동일한 레코드를 두 번 검색합니다.
 2. **STARTBR** 명령 바로 뒤에 **READPREV** 명령을 실행하는 경우 **STARTBR** 명령 **RIDFLD**가 데이터 세트에 존재하는 레코드의 키를 지정해야 하며, 그렇지 않으면 **NOTFND** 상태가 발생합니다. 이 응답을 피하기 위해 **STARTBR** 명령 뒤에 **READNEXT** 명령을 실행하여 데이터 세트에 존재하는 레코드를 리턴할 수 있습니다. 그런 다음 **READPREV** 명령을 실행하십시오. 이는 **READNEXT** 명령이 검색한 동일한 레코드를 검색합니다.
 3. **KSDS**의 경우 **STARTBR** 명령에서 일반 키를 사용한 경우 **READPREV** 명령을 사용할 수 없습니다. 이 경우에 해당 명령을 사용하려고 시도하면 **INVREQ** 상태가 리턴됩니다.
 - 찾아보기가 시작한 현재 찾아보기 위치를 변경하려면(찾아보기 재배치), **RESETBR** 명령, **READNEXT** 명령 또는 **READPREV** 명령을 사용하십시오.
 1. **VSAM**의 경우, 다음 **READNEXT** 또는 **READPREV** 명령을 실행할 때 **RIDFLD**에서 값을 설정하여 찾아보기를 재배치할 수 있습니다.
 - **RIDFLD**를 설정하여 찾아보기를 재배치할 때, 레코드 ID는 이전 **STARTBR** 또는 **RESETBR** 명령에서와 같은 양식(키, **RBA**, **XRBA** 또는 **RRN**)를 가져야 합니다. 키를 사용하거나 16진 0을 사용하여 파일의 시작을 표시하거나, **KSDS**의 경우 **GENERIC**, **GTEQ** 및 **KEYLENGTH(0)** 옵션을 사용하여 파일 시작을 표시할 수 있습니다. **KEYLENGTH(0)** 옵션을 사용하는 경우, 값이 사용되지 않

더라도 RIDFLD 키워드가 필요하며 명령이 완료된 후 CICS가 1의 일반 키 길이를 사용 중임을 주의하십시오. (X'FF' 문자는 **READNEXT** 또는 **READPREV** 명령을 사용한 재배치에 사용될 수 없습니다.)

- **READPREV** 명령에서 KEYLENGTH를 지정하여 일반 키의 길이를 변경할 수 있는데, 이는 현재 일반 키 길이와 다르며 전체 길이와 같지 않습니다. 이 방식으로 일반 키의 길이를 변경하는 경우 RIDFLD 옵션에 의해 지정된 일반 키로 재배치합니다.

또한 찾아보기의 특성을 변경하려는 경우 **RESETBR** 명령을 대신 사용하십시오.

2. **RESETBR** 명령은 **STARTBR** 명령과 동일한 방식으로 새 찾아보기 위치를 지정할 수 있습니다. 다음을 수행할 수 있습니다.

- 특정 레코드를 식별합니다.
 - 16진 0의 키를 사용하여 파일의 시작을 표시합니다.
 - X'FF' 문자의 키를 사용하여 파일 끝을 표시합니다.
 - KSDS의 경우, GENERIC, GTEQ, KEYLENGTH(0) 옵션을 사용하여 파일의 시작을 표시하십시오. KEYLENGTH(0) 옵션을 사용하는 경우, 값이 사용되지 않더라도 RIDFLD 키워드가 필요하며 명령이 완료된 후 CICS가 1의 일반 키 길이를 사용 중임을 주의하십시오.
- 찾아보기의 특성을 변경하려면(예를 들어, 정확한 일치 대신 일반 키를 사용하여 검색하려면) **RESETBR** 명령을 사용하십시오. 이 명령을 사용하여 다음을 수행할 수 있습니다.
- 키의 양식을 키에서 RBA로 변경합니다.
 - 일반 및 전체 키 사이 또는 『동일』 및 『크거나 같음』 검색 사이에서 전환합니다.
- 특정 조건에서는 707 페이지의 『스킵 순차 처리』에서 설명한 대로 사용자가 키를 변경할 때 CICS가 VSAM 순차 생략 처리를 사용합니다.
- 찾아보기를 종료하려면 **ENDBR** 명령을 사용하십시오. 이 명령의 경우 RIDFLD가 없습니다. 파일의 마지막 레코드를 지나 찾아보려고 시도하면 ENDFILE 상태가 발생합니다. 동일한 파일에 대한 갱신 조작(**READ UPDATE**, RIDFLD를 갖는 **DELETE** 또는 **WRITE** 명령)를 수행하기 전에 **ENDBR** 명령을 실행해야 합니다. 그렇지 않으면 사용자 자신의 트랜잭션 내에서 교착상태를 포함한 예측 불가능한 결과를 얻습니다. 또한 **SYNCPPOINT** 및 **SYNCPPOINT ROLLBACK** 명령을 사용하여 찾아보기를 종료할 수도 있습니다. 내재된 동기점이 태스크 종료 시에 발생하는 경우 찾아보기도 종료합니다.

BDAM 데이터 세트로부터 레코드 찾기

레코드 식별 필드는 데이터 세트에 정의된 주소 지정 메소드를 준수하는 블록 참조(예: TTR 또는 MBBCCHHR)를 포함해야 합니다. 처리는 지정된 블록으로 시작하고 찾아보기를 종료할 때까지 각 후속 블록을 계속합니다.

데이터 세트가 블록화 레코드편을 포함하면, 처리는 첫 번째 블록의 제1 레코드에 시작하고, 레코드 식별 필드의 목차에 상관없이, 각 추후 레코드를 계속합니다.

다시 말하면, CICS는 레코드를 식별하기 위해 TTR 또는 RIDFLD의MBBCHHR 서브필드에 보관된 정보만을 사용합니다. 물리적 키와 관련 레코드와 같은 다른 모든 정보 또는 논리적 키를 무시합니다.

READNEXT 명령 뒤에, CICS는 RIDFLD를 검색된 레코드의 전체 ID로 갱신합니다. 예를 들어, 찾아보기는 차단된, 키 지정 데이터 세트의 첫 번째 레코드로 시작되고 논리 키에 의한 블록 해제가 수행된다고 가정해 보십시오. STARTBR 명령을 실행하기 전에 첫 번째 블록의 TTR(주소 지정 메소드로 가정)를 레코드 ID 필드에서 놓으십시오. 첫 번째 READNEXT 명령 뒤에 레코드 ID 필드는 X'0000010504'를 포함할 수도 있습니다. 여기서 X'000001'은 TTR 값을 나타내고, X'05'는 블록 키(길이 1) 및 X'04'는 논리 레코드 키를 나타냅니다.

이제 차단된, 키 지정되지 않은 데이터 세트가 상대 레코드 블록 해제를 사용하여 검색되는 중이고, 세 번째 상대 트랙에서 두 번째 물리적 블록의 두 번째 레코드가 명령에 의해 판독된다고 가정하고, 첫 번째 블록의 TTR(주소 지정 메소드로 가정됨)을 레코드 ID 필드에 놓으십시오. 첫 번째 READNEXT 명령 뒤에, 애플리케이션으로 리턴될 때 레코드 ID 필드에는 X'00020201'이 포함됩니다. 여기서 X'0002'는 트랙을 표시하고, X'02'는 블록을 표시하고, X'01'은 0과 관련된 블록에서 레코드 수를 표시합니다.

참고: 블록화된 데이터 세트를 찾을 때 STARTBR 명령에 DEBREC 및 DEBKEY 옵션을 지정하십시오. 이를 사용하면 CICS가 RIDFLD에서 올바른 내용을 리턴할 수 있습니다. STARTBR 명령에 DEBREC을 지정하면 상대 레코드 번호가 리턴됩니다. STARTBR 명령에 DEBKEY를 지정하면 논리적 레코드 키가 리턴됩니다.

차단된 파일을 찾아볼 때 DEBREC 또는 DEBKEY를 생략하지 마십시오. 이를 생략하면 논리 레코드는 블록에서 검색되고, 길이 매개변수는 논리 레코드 길이와 동일하게 설정됩니다. **그러나 RIDFLD는 레코드의 전체 ID로 갱신되지 않습니다. 이 메소드를 사용하지 마십시오.**

차단된 BDAM 데이터 세트에서 읽을 때 DEBREC 또는 DEBKEY 옵션을 생략하는 경우에 발생하는 것과 이를 비교하십시오. 이 경우 **전체** 블록을 검색하고, 길이 매개변수는 블록의 길이와 동일하게 설정됩니다.

DEBKEY 또는 DEBREC 옵션이 지정된 원격 BDAM 파일의 경우 KEYLENGTH(명시적으로 지정된 경우)는 키의 전체 길이(즉, 모든 지정된 서브필드)여야 합니다.

스킵 순차 처리

가능할 때, CICS는 VSAM 『스킵 순차』 처리를 사용하여 찾아보기를 가속화합니다. 스킵 순차 처리는 RIDFLD가 키 양식에 지정된 경우에 파일의 정방향 찾아보기에만 적용됩니다.

CICS는 사용자가 READNEXT 명령의 RIDFLD에서 키 값을 증가하고 다른 키 관련 스펙(예: KEYLENGTH)을 작성할 때 이를 사용합니다. 이 상황에서, VSAM은 처음부터 다시 위치를 지정하기 보다 색인을 통해 앞으로 읽어서 다음으로 원하는 레코드를 찾습니다. 이 방법은 검색 중인 레코드가 상대적으로 서로 근접하지만 반드시 인접하지 않은 경우에 더 빠릅니다. 레코드가 큰 파일에서 멀리 떨어져 있는 경우 반대 효과가 있을 수 있습니다. 다시 위치 지정 중인 키가 파일에서 더 높이 있고, 장기 색인 스캔을 초래할 수 있음을 아는 경우, 처음부터 다시 위치 지정을 강요하는 RESETBR 명령을 사용할 것으로 고려할 수 있습니다.

CICS 명령을 사용하여 레코드 갱신

레코드를 갱신하려면 우선 UPDATE 옵션을 지정하는 파일 제어 읽기 명령 중 하나를 사용하여 이를 검색해야 합니다. 레코드는 직접 읽기와 동일한 방식으로 식별됩니다.

이 태스크 정보

KSDS 또는 ESDS에서 레코드는(직접 읽기로) 기본 또는 정의된 경로를 참조하는 파일 정의를 사용하여 액세스할 수 있습니다. RLS 모드로 열린 파일의 경우, 레코드가 이미 활성 잠금을 사용하여 VSAM에서 잠금된 경우 요청이 대기하지 않도록 EXEC CICS 명령에 NOSUSPEND 옵션뿐만 아니라 UPDATE 옵션도 지정할 수 있습니다.

애플리케이션을 사용하여 수정한 후에, 레코드는 쓰기 중인 레코드를 식별하지 않는 REWRITE 명령을 사용하여 데이터 세트에 다시 쓰여집니다. 작업 단위 내에서 각 REWRITE 명령은 공통 키워드(TOKEN)를 사용하여 이전의 READ UPDATE와 연관되어야 합니다. 언제든지 미해결 TOKEN을 사용하지 않는 한 개의 READ UPDATE를 포함할 수 있습니다. 작업 단위 내에서 여러 개의 동시 갱신 요청을 시도하면 TOKEN을 사용하지 않는 동일한 데이터 세트에서의 시도는 CICS에서 차단됩니다. 레코드를 다시 쓰거나 삭제하지 않고 READ UPDATE에서 보류된 문자열을 해제하려면 UNLOCK 명령을 사용하십시오. UNLOCK 명령은 READ 명령용으로 확보된 모든 CICS 저장영역을 해제하고 READ 명령으로 보류된 VSAM 자원을 해제합니다. TOKEN이 UNLOCK 명령으로 지정된 경우, CICS는 TOKEN에 동일한 값을 포함하는 미해결 READ UPDATE와 이를 일치시키려고 합니다. (TOKEN 옵션에 대한 자세한 설명은 709 페이지의 『TOKEN 옵션』의 내용을 참조하십시오.)

갱신 또는 비갱신 명령에 대해 RIDFLD 옵션으로 지정한 레코드 ID를 사용하여 검색되는 레코드를 식별해야 합니다. READ UPDATE 명령이 완료되면 바로 RIDFLD 데이터 영역은 애플리케이션에서 다시 사용할 수 있습니다. REWRITE 및 UNLOCK 명령은 RIDFLD 옵션을 사용하지 않습니다.

파일이 RLS 모드로 열리는 경우 찾아보기 조작 중에 검색되는 레코드는 찾아보기 중에만 갱신됩니다(712 페이지의 『찾아보기에서 레코드 갱신 및 삭제(VSAM RLS만)』 참조). 비RLS 모드로 열리는 파일은 애플리케이션이 찾아보기를 종료하고 READ

UPDATE 명령으로 필요한 레코드를 읽은 후 갱신을 수행해야 합니다. READ UPDATE 명령을 실행하기 전에 찾아보기를 종료하지 않으면 교착상태가 발생할 수 있습니다.

갱신되는 레코드(직접 읽기인 경우)는 애플리케이션에서 제공되는 저장영역의 영역 또는 CICS로 설정되는 저장영역에서 읽을 수 있습니다. READ UPDATE 명령의 경우, CICS SET 저장영역은 다음 REWRITE, UNLOCK, DELETE(RIDFLD를 사용하지 않음) 또는 SYNCPOINT 명령 중에서 가장 먼저 발생하는 명령 실행까지 유효하게 유지됩니다.

KSDS의 경우 레코드의 기본 키는 레코드가 수정될 때 수정되면 안됩니다. 이와 유사하게 경로를 사용하여 갱신이 수행 중인 경우, 레코드 식별에 사용되는 대체 키도 수정하면 안됩니다. 다른 대체 키는 수정 가능합니다. 파일 정의에서 가변 길이 레코드가 허용되며 레코드 길이는 변경 가능합니다.

FROM 옵션으로 작성되는 레코드를 지정하십시오. FROM 옵션은 작성되는 레코드가 포함되는 기본 저장영역의 영역을 지정합니다. 일반적으로 이 영역은 애플리케이션이 소유하는 저장영역의 일부입니다. REWRITE 명령을 사용하여 FROM 영역은 일반적으로(필수는 아님) READ UPDATE 명령의 해당 INTO 영역과 동일합니다.

ESDS, 고정 길이 RRDS 또는 고정 길이 KSDS의 레코드 길이는 갱신 시에 변경되면 안됩니다. 그렇지만 레코드 길이는 가변 길이 레코드를 사용하여 KSDS를 다시 작성하는 경우에 변경 가능합니다.

고정 길이 레코드 포함으로 CICS에 정의된 파일인 경우 다시 작성 중인 레코드 길이는 **원래 길이와 동일해야 합니다**. 그렇지만 고정 길이 레코드가 포함된 파일에 쓰는 경우, LENGTH 옵션을 지정할 필요가 없습니다. 그렇게 하면 해당 값이 정의된 값에 대해 확인되고 값이 일치하지 않으면 LENGERR 조건이 수신됩니다.

가변 길이 레코드의 경우, READ UPDATE 및 REWRITE 명령 모두에 LENGTH 옵션을 지정해야 합니다. 길이는 VSAM에 정의된 최대보다 크면 안됩니다.

TOKEN 옵션

TOKEN 옵션은 갱신 명령에 대한 올바른 읽기에 대해 CICS에서 제공되는 태스크 내의 고유 값으로, 연관된 REWRITE, DELETE 또는 UNLOCK 명령으로 이를 CICS에 리턴합니다. 태스크에 의해 갱신되는 파일마다, 어느 때든지 TOKEN 옵션을 지정하지 않는 UPDATE 옵션으로 하나의 미해결 읽기 요청만 가질 수 있습니다.

갱신 명령에 대해 읽기와 함께 TOKEN 옵션을 포함하고, 연관된 REWRITE, DELETE 또는 UNLOCK 명령에서 토큰을 지정하여, 동일한 태스크로 동일한 데이터 세트에서 여러 개의 동시 갱신사항을 수행할 수 있습니다. RLS 이외 모드에서 액세스되는 파일에 대해, 요청과 연관되는 TOKEN과 관계없이 두 개 이상의 레코드가 동일한 CI에서 갱신되는 경우 동시 갱신 세트가 실패합니다. 또한 찾아보기에서 지정된 REQID에 대

해 단 하나의 토큰만 올바른 상태로 유지됩니다. 이 토큰은 마지막 READNEXT 또는 READPREV 명령에서 리턴된 토큰입니다(712 페이지의 『찾아보기에서 레코드 갱신 및 삭제(VSAM RLS만)』 참조).

TOKEN 옵션을 포함하는 갱신 요청에 대해 읽기를 function ship할 수 있습니다. 그러나 이 키워드를 인식하지 않는 CICS 제품군의 멤버에 TOKEN을 지정하여 요청을 function ship하는 경우 요청이 실패합니다.

조건부 VSAM 파일 갱신 요청

RLS 모드에서 열린 파일에 대한 파일 제어 갱신 요청 시 잠금을 받은 즉시 조건부 요청을 작성하여 잠금을 기다리는 것을 피할 수 있습니다. 요청에 NOSUSPEND 옵션을 지정하여 이를 수행할 수 있습니다. 또 다른 태스크가 이미 활성 잠금을 보유 중인 경우에는 CICS는 요청을 큐 처리하는 대신 RECORDBUSY 조건을 리턴합니다.

READ, READNEXT, READPREV, WRITE, REWRITE 및 DELETE 명령에 NOSUSPEND를 지정할 수 있습니다.

LOCKED 및 RECORDBUSY 응답 사이를 구별하는 것이 중요합니다.

- LOCKED 응답은 요청이 보존된 잠금에 의해 잠겨 있는 레코드에 액세스하려고 시도할 때 발생합니다.
- RECORDBUSY 응답은 요청이 활성 잠금에 의해 잠긴 레코드에 액세스하려고 시도할 때 발생합니다. 이는 DEADLOCK에 의해 초래될 수 있고 이 경우 재시도가 작동하지 않을 수도 있음을 기억하십시오. 조건을 해결하기 위해 롤백하거나 롤백하지 않고 SYNCPOINT를 실행해야 할 수도 있습니다.

참고: CICS가 RECORDBUSY 응답을 리턴하기 전에 1초 이상이라도 NOSUSPEND 대기를 지정하는 요청.

요청에 NOSUSPEND를 지정하지 않으면 CICS는 레코드가 이미 활성 잠금에 의해 잠기지 않은 경우 잠금을 기다리게 합니다. NOSUSPEND를 지정하는 경우에는 요청은 레코드가 활성 잠금에 의해 잠긴 경우 RECORDBUSY 응답을 수신합니다.

보유된 잠금에 의해 잠긴 레코드에 대해 NOSUSPEND 옵션을 사용하거나 사용하지 않고 요청을 실행하면 CICS는 LOCKED 응답을 리턴합니다.

NOSUSPEND가 파일 제어 명령에서 작동하는 방법은 NOSUSPEND가 다른 CICS 명령에서 작동하는 방법과 비교할 때 약간의 차이가 있습니다. HANDLE CONDITION(RECORDBUSY)을 실행하는 경우에는 NOSUSPEND가 후속 파일 제어 요청에서 가정되도록 하지 않습니다. HANDLE CONDITION(QBUSY)을 지정하면 NOSUSPEND가 명시적으로 지정되지 않더라도 후속 트랜지언트 데이터 명령에서 가정되게 합니다.

BDAM 데이터 세트로부터 레코드 갱신

블록 해제를 지정하는 REWRITE 명령에서 변수 차단되거나 차단 해제된 BDAM 레코드의 레코드 길이를 변경할 수 없습니다. REWRITE 명령에서도 정의되지 않은 형식 BDAM 레코드의 레코드 길이를 변경할 수 없습니다.

참고: 차단된 BDAM 레코드가 갱신을 위해 읽힐 때, CICS은 포함하는 블록의 독점 제어를 지속합니다. 또는, 독점 제어(UNLOCK 명령에 의해) 릴리스되기 전에 첫 번째(rewrite 명령에 의해) 갱신되기 전에 블록으로부터 2번째 레코드를 읽기 위한 시도가 교착을 야기시킵니다.

CICS 명령을 사용한 레코드 삭제

파일에서 레코드 또는 레코드 그룹을 삭제하려면 DELETE 명령을 사용하십시오. 먼저 갱신을 위해 레코드를 검색해야 합니다. RLS 모드에서 열린 파일의 경우, 찾아보기 중에 레코드를 삭제할 수 있습니다.

이 태스크 정보

전체 키가 DELETE 명령과 함께 제공되는 경우, 해당 키를 갖는 단일 레코드가 삭제됩니다. 그러므로 데이터 세트가 고유하지 않은 대체 키를 허용하는 대체 색인 경로를 사용하여 액세스되고 있는 경우 해당 키를 갖는 첫 번째 레코드만 삭제됩니다. 삭제 후, 동일한 대체 키를 갖는 추가 레코드가 DUPKEY 조건을 얻는 경우 사용자도 해당 조건을 얻기 때문에 추가 레코드가 존재하는지 여부를 압니다.

699 페이지의 『직접 읽기(READ 명령 사용)』에서 설명하는 NOSUSPEND 옵션은 파일을 갱신하기 위해 CICS 찾아보기 명령을 사용할 때 해당 명령에 적용됩니다.

프로시저

- ESDS에서 절대 레코드를 삭제할 수 없습니다.
- KSDS 또는 RRDS의 단일 레코드를 삭제하려면 다음 세 방법 중 하나를 사용하십시오.
 1. READ UPDATE 명령을 사용하여 갱신을 위해 검색한 후, RIDFLD 옵션을 지정하지 말고 DELETE 명령을 실행하십시오.
 2. RIDFLD 옵션을 지정하고 DELETE 명령을 실행하십시오.
 3. RLS 모드에서 열린 파일의 경우, UPDATE 옵션과 함께 READNEXT 또는 READPREV 명령으로 레코드를 검색한 후 DELETE 명령을 실행하십시오. 이 방법은 712 페이지의 『찾아보기에서 레코드 갱신 및 삭제(VSAM RLS만)』에서 설명합니다.
- KSDS 또는 RRDS에서 레코드 그룹을 삭제하려면 DELETE 명령과 함께 일반 키를 사용하십시오. 단일 레코드를 삭제하는 대신, 그의 키가 일반 키와 일치하는 파일의 모든 레코드가 단일 명령으로 삭제됩니다. 액세스가 대체 색인 경로를 이용하는

경우 삭제된 레코드는 모두 그의 대체 키가 일반 키와 일치하는 레코드입니다. 그러나 KEYLENGTH 값이 전체 키의 길이와 동일한 경우(중복 키가 허용되는 경우에도) 이 방법은 사용할 수 없습니다. NUMREC 옵션이 명령에 포함되는 경우 삭제되는 레코드 수가 애플리케이션에 리턴됩니다.

- GENERIC 옵션을 갖는 **DELETE** 명령은 주의해서 사용해야 합니다. 지정된 일반 키가 매우 큰 수의 레코드와 일치하는 경우, CICS 작업 단위는 동기점까지 아주 큰 수의 레코드 잠금을 보유할 것입니다. 이것은 CICS 리전 내에서 저장영역 문제를 유발하거나, 파일이 RLS 파일인 경우 결합 기능 자원의 과도한 사용을 유발하고 다른 시스템에 영향을 줄 수 있습니다. 대신, 레코드의 각 범위가 삭제된 후 일련의 일반 삭제(더 적은 레코드와 일치하는 일반 키를 사용하여) 및 동기점 사용을 고려하십시오.

찾아보기에서 레코드 갱신 및 삭제(VSAM RLS만)

RLS 모드에서 액세스하는 파일의 경우, READNEXT 또는 READPREV 명령에 UPDATE 옵션을 지정하고 DELETE 또는 REWRITE 명령을 실행하여 레코드를 갱신 또는 삭제할 수 있습니다.

이 태스크 정보

찾아보기 명령이 TOKEN을 리턴하는 경우, TOKEN은 다음 찾아보기 요청까지 올바르게 유지됩니다. TOKEN에 대해 동일한 값을 지정하는 REWRITE, DELETE 또는 UNLOCK 명령이나, 동일한 REQID를 지정하는 READNEXT, READPREV 또는 ENDBR 명령에 의해 TOKEN이 무효화됩니다. UPDATE 및 TOKEN 옵션이 있는 많은 READNEXT 명령을 실행하는 경우, TOKEN은 서로를 무효화하고 마지막 TOKEN만 사용 가능합니다. (TOKEN 옵션에 대한 자세한 설명은 709 페이지의 『TOKEN 옵션』의 내용을 참조하십시오.)

찾아보기에서 UPDATE 옵션을 사용할 때 다음 규칙을 따릅니다.

- RLS 모드에서 파일에 액세스하는 경우에만 찾아보기 내에서 UPDATE를 지정할 수 있습니다. RLS가 아닌 모드에서 액세스되는 파일에 대해 UPDATE를 지정하면, CICS는 INVREQ 상태를 리턴합니다.
- STARTBR 또는 RESETBR 명령이 아니라 READNEXT 및 READPREV 명령에서만 UPDATE를 지정할 수 있습니다.
- CICS는 찾아보기 순서에서 단 하나의 TOKEN만 지원하고, 각 READNEXT 또는 READPREV 명령의 TOKEN 값은 이전 값을 겹쳐씹습니다.
- 동일한 찾아보기 내에서 갱신 및 갱신 이외 요청을 혼합할 수 있습니다.
- READNEXT, DELETE 또는 UNLOCK 명령에서 해당하는 READNEXT 또는 READPREV 명령으로 리턴될 TOKEN을 지정해야 합니다.

UPDATE 잠금

READNEXT 또는 READPREV 명령에 UPDATE를 지정하면 독점 잠금이 생깁니다. 브라우저 내에서 이러한 독점 잠금의 지속 기간은 애플리케이션이 취하는 조치와 파일이 복구 가능한지 여부에 따라 다릅니다.

- 파일이 복구 가능하고 브라우저에서 갱신을 위해 읽기에 의해 획득된 마지막 레코드를 DELETE 또는 REWRITE하기로 결정하면(연관된 토큰을 사용) VSAM 독점 잠금은 UOW 완료까지 활성인 채로 남아 있습니다. 즉, 성공적인 동기점 또는 롤백 때까지입니다.
- 파일이 복구 불가능하고 획득된 마지막 레코드를 DELETE 또는 REWRITE하기로 결정한 경우에는 다음 번에 ENDBR 명령을 실행하거나 후속 READNEXT 또는 READPREV 명령을 실행할 때 잠금이 해제됩니다. 이는 682 페이지의 『RLS 레코드 레벨 잠금』에서 보다 자세히 설명됩니다.
- 마지막 레코드 읽기를 갱신하지 않기로 결정한 경우에는 CICS는 프로그램이 찾아보기 중에 또 다른 READNEXT 또는 READPREV 명령을 실행하거나 찾아보기를 종료할 때 독점 잠금을 해제합니다.

참고: UNLOCK 명령은 찾아보기 작업 중에 획득된 레코드에 대해 VSAM이 보유 중인 RLS 독점 잠금을 해제하지 않습니다. 찾아보기 내에 UNLOCK은 마지막 요청이 리턴한 TOKEN을 무효화합니다. 찾아보기에서 또 다른 READNEXT 또는 READPREV는 또한 이전 READNEXT 또는 READPREV UPDATE 명령에 의해 판독되는 레코드의 TOKEN을 무효화합니다. 그러므로 특정 레코드를 갱신하지 않기로 결정하는 애플리케이션에서 UNLOCK을 사용할 필요가 없습니다.

CICS 명령을 사용한 레코드 추가

WRITE 명령을 사용하여 VSAM 데이터 세트의 파일에 새 레코드를 추가하십시오. 새 레코드는 항상 애플리케이션이 제공하는 영역으로부터 기록되어야 합니다.

이 태스크 정보

이들 지시사항은 레코드 쓰기를 위한 일반 원칙 및 VSAM 데이터 세트의 여러 유형에 대한 특정 정보를 다룹니다. BDAM 데이터 세트에 레코드 추가에 관한 특정 정보는 715 페이지의 『레코드를 BDAM 데이터 세트에 추가』의 내용을 참조하십시오.

WRITE 명령을 사용할 때,

프로시저

- KSDS에 레코드를 추가하려면 RIDFLD(레코드 식별 필드) 옵션을 사용하여 레코드의 기본 키를 지정하십시오. 레코드의 기본 키는 데이터 세트에서 레코드가 삽입될 위치를 식별합니다. 키가 레코드의 일부이지만, CICS는 애플리케이션이 키를 개별적으로 지정해야 합니다.

- ESDS 기본 데이터 세트에 레코드를 추가하려면 레코드가 파일의 끝에 추가되어야 합니다. ESDS에서 기존 레코드 사이에 레코드를 삽입할 수 없습니다. 조작이 완료된 후, 레코드가 위치한 파일의 상대 바이트 주소가 애플리케이션으로 리턴됩니다.
- 대체 색인 경로를 통해 KSDS 또는 표준 ESDS에 레코드를 추가하려면 RIDFLD 옵션을 사용하여 대체 키를 지정하십시오. KSDS의 경우, 레코드는 기본 키에 의해 판별되는 위치에서 데이터 세트에 삽입됩니다. ESDS의 경우 레코드는 데이터 세트의 끝에 위치합니다.

참고: 확장 ESDS 데이터 세트는 대체 색인을 지원하지 않습니다.

- RRDS에 레코드를 추가하려면 RIDFLD 옵션을 사용하여 상대 레코드 번호를 지정하십시오. 레코드는 RRN에 대응하는 위치에 있는 파일에 저장됩니다.
- 필요한 경우 KEYLENGTH 옵션을 사용하여 RIDFLD 옵션에서 지정한 키의 길이를 명시하십시오.
 1. 레코드 식별 필드에 대해 RBA, XRBA 또는 RRN을 지정하고 RBA, XRBA 또는 RRN 옵션을 지정하는 경우, KEYLENGTH 옵션은 필수가 아닙니다.
 2. 다른 모든 상황에서는 KEYLENGTH 옵션이 필수입니다. 데이터 세트에 대해 정의된 길이와는 다른 KEYLENGTH를 지정하는 경우 INVREQ 조건이 발생합니다.
- 기록될 레코드를 포함하는 기본 저장영역을 지정하려면 FROM 옵션을 사용하십시오. 일반적으로 이 영역은 애플리케이션이 소유하는 저장영역의 일부입니다.
- 필요한 경우 LENGTH 옵션을 지정하십시오.
 1. 고정 길이 레코드를 갖는 파일에 쓸 때는 LENGTH 옵션이 필수가 아닙니다. CICS는 클러스터 정의에 지정된 길이를 기록될 레코드의 길이로 사용합니다. LENGTH 옵션을 지정하는 경우, 해당 값이 정의된 값에 대해 확인되고 값이 일치하지 않는 경우 LENGERR 조건을 얻습니다.
 2. 가변 길이 레코드를 갖는 파일에 쓸 때는 항상 LENGTH 옵션을 포함하십시오. 지정된 값이 클러스터 정의에서 허용되는 최대값을 초과하는 경우, 명령이 실행될 때 LENGERR가 발생합니다. LENGERR는 가변 길이 레코드를 갖는 파일에 액세스할 때 LENGTH 옵션이 누락되는 경우에도 발생합니다.
- 파일이 RLS 모드에서 열리고 레코드가 이미 잠겨있을 가능성이 있는 경우, NOSUSPEND 옵션을 지정할 수 있습니다. NOSUSPEND 옵션은 레코드가 활성 잠금을 사용하여 VSAM에 의해 잠긴 경우 요청이 대기하지 않게 합니다. 685 페이지의 『잠금에 대한 활성 및 보유 상태』에서 활성 잠금에 대해 설명합니다.
- KSDS, ESDS 또는 경로에 추가할 둘 이상의 레코드가 있고 후속 요청의 키가 오름차순으로 있는 경우 MASSINSERT 옵션 사용이 성능을 향상시킵니다. (성능 향상은 키가 오름차순 순서로 있는 경우에만 얻어집니다.)
 1. 순서의 각 WRITE 명령에서 MASSINSERT 옵션을 지정하십시오.

2. MASSINSERT 조작이 완료되고 모든 레코드가 기록된 후, UNLOCK 명령을 실행하여 모든 레코드가 파일에 기록되고 해당 위치가 해제되도록 하십시오. 항상 동일한 데이터 세트에 대해 갱신 조작(읽기 갱신, RIDFLD를 사용한 삭제 또는 쓰기)을 수행하기 전에 UNLOCK 명령을 실행하십시오. 그렇지 않은 경우 교착상태를 포함하여 예측 불가능한 결과를 얻을 수 있습니다. UNLOCK 명령을 실행하지 않는 경우, MASSINSERT 조작은 동기점이 발행될 때 또는 태스크 종료 시에 완료됩니다.

참고: READ 명령이 불완전한 MASSINSERT 조작에 의해 추가된 레코드를 반드시 검색하지는 않습니다.

MASSINSERT에 대한 자세한 정보는 716 페이지의 『유효 데이터 세트 운영』의 VSAM 데이터 세트를 참조하십시오.

ESDS에 쓰기 위한 CICS 잠금

ESDS에 대한 CICS 쓰기 작업은 RLS 및 비RLS 모드 액세스 둘 모두의 경우 단일 스레드입니다. 그러나 직렬화를 위해 보관된 잠금은 비RLS 모드와 비교할 때 RLS 모드 액세스가 약간 더 깁니다. ESDS 파일에 새 레코드를 추가하는 이러한 트랜잭션의 태스크 우선순위를 늘려서 오버헤드의 증가 가능성을 보상할 수 있습니다.

ESDS RLS 모드를 비RLS 모드에서 전환할 때 새 레코드를 추가하는 이러한 트랜잭션의 제한시간이 늘어나는 것을 볼 수도 있습니다.

ESDS에 RLS 모드 액세스를 사용하면 가용성 문제를 일으킬 수도 있습니다. CICS 리전이 ESDS에 쓰는 동안 실패하면 데이터 세트는 CICS 리전이 재시작될 때까지 잠길 수도 있습니다. ESDS에 RLS 모드 액세스를 사용하지 않는 것이 좋습니다.

레코드를 BDAM 데이터 세트에 추가

BDAM 데이터 세트에 추가된 레코드는 미정의 또는 변수 길이(키 지정 또는 키 지정되지 않음), 키 지정 고정 길이, 키 지정되지 않은 고정 길이 또는 차단된 변수 길이일 수 있습니다.

이 태스크 정보

레코드를 BDAM 데이터 세트에 추가할 때 다음을 염두에 두십시오.

- 미정의 또는 변수 길이 레코드(키 지정 또는 키 지정되지 않음)를 추가할 때 각 레코드가 추가될 트랙을 지정해야 합니다. 트랙에서 공간이 사용 가능한 경우에는 레코드는 이전에 마지막으로 작성된 레코드를 뒤따라 작성되고, 레코드 번호는 레코드의 레코드 ID 필드의 『R』 부분에 놓입니다. 트랙 스펙은 상대 블록을 제외한 임의의 형식일 수 있습니다. 구역이 있는 10진 상대 형식을 사용하는 경우에는 레코드 번호는 레코드 ID 필드의 일곱 번째와 여덟 번째 위치에 2바이트 구역이 있는 10진수로서 리턴됩니다.

확장 검색 옵션을 사용하면 지정된 트랙에서 사용 가능한 공간이 없는 경우 레코드를 또 다른 트랙에 추가할 수 있습니다. 레코드가 추가되는 위치는 사용 중인 레코드 ID 필드에서 애플리케이션으로 돌아갑니다.

길이가 정의되지 않은 레코드를 추가할 때는 LENGTH 옵션을 사용하여 레코드의 길이를 지정하십시오. 미정의 레코드를 검색할 때는 애플리케이션은 해당 길이를 찾아야 합니다.

- 키 지정 고정 길이 레코드를 추가할 때는 먼저 레코드를 추가할 수 있는 더미 레코드 또는 『슬롯』으로 데이터 세트를 포맷해야 합니다. 'X'FF' 키로 더미 레코드를 표시합니다. 데이터의 첫 번째 바이트에는 레코드 번호가 포함됩니다.
- 키 지정되지 않은 고정 길이 레코드를 추가할 때 레코드 ID 필드에 블록 참조를 제공하십시오. 새 레코드는 지정된 위치에 작성되어 해당 위치의 이전 내용을 제거합니다.
- 키 지정 고정 길이 레코드를 추가할 때 트랙 정보는 더미 키와 레코드를 검색하는 데에만 사용되며 이는 발견되면 새 키와 레코드와 대체됩니다. 새 레코드의 위치는 레코드 ID 필드의 블록 참조 서브필드에서 애플리케이션으로 리턴됩니다.

예를 들어, 다음 ID 필드가 있는 레코드의 경우:

```
0 3 0 ALPHA
T T R KEY
```

검색은 상대 트랙 3에서 시작됩니다. 제어가 애플리케이션으로 리턴되면 레코드 ID 필드는 다음과 같습니다.

```
0 4 6 ALPHA
```

레코드가 이제 상대 트랙 4에서 레코드 6임을 보여줍니다.

- 변수 길이 차단된 레코드를 추가할 때는 각 레코드에 4바이트 레코드 설명 필드(RDF)를 포함해야 합니다. 첫 번째 2바이트는 레코드 길이를 지정합니다(4바이트 RDF 포함). 나머지 2바이트는 0으로 구성됩니다.

유효 데이터 세트 운영

데이터베이스와 데이터 세트 운영의 효율성은 CICS 시스템의 성능에서 중요한 요소입니다. VSAM 및 BDAM 데이터 세트를 사용하여 응답 시간을 최대화하려면 다음 정보를 사용하십시오.

VSAM에서 효율성 및 그로 인한 응답 시간에 미치는 기본 영향은 직렬 사용 자원(레코드 키, 제어 간격 및 문자열) 및 저장영역 사용 및 프로세서 오버헤드에 대한 경험에서 나옵니다. 이러한 상황에서는 일반적으로 한 영역에서 개선사항을 작성하면 다른 영역이 희생될 수도 있습니다.

VSAM 데이터 세트

VSAM 데이터 세트를 사용하여 경합 지연을 최소화하려면 다음을 수행하십시오.

- VSAM 자원이 독점 사용을 위해 예약한 시간을 최소화합니다. 독점 사용 큐에 넣기는 CICS 및 VSAM이 동시 갱신을 방지하는 방법입니다.

676 페이지의 『RLS 모드에서 파일에 액세스』에 설명된 VSAM 레코드 레벨 공유를 사용하는 경우에는, VSAM은 갱신이 요청된 레코드를 다른 트랜잭션이 동시에 갱신하려고 시도할 수 없도록 잠급니다. 파일이 복구 가능한 경우에는 VSAM은 다음 번 동기점에서 잠금을 해제합니다. 파일이 복구 가능하지 않으면 VSAM은 요청이 완료될 때 잠금을 해제합니다. 파일의 복구성은 통합된 카탈로그 기능(ICF) 카탈로그에 정의됩니다.

VSAM 레코드 레벨 공유를 사용하지 않으면 CICS는 기본 클러스터 레코드 키를 사용하여 갱신 요청을 일련화합니다. 요청된 레코드를 포함하는 전체 VSAM 제어 간격(CI)은 레코드에서 개별 명령(예: UPDATE 옵션이 있는 READ 명령)이 실행되는 동안 독점 사용을 위해 보유됩니다. 각 명령이 완료되면 제어 간격이 해제되고 요청된 레코드만이 잠긴 상태로 남아 있습니다. 복구 불가능한 데이터 세트의 경우 레코드의 VSAM 독점 사용과 CICS 독점 사용은 둘 모두 VSAM 기간에서 갱신 요청이 완료될 때 종료됩니다. 예를 들어, REWRITE 명령이 완료될 때입니다. 그러나 복구 가능한 데이터 세트의 경우 CICS 독점 사용은 태스크가 종료되거나 SYNCPOINT 명령을 실행할 때까지는 종료되지 않습니다. 복구 가능성은 데이터 세트 자원 정의에 지정됩니다. FILE 자원 정의에 대한 자세한 정보는 CICS 자원 정의 안내서에 있는 FILE 정의 속성을 참조하십시오.

- 가능한 짧은 시간 동안 VSAM 데이터 세트에 위치를 보유하십시오. 테이블 1은 어떤 명령이 위치를 보유하고 언제 보유가 해제되는지를 보여줍니다.

표 33. 위치를 보유하는 명령 및 보유가 해제되는 시기

명령	다음 시기에 VSAM이 해제함
READ.. UPDATE	REWRITE/DELETE/UNLOCK
WRITE.. MASSINSERT	UNLOCK
STARTBR	ENDBR

VSAM 데이터 세트에서 진행 중인 각 요청에는 하나 이상의 문자열이 필요합니다. 위치를 보유하는 요청은 보유 위치를 해제하라는 명령이 실행될 때까지 문자열을 단단히 묶습니다. 위치를 보유하지 않는 요청은 요청이 완료되는 즉시 문자열을 해제합니다.

VSAM 데이터 세트를 사용할 때 프로세서 오버헤드를 최소화하려면 다음을 수행하십시오.

- 여러 레코드를 순서대로 추가 중이라면 MASSINSERT 옵션을 사용하십시오. 이는 프로세서 오버헤드를 최소화하여 성능을 개선하므로 응답 시간도 개선합니다. ESDS

및 KSDS의 경우 MASSINSERT를 사용하여 레코드를 추가하면 CICS가 순차 VSAM 처리를 사용하게 됩니다. 이는 VSAM이 분할이 요청될 때 제어 간격 내에 레코드를 배치하는 방법을 변경하므로 결과로 분할 수가 적어지고 영향 받은 CI 내에 사용하지 않는 공간이 줄어듭니다.

- 키가 상대적으로 가깝지만 반드시 인접할 필요는 없는 여러 레코드를 순서대로 읽는 중이라면 건너뛰기 순차 처리를 사용하십시오. (건너뛰기 순차 처리는 찾아보기 시작 (STARTBR 명령)으로 시작합니다.) 각 레코드는 READNEXT 명령으로 검색되지만 RIDFLD로 지정되는 키 피드백 영역에는 READNEXT 명령이 실행되기 전에 다음 번 요청되는 레코드의 키가 공급됩니다.
- 키가 공통 문자 문자열로 시작되는 레코드 그룹을 삭제할 때는 DELETE 명령에 GENERIC 옵션을 사용하십시오. CICS는 내부적으로 일반 DELETE를 최적화합니다.

BDAM 데이터 세트

BDAM 데이터 세트는 VSAM보다는 덜 효율적입니다. CICS가 몇몇 단일 스레드 처리를 수행하고 BDM 데이터 세트 요청을 처리하기 위해 몇몇 운영 체제 대기를 실행하기 때문입니다. 그러므로 가능하면 BDM 데이터 세트 대신에 상대 레코드 VSAM 데이터 세트 또는 상대 바이트 주소(RBA)로 주소 지정된 ESDS(Entry-Sequenced Data Set)를 사용해야 합니다.

BDAM 데이터 세트를 갱신 모드에서 사용 중인 경우에는 선택하는 데이터 세트 무결성을 사용하여 성능이 상당한 영향을 받는다는 점을 알고 있어야 합니다.

BDAM 데이터 세트의 파일 제어 테이블 SERVREQ 피연산자에 독점 제어를 지정하는 경우에는 CICS는 운영 체제에 동시 갱신을 방지하도록 요청합니다. 그러나 이는 상당한 오버헤드와 관련됩니다.

효율적인 찾아보기(비RLS 모드에서)

데이터 세트 찾아보기는 시스템 자원을 독점할 수 있는 많은 출력 화면을 생성하는 트랜잭션에서 종종 출력의 소스입니다. DELAY 또는 SUSPEND 명령을 실행하여 다른 태스크가 제어를 가져오도록 허용할 수 있습니다.

긴 찾아보기는 시스템에 심각한 로드를 가져오고 다른 트랜잭션을 잠그며, 전체 응답 시간을 늘리는 것은 물론 BMS, 태스크 제어, 터미널에 필요한 오버헤드를 초래합니다.

그 이유는 제어 철학이 터미널 운영자가 몇몇 데이터 레코드에 액세스하는 트랜잭션을 시작하고 정보를 처리하며 운영자에게 결과를 리턴한다는 가정을 기반으로 하기 때문입니다. 이 프로세스는 CICS가 멀티태스킹을 수행할 수 있게 하는 무수한 대기를 수반합니다. 그러나 CICS는 인터럽트 구동 멀티태스킹 시스템이 아니므로 처리에 비해 적은 양의 I/O를 수반하는 태스크가 우선순위에 관계없이 시스템을 독점할 수 있습니다. 제어 간격에서 레코드가 많은 데이터 세트 찾아보기는 딱 이러한 트랜잭션입니다.

DELAY 또는 SUSPEND 명령을 정기적으로 실행하여 이를 방지할 수 있으므로 다른 태스크가 제어를 가져올 수 있습니다. 찾아보기가 페이지가 매겨진 출력을 생성하는 경우, 1020 페이지의 『페이지 빌드 및 라우팅 작업』에 제한된 방법 중 하나로 트랜잭션을 분할할 것을 고려해야 합니다.

제 8 부 터미널 제어

CICS API(Application Programming Interface)에는 터미널, 터미널 제어 명령 및 BMS(Basic Mapping Support)(BMS)와 통신하기 위한 두 가지 명령 세트가 포함되어 있습니다.

터미널 제어 인터페이스

터미널 제어는 두 가지 중 한층 기본적인 것입니다. 추가 프로그래밍이 필요하지만, 유연성과 기능을 제공합니다. 특히, 터미널 제어 레벨에서 코딩하는 경우 사용자 애플리케이션에서 디바이스 데이터 스트림을 빌드해야 합니다.

터미널 제어 명령은 다양한 디바이스에 적용되어, 명령이 지원하는 터미널과 터미널을 제어하는 액세스 방법에 대한 프로그램의 감도를 줄입니다. 명령 자체 외에도, CICS는 데이터 변환, 입력 및 출력의 동기화, 그리고 터미널 또는 논리 장치로(부터) 읽거나 쓰기 위해 필요한 세션 제어를 제공합니다. 이는 복잡하고 서로 다른 개별적인 통신 액세스 방법의 API로부터 사용자를 보호하는데 도움이 됩니다.

BMS

BMS는 상위 언어 레벨에서 터미널과 통신할 수 있도록 합니다. BMS가 데이터를 형식화하므로, 사용자는 데이터 스트림의 세부사항을 몰라도 됩니다. 따라서 초기에 코딩하고 유지보수하는 것이 더 쉬워집니다(특히, 애플리케이션이 새 유형의 터미널을 지원해야 하는 경우). 그러나 BMS 경로 길이는 더 길어지고(BMS 자체는 터미널 제어를 사용함) BMS는 터미널 제어가 지원하는 모든 터미널 유형을 지원하지는 않습니다. BMS는 애플리케이션 개발에서 기본 맵핑 지원에서 설명됩니다.

BMS는 일부 유연성과 기능이 덜하지만, 터미널 제어보다 특정 통신 디바이스 및 머신의 특성에서 한층 보호합니다.

이 섹션에서는 다음에 대해 설명합니다.

- 723 페이지의 제 24 장 『터미널 액세스 방법 지원』
- 725 페이지의 제 25 장 『터미널 제어 명령』
- 731 페이지의 제 26 장 『데이터 전송 명령 사용』
- 733 페이지의 제 27 장 『디바이스 제어 명령』
- 735 페이지의 제 28 장 『터미널 디바이스 지원』
- 739 페이지의 제 29 장 『터미널 정보 찾기』
- 743 페이지의 제 30 장 『SNA 사용』
- 747 페이지의 제 31 장 『순차 터미널 지원 사용』
- 751 페이지의 제 32 장 『일괄처리 데이터 상호 교환 사용』

제 24 장 터미널 액세스 방법 지원

z/OS용 CICS Transaction Server, 버전 5 릴리스 3 는 순차 디바이스에 의해 시뮬레이션된 터미널에 대해 GDDM 및 SAM(Sequential Access Method)을 사용하여 그래픽 터미널에 대해 z/OS Communications Server, BGAM(Basic Graphics Access Method) 액세스 방법에 대한 인터페이스를 통해 직접 터미널을 지원합니다.

CICS는 액세스 방법을 통하기 보다는 운영 체제 서비스를 통해 터미널로 운영 체제 콘솔도 지원합니다. 콘솔에 대한 터미널 제어 인터페이스는 다른 터미널에서도 동일하지만(특정 콘솔에는 특정 제한사항이 있을 수 있음) BMS는 사용할 수 없습니다. *CICS* 자원 정의 안내서에서 CICS에 의해 지원되는 터미널의 전체 목록을 찾을 수 있습니다.

제 25 장 터미널 제어 명령

터미널 제어 명령은 기본 데이터 전송, 제어 기능, 터미널에 대한 정보 검색을 위해 사용됩니다. 터미널 제어는 또한 특수 디바이스 그룹 명령을 제공합니다.

이 절에 설명된 명령은 명령을 실행하는 태스크의 프린시פל 기능에만 적용됩니다. 해당 기능은 다음 중 하나입니다.

- SAM을 통하여 연결된 디바이스
- z/OS Communications Server를 통해 연결된 LU 유형 0, 1, 2, 3 또는 4.

참고: 이 절에서는 대체 또는 프린시פל 기능으로 보내는지 여부에 관계없이 프로그램 간 통신에 대해 다루지 않습니다. 이는 별도의 매뉴얼에서 다루지며, APPC 명령은 *CICS 분산 트랜잭션 프로그래밍 안내서*에서 다뤄집니다.

터미널 제어 명령은 네 그룹으로 나뉩니다.

- 기본 데이터 전송 명령: RECEIVE, SEND 및 CONVERSE
- 디바이스 제어를 전송하거나, 전송을 동기화하거나, 세션을 종료하거나, 유사한 제어 기능을 수행하는 명령
- 터미널에 대해 알려주는 명령: ASSIGN 및 INQUIRE.
- 특수 디바이스 그룹 명령: 일괄처리 데이터 교환(BDI) 명령

전송/수신 모드

터미널 및 논리 장치는 모두 “반이중, 플립 플롭” 모드에서 작동합니다. 이는 특히 주어진 순간에 대화의 한 파트너가 전송 모드에 있고(데이터 또는 제어 명령을 전송할 수 있음) 다른 파트너가 수신 모드(수신으로 제한됨)에 있음을 의미합니다.

이 프로토콜은 공식적으로 정의되고 SNA용 z/OS Communications Server에 의해 시행됩니다. CICS는 모든 조작이 동일하지 않도록 다른 액세스 방법 하에 접속된 터미널에 대해 동일한 규칙을 준수하지만, 하드웨어 및 액세스 방법 둘 다 다르게 작동합니다.

터미널이 태스크의 프린시פל 기능인 경우 해당 대화 파트너는 태스크입니다. 태스크와 연관되지 않을 때, 해당 대화 파트너는 CICS의 터미널 제어 구성요소입니다. Communications Server 환경의 태스크 사이에, 대화는 두 파트너 중 하나가 첫 번째로 전송할 수 있는 중립 상태에 남아 있습니다. 보통 터미널이 첫 번째로 이동하여, 태스크를 초기화하는 요청되지 않은 입력을 전송합니다(270 페이지의 『태스크 시작 방법』 참조).

이 전송은 또한 전송/수신 역할을 되돌립니다. 이 후에, 터미널은 수신 모드에 있고 접속된 태스크에 의해 표시되는 CICS는 전송 모드에 있습니다. 태스크가 대화 방향을 명시적으로 변경할 때까지, 태스크는 실행하는 SEND 수에 상관없이 전송 모드에서 시작하여 유지됩니다. 태스크를 수신 모드에 둘 수 있는 한 가지 방법은 SEND 명령에 INVITE 옵션을 지정하는 것입니다. INVITE가 있는 SEND 후에, 태스크는 수신 모드에 있고 다시 터미널로 전송하기 전에 RECEIVE를 실행해야 합니다. INVITE를 앞에 지정하지 않고 RECEIVE를 실행하여 수신 모드에 태스크를 둘 수도 있습니다. INVITE는 전송을 최적화합니다.

요청되지 않은 입력으로 초기화된 태스크에서 첫 번째 RECEIVE 명령은 전송/수신 모드 측면에서 계산하지 않습니다. 관련된 입력 메시지는 전송되지 오래되었기 때문입니다. (이 명령이 태스크를 시작했습니다.) 이 RECEIVE는 단지 메시지가 태스크에 액세스 가능하도록 만들고, 관련된 EIB 필드를 설정합니다.

ATI 태스크(CICS에 의해 자동으로 초기화된)는 요청되지 않은 입력으로 시작된 태스크와 같이 전송 모드에서도 시작합니다.

태스크가 Communications Server 또는 네트워크 오류가 발생할 때 비터미널 조작을 정상적으로 실행하고 수행하는 경우, 태스크는 오류를 인식하지 못하고 다음 터미널 제어 요청을 시도할 때까지 처리를 계속합니다. 이 시점에서 태스크는 TERMERR을 수신합니다. 태스크가 추가 터미널 제어 요청을 실행하지 않으면, TERMERR 또는 ABEND를 수신하지 않습니다.

터미널에 대한 경합

CICS는 프린시פל 기능으로 필요한 터미널이 사용 가능하게 될 때 바로 자동 태스크 초기화(ATI)에 대한 요청을 만족시킵니다.

그러나 태스크가 터미널에서 종료하고, CICS가 해당 터미널에 대한 ATI 요청을 가지고 있는 경우, 요구되지 않은 입력에 의해 특정 태스크를 초기화하려는 터미널 사용자와 ATI 태스크를 초기화하려는 CICS 사이에 경합이 있을 수 있습니다. 이러한 상황에서, CICS는 항상 자체를 경합 패자로 설정합니다. 즉, 이전 트랜잭션의 종료 이후에 요구되지 않은 입력을 터미널이 충분히 빠르게 전송하는 경우, CICS는 태스크를 작성하여 이를 처리하고 ATI 요청 이행을 지연합니다. 이는 의도적인 것으로, 경합 상황에서 사용자에게 우선순위를 제공합니다.

RETURN IMMEDIATE

간혹 사용자 개입을 허용하지 않고 터미널에서 연속으로 특정 태스크 순서를 실행해야 합니다. CICS는 태스크를 종료하는 RETURN 명령의 IMMEDIATE 옵션으로, 이를 수행할 수 있는 방법을 제공합니다.

RETURN IMMEDIATE로, CICS는 터미널에서의 입력을 허용하지 않고 해당 터미널에서 태스크에 대한 다른 대기 요청을 지지하기 직전에 TRANSID 옵션에 이름 지정

된 트랜잭션을 실행하도록 태스크를 초기화합니다. 기존 태스크는 데이터를 새 태스크로 전달할 수 있습니다. 사용자가 요청하지 않은 입력으로 태스크를 초기화했었던 것처럼 RECEIVE로 이 데이터에 액세스하지만, 입출력(I/O)은 발생하지 않습니다. 요청하지 않은 입력에 의해 초기화된 태스크의 첫 번째 RECEIVE처럼, 이 RECEIVE는 전송/수신 상태에 영향이 없습니다. 단지 전달된 데이터는 새 태스크에 대해 사용 가능하도록 만듭니다. 터미널이 브래킷 프로토콜을 사용하는 경우(746 페이지의 『브래킷 프로토콜로 인터럽트 방지』에 설명된 것처럼), CICS는 보통 때와 같이 첫 번째 태스크의 끝에서 대괄호를 종료하지 않습니다. 그 대신 다음 태스크를 포함하도록 대괄호로 계속합니다. 결국, 태스크 사이의 대괄호 끝에서 키보드 자동 열기가 발생하지 않습니다.

상황에 벗어나서 말하기

사용자가 “talk”(키 및 전송)할 때와, “listen”(출력 대기)할 때 사용자에게는 일반적으로 명확합니다. 애플리케이션이 이를 명확하게 만들기 때문입니다.

3270 표시장치와 많은 다른 터미널에서, 이 규칙을 강화하기 위해 사용자가 전송한 후 키보드가 잠깁니다. 키보드는 태스크가 잠금을 해제할 때까지 잠금 상태로 유지되고, 일반적으로 RECEIVE 이전의 SEND에 대해, 또는 태스크의 마지막 SEND에 대해서는 수행되지 않습니다. 이는 사용자가 프로토콜을 중단하기 위해 특별히 어떤 것을 수행해야 함을 의미합니다(키보드 재설정 키 누름).

사용자가 이를 수행할 경우 발생하는 사항은? z/OS Communications Server 아래에서 연결된 터미널의 경우, 이 프로토콜을 위반하면 읽기 우선 큐 처리가 강요되지 않는 한 태스크가 이상 종료됩니다(코드 ATCV). 읽기 우선 큐 처리는 논리 장치 및 태스크가 언제든지 전송 및 수신할 수 있도록 합니다. CICS는 태스크에서 필요로 할 때까지 임시 저장영역에 입력 메시지를 저장합니다. 태스크 끝까지 읽혀지지 않은 입력은 버려집니다. 읽기 우선 큐 처리는 트랜잭션 레벨에서 적용됩니다. (이는 트랜잭션 규칙이 실행되는 PROFILE의 RAQ 옵션에서 지정됩니다.) 읽기 우선 큐 처리는 LU 유형 4 디바이스에만 적용되며, 원래 호환성을 위해 제공되었으므로, 트랜잭션이 BTAM 연결 및 z/OS Communications Server 연결 터미널 둘 다를 같은 방식으로 지원하도록 합니다. BTAM은 더 이상 지원되지 않으므로, 읽기 우선 큐 처리는 더 이상 사용하면 안 됩니다.

순차 터미널은 전송/수신 규칙에서 다른 터미널과 다릅니다. 입력은 사전 준비된 파일이므로, 태스크가 입력 메시지를 요청할 때마다 CICS가 이 메시지를 제공하여, 프로토콜을 중단할 수 없습니다. 입력이 부적절하게 준비되거나, 태스크가 처리하도록 프로그래밍된 입력이 아닌 경우, 해당 입력과 동기화되지 않는 태스크는 입력을 일찍 소모하거나 일부 입력을 읽지 못할 수 있습니다.

인터럽트

z/OS Communications Server는 전송하려고 함을 해당 파트너에 알리기 위해 수신 모드에서 터미널에 대한 메커니즘을 제공합니다. 이는 Communications Server에서의 신호 데이터 플로우로, 태스크의 다음 SEND, RECEIVE 또는 ISSUE DISCONNECT 명령에서 발견됩니다.

신호 플로우가 발생할 때, CICS는 SIGNAL 상태를 발생하고 EIB에서 EIBSIG를 설정합니다. SIGNAL 상태에 대한 CICS 기본 조치는 무시하는 것입니다. 신호가 효과를 갖도록 하려면, 태스크가 먼저 신호를 감지한 후 대화 방향을 변경하여 이 신호를 지지해야 합니다.

3270 표시장치 터미널 및 일부 다른 터미널에서, ATTENTION 키는 인터럽트를 생성하는 키입니다. 모든 터미널에 이 기능이 있는 것은 아니지만, z/OS Communications Server에서, 바인드 이미지도 이에 대한 지원을 표시해야 하며 그렇지 않으면 z/OS Communications Server는 인터럽트를 무시합니다.

터미널 대기

SEND 명령에 WAIT 옵션을 사용할 때, 출력 조작이 완료될 때까지 CICS는 제어를 사용자 태스크로 리턴하지 않습니다.

태스크가 WAIT를 지정하지 않고 SEND 명령을 실행할 때, CICS는 사용자 태스크에 대해 해당되는 전체 터미널 처리 또는 전송을 최적화하기 위해 출력의 전송을 지연할 수 있습니다. 이를 수행할 때, CICS는 출력 메시지를 저장하고 태스크가 디스패치 가능하도록 만들므로, 계속 실행 가능합니다. ISSUE COPY 및 ISSUE ERASE 명령(출력도 전송하는)은 WAIT 없이 비슷하게 작동합니다.

WAIT 옵션을 사용하는 경우, 출력 조작이 완료될 때까지 CICS는 제어를 사용자 태스크에 리턴하지 않습니다. 이 대기로 응답 시간 및 메모리 점유에 대해 영향이 수반되는 태스크의 경과 시간이 길어지지만, 계속하기 전에 SEND에 오류가 발생했는지 여부를 태스크가 알도록 합니다. 이 대기 중 일부를 방지하고 SEND 후에 수행할 처리가 있는 경우 운영 완료를 계속 확인할 수 있습니다. WAIT 없이 SEND를 실행하고, 처리를 계속한 후, SEND 결과를 알아야 하는 지점에서 WAIT TERMINAL 명령을 실행합니다.

입력 전송이 필요한 RECEIVE 명령을 실행할 때 태스크는 항상 대기합니다. RECEIVE를 완료하려면 먼저 전송이 발생해야 하기 때문입니다. 그러나 RECEIVE가 터미널 입출력(I/O)에 해당되지 않는 경우가 있습니다. 요청되지 않는 터미널 입력에 의해 초기화된 태스크의 첫 번째 RECEIVE가 가장 빈번한 예이지만, 다음 절에 설명된 대로 다른 예도 있습니다.

또한 사용자 터미널을 포함하는 명령을 실행할 때, CICS는 새 명령을 처리하기 전에 이전 명령이 완료되는지(여기에는 지연된 전송이 포함됨) 확인합니다.

제 26 장 데이터 전송 명령 사용

세 명령이 태스크의 프린시פל 기능인 터미널 또는 논리 장치로부터 데이터를 전송합니다.

명령은 다음과 같습니다.

RECEIVE

터미널에서 데이터를 읽습니다.

SEND

터미널에 데이터를 씁니다.

CONVERSE

터미널에 데이터를 쓰고, 입력을 기다리며, 입력을 읽습니다.

CONVERSE는 본질적으로 SEND 및 RECEIVE의 조합이고, 보통 RECEIVE가 뒤에 있는 SEND와 같습니다. 특정 경우에 SEND 및 RECEIVE 대신 CONVERSE를 사용해야 합니다(예를 들어, 구조화된 필드 데이터를 특정의 3270 디바이스로 전송하는 경우). 다른 경우에는 CONVERSE가 제공되지 않으므로 SEND 및 RECEIVE를 사용해야 합니다. 이러한 경우는 737 페이지의 표 36에 명시되어 있습니다.

SEND, RECEIVE 및 CONVERSE 명령은 참조 -> 애플리케이션 개발에서 CICS 명령 요약에서 전체적으로 설명됩니다. 이 명령들은 디바이스 그룹별로 구분됩니다. 여러 디바이스 및 액세스 방법에 대한 옵션이 다르기 때문입니다. 735 페이지의 제 28 장 『터미널 디바이스 지원』에서는 특정 디바이스에 대해 사용할 디바이스 그룹을 알려줍니다.

RECEIVE에서 가져오는 사항

터미널이 전송한 것과 애플리케이션이 수신한 것 둘 다를 의미하기 위해 입력 메시지 및 전송 용어를 사용합니다. 가장 일반적인 터미널 유형의 경우, 애플리케이션은 터미널이 전송한 데이터를 수신합니다. 예를 들어 3270 표시장치는 단일 엔티티로 해당 버퍼에서 변경된 것을 전송하고, 터미널과 연관된 태스크는 단일 RECEIVE 명령에 대한 응답으로 보통 전체 메시지를 가져옵니다.

그러나 입력 메시지 및 물리적 전송은 항상 같지 않고, 어느 하나의 일대일 관계가 RECEIVE 명령에 영향을 줄 수 있는 몇 가지 요인이 있습니다.

입력 체이닝

일부 SNA 디바이스는 긴 입력 메시지를 여러 개의 물리적 전송으로 분리합니다. 이러한 프로세스를 체이닝이라고 합니다.

CICS는 태스크와 연관된 터미널이 정의된 방법에 따라 구성요소 전송을 단일 입력 메시지로 어셈블하거나 개별적으로 나타냅니다. 이는 체이닝된 입력 메시지를 읽어야 하는 RECEIVE 수에 영향을 줍니다. 인바운드 체이닝에 대한 세부사항은 743 페이지의 『입력 데이터 체이닝』에 설명되어 있습니다.

논리 메시지

일부 디바이스가 긴 입력을 여러 전송으로 분리하는 것처럼, 다른 디바이스는 짧은 입력을 블록화하여 단일 전송으로 이 입력을 전송합니다.

다시, CICS는 블록화를 해제하는 옵션을 CICS 또는 수신 프로그램에 제공합니다. 이 선택사항은 또한 단일 RECEIVE에 대해 가져오는 데이터 양에 영향을 줍니다. (이 주제에 대한 추가사항은 744 페이지의 『논리 레코드 처리』의 내용을 참조하십시오.)

NOTRUNCATE 옵션

입력 데이터의 길이가 프로그램 예상보다 길 경우 RECEIVE당 하나의 입력 메시지 규칙에 대한 예외가 발생합니다.

이러한 상황이 발생하고 RECEIVE 명령에 NOTRUNCATE가 지정되는 경우, CICS는 초과 데이터를 저장하고 터미널에 대해 해당되는 읽기가 없는 프로그램에서 후속 RECEIVE 명령을 충족시키기 위해 이를 사용합니다. NOTRUNCATE를 사용 중인 경우, EIB의 필드 EIBCOMPL이 설정(즉, X'FF'로 설정)될 때까지 RECEIVE 명령을 실행하십시오. CICS는 사용 가능한 입력 메시지가 더 이상 없을 때 EIBCOMPL을 끕니다.

NOTRUNCATE가 없을 경우, CICS는 초과 데이터를 버리고, EIBCOMPL을 켜 후 LENGERR 조건을 발생합니다. LENGTH 옵션(제공한 경우)에서 이름 지정된 데이터 영역에서 절단 이전의 데이터 진짜 길이를 보고합니다.

인쇄 키

CICS 시스템에 “인쇄” 키로 정의된 PA 키가 있는 경우, 보통의 전송/수신 순서에 대한 다른 예외가 발생할 수 있습니다.

태스크가 RECEIVE를 실행하고, 사용자가 응답으로 “인쇄(print)” 키를 누르면, CICS는 이 입력을 인터셉트하고 요청을 이행하기 위해 필요한 처리를 수행한 후, 다시 터미널을 수신 모드에 둡니다. 사용자는 원래의 RECEIVE를 충족시키기 위해 다른 입력을 전송해야 합니다. (“인쇄” 키에 대한 자세한 정보는 894 페이지의 『표시장치 화면 인쇄』에서 CICS 인쇄 키를 참조하십시오.)

제 27 장 디바이스 제어 명령

데이터 전송 명령에 추가로, 터미널의 CICS API는 데이터가 아니라 명령어나 제어 명령을 터미널이나 이를 제어하는 액세스 메소드에 전송하는 명령 시리즈를 포함합니다.

이러한 명령은 해당 기능에 대한 간략한 설명과 함께 다음 테이블에 나열됩니다. 이러한 모든 명령이 모든 터미널에 적용되는 것은 아니고, 일부 다른 양식이 서로 다른 터미널에 적용됩니다. 735 페이지의 제 28 장 『터미널 디바이스 지원』의 내용을 참조하십시오.

다음 테이블의 터미널은 항상 명시적으로 언급된 곳을 제외하고는 항상 명령을 실행하는 태스크의 프린시פל 기능입니다. 이는 일반적으로는 터미널로 간주되지 않는 논리 장치 유형일 수 있습니다.

표 34. 터미널과 논리 장치의 제어 명령

명령	조치
FREE	터미널이 현재 태스크가 종료되기 전에 다른 태스크에서 사용될 수 있도록 태스크에서 터미널을 해제합니다.
ISSUE COPY	TERMINAL 옵션에 이름 지정된 터미널의 버퍼 내용을 태스크가 소유하는 터미널의 버퍼로 복사합니다. 두 터미널 모두 3270이어야 합니다.
ISSUE DISCONNECT	CICS와 터미널 간의 세션의 종료를 태스크 종료 시로 스케줄합니다.
ISSUE EODS	데이터 세트 끝 기능 관리 헤더를 전송합니다(3650 해독기 논리 장치 전용).
ISSUE ERASEUP	터미널의 모든 비보호 필드를 지웁니다(3270 디바이스 전용).
ISSUE LOAD	PROGRAM 옵션에서 이름 지정된 프로그램을 로드하도록 터미널에 지시합니다(3650 해독기 논리 장치 전용).
ISSUE PASS	터미널을 CICS로부터 분리 및 LUNAME 옵션에 이름 지정된 z/OS Communications Server 애플리케이션으로의 전송을 실행 태스크의 끝에 스케줄합니다.
ISSUE PRINT	터미널 버퍼를 인쇄 요청에 적합한 첫 번째 프린터로 복사합니다(3270 표시 장치 전용).
WAIT SIGNAL	터미널이 SIGNAL 데이터 플로우 명령을 전송할 때까지 실행 태스크를 일시중단합니다.
WAIT TERMINAL	이전의 터미널 작업이 완료될 때까지 실행 태스크를 일시중단합니다.

제 28 장 터미널 디바이스 지원

하드웨어 및 액세스 방법 감도는 터미널과 통신하기 위해 BMS를 사용하는 것과 터미널 제어 명령을 사용하는 것 사이의 주요 차이점 중 하나입니다. BMS는 일부 기능을 잃으면서 하드웨어 종속성으로부터 애플리케이션을 보호하지만 터미널 제어는 모든 기능을 제공합니다.

전체 기능이 제공되더라도 모든 터미널 제어 명령이 모든 디바이스에 적용되는 것은 아닙니다. 일부 명령의 경우, 적용되는 옵션 및 발생 가능한 예외 조건을 판별하려면 터미널 유형을 알아야 합니다. 일부 명령의 경우 사용 중인 액세스 방법도 알아야 합니다. 다음 테이블은 임의 터미널에 적용되는 명령 및 액세스 방법 조합을 보여줍니다. 여러 유형의 터미널을 지원해야 하는 경우, 739 페이지의 제 29 장 『터미널 정보 찾기』에서 설명하는 대로 명령을 사용하여 자체 프린시פל 기능을 소유하는 태스크 유형을 알 수 있습니다.

테이블을 사용하려면 표 35의 첫 번째 열에서 프로그램이 지원해야 하는 터미널 유형을 검색하십시오. 두 번째 열의 값을 사용하여 737 페이지의 표 36 첫 번째 열에서 해당하는 명령 그룹을 찾으십시오. 이 테이블의 두 번째 열은 액세스 방법을 보여주고 세 번째는 사용할 수 있는 명령을 보여줍니다. 명령 자체는 참조 -> 애플리케이션 개발에서 CICS 명령 요약에서 자세히 설명합니다. 명령 버전이 둘 이상인 경우, 테이블에는 사용할 명령도 표시됩니다. 이 정보는 명령 다음에 괄호 안에 표시됩니다.

표 35. CICS에서 지원되는 디바이스

디바이스	명령 사용 대상
2260, 2265	2260
3101(TWX 33/35로 지원)	3767
3230	3767
3270 표시장치, 3270 프린터	LU 유형 2/3
3270 표시장치, 3270 프린터(비SNA)	3270 논리
3270 표시장치, 3270 프린터	3270 표시장치
SCS 프린터	SCS
3600 파이프라인 모드	3600 파이프라인
3601	3600-3601
3614	3600-3614
3630, 3600(3631, 3632, 3633, 3643, 3604)으로 접속	3600 항목 사용
3641, 3644, 3646, 3647(3767로 접속)	3767
3643(LU 유형 2로 접속)	LU 유형 2/3
3642, 3645(SCS 프린터로 접속)	SCS
3650 해독기 LU	3650 해독기

표 35. CICS에서 지원되는 디바이스 (계속)

디바이스	명령 사용 대상
3650 호스트 통신 LU(3270)	3650-3270
3650 호스트 통신 LU(3653)	3650-3653
3650 호스트 명령 LU(3680, 3684)	3650-3680
3650 해독기 LU	3650 해독기
3650 호스트 통신 LU(3270)	3650-3270
3650 호스트 통신 LU(3653)	3650-3653
3650 호스트 명령 LU(3680, 3684)	3650-3680
3730	3790 전체 기능 또는 조회
3767 대화식 LU	3767
3770 대화식 LU	3767
3770 전체 기능 LU	3790 전체 기능 또는 조회
3770 일괄처리 LU(3771, 3773, 3774)	3770
3790 전체 기능 또는 조회	3790 전체 기능 또는 조회
3790 3270 표시장치 LU	3790 3270 표시장치
3790 SCS 프린터	3790 SCS
3790 3270 프린터	3790 3270 프린터
4700(3600으로 지원)	3600 항목 사용
5280, 3270으로 접속	3270 항목 사용
5520, 3790 전체 기능 LU로 지원	3790 전체 기능 또는 조회
5550(3270으로 지원)	3270 항목 사용
5937(3270으로 지원)	3270 항목 사용
6670	LU 유형 4
DPCX 아래 8130, 8140(3790으로 지원)	3790 전체 기능 또는 조회
8100 DPPX/BASE - 호스트 표시 서비스 또는 호스트 트랜잭션 기능 사용(3790으로 접속)	3790 전체 기능 또는 조회
8100 DPPX/DSC, DPCX/DSC, 8775 접속 포함(3270으로 지원)	LU 유형 2/3
8775	LU 유형 2/3
8815	APPC
Displaywriter, 3270으로 지원	3270 항목 사용
Displaywriter, APPC로 지원	APPC
INTLU(대화식 LU)	3767
PC, PS/2, 3270으로 접속	3270 항목 사용
Scanmaster	APPC
3650 파이프라인으로 지원되는 시리즈/1	3600 파이프라인
3790 전체 기능 LU로 지원되는 시리즈/1	3790 전체 기능 또는 조회
3770으로 지원되는 System/32(5320)	3770 항목 사용
3770으로 지원되는 System/34(5340)	3770 항목 사용
System/34(5340)	System/3
System/36(System/34로 지원)	System/34 항목 사용
System/38(5381), 3770으로 접속	3770 항목 사용

표 35. CICS에서 지원되는 디바이스 (계속)

디바이스	명령 사용 대상
System/38(5381), APPC로 접속	APPC
TWX 33/35 NTO	3767
WTYY NTO	3767

표 36. 디바이스 유형별 터미널 제어 명령

디바이스 그룹 이름	액세스 방법	적용 가능 명령
2260	비 z / O S Communications Server	RECEIVE(2260), SEND(2260), CONVERSE(2260), ISSUE DISCONNECT(기본값), ISSUE RESET
3270 표시장치	비 z / O S Communications Server	RECEIVE(3270 표시장치), SEND(3270 표시장치), CONVERSE(3270 표시장치), ISSUE COPY(3270 표시장치), ISSUE DISCONNECT(기본값), ISSUE ERASEUP, ISSUE PRINT, ISSUE RESET
LU 유형 2/3(3270 SNA)	z / O S Communications Server	RECEIVE(LU 유형 2/3), SEND(LU 유형 2/3), CONVERSE(LU 유형 2/3), ISSUE COPY(3270 논리), ISSUE DISCONNECT(기본값), ISSUE ERASEUP, ISSUE PASS, ISSUE PRINT
3270 논리(3270 비 SNA)	z / O S Communications Server	RECEIVE(3270 논리), SEND(3270 논리), CONVERSE(3270 논리), ISSUE COPY(3270 논리), ISSUE DISCONNECT(기본값), ISSUE ERASEUP, ISSUE PASS, ISSUE PRINT
SCS	z / O S Communications Server	SEND(SCS), CONVERSE(SCS), ISSUE DISCONNECT(기본값), ISSUE PASS
3600 파이프라인	z / O S Communications Server	RECEIVE(3600 파이프라인), SEND(3600 파이프라인), ISSUE DISCONNECT(기본값), ISSUE PASS
3600-3601	z / O S Communications Server	RECEIVE(3600-3601), SEND(3600-3601), CONVERSE(3600-3601), ISSUE DISCONNECT(기본값), ISSUE PASS, WAIT SIGNAL
3600-3614	z / O S Communications Server	RECEIVE(3600-3614), SEND(3600-3614), CONVERSE(3600-3614), ISSUE DISCONNECT(기본값), ISSUE PASS
3650 해독기	z / O S Communications Server	RECEIVE(3650), SEND(3650 해독기), CONVERSE(3650 해독기), ISSUE DISCONNECT(기본값), ISSUE EODS, ISSUE LOAD, ISSUE PASS
3650-3270	z / O S Communications Server	RECEIVE(3650), SEND(3650-3270), CONVERSE(3650-3270), ISSUE DISCONNECT(기본값), ISSUE ERASEUP, ISSUE PASS, ISSUE PRINT
3650-3653	z / O S Communications Server	RECEIVE(3650), SEND(3650-3653), CONVERSE(3650-3653), ISSUE DISCONNECT(기본값), ISSUE PASS

표 36. 디바이스 유형별 터미널 제어 명령 (계속)

디바이스 그룹 이름	엑세스 방법	적용 가능 명령
3650-3680	z / O S Communications Server	RECEIVE(3650), RECEIVE(3790 전체 기능 또는 조회), SEND(3650-3680), SEND(3790 전체 기능 또는 조회), CONVERSE(3650-3680), ISSUE DISCONNECT(기본값), ISSUE PASS
3767	z / O S Communications Server	RECEIVE(3767), SEND(3767), CONVERSE(3767), ISSUE DISCONNECT(기본값), ISSUE PASS, WAIT SIGNAL
3770	z / O S Communications Server	RECEIVE(3770), SEND(3770), CONVERSE(3770), ISSUE DISCONNECT(기본값), ISSUE PASS, WAIT SIGNAL
3790 전체 기능 또는 조회	z / O S Communications Server	RECEIVE(3790 전체 기능 또는 조회), SEND(3790 전체 기 능 또는 조회), CONVERSE(3790 전체 기능 또는 조회), ISSUE DISCONNECT(기본값), ISSUE PASS, WAIT SIGNAL
3790 3270 표시장치	z / O S Communications Server	RECEIVE(3790 3270 표시장치), SEND(3790 3270 표시장 치), CONVERSE(3790 3270 표시장치), ISSUE DISCONNECT(기본값), ISSUE ERASEAUP, ISSUE PASS, ISSUE PRINT
3790 3270 프린터	z / O S Communications Server	SEND(3790 3270 프린터), ISSUE DISCONNECT(기본값), ISSUE ERASEAUP, ISSUE PASS
3790 SCS	z / O S Communications Server	SEND(3790 SCS), ISSUE DISCONNECT(기본값), ISSUE PASS
LU 유형 4	z / O S Communications Server	RECEIVE(LU 유형 4), SEND(LU 유형 4), CONVERSE(LU 유형 4), ISSUE DISCONNECT(기본값), ISSUE PASS, WAIT SIGNAL
외부 제어기(일괄처리 데이터 상호 교환)	z / O S Communications Server	ISSUE ABORT, ISSUE ADD, ISSUE END, ISSUE ERASE, ISSUE NOTE, ISSUE QUERY, ISSUE RECEIVE, ISSUE REPLACE, ISSUE SEND, ISSUE WAIT
기타 모두	z / O S Communications Server	RECEIVE(SNA-용 Communications Server 기본값), SEND(SNA-용 Communications Server 기본값), CONVERSE(SNA-용 Communications Server 기본값), ISSUE PASS
기타 모두	비 z / O S Communications Server	RECEIVE(비 z / OS Communications Server 기본값), SEND(비 z / OS Communications Server 기본값), CONVERSE(비 z / OS Communications Server 기본값)

제 29 장 터미널 정보 찾기

일부 애플리케이션은 둘 이상의 터미널 유형을 지원해야 하며 유형은 별도의 코드가 필요할 만큼 완전히 다른 경우도 있습니다. 이런 프로그램을 작성 중이고 현재 통신 중인 터미널 정렬을 판별해야 하는 경우, ASSIGN 명령으로 이를 찾을 수 있습니다.

ASSIGN은 태스크 실행에 대한 다양한 정보를 리턴하며 여기에는 해당 프린시פל 기능을 설명하는 몇 개의 필드가 포함됩니다. 표 37에서는 터미널 제어 조작에 직접 연관된 정보가 나열됩니다. BMS 및 기타 태스크 관점에 연관된 다른 ASSIGN 옵션이 있습니다. *CICS 애플리케이션 프로그래밍 참조서*에서 모든 ASSIGN 옵션에 대한 세부사항을 확인할 수 있습니다. 테이블의 컬럼 2에서 언급하는 "터미널"은 항상 태스크의 프린시פל 기능입니다.

표 37. 터미널에 대한 ASSIGN 명령 옵션

ASSIGN 옵션	리턴된 정보
ALTSCRNHT ALTSCRNWD	터미널 화면의 대체 높이 및 너비(해당 터미널 정의에서). SCRNHT 및 SCRNWD도 참조하십시오.
APLKYBD	터미널에 APL 키보드가 있는지 여부
APLTEXT	터미널에 APL 텍스트 기능이 있는지 여부
BTRANS	터미널에 배경 투명도 기능이 있는지 여부
COLOR	터미널에 확장 색상 기능이 있는지 여부
DEFSCRNHT DEFSCRNWD	터미널 화면의 기본 높이 및 너비(해당 터미널 정의에서). SCRNHT 및 SCRNWD도 참조하십시오.
DELIMITER	터미널에 대한 데이터 링크 제어 문자(3600 터미널에만 해당)
DESTID DESTIDLENGTH	외부 대상 ID 및 해당 길이(BDI 조작에만 해당)
DSSCS	터미널이 SCS 데이터 스트림 디바이스인지 여부
DS3270	터미널이 3270 데이터 스트림 디바이스인지 여부
EXTDS	터미널이 "조회 구조 필드" 순서를 지원하는지 여부
EWASUPP	터미널이 "지우기 쓰기 대체" 순서를 지원하는지 여부(예: 대체 화면 크기 기능)
FACILITY	터미널에 대한 4자의 ID
FCI	태스크에 연관된 프린시פל 기능 유형(터미널, 큐 등)
GCHARS GCODES	터미널에 연관된 그래픽 문자 세트 글로벌 ID 및 코드 페이지 글로벌 ID
HILIGHT	터미널에 확장 강조표시 기능이 있는지 여부
KATAKANA	터미널이 가타카나를 지원하는지 여부
LANGINUSE	3자의 니모닉
MSRCONTROL	터미널이 자기 슬롯 리더 제어를 지원하는지 여부
NATLANGINUSE	현재 태스크에 대해 사용 중인 자국어
NETNAME	SNA 네트워크에서 터미널에 대한 8자의 ID

표 37. 터미널에 대한 ASSIGN 명령 옵션 (계속)

ASSIGN 옵션	리턴된 정보
NUMTAB	올바른 통장 영역에 인쇄 요소를 배치하기 위해 필요한 탭 수(2980에만 해당)
OPID OPCLASS	터미널에 사인온한 사용자의 운영자 ID 코드 및 운영자 클래스
OUTLINE	터미널에 필드 개요 기능이 포함되었는지 여부
PARTNS	터미널이 화면 파티션을 지원하는지 여부
PS	터미널이 프로그래밍된 기호 기능을 포함하는지 여부
SCRNHT SCRNWD	현재 태스크에 대한 터미널 화면의 높이 및 너비
SIGDATA	터미널에서 수신한 SIGNAL 데이터
SOSI	터미널에 혼합 EBCDIC/더블바이트 문자 세트 기능 포함 여부
STATIONID TELLERID	터미널의 스테이션 및 텔러 ID(2980에만 해당)
TERMCODE	터미널의 유형 및 모델 번호
TERMPRIORITY	터미널 우선순위 값
TEXTKYBD	터미널에 TEXTKYBD 기능이 포함되었는지 여부
TEXTPRINT	터미널에 TEXTPRINT 기능이 포함되었는지 여부
UNATTEND	터미널이 무인인지 여부
USERID USERNAME USERPRIORITY	터미널에 사인온한 사용자에 대한 8자의 ID, 20자의 이름 및 우선순위
VALIDATION	터미널에 유효성 검사 기능이 포함되었는지 여부

INQUIRE TERMINAL 명령을 사용하여 자체 터미널 또는 임의의 기타 터미널을 찾을 수 있습니다. INQUIRE TERMINAL은 터미널 정의에서 정보를 리턴하며 ASSIGN은 현재 태스크에서 해당 터미널의 사용에 대해 설명합니다. 그렇지만 많은 옵션, 특히 하드웨어 특성에 대해 리턴되는 정보는 동일합니다. INQUIRE TERMINAL은 CICS 시스템 프로그래밍 참조서의 INQUIRE TERMINAL에서 설명됩니다.

터미널 제어 운영에 대한 EIB 피드백

CICS는 EIB에서 BMS에 의해 생성된 것을 비롯하여, 터미널 제어 명령의 처리 결과를 보고합니다.

터미널 운영의 복잡도 때문에, 많은 EIB 필드는 터미널 명령에 고유합니다. 프린시플 기능에 적용되는 것은 741 페이지의 표 38에 나열되어 있습니다. (다른 필드는 LU 유형 6.1, APPC 및 MRO 운영에만 관련됩니다. 이 운영에 대한 프로그래밍 정보는 참조 -> 애플리케이션 개발에서 CICS 명령 요약의 내용을 참조하십시오.)

EIB 필드는 CICS가 제어를 사용자 태스크로 리턴할 때 게시되며, 항상 필드가 적용되는 최신 명령을 설명합니다. 이는 대체 기능을 통한 프로그램간 통신을 수행하고 프린시펄 기능을 사용 중인 경우, 하나의 결과가 다른 것의 결과를 오버레이하기 위해 EIB를 확인해야 합니다.

또한 태스크가 터미널로부터의 요청되지 않은 입력으로, 또는 동일한 터미널에서 이전 태스크에 있는 RETURN IMMEDIATE에 의해 초기화될 때, 입력을 설명하는 EIB 필드가 태스크 시작 시 설정되지 않음을 의미합니다. 입력에 대한 액세스를 얻고 EIB 필드를 게시하려면 RECEIVE를 실행해야 합니다.

참고: 데이터 자체가 아니라 EIB 값에만 관심이 있는 경우, INTO 및 SET 옵션 모두를 RECEIVE에서 생략하십시오.

다음은 프린시펄 기능에 적용되는 필드입니다.

표 38. 터미널 제어 명령에 적용되는 EIB 필드

필드	내용
EIB AID	마지막 입력 조작의 어텐션 ID(AID)(3270만 적용, 769 페이지의 『3270 터미널에서의 입력』 참조)
EIB ATT	입력에 접속 헤더 데이터(접속 FMH)가 포함되는지 여부
EIB COMPL	방금 실행한 RECEIVE 명령이 모든 입력 데이터를 사용했는지, 또는 추가 RECEIVE가 필요한지 여부(743 페이지의 『출력 데이터 체이닝』 참조).
EIB POSN	마지막 입력 조작 시 커서 위치(3270만 적용)
EIB EOC	마지막 RECEIVE의 입력에 체인 끝 표시자가 나타났는지 여부
EIB FMH	방금 수신 또는 검색한 사용자 데이터에 FMH가 있는지 여부
EIB FREE	방금 사용한 기능이 사용 가능하도록 해제되었는지 여부
EIB RCODE, EIB RESP, EIB RESP2	이전에 실행된 명령의 CICS 응답 코드 값 참고: 전송이 지연될 수 있는 출력 명령에 대해, 이 값은 최종 전송이 아니라 명령의 초기 CICS 처리만 반영합니다(728 페이지의 『터미널 대기』 참조).
EIB SIG	터미널이 SIGNAL을 전송했는지 여부
EIB TRMID	(CICS) 터미널의 ID

제 30 장 SNA 사용

논리 장치와의 통신은 논리 장치의 유형에 따라 달라지는 규칙(프로토콜)으로 관리됩니다. 이 절에서는 애플리케이션이 이 프로토콜을 준수하도록 하고 최상의 사용이 되도록 하기 위해 CICS에서 제공되는 옵션에 대해 설명합니다.

입력 데이터 체이닝

일부 SNA 디바이스는 전송을 위해 긴 입력 메시지를 세그먼트화합니다. 각각의 개별 세그먼트를 요청 단위(RU)라고 하며, 전체 논리 메시지를 체인이라고 합니다. CICS는 체인을 어셈블하는 사람을 관리하는 터미널 정의 BUILDCHAIN에서 옵션을 제공합니다.

터미널의 BUILDCHAIN 값이 YES이면, CICS는 체인을 어셈블하고 단일 RECEIVE 명령에 대한 응답으로 전체 메시지를 프로그램에 제시합니다. 이 선택사항은 전체 체인이 애플리케이션에 제시되기 전에 완전하고 사용 가능한지 확인합니다.

BUILDCHAIN=NO이면, 애플리케이션은 체인을 어셈블합니다. CICS는 RECEIVE마다 하나의 RU를 제공합니다. 애플리케이션은 체인에서 마지막 RU를 수신한 시기를 알려줍니다. CICS가 이 때 EOC(체인 끝) 조건을 발생하기 때문입니다. 체인에 단 하나의 RU가 있거나 CICS가 체인을 어셈블할 때, 또는 입력이 3270 표시장치와 같은 인바운드 체이닝을 지원하지 않는 터미널에서 제공될 때도 CICS는 이 조건을 발생합니다. EOC 조건은 오류로 간주되지 않습니다. 발생 시 CICS 기본 조치는 조건을 무시하는 것입니다.

EOC는 EODS(데이터 세트의 끝) 또는 INBFMH(인바운드 FMH) 조건 또는 둘 다에 대해 동시에 발생할 수 있습니다. 이 조건과 EOC 둘 다가 활성 HANDLE CONDITION 명령의 대상인 경우 두 조건 중 하나가 EOC보다 우선합니다.

출력 데이터 체이닝

SNA용 z/OS Communications Server는 아웃바운드 및 인바운드 터미널 데이터의 체이닝을 지원합니다. 출력 메시지의 길이가 아웃바운드 RU 크기를 초과하는데 터미널이 아웃바운드 체인을 지원하는 경우, CICS는 RU 크기 세그먼트로 메시지를 분리하여 개별적으로 전송합니다.

애플리케이션은 체이닝이 여러 SEND 명령에 걸쳐 조금씩 단일 출력 메시지를 CICS로 전달하여 허용된다는 사실을 이용할 수 있습니다. 이와 같이 수행하려면, 메시지를 완료하는 것을 제외하고 각 SEND에서 CNOTCOMPL(“체인이 완료되지 않음”) 옵션을 지정합니다. (메시지 세그먼트는 특정 길이가 될 필요는 없습니다. CICS는 필요한 만

큼 많은 RU를 어셈블하고 전송합니다.) 트랜잭션이 실행 중인 PROFILE 정의는 이를 수행하도록 CHAINCONTROL=YES를 지정해야 합니다.

참고: 전체 논리 메시지(즉, 전체 체인)에 적용하는 옵션은 체인에 대한 첫 번째 SEND 명령에만 표시되어야 합니다. 여기에는 FMH 및 LAST와, 3601의 경우 LDC가 포함 됩니다.

논리 레코드 처리

일부 디바이스는 입력 메시지를 블로킹하고 단일 전송으로 여러 입력을 전송합니다. CICS에서는 CICS 또는 애플리케이션이 입력 블로킹을 해제해야 하는지 여부를 지정할 수 있습니다.

선택사항은 현재 프로파일이 실행 중인 PROFILE의 LOGREC 옵션으로 표시됩니다.

LOGREC(NO)의 경우, CICS는 RECEIVE의 응답으로 전체 입력 메시지를 제공합니다. (입력이 체인 연결되어 있지 않거나 BUILDCHAIN=YES인 것으로 가정합니다.) 사용자에게 입력 블로킹 해제의 책임이 있습니다. BUILDCHAIN=NO의 경우, RECEIVE는 체인에서 한 번에 하나의 RU를 검색합니다. 일반적으로, 논리 레코드 범위는 RU가 아니므로, 단일 RU에는 하나 이상의 완전한 논리 레코드가 포함됩니다. LU 유형 4 디바이스는 예외입니다. 이 디바이스에서 논리 레코드는 하나의 RU에서 시작하여 다른 RU에서 계속할 수 있습니다. 이러한 이유로, 이 디바이스에 대해 사용자 자신의 블로킹 해제를 수행하는 경우 BUILDCHAIN=YES가 권장됩니다.

PROFILE이 LOGREC(YES)를 지정하는 경우, CICS는 각 RECEIVE 명령에 대한 응답으로 하나의 논리 레코드를 제공합니다(CICS가 입력 체인을 어셈블링하는지 여부에 관계없이).

RU에 두 개 이상의 논리 레코드가 포함되는 경우, 레코드는 줄 바꾸기(NL) 문자 X'15', 레코드 간 분리문자(IRS 문자) X'1E' 또는 투명(TRN) 문자 X'35'로 구분됩니다. NL 문자가 사용되는 경우, 데이터가 프로그램에 전달될 때 제거되지 않고 논리 레코드의 끝에 표시됩니다. 그러나 IRS 문자가 사용되는 경우에는 제거됩니다. 구분 기호가 투명 문자인 경우, 논리 레코드는 투명 모드에서 일반 데이터로 간주되는 NL 및 IRS를 비롯하여 어떤 문자도 포함할 수 있습니다. 그러나 종료 TRN은 제거됩니다. CICS는 TRN으로 구분되는 논리 레코드를 256자로 제한합니다.

응답 프로토콜

SNA를 위한 z/OS Communications Server에서 CICS는 아웃바운드 데이터를 위한 확정 응답 또는 예외 응답 프로토콜의 사용을 허용합니다.

예외 응답 아래에서 터미널은 오류가 발생한 경우에만 SEND를 수신확인합니다. 태스크가 예외 응답을 사용 중인 경우에는 CICS는 태스크를 종료하기 전에 태스크에서 마

지막 SEND(유일한 SEND일 수 있음)가 완료되기를 기다리지 않습니다. 결과적으로 오류가 발생하면 이를 태스크에 보고할 수 없습니다. 이 경우 오류는 이 용도로 작성된 CICS 제공 태스크에 보고됩니다.

확정 응답은 터미널이 모든 SEND를 수신확인할 것을 요구하고, CICS는 마지막 SEND 시에 응답을 받을 때까지 태스크를 종료하지 않습니다. 확정 응답 프로토콜을 사용하면 약간의 성능 불이익이 있지만 일부 애플리케이션에서는 필요할 수 있습니다.

태스크가 실행 중인 PROFILE의 MSGINTEG 옵션은 어떤 응답 모드가 사용되는지를 판별합니다. 그러나 MSGINTEG(NO)(예외 응답)를 선택하는 경우에는 여전히 DEFRESP 옵션을 사용하여 특정 SEND에 확정 응답을 요청할 수 있습니다. 이 방법으로 확정 응답을 선택적으로 사용하여 필요할 때에만 성능 불이익을 부과할 수 있습니다. 계속하기 전에 데이터의 전달을 검증해야 하는 트랜잭션의 경우 DEFRESP 옵션이 마지막 SEND에서 사용되어야 합니다.

기능 관리 헤더 사용

SNA 구조는 기능 관리 헤더(FMH)라고 하는, 일부 메시지와 함께 제공되는 헤더 필드의 특정 유형을 정의합니다.

헤더는 메시지에 대한 정보와 처리 방법을 전달합니다. 일부 논리 장치의 경우 FMH의 사용은 필수적이고, 다른 논리 장치의 경우에는 선택적이며, 일부 경우에는 FMH를 사용할 수 없습니다. 특히, FMH는 LU 유형 2와 LU 유형 3 터미널에 적용되지 않는데, 이 터미널은 가장 일반적인 3270 디바이스입니다.

인바운드 FMH

FMH가 입력 메시지에 제시될 때, CICS는 트랜잭션이 FMH를 제거할 것인지 또는 RECEIVE를 실행한 애플리케이션에 전달할 것인지 여부를 결정하기 위해 트랜잭션이 실행되는 PROFILE 정의를 조사합니다.

PROFILE은 어떤 FMH도 전달되지 않거나, 데이터 세트 끝을 표시하는 FMH만 전달하거나, 모든 FMH가 전달되도록 지정할 수 있습니다. 또한 FMH가 일괄처리 데이터 교환 프로그램으로 전달되도록 하는 옵션도 있습니다.

FMH가 있는 경우, 입력 메시지의 초기 바이트를 점유합니다. 해당 길이는 디바이스 유형에 따라 다릅니다. CICS는 존재하고 있음을 사용자에게 알리기 위해 EIB에서 EIBFMH 필드를 설정하고(X'FF'), HANDLE CONDITION 명령을 통하거나 RESP 값을 테스트하여 발견할 수 있는 INBFMH 상태를 발생할 수도 있습니다.

아웃바운드 FMH

출력에서, FMH는 애플리케이션이나 CICS에 의해 빌드될 수 있습니다.

프로그램이 FMH를 제공하는 경우, 출력 데이터 앞에 위치시키고 SEND 명령에 FMH 옵션을 지정합니다. CICS가 FMH를 빌드하기 위한 것이라면, FMH 옵션을 완료하고 생략하기 위해 CICS에 대한 메시지의 처음 세 바이트를 예약합니다. CICS는 필요로 하는 디바이스에 대해서만 FMH를 빌드하므로, 선택사항인 디바이스에는 제공하지 말아야 합니다.

브래킷 프로토콜로 인터럽트 방지

브래킷은 두 LU 사이의 대화가 세 번째 LU의 요청으로 인터럽트되지 않도록 하기 위한 SNA 프로토콜입니다. CICS는 CICS 태스크와, 태스크 지속 기간 동안 해당 프린시פל 기능 사이의 대화 인터럽트를 방지하기 위해 브래킷 프로토콜을 사용합니다.

태스크에 대체 기능이 있는 경우 동일한 이유로 브래킷 프로토콜이 사용됩니다. 논리 장치는 태스크를 초기화하기 위해 요청되지 않은 입력을 전송하는 경우에 브래킷을 시작하고, CICS는 태스크를 자동으로 초기화하는 경우 브래킷을 시작합니다. IMMEDIATE 옵션이 마지막 RETURN 명령에 나타나지 않으면, CICS는 태스크 종료 시 브래킷을 종료합니다. RETURN IMMEDIATE는 입력을 허용하지 않고 프린시פל 기능에서 다른 태스크를 초기화할 수 있도록 합니다. CICS는 브래킷이 사용 중일 때 종료 태스크와 해당되는 후속 태스크 사이의 브래킷을 종료하지 않고 이를 수행합니다.

CICS에서는 SNA용 z/OS 통상 서버 환경에서 많은 디바이스에 대해 브래킷을 사용해야 합니다. 다른 경우에 브래킷 사용은 터미널 정의에 있는 BRACKET 옵션의 값으로 판별됩니다. 브래킷 프로토콜은 SNA의 기능이므로, SNA 이외 디바이스에 대해 BRACKET(YES)을 지정하는 경우, CICS는 엄격한 브래킷 프로토콜을 따르지도, 시행하지도 않습니다.

일반적으로, 브래킷 프로토콜은 애플리케이션에 투명하지만 SEND 명령에서 LAST 옵션을 사용하여 브래킷 프로토콜에 관련된 플로우를 계속 최적화할 수 있습니다. 특정 SEND가 태스크에서 터미널에 대한 마지막 명령인 것을 알면, LAST 옵션을 추가하여 성능을 개선할 수 있습니다. LAST는 Communications Server가 데이터와 함께 “브래킷 끝” 지시자를 전송할 수 있도록 하며 태스크 종료 시 전송하기 위해 별도의 전송을 저장합니다. 프로그램 빌드 체인에서 마지막 출력을 전송 중인 경우(CNOTCOMPL 사용), LAST는 적용되도록 체인에 대한 첫 번째 SEND에 지정해야 합니다.

태스크에 수행할 중요한 작업이 있거나 해당되는 마지막 SEND 후에 상당한 지연이 발생할 수 있는 경우, FREE 명령을 발행할 수 있습니다. FREE는 다른 태스크에서 사용하도록 터미널을 해제합니다.

제 31 장 순차 터미널 지원 사용

CICS가 지원하는 많은 유형의 터미널 중 하나는 실제로 터미널이 아니지만, 터미널을 시뮬레이트하는 한 쌍의 순차 디바이스 또는 파일입니다.

쌍의 하나는 터미널의 입력 측을 나타내고, 테이프나 DASD에서 카드 리더, 스푼 파일 또는 SAM 파일이 될 수 있습니다. 다른 것은 출력을 나타내며, 프린터, 펀치, 스푼 또는 SAM 파일이 될 수 있습니다. 많은 디바이스-유형 조합이 허용되고 쌍 중 어느 하나가 빠질 수 있습니다. 입력 전용 또는 출력 전용 순차 터미널이 있을 수 있습니다.

터미널 제어 명령(특히 RECEIVE, SEND 및 CONVERSE)과 함께 순차 터미널을 구성하는 디바이스 또는 파일에 대해 읽고 씁니다. (BMS 역시 순차 터미널을 지원합니다. 932 페이지의 『비3270 터미널용 특수 옵션』의 내용을 참조하십시오.)

순차 터미널 지원의 원래 목적은 애플리케이션 개발자가 실제 터미널에 대한 액세스를 갖기 전에 온라인 코드를 테스트할 수 있도록 허용하는 것이었습니다. 이 요구사항은 요즘은 드물게 발생하지만, 순차 터미널은 다음에 대해 여전히 유용합니다.

인쇄 891 페이지의 『비CICS 프린터를 위한 프로그래밍』의 내용을 참조하십시오. 순차 터미널은 간혹 터미널 제어 명령이나 BMS가 필요한 저속 CICS 프린터로 보내지고, 간혹 고속 시스템 프린터(스푼 또는 트랜지언트 데이터 명령)로 보내지는 출력에 유용합니다. 고속 프린터를 순차 터미널로 정의하는 경우, 터미널 제어 또는 BMS 명령을 사용할 수 있고 두 유형의 프린터 모두에 대해 동일한 코드를 사용할 수 있습니다. (디바이스 데이터 스트림에서 차이가 있는 경우, 완전한 투명성을 위해 BMS를 사용해야 합니다.)

회귀 테스트

순차 터미널에서 실행되는 테스트는 입력 및 출력 모두의 영구 레코드를 유지합니다. 이로서 조직적이고 확인 가능한 초기 테스트가 가능합니다. 또한 수정 후에 테스트를 반복하고, 지정된 입력 세트가 변경 후에도 이전과 같이 동일한 출력 세트를 생성하도록 합니다.

초기화 일부 설치에서는 프로그램 목록 테이블 프로그램보다 먼저 하나 이상의 초기화 트랜잭션을 실행하기 위해 순차 터미널을 사용합니다. 순차 터미널에서 초기화된 트랜잭션은 터미널이 운영되고 바로 실행을 시작하고, 입력이 소모될 때까지 CICS가 처리할 수 있는 속도만큼 빠르게 계속합니다. 따라서 순차 터미널의 입력은 시동 후 즉시(순차 터미널이 초기에 가동 중인 경우), 나중에(운영 중인 때) 또는 제어되는 종료의 일부로 처리될 수 있습니다.

순차 터미널에 대한 코딩 고려사항

순차 터미널에서 제출된 입력 데이터는 전기 통신 디바이스에서 제공되는 양식으로 되어 있어야 합니다.

예를 들어, 첫 번째 레코드는 보통 트랜잭션 코드로 시작하여, CICS에 실행할 트랜잭션을 알려줍니다. 트랜잭션 코드는 실제 터미널에서와 같이 입력의 첫 번째 위치에서 시작해야 합니다. 이로서 복잡한 형식의 입력이 요구되는 애플리케이션을 테스트하는 것으로 기능이 제한됩니다. 예를 들어, 모든 복합적 제어 순서 때문에, 형식화된 3270 입력 스트림을 순차 파일로 표현하기 위한 어떤 프로비전도 없습니다. 그러나 입력으로 형식화되지 않은 3270 데이터 스트림(또는 다른 유사한 스트림)을 사용할 수 있고, 계속해서 BMS를 사용하여 출력을 형식화할 수 있습니다.

입력 파일을 빌드할 때, 데이터 끝 지시(EODI) 문자를 각 입력 레코드 다음에 위치시킬 수 있습니다. EODI 문자는 시스템 초기화 테이블에서 정의됩니다. 기본값은 백슬래시('\', X'E0')이지만, 사용자의 설치에서 다른 값을 정의하였을 수 있습니다.

입력 스트림을 처리할 때 CICS는 EODI 문자만 주시합니다. CICS는 입력 파일 또는 디바이스의 레코드 구조를 분석하지 않으며, 이는 각 입력 범위가 입력 파일의 레코드가 될 수 있음을 의미합니다. 그러나 각 입력이 올바르게 처리되도록 하려면 새 물리적 레코드에서 각각의 입력을 시작해야 합니다.

입력 레코드의 길이(EODI 사이의 문자 수)는 입력 버퍼의 크기(순차 터미널 정의의 LINE 구성요소에 있는 INAREAL 값)를 초과하면 안됩니다. 초과하면, 긴 레코드를 RECEIVE하려고 하는 트랜잭션이 이상 종료하고, CICS는 입력 처리를 재개하기 전에 다음 EODI 뒤에 입력 파일을 위치시킵니다.

입력의 파일 끝(EOF) 표시자 역시 EODI 표시자로 작동합니다. 파일 끝(EOF)이 발견된 이후에 실행된 RECEIVE 명령은 이상 종료를 발생시킵니다.

인쇄 형식화

순차 터미널의 정의에서 출력 반이 라인 프린터임을 표시하는 경우, 여러 행의 출력을 단일 SEND로 쓸 수 있습니다.

이 유형의 디바이스의 경우, CICS는 각각의 줄 바꾸기 문자(X'15') 다음이나 행 길이로 정의된 문자 수 이후에(둘 중 먼저 발생하는 것에 따라) 출력 메시지를 행으로 분리합니다. 행 길이는 터미널 정의에서 LPLEN 값으로 정의됩니다. 각 SEND는 새 행을 시작합니다.

GOODNIGHT 규칙

CICS는 CICS(또는 트랜잭션 자체)가 모든 입력을 고갈하거나 터미널이 서비스를 벗어날 때까지 순차 터미널에서 트랜잭션 초기화를 계속합니다. CICS가 입력 파일의 끝을

지나 읽기를 시도하지 않도록 하려면(이러한 경우 트랜잭션 이상 종료 발생함) 마지막 입력이 CESF GOODNIGHT 트랜잭션이면 됩니다. 그러면 터미널이 꺼졌다는 신호를 보내고 서비스 외부에 둡니다.

또는 실행된 마지막 트랜잭션이 최종 출력 후에 서비스 외부에 터미널을 둘 수 있습니다. CICS가 서비스 외부에 놓지 않고 해당되는 원래 입력을 처리하면, 보통 서비스의 순차 터미널에서 추가 입력을 입력할 수 없습니다.

제 32 장 일괄처리 데이터 상호 교환 사용

CICS 일괄처리 데이터 상호 교환 프로그램은 애플리케이션과 아웃바운드 제어기의 일괄처리 데이터 상호 교환 논리 장치의 파트인 데이터 세트(또는 대상) 사이의 통신 또는 일괄처리 논리 장치나 LU 유형 4 논리 장치의 선택 매체를 사용하는 통신을 위해 제공됩니다. 이 매체는 프린터나 콘솔 같은 필수 디바이스를 나타냅니다.

외부 제어기는 프로그램 가능한 서브시스템(예: DPCX를 실행하는 IBM 3770 Data Communication System, IBM 3790 Data Communication System 또는 IBM 8100 System)에 대한 일반적인 참조로 SNA 프로토콜을 사용합니다. SNA 프로토콜 및 사용 가능한 데이터 세트에 대한 세부사항은 *IBM 3767/3770/6670 Guide*에서 제공됩니다. 그림 122에서는 일괄처리 데이터 상호 교환에 대한 개요를 제공합니다.

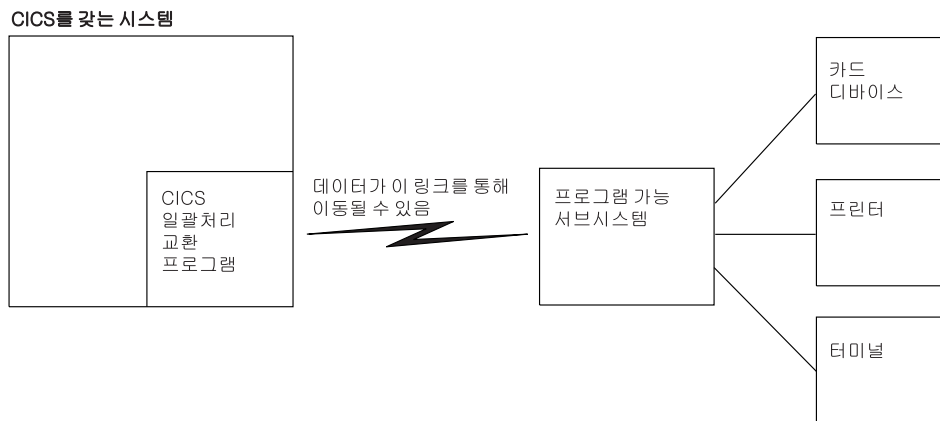


그림 122. CICS 일괄처리 데이터 상호 교환

다음 일괄처리 데이터 상호 교환 명령이 제공됩니다.

ISSUE QUERY

CICS 애플리케이션으로의 데이터 세트 전송을 시작합니다.

ISSUE RECEIVE

데이터 세트에서 레코드를 읽거나 입력 매체에서 데이터를 읽습니다.

ISSUE SEND

데이터를 이름 지정된 데이터 세트 또는 선택한 매체로 전송합니다.

ISSUE ADD

레코드를 데이터 세트에 추가합니다.

ISSUE REPLACE

데이터 세트에서 레코드를 갱신(바꾸기)합니다.

ISSUE ERASE

데이터 세트에서 레코드를 삭제합니다.

ISSUE END

데이터 세트 처리를 종료합니다.

ISSUE ABORT

데이터 세트 처리를 비정상 종료합니다.

ISSUE NOTE

데이터 세트에서 다음 레코드 번호를 요청합니다.

ISSUE WAIT

조작 완료를 대기합니다.

제어기가 LU 유형 4 논리 장치인 경우 ISSUE ABORT, ISSUE END, ISSUE RECEIVE, ISSUE SEND, ISSUE WAIT 명령만 사용할 수 있습니다.

데이터 세트가 DPCX/DXAM 데이터 세트인 경우, ISSUE ADD, ISSUE ERASE, ISSUE REPLACE 명령만 사용할 수 있습니다.

일괄처리 데이터 상호 교환 명령 실행 중에 발생하는 모든 예외 조건 처리 방법에 대한 정보는 423 페이지의 『예외 조건 처리』의 내용을 참조하십시오.

대상 선택 및 ID

모든 일괄처리 데이터 상호 교환 명령(ISSUE RECEIVE 제외)은 대상을 지정하는 옵션을 포함합니다. 이는 일괄처리 데이터 상호 교환 논리 장치의 이름 지정된 데이터 세트 또는 일괄처리 논리 장치 또는 LU 유형 4 논리 장치의 선택 매체일 수 있습니다.

이름 지정된 데이터 세트로 대상을 선택하는 경우, DESTID 및 DESTIDLENG 옵션은 데이터 세트 이름 및 해당 길이(최대 8자)를 제공하기 위해 항상 지정해야 합니다. 디스켓이 포함된 대상인 경우, VOLUME 및 VOLUMELENG 옵션을 지정하여 볼륨 이름 및 해당 길이(최대 6자)를 제공할 수 있습니다. 볼륨 이름은 조작에서 사용되는 데이터 세트가 포함된 디스켓을 식별합니다. 멀티디스켓 대상에 대해 VOLUME 옵션을 지정하지 않으면, 필요한 데이터 세트를 찾을 때까지 모든 디스켓이 검색됩니다.

대상으로 이름 지정된 데이터 세트를 사용하는 대신, 다양한 매체를 CONSOLE, PRINT, CARD 또는 WPMEDIA1-4 옵션 대신 지정할 수 있습니다. 이 매체는 ISSUE ABORT, ISSUE END, ISSUE SEND 또는 ISSUE WAIT 명령에만 지정할 수 있습니다.

확정 응답(DEFRESP 옵션)

CICS는 터미널 제어 명령을 사용하여 일괄처리 데이터 상호 교환에 지정된 기능을 수행합니다. 터미널 제어 출력 요청이 작성되도록 하는 해당 명령에 대해 DEFRESP 옵션

션을 지정할 수 있습니다. 이 옵션은 SEND 터미널 제어 명령의 DEFRESP 옵션과 같은 효과가 있습니다. 즉, 외부 제어기에서의 확정 응답 요청은 CICS 태스크의 메시지 무결성 스펙과 무관합니다. DEFRESP 옵션은 ISSUE ADD, ISSUE ERASE, ISSUE REPLACE, ISSUE SEND 명령에 대해서 지정할 수 있습니다.

기능 완료 대기(NOWAIT 옵션)

터미널 제어 출력 요청을 작성하는 이 일괄처리 데이터 상호 교환 명령에 대해 NOWAIT 옵션을 지정할 수 있습니다. 이 옵션은 CICS 태스크 처리가 계속되도록 허용하는 효과가 있습니다. NOWAIT 옵션을 지정하는 경우가 아니라면 태스크 활동은 일괄처리 데이터 상호 교환 명령이 완료될 때까지 일시중단됩니다. NOWAIT 옵션은 ISSUE ADD, ISSUE ERASE, ISSUE REPLACE, ISSUE SEND 명령에서만 지정할 수 있습니다.

NOWAIT 옵션을 사용하는 일괄처리 데이터 상호 교환 명령이 실행된 후, 태스크 활동은 명령이 완료되기를 기다리는 프로그램의 적절한 시점에서 ISSUE WAIT 명령으로 일시중단될 수 있습니다.

제 33 장 터미널 제어: 성능 향상을 위한 디자인

이 목록은 성능 향상을 위한 디자인 시 터미널 제어에 대해 고려해야 하는 사항을 보여줍니다.

- 터미널로 전송된 데이터 스트림의 길이 최소화.

좋은 화면 디자인과 효과적인 3270 하드웨어 기능 사용은 원격 처리 링크에서 전송된 바이트 수에 상당한 영향을 줄 수 있습니다. 대부분의 경우 이는 트랜잭션이 사용하는 경로의 가장 느린 부분이기 때문에 바이트 수를 가능한 작게 유지하는 것이 중요합니다. 따라서 데이터 스트림의 효율성은 응답 시간과 회선 사용 둘 다에 영향을 줍니다.

- 화면당 하나의 물리적 **SEND** 명령만 사용.

일반적으로 단일 BMS 호출을 사용하여 화면을 작성하는 것이 일련의 **SEND MAP** **ACCUM** 명령을 사용하여 화면을 빌드하는 것보다 더 효율적입니다. 터미널로 하나의 물리적 출력으로 된 화면을 전송하는 것이 중요합니다. 여러 개의 명령을 사용하는 추가 프로세서 오버헤드와 추가 회선 및 액세스 방법 오버헤드로 인해 화면을 나누어서 빌드하고 별도의 명령을 사용하여 각 파트를 전송하는 것은 **매우** 비효율적입니다.

- **CONVERSE** 명령 사용.

SEND 및 **RECEIVE** 명령 대신 **CONVERSE** 명령(또는 프로그램이 대화식인 경우 **SEND**, **WAIT**, **RECEIVE** 명령 순서)을 사용하십시오. 기능적으로는 같지만 **CONVERSE** 명령은 CICS 서비스 인터페이스를 한 번만 교차합니다(프로세서 시간이 절약됨).

- 메시지 무결성 옵션 사용 제한.

트랜잭션의 최종 **SEND** 명령에 **WAIT** 옵션을 지정하는 것과 마찬가지로 **CEDA**의 **MSGINTEG** 옵션을 사용하면 마지막 메시지가 정상적으로 전달될 때까지 CICS가 트랜잭션을 계속 실행해야 합니다. **PROFILE** 정의의 **PROTECT** 옵션은 메시지 무결성을 나타내며 시스템이 입력 및 출력 메시지를 모두 로그하게 합니다(I/O 및 프로세서 오버헤드에 추가됨).

- **SEND** 명령에서 **DEFRESP** 옵션 사용 방지.

트랜잭션이 출력 메시지 전달을 검증하지 않는 한 **SEND** 명령에서 **DEFRESP** 옵션을 사용하지 마십시오. **SGINTEG**와 동일한 방법으로 트랜잭션 종료를 지연시킵니다.

- 불필요한 트랜잭션 사용 방지.

사용자가 잘못된 트랜잭션에 진입하거나 CLEAR 키를 불필요하게 사용할 수 있게 하여 터미널 입력, 태스크 제어 처리, 터미널 출력, 오버헤드에 추가하는 상황을 피하십시오. 좋은 화면 설계와 표준화된 PF 및 PA 키 지정은 이를 최소화해야 합니다.

- 맵 없이 형식화되지 않은 데이터 전송.

터미널에 대한 출력이 전적으로 또는 대부분 형식화되지 않은 경우, BMS 명령이 아닌 터미널 제어 명령을 사용하여 전송할 수 있습니다(즉, MAP 또는 TEXT 옵션 없이 BMS SEND 명령 사용).

제 34 장 터미널의 3270 제품군

3270은 공통 특성을 공유하고 터미널과 호스트 프로세서 간에 통신하기 위해 동일한 인코딩된 데이터 형식을 사용하는 지원되는 제어 장치가 있는 표시장치 및 프린터 터미널의 제품군입니다. 이 데이터 형식은 3270 데이터 스트림으로 알려져 있습니다.

3270은 많은 기능과 특성이 있는 복합 디바이스입니다. 여기에서는 기본 조작에 대해서만 다루고 CICS가 3270을 지원하는 방식에 대해 강조합니다. 3270 기능에 대한 종합적인 논의, 프로그래밍 및 데이터 스트림 형식은 IBM 3270 Data Stream Programmers Reference의 내용을 참조하십시오. IBM 3270 Data Stream Device Guide에는 또한 중요 정보가 들어 있습니다. 이는 주로 터미널 제어를 사용하는 프로그래머를 위한 것이지만 BMS 프로그래머에게 도움이 되는 정보 또한 포함되어 있습니다. 몇몇 특수 기능에 대한 BMS 지원은 1010 페이지의 『특수 하드웨어에 대한 지원』에서 설명됩니다.

이 설명은 표시장치 터미널에 집중되지만 대부분의 자료는 3270 프린터에도 동일하게 적용됩니다. 3270 프린터는 3270 표시장치와 동일한 데이터 스트림을 승인하고 화면 이미지를 하드카피 형식으로 전달합니다. 대부분의 차이점은 프린터에는 없는 입력과 관련되어 있습니다.

그러나 프린터에서 추가로 형식화 기능을 사용할 수 있고 인쇄된 출력을 원하는 프린터로 보낼 때에는 특별한 주의사항이 있습니다. 자세한 정보는 877 페이지의 제 44 장 『인쇄 및 스폴 파일』의 내용을 참조하십시오.

3270 버퍼

3270 디바이스와의 통신은 해당되는 문자 버퍼를 통해 발생하며, 이 문자 버퍼는 프로세서에서 메모리와 유사한 하드웨어 저장영역 메커니즘입니다. 3270에 대한 출력이 버퍼로 전송됩니다. 버퍼는 다시 표시장치 터미널의 표시 및 프린터 터미널의 인쇄 메커니즘을 구동합니다.

반대로, 769 페이지의 『3270 터미널에서의 입력』에 설명된 것처럼, 키보드 입력은 버퍼를 통해 호스트에 도달합니다.

화면의 각 위치는 버퍼에서의 위치에 해당되며, 해당 버퍼 위치의 내용에 따라 화면에 표시되는 내용이 결정됩니다. 화면이 필드에서 형식화될 때, 각 필드의 첫 번째 위치는 필드의 특정 표시 특성을 저장하기 위해 사용되며 데이터를 표시하기 위해 사용할 수 없습니다. (공백으로 표시됩니다.) 3270의 원래 모델에서, 이 바이트는 모든 표시 특성을 저장하기에 충분했습니다. 더 많은 표시 특성을 가지고 있는 나중 모델에서, 화면에

서 고정 위치와 연관되지 않은 버퍼 저장영역에 추가 정보가 보존됩니다. 표시 특성에 대한 자세한 정보는 760 페이지의 『표시장치 특성』의 내용을 참조하십시오.

출력 데이터 스트림

3270 표시장치를 작성하려면 읽기 명령(1바이트), 쓰기 제어 문자 또는 WCC(1바이트) 및 표시장치 데이터(바이트의 변수 수)로 구성되는 데이터 스트림을 보냅니다.

WCC 및 표시장치 데이터는 항상 표시되지는 않습니다. 쓰기 명령은 WCC가 뒤에 오는지 및 데이터가 표시 가능 또는 반드시 표시되어야 하는지를 판별합니다.

BMS를 사용하는 경우 CICS가 전체 데이터 스트림을 빌드합니다. WCC는 SEND 명령의 옵션에서 어셈블되고 쓰기 명령은 다른 SEND 옵션 및 실행 중인 트랜잭션의 PROFILE 정보에서 선택됩니다. 표시장치 데이터는 맵 또는 제공한 텍스트 데이터에서 빌드되며 BMS는 이를 3270 형식으로 변환합니다.

터미널 제어 명령을 사용하는 경우(예: SEND) CICS는 여전히 쓰기 명령을 제공하며 동일한 정보에서 빌드됩니다. 그렇지만 WCC를 제공하고 표시장치 데이터를 3270 형식으로 표시해야 합니다.

3270 쓰기 명령

5개의 3270 명령으로 터미널로 데이터 또는 지시사항을 보낼 수 있습니다.

- Write
- Erase/write
- Erase/write alternate
- 모든 비보호 필드 지우기
- 구조화된 필드 쓰기

3270 **write** 명령은 화면(또는 프린터)이 구동되는 3270 버퍼로 데이터 뒤에 오는 데이터를 전송합니다. **Erase/write** 및 **erase/write alternate**도 데이터를 전송하지만 우선 버퍼를 지웁니다. (즉, 모두 널값으로 설정합니다.) 터미널에 대체 화면 크기 기능이 있는 경우, 이는 버퍼 크기도 판별합니다(화면의 행 및 열 수).

이 기능이 있는 터미널에는 기본 크기 및 대체 크기의 두 크기가 있습니다. **erase/write** 명령은 후속 조작에서 기본 크기가 사용되도록 합니다(다음 **erase/write** 또는 **erase/write alternate** 명령 때까지). **erase/write alternate** 명령은 대체 크기를 선택합니다.

CICS는 **write** 명령을 사용하여 데이터를 전송하며 ERASE 옵션을 **SEND** 명령에 포함하는 경우에는 제외됩니다.

- 지우기를 지정하면 CICS가 실행 중인 트랜잭션에 연관된 PROFILE 정의를 사용하여 **erase/write** 또는 **erase/write alternate** 사용 여부를 판별합니다.
- 기본값 지우기를 지정하면 CICS는 **erase/write** 명령을 사용하고 화면을 기본 크기로 설정합니다.
- 지우기 대체를 지정하면 CICS는 **erase/write alternate**를 사용하고 대체 크기로 화면을 설정합니다.)

erase unprotected to address 명령은 비보호 필드에 대한 버퍼 스캔이 수행되도록 합니다. 이 필드는 761 페이지의 『3270 필드 속성』에서 더 자세하게 정의됩니다. 발견되는 이런 필드는 널(null)로 설정됩니다. 이 선택적 지우기는 945 페이지의 『기호 및 물리적 맵핑 병합』의 **SEND CONTROL** 명령에서 설명하는 대로 데이터 입력 조작에서 유용합니다. 이 명령 뒤에는 WCC 또는 데이터가 오지 않으며 명령만 전송합니다.

구조화된 필드 쓰기 명령은 뒤에 오는 데이터가 3270 구조 필드로 해석되도록 합니다. 구조 필드는 3270의 일부 고급 함수 기능에 필요합니다. 여기에서 설명되지는 않지만 STRFIELD 옵션이 포함된 터미널 제어 **SEND** 명령으로 이를 작성합니다. 자세한 정보는 IBM 3270 Data Stream Device Guide의 내용을 참조하십시오.

쓰기 제어 문자

3270 **write**, **erase/write** 또는 **erase/write alternate** 명령 뒤에 오는 바이트는 쓰기 제어 문자 또는 WCC입니다. WCC는 3270에게 다음 수행 여부를 지시합니다.

- 가청 알람 소리 표시
- 키보드 잠금 해제
- 수정된 데이터 태그 끄기
- 인쇄 시작(터미널이 프린터인 경우)
- 구조 필드 다시 설정
- 인바운드 응답 모드 다시 설정

BMS에서 CICS는 **SEND MAP** 명령에서 ALARM, FREEKB, FRSET, PRINT 옵션을 사용하여 WCC를 작성합니다. 터미널 제어 명령을 사용하는 경우, CTLCHAR 옵션으로 WCC를 명시적으로 지정할 수 있습니다. 사용하지 않으면 CICS는 키보드를 잠금 해제하고 수정된 데이터 태그(이 태그에 대해서는 761 페이지의 『3270 필드 속성』에서 설명)를 끄는 WCC를 생성합니다.

3270 표시장치 필드

표시장치 데이터는 표시할 문자와 문자를 표시할 방법 및 위치에 관한 디바이스에 대한 지시사항 조합으로 구성됩니다.

일반적인 환경에서, 773 페이지의 『형식화되지 않은 모드』에 설명된 것처럼, 이 데이터는 필드 정의 없이 화면을 쓸 수 있어도 필드 정의 시리즈로 구성됩니다.

지우는 쓰기 명령 이후에, 화면에서 모든 필드를 정의해야 합니다. 이후에, 일반 쓰기 명령을 사용하고 변경하려는 필드만 전송할 수 있습니다.

필드를 정의하려면 3270에 다음 사항을 알려야 합니다.

- 표시할 방법
- 해당 내용
- 화면에서 이동하는 위치(즉, 버퍼에서 해당되는 시작 위치)

표시장치 특성

화면의 각 필드에는 속성이라고 하는 표시장치 특성 세트가 있습니다. 속성은 3270에 필드 표시 방법을 알려주며, BMS 또는 터미널 제어 명령 사용 여부를 알아야 합니다.

속성은 다음 두 카테고리에 속합니다.

필드 속성

여기에는 다음이 포함됩니다.

- 보호(운영자가 필드를 수정할 수 있는지 여부)
- 수정(운영자가 필드를 수정했는지 여부)
- 표시장치 강도

모든 3270은 필드 속성을 지원합니다. 761 페이지의 『3270 필드 속성』에 사용자 선택사항에 대한 설명이 있습니다.

필드 속성은 필드의 첫 번째 문자 위치에 저장됩니다. 이 바이트는 화면에서 하나의 위치를 차지하며 필드 속성을 저장할 뿐만 아니라 필드 시작을 표시합니다. 필드는 다음 속성 바이트까지(즉, 다음 필드의 시작까지) 계속됩니다. 다음 필드가 동일한 행에서 시작하지 않으면, 다른 필드가 발생 발견될 때까지 현재 필드는 현재 행의 끝에서 다음 행의 끝까지 줄 바꿈기를 합니다. 마지막 행으로 종료되지 않는 필드는 첫 번째로 리턴됩니다.

확장 필드 속성

(일반적으로 확장 속성으로 축약됩니다.) 이 속성은 3270의 모든 모델에서 제시되지 않습니다. 따라서, 그래픽 사용자 인터페이스를 디자인할 때 사용할 수 있는 속성을 인식해야 합니다. 확장 속성에는 강조표시 및 아웃라인의 특수 양식(2바이트 문자 세트에 대한 프로비전 및 여러 기호 세트를 사용할 수 있는 기능)이 포함됩니다. 763 페이지의 표 39에 7개의 확장 속성과 취할 수 있는 값이 나열되어 있습니다.

3270 필드 속성

필드 속성 바이트는 보호, 수정 및 필드의 표시 강도 속성을 보유합니다. 이러한 각 속성의 선택사항은 BMS가 형식을 정의하는 데 사용하는 용어를 사용하여 여기에 설명되어 있습니다. 터미널 제어 명령을 사용하면 선택한 값을 반영하기 위해 속성 바이트에 해당하는 비트를 설정해야 합니다.

(비트 지정을 위한 *IBM 3270 정보 표시 시스템 데이터 스트림 프로그래머의 참조 매뉴얼*을 참조하십시오. 또한 이 영역에서 CICS의 도움말에 대한 942 페이지의 『속성 값 정의: DFHBMSCA』의 내용을 참조하십시오.)

보호

속성 바이트에서 위로 2 비트 위치를 사용하는 보호 속성을 위한 네 개의 선택사항이 있습니다. 이들은 다음과 같습니다.

보호되지 않음

운영자는 비보호 필드에 데이터 문자를 입력할 수 있습니다.

숫자 전용

이 지정의 영향은 터미널의 키보드 유형에 따라 다릅니다. 데이터 입력 키보드에서는 운영자가 이동하지 않고도 숫자를 입력할 수 있도록 숫자 시프트가 이동합니다. "숫자 잠금" 특수 기능이 있는 키보드에서는 운영자가 숫자 0부터 9까지, 마침표(소수점), 대시(마이너스 부호) 또는 DUP 키 중 하나를 제외한 키를 사용하는 경우 키보드가 잠깁니다. 이는 운영자가 필드에 알파벳 데이터를 입력하는 것을 막습니다. 물론 수신 프로그램은 숫자 양식을 예상하고 있음을 확인하기 위해 여전히 입력을 검사해야 합니다. 숫자 잠금 기능이 없으면 숫자 전용 필드에는 어떤 데이터라도 허용됩니다.

보호됨 운영자는 보호 필드에 키를 입력할 수 없습니다. 이를 수행하려고 하면 키보드를 잠깁니다.

자동 스킵

운영자는 자동 스킵 필드에 키를 입력할 수 없지만 커서는 다르게 행동합니다. (커서는 다음 키 스트로크의 위치를 나타냅니다. 이에 대한 자세한 정보는 769 페이지의 『3270 터미널에서의 입력』의 내용을 참조하십시오.) 커서가 새 필드로 이동할 때마다(이전 필드가 가득 찼거나 필드 이동 키가 사용되었기 때문에) 커서는 해당 경로에서 자동 스킵 필드를 건너뛰고 보호되지 않았거나 숫자 전용인 첫 번째 필드로 이동합니다.

수정

필드 속성 바이트의 정보의 두 번째 항목은 수정된 데이터 태그 또는 **MDT**라고 불리는 단일 비트만을 차지합니다. MDT는 필드가 수정되었는지 여부를 나타냅니다. 하드웨어는 운영자가 필드 내용을 변경할 때마다 자동으로 이 비트를 실행합니다. MDT 비

트는 CICS가 정상적으로 사용하는 읽기 명령의 경우 필드가 인바운드 데이터에 포함되는지 여부를 판별하기 때문에 중요합니다. 비트가 켜져 있으면(즉, 필드가 변경되었음), 3270은 필드를 보내고, 그렇지 않으면 필드를 보내지 않습니다.

필드를 화면에 보낼 때 프로그램을 사용하여 MDT를 켤 수도 있습니다. 이 기능을 사용하면 운영자가 이를 변경할 수 없거나 변경하지 않더라도 읽기에 필드가 리턴됩니다. BMS SEND 명령에서 FRSET 옵션을 사용하면 프로그램이 화면에서 모든 필드의 태그를 끌 수 있습니다. 프로그램을 사용하여 개별 태그를 끌 수 없습니다. 터미널 제어 명령을 사용 중인 경우에는 WCC에서 비트를 켜서 개별 태그를 끌 수 있습니다.

강도

속성 바이트에 저장된 세 번째 특성은 필드의 표시 강도입니다. 세 개의 상호 배타적인 선택사항이 있습니다.

정상 강도

필드는 디바이스의 정상 밝기로 표시됩니다.

밝기 필드는 강조표시되어 보일 수 있도록 정상 강도보다 높게 표시됩니다.

표시 안함

필드가 아예 표시되지 않습니다. 필드는 버퍼에 데이터를 포함할 수 있고, 운영자는 여기에 키를 입력할 수 있지만(보호되거나 자동 스킵되지 않은 경우), 데이터가 화면에 표시되지 않습니다.

표시 강도에 2비트가 사용되는데 이는 앞에 나열된 세 개보다 하나 많은 값을 표시할 수 있게 해줍니다. 연관된 특수 하드웨어 기능이 있는 터미널의 경우 필드가 라이트 펜 감지 가능 또는 커서 선택 가능인지 여부를 판별하는 데 이러한 동일한 2비트가 사용됩니다. 2비트만이 있으므로 강도와 선택 가능성의 모든 조합이 가능한 것은 아닙니다. 절충안은 밝기 필드는 항상 감지 가능하고, 비표시장치 필드는 절대 감지 불가능하고, 보통 강도 필드는 둘 중 어느 것도 아니라는 점입니다. 1014 페이지의 『커서 및 펜 감지 가능 필드』에는 이러한 기능에 대한 자세한 정보가 있습니다.

기본 색상

몇몇 터미널은 확장 속성에 포함된 확장 색상 없이 또는 여기에 추가로 기본 색상을 지원합니다. 이러한 터미널 앞에는 운영자가 기본 또는 기본 색상을 선택할 수 있는 모드 스위치가 있습니다. 기본 색상은 필드 속성이 밝기 강도를 지정하지 않는 한 특성을 녹색으로 표시합니다. 그렇지 않은 경우 흰색입니다. 기본 색상 모드에서는 보호 및 강도 비트가 네 가지 색상, 일반적으로 흰색, 빨간색, 파란색 및 녹색 중에서 선택하기 위해 함께 사용됩니다. 보호 비트는 색상을 결정할 뿐만 아니라 보호 기능을 유지합니다. 기본 색상 이외에 확장 색상을 사용하는 경우에는 3270 터미널의 경우 "흰색"을 색상으로 지정할 수 없음을 유의하십시오. 터미널에서 흰색으로 표시되는 "중성"을 지정해야 합니다.

확장 속성

앞서 설명된 필드 속성에 추가로 몇몇 3270 터미널에는 확장 속성도 있습니다. 다음 테이블은 확장 속성 유형을 첫 번째 열에 나열하고 각 유형에 가능한 값을 두 번째 열에 나열합니다.

표 39. 3270 확장 속성

속성 유형	값
확장 색상	파란색, 빨간색, 분홍색, 초록색, 청록색, 노란색, 중성
확장 강조표시	깜박이는, 반전 비디오, 밑줄
필드 아웃라인	위, 아래, 왼쪽 및 오른쪽 줄의 조합
배경 투명도	배경 투명도, 배경 오파크
필드 유효성 검증	필드에 입력해야 합니다. 필드를 채워야 합니다. 필드가 입력을 트리거합니다.
프로그램 기호 세트	기호 세트를 식별하는 숫자 참고: 터미널과 연관된 제어 장치에는 기본 기호 세트가 포함되고 최대 5개까지 추가로 저장할 수 있습니다. 이들 중 하나를 사용하려면 기호 세트를 사용 전에 제어기에 로드해야 합니다. 터미널 제어 SEND 명령을 사용하여 이를 수행할 수 있습니다.
SO/SI 작성	2바이트 문자를 나타내는 시프트 문자가 있을 수 있습니다. 시프트 문자가 없습니다.

IBM 3270 정보 표시장치 시스템 데이터 스트림 프로그래머의 참조 매뉴얼에는 확장 속성에 대한 세부사항이 포함되고 기본값을 판별하는 방법을 설명합니다. ASSIGN 및 INQUIRE 명령을 사용하여 사용자의 특정 터미널에 어떤 확장 속성이 있는지 판별할 수 있습니다. 이러한 명령은 739 페이지의 제 29 장 『터미널 정보 찾기』에 설명되어 있습니다.

일부 3270 모델에서는 전체적으로 필드의 값과는 다른 한 필드 내에 개별 문자에 확장 속성 값을 지정할 수 있습니다. 일반적으로 BMS는 문자 속성에 대해 명시적 프로 비저닝을 만들지 않기 때문에 이를 수행하려면 터미널 제어 명령을 사용해야 합니다. 그러나 984 페이지의 『텍스트 행』에 설명된 대로 BMS 하의 텍스트 출력에서 문자 속성의 제어 순서를 삽입할 수 있습니다. 767 페이지의 『설정 속성 순서』에서는 이러한 순서의 형식을 설명합니다.

데이터 스트림에서의 순서

터미널 제어 명령을 사용하여 3270에 쓸 때에는 속성, 위치 및 필드 내용을 표현하기 위해 아웃바운드 데이터를 형식화해야 합니다.

이 태스크 정보

BMS를 사용 중인 경우에는 이 모든 것이 사용자를 위해 수행되고 사용자는 769 페이지의 『3270 터미널에서의 입력』로 이동할 수 있습니다.

3270 데이터 스트림에서 필드를 정의할 때에는 시작 필드(SF) 또는 시작 필드 확장(SFE) 순서부터 시작하십시오. 순서는 3270에 대한 지시사항입니다. 이는 버퍼를 로드하는 방법을 알려줍니다. 길이는 1바이트이고 일반적으로 순서에 고유한 형식으로 되어 있고 데이터가 뒤따릅니다.

시작 필드 순서

SF 순서는 모든 모델에서 지원되고 필드 속성과 필드의 표시 내용을 지정할 수 있지만 확장 속성은 지정할 수 없습니다.

SF를 사용하여 필드를 정의하려면 그림 123에서처럼 데이터 스트림에 순서를 삽입해야 합니다.

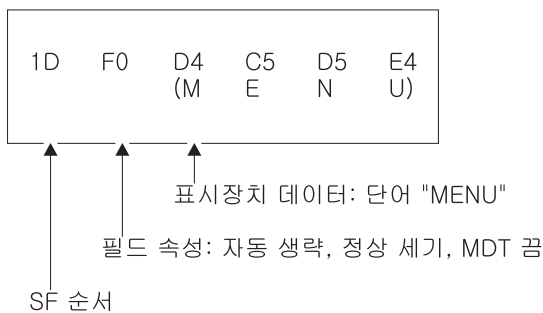


그림 123. SF 순서를 사용하는 필드 정의

확장 속성을 지정해야 하고 터미널이 이를 지원하는 경우에는 시작 필드 확장 순서를 대신 사용해야 합니다. 보다 복잡한 속성 정보 때문에 SFE에는 다른 형식이 필요합니다. 확장 속성은 바이트 쌍으로 표현됩니다. 첫 번째 바이트는 어떤 속성 유형이 정의되고 있는지를 나타내는 코드이고 두 번째 바이트는 해당 속성의 값입니다. 필드 속성은 집합적으로 추가 속성 유형으로 취급되고 또한 바이트 쌍으로 표현됩니다. SFE 순서 직후에, 속성 쌍의 1바이트 계수를 제공한 다음, 속성 쌍, 그리고 마지막으로 표시 데이터를 제공합니다. 전체 순서는 765 페이지의 그림 124에 표시되어 있습니다.

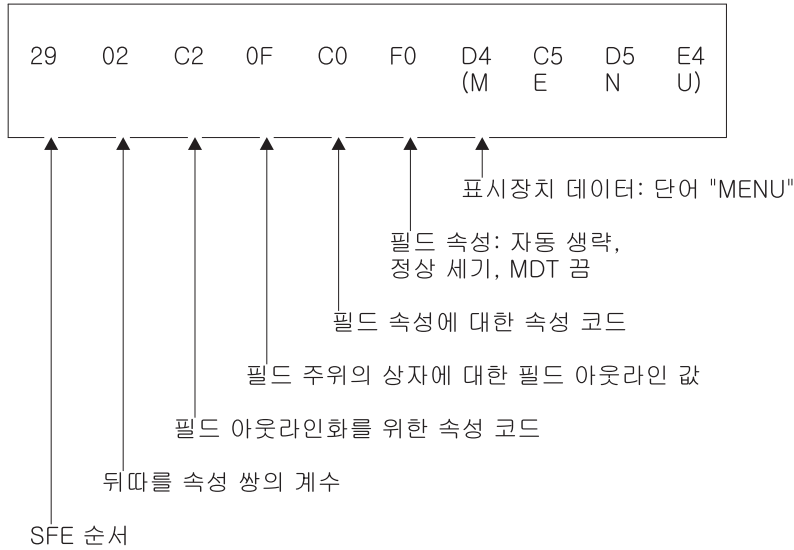


그림 124. SFE 순서를 사용하는 필드 정의

수정 필드 순서

필드가 화면에 있으면 수정 필드(MF)라고 불리는 SFE와 형식이 거의 동일한 명령으로 이를 변경할 수 있습니다.

SFE와의 유일한 차이는 다음과 같습니다.

- 필드가 존재해야 합니다.
- 명령 코드는 X'29' 대신에 X'2C'입니다.
- 현재 값에서 변경하고 싶은 속성만을 보내고, 변경하고 싶은 표시 데이터만 보냅니다.
- 널값은 속성을 특정 터미널의 기본값으로 설정합니다. 속성을 생략하여 SFE 순서로 이를 달성할 수 있습니다.

예를 들어, 앞의 예제의 "메뉴" 필드를 터미널의 기본 색상으로 변경하고 밑줄을 추가하려면 766 페이지의 그림 125에서 순서가 필요합니다.

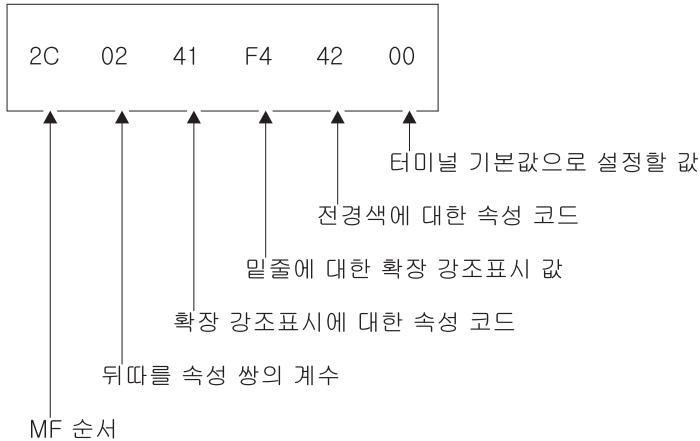


그림 125. MF 순서 내에서 필드 속성 변경

설정 버퍼 주소 순서

SF 및 SFE 순서는 정의하고 있는 필드를 버퍼의 현재 위치에 두고, MF는 이 위치에서 필드를 수정합니다. 필드가 전송된 마지막 문자를 따르지 않는 한(즉, 현재 버퍼 위치에서 시작함), 필드를 배치하거나 변경하려는 위치를 표시하기 위해 이러한 순서 앞에 설정 버퍼 주소(SBA) 순서를 배치해야 합니다.

이를 수행하려면 그림 126에서처럼 2바이트 주소가 뒤따르는 SBA 순서를 전송하십시오.

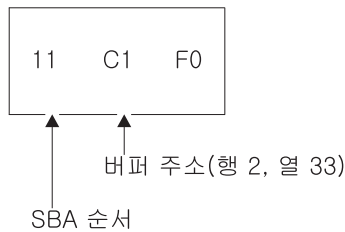


그림 126. SBA 순서

그림의 주소는 80열 화면에서 행 2, 열 33인 위치 112(X'70')의 "12비트" 주소입니다. 카운팅은 첫 번째 행과 열(0 위치)에서 시작되고 행을 따라 진행됩니다. "14비트"와 "16비트"의 두 개의 서로 다른 주소 지정 설계가 사용되었습니다. 버퍼 위치는 모두가 순차적으로 번호 지정되지만 12비트와 14비트 주소 지정에서는 주소의 모든 비트가 사용되지는 않으므로 순차적으로 나타나지 않습니다. (X'70'(B'1110000') 그림에서는 주소의 맨 오른쪽 바이트의 낮은 순서 6비트에서 B'110000'로 나타나고 왼쪽 바이트의 낮은 순서 6비트에서는 B'000001'로 나타납니다.) IBM 3270 정보 표시 시스템 데이터 스트림 프로그래머의 참조 매뉴얼은 주소를 형성하는 방법을 설명합니다.

SF, SFE 또는 MF 순서 후에 현재 버퍼 주소는 데이터가 있는 경우 데이터 직후에, 또는 데이터가 없는 경우에는 필드 속성 직후에 채우지 않은 버퍼에서 첫 번째 위치를 가리킵니다.

설정 속성 순서

단일 문자 위치의 속성을 설정하려면 지정하려는 각 속성의 설정 속성(SA) 순서를 사용하십시오.

예를 들어, 문자를 감박이게 만들려면 그림 127에서 순서가 필요합니다.

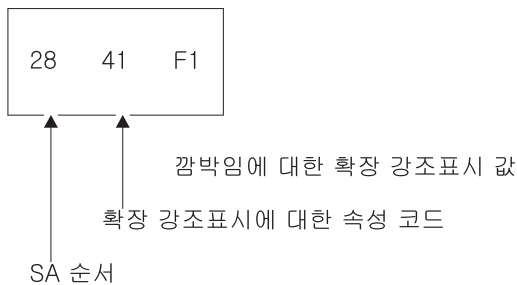


그림 127. 문자를 감박이게 하기 위한 SA 순서

SA 순서를 사용하여 지정하는 속성은 현재 버퍼 위치에 지정되고, 같은 방법으로 해당 필드 정의는 현재 버퍼 위치에 배치되므로 일반적으로 SA 앞에 SBA 순서를 배치해야 합니다.

아웃바운드 데이터 스트림 예제

아웃바운드 데이터 스트림 예제는 데이터 스트림을 빌드하는 방법에 대한 설명을 보강하기 위해 특정 3270 화면을 그리기 위해 필요한 데이터 스트림을 보여줍니다.

그림 128에서는 직원들이 직장에서 사용하는 자동차를 추적하고 새 자동차를 기록하는 데 사용되는 애플리케이션의 일부인 예제 화면을 보여줍니다. 유일한 입력은 직원 식별 번호, 자동차 번호판(태그) 및 자동차가 다른 주에서 온 경우에는 라이선싱 상태입니다.

직원 번호: _____

자동차 레코드
태그 번호: _____

상태: _____

그림 128. 데이터 입력 화면의 예제

참고: 이는 설명을 다루기 쉽게 짧게 만들기 위해 설계된 비현실적으로 단순한 화면입니다. 이는 화면 설계의 승인된 표준을 따르지 않으므로 이를 모델로 사용해서는 안 됩니다.

이 화면에는 8개의 필드가 있습니다.

1. 행 1 열 26에 화면 제목, "자동차 레코드"
2. 레이블 필드 "직원 번호"(행 3, 열 1)은 운영자가 다음 필드에 입력할 내용을 표시합니다.
3. 6개 자리 길이의 직원 번호 입력 필드(행 3, 열 14)
4. 레이블 필드, "태그 번호:" 행 3, 열 21
5. 행 3, 열 31에 8개 자리 길이의 입력 필드(태그 번호)
6. 행 3, 열 40의 레이블 필드 "상태"
7. 행 3, 열 49에 2개 자리 길이의 입력 필드(상태)
8. 행 3, 열 52에 이전(상태) 입력 필드의 끝을 표시하기 위한 필드

표 40에서는 아웃바운드 데이터 스트림을 보여줍니다.

표 40. 3270 출력 데이터 스트림

바이트	내용	참고
1	X'F5'	데이터 스트림을 시작하는 3270 명령으로서 이 경우에는 erase/write
2	X'C2'	WCC; 이 값은 키보드를 잠금 해제하지만 알람을 울리거나 MDT를 재설정하지 않습니다.
3	X'11'	첫 번째 필드를 배치하기 위한 SBA 순서...
4-5	X'40D6'	12비트 주소 지정을 사용하여 24*80 화면에서 행 1, 열 23의 주소
6	X'1D'	첫 번째 필드 정의를 시작하기 위한 SF 순서
7	X'F8'	필드 속성 바이트. 이 조합은 MDT가 처음에는 꺼져 있는 자동 건너뛰기와 밝기인 필드를 표시합니다.
8-17	'자동차 레코드'	필드의 내용을 표시합니다.
18-20	X'11C260'	현재 버퍼 위치를 두 번째 필드의 행 3, 열 1로 재설정하기 위한 SBA 순서.
21	X'1D'	두 번째 필드의 SF 순서.
22	X'F0'	필드 속성 바이트. 자동 건너뛰기, 보통 강도, MDT 꺼짐.
23-34	'직원 번호:'	필드의 내용을 표시합니다.
35	X'29'	네 번째 필드를 시작하기 위한 SFE 순서. SFE는 SF 대신에 필수입니다. 확장 속성을 지정해야 하기 때문입니다. 이 필드는 이전 필드가 끝난 직후에 시작하므로 앞에 SBA 순서를 둘 필요가 없습니다.
36	X'02'	지정된 속성 유형의 계수(여기에서는 2: 필드 아웃라인과 필드 속성).
37	X'41'	확장 강조표시의 속성 유형을 표시하는 코드.
38	X'F4'	밑줄을 표시하는 확장 강조표시 값.
39	X'C0'	필드 속성의 속성 유형을 표시하는 코드.
40	X'50'	숫자 전용, 보통 강조, MDT 꺼짐의 필드 속성 값. 이 필드의 초기 데이터가 다음에 나타날 수 있지만 아무것도 없습니다.

표 40. 3270 출력 데이터 스트림 (계속)

바이트	내용	참고
41	X'13'	3270에 커서를 현재 버퍼 위치에 배치하라고 알려주는 삽입 커서(IC) 순서. 운영자가 데이터를 입력해야 하는 첫 번째 필드의 시작 부분인, 현재 버퍼 위치에 배치하기를 원합니다.
42-44	X'11C2F4'	직원 번호에 필요한 6개 자리를 남기기 위해 행 3, 열 21에 배치하기 위한 SBA 순서. "태그 번호" 레이블 필드의 시작 부분은 직원 번호 입력 필드의 끝 부분을 표시하므로 사용자는 너무 긴 숫자를 입력하려고 하면 즉시 알게 됩니다.
45	X'1D'	필드를 시작하기 위한 SF 순서.
46	X'F0'	필드 속성 바이트. 자동 건너뛰기, 보통 강도, MDT 꺼짐.
47-55	'태그 번호:'	표시장치 데이터. 필드 간에 더 많은 공백을 위해 레이블 앞에 두 개의 공백을 첨부합니다. (별도의 필드를 사용할 수도 있었지만 몇몇 문자만 사용해도 되므로 이것이 더 쉬운 방법입니다.)
56	X'29'	SFE(다음 필드는 필드 아웃라인을 원하는 또 다른 입력 필드이므로 SFE를 다시 사용합니다.)
57	X'02'	속성 유형의 계수.
58-59	X'41F4'	밀줄 값이 있는 확장 강조표시의 코드.
60-61	X'C040'	필드 속성의 코드 및 보호되지 않은 속성, 보통 강조, MDT 꺼짐.
62-64	X'11C3C7'	태그를 위해 8자리를 남기기 위해 행 3, 열 40에 다시 배치하기 위한 SBA 순서.
65	X'1D'	필드를 시작하기 위한 SF.
66	X'F0'	필드 속성 바이트. 자동 건너뛰기, 보통 강도, MDT 꺼짐.
67-74	'상태:'	필드 데이터(간격을 위해 다시 한 번 앞부분에 2개의 공백).
75-80	X'290241F4C040'	상태 입력 필드의 SFE 순서 및 속성 스펙(속성은 태그 입력 필드의 속성과 동일함).
81-82	X'0000'	상태 필드의 (초기) 내용. 다른 입력 필드에서처럼 이 값을 생략할 수도 있었지만 현재 버퍼 위치를 필드 끝으로 이동하기 위해서는 SBA 순서가 필요하고 이것이 더 짧습니다.
83	X'1D'	SF. 마지막 필드는 이전 필드의 끝부분을 표시하므로 사용자는 상태 코드에 두 개 문자 이상을 입력하려고 시도하지 않게 됩니다. 여기에는 초기 데이터가 없고 속성 바이트만이 있습니다. 이 필드는 때로 "스토퍼" 필드라고 불립니다.
84	X'F0'	필드 속성 바이트. 자동 건너뛰기, 보통 강도, MDT 꺼짐.

참고: 터미널 제어 명령을 사용하고 사용자 고유의 데이터 스트림을 빌드하는 경우에는 SEND 명령의 FROM 매개변수에 제공하는 데이터는 테이블에서 3바이트에 시작합니다. CICS는 SEND 명령의 옵션에서 쓰기 명령 및 WCC를 제공합니다.

3270 터미널에서의 입력

키보드 입력은 버퍼를 통해 호스트에 도달합니다. 3270 터미널에는 사용 가능한 여러 개의 많은 키보드 배열이 있지만 어떤 배열에서든지 키는 데이터 키, 키보드 제어 키 또는 어텐션 키의 세 카테고리 중 하나에 해당합니다.

데이터 키

데이터 키에는 모든 친숙한 문자, 숫자, 구두점 표시 및 특수 문자가 포함됩니다. 데이터 키를 누르면 커서로 표시되는 위치에서 버퍼의 내용(및 화면)이 변경됩니다. 커서는 화면상의 위치(즉, 버퍼)에 대한 가시적 포인터이며 여기에 다음 데이터 키 스트로크가 저장됩니다. 운영자가 데이터를 입력하면 커서는 화면 상의 다음 위치로 이동하고, 화면에서 형식화된 자동 건너뛰기 속성이 정의된 필드는 건너뜁니다.

키보드 제어 키

키보드 제어 키는 커서를 새 위치, 지우기 필드 또는 개별 버퍼 위치로 이동하고, 문자가 삽입되게 하거나 키보드가 버퍼를 수정하는 위치나 방법을 변경합니다.

주의 키

이전 그룹, 데이터 및 키보드 제어 키의 키는 호스트와의 상호작용을 일으키지 않습니다. 이들은 전적으로 디바이스와 해당 제어 장치에 의해 처리됩니다. 버퍼가 호스트에 대한 전송이 준비되었음을 신호하는 어텐션 키. 호스트가 터미널에 읽기를 실행하면 CICS에서의 일반적인 상황에서는 이 경우 전송이 발생합니다.

주의 키에는 5개의 유형이 있습니다.

- ENTER
- F(함수) 키
- CLEAR
- PA(프로그램 어텐션) 키
- CNCL(취소 키, 일부 키보드 모델에만 있음)

주의 키를 누르는 것 외에도 전송을 일으킬 수 있는 다른 운영자 조치가 있습니다.

- 식별 카드 리더 사용
- 자기 슬롯 리더 또는 핸드 스캐너 사용
- 라이트 펜 또는 커서 선택 키로 어텐션 필드 선택
- 트리거 필드에서 커서 이동

트리거 필드 기능은 일부 터미널 모델에서는 확장 속성이 제공되지만 이전에 나열된 다른 모든 조치에는 특수 하드웨어가 필요하고 대부분의 경우 화면(버퍼)가 사전에 적절하게 설정되어야 합니다. 이러한 기능에 대해서는 1010 페이지의 『특수 하드웨어에 대한 지원』에서 이야기합니다. 이 섹션에서는 표준 기능에 대해 집중합니다.

AID

3270은 인바운드 데이터 스트림의 첫 번째 바이트에서 인코딩된 값에 의한 전송을 초래하는 키를 식별합니다. 이 값은 어텐션 ID 또는 AID라고 부릅니다.

일반적으로 터미널 운영자가 데이터를 전송하기 위해 선택하는 키는 애플리케이션 설계자가 지시합니다. 설계자는 다양한 어텐션 키에 특정 의미를 지정하고, 사용자는 애플리케이션을 사용하기 위해서는 이러한 의미에 대해 알아야 합니다. (종종, 이러한 몇몇 개의 키만이 사용됩니다. 예를 들어, 일반 입력을 위한 ENTER, 애플리케이션에 제어를 종료하기 위한 하나의 기능 키, 부분적으로 완료된 트랜잭션 순서를 취소하기 위한 또 다른 키입니다. 여러 선택사항이 있으면 키 정의를 화면에 나열할 수 있으므로 사용자는 이들을 모두 기억할 필요는 없습니다.)

두 그룹의 어텐션 키 간에는 중요한 차이점이 있는데 애플리케이션 설계자는 이를 유의해야 합니다. 호스트가 CICS가 일반적으로 사용하는 명령인 "수정된 읽기" 명령을 발행할 때 ENTER 및 기능 키는 버퍼에서 데이터를 전송합니다. CLEAR, CNCL 및 PA 키는 AID(즉, 사용된 키의 ID)를 언더라도 데이터를 전송하지 않습니다. 이들은 축약형 읽기 키라고 불립니다. 이들은 "취소" 등과 같은 단순 요청을 전달하는 데 유용하지만 동반 데이터를 필요로 하는 요청에는 유용하지 않습니다. 실제로, 많은 설계자들은 비데이터 요청이더라도 기능 키를 사용하고 동반 데이터는 무시합니다.

참고: CLEAR 키에는 전체 버퍼를 널(null)로 설정하는 추가적인 효과가 있으므로 문자 그대로 전송할 데이터가 없습니다. CLEAR는 또한 터미널에 대체 화면 크기 기능이 있는 경우 화면 크기를 기본값으로 설정하고 화면을 773 페이지의 『형식화되지 않은 모드』에 설명된 대로 형식화되지 않은 모드에 넣습니다.

3270 터미널에서 읽기

3270에는 읽기 버퍼와 수정된 읽기의 두 개의 기본 읽기 명령이 있습니다.

두 명령 모두 인바운드 데이터 스트림은 다음으로 구성된 3바이트 읽기 헤더로 시작합니다.

- 어텐션 ID(AID), 1바이트
- 커서 주소, 2바이트

이전 섹션에서 설명된 대로, AID는 어떤 조치나 어텐션 키가 전송을 일으키는지를 표시합니다. 커서 주소는 전송 당시에 커서의 위치를 표시합니다. CICS는 이 정보를 RECEIVE 명령 완료 시에 EIB에 EIBAID 및 EIBCPOSN 위치에 저장합니다.

읽기 버퍼 명령은 읽기 헤더 뒤에 전체 버퍼를 가져오고, 수신 프로그램은 위치를 기반으로 원하는 정보를 추출할 책임이 있습니다. 이는 주로 진단 및 기타 특수 용도를 위한 것이며 CICS는 이를 BUFFER 옵션이 지정된 경우에만 RECEIVE 명령을 실행할 때 사용합니다. CICS는 요구하지 않은 터미널 입력을 읽기 위해 읽기 버퍼를 사용하지 않으므로 BUFFER 옵션은 이 방식으로 시작된 트랜잭션의 첫 번째 RECEIVE에서 사용할 수 없습니다.

CICS가 일반적으로 사용하는 명령인 수정된 읽기를 사용하면 훨씬 더 적은 데이터가 전송됩니다. 축약형 읽기 키(CLEAR, CNCL 및 PA)의 경우 읽기 헤더만이 들어옵니다. 다른 어텐션 키(ENTER 및 PF)의 경우, 변경된 화면의 필드(정확히 말하면 MDT가 커짐)는 읽기 헤더를 따릅니다. 이 형식에 대해서는 다음 섹션에서 설명합니다. 트리거 필드, 라이트 펜 발견 또는 커서 선택 때문에 전송이 발생하면 정보의 양과 형식은 약간 다릅니다. 이러한 특수 형식은 1010 페이지의 『특수 하드웨어에 대한 지원』에서 설명됩니다. SCS 프린터에서 프로그램 어텐션 키에서의 입력은 또한 예외입니다. 해당 데이터 스트림의 설명에 대해서는 885 페이지의 『SCS 입력』의 내용을 참조하십시오.

인바운드 필드 형식

터미널 제어 명령을 사용 중인 경우에는 3270이 데이터를 전송하는 형식을 알고 있어야 합니다.

BMS를 사용 중인 경우에는 BMS가 입력을 자동으로 변환해주므로 773 페이지의 『형식화되지 않은 모드』으로 건너뛸 수 있습니다.

각 수정된 필드는 다음과 같습니다.

- SBA 순서
- 필드의 첫 번째 데이터 위치의 2바이트 주소
- SF 순서
- 필드 내용

필드의 널이 아닌 문자만이 전송됩니다. 널은 나타나는 곳 어디에서나 건너뛹니다. 항목이 필드를 채우지 않았고, 필드가 처음에 널이었으면, 키 지정 문자만이 전송되어 인바운드 데이터의 길이를 줄여줍니다. 널(null)(X'00')은 화면에서 구별할 수 없을지라도 공백(X'40')과 같지 않습니다. 공백은 전송되므로 일반적으로 전송을 최소화하려면 필드를 공백이 아니라 널(null)로 초기화합니다.

3270 읽기 명령은 터미널이 필드 내용과 함께 속성 값을 리턴해야 함을 지정할 수 있지만 CICS는 이 옵션을 사용하지 않습니다. 결과적으로, 버퍼 주소는 앞선 속성 바이트가 아니라 필드 데이터의 첫 번째 바이트의 위치입니다(해당하는 아웃바운드 데이터 스트림에서처럼).

참고: 커서 선택 키, 트리거 필드, 자기 슬롯 리더 등등과 같은 입력을 위한 3270의 특수 기능은 다양한 입력 형식을 생성합니다. 자세한 내용은 1013 페이지의 『필드 선택 사항 기능』의 내용을 참조하십시오.

입력 데이터 스트림 예제

이 예제는 이전 아웃바운드 데이터 스트림 예제로부터 데이터 입력 화면을 사용하여 작성되는 입력 데이터 스트림을 보여줍니다.

아웃바운드 데이터 스트림을 설명하려면 767 페이지의 그림 128에 표시된 화면을 사용하는 운영자가 다음을 수행한 것으로 가정합니다.

- 직원 ID 필드에 "123456" 놓기
- 태그 번호에 "ABC987" 놓기
- ENTER를 눌러 마지막 상태 필드 생략하기

다음은 결과로 나오는 인바운드 데이터 스트림입니다.

표 41. 3270 입력 데이터 스트림

바이트	내용	참고
1	X'7D'	AID, 이 경우에는 ENTER 키입니다.
2-3	X'C3C5'	커서 주소: 행 3, 열 38. 운영자가 이전에 마지막 데이터 키 스트로크 이후에 사용한 위치입니다.
4	X'11'	SBA, 버퍼 주소가 뒤따르는 내용을 나타냅니다.
5-6	X'C26E'	행 3, 열 15의 주소이며 이는 따를 필드의 시작 위치입니다.
7-12	'123456'	입력, 운영자가 입력한 고용 번호.
13-15	X'11C3D1'	행 3, 열 32의 버퍼 주소를 나타내는 SBA 순서.
16	X'1D'	SF, 뒤따르는 또 다른 입력 필드를 나타냅니다.
17-22	'ABC987'	입력 필드: 플레이트 번호. 8자 길이였던 필드로부터 6자만이 입력되었음을 유의하십시오. 운영자가 나머지 널(null) 위치를 완료하지 않았기 때문입니다.

세 번째 입력 필드(상태 코드)는 입력 데이터 스트림에 나타나지 않습니다. 이는 MDT가 켜지지 않았기 때문입니다. 처음에는 꺼져 있는데 운영자가 필드에 입력하여 이를 켜지 않았습니다. CICS는 일반적으로 Read Modified All을 실행하므로 바이트 7에서 SF가 필요하지 않는다는 점 또한 유의하십시오.

형식화되지 않은 모드

3270의 고 기능은 해당 필드 구조를 중심으로 생기므로 형식화되지 않은 모드라고 불리는 모드에서 필드 없이 3270을 사용하는 것이 가능합니다. 이 모드에서는 정의된 필드가 없고 전체 화면(버퍼)이 이전의 더 단순한 터미널처럼 인바운드 및 아웃바운드 데이터의 단일 문자열처럼 작동합니다.

형식화되지 않은 모드로 쓸 때에는 데이터를 화면에서 특정 위치로 지시하기 위해 SBA 순서를 포함할 수는 있지만 데이터에 필드를 정의하지 않습니다. SBA 순서를 선행하는 데이터는 커서의 현재 위치에서 시작하여 작성됩니다. (지우기 또는 쓰기 명령을 사용하는 경우에는 커서는 화면의 왼쪽 상단 구석에서 자동으로 0으로 설정됩니다.)

형식화되지 않은 화면을 읽을 때 첫 번째 3바이트는 형식화된 화면을 읽을 때처럼 읽기 헤더(AID 및 커서 주소)입니다. 나머지 바이트는 0 위치에서 시작하는 전체 버퍼의 내용입니다. 어떤 필드도 없기 때문에 SBA나 SF 순서가 제공되지 않습니다. 읽기 명령

이 수정된 것으로 읽힌 경우에는 널이 억제되므로 입력 데이터가 화면에서 정확히 어디에 있는지를 판별하는 것이 항상 가능하지는 않습니다.

형식화되지 않은 화면을 읽기 위해 BMS RECEIVE MAP 명령을 사용할 수 없습니다. BMS는 966 페이지의 『MAPFAIL 및 기타 예외 조건』에 설명된 대로 형식화되지 않은 입력을 발견할 때 MAPFAIL 조건을 발생시킵니다. CICS에서 터미널 제어 RECEIVE 명령을 사용해서만 형식화되지 않은 데이터를 읽을 수 있습니다.

참고: CLEAR 키는 버퍼를 널로 설정하고 필드의 한계를 정하는 모든 속성 바이트를 지우므로 화면이 형식화되지 않은 모드가 됩니다.

제 9 부 간격 제어 및 태스크 제어

태스크 타이밍 및 활동 제어.

제 35 장 간격 제어

CICS 간격 제어 서비스는 관련되는 기능을 제공합니다.

Java 및 C++

여기에 설명된 API(Application Programming Interface)는 EXEC CICS API로서 Java 프로그램에서 사용하지 않습니다. 간격 제어 서비스에 액세스하기 위해 JCICS 클래스를 사용하는 Java 프로그램에 대한 정보는 애플리케이션 개발에서 JCICS를 사용하는 Java 프로그래밍 및 JCICS Javadoc 문서를 참조하십시오. CICS C++ 클래스를 사용하는 C++ 프로그램에 대한 정보는 애플리케이션 개발에서 CICS C++ OO 기반 클래스 사용의 내용을 참조하십시오.

간격 제어 명령을 사용하여 다음을 수행할 수 있습니다.

- 지정된 시간에 또는 특정 간격 후에 태스크를 시작하고, 데이터를 태스크로 전달합니다(START 명령).
- START 명령에서 전달된 데이터를 검색합니다(RETRIEVE 명령).
- 태스크의 처리를 지연합니다(DELAY 명령).
- 지정된 시간이 만기될 때 알람을 요청합니다(POST 명령).
- 이벤트가 발생하기를 기다립니다(WAIT EVENT 명령).
- 이전 간격 제어 명령의 효과를 취소합니다(CANCEL 명령).
- 현재 날짜 및 시간을 요청합니다(ASKTIME 명령).
- 날짜 및 시간의 형식을 선택합니다(FORMATTIME 명령). 21세기의 날짜를 처리할 수 있도록 도와주는 옵션을 사용할 수 있습니다.

참고: EXEC CICS START TRANSID() TERMID(EIBTRMID)를 사용하여 원격 트랜잭션을 시작하지 마십시오. 대신 EXEC CICS RETURN TRANSID() IMMEDIATE를 사용하십시오. 이 방식으로 사용되는 START는 불필요하게 통신 자원을 묶어서, 연결된 리전에서 성능 저하를 유도할 수 있습니다.

WAIT EVENT, START, WAIT 옵션이 있는 RETRIEVE, CANCEL, DELAY 또는 POST 명령을 사용하는 경우, 동적 트랜잭션 라우팅을 수행하기 위해 사용자 능력에 반대의 영향을 주는 트랜잭션간 연관관계를 작성할 수 있습니다.

WAIT EVENT의 타이머 이벤트 제어 영역에 대한 저장영역은 ISOLATE(YES)를 지정한 경우 공유 저장영역에 상주해야 합니다.

CICS가 트랜잭션 분리를 사용하거나 사용하지 않고 실행 중인 경우, CICS는 타이머 이벤트 제어 영역이 읽기 전용 저장영역에 있는지 확인합니다.

이러한 명령을 실행하는 프로그램의 잠재된 문제점을 쉽게 식별하기 위해 CICS Interdependency Analyzer를 사용할 수 있습니다. 이 유틸리티에 대한 자세한 정보는 CICS Interdependency Analyzer for z/OS의 내용을 참조하고, 트랜잭션 연관관계에 대한 자세한 정보는 379 페이지의 『연관관계』의 내용을 참조하십시오.

요청 ID

요청과, 요청과 연관된 데이터를 식별하는 수단으로, CICS가 각 DELAY , POST 및 START 명령에 고유한 요청 ID를 지정됩니다. REQID 옵션을 사용하여 사용자 고유의 요청 ID를 지정할 수 있습니다. 그렇지 않은 경우, CICS는 (POST 및 START 명령에 대해서만) 고유한 요청 ID를 지정하고 EXEC 인터페이스 블록(EIB)의 EIBREQID 필드에 위치시킵니다. CANCEL 명령에 의해 나중에 요청이 취소되도록 하려면 요청 ID를 지정해야 합니다.

만기 시간

시간 제어 함수가 시작될 시간은 만기 시간으로 알려져 있습니다. 만기 시간을 하루 중의 시간(TIME 옵션을 사용하여)으로 절대적으로 지정하거나 함수가 수행되기 전에 경과할 간격(INTERVAL 옵션을 사용하여)으로 지정할 수 있습니다.

DELAY 명령의 경우 FOR 및 UNTIL 옵션을 사용할 수 있고, POST 및 START 명령의 경우 AFTER 및 AT 옵션을 사용할 수 있습니다.

참고: C 및 C++ 언어는 TIME 및 INTERVAL 옵션이 사용하는 팩형 10진수 유형에 대한 지원을 제공하지 않습니다.

간격을 사용하여 CICS에 현재 시간으로부터 지정된 시간, 분 및 초 안에 트랜잭션을 시작하도록 지시합니다. 0이 아닌 INTERVAL 값은 항상 미래의 시간, 즉 현재 시간 더하기 사용자가 지정하는 간격을 표시합니다. 시간은 0 - 99 범위일 수 있지만, 분과 초는 59를 넘어서는 안됩니다. 예를 들어, 40시간 10분에 태스크를 시작하려면 다음과 같이 코딩합니다.

```
EXEC CICS START INTERVAL(401000)
```

절대 시간을 사용하여 CICS에 hhmmss를 다시 사용하는 특정 시간에 트랜잭션을 시작하도록 지시할 수 있습니다. 예를 들어, 오후 3시 30분에 트랜잭션을 시작하려면 다음과 같이 코딩합니다.

```
EXEC CICS START TIME(153000)
```

절대 시간은 항상 현재 시간 이전의 자정에 상대적으로 따라서 현재 시간보다 더 빠를 수 있습니다. TIME은 명령이 실행되는 시간에 상대적으로 미래 또는 과거일 수 있습니다. CICS는 다음 규칙을 사용합니다.

- 이전 6시간 이내의 언제든지 시작할 태스크를 지정하는 경우 태스크를 즉시 시작합니다. 이것은 이전 6시간이 자정을 포함하는지와 상관없이 발생합니다. 예를 들어, 다음과 같습니다.

```
EXEC CICS START TIME(123000)
```

이 명령은 월요일 05:00 또는 07:00에서 발행되어 같은 날 12:30에 만기합니다.

```
EXEC CICS START TIME(020000)
```

이 명령은 월요일 05:00 또는 07:00에 실행되어 즉시 만기합니다. 지정된 시간이 앞선 6시간 안에 있기 때문입니다.

```
EXEC CICS START TIME(003000)
```

월요일 05:00에 실행되는 이 명령은 지정된 시간이 앞선 6시간 안에 있기 때문에 즉시 만기합니다. 그러나 월요일 07:00에 실행되는 경우 화요일 00:30에 만기합니다. 지정된 시간이 앞선 6시간 이내가 아니기 때문입니다.

```
EXEC CICS START TIME(230000)
```

월요일 02:00에 실행되는 이 명령은 지정된 시간이 앞선 6시간 이내에 있기 때문에 즉시 만기합니다.

- 23보다 더 큰 시간 구성요소를 갖는 시간을 지정하는 경우, 현재 날보다 하루 앞선(어제) 시간을 지정하고 있습니다. 예를 들어, 250000의 시간은 어제 오전 1시를 나타내고, 490000의 시간은 그저께 오전 1시를 나타냅니다.

DELAY, POST 또는 START 명령에서 만기 시간이나 간격 옵션을 지정하지 않는 경우, CICS는 즉시를 의미하는 INTERVAL(0)의 기본값을 사용하여 응답합니다.

상호 시스템 링크의 양쪽 끝이 서로 다른 시간대에 있을 수 있기 때문에, 트랜잭션이 원격 시스템에서 시작될 때는 만기 시간의 INTERVAL 양식을 사용해야 합니다.

시스템이 실패하는 경우, 만기하지 않은 START 명령과 연관된 시간이 재시작을 지나도 기억됩니다.

참고:

1. 약하게 사용되는 시스템에서는, 지정되는 간격 시간이 1초의 1/4만큼 초과될 수 있습니다.
2. 만기 시간이 가능한 CICS 종료 이내에 들어가는 경우, 태스크가 실행하려고 시도하기 전에 CICS의 상태를 테스트할지 여부를 고려해야 합니다. **INQUIRE SYSTEM**의 CICSSTATUS 옵션을 사용하여 이를 수행할 수 있습니다. 정상 종료 중에 태스크는 사용자에게만 알려지는 결과를 갖고 PLT 프로그램과 동시에 실행할 수 있습니다.

제 36 장 태스크 제어

CICS 태스크 제어 기능은 태스크 활동을 동기화하거나 자원 사용을 제어하는 기능을 제공합니다.

Java 및 C++

여기에 설명된 API(Application Programming Interface)는 Java 프로그램에서 사용되지 않은 EXEC CICS API입니다. 태스크 제어 서비스 CICS에 액세스하기 위해 JCICS 클래스를 사용하는 Java 프로그램에 대한 정보는 애플리케이션 개발에서 JCICS를 사용하는 Java 프로그래밍 및 JCICS Javadoc 문서를 참조하십시오. CICS C++ 클래스를 사용하는 C++ 프로그램에 대한 정보는 애플리케이션 개발에서 CICS C++ OO 기반 클래스 사용의 내용을 참조하십시오.

CICS는 CICS 시스템 프로그래머가 설정한 값을 기준으로 우선순위를 지정합니다. 프로세서 제어는 처리 준비가 완료된 우선순위가 가장 높은 태스크에 지정되며 더 이상 CICS 또는 애플리케이션에서 수행할 후속 작업이 없으면 운영 체제로 리턴됩니다.

다음은 수행할 수 있습니다.

- 우선순위가 높은 태스크가 처리되도록 태스크를 일시중단(SUSPEND 명령)합니다. 그러면, 프로세서를 많이 사용하는 태스크가 프로세서를 독점하는 것이 방지됩니다. 다른 적합한 태스크가 선행 및 종료되거나 처리를 일시중단한 경우, 제어는 발행 중인 태스크로 리턴됩니다. 즉, 태스크는 처리 가능 상태로 유지됩니다.
- 태스크별로 자원 사용을 스케줄하십시오(ENQ 및 DEQ 명령). 이는 둘 이상의 태스크가 자원을 동시에 사용하는 것을 방지하기 위해 유용할 때가 있습니다. 즉, 해당 자원을 직렬로 다시 사용할 수 있도록 만듭니다. 자원을 사용하는 각 태스크는 큐에 넣기 명령(ENQ)을 실행해야 합니다. 이를 수행하는 첫 번째 태스크는 즉시 자원을 사용할 수 있지만 HANDLE CONDITION ENQBUSY 명령이 실행되지 않았다면 다른 태스크가 발행한 자원에 대한 후속 ENQ 명령으로 인해 해당 태스크는 자원이 사용 가능해질 때까지 일시중단됩니다.

NOSUSPEND 옵션이 ENQ 명령에 코딩되면 제어는 항상 프로그램에서 다음 지시 사항으로 리턴됩니다. EIBRESP 필드 내용을 검사하여 ENQ 명령 성공 여부를 확인할 수 있습니다.

자원을 사용하는 각 태스크는 자원 사용을 종료했을 때 큐에서 해제 명령(DEQ)을 실행해야 합니다. 그렇지만 큐에 넣기/큐에서 해제 메커니즘을 사용할 때 ENQ 및 DEQ 명령을 실행하는 둘 이상의 태스크가 서로 연관된 지정 순서로 해당 명령을 실행

행하는 것은 보장할 수 없습니다. 액세스 순서 제어 방법에 대해서는 783 페이지의 『자원에 대한 액세스의 순서 제어』의 내용을 참조하십시오.

- 태스크에 지정된 우선순위를 변경하십시오(CHANGE TASK PRIORITY 명령).
- MVS 형식 ECB 완료 시 이를 게시하는 이벤트를 대기하십시오.

WAITCICS 및 WAIT EXTERNAL의 두 명령을 사용할 수 있습니다. 이 명령으로 인해 발행 태스크는 ECB 중 하나가 게시될 때까지 즉, 이벤트 중 하나가 발생할 때까지 일시중단됩니다. 태스크는 하나 이상의 ECB를 대기할 수 있습니다. 둘 이상의 ECB를 대기하는 경우 ECB 중 하나가 게시될 때 태스크가 실행됩니다. 각 ECB가 게시될 수 있는 가장 빠른 시간보다 늦지 않게 지워지도록(2진 0으로 설정) 해야 합니다. CICS가 이를 대신 수행할 수 없습니다. 이전에 게시되었지만 아직 지워지지 않은 ECB를 대기하는 경우, 태스크는 일시중단되지 않고 WAITCICS 또는 WAIT EXTERNAL이 실행되지 않더라도 계속 실행됩니다.

WAIT EXTERNAL은 일반적으로 오버헤드가 적지만 연관된 ECB는 항상 MVS POST 기능을 사용하거나 최적화된 게시(비교 및 스왑(CS) 지시사항 사용)를 통해 게시되어야 합니다. 다른 방법으로 게시되면 안됩니다. 게시 방법에 확신이 없다면 WAITCICS 명령을 사용하십시오. WAIT EXTERNAL 명령으로 전달된 ECB를 처리하는 경우, CICS는 ECB를 확장하고 MVS POST 엑시트 기능을 사용합니다. 지정된 ECB는 한 번에 둘 이상의 태스크에서 대기(또는 임의 태스크의 ECBLIST에 두 번 표시) 중이면 안됩니다. 이 규칙을 준수하지 않으면 INVREQ 응답이 표시됩니다.

WAITCICS는 ECB가 MVS POST 기능 또는 최적화된 게시 이외의 방법으로 게시되는 경우에 사용되어야 합니다. 예를 들어, 애플리케이션이 값을 이동하여 ECB를 게시하는 경우 WAITCICS를 사용해야 합니다. (WAITCICS 명령은 MVS POST 기능이나 최적화된 게시를 사용하여 게시되는 ECB에도 사용할 수 있습니다.) CICS가 MVS WAIT로 이동하면 WAITCICS 명령을 실행한 태스크로 대기 중인 모든 ECB의 MVS에 목록을 전달합니다. MVS WAIT에서 CICS가 전달한 ECBLIST는 중복된 주소 및 MVS 이상 종료 CICS를 포함합니다.

MVS POST, WAIT EXTERNAL, WAITCICS, ENQ 또는 DEQ 명령을 사용하는 경우, 동적 트랜잭션 라우팅을 수행하는 기능에 악영향을 주는 트랜잭션간 연관 관계를 작성할 수 있습니다.

이러한 명령을 실행하는 프로그램의 잠재된 문제점을 쉽게 식별하기 위해 CICS Interdependency Analyzer를 사용할 수 있습니다. 이 기능에 대한 자세한 정보는 CICS Interdependency Analyzer for z/OS의 내용을 참조하고 트랜잭션 연관관계에 대한 자세한 정보는 379 페이지의 『연관관계』의 내용을 참조하십시오.

자원에 대한 액세스의 순서 제어

ENQ 및 DEQ 명령 대신 특정 순서로 두 개 이상의 태스크에 의해 자원에 액세스하려면, 하나 이상의 hand-post된 ECB와 함께 하나 이상의 WAITCICS 명령을 사용하십시오.

ECB를 hand-post하기 위해, CICS 태스크는 4바이트 필드를 2진 0의 지워진 상태나 X'40008000'의 게시된 상태로 설정합니다. 태스크는 START 명령을 사용하여 다른 태스크를 시작하거나 ECB의 주소를 전달할 수 있습니다. 시작된 태스크는 RETRIEVE 명령을 통해 주소를 수신합니다.

어느 태스크나 ECB를 설정하거나 이에 대해 대기할 수 있습니다. 태스크가 자원에 액세스하는 순서를 제어하려면 ECB를 사용하십시오. 필요한 경우 두 개의 태스크가 두 개 이상의 ECB를 공유할 수 있습니다. 원하는 만큼 태스크를 제어하기 위해 이 기술을 확장할 수 있습니다.

참고: 한 번에 단 하나의 태스크만 지정된 ECB에 대해 대기할 수 있습니다.

784 페이지의 그림 129의 예제는 두 개의 태스크가 hand-post된 ECB와 WAITCICS 명령을 사용하여 임시 저장영역 큐에 순차적으로 액세스할 수 있는 방법을 보여줍니다.

예제는 784 페이지의 그림 130에 있는 포인터에 의해 주소 지정된 두 개의 ECB(ECB1 및 ECB2)를 사용합니다.

이론적으로, 이 태스크는 영원히 임시 저장영역 큐를 통하여 데이터를 교환할 수 있습니다. 실제로, 일부 코드는 정돈된 방식으로 프로세스를 닫기 위해 포함됩니다.

Task A Delete temporary storage queue Clear ECB1 (set to X'00000000') Clear ECB2 EXEC CICS START task B (pass addresses of PTR_ECB1_ADDR_LIST and PTR_ECB2_ADDR_LIST) LOOP: EXEC CICS WAITCICS ECBLIST(PTR_ECB1_ADDR_LIST) NUMEVENTS(1) Clear ECB1 Read TS queue Process data Delete TS queue Write to TS queue Post ECB2 Go to start of loop	Task B EXEC CICS RETRIEVE (addresses passed) LOOP: Write to TS queue Post ECB1 (set to X'40008000') EXEC CICS WAITCICS ECBLIST(PTR_ECB2_ADDR_LIST) NUMEVENTS(1) ClearECB2 Read TS queue Process data Delete TS queue Go to start of loop

그림 129. 공유 자원에 대한 액세스를 제어하기 위해 WAITCICS를 사용하는 두 개의 태스크

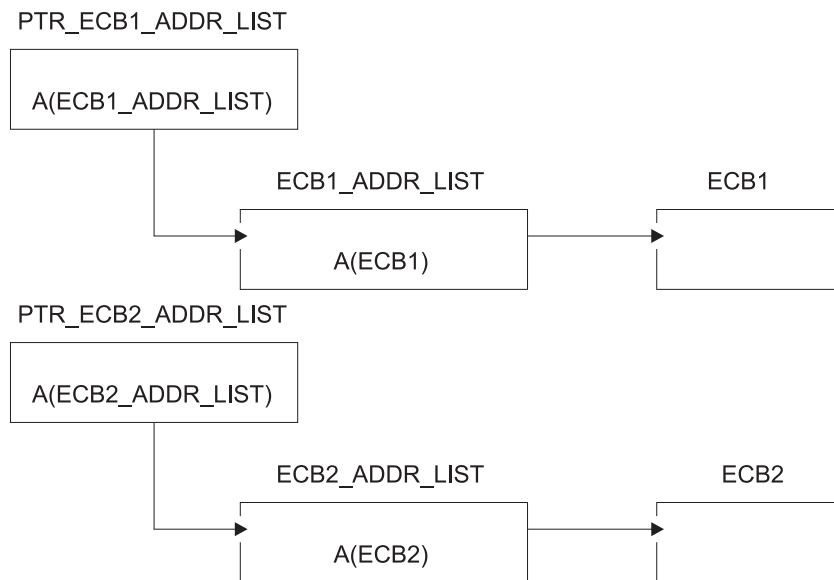


그림 130. WAITCICS 예제에서 사용되는 ECB 포인터

423 페이지의 『예외 조건 처리』에서는 태스크 제어 명령 처리 동안 발생할 수 있는 예외 상태가 처리되는 방법을 설명합니다.

제 10 부 저장영역 보호 및 트랜잭션 분리

CICS 코드, 제어 블록 및 기타 트랜잭션을 애플리케이션이 겹쳐쓰지 않도록 보호.

제 37 장 CICS 저장영역 보호 및 트랜잭션 분리

저장영역 제어는 저장영역 보호 및 트랜잭션 분리의 영향을 받습니다. 저장영역 보호는 애플리케이션에서 CICS 코드 및 제어 블록을 보호하고, 트랜잭션 분리는 서로 태스크를 보호합니다.

서브시스템 저장영역 보호 기능을 사용하면 애플리케이션이 실수로 CICS 코드와 제어 블록을 겹쳐쓰는 일이 없도록 할 수 있습니다. 의도적인 CICS 코드 또는 제어 블록 겹쳐쓰기에 대해서는 보호할 수 없습니다. CICS는 애플리케이션이 CICS 저장영역을 수정하기 위해 필요한 액세스(실행 키)를 확보하지 못하도록 할 수 없습니다.

트랜잭션 분리는 트랜잭션 데이터에 대한 보호를 제공하기 위해 이 저장영역 보호를 확장합니다. 애플리케이션이 실수로 다른 트랜잭션의 트랜잭션 데이터를 겹쳐쓰면, CICS 시스템의 신뢰성 및 가용성과, 시스템에서 데이터의 무결성에 영향을 미칠 수 있습니다.

저장영역 보호를 사용하는 것은 선택사항입니다. CICS 시스템 초기화 매개변수를 사용하여 저장영역 보호 기능의 사용 여부를 지정합니다. 관련 매개변수에 대한 정보는 성능 개선의 저장영역 보호의 내용을 참조하십시오.

저장영역 제어

CICS 저장영역 제어 기능은 중간 작업 영역을 제공하기 위한 기본 저장영역과 트랜잭션 처리에 필요한 다른 기본 저장영역에 대한 요청을 제어합니다.

Java 및 C++

여기에 설명된 API(Application Programming Interface)는 Java 프로그램에서 사용되지 않은 CICS API입니다. 저장영역 제어 서비스에 액세스하기 위해 JCICS 클래스를 사용하는 Java 프로그램에 대한 정보는 애플리케이션 개발에서 JCICS를 사용하는 Java 프로그래밍 및 JCICS Javadoc 문서를 참조하십시오. CICS C++ 클래스를 사용하는 C++ 프로그램에 대한 정보는 애플리케이션 개발에서 CICS C++ OO 기반 클래스 사용의 내용을 참조하십시오.

CICS 저장영역 제어 기능

CICS는 애플리케이션의 특정한 요청 없이, 자동으로 각 명령 레벨 프로그램 내에서 작업 중인 저장영역을 사용 가능하도록 만들고, 태스크 내에서도 태스크 사이에 중간 저장영역에 대한 다른 기능을 제공합니다. 303 페이지의 『성능 향상을 위한 디자인』에서는 개별 프로그램 내에서 저장영역에 대해 설명합니다. 그러나 CICS에서 자동으로 제공되는 작업 중인 저장영역 외에도 작업 중인 저장영역이 필요할 경우, 다음 명령을 사용할 수 있습니다.

- 기본 저장영역을 가져오고 초기화하기 위한 GETMAIN
- 기본 저장영역을 해제하기 위한 FREEMAIN

GETMAIN 명령에 INITIMG 옵션을 제공하여 비트 구성에 획득된 기본 저장영역을 초기화할 수 있습니다(예를 들어, 0 또는 EBCDIC 공백).

CICS는 태스크가 정상적으로 또는 비정상적으로 종료될 때 태스크와 연관된 모든 기본 저장영역을 해제합니다. 여기에는 획득되었지만, SHARED 옵션으로 확보된 영역의 경우를 제외하고 애플리케이션에 의해 그 뒤로 해제되지 않은 저장영역이 포함됩니다. GETMAIN 명령의 이 옵션은 태스크가 완료될 때 저장영역이 자동으로 해제되지 않도록 합니다.

SHARED 옵션과 함께 GETMAIN 명령을 사용하고 FREEMAIN 명령을 사용하는 경우, 동적 트랜잭션 라우팅을 수행하기 위한 능력에 반대의 영향을 미치는 트랜잭션 간 연관관계를 작성할 수 있습니다.

이러한 명령을 실행하는 프로그램의 잠재된 문제점을 쉽게 식별하기 위해 CICS Interdependency Analyzer를 사용할 수 있습니다. 이 유틸리티에 대한 정보는 CICS Interdependency Analyzer for z/OS의 내용을 참조하고, 트랜잭션 연관관계에 대한 정보는 379 페이지의 『연관관계』의 내용을 참조하십시오.

요청을 발행할 때 사용 가능한 저장영역이 없는 경우, NOSUSPEND 옵션을 지정하지 않는 한, 공백이 사용 가능할 때까지 CICS가 사용자 태스크를 일시중단합니다. 태스크가 일시중단되는 동안, 트랜잭션 정의에 SPURGE(YES) 및 DTIMOUT(mmss)가 지정되는 경우 태스크가 취소될 수 있습니다(제한시간 초과). 저장영역을 사용할 수 없는 경우 NOSUSPEND는 제어를 사용자 프로그램으로 리턴하여, 사용자가 대체 처리를 수행할 수 있도록 합니다(적절한 경우).

이 섹션에서는 다음에 대해 설명합니다.

- 787 페이지의 제 37 장 『CICS 저장영역 보호 및 트랜잭션 분리』
- 792 페이지의 『애플리케이션에 대한 저장영역 키 정의』
- 795 페이지의 『실행 및 저장 키 선택』
- 800 페이지의 『트랜잭션 분리로 보호』
- 802 페이지의 『MVS 서브공간』

저장영역 보호

CICS는 사용자 키 또는 CICS 키 저장영역에서 애플리케이션을 실행할 수 있도록 합니다. CICS 저장영역은 사용자 키 저장영역(기본값)에서 실행되는 애플리케이션에 의해 겹쳐쓰기 되지 않도록 자동으로 보호됩니다.

사용자 애플리케이션에서 CICS 코드와 제어 블록(CICS 내부 데이터 영역)을 분리하는 개념은 그림 131에 설명되어 있습니다.

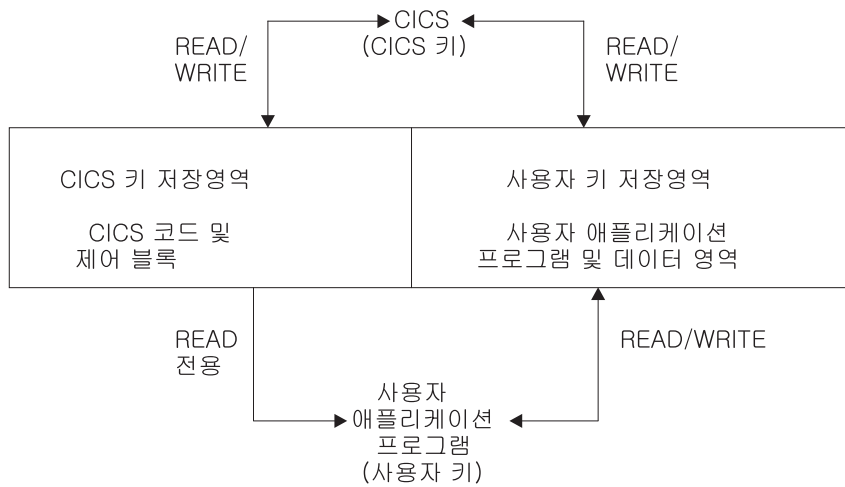


그림 131. 사용자 애플리케이션에서 CICS 코드 및 제어 블록 보호

관련 정보:

『저장영역 카테고리』

저장영역 보호 기능이 활성화된 상태에서 실행할 때, 애플리케이션을 실행할 수 있는 두 가지 유형의 저장영역이 있습니다. CICS 키 저장영역과 사용자 키 저장영역입니다.

➡ 참조 -> 시스템 정의에서 STGPROT 시스템 초기화 매개변수

저장영역 카테고리

저장영역 보호 기능이 활성화된 상태에서 실행할 때, 애플리케이션을 실행할 수 있는 두 가지 유형의 저장영역이 있습니다. CICS 키 저장영역과 사용자 키 저장영역입니다.

CICS 키 저장영역

대부분의 CICS 시스템 코드 및 제어 블록에 사용되며, 겹쳐쓰기 보호가 필요한 데이터 영역, 다른 코드 및 설치를 판별할 때 사용됩니다.

트랜잭션 분리가 활성 상태인 CICS 리전에서, CICS 키 프로그램은 CICS 키 및 사용자 키 저장영역에 대한 읽기/쓰기 액세스 권한을 가지고 있습니다.

사용자 키 저장영역

대부분의 애플리케이션과 해당되는 데이터 영역에 사용됩니다.

두 가지의 연관된 실행 모드가 있습니다.

1. 많은 CICS 시스템 프로그램이 **CICS 키**로 실행됩니다. CICS 키에서 실행 중인 프로그램에는 CICS 키 및 사용자 키 저장영역 둘 다에 대한 읽기-쓰기 액세스 권한을 가지고 있습니다.

2. 애플리케이션은 보통 사용자 키로 실행됩니다. 사용자 키로 실행 중인 프로그램은 사용자 키 저장영역에 대한 읽기-쓰기 액세스 권한을 갖지만, CICS 키 저장영역에 대해서는 읽기 액세스 권한을 갖습니다.

따라서 『사용자 키』 및 『CICS 키』 용어는 저장영역 카테고리나 프로그램 실행 둘 다에 적용됩니다. 이 용어는 TRANSACTION 정의에서 사용되는 자원 정의 키워드에 반영됩니다. 자세한 정보는 참조 -> 시스템 정의에서 FILE 속성 정의의 내용을 참조하십시오.

실행 키는 애플리케이션이 CICS 키 저장영역에 대해 가지고 있는 액세스 유형을 제어합니다. 기본값은 애플리케이션에 사용자 키의 제어가 제공되는 것입니다. CICS 키로 실행되는 것이 중요한 프로그램에 대해서만 CICS 키를 정의하십시오. CICS 키로 실행할 것을 선택할 수 있는 프로그램은 일반적으로 시스템 프로그래머가 작성하고, 사용자 애플리케이션 지원에서 특수 기능을 제공하도록 디자인된 프로그램입니다. 이와 같은 프로그램은 애플리케이션의 일부가 아니라 CICS의 확장인 것으로 간주됩니다. 이와 같은 프로그램의 일부 예제는 797 페이지의 『CICS 키 애플리케이션』에 설명되어 있습니다.

저장영역 보호 기능은 CICS 키로 실행할 것을 선택하는 프로그램에 의해 CICS 키 코드 및 제어 블록이 겹쳐쓰여지지 않도록 보호하지 않습니다.

실행 키 정의

CICS 키로 사용자 프로그램을 실행하려면, 프로그램 자원 정의에서 실행 키 매개변수 (EXECKEY)를 사용합니다. EXECKEY에 대한 설명은 795 페이지의 『실행 및 저장 키 선택』의 내용을 참조하십시오. EXECKEY 매개변수는 CICS가 제어를 애플리케이션으로 전달하는 키를 판별합니다.

트랜잭션 분리

트랜잭션 분리는 MVS 하위 공간 그룹 기능을 사용하여 트랜잭션 사이에 보호를 제공합니다. 이로서, 하나의 트랜잭션과 연관되는 애플리케이션이 실수로 다른 트랜잭션의 데이터 위에 겹쳐쓸 수 없습니다.

트랜잭션 분리의 이점과 해당되는 연관된 지원은 다음과 같습니다.

- 시스템 가동 중단 감소
- 애플리케이션 데이터 보호
- 올바른 주소로 전달하는 애플리케이션으로부터 CICS 보호
- 애플리케이션 개발 지원

시스템 가동 중단 감소

트랜잭션 분리는 사용자 키 트랜잭션의 저장영역이 실수로 겹쳐쓰게 되도록 하는 사용자 키 애플리케이션에서의 코딩 오류로 인한 데이터 손상 및 계획되지 않은 CICS 시

시스템 가동 중단을 방지합니다. 우연한 트랜잭션 데이터 겹쳐쓰기를 방지하면 CICS 리전의 가용성 및 신뢰성이 크게 개선됩니다.

애플리케이션 데이터 보호

애플리케이션이 CICS 코드 또는 데이터 위에 겹쳐쓰는 경우, CICS는 결과로 실패할 수 있습니다. 애플리케이션이 다른 애플리케이션의 코드 위에 겹쳐쓰는 경우, 다른 애플리케이션은 실패하게 됩니다. 이는 프로덕션 리전에서 심각한 인터럽트이지만, 효과가 즉각적이고 터미널 사용자가 실패한 트랜잭션을 재시도할 수 있도록 프로그램이 일반적으로 복구될 수 있습니다. 그러나 한 트랜잭션의 애플리케이션이 다른 트랜잭션의 데이터 위에 겹쳐쓰면, 결과는 종종 바로 분명하게 나타나지 않습니다. 오류 데이터가 데이터베이스에 기록될 수 있고, 오류 원인을 판별할 수 없게 되는 나중까지 오류가 발견되지 않고 유지될 수 있습니다. 데이터 겹쳐쓰기의 결과는 종종 코드 겹쳐쓰기보다 한층 더 심각합니다.

올바르지 않은 주소 전달로부터 CICS 보호

CICS에서는 다른 위치의 자체 코드를 CICS 처리에 삽입하기 위해 XPI라고 하는 프로그래밍 인터페이스 및 사용자 엑시트를 제공합니다. 이는 CICS가 프로그램 대신 저장영역을 수정하도록 하는 EXEC CICS 명령을 애플리케이션이 실행하지만 프로그램이 저장영역을 소유하지 않는 경우에 발생합니다. 이전 릴리스에서는, CICS가 전달된 주소에 의해 참조되는 저장영역의 소유권을 확인하지 않고, 이와 같은 명령을 실행하여서 결과적으로 저장영역 위반이 발생했습니다.

CICS는 저장영역의 시작 주소 유효성을 검증하고, 명령 실행 전에 해당 주소로 시작하는 저장영역에 대해 애플리케이션이 쓰기 액세스 권한을 가지도록 설정합니다.

이 주소 확인은 CMDPROT 시스템 초기화 매개변수를 사용하여 제어됩니다. 프로그램이 API에서 출력 필드로서 올바르게 않은 주소를 CICS로 전달하는 경우 AEYD 이상 종료가 발생합니다. 이는 저장영역 보호 및 트랜잭션 분리의 영향을 전혀 받지 않습니다.

애플리케이션 개발 지원

트랜잭션 분리는 테스트 및 디버깅 단계에서 애플리케이션 개발을 지원합니다. 애플리케이션이 CICS 또는 다른 애플리케이션을 겹쳐쓰려고 하거나, CICS가 쓰도록 애플리케이션이 소유하고 있지 않은 저장영역 주소를 전달하려는 경우, CICS는 태스크를 즉시 이상 종료하고 을 소유하지 않는 저장영역 주소를 통과하려고 하면, CICS는 바로 태스크를 이상 종료하고, 겹쳐쓰려고 한 위험할 수 있는 프로그램의 이름 및 영역을 보고합니다. 애플리케이션 개발 환경에서 공통 문제점을 디버깅하면 많은 시간이 절약됩니다.

애플리케이션에 대한 저장영역 키 정의

애플리케이션이 사용할 수 있는 많은 CICS 데이터 영역 및 애플리케이션 데이터 영역에 대해 사용자 키 저장영역과 CICS 키 저장영역 사이에서 선택할 수 있습니다.

이 태스크 정보

데이터 영역에 따라서 다음 중 하나를 사용하여 저장영역 키를 선택합니다.

- 시스템 초기화 매개변수
- 자원 정의 옵션
- GETMAIN 명령의 옵션

애플리케이션이 액세스해야 하는 저장영역에 대한 저장영역 키 정의는 다음 섹션에서 설명됩니다.

시스템 전체 저장영역

각 CICS 리전에 대해 설치되는 공통 작업 영역(CWA) 및 터미널 제어 테이블 사용자 영역(TCTUA)에 대해 사용자 키와 CICS 키 저장영역 중에서 선택할 수 있습니다.

이러한 영역이 사용자 키 저장영역에 있으면 모든 프로그램에는 이에 대한 읽기-쓰기 액세스가 있습니다. 이러한 영역이 CICS 키 저장영역에 있으면, 사용자 키 애플리케이션은 읽기 전용 액세스로 제한됩니다. CWA 및 TCTUA의 저장영역 키는 각각 시스템 초기화 매개변수 CWAKEY 및 TCTUAKEY에 의해 설정됩니다. 두 경우 모두 기본 옵션은 CICS가 사용자 키 저장영역을 획득하는 것입니다.

관련 참조

 참조 -> 시스템 정의에서 CWAKEY 시스템 초기화 매개변수

 참조 -> 시스템 정의에서 TCTUAKEY 시스템 초기화 매개변수

태스크 수명 저장영역

CICS가 트랜잭션 접속 시에 획득하는 저장영역에 또는 트랜잭션에 있는 개별 애플리케이션에 직접 관련된 저장영역의 요소에 사용자 키 또는 CICS 키 저장영역이 사용되는지 여부를 지정할 수 있습니다.

사용자 키 또는 CICS 키 저장영역이 사용되는지 여부를 지정하려면 트랜잭션 자원 정의에서 TASKDATAKEY 옵션을 사용하십시오. 이 옵션은 다음 저장영역에 할당된 저장영역 유형을 관리합니다.

- 트랜잭션 작업 영역(TWA) 및 EXEC 인터페이스 블록(EIB)
- CICS가 애플리케이션의 각 실행에 대해 획득한 작업 중인 저장영역의 사본
- 다음 저장영역 요청에 대한 응답으로 애플리케이션에서 획득한 모든 저장영역:

- GETMAIN 명령의 결과로서 명시적인 요청
- SET 옵션을 사용하는 CICS 명령의 결과로서 암시적 요청

TASKDATAKEY 매개변수를 지정하는 방법에 대한 정보는 참조 -> 시스템 정의에서 TRANSACTION 속성의 내용을 참조하십시오.

태스크 수명 저장영역과 프로그램 작업 중인 저장영역에 대해 TASKDATAKEY가 제어하는 사항에 대한 설명은 794 페이지의 그림 132의 내용을 참조하십시오.

EXEC CICS 명령에 대한 프로그래밍 정보는 참조 -> 애플리케이션 개발에서 CICS 명령 요약의 내용을 참조하십시오.

특히 엑시트와 PLT 프로그램을 위한 프로그램 작업 중인 저장영역

CICS는 글로벌 사용자 엑시트, 태스크 관련 사용자 엑시트, 사용자 대체가능 모듈과 PLT 프로그램을 위해 획득된 저장영역을 위한 기억 키를 판별하기 위해 호출 트랜잭션의 TASKDATAKEY 옵션을 사용합니다.

저장영역 키가 여러 유형의 프로그램에 미치는 영향을 포함하여 이에 대한 프로그래밍 정보는 CICS 사용자 정의 안내서를 참조하십시오.

COMMAREA를 사용하여 데이터 전달

의사 대화식 애플리케이션에서 CICS는 RETURN 명령에 지정하는 COMMAREA가 항상 읽기-쓰기 모드에서 대화의 다음 프로그램에 액세스 가능한지 확인하십시오.

둘 이상의 프로그램을 구성하는 트랜잭션 내에서 COMMAREA를 전달할 때에도 이와 마찬가지로입니다(LINK 또는 XCTL 명령 사용). CICS는 대상 프로그램에 COMMAREA에 대한 읽기-쓰기 액세스가 있는지 확인합니다.

GETMAIN 명령

GETMAIN 및 GETMAIN64 명령은 연관된 트랜잭션 자원 정의에 지정된 TASKDATAKEY 옵션에 상관없이, 애플리케이션이 사용자 키 또는 CICS 키 저장영역을 명시적으로 요청할 수 있도록 USERDATAKEY 및 CICS DATAKEY 옵션을 제공합니다.

예를 들어, 사용자 키에서 실행하고 있는 프로그램으로 전달하거나 리턴하기 위해 사용자 키 저장영역을 획득하기 위해 TASKDATAKEY(CICS)를 지정하여 실행하고 있는 애플리케이션에 대해 이러한 옵션을 사용할 수 있습니다.

CICS 키 저장영역을 획득하기 위해 GETMAIN 명령을 실행하는 EXEC KEY(CICS)가 정의된 프로그램에서, FREEMAIN 또는 FREEMAIN64 명령이 EXEC KEY(CICS)로 정의된 프로그램에 의해 실행되는 경우에만 이와 같은 저장영역은 명시적으로 사용 가능하도록 해제할 수 있습니다. EXEC KEY(USER)로 정의된 애플리케이션이

FREEMAIN 명령을 사용하여 CICS 키 저장영역을 사용 가능하도록 해제하려는 경우, CICS는 INVREQ 상태를 리턴합니다. 그러나 애플리케이션은 EXECKEY 옵션에 상관없이 FREEMAIN 명령으로 사용자 키 저장영역을 사용 가능하도록 해제할 수 있습니다.

CICS는 CICS 키 또는 사용자 키 사용 여부에 관계없이 태스크 종료 시 애플리케이션에 의해 획득되는 모든 태스크 수명 저장영역을 사용 가능하도록 해제합니다. 또한 연관된 트랜잭션 자원 정의의 이 옵션에서 STORAGECLEAR(YES)를 지정할 수도 있습니다. 그러면 다른 태스크가 민감한 데이터를 볼 수 없도록 저장영역을 지웁니다.

명령에 대한 프로그래밍 정보는 참조 -> 애플리케이션 개발에서 CICS API 명령의 내용을 참조하십시오. 자원 정의에 대한 정보는 제품 개요의 자원 정의 개요의 내용을 참조하십시오.

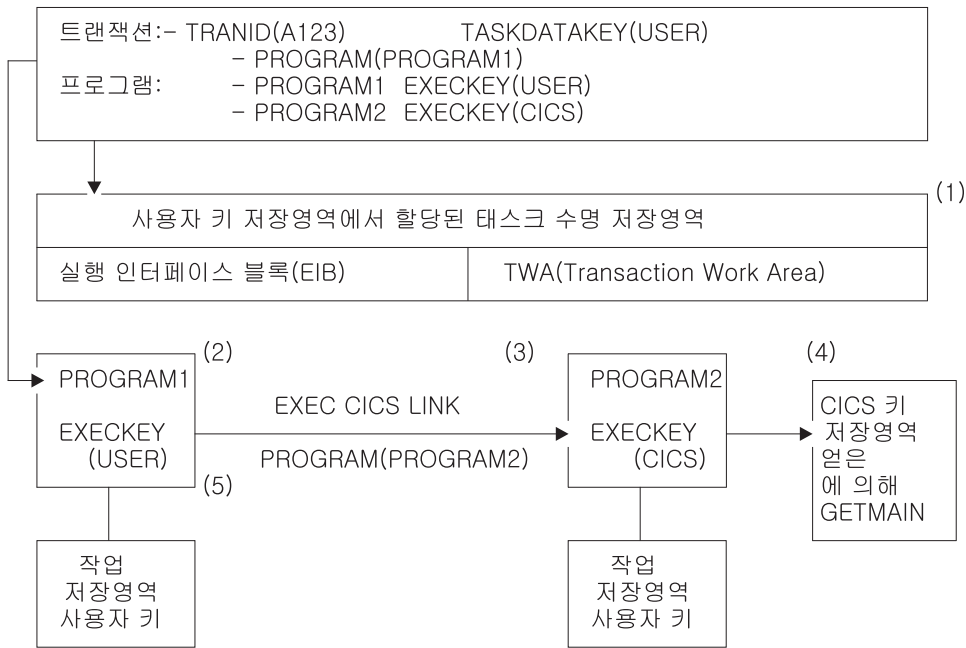


그림 132. TASKDATAKEY 및 EXECKEY 옵션 사용의 그림

이 예제에서, 트랜잭션 A123은 TASKDATAKEY(USER) 및 PROGRAM(PROGRAM1)으로 정의됩니다. PROGRAM1은 EXECKEY(USER)로 정의되고 PROGRAM2는 EXECKEY(CICS)로 정의됩니다. PROGRAM1은 PROGRAM2에 링크하고, PROGRAM2는 GETMAIN 요청을 사용하여 CICS 키 저장영역을 확보합니다.

참고:

1. TASKDATAKEY 옵션은 TWA(TRANSACTION WORK AREA) 및 실행 인터페이스 블록(EIB)이 사용자 키 저장영역에서 할당되는지 확인합니다. 사용자 키로 실행되는(EXECKEY(USER)로 지정) PROGRAM1에 대해 필요합니다.

2. PROGRAM1은 사용자 키로 실행되고(EXECKEY로 제어), 사용자 키 저장영역에서 확보되는 작업 중인 저장영역을 가지고 있습니다(TASKDATAKEY 옵션으로 제어). GETMAIN 명령을 사용하거나, CICS 명령에서 SET 옵션을 사용하여 프로그램이 확보하는 다른 저장영역은 사용자 키 저장영역에도 확보됩니다.
3. PROGRAM2는 CICS키로 실행되지만(EXECKEY로 제어), TASKDATAKEY 옵션으로 제어되는 사용자 키 저장영역에 확보된 해당되는 작업 중인 저장영역을 수반합니다.
4. PROGRAM2는 CICS DATAKEY 옵션을 사용하여 명시적 GETMAIN 명령을 실행하고, CICS 키로 실행하므로 PROGRAM1으로 제어를 리턴하기 전에 CICS 키 보호 저장영역에 데이터를 저장할 수 있습니다.
5. PROGRAM1은 PROGRAM2가 획득한 CICS 키 보호 저장영역에 쓸 수 없지만, PROGRAM2가 쓴 것을 읽을 수 있습니다.

EXECKEY(CICS) 및 TASKDATAKEY(CICS)를 지정해야 하는지 여부를 결정할 때, 이러한 옵션을 필요하게 만드는 모든 응답을 고려해야 합니다.

해당되는 저장영역 보호 키를 수정하는 프로그램은 저장영역에 액세스하려고 할 때 올바른 키로 실행 중인지 확인해야 합니다. CICS는 프로그램을 호출할 때 프로그램 정의에서 정의된 EXECKEY만 사용할 수 있습니다.

실행 및 저장 키 선택

저장영역 보호를 사용하여 CICS를 실행 중인 경우, 대부분의 애플리케이션은 사용자 키로 실행해야 하며, 해당되는 모든 저장영역은 사용자 키로 확보됩니다. 단지 사용자 키에서 허용되지 않는 기능을 사용하는 해당 프로그램에 대해, 또는 특수한 『시스템 유형』 트랜잭션 또는 벤더 패키지에 대해 연관된 트랜잭션 정의에 TASKDATAKEY(CICS)를, 프로그램 정의에 EXECKEY(CICS)를 정의해야 합니다.

이 태스크 정보

모든 구성요소 프로그램이 EXECKEY(CICS)가 있는 해당 트랜잭션에 대해, 그리고 해당되는 태스크 수명 및 작업 중인 저장영역이 사용자 키 애플리케이션에 의해 겹쳐지지 않도록 보호하려고 하는 트랜잭션에 대해 TASKDATAKEY(CICS)만 지정해야 합니다. 예를 들어, CICS 제공 트랜잭션(예: CEDF)은 TASKDATAKEY(CICS)로 정의됩니다.

TASKDATAKEY(CICS)로 정의된 트랜잭션의 일부를 형성하는 프로그램에 EXECKEY(USER)를 지정할 수 없습니다. 이 상황에서 사용자 키 프로그램은 해당되는 고유한 작업 중인 저장영역에 쓰지 못할 수도 있기 때문입니다. 저장영역 보호의 활

성화 여부에 관계없이, TASKDATAKEY(CICS)로 정의된 트랜잭션 내에서 프로그램이 EXECKEY(USER)로 정의된 경우 트랜잭션은 AEZD 이상 종료로 이상 종료됩니다.

해당되는 호출자의 실행 키를 상속하도록 프로그램을 정의할 수 없습니다. 실행 키 및 데이터 저장 키는 각각 해당되는 프로그램과 연관 트랜잭션 자원 정의에서 각 프로그램에 대해 파생되며, 사용자가 명시적으로 지정하거나 기본값을 허용할 수 있습니다. 기본값은 항상 사용자 키입니다. 표 42에서는 옵션의 다양한 조합을 요약합니다.

표 42. KEY 옵션의 조합

EXECKEY	TASKDATAKEY	권장되는 사용법 및 설명
USER	USER	CICS API를 사용하는 일반 애플리케이션의 경우.
USER	CICS	허용되지 않음. CICS는 TASKDATAKEY(CICS)로 정의된 트랜잭션 아래에서 호출된 EXECKEY(USER)로 정의된 프로그램을 이상 종료합니다.
CICS	USER	제한된 MVS 요청을 실행하거나 CICS 키 저장영역을 수정해야 하는 프로그램의 경우.
CICS	CICS	CICS에 대한 확장기능으로 작동하거나, 동일한 보호가 요구되는 트랜잭션(및 구성요소 프로그램)의 경우.

사용자 키 애플리케이션

대부분의 애플리케이션에 대해, EXECKEY(USER)로 프로그램을 정의하고 TASKDATAKEY(USER)로 관련 트랜잭션을 정의해야 합니다.

CICS 저장영역 보호 기능에서 최대의 이점을 얻으려면, 사용자 키 저장영역에서 애플리케이션을 실행하도록 합니다. 옵션에서 USER를 지정하면 다음과 같은 효과가 있습니다.

EXECKEY(USER)

CICS가 호출 시 사용자 키의 프로그램에 제어를 부여하도록 지정합니다. EXECKEY(USER)로 정의된 프로그램은 CICS 키 저장영역에 대한 읽기 전용 액세스로 제한됩니다. 여기에는 다음이 포함됩니다.

- CICS 자체에 속하는 저장영역
- TASKDATAKEY(CICS)로 정의된 사용자 트랜잭션에 속하는 CICS 키 저장영역
- EXECKEY(CICS)로 정의되어 CICS 키 저장영역에 로드되는 애플리케이션
- 트랜잭션 분리가 활성화된 CICS 리전에서, 사용자 키 프로그램은 자체의 고유 트랜잭션과 공유된 DSA 저장영역의 사용자 키 태스크 수명 저장영역에 대해 읽기/쓰기 액세스 권한을 갖습니다.

TASKDATAKEY(USER)

TWA(TRANSACTION WORK AREA) 및 실행 인터페이스 블록(EIB)과 같은 모든 태스크 수명 저장영역은 사용자 키 저장영역에서 확보됨을 지정합니다.

또한 트랜잭션 내의 프로그램에 직접 관련되는 모든 저장영역은 사용자 키 저장영역에서 확보됨을 의미합니다.

그러나 ISOLATE(YES)로 정의된 트랜잭션의 사용자 키 프로그램은 단지 해당되는 자체의 태스크에 대한 사용자 키 태스크 수명 저장영역에 대한 액세스만 갖습니다.

USER는 EXECKEY 및 TASKDATAKEY 옵션 둘 다에 대해 기본값이므로, 기존 애플리케이션에 대한 자원 정의에 대해 변경사항을 수행하지 않아도 됩니다.

CICS 키 애플리케이션

일부 애플리케이션은 나중에 나열되는 특정 기능을 사용해야 하기 때문에 EXECKEY(CICS)로 정의되어야 합니다.

대부분의 애플리케이션은 EXECKEY(USER)로 정의할 수 있으며 이는 기본값으로 대부분의 경우에서 사용되도록 권장되는 옵션입니다. 여기에는 DL/I 또는 DB2를 사용하는 프로그램과 자원 관리 인터페이스(RMI) 또는 LINK 명령으로 벤더 제품에 액세스하는 제품이 포함됩니다.

EXECKEY(CICS)를 광범위하게 사용하면 CICS 코드 및 CICS 키에서 실행 중인 애플리케이션으로 겹쳐쓰기 중인 제어 블록 보호가 없기 때문에 저장영역 보호 기능에서 제공되는 보호가 약화됩니다. 트랜잭션 정의의 ISOLATE 속성은 CICS 키(즉, EXECKEY(CICS)로 정의된 프로그램)에서 실행되는 애플리케이션에 대한 보호를 제공하지 않습니다. EXECKEY(USER)로 정의될 때 보호 예외를 발생시키는 모든 애플리케이션은 수정이 허락되지 않는 저장영역을 수정하려고 시도하는 이유를 판별하기 위해 평가해야 합니다. 사용자가 애플리케이션이 적절하게 여기에서 설명하는 기능을 사용한다고 판단되는 경우에만 프로그램 정의를 EXECKEY(CICS)로 변경해야 합니다.

- 프로그램은 CICS API를 통하지 않고 MVS 매크로 또는 서비스를 직접 사용합니다. 사용자 키 프로그램에서 지원되는 유일한 MVS 매크로는 SPIE, ESPIE, POST, WAIT, WTO, WTOR입니다. EXECKEY(USER) 프로그램에서 GTF 추적 요청을 발행할 수도 있습니다. 프로그램이 다른 MVS 매크로 또는 서비스를 사용하는 경우는 EXECKEY(CICS)로 정의해야 합니다. 일부 특정 예제는 다음과 같습니다.

- 동적 할당 사용(DYNALLOC 매크로, SVC 99)
- MVS GETMAIN 및 FREEMAIN 또는 STORAGE 요청 사용
- MVS OPEN, CLOSE 또는 다른 파일 액세스 요청 사용

일부 MVS 매크로 및 서비스를 직접 사용하는 것은 EXECKEY(CICS)로 정의된 CICS 애플리케이션에서 조차 바람직하지 않습니다. 이는 요청이 충족될 때까지 MVS가 전체 CICS 리전을 일시중단할 수 있기 때문입니다.

일부 COBOL, PL/I, C, C++ 언어 명령문 및 컴파일러 옵션으로 인해 운영 체제 기능이 호출됩니다. CICS 애플리케이션에서 사용하면 안되는 사항에 대한 정보는 25 페이지의 제 3 장 『COBOL 애플리케이션 개발』, 53 페이지의 제 4 장 『C 및 C++로 프로그래밍』, 65 페이지의 제 5 장 『PL/I 애플리케이션 개발』의 내용을 참조하십시오. 이 기능 중 일부는 이전 CICS 릴리스에서 작동하거나 최소한 애플리케이션이 실패되도록 하지는 않았을 수 있습니다. 프로그램이 EXECKEY(USER)로 정의되는 해당 기능은 작동하지 않습니다. 금지된 옵션 또는 명령문을 사용하여 보호 예외가 발생하는 경우 프로그램을 EXECKEY(CICS)로 다시 정의하는 것보다는 프로그램에서 이를 제거해야 합니다. 금지된 명령문 및 옵션을 사용하면 CICS의 전체 실행에 부작용을 초래할 수 있기 때문에 제거해야 합니다.

- 프로그램은 CWA를 수정해야 하며 CWA는 CICS 키 저장영역(CWAKEY=CICS)에 있습니다.

CWAKEY(CICS)를 지정하여 CWA를 보호하려면 CWA 수정이 허용되는 프로그램을 가능한 적은 수로 제한하고 한 개만 사용하는 것이 가장 좋습니다. 보호된 CWA에 대한 액세스 제어 방법에 대한 정보는 315 페이지의 『CWA(Common Work Area) 사용』의 내용을 참조하십시오.

- 프로그램은 TCTUA를 수정해야 하며 TCTUA는 CICS 키 저장영역(TCTUAKEY=CICS)에 있습니다.

저장영역 보호 환경에서 TCTUA 사용에 대한 정보는 319 페이지의 『TCTTE 사용자 영역(TCTUA) 사용』의 내용을 참조하십시오.

- 프로그램은 PLT 프로그램, TASKDATAKEY(CICS)로 정의된 트랜잭션, 태스크 관련 또는 글로벌 사용자 엑시트 프로그램 또는 사용자 대체 가능 프로그램에서 호출할 수 있습니다.
- 프로그램은 CICS 제어 블록을 수정합니다. 예를 들어, CICS 제어 블록을 조작해야 하는 일부 벤더 제품입니다. 이 제품은 EXECKEY(CICS)로 정의해야 합니다.
- 프로그램은 CICS에 대한 사용자 확장을 제공하고 보호 및 CICS 시스템 코드와 같은 데이터 액세스가 필요합니다. 예를 들어, 이런 프로그램은 CICS 설치에서 중요한 파트이며 연관된 저장영역(예: CICS 저장영역)은 일반 애플리케이션에서 보호되어야 하는 것으로 고려할 수 있습니다.
- CICS는 항상 프로그램 자원 정의에 지정된 옵션에 상관없이 다음 유형의 사용자 작성 프로그램에 대해 CICS 키에 제어를 제공합니다.
 - 글로벌 사용자 엑시트(GLUEs)
 - 태스크 관련 사용자 엑시트(TRUEs)

- 사용자 대체 가능 모듈(URMs)
- 프로그램 목록 테이블(PLT) 프로그램
- 사용자 정의 EP 어댑터 처리 동기 이벤트

CICS는 제어가 PLT 프로그램, 글로벌 또는 태스크 관련 사용자 엑시트 또는 사용자 대체 가능 프로그램으로 전달될 때 해당 프로그램 자원 정의에 지정된 EXECKEY에 상관없이 호출된 첫 번째 프로그램이 CICS 키에서 실행되도록 합니다. 그렇지만 이 첫 번째 프로그램이 다른 프로그램에 LINK 또는 XCTL되는 경우, 이 프로그램은 해당 프로그램 정의에 지정된 키에서 실행됩니다. 이 후속 프로그램이 이런 유형의 상황에서 자주 발생하기 때문에 CICS 키 데이터 영역에 써야 하는 경우, EXECKEY(CICS)로 정의해야 합니다.

트랜잭션 분리가 활성화된 CICS 리전에서, 이 TRUE 및 GLUE는 CICS가 엑시트 프로그램에 제어를 제공하는 현재 모드에 따라 기본 공간이나 서브공간(802 페이지의 『MVS 서브공간』 참조)에서 실행됩니다. 또한, 모든 애플리케이션 저장영역을 수정할 수 있습니다. URM 및 PLT 프로그램은 기본 공간에서 실행됩니다.

저장영역 보호로 실행 중인 CICS 리전에서 GLUE, TRUE, URM, PLT 프로그램 실행에 대한 프로그래밍 정보는 CICS 사용자 정의 안내서에서 엑시트 프로그램 및 CICS 저장영역 보호 기능을 참조하십시오.

두 개의 트랜잭션이 태스크 수명 저장영역을 공유하는 연관관계를 포함하는 경우 트랜잭션은 ISOLATE(NO)로 정의하거나 프로그램은 EXECKEY(CICS)으로 정의되어야 합니다. CICS Interdependency Analyzer를 사용하여 트랜잭션 연관관계 원인을 확인할 수 있습니다. 이 유틸리티에 대한 자세한 내용은 CICS Interdependency Analyzer for z/OS의 내용을 참조하십시오. CICS 시스템 코드 및 데이터가 여전히 보호되고 있기 때문에 이 옵션 중 첫 번째가 권장되는 옵션입니다.

테이블

실행 프로그램뿐만 아니라 테이블 매핑 세트, 파티션 세트를 프로그램 자원으로 정의할 수 있습니다. EXECKEY는 해당 오브젝트와 연관성이 적는데 그 이유는 해당 오브젝트가 실행되지 않기 때문입니다. 그렇지만 EXECKEY는 비실행 오브젝트가 로드될 때 이를 제어하고 결국 다른 프로그램이 해당 오브젝트에 저장되는지 여부에 영향을 줍니다.

매핑 세트 및 파티션 세트

매핑 세트는 재진입 가능하지 않습니다. (BMS 자체가 절대 화면 위치를 계산할 때 매핑에서 필드를 갱신합니다.) 그렇지만 매핑 세트는 애플리케이션으로 수정하면 안됩니다. CICS로만 수정해야 하며 항상 CICS 키에서 실행됩니다. CICS는 항상 매핑 세트와 파티션 세트를 CICS 키 저장영역에 로드합니다.

저장영역 보호 예외 조건

사용자 키에서 실행 중인 애플리케이션이 CICS 키 저장영역 수정을 시도하면 보호 예외가 발생합니다. 보호 예외는 정상 CICS 프로그램 오류 처리로 처리되며 위반 트랜잭션은 ASRA, 이상 종료로 이상 종료됩니다. 예외 조건은 다른 보호 저장영역을 참조하려고 시도한 것처럼 트랜잭션에 표시됩니다. CICS 오류 처리는 참조가 CICS 키 동적 저장영역(DSA)에 대한 것인지를 확인하고 메시지(DFHSR0622)를 콘솔로 전송합니다. 그렇지 않으면 CICS는 다른 모든 ASRA, 이상 종료와 다르게 실패를 처리하지 않습니다. 저장영역 보호 예외 조건에 대한 자세한 정보는 *CICS 문제점 판별 안내서*의 트랜잭션 이상 종료 코드 처리를 참조하십시오.

트랜잭션 분리로 보호

사용자 트랜잭션을 위해 저장영역과 실행 키를 지정할 수 있는 데 추가로 트랜잭션 분리를 원하는지 여부를 지정할 수도 있습니다. 트랜잭션 분리는 저장영역 보호의 위에 빌드되고 이는 STGPROT=YES가 지정되어야 함을 의미합니다. 트랜잭션 분리는 저장영역 보호에 의해 소개된 매개변수를 사용하고 이들은 EXECKEY 및 TASKDATAKEY입니다.

TRANISO 시스템 초기화 매개변수를 사용하여 전체 CICS 리전에 대해 트랜잭션 분리를 전체적으로 제어할 수 있습니다. 개별 트랜잭션의 경우 트랜잭션 자원 정의의 ISOLATE 옵션을 사용하면 각 트랜잭션 및 프로그램에 적용해야 하는 보호 레벨을 지정할 수 있습니다.

ISOLATE [YES or NO]

이러한 옵션의 기본값은 대부분의 경우 기존 애플리케이션에는 자원 정의에 대한 변경 사항이 필요하지 않음을 의미합니다. 그러나 필요한 경우 트랜잭션이 기본값인 전체 보호 기준을 충족하는 데 실패하는 경우 계속해서 작동할 수 있도록 보호를 조정할 수 있습니다.

트랜잭션 A(TXNA)에 의해 호출되는 사용자 키 프로그램은 TXNA의 사용자 키 태스크 수정 저장영역 및 공유 사용자 저장영역을 읽고 쓸 수 있습니다. 게다가, TXNA에는 트랜잭션 B(TXNB)의 사용자 키 태스크 수정 저장영역에 대한 액세스가 없습니다.

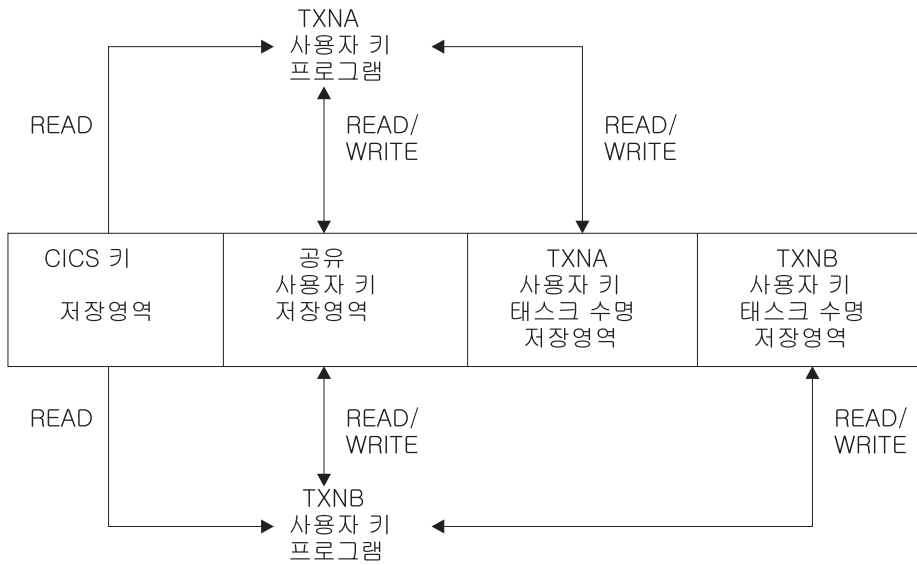


그림 133. ISOLATE(YES)로 정의된 두 개의 트랜잭션

트랜잭션이 ISOLATE(NO)로 정의된 경우에는 해당 사용자 키 태스크 수명은 ISOLATE(NO)로도 정의된 다른 모든 트랜잭션에 표시 가능합니다. 그러나 이는 ISOLATE(YES)로 정의된 트랜잭션으로부터 보호됩니다.

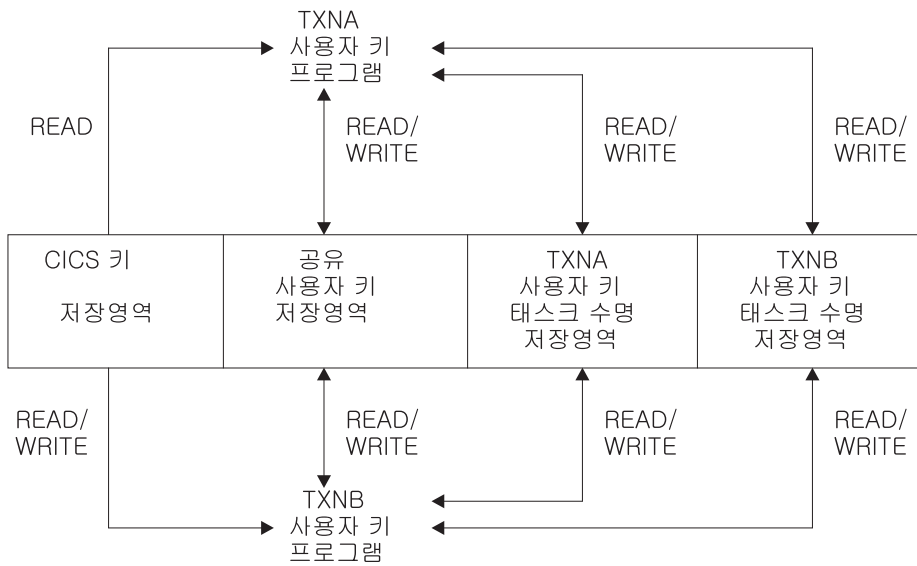


그림 134. 서로의 태스크 수명 저장영역에 대한 읽기/쓰기 액세스가 있는 ISOLATE(NO)로 정의된 두 개의 트랜잭션

사용자 키 저장영역은 ISOLATE(YES)를 설정하더라도 CICS 키 프로그램으로부터 보호되지 않습니다.

MVS 서브공간

MVS에서 저장영역 분리를 위해 서브공간 그룹 기능을 사용하여 주소 공간내에서의 데이터 무결성을 보존할 수 있습니다. 서브공간 그룹 기능은 하드웨어를 사용하여 트랜잭션 데이터에 대한 보호를 제공합니다. 서브공간 그룹은 서브공간의 그룹이며 기본 공간이 일반 MVS 주소 공간인 한 개의 기본 공간입니다.

서브공간 그룹 기능은 기반이 되는 기본 공간의 부분 매핑을 제공하기 때문에 기본 공간에서 저장영역의 지정된 영역만 특정 서브공간에 노출됩니다. 각 서브공간은 기본 공간에서 저장영역에 대한 다른 서브셋을 표시합니다. 트랜잭션 분리가 지정되는 경우, EXECKEY(USER)로 정의된 프로그램은 자체 서브공간에서 실행되고 모든 공유 저장영역 또는 CICS 저장영역에 대해 적절한 액세스를 사용해야 합니다. 따라서 사용자 트랜잭션은 주소 공간에 대한 자체 『보기』로 제한됩니다.

EXECKEY(CICS)로 정의된 프로그램은 기본 공간에서 실행됩니다.

트랜잭션을 위한 서브공간 및 기본 공간

일반적으로 트랜잭션 분리는 사용자 키 프로그램이 별도(고유)의 서브공간에 할당되도록 하며 다음을 포함합니다.

- 사용자 동적 저장영역(UDSA 또는 EUDSA) 중 하나에서 할당되는 사용자 자체 태스크의 사용자 키 태스크 수명 저장영역에 대한 읽기 및 쓰기 액세스
- 공유 저장영역에 대한 읽기 및 쓰기 액세스. 이는 GETMAIN 명령에 SHARED 옵션을 사용하여 확보하는 저장영역(SDSA 또는 ESDSA).
- 다른 태스크의 CICS 키 태스크 수명 저장영역에 대한 읽기 액세스(CDSA 또는 ECDSA)
- CICS 코드에 대한 읽기 액세스
- CICS API로 액세스 가능한 CICS 제어 블록에 대한 읽기 액세스

다른 태스크의 사용자 키 태스크 수명 저장영역에 대해서는 어떤 액세스도 가능하지 않습니다.

새 트랜잭션 자원 정의 속성에 대한 기본값은 기존 애플리케이션이 트랜잭션 분리로 운영되도록 지정합니다. (ISOLATE 옵션에 대한 기본값은 YES입니다.) 기존 애플리케이션은 트랜잭션 분리 요구사항을 준수하는 경우 수정되지 않고 실행됩니다.

그렇지만 애플리케이션 중 소수는 다음 중 하나라도 수행하는 경우 특수한 정의가 필요할 수도 있습니다.

- MVS 매크로 직접 실행
- CICS 제어 블록 수정

- 다른 태스크 저장영역에 액세스하거나 이를 공유하는 한 태스크에 대한 합법적인 요구를 포함

일부 기존 트랜잭션은 다양한 방식으로 태스크 수명 저장영역을 공유하고 이 공유로 인해 서로 분리되어 실행되지 못할 수도 있습니다. 이런 트랜잭션이 계속 실행되도록 이런 모든 트랜잭션이 실행 가능한 한 개의 공통 서브공간이 제공됩니다. 그러면 트랜잭션은 고유의 서브공간에서 실행 중인 시스템의 다른 트랜잭션에서 분리되지만 공통 서브공간에서 서로의 데이터를 공유할 수 있습니다.

CICS 키 프로그램은 기본 공간에서 실행되기 때문에 모든 CICS 키 저장영역 및 사용자 키 저장영역에 대해 읽기/쓰기 액세스를 가집니다.

공통 서브공간 및 공유 저장영역

애플리케이션이 유효한 방식으로 서로의 저장영역에 액세스하는 몇 개의 트랜잭션이 있을 수 있습니다. 나중에 MVS POST에 의해 또는 다른 태스크에 의해 『지표』로 게시되는 하나 이상의 이벤트 제어 블록(ECB)을 태스크가 대기하는 경우가 이런 경우입니다.

예를 들어, 태스크는 자체 저장영역 일부에 대한 주소를 다른 태스크로 전달하고(임시 저장영역 큐 또는 일부 다른 방법을 통해) 다른 태스크가 ECB를 게시하여 갱신된 저장영역이 있음을 알리도록 대기할 수 있습니다. 확실한 것은 원래 태스크가 고유 서브공간에서 실행되면 게시 태스크가 CICS 키에서 실행되는 경우를 제외하고는 게시 태스크가 갱신을 시도하고 ECB를 게시할 때 실패합니다.

CICS는 저장영역을 공유해야 하는 트랜잭션이 서브공간 그룹 환경에서 계속 작업할 수 있도록 다음 방법을 지원합니다.

- 관련된 모든 트랜잭션이 공통 서브공간에서 실행되도록 지정할 수 있습니다. 공통 서브공간은 공존하기 위해 저장영역을 공유해야 하는 태스크를 허용하며 이를 시스템의 다른 트랜잭션과 분리합니다. 공통 서브공간에 지정된 트랜잭션에는 다음과 같은 특성이 있습니다.
 - 트랜잭션은 다른 태스크 수명 저장영역에 대해 읽기 및 쓰기 액세스를 가집니다.
 - 고유 서브공간에서 실행되는 트랜잭션의 모든 유형의 저장영역에 액세스할 수 없습니다.
 - CICS 저장영역에 대해서는 읽기 액세스만 가능합니다.

이전 CICS 릴리스의 사용자 키에서 작동하던 관련 트랜잭션의 모든 그룹은 공통 서브공간에서 실행되도록 ISOLATE(NO)로 정의된 경우 z/OS용 CICS Transaction Server, 버전 5 릴리스 3 밑에서 작동해야 합니다. 이를 통해 CICS 및 관련 지원을 설치한 후에 단계적으로 스테이징되는 자체적인 고유 서브공간으로 트랜잭션이 분리되도록 하여 업그레이드를 지원합니다.

- SHARED 옵션을 지정하여 저장영역을 확보하여 공통 저장영역이 SHARED 저장 영역에 있도록 할 수 있습니다.
- 저장영역을 공유 중인 트랜잭션의 애플리케이션이 EXECKEY(CICS)로 모두 정의되도록 할 수 있습니다. 그러면 자체 프로그램은 모든 저장영역에 대해 읽기/쓰기 액세스가 있는 기본 공간에서 해당 프로그램이 실행됩니다. 그렇지만 이 방법은 저장영역을 보호하지 않기 때문에 권장되지는 않습니다.

CICS 상호의존 분석기를 사용하여 WAIT EVENT, WAITCICS, WAIT EXTERNAL 및 MVS POST와 같은 명령이 포함된 트랜잭션을 식별할 수 있습니다. 이 유틸리티에 대한 자세한 내용은 CICS Interdependency Analyzer for z/OS의 내용을 참조하십시오.

제 11 부 트랜지언트 데이터 및 임시 저장영역

트랜지언트 데이터 및 임시 저장영역 큐 제어.

제 38 장 트랜지언트 데이터 제어

CICS 트랜지언트 데이터 제어 기능은 일반화된 큐 처리 기능을 제공합니다. 데이터는 후속적인 내부 또는 외부 처리를 위해 큐에 넣을 수(저장할 수) 있습니다. 애플리케이션에 지정된 선택 대상 데이터는 사전정의된 기호식 *리전 내부* 또는 *리전 외부* 트랜지언트 데이터 큐로(부터) 라우팅될 수 있습니다.

트랜지언트 데이터 큐는 CICS 리전에 할당된 기능과 연관되는 경우 리전 내부이고, 데이터가 CICS 리전의 외부인 대상으로 보내지는 경우 리전 외부입니다. 트랜지언트 데이터 큐는 애플리케이션에 의해 첫 번째 참조 이전에 정의 및 설치해야 합니다.

다음은 수행할 수 있습니다.

- 트랜지언트 데이터 큐에 데이터를 씁니다(WRITEQ TD 명령).
- 트랜지언트 데이터 큐에서 데이터를 읽습니다(READQ TD 명령).
- 리전 내부 트랜지언트 데이터 큐를 삭제합니다(DELETEQ TD 명령).

TD 키워드가 생략되는 경우, 명령은 임시 저장영역에 대한 것으로 가정됩니다. (임시 저장영역에 대한 자세한 정보는 813 페이지의 제 39 장 『임시 저장영역 제어』의 내용을 참조하십시오.)

Java 및 C++

여기에 설명된 API(Application Programming Interface)는 CICS API로서 Java 프로그램에서 사용하지 않습니다. 트랜지언트 데이터 서비스에 액세스하기 위해 JCICS 클래스를 사용하는 Java 프로그램에 대한 정보는 애플리케이션 개발에서 JCICS를 사용하는 Java 프로그래밍 및 JCICS Javadoc 문서를 참조하십시오. CICS C++ 클래스를 사용하는 C++ 프로그램에 대한 정보는 애플리케이션 개발에서 CICS C++ OO 기반 클래스 사용의 내용을 참조하십시오.

리전 내부 트랜지언트 데이터 큐

『리전 내부』는 별도의 태스크로서 실행하는 하나 이상의 프로그램에서 사용할 직접 액세스 저장영역 디바이스의 데이터를 말합니다. 이들 내부 큐로 보내지거나 나오는 데이터를 리전 내부 데이터라고 하며, 가변 길이 레코드로 구성되어야 합니다.

모든 리전 내부 트랜지언트 데이터 대상은 CICS가 관리하는 동일한 VSAM 데이터 세트에서 큐로서 보유됩니다. 리전 내부 대상은 리전 내부 데이터 세트에서 큐를 찾는 정보를 포함한 자원 정의가 필요합니다. 리전 내부 큐는 터미널 또는 출력 데이터 세트와 연관될 수 있습니다. 데이터가 사용자 태스크에 의해 큐에 기록될 때, 큐는 나중에 CICS 리전 내의 다른 태스크에 의해 입력 데이터로 사용될 수 있습니다. 모든 액세스는 순차

적이며, 읽기 및 쓰기 포인터에 의해 통제됩니다. 레코드를 읽은 후에는 또 다른 태스크가 해당 레코드를 나중에 읽을 수 없습니다. 리전 내부 데이터는 궁극적으로 터미널로의 요청 시에 전송되거나 출력 데이터 세트로부터 순차적으로 검색될 수 있습니다.

리전 내부 데이터의 일반 사용은 다음과 같습니다.

- 메시지 교환
- 브로드캐스팅
- 데이터베이스 액세스
- 출력을 여러 터미널로 라우팅(예를 들어, 주문 분산을 위해)
- 데이터의 큐 처리(예를 들어 순서 번호 또는 도착순 우선순위를 위해)
- 데이터 컬렉션(예를 들어, 2780 데이터 전송 터미널로부터의 일괄처리 입력의 경우)

다음 세 유형의 리전 내부 트랜지언트 데이터 큐가 있습니다.

- **복구 불가능 복구 불가능** 리전 내부 트랜지언트 데이터는 CICS의 웹 스타트 시에만 복구됩니다. 작업 단위(UOW)가 복구 불가능 리전 내부 큐를 갱신하고 그 뒤에 갱신을 백아웃하는 경우, 큐에 대해 작성된 갱신은 백아웃되지 **않습니다**.
- **물리적으로 복구 가능** 물리적으로 복구 가능한 리전 내부 트랜지언트 데이터 큐는 웹 및 긴급 재시작 시에 복구됩니다. UOW가 물리적으로 복구 가능한 리전 내부 큐를 갱신하고 그 뒤에 갱신을 백아웃하는 경우, 큐에 수행된 갱신은 백아웃되지 **않습니다**.
- **논리적으로 복구 가능** 논리적으로 복구 가능한 리전 내부 트랜지언트 데이터 큐는 웹 및 긴급 재시작 시에 복구됩니다. UOW가 논리적으로 복구 가능한 리전 내부 큐를 갱신하고 그 뒤에 수행된 변경사항을 백아웃하는 경우, 큐에 작성된 변경사항도 백아웃됩니다. 웹 및 긴급 재시작 시에 논리적으로 복구 가능한 리전 내부 큐의 확장 상태가 복구됩니다. 인플라이트 UOW는 무시됩니다.

애플리케이션이 읽기, 쓰기 또는 삭제 요청을 발행하려고 시도 중이고 인다우트 실패를 겪는 경우, 큐 정의에 WAIT(YES) 및 WAITACTION(REJECT)가 지정된 경우 LOCKED 응답을 수신할 수 있습니다.

리전 외부 큐

리전 외부 큐(데이터 세트)는 CICS 리전 외부의(또는 그 안에 있는) 프로그램이 액세스할 수 있는 모든 순차 디바이스(DASD, 테이프, 프린터 등)에 상주합니다.

일반적으로 순차 리전 외부 큐는 CICS 리전 외부에서 데이터를 저장 및 검색하는 데 사용됩니다. 예를 들어, 한 태스크가 원격 터미널에서 데이터를 읽고, 데이터를 편집하고, 요청을 또 다른 리전에서의 후속 처리를 위해 데이터 세트에 쓸 수 있습니다. 로깅 데이터, 통계 및 트랜잭션 오류 메시지가 리전 외부 큐에 쓸 수 있는 데이터의 예제입니다.

니다. 일반적으로 CICS가 작성하는 리전 외부 데이터는 비CICS 프로그램에 대한 후속 일괄처리 입력을 위한 것입니다. 데이터는 또한 프린터 같은 출력 디바이스로 라우팅될 수 있습니다.

외부 대상으로 가거나 오는 데이터를 리전 외부 데이터라고 부르며 고정 길이나 가변 길이, 블록화 또는 비블록화된 순차 레코드로 구성됩니다. 리전 외부 대상에 대한 레코드 형식은 시스템 프로그래머에 의해 TDQUEUE 자원 정의에서 정의되어야 합니다. 큐 정의에 대한 자세한 내용은 *CICS 자원 정의 안내서*의 내용을 참조하십시오.

JCL을 사용하여 리전 외부 큐에 대한 데이터 세트 정의를 작성하는 경우, 데이터 세트의 DD문이 FREE=CLOSE 피연산자를 포함해서는 안 됩니다. FREE=CLOSE를 지정하는 경우, 큐가 처리완료된 후 다시 열린 후에 읽으려는 시도가 IOERR 상태를 수신할 수 있습니다. *CICS 시스템 정의 안내서*에 CICS에 데이터 세트 정의에 대한 자세한 정보가 있습니다.

간접 큐

리전 내부 및 리전 외부 큐는 간접 큐로 사용할 수 있습니다. 간접 큐는 데이터를 변경되어야 하는 프로그램 자체가 아니라 트랜지언트 데이터 정의만 갖는 여러 큐 중 하나로 라우팅될 수 있다는 점에서 프로그램 유지보수에서 약간의 유연성을 제공합니다.

트랜지언트 데이터 정의가 변경되었을 때, 애플리케이션은 계속 원래 기호 이름을 사용하여 큐로 데이터를 라우팅합니다. 그러나 이 이름은 이제 새 기호 이름을 참조하는 간접 큐입니다. 간접 큐는 트랜지언트 데이터 자원 정의를 사용하여 설정되기 때문에, 애플리케이션 프로그래머는 대개 이것이 수행되는 방법에 관심을 가질 필요가 없습니다. 트랜지언트 데이터 자원 정의에 대한 추가 정보는 TDQUEUE 자원 정의 *CICS 자원 정의 안내서*에 있습니다.

ATI(Automatic Transaction Initiation)

리전 내부 큐에 대해 CICS는 ATI(Automatic Transaction Initiation) 옵션을 제공합니다. ATI의 기초는 0이 아닌 트리거 레벨을 특정 리전 내부 대상에 지정하여 시스템 프로그래머가 작성합니다.

큐에 있는 몇 개의 항목(하나 이상의 프로그램이 실행한 WRITEQ TD 명령으로 작성)이 지정한 트리거 레벨에 도달하면 큐 정의에 지정된 트랜잭션은 자동으로 시작됩니다. 제어가 큐의 데이터를 처리하는 프로그램으로 전달됩니다. 프로그램은 반복적으로 READQ TD 명령을 실행하여 큐를 줄여야 합니다.

큐가 비게 되면 새 ATI 주기가 시작됩니다. 즉, 이전 태스크 실행이 종료되면 지정된 트리거 레벨이 다시 도달할 때 새 태스크가 시작되도록 스케줄됩니다. 새 ATI 주기가 시작되는 정확한 위치는 큐가 논리적으로 복구 가능하게 정의되어 있는지에 따라 다릅니다.

니다. 큐가 없음 또는 물리적인 복구 가능 속성(RECOVSTATUS)으로 정의된 경우, 새 ATI 주기는 큐가 QZERO로 읽혀질 때 시작됩니다. 그렇지만 큐가 논리의 복구 가능 속성으로 정의되면 새 ATI 주기는 큐가 QZERO로 읽힌 후에 태스크가 종료된 후에만 시작됩니다.

자동으로 시작된 태스크가 큐를 비우지 않으면 큐에 대한 액세스는 금지되지 않습니다. 태스크는 큐가 비워지기 전에 일반적으로 또는 비정상 종료됩니다(즉, READQ TD 명령에 대한 응답에서 QZERO 조건이 발생하기 전). 큐 내용이 터미널로 보내지고 이전 태스크가 정상적으로 완료되면 QZERO가 도달하지 않았다는 사실은 트리거 처리가 다시 설정되지 않고 동일한 태스크가 재시작되었음을 나타냅니다. 후속 WRITEQ TD 명령은 트리거 처리가 다시 설정되지 않은 경우 새 태스크를 트리거하지 않습니다.

큐 내용이 파일로 보내지는 경우, 태스크 종료는 QZERO와 동일한 효과가 있습니다. (즉, 트리거 처리가 다시 설정됩니다.) 다음 WRITEQ TD 명령은 트리거 트랜잭션을 시작합니다(트리거 레벨에 도달한 경우).

큐의 트리거 레벨이 0인 경우 태스크가 자동으로 시작되지 않습니다.

큐가 논리적으로 복구 가능한 경우, 트리거 트랜잭션 시작은 다음 동기점까지 지연됩니다.

마지막으로 트리거된 트랜잭션이 큐를 지우기 전에 이상 종료되었거나 MXT 한계에 도달하여 트랜잭션이 시작되지 않아서 이미 트리거 레벨이 초과된 경우 다른 태스크가 스케줄되지 않습니다. 이는 QZERO가 트리거 처리를 다시 설정하기 위해 실행되지 않아서입니다. 큐 내용이 파일을 대상으로 하는 경우, 태스크 종료는 트리거 처리를 다시 설정하고 다음 WRITEQ TD 명령이 새 태스크를 트리거하는 것을 의미합니다.

큐가 비게 될 경우 자동으로 시작된 태스크가 완료되도록 하려면 애플리케이션은 다른 애플리케이션 종속 요소(예: 예상된 레코드 수)보다 우선적으로 QZERO 조건에 대해 애플리케이션이 테스트해야 합니다. QZERO 조건만 비어있는 큐를 표시합니다.

큐 내용이 다른 시스템으로 전송되면 세션 이름은 EIBTRMID에서 보유됩니다. 트랜잭션(시스템 대상으로 시작)이 이상 종료되면 새 트랜잭션은 터미널과 동일한 방법으로 시작됩니다.

ATI에 트랜지언트 데이터 트리거 메커니즘을 사용하는 경우, 동적 트랜잭션 라우팅을 수행하는 기능에 악영향을 주는 트랜잭션간 연관관계가 작성될 수도 있습니다. 트랜잭션 연관관계에 대한 자세한 정보는 379 페이지의 『연관관계』의 내용을 참조하십시오.

트리거 트랜잭션은 불확실한 장애로 인해 문제가 발생하면 다른 곳으로 이동할 수 있습니다. 다른 트리거 트랜잭션은 이동한 UOW가 다음 재동기화 중에 작성한 변경사항을 확약하거나 취소할 때까지 접속되지 않습니다.

트랜지언트 데이터 트리거 레벨 트랜잭션 때문에 시작된 트랜잭션은 이전 홉 데이터를 작성하지 않고 원래 위치에서 실행되는 것처럼 처리됩니다. 이전 홉 데이터에 대한 정보는 이전 홉 데이터 특성을 참조하십시오.

제 39 장 임시 저장영역 제어

CICS 임시 저장영역 제어 기능은 애플리케이션 프로그래머에게 기본 저장영역, 직접 액세스 저장영역 디바이스의 보조 저장영역 또는 임시 저장영역 데이터 공유 풀에 있는 임시 저장영역 큐에 데이터를 저장하는 기능을 제공합니다. 임시 저장영역 큐에 저장된 데이터는 임시 데이터라고 합니다.

다른 임시 저장영역 위치 및 각 위치의 임시 저장영역 큐에 사용할 수 있는 기능에 대한 개요는 기본 임시 저장영역: 성능 개선의 모니터링 및 성능 조정의 내용을 참조하십시오.

CICS 임시 저장영역 제어 기능에는 다음 용도의 명령이 포함됩니다. 이 명령 각각에서 TS 키워드는 생략 가능합니다. 임시 저장영역은 지정되지 않은 것으로 간주됩니다.

- 임시 저장영역 큐에 데이터 쓰기(WRITEQ TS 명령).
- 임시 저장영역 큐에서 데이터 갱신(WRITEQ TS REWRITE 명령).
- 임시 저장영역 큐에서 데이터 읽기(READQ TS 명령).
- 임시 저장영역 큐에서 다음 데이터 읽기(READQ TS NEXT 명령).
- 임시 저장영역 큐 삭제(DELETEQ TS 명령).

프로그래머가 임시 저장영역 큐를 사용하는 방법에 대한 자세한 정보는 295 페이지의 『임시 저장영역 큐』의 내용을 참조하십시오.

임시 저장영역 제어 명령 실행 중에 발생하는 예외 조건은 423 페이지의 『예외 조건 처리』에서 설명됩니다.

이 명령을 사용하는 경우, 동적 트랜잭션 라우팅을 수행하는 기능에 악영향을 주는 트랜잭션간 연관관계가 작성될 수도 있습니다.

이러한 명령을 실행하는 프로그램의 잠재된 문제점을 쉽게 식별하기 위해 CICS Interdependency Analyzer의 스캐너 및 콜렉터 구성요소를 사용할 수 있습니다. 이 기능에 대한 자세한 정보는 CICS Interdependency Analyzer for z/OS의 내용을 참조하고 트랜잭션 연관관계에 대한 자세한 정보는 379 페이지의 『연관관계』의 내용을 참조하십시오.

Java 및 C++

여기에 설명된 API(Application Programming Interface)는 Java 프로그램에서 사용되지 않은 CICS API입니다. 임시 저장영역 서비스에 액세스하기 위해 JCICS 클래스를 사용하는 Java 프로그램에 대한 정보는 애플리케이션 개발에서 JCICS를 사용하는 Java 프로그래밍 및 JCICS Javadoc 문서를 참조하십시오. CICS C++ 클래스를 사용하는

C++ 프로그램에 대한 정보는 애플리케이션 개발에서 CICS C++ OO 기반 클래스 사용의 내용을 참조하십시오.

임시 저장영역 제어에 대한 일반적인 사용

한 개의 레코드만 포함하는 임시 저장영역 큐는 해당 기호 이름으로 액세스할 수 있는 단일 데이터 단위로 처리 가능합니다. 이런 방식으로 임시 저장영역 제어를 사용하면 일반적으로 스크래치패드 기능이 제공됩니다. 이 유형의 저장영역은 READQ TS 명령에 ITEM 옵션을 지정해서 액세스해야 합니다. 그렇지 않으면 ITEMERR 조건이 발생할 수 있습니다.

일반적으로 둘 이상의 레코드가 포함된 임시 저장영역 큐는 직접 액세스 또는 레코드에 대한 반복 액세스가 필요한 경우에만 사용해야 합니다. 트랜지언트 데이터 제어는 순차 데이터 세트에 대한 효율적인 처리 기능을 제공합니다.

임시 저장영역 큐를 사용하는 몇 가지는 다음과 같습니다.

터미널 페이징

태스크는 직접 액세스 데이터 세트에서 큰 마스터 레코드를 검색하고 이를 여러 개의 화면 이미지로 형식화한 후(BMS 사용) 보조 저장영역에 화면 이미지를 임시로 저장하고 필요한 "페이지"(화면 이미지)를 터미널 운영자에게 요청할 수 있습니다. 애플리케이션 프로그래머는 페이지별 고급 페이지, 페이지의 고급 또는 백업 비례수 등에 프로그램을 제공할 수 있습니다(일반화된 루틴 또는 단일 애플리케이션에 고유한 방식 사용).

데이터 세트 일시중단

데이터 콜렉션 태스크가 터미널에서 진행 중이라고 가정해 보십시오. 태스크는 하나 이상의 입력 단위를 읽고 터미널 운영자가 일부 코딩된 입력으로 프로세스를 인터럽트하도록 허용할 수 있습니다. 인터럽트되지 않으면 태스크가 데이터 콜렉션 프로세스를 반복합니다. 인터럽트되면 태스크가 해당하는 불완전 데이터를 임시 저장영역으로 쓰고 종료합니다. 터미널은 이제 다른 트랜잭션(높은 우선순위의 조회일 가능성이 높음)을 처리할 수 있습니다. 터미널이 데이터 콜렉션을 계속할 수 있게 되면 운영자는 『재개』 모드에서 태스크를 시작하고 이로 인해 태스크는 임시 저장영역에서 일시중단된 데이터를 호출하고 인터럽트된 적이 없었던 것처럼 계속 진행됩니다.

사전 인쇄 양식

애플리케이션은 사전 인쇄 양식에 출력으로 데이터가 쓰여지는 것을 허용할 수 있습니다. 이 데이터는 도착하는 대로 임시 저장영역에 저장됩니다. 모든 데이터가 저장되면 우선 유효성이 검증되고 사전 인쇄 양식의 형식에 필요한 순서대로 전송됩니다.

제 12 부 CICS 문서

CICS 문서 핸들러를 사용하여 문서로 알려진 형식화된 데이터 영역을 빌드할 수 있습니다.

형식화된 이러한 영역 또는 문서를 사용하는 방법에 대한 몇 가지 예는 다음과 같습니다.

- COMMAREA 구성
- 웹 클라이언트가 표시할 HTML 데이터 전송
- 인쇄를 위한 표준 형식 작성(예: 자체 레터헤드, 주소 등을 사용)

제 40 장 CICS 문서 및 문서 템플릿

애플리케이션은 EXEC CICS DOCUMENT API(Application Programming Interface)의 명령을 사용하여 문서를 작성하고 작성한 문서에 데이터를 배치할 수 있습니다. 문서 템플릿은 오프라인에서 또는 다른 CICS 프로그램에서 작성할 수 있는 문서의 일부이고 문서를 작성하기 위해 애플리케이션에서 사용됩니다.

문서 및 문서 템플릿은 주로 CICS 웹 지원에서 제공되는 웹 페이지를 생성하는 데 사용됩니다. 문서 및 문서 템플릿은 HTTP 요청 또는 응답의 본문으로 사용되는 HTML을 포함할 수 있습니다. 그러나 이러한 용도로만 제한되지 않습니다.

Java 애플리케이션에서 CICS Java 클래스 라이브러리(JCICS)를 사용하여 문서 서비스에 액세스할 수 있습니다. **Document** 클래스는 EXEC CICS DOCUMENT 명령의 Java 구현을 제공합니다. 클래스 문서의 경우 JCICS 클래스 참조의 Javadoc을 참조하십시오. Java 애플리케이션은 다른 프로그래밍 언어로 쓰여진 애플리케이션에서 작성된 문서를 검색하고 JCICS 클래스를 사용하는 문서에 대해 작업할 수 있습니다.

문서

EXEC CICS DOCUMENT CREATE 명령을 사용하여 애플리케이션에서 비어 있는 문서를 작성한 다음 후속 DOCUMENT INSERT 명령으로 내용을 빌드할 수 있습니다. 또는 DOCUMENT CREATE를 사용하여 한 단계에서 문서를 작성 및 빌드할 수 있습니다. 애플리케이션에서 지정된 데이터 또는 문서 템플릿을 사용하거나 다른 문서를 사용하는 문서를 작성할 수 있습니다. 문서 핸들러는 후속 호출에서 문서를 식별하는 데 사용되는 토큰(DOCTOKEN)을 리턴합니다.

문서가 작성된 후에는 한 개 이상의 DOCUMENT INSERT 명령을 실행하여 그 내용을 확장할 수 있습니다. 다시, 애플리케이션에서 지정된 데이터 또는 문서 템플릿 또는 다른 문서를 추가할 수 있습니다. 또한 데이터 블록 사이 문서에 책갈피를 삽입할 수 있고 책갈피를 사용하여 문서 중간에서 데이터를 추가하거나 바꿀 수 있습니다.

애플리케이션에서 작성된 문서는 문서가 작성되는 CICS 태스크의 길이 동안만 존재합니다. 이는 CICS 태스크에서 마지막 프로그램이 CICS에 제어를 리턴할 때 해당 태스크의 수명 중에 작성된 모든 문서가 삭제됨을 의미합니다. 문서가 다른 태스크에 사용될 예정인 경우 종료 전에 문서를 저장하는 것은 애플리케이션의 책임입니다. DOCUMENT RETRIEVE 명령을 사용하여 문서의 사본을 얻을 수 있습니다. 그러면 애플리케이션은 임시 저장영역 큐와 같이 선택한 위치에 이 사본을 저장할 수 있습니다. 이 사본은 문서를 다시 작성하는 데 사용할 수 있습니다.

문서 템플리트

문서 템플리트는 애플리케이션에서 사용되는 방식에 따라 여러 다른 소스에서 검색할 수 있습니다. 문서 템플리트의 소스는 다음 중 하나일 수 있습니다.

- 파티션된 데이터 세트
- CICS 프로그램
- CICS 파일
- z/OS UNIX System Services 파일
- 임시 저장영역 큐
- 트랜지언트 데이터 큐
- 엑시트 프로그램

문서 템플리트는 문서 템플리트의 소스를 지정하는 DOCTEMPLATE 자원 정의를 사용하여 정의합니다.

문서 템플리트는 정적 데이터 및 기호도 포함할 수 있습니다. 기호는 템플리트가 문서에 추가될 때 즉, DOCUMENT CREATE 또는 DOCUMENT INSERT 명령이 실행될 때 분석되는 변수 데이터를 나타냅니다. 기호를 대체할 값은 애플리케이션에서 DOCUMENT CREATE 명령에 SYMBOLLIST 옵션을 사용하거나 DOCUMENT SET 명령에 SYMBOLLIST 또는 SYMBOL 옵션을 사용하여 지정됩니다.

또한 문서 템플리트는 기호의 기본값을 설정하고, 기호를 식별하며 다른 템플리트를 임베드하기 위한 임베디드 명령을 포함할 수 있습니다.

애플리케이션에서 사용되는 것 이외에도 문서 템플리트는 애플리케이션 없이 웹 클라이언트의 HTTP 요청에 정적 응답을 제공할 수 있도록 URIMAP 정의에서 지정할 수 있습니다. CICS는 문서 템플리트를 웹 페이지 본문으로 사용하여 응답을 생성합니다.

기호 및 기호 목록

기호는 문서 템플리트에 사용되어 애플리케이션이 현재 태스크에 적합한 데이터(예: 사용자 이름 또는 주문 번호)로 문서를 사용자 정의할 수 있도록 합니다. 기호는 템플리트가 문서에 추가될 때 즉, DOCUMENT CREATE 또는 DOCUMENT INSERT 명령이 실행될 때 분석되는 변수 데이터를 나타냅니다.

각 기호에는 이름이 있고 그에 지정되는 값을 가질 수 있습니다. 기호를 포함하는 문서 템플리트가 문서에 삽입되면 CICS 문서 핸들러는 문서 템플리트의 기호가 해당 값으로 대체되는 기호 대체 프로세스에서 기호에 지정된 값을 사용합니다. 예를 들어, 기호 ORDER_NUMBER에 값 0012345가 지정되어 문서 템플리트에서 다음과 같이 사용되는 경우:

Thank you! Your order number is &ORDER_NUMBER;.

기호 대체 후 완료된 문서는 다음과 같습니다.

Thank you! Your order number is 0012345.

기호 참조는 완료된 문서의 텍스트에서 기호의 값이 대체될 위치에 있는 문서 템플릿에 포함됩니다. 기호 참조는 앰퍼샌드(&)가 선행되고 세미콜론(;)으로 종료되는 기호 이름으로 구성됩니다. 문서 템플릿 내 모든 위치에서 기호를 사용할 수 있습니다. 임베디드 템플릿 명령 #set를 사용하여 기호의 기본값을 문서 템플릿의 모든 위치에 포함시킬 수도 있습니다.

문서 템플릿을 사용하여 문서를 작성하는 애플리케이션은 템플릿이 사용될 때 대체될 기호에 값을 지정해야 합니다. DOCUMENT SET 명령 또는 DOCUMENT CREATE 명령을 사용하여 이를 수행할 수 있습니다.

DOCUMENT SET 명령에서 SYMBOL 및 VALUE 옵션을 사용하여 개별 기호에 값을 지정할 수 있습니다. 또는 기호 목록을 지정하여 단일 명령으로 여러 기호를 정의할 수 있습니다. 기호 목록은 각각 이름, 등호 및 값을 포함하는 한 개 이상의 정의와 1바이트 분리문자(기본적으로 앰퍼샌드임)로 구성되는 문자열입니다. DOCUMENT CREATE 또는 DOCUMENT SET 명령에서 SYMBOLLIST 옵션을 사용하여 지정할 수 있습니다. 예를 들어, 다음은 기호 ORDER_NUMBER에 값 '0012345'를 지정하고 기호 COUNTRY에 값 'Germany'를 지정하는 기호 목록입니다.

```
ORDER_NUMBER=0012345&COUNTRY=Germany
```

기호에 지정된 값은 기호 테이블에 보유됩니다. 각 문서 템플릿과 연관된 기호 테이블이 있습니다. #set 명령이 사용되거나 애플리케이션이 기호의 값을 제공하는 경우 기호 이름 및 그 연관 값이 기호 테이블에 추가됩니다. 기호 테이블에 기호가 이미 있을 경우 그 값은 새 값으로 바뀝니다. 애플리케이션에서 제공되는 기호 정의는 #set 명령에서 제공되는 기본값을 덮어씁니다.

문서 템플릿이 DOCUMENT CREATE 또는 DOCUMENT INSERT 명령을 통해 문서에 삽입되면 기호 테이블에 지정되는 해당 현재 값을 사용하여 기호 테이블에 있는 모든 기호에 대한 기호 대체가 수행됩니다. 애플리케이션 또는 #set 명령에서 기호가 지정되지 않은 경우 기호 대체는 발생하지 않으며 완료된 문서는 기호 이름 또는 기호를 지정하는 #echo 명령을 포함합니다.

기호를 포함하는 템플릿을 사용하는 경우 문서에 이 템플릿을 삽입하기 전에 기호에 필요로 하는 값을 지정해야 합니다. 템플릿을 삽입했지만 템플릿 내에 있는 기호에 값을 지정하지 않은 경우 기호는 대체되지 않습니다(이는 기호 목록 또는 기호 값을 지정하지 않고 템플릿에서 문서를 작성하는 경우에 발생함). 문서에 템플릿을 삽입한 후에는 CICS가 기호를 대신해서 문서에 입력한 값을 변경할 수 없습니다. 템플릿을 삽입한 후에 기호 값을 지정하면 값이 기호 테이블에 배치되어 다음에 템플릿이 문서에 삽입될 때 사용되지만 변경사항은 문서에 이미 삽입된 값에는 영향을 미치지 않습니다.

기호 및 기호 목록의 데이터 형식

DOCUMENT 애플리케이션 프로그래밍 인터페이스에서 기호 및 기호 목록에 대한 지원은 HTML 스펙(<http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4>)에 설명된 대로 데이터를 **application/x-www-form-urlencoded**의 콘텐츠 유형으로 해석하도록 디자인됩니다. 이는 데이터가 웹 페이지의 양식에 입력되었을 때 데이터가 리턴되는 형식입니다(웹 페이지는 CICS 문서 템플릿에 대한 가장 일반적인 용도임).

url로 인코딩된 데이터 형식은 앰퍼샌드로 구분되는 이름과 값 쌍의 시퀀스입니다. 예를 들어, 다음과 같습니다.

```
firstname=Irving&lastname=Berlin
```

값은 일반적으로 사용자가 양식에 입력한 데이터입니다.

그러나 양식에 입력된 데이터에 url로 인코딩된 데이터 형식으로 된 특수한 의미를 가지는 문자가 포함되는 경우 양식에 입력된 문자는 이름과 값 쌍을 구분하기 위해 사용된 문자와 구별되어야 합니다. 이를 수행하기 위해 문자 이스케이핑 프로세스가 수행됩니다. 이스케이프된 문자는 세 문자 시퀀스 %xx로 대체된 문자입니다. 여기서 xx는 문자 ASCII 인코딩의 16진 값입니다.

앰퍼샌드(&) 및 등호(=), 또한 퍼센트 부호(%) (이스케이프 시퀀스를 시작하는 데 사용되므로)는 모두 url로 인코딩된 데이터로 이스케이프되어야 합니다. 공백 문자는 일부 경우에 구분 기호로 사용되기 때문에 공백 문자도 항상 이스케이프됩니다. 그러나 공백은 너무 흔하기 때문에 url로 인코딩된 데이터 유형을 통해 공백이 이스케이프 시퀀스가 아닌 더하기 부호(+)로 인코딩되게 할 수 있습니다. 따라서 양식에 입력된 더하기 부호도 이스케이프되어야 함을 의미합니다.

예를 들어, 양식에 다음 이름 및 값이 포함되는 경우입니다.

```
sum          8+11=19
rate         19%
composers    George & Ira Gershwin
```

이 데이터의 이스케이프된 url 인코딩 표시는 다음과 같습니다.

```
sum=8%2b11%3d19&rate=19%25&composers=George+%26+Ira+Gershwin
```

값에 있는 공백 문자가 더하기 부호로 인코딩되었고 값에 있는 더하기 부호, 등호, 퍼센트 부호 및 앰퍼샌드는 이스케이프된 시퀀스로 인코딩되었습니다.

EXEC CICS DOCUMENT CREATE 또는 DOCUMENT SET 명령의 SYMBOLLIST 옵션을 사용하여 지정하는 기호 목록은 원칙적으로 url로 인코딩된 형식의 이름 및 값 쌍의 목록을 포함합니다. 그러나 CICS는 다음과 같은 방식으로 이 구문을 확장했습니다.

- 공백은 이스케이프되지 않아도 됩니다. 공백은 한 개의 공백 또는 더하기 부호 또는 이스케이프 시퀀스 %20으로 표시할 수 있습니다.
- 이름과 값 쌍 사이의 구분 기호는 앰퍼샌드가 아니어도 됩니다. 명령의 DELIMITER 옵션을 사용하여 대체 구분 기호를 지정할 수 있습니다.

일반적으로 기호 값이 기호 테이블에 삽입되면 기호 값에 대해 이스케이프 해제 처리가 수행되기 때문에 특수 코딩을 사용하여 지정된 모든 문자는 의도한 문자로 변환됩니다. 예를 들어, 더하기 부호는 공백으로 변환되고 이스케이프 시퀀스는 정확한 문자로 변환됩니다. 그러나 DOCUMENT CREATE 또는 DOCUMENT SET 명령에 UNESCAPED 옵션을 지정하면 변환이 발생하지 않고 기호 값은 사용자가 입력한 그대로 기호 테이블에 입력됩니다.

HTML 설명의 기호

문서 템플리트의 HTML 설명 내에서 기호는 일반적으로 무시되고 CICS 문서 핸들러는 기호 대체를 수행하지 않습니다. HTML 설명은 열기 마크업 <!-- 및 닫기 마크업 -->에 의해 구분됩니다.

문서 템플리트의 HTML 설명 내에서 기호 대체가 발생하도록 하려는 경우 설명의 열기 및 닫기 구분 기호 대신 기호를 사용하고 문서 템플리트에 #set 명령을 사용하여 해당 기호에 대한 값으로 열기 및 닫기 구분 기호를 제공할 수 있습니다. 이를 수행하면 CICS가 주변에 구분 기호 대신 기호를 사용하는 설명 전체에 대해 기호 대체를 수행합니다.

예를 들어, 문서 템플리트는 다음을 포함할 수 있습니다.

```
<!--#set var=OC value='<!--'-->
<!--#set var=CC value='-->'-->
```

```
&OC; A comment containing my text &SYM; &CC;
```

애플리케이션이 기호 SYM에 'Example text' 값을 지정하는 경우 CICS는 해당 기호의 값을 대체하고 기호 OC 및 CC의 값도 대체하여 다음과 같은 HTML 출력을 생성합니다.

```
<!-- A comment containing my text Example text -->
```

문서 템플리트의 캐싱 및 새로 고치기

성능을 개선하기 위해 CICS 문서 핸들러는 대부분의 문서 템플리트 사본을 캐시합니다. 애플리케이션은 템플리트를 참조할 때 캐시된 사본을 사용하여 성능을 개선합니다. 문서 템플리트가 변경되면 언제든지 캐시된 사본을 새로 고칠 수 있습니다. 프로그램의 새 사본과 문서 템플리트로 정의되는 엑시트 프로그램을 단계적으로 시작할 수도 있습니다.

CICS는 항상 다음 유형의 문서 템플릿 사본을 캐시합니다.

- 파티션된 데이터 세트의 템플릿
- CICS 파일의 템플릿
- z/OS UNIX System Services 파일의 템플릿
- 임시 저장영역 큐의 템플릿
- 트랜지언트 데이터 큐의 템플릿

CICS가 실행되는 동안 이러한 문서 템플릿 유형 중 하나가 개별적으로 설치되면 CICS 문서 핸들러의 저장영역으로 읽어 들입니다. 문서 템플릿에 액세스하기 위한 애플리케이션 요청에서 캐시된 템플릿 사본을 수신하므로 CICS는 문서 템플릿이 저장되는 위치에 매번 액세스하지 않아도 됩니다. CICS 시동 중 설치되는 문서 템플릿은 이때 캐시되지 않습니다. 이러한 문서 템플릿 각각은 애플리케이션에서 처음으로 참조될 때 캐시됩니다.

캐시된 문서 템플릿을 변경하는 경우 CEMT 또는 EXEC CICS SET DOCTEMPLATE NEWCOPY 명령을 사용하여 문서 템플릿의 캐시된 사본을 새로 고칠 수 있습니다. (EXEC CICS DOCUMENT API의 일부가 아닌 SET DOCTEMPLATE 명령으로 템플릿의 48자 이름이 아닌 문서 템플릿을 정의하는 DOCTEMPLATE 자원 정의의 이름을 지정해야 함에 주의하십시오.)

위에 나열된 문서 템플릿의 유형에 대해 SET DOCTEMPLATE NEWCOPY 명령은 CICS 문서 핸들러에 의해 현재 캐시되는 문서 템플릿의 사본을 삭제하고 이 사본을 문서 템플릿이 저장되는 위치에서 읽은 사본과 바꿉니다. (파티션된 데이터 세트에 있는 템플릿의 경우 CICS는 먼저 BLDL(빌드 목록)을 수행하여 가장 최근 디렉토리 정보를 얻고 멤버를 다시 읽습니다.) 새 캐시된 사본이 작성되면 문서 템플릿 사용에 대한 후속 요청은 새 사본을 사용합니다. 새 사본은 다른 태스크의 요청과 마찬가지로 같은 태스크 내에 있는 나중 요청이 사용합니다.

CICS 시스템의 저장영역이 부족해지면 문서 핸들러가 문서 템플릿의 캐시된 사본 일부를 삭제하여 저장영역 제한조건을 완화하려 시도합니다. 삭제할 문서 템플릿은 캐시된 사본이 작성된 후의 시간을 고려하여(새로 작성된 사본이 즉시 삭제되지 않도록) 크기(가장 큰 크기부터 시작) 순으로 선택됩니다.

CICS 시스템이 워 스타트로 다시 시작되면 이전에 캐시된 문서 템플릿이 다시 로드되지 않습니다. 캐시는 애플리케이션이 각 문서 템플릿을 처음으로 참조할 때 다시 채워집니다.

문서 템플릿에 대해 수집된 CICS 통계는 각 문서 템플릿이 참조되는 횟수 및 캐시된 사본이 작성되고 새로 고쳐지고 사용 및 삭제된 횟수를 보여줍니다.

CICS 프로그램의 템플리트

CICS 프로그램에서 검색된 문서 템플리트는 문서 핸들러에 의해 캐시되지 않습니다. 프로그램이 이미 CICS의 임의 위치에 캐시된 상태이기 때문입니다.

이러한 유형의 문서 템플리트에 대해 SET DOCTEMPLATE NEWCOPY 명령을 사용하여 새 프로그램 사본을 단계적으로 시작할 수 있습니다. 이 명령은 지정된 프로그램의 SET PROGRAM PHASEIN에 해당됩니다. 문서 템플리트를 사용하기 위한 후속 요청(같은 태스크 내 나중 요청 포함)은 새 사본을 사용합니다.

엑시트 프로그램의 템플리트

엑시트 프로그램에서 생성되는 문서 템플리트의 경우 엑시트 프로그램이 문서 템플리트 사본을 문서 핸들러로 캐시해야 하는지 여부를 지정합니다(엑시트 매개변수 목록에서). 기본값은 문서 템플리트를 캐시하지 않는 것입니다. 동적으로 변경되는 템플리트는 캐시되지 않아야 하지만 템플리트가 변경되지 않는 경우 캐싱은 요청의 성능을 향상시키기 때문에 적합합니다. 엑시트 프로그램이 캐싱을 지정하지 않을 경우 캐시된 사본은 애플리케이션이 문서 템플리트를 처음으로 참조할 때 작성됩니다.

이러한 유형의 문서 템플리트에 대해 SET DOCTEMPLATE NEWCOPY 명령을 사용하여 엑시트 프로그램의 새 사본을 단계적으로 시작할 수 있습니다. 이 명령은 지정된 엑시트 프로그램의 SET PROGRAM PHASEIN에 해당됩니다. 명령을 실행하면 CICS는 문서 템플리트의 캐시된 사본을 삭제하고, 프로그램의 새 사본을 단계적으로 시작하며, 엑시트 프로그램이 캐싱을 지정하는 경우 문서 템플리트의 캐시된 사본을 새로 작성합니다. 새로 고쳐진 엑시트 프로그램은 캐싱을 수행해야 하는지 여부 및 CICS가 변경을 승인하는지 여부에 대해 다른 설정을 지정할 수 있습니다.

문서에 대한 코드 페이지 변환

애플리케이션이 작성하는 문서가 다른 플랫폼에서 실행되는 시스템으로 전송될 수 있습니다(예: 문서가 클라이언트에게 웹 페이지를 제공하는 경우). CICS 시스템이 사용하는 코드 페이지에 있는 텍스트 형식 데이터는 대상 시스템에서 사용되는 코드 페이지로 변환되어야 합니다. 이 프로세스를 코드 페이지 변환이라고 합니다.

CICS 시스템이 사용하는 코드 페이지는 CICS가 HTTP 클라이언트 역할을 하는 경우를 제외하고 보통 호스트 코드 페이지라고 합니다. 대상 시스템이 사용하는 코드 페이지는 클라이언트 코드 페이지라고 합니다. 또는 대상 시스템이 웹 클라이언트 또는 ASCII를 사용하는 서버인 경우 이 코드 페이지는 문자 세트로 참조될 수 있습니다.

CICS 문서가 작성된 코드 페이지에 대한 정보를 해당 문서가 포함하게 할 수 있습니다. EXEC CICS DOCUMENT CREATE 및 EXEC CICS DOCUMENT INSERT 명령을 사용하여 문서를 작성할 때 HOSTCODEPAGE 옵션을 TEXT, FROM,

TEMPLATE 및 SYMBOL 옵션과 함께 사용하여 데이터의 해당 블록에 대한 코드 페이지를 표시할 수 있습니다. 각 블록은 서로 다른 코드 페이지에서 지정할 수 있습니다.

EXEC CICS DOCUMENT RETRIEVE 명령을 사용하여 전송할 문서를 검색할 때 CHARACTERSET 옵션을 지정하여 문서 핸들러가 각 호스트 코드 페이지의 모든 개별 블록을 대상 시스템에서 사용하기에 적합한 단일 클라이언트 코드 페이지로 변환하게 할 수 있습니다.

CICS 웹 지원에서 CICS 문서가 EXEC CICS WEB API 명령을 사용하여 웹 인식 애플리케이션이 전송할 문서로 지정되는 경우 EXEC CICS DOCUMENT RETRIEVE 명령은 이 애플리케이션에서 사용되지 않습니다. 대신 CICS 문서의 문서 토큰이 지정되고 CICS는 문서의 검색을 관리합니다. 클라이언트 코드 페이지로의 변환은 애플리케이션이 EXEC CICS WEB API 명령에서 지정하는 옵션에 따라 CICS에서 처리됩니다.

또한 CICS 웹 지원에서 웹 클라이언트에 정적 응답을 제공하도록 CICS 문서 템플릿이 URIMAP 정의에서 지정되는 경우 클라이언트 코드 페이지로의 변환은 CICS에 의해 처리됩니다. 템플릿이 있는 호스트 코드 페이지와 이 코드 페이지가 변환되어야 하는 클라이언트 코드 페이지는 URIMAP 정의에서 지정됩니다. 정적 응답이 요구되는 경우 CICS는 템플릿을 사용하여 문서를 작성하고 문서를 검색하며 적절한 코드 페이지 변환을 수행합니다.

제 41 장 문서 템플릿 설정

문서 템플릿은 DOCTEMPLATE 자원 정의를 사용하여 정의하는 CICS 자원입니다.

EXEC CICS DOCUMENT API 명령에서 템플릿을 참조하기 위해 사용되는 템플릿의 이름은 자원 정의의 TEMPLATENAME 속성에서 지정됩니다. DOCTEMPLATE 자원 정의는 템플릿의 자원, 데이터의 형식(2진 또는 EBCDIC) 및 CICS가 템플릿의 각 레코드에 캐리지 리턴 줄 바꾸기를 추가하는지 여부도 지정합니다.

파티션된 데이터 세트의 템플릿

BMS 맵핑 세트 정의에서 작성된 HTML 템플릿은 파티션된 데이터 세트에 저장됩니다.

파티션된 데이터 세트에 저장된 문서 템플릿에는 해당 DOCTEMPLATE 자원 정의에 MEMBERNAME 속성이 있습니다.

CICS는 사용자가 해당 DOCTEMPLATE 정의를 설치할 때 파티션된 데이터 세트에서 템플릿의 사본을 로드합니다. CICS가 실행되는 동안 파티션된 데이터 세트에서 템플릿을 수정하는 경우 변경사항을 활성화하려면 SET DOCTEMPLATE NEWCOPY 명령을 실행하십시오. CICS는 먼저 BLDL(목록 빌드)을 수행하여 가장 최근 디렉토리 정보를 얻고 멤버를 다시 읽습니다.

템플릿을 저장하는 데 사용되는 파티션된 데이터 세트의 형식은 다음 레코드 형식 중 하나일 수 있습니다.

- FB(고정 길이 블록화됨)
- VB(변수 길이 블록화됨)
- U(블록화되지 않음)

레코드는 다음의 경우에 순서 번호를 포함할 수 있습니다.

- 레코드 형식이 FB이고 레코드 길이가 80인 경우 순서 번호는 위치 73에서 80까지 범위에 있어야 합니다.
- 레코드 형식이 VB인 경우 순서 번호는 위치 1에서 8까지 범위에 있어야 합니다.

다른 경우에는 레코드에 순서 번호가 없어야 합니다. 순서 번호를 사용하는 경우 해당 번호는 모든 레코드에 표시되어야 하고 부분적으로 순서화된 멤버는 사용하지 않습니다.

파티션된 데이터 세트에 저장된 템플릿은 CICS 문서 핸들러에 의해 캐시됩니다.

z/OS UNIX System Services 파일의 템플리트

z/OS UNIX System Services 파일은 문서 템플리트로 정의할 수 있습니다. z/OS UNIX 파일이 작성 및 편집하기에 가장 편리한 문서 템플리트 유형이라는 것을 알게 될 것입니다.

z/OS UNIX 파일에 저장되는 문서 템플리트는 해당 DOCTEMPLATE 자원 정의에 HFSFILE 속성이 있습니다.

CICS 웹 지원에서 URIMAP 정의를 사용하여 z/OS UNIX 파일을 정적 응답으로 직접 전달할 수 있습니다. 이 경우에 z/OS UNIX 파일은 문서 템플리트로 정의하지 않아도 됩니다(문서 템플리트인 경우에도). 그러나 기호 대체를 수행하거나 웹 인식 애플리케이션이 파일에 액세스해야 하는 경우에는 이 파일을 문서 템플리트로 정의해야 합니다.

DOCTEMPLATE 정의에서 최대 255자 길이의 완전한 파일 이름을 제공해야 합니다. CICS 리전에는 z/OS UNIX에 대한 액세스 권한이 있어야 하고 파일 자체 및 파일이 포함된 디렉토리에 대한 액세스 권한이 있어야 합니다. CICS의 Java 애플리케이션의 CICS 리전에 z/OS UNIX 디렉토리 및 파일에 대한 액세스 권한 제공에서는 이를 실행하는 방법에 대해 설명합니다.

파일의 모든 데이터는 문서 템플리트로 사용됩니다.

z/OS UNIX 파일에 저장된 템플리트는 CICS에 의해 캐시됩니다. SET DOCTEMPLATE NEWCOPY 명령을 사용하여 현재 캐시되는 문서 템플리트의 사본을 삭제하고 문서 템플리트가 저장되는 위치에서 읽은 사본으로 바꿀 수 있습니다.

CICS 파일, 임시 저장영역 또는 트랜지언트 데이터의 템플리트

문서 템플리트에서 애플리케이션의 동적 데이터를 사용하려는 경우 다음 중 한 자원의 사용을 고려합니다.

사용할 자원은 다음에 따라 다릅니다.

- 애플리케이션이 데이터를 저장하는 방식
- 기존 데이터를 바로 템플리트에서 사용할 수 있는지 또는 수정이 필요한지 여부
- 템플리트에서 데이터를 사용한 후 데이터를 보존할지 여부

일반적으로 템플리트가 문서에 삽입되면 자원에 포함된 모든 데이터가 사용됩니다.

이러한 자원에 저장된 문서 템플리트에는 DOCTEMPLATE 자원 정의에 속성 FILE, TSQUEUE 및 TDQUEUE가 있습니다.

속성 TSQUEUE 또는 TDQUEUE로 정의되는 DOCTEMPLATE를 사용하는 경우 DOCTEMPLATE 버퍼의 끝에 2바이트(캐리지 리턴 및 줄 바꾸기)가 추가됩니다.

임시 저장영역

ITEM 번호에 따라 순차적으로 큐를 읽기 때문에 다른 애플리케이션이 읽은 레코드가 어떤 레코드인지에 관계없이 큐에 있는 모든 레코드를 읽습니다.

트랜지언트 데이터

리전 외부 트랜지언트 데이터 큐는 문서 템플릿으로 사용하기에 적합합니다. 큐에 있는 모든 레코드를 읽습니다.

리전 내부 트랜지언트 데이터 큐는 문서 템플릿으로 사용하기에 적합합니다. 트랜지언트 데이터는 파괴성 읽기를 사용하기 때문에 트랜지언트 데이터 큐의 데이터를 템플릿에 삽입하면 이 큐의 내용은 더 이상 다른 애플리케이션이 사용할 수 없게 됩니다.

CICS 파일

- ESDS(Entry-Sequenced Data Set)의 경우 상대 바이트 주소의 순서로 읽습니다.
- RRDS(Relative Record Data Set)의 경우 상대 레코드 번호의 순서로 읽습니다.
- 기타 데이터 세트의 경우 키 필드의 순서로 읽습니다.

문서 템플릿으로 사용할 CICS 파일은 BROWSE 속성으로 지정되어야 합니다.

이러한 형식 중 어느 하나로 저장되는 템플릿은 CICS에 의해 캐시됩니다. SET DOCTEMPLATE NEWCOPY 명령을 사용하여 현재 캐시되는 문서 템플릿의 사본을 삭제하고 문서 템플릿이 저장되는 위치에서 읽은 사본으로 바꿀 수 있습니다.

CICS 프로그램의 템플릿

다른 형식의 경우 첫 번째 요청 후 캐싱으로 오버헤드를 최소화하기는 하지만 프로그램의 문서 템플릿은 템플릿 검색에 대한 오버헤드가 가장 낮습니다. 프로그램은 동일한 템플릿을 여러 애플리케이션이 사용하는 경우 선택할 수 있는 적합한 형식입니다. CICS 프로그램은 템플릿이 자주 변경되지 않는 경우에 가장 적합합니다. 프로그램은 쉽게 편집할 수 없고 다시 컴파일해야 하기 때문입니다.

이 태스크 정보

CICS 프로그램에 포함된 문서 템플릿은 해당 DOCTEMPLATE 자원 정의에 PROGRAM 속성이 있습니다.

CICS 프로그램에 포함된 템플릿은 CICS 문서 핸들러로 캐시되지 않습니다. 프로그램이 이미 CICS 로더에 의해 캐시되었습니다.

이러한 유형의 문서 템플릿에 대해 SET DOCTEMPLATE NEWCOPY 명령을 사용하여 새 프로그램 사본을 단계적으로 시작할 수 있습니다. 이 명령은 지정된 프로그램의 SET PROGRAM PHASEIN에 해당됩니다. 문서 템플릿을 사용하기 위한 후속 요청(같은 태스크 내 나중 요청 포함)은 새 사본을 사용합니다.

템플릿을 포함하는 프로그램을 코드화하려면 다음을 수행하십시오.

프로시저

1. 다음을 포함하는 어셈블러 CSECT를 코드화하십시오.
 - a. 템플릿의 시작을 선언하는 ENTRY문
 - b. 템플릿에 포함시킬 텍스트를 정의하는 문자 상수(DC문)
 - c. END문
 - d. 문서 템플릿에 대한 프롤로그 및 에필로그를 생성하기 위한 DFHDHTL 매크로 호출 829 페이지의 『DFHDHTL - 프로그램 템플릿 프롤로그 및 에필로그 매크로』에서는 이 매크로를 문서화합니다.

예제:

```
DOCTPROG CSECT
DOCTPROG AMODE 31
DOCTPROG RMODE ANY
    DFHDHTL TYPE=INITIAL,ENTRY=WKLYHDR
    DC    CL4'<HR>'
    DC    CL29'<H2>Weekly Status Report</H2>'
    DFHDHTL TYPE=FINAL
    END    WKLYHDR
```

템플릿 프로그램은 정확한 8바이트 배수여야 하기 때문에 DFHDHTL 매크로가 사용됩니다. 여기에 해당되지 않는 경우에는 바인더가 위조 2진 문자를 템플릿의 끝에 삽입할 수 있습니다. 그러면 이 템플릿이 문서에서 사용될 때 예측하지 못한 출력이 생성될 수 있습니다. DFHDHTL 매크로는 필요한 채우기를 작성합니다.

2. 프로그램을 어셈블하고 CICS 애플리케이션 라이브러리로 링크 편집합니다. 프로그램에 지정한 이름이 시작점의 이름과 다를 수 있습니다.
3. PROGRAM 속성에서 프로그램의 이름을 지정하는 DOCTEMPLATE 자원 정의를 작성하고 설치하십시오.

다음에 수행할 작업

CICS가 첫 번째 참조에서 프로그램을 자동 설치하거나 사용자가 PROGRAM 자원 정의를 작성 및 설치할 수 있습니다.

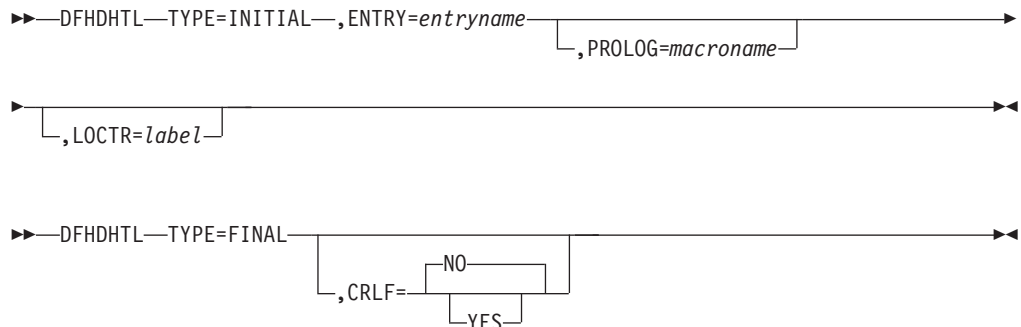
DFHDHTL - 프로그램 템플리트 프롤로그 및 에필로그 매크로

DFHDHTL 매크로는 CICS 프로그램에 포함된 문서 템플리트의 프롤로그와 에필로그를 생성하는 데 사용됩니다(DOCTEMPLATE 자원 정의에서 속성 PROGRAM과 함께 사용).

다음 템플리트는 시작점과 모듈의 끝 사이 로드 모듈의 영역을 차지하는 것으로 정의됩니다. 그러나 템플리트 프로그램의 소스가 정확한 8바이트 배수를 생성하지 않는 경우 연계 편집기 또는 바인더가 웹 브라우저에 전송되면 예측하지 못한 영향을 미치는 위조 2진 문자를 삽입할 수 있습니다. DFHDHTL 매크로 템플리트 로드 모듈이 항상 정확한 8바이트 배수가 되도록 하는 데 도움이 됩니다.

TYPE=INITIAL에 의해 작성되는 프롤로그는 사용자 지정 매크로(DFHVM으로 기본값 설정됨) 및 템플리트 본문을 추적하는 새 위치 카운터(LOCTR)를 호출하여 모듈 아이캐처를 생성합니다.

TYPE=FINAL에 의해 작성되는 에필로그는 지금까지의 템플리트의 길이를 계산하고 그 값이 홀수인 경우 한 개의 공백 문자를 생성합니다. 이렇게 하면 시작점이 반자 경계에 있게 할 수 있습니다. 그러면 에필로그가 프롤로그의 원래 LOCTR로 되돌아가서 모듈 길이를 8바이트 배수로 만들기 위해 시작점 앞에 충분한 문자를 생성하여 모듈 크기를 반올림합니다.



TYPE={INITIAL | FINAL}

프롤로그(INITIAL)가 생성될지 또는 에필로그(FINAL)가 생성될지 지정합니다.

ENTRY=entryname

모듈 항목에 레이블을 지정합니다. 이는 템플리트가 시작되는 위치입니다.

PROLOG=macroname

프롤로그에서 모듈 ID 데이터를 생성하기 위해 호출할 매크로를 지정합니다. 기본값은 DFHVM입니다.

LOCTR=label

프롤로그 코드에서 생성된 템플리트를 분리하는 내부적으로 생성된 위치 카운터(LOCTR)문에 레이블을 지정합니다. 기본값은 내부적으로 생성된 이름입니다.

CRLF=NO | YES

에필로그 코드를 닫기 CRLF(캐리지 리턴 줄 바꾸기)로 종료할지 여부를 지정합니다. 기본값은 NO입니다.

엑시트 프로그램의 템플리트

엑시트 프로그램은 템플리트의 내용이 동적으로 변하는 경우 또는 비CICS 자원(예: DB2)에서 템플리트의 내용을 검색하려는 경우에 가장 적합합니다.

애플리케이션이 엑시트 프로그램에서 작성되는 것으로 정의되는 템플리트를 요청하면 CICS가 지정된 프로그램을 호출하고 통신 영역을 전달합니다.

엑시트 프로그램에 포함된 템플리트는 프로그램의 엑시트 매개변수 목록에서 지정하는 경우에만 CICS에 의해 캐시됩니다. 기본값은 템플리트가 캐시되지 않는 것입니다. 엑시트 프로그램이 캐싱을 지정하지 않을 경우 캐시된 사본은 애플리케이션이 문서 템플리트를 처음으로 참조할 때 작성됩니다.

SET DOCTEMPLATE NEWCOPY 명령을 사용하여 엑시트 프로그램의 새 사본을 단계적으로 시작할 수 있습니다. 이 명령은 지정된 엑시트 프로그램의 SET PROGRAM PHASEIN에 해당됩니다. 이 명령을 실행하면 CICS는 문서 템플리트의 캐시된 사본을 삭제하고 프로그램의 새 사본을 단계적으로 시작하며 엑시트 프로그램이 캐싱을 지정하는 경우 문서 템플리트의 새 캐시된 사본을 작성합니다. 새로 고쳐진 엑시트 프로그램은 캐싱을 수행해야 하는지 여부 및 CICS가 변경을 승인하는지 여부에 대해 다른 설정을 지정할 수 있습니다.

엑시트 프로그램에서 템플리트를 위한 통신 영역

문서 템플리트를 제공하는 엑시트 프로그램의 통신 영역은 CICS 제공 카피북에 의해 맵핑됩니다.

제공되는 카피북은 다음과 같습니다.

- DFHDHTXD(어셈블러)
- DFHDHTXH(C)
- DFHDHTXL(PL/I)
- DFHDHTXO(COBOL)

통신 영역에는 다음과 같은 필드가 포함됩니다.

dhtx_template_name_ptr

요청되고 있는 템플리트의 이름(최대 48자)에 대한 포인터를 포함합니다.

dhtx_buffer_ptr

엑시트 프로그램이 템플리트를 리턴하는 CICS 제공 버퍼의 포인터를 포함합니다.

dhtx_buffer_len

(전자 2진) 엑시트 프로그램이 템플리트를 리턴하는 CICS 제공 버퍼의 길이를 포함합니다.

dhtx_message_len

(전자 2진) 이 필드를 사용하여 엑시트 프로그램을 템플리트 리턴에 사용할 수 없을 때 발행하는 메시지 길이를 리턴합니다. 메시지가 없는 경우 값 0을 리턴합니다.

dhtx_message_ptr

이 필드를 사용하여 엑시트 프로그램이 성공하지 못한 이유를 설명하는 메시지 포인터를 리턴합니다. CICS는 CSDH 트랜지언트 데이터 대상에 이 메시지를 씁니다. 메시지가 없는 경우 값 0을 리턴합니다.

dhtx_template_len

(전자 2진) 이 필드를 사용하여 템플리트의 실제 길이를 리턴합니다.

dhtx_append_cr1f

문자 '1'(추가) 또는 '0'(추가하지 않음)을 사용하여 각 행의 끝에 캐리지 리턴 및 줄 바꾸기를 추가할지 여부를 지정합니다.

dhtx_return_code

(전자 2진) 이 필드를 사용하여 다음과 같이 엑시트 프로그램이 성공적으로 템플리트를 리턴했는지 여부를 표시합니다.

- 리턴 코드 0은 엑시트가 템플리트를 리턴했음을 나타냅니다.
- 리턴 코드 8은 엑시트가 템플리트를 리턴하지 않았음을 나타냅니다. 이 경우 CICS는 애플리케이션에서 TEMPLATERR 조건을 제기합니다.

dhtx_cache_response

문자 '1'(캐시) 또는 '0'(캐시하지 않음)을 사용하여 엑시트 프로그램의 출력이 CICS 문서 핸들러에 의해 캐시되어야 하는지 여부를 지정합니다. dhtx_cache_response의 값은 '0'으로 초기화되어 엑시트가 이 값을 변경하지 않으면 기본 조치는 엑시트 프로그램의 응답을 캐시하지 않는 것입니다.

문서 템플리트가 캐시될 때 후속 요청은 캐시된 사본을 수신합니다. EXEC CICS SET DOCTEMPLATE NEWCOPY 명령이 실행되어 엑시트 프로그램 및 캐시된 사본을 새로 고칠 때까지 캐시된 사본이 사용 가능한 동안은 엑시트 프로그램이 호출되지 않습니다. 새로 고쳐진 엑시트 프로그램은 dhtx_cache_response에 다른 값을 지정할 수 있고 CICS는 변경을 승인합니다.

동적으로 변경되는 템플리트는 캐시되지 않아야 하지만 템플리트가 변경되지 않는 경우 캐싱은 요청의 성능을 향상시키기 때문에 적합합니다.

리턴될 템플릿이 `dhtx_buffer_len`보다 길면 이 템플릿은 길이 `dhtx_buffer_len`에 맞게 잘라야 하고 엑시트 프로그램은 `dhtx_template_len`에 필요한 길이를 설정해야 합니다. 그러면 더 큰 버퍼로 엑시트 프로그램이 다시 호출됩니다.

엑시트 프로그램이 리턴 코드 8을 설정한 경우 설명 메시지를 리턴할 수 있고 이 메시지는 CSDH 트랜지언트 데이터 대상에 기록됩니다. `dhtx_message_ptr` 및 `dhtx_message_len`에 메시지 주소 및 길이를 각각 리턴하십시오. 메시지를 포함하는 저장영역은 엑시트 프로그램의 호출자에 대해 액세스 가능해야 합니다. 예를 들어, 엑시트 프로그램은 GETMAIN 명령을 실행하여 메시지의 저장영역을 확보할 수 있습니다. CICS는 명령에 SHARED 옵션을 지정하지 않는 한 태스크가 종료되면 저장영역을 해제합니다.

문서 템플릿에서 기호 사용

완료된 문서의 텍스트에서 기호의 값이 대체될 위치에 있는 문서 템플릿에서 기호 참조 또는 `#echo` 명령을 사용합니다. 템플릿에서 `#set` 명령을 사용하여 기호의 기본값을 지정할 수 있습니다.

시작하기 전에

기호의 이름은 대문자 및 소문자, 숫자 및 특수 문자 달러 표시(\$), 밑줄(_), 하이픈(-), 번호 기호(#), 마침표(.) 및 at 기호(@)만 포함해야 합니다. 이름은 대소문자를 구분하기 때문에 대문자는 소문자와 다른 것으로 간주됩니다.

프로시저

- 앰퍼샌드(&)가 선행되고 세미콜론(;)으로 종료되는 기호 이름을 지정하여 문서 템플릿에 기호 참조를 포함시킬 수 있습니다. 예를 들어, 기호 `ORDER_NUMBER`는 다음과 같이 템플릿에서 지정할 수 있습니다.

```
Thank you! Your order number is &ORDER_NUMBER;.
```

- 문서 템플릿에서 `#echo` 명령을 사용하여 기호를 지정할 수도 있습니다. 명령에 기호 이름을 지정하십시오. 하지만 앰퍼샌드와 세미콜론은 사용하지 마십시오. 예를 들어, 기호 `USER_NAME`은 다음과 같이 지정할 수 있습니다.

```
Welcome to the site,  
<!--#echo var=USER_NAME-->!
```

- 템플릿에서 `#set` 명령을 사용하여 기호의 기본값을 지정할 수 있습니다. 예를 들어, 기호 `USER_NAME`을 포함시키고 이 기호의 기본값을 다음과 같이 설정할 수 있습니다.

```
<!--#set var=USER_NAME value='New User'-->  
Welcome to the site,  
<!--#echo var=USER_NAME-->!
```

또는 다음과 같이 설정할 수 있습니다.

```
<!--#set var=USER_NAME value='New User'-->
Welcome to the site, &USER_NAME;!
```

이러한 템플리트 중 하나가 USER_NAME의 기본값과 함께 사용될 때 문서는 다음과 같이 표시됩니다.

```
Welcome to the site, New User!
```

다음에 수행할 작업

문서 템플리트를 사용하여 문서를 작성하는 애플리케이션은 템플리트가 사용될 때 대체될 기호에 대한 값을 정의해야 합니다. 기호에 대해 기본값을 제공하면 이 값은 애플리케이션이 해당 기호에 대한 값을 정의하지 않는 경우에 사용됩니다. 기호의 값은 문서에 템플리트를 삽입할 때 또는 그 전에 정의해야 함에 유의하십시오. 템플리트가 삽입된 후에는 기호의 대체된 값을 변경할 수 없습니다.

임베디드 템플리트 명령

CICS 문서 핸들러는 문서 템플리트에 임베드될 수 있는 세 개의 명령을 인식합니다. 지원되는 세 개의 명령은 #set, #echo 및 #include입니다.

임베디드 템플리트 명령의 구문

임베디드 템플리트 명령은 정보 포함 명령의 구문 규칙을 따릅니다. 정보 포함 명령은 왼쪽 꺾쇠괄호, 느낌표, 하이픈, 숫자 부호(해시)로 시작되며 그 뒤에는 명령이 옵니다. 이 명령은 문자 하이픈, 하이픈, 오른쪽 꺾쇠괄호로 종료됩니다. 예를 들어, 다음과 같습니다.

```
<!--#command-->
```

정보 포함을 시작하고 종료하는 데 사용되는 문자는 코드 페이지 037에 있어야 하고 그렇지 않은 경우 이 명령을 무시합니다. 이러한 문자 순서에 대한 등가 16진수는 X'4C5A60607B' 및 X'60606E'입니다.

#echo 명령

#echo 명령은 템플리트가 문서에 삽입될 때 대체되어야 하는 기호를 식별합니다. 예를 들어, 다음 #echo 명령은 명령 위치에서 문서로 대체될 기호 ASYM을 식별합니다.

```
This is the <!--#echo var=ASYM--> symbol.
```

템플리트를 사용할 경우 #echo 명령을 포함하는 문자열은 기호에 대해 정의된 값으로 완전히 대체됩니다. 해당 이름에 기호 정의가 제공되지 않은 경우 #echo 명령은 출력 데이터에 남습니다. 기호 값은 애플리케이션이 정의하거나 템플리트에서 #set 명령을 통해 기본값으로 정의할 수 있습니다.

#set 명령

#set 명령은 기호 값을 설정하는 데 사용됩니다. 이 명령은 기호의 기본값을 설정하는 데 유용합니다. 예를 들어 다음 #set 명령은 기호 ASYM에 대해 기본값 'first'를 지정합니다.

```
<!--#set var=ASYM value='first'-->
```

#set 명령은 다른 #set 명령을 덮어쓸 수 있습니다. 동일한 기호 이름의 템플릿에 둘 이상의 #set 명령을 포함시키는 경우 마지막 명령이 사용됩니다.

#set 명령이 적용되는 기호에 DOCUMENT SET 명령을 사용하여 이미 값을 지정한 경우 템플릿에 있는 #set 명령은 무시합니다. DOCUMENT SET 명령을 사용하여 값을 지정한 기호는 다른 DOCUMENT SET 명령을 실행해야만 변경할 수 있습니다.

#set 명령은 #echo 명령과 조합하여 사용하거나 기호 이름 앞에 앤퍼샌드(&)를 배치하고 세미콜론(;)으로 종료하여 지정된 기호 참조에 적용할 수 있습니다.

#include 명령

#include 명령을 사용하면 템플릿을 다른 템플릿 내에 임베드할 수 있습니다. 최대 32 레벨까지의 임베드가 허용됩니다.

예를 들어, 다음과 같습니다.

```
<!--#include template=templatename-->
```

*templatename*은 DOCTEMPLATE 자원 정의에서 정의되는 템플릿의 이름입니다(48 바이트 이름). 템플릿 이름도 큰따옴표로 묶을 수 있습니다.

제 42 장 문서 및 문서 템플리트를 사용하여 프로그래밍

이 섹션에서는 애플리케이션에서 문서 및 문서 템플리트를 사용하는 방법을 설명합니다.

문서 작성

DOCUMENT CREATE 명령을 사용하여 비어 있는 문서 또는 데이터 포함 문서를 작성할 수 있습니다. 데이터는 문자열, 2진 데이터 블록, 문서 템플리트 또는 데이터 버퍼일 수 있습니다.

이 태스크 정보

데이터를 포함하는 문서를 작성하기 위해 EXEC CICS DOCUMENT CREATE 명령에 다음과 같은 옵션을 지정할 수 있습니다.

- 문자열을 포함합니다(TEXT 옵션).
- 2진 데이터 블록을 포함합니다(BINARY 옵션).
- 템플리트 이름으로 지정되는 문서 템플리트를 포함합니다(TEMPLATE 옵션).
- 데이터 버퍼의 내용을 포함합니다(FROM 옵션).
- 문서 템플리트에 있는 기호 또는 FROM 옵션에서 지정된 항목에 값을 지정합니다(SYMBOLLIST 옵션).

DOCUMENT CREATE 명령에는 DOCTOKEN 옵션이 있고 이 옵션은 필수이며 16 바이트 데이터 영역이 필요합니다. 문서 핸들러는 DOCTOKEN 피연산자를 사용하여 후속 호출에서 문서를 식별하는 데 사용되는 토큰을 리턴합니다.

다음 예제는 애플리케이션이 EXEC CICS DOCUMENT 명령을 사용하여 문서를 작성하는 여러 방식을 보여줍니다. Java 애플리케이션에서 CICS Java 클래스 라이브러리(JCICS)를 사용하여 문서 서비스에 액세스할 수 있습니다. **Document** 클래스는 EXEC CICS DOCUMENT 명령의 Java 구현을 제공합니다. 클래스 문서의 경우 JCICS 클래스 참조의 Javadoc을 참조하십시오.

프로시저

1. 비어 있는 문서를 작성하고 그 토큰을 리턴하려면 DOCTOKEN 옵션과 함께 EXEC CICS DOCUMENT CREATE 명령을 사용하십시오. 다음 예제는 비어 있는 문서를 작성하고 16자 변수 MYDOC로 토큰을 리턴합니다.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(MYDOC)
```

2. TEXT 옵션을 사용하여 애플리케이션이 지정하는 문자열을 포함하는 문서를 작성하십시오. 예를 들어, DOCTEXT라는 문자열 변수를 정의하고 이 변수를 문서에 추가될 텍스트의 예제임으로 초기화하면 다음 명령을 사용하여 이 문자 텍스트 문자열로 구성된 문서를 작성할 수 있습니다.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(MYDOC1)
      TEXT(DOCTEXT)
      LENGTH(53)
```

이 문자열은 변경되지 않은 문서에 추가되고 CICS는 이 문서에서 기호 대체를 수행하지 않습니다.

3. BINARY 옵션을 사용하여 2진 데이터를 포함하는 문서를 작성하십시오. 이 문서는 데이터가 전송될 때 코드 페이지 변환을 거치지 않습니다. 다음 예제는 2진 데이터로 된 데이터 영역의 내용으로 구성되는 문서를 작성합니다.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(MYDOC2)
      BINARY(DATA-AREA)
```

CICS는 이 데이터에 대해 기호 대체를 수행하지 않고 문서를 수신인에게 전송할 때 데이터가 클라이언트 코드 페이지로 변환되지 않도록 데이터에 표시합니다.

4. TEMPLATE 옵션을 사용하여 CICS에 DOCTEMPLATE 자원 정의를 사용하여 정의한 문서 템플릿을 사용하는 문서를 작성하십시오.

- TEMPLATENAME과 같은 48바이트 변수를 정의하고 이 변수를 DOCTEMPLATE 자원 정의의 TEMPLATENAME 속성에서 지정된 템플릿의 48자 이름의 값으로 초기화하십시오.
- 문서 템플릿에 기호가 포함되지 않거나 기호에 기본값을 사용하려는 경우 SYMBOLLIST 옵션 없이 DOCUMENT CREATE 명령을 사용할 수 있습니다. 예를 들어, 다음과 같습니다.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(MYDOC3)
      TEMPLATE(TEMPLATENAME)
```

문서 템플릿이 문서에 배치될 때 또는 그 전에만 기호 대체의 값을 지정할 수 있음에 유의해야 합니다. 템플릿이 삽입된 후에는 기호의 대체된 값을 변경할 수 없습니다.

- 문서 템플릿에서 기호의 값을 설정하려면 SYMBOLLIST 옵션과 함께 DOCUMENT CREATE 명령을 사용하십시오. 예를 들어, 다음과 같습니다.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(MYDOC3)
      TEMPLATE(TEMPLATENAME)
      SYMBOLLIST('ORDER_NUMBER=0012345')
      LISTLENGTH(20)
```

5. FROM 옵션을 사용하여 데이터 버퍼를 사용하는 문서를 작성하십시오. 데이터 버퍼는 문서 템플릿에 포함된 기호 참조와 같은 방식으로 대체되는 기호 참조를 포함할 수 있습니다. 예를 들어, 다음과 같습니다.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(MYDOC4)
      FROM(BUFFER)
      SYMBOLLIST('ORDER_NUMBER=0012345')
      LENGTH(LEN)
```

기호 값 정의

애플리케이션은 DOCUMENT SET 명령 또는 DOCUMENT CREATE 명령을 사용하여 문서 템플릿에서 기호의 값을 정의할 수 있습니다. 기호 값은 템플릿이 사용될 때 DOCUMENT CREATE 명령 또는 DOCUMENT INSERT 명령에 의해 대체됩니다.

이 태스크 정보

DOCUMENT SET 명령에서 SYMBOL 및 VALUE 옵션을 사용하여 개별 기호를 정의할 수 있습니다. 또는 DOCUMENT CREATE 또는 DOCUMENT SET 명령에서 SYMBOLLIST 옵션을 사용하여 단일 명령으로 여러 기호를 정의할 수 있습니다. 기본적으로 기호 목록의 기호 분리문자는 앰퍼샌드이지만 명령에서 DELIMITER 옵션을 사용하여 이를 덮어쓸 수 있습니다.

애플리케이션을 디자인할 때 다음에 유의하십시오.

- 기호의 값은 문서에 템플릿을 삽입할 때 또는 그 전에 정의해야 합니다. 템플릿이 삽입된 후에는 기호의 대체된 값을 변경할 수 없습니다.
- 문서 템플릿의 모든 기호를 정의해야 합니다. 애플리케이션 또는 #set 명령으로 문서 템플릿에 기호가 정의되지 않는 경우 해당 기호에 대한 기호 대체는 발생하지 않으며 완료된 문서는 기호 이름 또는 기호를 지정하는 #echo 명령을 포함합니다.
- DOCUMENT SET 명령을 사용하여 애플리케이션에서 제공되는 기호 정의는 #set 명령으로 기호에 제공되는 기본값을 덮어씁니다.
- DOCUMENT SET 명령을 사용하여 값을 지정한 기호는 다른 DOCUMENT SET 명령을 실행해서 변경할 수 있습니다. 문서 템플릿이 문서에 삽입된 후에 이를 수행하면 새 값이 기호 테이블에 배치되고 다음에 템플릿이 문서에 삽입될 때 사용됩니다. 변경사항은 문서에 이미 삽입된 값에는 영향을 미치지 않습니다.

839 페이지의 『기호 및 기호 목록을 지정하기 위한 규칙』에서는 기호 이름, 기호 값의 특수 문자 및 공백과 기호 목록 분리문자에 관련된 규칙을 나열합니다.

프로시저

1. 개별 기호를 정의하려면 SYMBOL 및 VALUE 옵션과 함께 DOCUMENT SET 명령을 사용하십시오. SYMBOL 옵션은 기호의 이름을 지정하고 VALUE 옵션은 해당 기호의 값을 지정합니다. 839 페이지의 『기호 및 기호 목록을 지정하기 위한 규칙』의 규칙을 따르십시오. 이 주제에서는 UNESCAPED 옵션의 영향을 설명합니다.
 - a. 문서에 문서 템플리트를 배치하는 DOCUMENT INSERT 명령을 실행하기 전에 DOCUMENT SET 명령을 실행해야 합니다.
 - b. 기호 값을 설정하기 위해 DOCUMENT SET 명령을 사용해야 하는 경우 DOCUMENT CREATE TEMPLATE 명령은 사용할 수 없습니다. 대신 비어 있는 문서를 먼저 작성한 다음 DOCUMENT SET 명령으로 기호 값을 정의한 후 DOCUMENT INSERT 명령을 사용하여 기호 참조를 포함하는 템플리트를 삽입할 수 있습니다.
2. 여러 기호 정의를 포함하는 기호 목록을 제공하려면 DOCUMENT CREATE 또는 DOCUMENT SET 명령에서 SYMBOLLIST 옵션을 사용하십시오. DOCUMENT CREATE에서 TEMPLATE 및 SYMBOLLIST를 함께 사용하면 템플리트가 문서에 추가될 때 기호 목록의 기호가 분석됩니다. DOCUMENT SET 명령을 사용하는 경우 문서에 문서 템플리트를 배치하는 DOCUMENT INSERT 명령을 실행하기 전에 DOCUMENT SET 명령을 실행해야 합니다.
 - a. 기호 값에 사용할 문자이기 때문에 기본 기호 분리문자 &(앰퍼샌드)가 적합하지 않은 경우 DELIMITER 옵션을 사용하여 대체 문자를 지정하십시오. 839 페이지의 『기호 및 기호 목록을 지정하기 위한 규칙』의 규칙을 따르십시오.
 - b. SYMBOLLIST 옵션을 사용하여 기호 목록을 포함하는 버퍼를 지정하십시오. 선택한 기호 분리문자로 분리되는 각 기호 정의를 지정하십시오. 818 페이지의 『기호 및 기호 목록』에서는 기호 목록의 예제를 보여줍니다. 839 페이지의 『기호 및 기호 목록을 지정하기 위한 규칙』의 규칙을 따르십시오. 이 주제에서는 UNESCAPED 옵션의 영향을 설명합니다.
 - c. LISTLENGTH 옵션을 사용하여 기호 목록의 길이를 지정하십시오. 사용자의 애플리케이션에 따라 기호의 값을 정의할 때마다 기호 목록의 정확한 목록 길이를 지정하는 대신 기호 목록의 고정 목록 길이를 제공하는 LISTLENGTH 옵션의 영구 값을 선택하는 것이 보다 편리하다는 것을 알 수 있습니다. 이를 수행하는 경우:
 - 선택한 고정 목록 길이는 입력 예상 기호 목록의 최대 길이를 포함할 수 있도록 충분한 길이여야 합니다.
 - 고정 목록 길이가 사용자가 입력하는 기호의 실제 길이보다 긴 경우 기호 목록의 끝에 추가 더미 기호를 포함시키십시오(예: '&END='). 이렇게 하면 목록의 마지막 기호 값에 예상치 못한 문자 또는 후미 공백이 발생하는 것을

방지할 수 있습니다. 템플리트나 문서에는 이러한 더미 기호를 포함시키지 마십시오. 후미 공백 또는 예상치 못한 문자는 더미 기호에 지정되어 문서에 표시되지 않습니다.

결과

정의하는 기호는 기호 테이블에 배치됩니다. DOCUMENT CREATE 명령에서 기호 목록을 정의하는 경우 즉시 기호 대체가 발생하고 문서 템플리트 및 지정된 기호를 사용하여 문서가 작성됩니다. DOCUMENT SET 명령을 사용하여 개별 기호 또는 기호 목록을 정의하는 경우 기호 테이블에 배치한 값은 해당 문서 템플리트를 사용하여 DOCUMENT CREATE 또는 DOCUMENT INSERT 명령을 실행할 때 기호 대체에 사용됩니다.

기호 및 기호 목록을 지정하기 위한 규칙

각 기호에는 이름과 값이 있습니다. 기호 이름을 선택하고 기호의 값을 개별적으로 또는 기호 목록에서 지정할 때 다음 규칙을 따릅니다.

기호 이름

기호의 이름은 대문자 및 소문자, 숫자 및 특수 문자 달러 표시(\$), 밑줄(_), 하이픈(-), 번호 기호(#), 마침표(.) 및 at 기호(@)만 포함해야 합니다. 이름은 대소문자를 구분하기 때문에 대문자는 소문자와 다른 것으로 간주됩니다.

문서 템플리트에 기호 참조를 포함시키기 위해 앰퍼샌드(&)가 선행되고 세미콜론(;)으로 종료되는 기호의 이름을 기호 참조로 지정할 수 있습니다. 또는 기호의 이름을 #echo 명령을 사용하여 지정할 수 있습니다. 애플리케이션에서 기호의 이름을 지정할 때 앰퍼샌드와 세미콜론은 사용하지 마십시오. 예를 들어, 템플리트에서 기호 참조 &mytitle;은 기호 목록의 기호 이름 *mytitle*에 해당됩니다.

기호 목록의 분리문자

DOCUMENT CREATE 및 DOCUMENT SETSYMBOLLIST 명령에서 SYMBOLLIST 옵션은 한 개 이상의 정의와 1바이트 분리문자로 구성되는 문자열을 지정합니다. 기본적으로 기호 분리문자는 앰퍼샌드이지만 명령에서 DELIMITER 옵션을 사용하여 이를 덮어쓸 수 있습니다. 기호 값 내에 기호 분리문자를 사용하려는 경우 특수한 처리가 필요하므로 기호 목록에서 기호 값 내에서 사용되지 않을 기호 분리문자를 선택해 보십시오. 비인쇄 문자를 사용할 수 있습니다.

허용되지 않는 DELIMITER 값이 여럿 있습니다. 허용되지 않는 값은 다음과 같습니다.

- 널(2진 X'00')
- 시프트인(2진 X'0E')

- 시프트 아웃(2진 X'0F')
- 공백(2진 X'40')
- 더하기 부호(2진 X'4E')
- 콜론(2진 X'7A')
- 등호(2진 X'7E')
- 퍼센트 부호(2진 X'6C')
- 백슬래시(2진 X'E0')

기호 값의 특수 문자

기호의 값은 모든 문자를 포함할 수 있습니다. 그러나 기호 값에 다음 문자를 포함시켜야 하는 경우 특수한 처리가 필요합니다.

- 더하기 부호(+)
- 퍼센트 부호(%)
- 등호(=)
- 기호 목록에서 기호 분리문자로 사용한 문자. 특수 처리는 기호 정의가 기호 목록에 제공될 때 이 문자에만 필요하고 DOCUMENT SET 명령을 SYMBOL 및 VALUE 옵션과 함께 사용하여 개별 기호 값을 설정할 때에는 적용되지 않습니다.

기호 값에서 이스케이프 시퀀스를 사용하여 문자(예: 특수한 의미를 가지는 문자)를 포함시킬 수 있습니다. 이스케이프 시퀀스는 퍼센트 부호, 그 뒤에 오는 두 16진수 문자(즉, 0 - 9, a - f 및 A - F)로 구성됩니다. 기호 값이 기호 테이블에 입력되면 퍼센트 부호와 그 뒤에 오는 두 개의 16진수가 두 자리로 선언된 단일 ASCII 문자의 EBCDIC 등가로 바뀝니다.

일부 유용한 조합은 표 43에 표시되어 있습니다.

표 43. 특수 기호를 나타내는 데 사용할 수 있는 이스케이프 시퀀스

문자	이스케이프 시퀀스
더하기 부호 +	%2B
퍼센트 부호 %	%25
등호 =	%3D
앰퍼샌드 &(기본 기호 분리문자)	%26

퍼센트 부호 뒤에 오는 문자가 두 개의 올바른 16진수가 아닌 경우 퍼센트 부호 및 그 뒤의 문자는 기호 목록에 표시되는 대로 기호 테이블에 입력됩니다.

이스케이프 시퀀스를 사용하지 않으려는 경우 DOCUMENT CREATE 또는 DOCUMENT SET 명령에서 UNESCAPED 옵션을 지정할 수 있습니다. 이 옵션을 지정하면 변환이 발생하지 않고 기호 값은 입력한 그대로 기호 테이블에 입력됩니다.

그러나 UNESCAPED 옵션은 기호 값 내에서 기호 분리문자로 사용했던 문자를 기호 목록에 포함시키는 것을 허용하지 않습니다. UNESCAPED 옵션을 사용하려면 기호 값 내에 사용되지 않을 기호 분리문자를 선택하십시오. 또는 DOCUMENT SET 명령에서 SYMBOL 및 VALUE 옵션을 사용하여 기호 분리문자로 사용했던 문자를 포함하는 기호 값을 지정할 수 있습니다. 이 기호 문자는 VALUE 옵션에 사용될 때 특별한 의미가 없기 때문입니다.

기호 값의 공백

기호 값에 공백 문자를 포함시키려는 경우 CICS에서는 다음 표현 중 하나를 사용할 수 있습니다.

- 공백 문자
- 16진수 20(%20)이 뒤에 오는 퍼센트 부호
- 더하기 부호

기호 값이 기호 테이블에 입력될 때 이러한 표현은 모두 공백으로 해석됩니다. 이는 더하기 부호를 사용하는 콘텐츠 유형 **application/x-www-form-urlencoded**에 대한 HTML 스펙을 확장합니다.

그러나 CICS가 기호 목록 또는 VALUE 옵션에 포함된 기호 값을 이스케이프 해제하지 못하도록 UNESCAPED 옵션을 사용하는 경우에는 공백 문자를 표시하기 위해 더하기 부호 또는 이스케이프 시퀀스 %20을 사용할 수 없습니다. 이러한 경우에는 공백 문자를 사용하여 공백을 표시해야 합니다. UNESCAPED 옵션이 사용될 때 더하기 부호는 공백으로 변환되지 않기 때문입니다.

예제: 이스케이프 시퀀스 없이 기호 정의

이 예제는 이스케이프 시퀀스를 사용하지 않고 문서 핸들러에 기호 값을 전달하는 방법을 보여줍니다. 이 목록의 기호 값에는 모두 이스케이프 처리를 거치지 않는 임베드된 더하기 부호, 퍼센트 부호 및 앰퍼샌드가 포함됩니다.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(ATOKEN)
      DELIMITER('!!')
      SYMBOLLIST('COMPANY=BLOGGS & SON!ORDER=NUTS+BOLTS')
      LISTLENGTH(37)
      UNESCAPED
```

이 예제에서 기호 COMPANY의 값은 'BLOGGS & SON'이고 기호 ORDER의 값은 'NUTS+BOLTS'입니다.

기호 분리문자로 앰퍼샌드 외의 문자를 사용하는 것은 앰퍼샌드를 'BLOGGS & SON'에서 사용할 수 있음을 의미합니다. 이 예제에서 사용된 기호 분리문자는 '!'이지만 기호 값에 표시되지 않는 비인쇄 문자를 사용하는 것이 가장 좋은 방법입니다.

UNESCAPED 옵션을 사용하면 'NUTS+BOLTS'의 더하기 부호가 공백으로 변환되지 않습니다. UNESCAPED 옵션을 사용했기 때문에 기호 값 'BLOGGS & SON'에서 공백이 필요한 위치를 표시하기 위해 더하기 부호 대신 공백 문자를 사용해야 합니다. 이는 데이터가 콘텐츠 유형 **application/x-www-form-urlencoded**에 대한 스펙을 더 이상 준수하지 않음을 의미합니다.

문서에 추가 데이터 추가

문서를 작성한 경우 한 개 이상의 DOCUMENT INSERT 명령을 실행하여 그 내용을 확장할 수 있습니다. 텍스트, 2진 데이터, 데이터 버퍼, 문서 템플릿 또는 기호 값을 삽입할 수 있습니다. 또한 문서에 책갈피를 삽입하고 삽입된 책갈피를 사용하여 나중 삽입 내용을 배치할 수 있습니다.

이 태스크 정보

DOCUMENT INSERT 명령에 다음과 같은 옵션을 지정할 수 있습니다.

- 문자열을 삽입합니다(TEXT 옵션).
- 2진 데이터 블록을 삽입합니다(BINARY 옵션).
- 템플릿 이름으로 지정되는 문서 템플릿을 삽입합니다(TEMPLATE 옵션).
- 데이터 버퍼의 내용을 삽입합니다(FROM 옵션).
- 기호 테이블에서 이름 지정된 기호의 값을 삽입합니다(SYMBOL 옵션).

기본적으로 지정하는 오브젝트는 문서의 끝에 추가됩니다. 문서 중간에 데이터를 삽입하기 위해 한 개 이상의 책갈피를 설정할 수 있습니다. 책갈피를 사용하면 문서에서 데이터 순서를 여전히 제어하는 순서대로 애플리케이션이 데이터 블록을 삽입할 수 있습니다.

책갈피는 데이터 블록 사이에 배치되는 레이블이며 데이터 블록 중간에 배치할 수 없습니다. DOCUMENT INSERT 명령을 사용하여 문서 구성 중에 책갈피를 문서에 배치할 수 있고 AT 옵션을 사용하여 후속 오브젝트를 삽입할 때 책갈피를 지정할 수 있습니다. 특수 책갈피 'TOP'은 이미 정의되어 있고 이 책갈피를 사용하여 문서의 맨 위에 데이터를 삽입할 수 있습니다.

프로시저

1. TEXT 옵션을 사용하여 애플리케이션에서 지정된 문자열을 삽입하십시오. 예를 들어, 다음과 같습니다.

```
EXEC CICS DOCUMENT INSERT
      DOCTOKEN(MYDOC)
      TEXT('Sample line 1.')
      LENGTH(15)
```

이 문자열은 변경되지 않은 문서에 추가되고 CICS는 이 문서에서 기호 대체를 수행하지 않습니다.

2. BINARY 옵션을 사용하여 2진 데이터 블록을 삽입하십시오. 이 데이터 블록은 데이터가 전송될 때 코드 페이지 변환을 거치지 않습니다. 예를 들어, 다음과 같습니다.

```
EXEC CICS DOCUMENT INSERT
      DOCTOKEN(MYDOC)
      BINARY(DATA-AREA)
```

CICS는 이 데이터에 대해 기호 대체를 수행하지 않고 문서를 수신인에게 전송할 때 데이터가 클라이언트 코드 페이지로 변환되지 않도록 데이터에 표시합니다.

3. TEMPLATE 옵션을 사용하여 문서 템플릿을 삽입하십시오. 문서 템플릿에서 기호의 값을 설정하려면 DOCUMENT INSERT 명령을 실행하기 전에 DOCUMENT SET 명령을 사용하여 개별 기호 또는 기호 목록을 지정하십시오. 837 페이지의 『기호 값 정의』에서는 이를 수행하는 방법을 설명합니다.
4. FROM 옵션을 사용하여 데이터 버퍼를 삽입하십시오. 데이터 버퍼는 문서 템플릿에 포함된 기호 참조와 같은 방식으로 대체되는 기호 참조를 포함할 수 있습니다.
5. SYMBOL 옵션을 사용하여 기호 값을 삽입하십시오. SYMBOL은 기호 테이블에 값이 설정된 올바른 기호의 이름을 지정합니다(DOCUMENT SET 또는 DOCUMENT CREATE 명령). 문서 핸들러는 기호와 연관된 값을 문서에 삽입합니다. 기호와 연관된 값을 문서에 삽입한 후에는 작성 중인 문서에서 해당 값을 변경할 수 없음에 유의하십시오. 이후에 기호에 대해 다른 값을 설정하는 경우 새 값은 다음에 기호가 문서에 삽입될 때 사용됩니다. 변경사항은 문서에 이미 삽입된 값에는 영향을 미치지 않습니다.
6. BOOKMARK 옵션을 사용하여 문서에 책갈피를 삽입하십시오. 책갈피는 문서를 구성하는 동안 배치되고 애플리케이션에 이후 단계에서 얻은 데이터를 삽입할 삽입 위치를 제공합니다. 책갈피의 이름의 길이는 16자여야 합니다. 책갈피 이름 사이에는 공백이 없어야 하고 선택한 이름의 길이가 16자 미만인 경우 오른쪽에 공백으로 채워야 합니다. 예를 들면 다음 명령 순서는 두 개의 텍스트 블록과 책갈피로 문서를 작성합니다.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(MYDOCBOOK)
      TEXT('Pre-bookmark text. ')
      LENGTH(19)

EXEC CICS DOCUMENT INSERT
      DOCTOKEN(MYDOCBOOK)
      BOOKMARK('ABookmark          ')

EXEC CICS DOCUMENT INSERT
      DOCTOKEN(MYDOCBOOK)
      TEXT('Post-bookmark text. ')
      LENGTH(20)
```


7. 다른 삽입 옵션과 함께 AT 옵션을 사용하여 오브젝트를 이전에 삽입된 책갈피 또는 특수 TOP 책갈피에 배치하십시오. 예를 들면 다음 명령은 예제 책갈피 ABookmark에 텍스트를 삽입합니다.

```
EXEC CICS DOCUMENT INSERT
      DOCTOKEN(MYDOCBOOK)
      TEXT('Inserted at a bookmark. ')
      LENGTH(25)
      AT('ABookmark          ')
```

완료된 문서는 다음과 같습니다.

Pre-bookmark text. Inserted at a bookmark. Post-bookmark text.

문서에서 데이터 바꾸기

문서에 책갈피를 배치하여 나중에 삽입을 통해 바꾸거나 삭제할 수 있는 데이터 영역을 제한할 수 있습니다. 이 기술을 통해 텍스트 또는 기타 데이터의 기본 항목을 문서에서 제공할 수 있으며 이러한 데이터는 애플리케이션이 해당 데이터를 대체할 수 있는 데이터가 없는 것으로 판단하는 경우 사용할 수 있습니다.

이 태스크 정보

문서에서 데이터의 기본 항목을 설정하고 바꾸거나 삭제하려면 다음을 수행하십시오.

프로시저

1. 문서를 작성하고 문서 시작부에 사용할 데이터를 지정하십시오. 다음 예제에서 문서는 일부 초기 텍스트를 사용하여 작성되고 그 토큰은 변수 MYDOCREP로 리턴됩니다.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(MYDOCREP)
      TEXT('Initial sample text. ')
      LENGTH(21)
```

2. DOCUMENT INSERT 명령을 사용하여 첫 번째 책갈피를 지정하십시오.

```
EXEC CICS DOCUMENT INSERT
      DOCTOKEN(MYDOCREP)
      BOOKMARK('BMark1          ')
```

3. DOCUMENT INSERT 명령을 사용하여 바꿀 텍스트 항목 또는 기타 데이터를 지정하십시오.

```
EXEC CICS DOCUMENT INSERT
      DOCTOKEN(MYDOCREP)
      TEXT('Text to be replaced. ')
      LENGTH(21)
```

4. DOCUMENT INSERT 명령을 사용하여 닫기 책갈피를 지정하십시오.

```
EXEC CICS DOCUMENT INSERT
      DOCTOKEN(MYDOCREP)
      BOOKMARK('BMark2          ')
```


5. DOCUMENT INSERT 명령을 사용하여 문서 끝에 사용할 데이터를 추가하십시오.

```
EXEC CICS DOCUMENT INSERT
      DOCTOKEN(MYDOCREP)
      TEXT('Final sample text. ')
      LENGTH(19)
```

이때 이 예제 문서의 논리 구조는 다음과 같습니다.

```
Initial sample text. <BMark1>Text to be replaced. <BMark2>Final
sample text.
```

책갈피의 이름은 문서에 표시되지 않습니다.

6. 두 책갈피 BMark1 및 BMark2 사이에 있는 텍스트를 바꾸려면 AT 및 TO 옵션과 함께 DOCUMENT INSERT 명령을 사용하십시오.

```
EXEC CICS DOCUMENT INSERT
      DOCTOKEN(ATOKEN)
      TEXT('Replacement Text. ')
      LENGTH(18)
      AT('BMark1          ')
      TO('BMark2          ')
```

이제 다음과 같은 예제 문서가 표시됩니다.

```
Initial sample text. Replacement Text. Final
sample text.
```

7. 두 책갈피 사이에 있는 텍스트를 삭제하려면 위와 같이 AT 및 TO 옵션과 함께 DOCUMENT INSERT 명령을 사용하십시오. 그러나 TEXT 또는 BINARY 옵션은 LENGTH가 0인 널(null) 문자열을 지정하는 데 사용하십시오.

문서 검색, 저장 및 재사용

애플리케이션에서 작성된 문서는 문서가 작성되는 CICS 태스크의 길이 동안만 존재합니다. 문서를 재사용하려면 애플리케이션이 사본을 검색하고 저장해야 합니다.

이 태스크 정보

이 명령 순서는 애플리케이션이 EXEC CICS DOCUMENT 명령을 사용하여 문서를 작성하고 작성한 문서를 검색하며 임시 저장영역 큐에 저장하고 동일한 또는 다른 애플리케이션에서 문서로 재사용하는 방법을 설명합니다.

Java 애플리케이션은 다른 프로그래밍 언어로 쓰여진 애플리케이션에서 작성된 문서를 검색하고 JCICS 클래스를 사용하는 문서에 대해 작업할 수 있습니다. 클래스 문서의 경우 JCICS 클래스 참조의 Javadoc을 참조하십시오.

프로시저

1. 애플리케이션에서 다음 변수를 정의하고 초기화하십시오.

- 문서 토큰을 보유하는 16바이트 필드 ATOKEN
- 검색된 문서를 보유하는 20바이트 버퍼 DOCBUF
- 검색된 데이터의 길이를 보유하는 FWORDLEN이라고 하는 전자 2진 필드
- 임시 저장영역 WRITE 명령의 길이를 보유하는 HWORDLEN이라고 하는 반자 2진 필드

2. DOCUMENT CREATE 명령을 사용하여 초기 문서를 작성하십시오.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(ATOKEN)
      TEXT('A sample document.')
      LENGTH(18)
```

애플리케이션이 검색한 문서를 보유하기 위해 필요한 버퍼 크기를 계산하는 데 도움을 주기 위해 문서의 크기를 변경하는 문서 명령(DOCUMENT CREATE 및 DOCUMENT INSERT 명령)에는 사용 가능한 DOCSIZE 옵션이 있습니다. 이 값은 RETRIEVE 명령이 실행될 때 원래의 해당 코드 페이지에서 문서의 사본을 포함하기 위해(제어 정보 포함) 필요한 최대 버퍼 크기입니다. 그러나 CHARACTERSET 옵션이 원래 EBCDIC 데이터(예: UTF-8)보다 더 많은 바이트 수를 필요로 하는 인코딩을 지정하는 경우 최대 크기는 변환된 문서를 저장할 만큼 크지 않을 수도 있습니다. 더미 버퍼와 MAXLENGTH 값 0으로 **DOCUMENT RETRIEVE**를 실행한 다음 LENGERR 조건을 처리하고 리턴된 LENGTH 값을 사용하여 버퍼를 할당하기 전에 실제 문서 길이를 판별할 수 있습니다.

3. 같은 태스크에서 **DOCUMENT RETRIEVE** 명령을 실행하여 애플리케이션의 자체 버퍼에서 문서 사본을 얻으십시오.

```
EXEC CICS DOCUMENT RETRIEVE
      DOCTOKEN(ATOKEN)
      INTO(DOCBUF)
      LENGTH(FWORDLEN)
      MAXLENGTH(20)
```

기본적으로 문서가 검색되면 애플리케이션 버퍼로 전달되는 데이터는 문서의 정확한 복제본을 재구성하는 데 필요한 제어 정보를 포함하는 양식에 저장됩니다. CICS는 문서 내용에 태그를 삽입하여 책갈피를 식별하고 코드 페이지 변환이 필요없는 블록을 구분합니다. 따라서 검색한 사본을 사용하여 작성하는 문서는 원래 문서와 같습니다. 원래 문서를 재작성하지 않아도 되는 경우 **DOCUMENT RETRIEVE** 명령을 다음과 같이 수정할 수 있습니다.

- a. 제어 정보가 없는 사본을 요청하려면 DATAONLY 옵션을 지정하십시오. 이 옵션을 사용하면 CICS가 모든 임베드된 태그를 생략합니다. 검색한 문서에 책갈피가 없고 코드 페이지 변환이 필요하지 않은 블록을 구분하는 표시가 없습니다.

- b. 사본 전체를 단일 클라이언트 코드 페이지로 변환하려면 CHARACTERSET 옵션을 지정하십시오.
4. 임시 저장영역 큐에 문서를 저장하십시오.

```
EXEC CICS WRITEQ TS
      QUEUE('AQUEUE')
      FROM(DOCBUF)
      LENGTH(HWORDLEN)
```
5. 같거나 다른 애플리케이션에서 애플리케이션의 버퍼로 저장된 데이터를 읽으십시오.

```
EXEC CICS READQ TS
      QUEUE('AQUEUE')
      INTO(DOCBUF)
      LENGTH(HWORDLEN)
```
6. FROM 옵션과 함께 **DOCUMENT CREATE** 명령을 사용하여 데이터 버퍼의 내용을 사용하는 새 문서(검색된 문서)를 작성하십시오.

```
EXEC CICS DOCUMENT CREATE
      DOCTOKEN(ATOKEN)
      FROM(DOCBUF)
      LENGTH(FWORDLEN)
```

다음에 수행할 작업

또한 DOCUMENT RETRIEVE 및 DOCUMENT INSERT 명령을 사용하여 전체 문서를 기존 문서에 삽입할 수 있습니다. 애플리케이션에서 다음 변수를 먼저 정의하고 초기화해야 합니다.

- 검색할 문서의 문서 토큰을 포함하는 16바이트 필드 RTOKEN
- 검색한 문서를 보유하기에 충분한 길이를 가지는 버퍼 DOCBUF
- 검색된 데이터의 길이를 보유하는 RETRIEVLN이라고 하는 전자 2진 필드
- 버퍼가 수신할 수 있는 데이터(예: DOCBUF의 길이)의 최대량을 보유하는 MAXLEN이라고 하는 전자 2진 필드
- 삽입 중인 문서의 문서 토큰을 포함하는 16바이트 필드 ITOKEN

다음 명령 순서는 ITOKEN으로 표시되는 다른 문서에 삽입되고 있는 RTOKEN으로 표시되는 문서를 보여줍니다.

```
EXEC CICS DOCUMENT RETRIEVE
      DOCTOKEN(RTOKEN)
      INTO(DOCBUF)
      LENGTH(RETRIEVLN)
      MAXLENGTH(MAXLEN)

EXEC CICS DOCUMENT INSERT
      DOCTOKEN(ITOKEN)
      FROM(DOCBUF)
      LENGTH(RETRIEVLN)
```

검색된 문서는 DOCUMENT INSERT 명령에서 지정된 문서 끝에 삽입되고 검색된 문서의 모든 제어 정보는 두 번째 문서에 표시됩니다. DOCUMENT INSERT 명령의 LENGTH 매개변수는 DOCUMENT RETRIEVE 명령에서 RETRIEVLN 필드로 리턴되는 값과 같아야 합니다.

문서 삭제

DOCUMENT DELETE 명령을 사용하여 트랜잭션 중에 더 이상 필요하지 않은 문서를 삭제할 수 있습니다. 명령 실행 시 문서에 할당된 저장영역은 즉시 해제됩니다. WEB CONVERSE, WEB SEND(클라이언트) 및 WEB SEND(서버) 명령의 DOCSTATUS(DOCDELETE) 옵션으로도 문서를 삭제할 수 있습니다.

이 태스크 정보

DOCUMENT DELETE, WEB CONVERSE, WEB SEND(클라이언트) 및 WEB SEND(서버) 모두는 DOCTOKEN을 사용하여 문서의 16바이트 2진 토큰을 지정합니다. EXEC CICS DOCUMENT API 명령을 사용하여 문서를 작성할 때 문서 토큰이 리턴됩니다.

DOCUMENT DELETE 명령을 사용하여 문서를 삭제하려면 다음을 수행하십시오.

1. 삭제할 문서의 DOCTOKEN을 지정하십시오. 예를 들어, 다음과 같습니다.

```
EXEC CICS DOCUMENT DELETE
      DOCTOKEN(MYDOC)
```

2. 문서가 문서 처리 도메인에서 삭제되고 저장영역은 즉시 해제됩니다. 명령에 ACTION(EVENTUAL)을 지정하는 경우 웹 도메인은 문서의 사본을 보존합니다.

WEB CONVERSE, WEB SEND(클라이언트) 및 WEB SEND(서버)를 통해 DOCSTATUS(DOCDELETE) 옵션을 지정하여 문서를 삭제할 수 있습니다. 이 옵션을 사용하여 애플리케이션은 일단 CONVERSE 또는 SEND 명령이 실행되면 문서가 더 이상 필요하지 않음을 표시할 수 있습니다. WEB SEND 명령이 완료되면 문서 처리 및 웹 도메인에서 문서가 삭제되고 저장영역은 즉시 해제됩니다.

명령에 DOCSTATUS(NODOCDELETE) 및 ACTION(EVENTUAL)을 지정하여 WEB SEND를 실행하는 경우 WEB RETRIEVE 명령을 사용하여 문서를 검색할 수 있습니다. DOCSTATUS(DOCDELETE) 옵션 또는 ACTION(IMMEDIATE) 옵션을 사용하면 웹 저장영역에서 문서를 영구적으로 제거하게 되고 제거된 문서는 검색할 수 없습니다. *CICS 애플리케이션 프로그래밍 참조서*에서는 문서 삭제 후의 문서 검색 제한사항에 대한 자세한 정보를 제공합니다.

제 13 부 이름 지정된 카운터 서버

이름 지정된 카운터 서버 이해 및 사용.

제 43 장 이름 지정된 카운터 서버

CICS는 병렬 Sysplex 환경에서 애플리케이션이 사용할 고유한 순서 번호 생성을 위한 기능을 제공합니다. 이 기능은 이름 지정된 카운터 서버에 의해 제어되는데, 숫자의 각 순서를 이름 지정된 카운터로 유지보수합니다.

순서 번호가 지정될 때마다, 대응하는 이름 지정된 카운터가 자동으로 증분됩니다. 기본적으로, 증분은 1로서 다음 요청이 순서의 다음 번호를 얻도록 합니다. EXEC CICS GET 명령을 사용하여 다음 숫자를 요청할 때 증분을 변화시킬 수 있습니다.

문서(예: 고객 주문, 송장, 발송 메모)에 대한 고유한 번호 얻기 또는 고객 파일의 고객 레코드 번호 할당을 위한 숫자 블록 얻기 같은 이 기능의 다양한 사용이 있습니다.

단일 CICS 리전에서는 고유한 숫자 할당을 제어하는 데 사용할 수 있는 다양한 방법이 있습니다. 예를 들어, CICS CWA(Common Work Area)를 사용하여 숫자를 사용하는 각 애플리케이션이 갱신하는 숫자를 저장할 수 있습니다. CWA 메소드의 문제점은 CWA가 CICS 주소 공간에 고유하며 동일한 애플리케이션을 실행 중인 다른 리전이 공유할 수 없다는 것입니다. CICS 공유 데이터 테이블을 사용하면 그런 서비스를 제공할 수 있지만, CICS 리전이 모두 동일한 MVS 이미지에 상주해야 합니다. 이름 지정된 카운터 기능은 결합 기능에서 이름 지정된 카운터를 유지보수하고 sysplex의 각 MVS 이미지에서 실행 중인 이름 지정된 카운터 서버를 통한 액세스를 제공하여 다른 메소드가 제공하는 모든 공유 어려움을 극복합니다. 이것은 병렬 Sysplex 전체의 모든 CICS 리전이 동일한 이름 지정된 카운터에 액세스할 수 있도록 합니다.

이름 지정된 카운터 서버를 사용할 때 각 정상 요청(다음 카운터 값을 지정하는)은 단일 결합 기능 액세스만 필요합니다. 이것은 이 목적을 위한 파일 사용과 비교할 때 성능에서 상당한 개선을 제공합니다. 이름 지정된 카운터 서버는 또한 둘 이상의 결합 기능 액세스가 결합 기능 데이터 테이블을 갱신하기 위해 필요하기 때문에 이름 지정된 카운터 서버는 이 측면에서 결합 기능 데이터 테이블보다 더 잘 수행합니다. 하드웨어 구성에 따라서 매초 수천 개의 이름 지정된 카운터 서버 요청을 쉽게 작성할 수 있습니다.

이 섹션에서는 다음에 대해 설명합니다.

- 852 페이지의 『이름 지정된 카운터 필드』
- 852 페이지의 『이름 지정된 카운터 풀』
- 854 페이지의 『이름 지정된 카운터 EXEC 인터페이스 사용』
- 856 페이지의 『이름 지정된 카운터 CALL 인터페이스 사용』
- 871 페이지의 『이름 지정된 카운터 복구』

이름 지정된 카운터 필드

각 이름 지정된 카운터는 카운터 이름, 현재 값, 최소값 및 최대값으로 구성됩니다.

카운터 이름

이름은 최대 16바이트까지 가능하며 문자 A - Z, 0 - 9, \$ @ # 및 _로 구성됩니다. 16바이트 미만의 이름은 후미 공백으로 채워져야 합니다.

현재 값

요청하는 애플리케이션에 지정될 다음 번호입니다.

최소값 카운터에 대한 최소 번호 및 REWIND 명령에 대한 응답으로 서버가 카운터를 다시 설정하는 번호를 지정합니다.

최대값 그 이후에는 카운터가 REWIND 명령에 의해 명시적으로(또는 WRAP 옵션에 의해 자동으로) 다시 설정되어야 하는, 카운터가 지정할 수 있는 최대 번호를 지정합니다.

모든 값은 내부적으로 8바이트(이중단어) 2진 숫자로 저장됩니다. EXEC CICS 인터페이스에서는 이들을 전자 부호 있는 2진 숫자나 이중단어 부호 없는 2진 숫자로 사용할 수 있습니다. 이것은 이중단어 명령(854 페이지의 『이름 지정된 카운터 EXEC 인터페이스 사용』 참조)을 사용하여 이름 지정된 카운터를 정의하고 명령의 부호 있는 전자 버전을 사용하여 서버로부터 요청 번호를 정의하는 경우 오버플로우 상태를 발생시킬 수 있습니다.

이름 지정된 카운터 풀

이름 지정된 카운터는 이름 지정된 카운터 풀에 저장되는데, 이 풀은 결합 기능의 목록 구조에 상주합니다. 각 풀은 해당 목록 구조가 256KB의 최소 크기를 갖고 정의되는 경우에도 수천 개의 이름 지정된 카운터를 보유할 수 있습니다.

풀에 대한 결합 기능 목록 구조를 정의한 후 풀에 대해 처음 이름 지정된 카운터 서버를 시작하여 이름 지정된 카운터 풀을 작성합니다. 풀 이름은 카운터 이름과 동일한 문자 세트의 1 - 8바이트입니다. 풀 이름은 임의의 허용되는 문자들로 구성될 수 있지만, DFHNCxxx 양식의 이름을 권장합니다.

사용자 필요에 맞게 여러 풀을 작성할 수 있습니다. 프로덕션 CICS 리전이 사용할 풀(예를 들어 DFHNCPRD로 부름) 및 테스트 및 개발 리전을 위한 다른 풀(예를 들어 DFHNCSTST 및 DFHNCDEV 같은 이름 사용)을 작성할 수 있습니다. 애플리케이션에서 논리 풀 이름을 사용하는 방법 및 이들이 런타임 시에 실제 풀 이름으로 해석되는 방법에 대한 정보는 853 페이지의 『이름 지정된 카운터 옵션 테이블』의 내용을 참조하십시오.

이름 지정된 카운터 서버에 대한 목록 구조 정의 및 이름 지정된 카운터 서버 시작은 *CICS System Definition Guide*에서 설명됩니다.

이름 지정된 카운터 옵션 테이블

POOL(*name*) 매개변수는 모든 EXEC CICS COUNTER 및 DCOUNTER 명령에서 옵션입니다. POOL 매개변수를 지정하는 경우, 실제 또는 논리 풀 이름을 참조할 수 있습니다. POOL 매개변수를 지정하거나 생략하든지 간에, CICS는 링크 목록에서 로드되는 이름 지정된 카운터 옵션 테이블에 대한 참조에 의해 실제 풀 이름을 해석합니다.

EXEC CICS COUNTER 및 DCOUNTER 명령에 대한 자세한 정보는 854 페이지의 『이름 지정된 카운터 EXEC 인터페이스 사용』의 내용을 참조하십시오.

이름 지정된 카운터 옵션 테이블인 DFHNCOPT는 이름 지정된 카운터 API 명령에 의해 참조되는 실제 풀 이름을 판별하기 위한 여러 가지 방법을 제공하는데, 이들 모두 구성에서 공유 데이터 세트, CSD 및 SYSIN 설정에서 설명됩니다. 이것은 또한 사용자 자신의 옵션 테이블을 작성하는 데 사용할 수 있는 DFHNCO 매크로를 설명합니다.

기본 옵션 테이블의 POOLSEL 매개변수는 API의 POOL(*name*) 옵션과 함께 작업합니다. 기본 옵션 테이블은 소스 및 오브젝트 양식으로 제공됩니다. 사전 생성된 버전은 *hlq*.SDFHLINK에 있으며, *hlq*.SDFHSAMP 라이브러리(여기에서 *hlq*는 CICS 설치 시에 설정되는 라이브러리 이름에 대한 상위 레벨 규정자를 나타냄)에서 제공되는 소스 버전이 다음 항목을 포함합니다.

```
DFHNCO  POOLSEL=DFHNC*,POOL=YES
DFHNCO  POOL=
END      DFHNCOPT
```

기본 옵션 테이블 항목은 다음과 같이 작업합니다.

POOLSEL=DFHNC*

이 풀 선택사항 매개변수는 문자 DFHNC로 시작하는 일반 논리 풀 이름을 정의합니다. 임의의 이름 지정된 카운터 API 요청이 이 일반 이름과 일치하는 풀 이름을 지정하는 경우, 풀 이름은 DFHNCO 항목의 POOL= 피연산자에 의해 판별됩니다. 이것이 기본 테이블에서는 POOL=YES이기 때문에 API 명령의 POOL(*name*) 옵션에서 전달되는 이름이 실제 이름으로 사용됩니다. 따라서, 기본 옵션 테이블은 DFHNC로 시작하는 모든 논리 풀 이름이 실제 풀 이름이라고 지정합니다.

POOL=

기본 테이블의 이 항목은 '기본' 항목입니다. POOLSEL 매개변수가 지정되지 않기 때문에 POOLSEL=*로 기본 설정되는데, 이것은 보다 명시적 일치를 찾지 못하는 POOL 매개변수의 모든 값과 일치하도록 선택됨을 의미합니다. 따라서 다음에 해당하는 임의의 이름 지정된 카운터 API 요청

- DFHNC 이외의 다른 값으로 시작하는 POOL 값 지정, 또는
- POOL 이름 매개변수를 함께 생략

은 기본 풀(이름 피연산자가 생략된 POOL= 옵션 테이블 매개변수로 표시됨)에 맵핑됩니다.

NCPLDFT 시스템 초기화 매개변수를 지정하여 CICS 리전이 사용할 기본 풀 이름을 지정할 수 있습니다. NCPLDFT가 생략되면 풀 이름의 기본값은 DFHNC001입니다.

다음 경우에 사용자 자신의 옵션 테이블을 작성할 필요가 없으며 이름 지정된 카운터 API 명령이 POOL 옵션을 지정할 필요가 없음을 알 수 있습니다.

- DFHNCxxx 양식의 풀 이름을 사용하거나,
- CICS 리전이 NCPLDFT 시스템 초기화 매개변수에 의해 정의될 수 있는 하나의 풀만 사용합니다.

참고:

1. DFHNCOPT 이름 지정된 카운터 옵션 테이블에 접미부가 붙지 않습니다. CICS 리전은 MVS 링크 목록에서 발견되는 첫 번째 테이블을 로드합니다.
2. 사용자 애플리케이션이 사용하는 각 이름 지정된 카운터 풀에 대해 CICS 리전과 동일한 MVS 이미지에서 실행 중인 하나의 이름 지정된 카운터 서버가 있어야 합니다.

이름 지정된 카운터 EXEC 인터페이스 사용

모든 이름 지정된 카운터 값은 내부적으로 이중단어 비부호화 2진 숫자로서 보유되지만, CICS API는 명령의 전자(COUNTER) 및 이중단어(DCOUNTER) 세트를 둘 다 제공하는데, 이들을 혼합해서는 안됩니다.

EXEC CICS 명령을 사용하여 이름 지정된 카운터에 대해 다음 조작을 수행할 수 있습니다.

DEFINE

최소 및 최대값을 설정하고 카운터가 시작할 현재 번호를 지정하여 새로운 이름 지정된 카운터를 정의합니다.

DELETE

이름 지정된 카운터 풀에서 이름 지정된 카운터를 삭제합니다.

GET

최대 숫자가 아직 할당되지 않은 경우, 이름 지정된 카운터에서 현재 숫자를 얻습니다.

WRAP 옵션 사용: 최대 숫자가 이전 요청에 할당된 경우, 카운터는 카운터 한계 조건에 있으며 WRAP 옵션을 지정하지 않는 한 요청은 실패합니다. 이 옵션은 카

운터 한계 조건에 있는 카운터가 자동으로 정의된 최소값으로 다시 설정되도록 지정합니다. 다시 설정 후, 최소값이 현재 숫자로 리턴되고, 카운터는 갱신되어 다음 요청에 대해 준비합니다.

INCREMENT 옵션 사용: 기본적으로 이름 지정된 카운터는 서버가 GET 요청에 현재 숫자를 지정한 후 1의 증분씩 갱신됩니다. 한 번에 둘 이상의 숫자를 원하는 경우, 현재 숫자로부터 숫자 블록을 효과적으로 예약하는 INCREMENT 옵션을 지정할 수 있습니다. 예를 들어, INCREMENT(50)을 지정하면 서버는 100 025를 리턴합니다.

- 애플리케이션은 100 025부터 100 074까지 사용할 수 있음
- 현재 숫자 (100 025)를 50씩 갱신하면 현재 숫자는 다음 요청에 대한 준비로 100 075입니다.

이 예제는 INCREMENT(50) 옵션으로 현재 값을 갱신해도 최대 값을 1이상 초과하지 않는다고 가정합니다. 현재 값과 최대 값 더하기 1 사이의 숫자 범위가 지정된 증분보다 작은 경우, REDUCE 옵션도 지정하지 않으면 요청은 실패합니다.

REDUCE 옵션 사용: 숫자의 남은 범위가 너무 작아서 INCREMENT 값을 충족할 수 없기 때문에(현재 값이 최대 값에 너무 가까움) 요청이 실패하지 않도록 하려면 REDUCE 옵션을 지정하십시오. REDUCE 옵션을 사용하면 서버는 카운터를 카운터 한계 조건에 두면서 모든 나머지 숫자를 지정할 수 있도록 증분을 자동으로 조정합니다.

WRAP 및 REDUCE 옵션 둘 다 사용: 두 옵션을 모두 지정하는 경우, 카운터의 상태에 따라서 하나만 유효합니다.

- 서버가 GET 요청을 받을 때 카운터가 이미 한계에 있는 경우, REDUCE 옵션은 효과가 없고 WRAP 옵션을 따릅니다.
- 서버가 GET 요청을 받을 때 카운터가 한계에 완전히 도달하지는 않지만 나머지 범위가 증분에 대해 너무 작은 경우 REDUCE 옵션을 따르고 WRAP 옵션은 효과가 없습니다.

COMPAREMIN 및 COMPAREMAX 옵션 사용: 이들 옵션을 사용하여 이름 지정된 카운터 GET(및 UPDATE) 조작을 지정된 범위 내에 있거나 지정된 비교 값 중 하나보다 크거나 작은 현재 값에 대해 조건부로 만들 수 있습니다.

QUERY

이름 지정된 카운터를 조회하여 현재, 최소, 최대 값을 얻습니다. 원자적인 방식으로 둘 이상의 이름 지정된 카운터 명령을 사용할 수 없으며, sysplex의 어딘가에 있는 다른 태스크에 의해 변경되지 않은 QUERY 명령에서 리턴된 정보에 의존할 수 없음을 주의하십시오. 이름 지정된 카운터 CALL 인터페이스를 사용하는 일괄 처리 애플리케이션이 이름 지정된 카운터에 액세스할 수 없기 때문에 CICS sysplex 전반의 ENQ 기능도 카운터를 잠글 수 없습니다. 현재 범위 내에 있거나 특정 숫

자보다 크거나 작은 현재 값에 따라 조작을 조건부로 만들려는 경우, 요청에서 COMPAREMIN 및 COMPAREMAX 매개변수를 사용하십시오.

REWIND

카운터 한계 조건에 있는 이름 지정된 카운터를 그의 정의된 최소 값으로 다시 되돌립니다.

UPDATE

이름 지정된 카운터의 현재 값을 새로운 현재 값으로 갱신합니다. 예를 들어, 현재 값을 데이터베이스에 있는 다음 사용 가능 키로 설정할 수 있습니다. GET 명령처럼, 이것은 COMPAREMIN 및 COMPAREMAX 값을 지정하여 조건부로 만들 수 있습니다.

이름 지정된 카운터 CALL 인터페이스 사용

CICS 이름 지정된 카운터 API 외에, CICS는 동일한 이름 지정된 카운터에 액세스하기 위해 일괄처리 애플리케이션에서 사용할 수 있는 호출 인터페이스를 제공합니다. 이것은 CICS와 일괄처리 프로그램을 둘 다 사용하는 애플리케이션이 있고 둘 다 지정된 범위의 고유한 숫자를 얻기 위해 동일한 이름 지정된 카운터에 액세스해야 하는 경우에 중요할 수 있습니다. 호출 인터페이스는 CICS 서비스에 종속되지 않으므로, CICS의 임의의 릴리스에서 실행하는 애플리케이션에서도 사용할 수 있습니다.

이름 지정된 카운터 호출 인터페이스는 CICS 명령 레벨 API를 사용하지 않으므로 시스템 초기화 매개변수 CMDPROT=YES는 효력을 갖지 않습니다. 인터페이스가 사용자 키에서 실행 중인 CICS 애플리케이션으로부터 호출되는 경우, 요청을 처리하는 동안 CICS 키로 전환하지만 CICS가 프로그램이 출력 매개변수 필드에 대한 쓰기 액세스 권한을 갖는지 검증하려고 시도하지 않습니다.

특정 풀을 다루는 애플리케이션 리전에 의한 첫 번째 요청이 자동으로 해당 풀에 대한 서버로의 연결을 구축합니다. 이 연결은 현재 MVS TCB(CICS의 경우 QR(quasi-reentrant) TCB임)와 연관되며 일반적으로 TCB가 작업 종료 시에 종료할 때까지 지속합니다. 이 연결은 연결이 설정된 TCB에서만 사용할 수 있습니다. 다른 TCB로부터 발행된 요청은 서버에 대한 별도의 연결을 구축합니다.

참고: 이름 지정된 카운터 서버 인터페이스는 내부적으로 MVS 이름/토큰 서비스를 사용합니다. 이것의 결과는 이름 지정된 카운터 인터페이스를 사용하는 작업이 MVS 체크포인트/재시작 서비스(APAR OW06685에서 설명함)를 사용할 수 없다는 점입니다.

애플리케이션 고려사항

이름 지정된 카운터 호출 가능 인터페이스를 사용하려면 애플리케이션에서 다음을 고려해야 합니다.

이 태스크 정보

1. 애플리케이션에 애플리케이션 언어의 매개변수 목록 정의를 정의하는 적합한 카피북이 포함되는지 확인하십시오. 카피북은 함수 코드와 리턴 코드의 기호 상수를 정의하고 또한 고급 언어의 호출 가능한 시작점을 정의합니다. 카피북은 DFHNCxxx 양식이고, 여기서 xxx는 다음과 같이 프로그래밍 언어를 나타냅니다.

ASM or **EQU** for Assembler
C for C/C++
COB for COBOL
PLI for PL/I

2. 애플리케이션이 호출 가능한 인터페이스 연계 루틴, DFHNCTR을 사용하여 링크 편집되는지 확인하십시오.
3. 명명된 카운터 서버 인터페이스 모듈 DFHNCIF 및 옵션 테이블 DFHNCOPT가 CICS 리전에 사용 가능한지 확인하십시오. 즉, 이러한 오브젝트는 STEPLIB 라이브러리, 링크 목록 라이브러리 또는 LPA에 있어야 합니다. 사용자 키에서 실행되는 CICS 애플리케이션을 지원하려면 DFHNCIF는 APF 권한이 있는 라이브러리에서 로드되어야 합니다. 기본 옵션 테이블 및 명명된 카운터 서버 인터페이스 모듈은 CICSTS53.CICS.SDFHLINK에 제공됩니다.

CICS는 지원되는 모든 언어의 카피북을 제공합니다.

어셈블러

표준 어셈블러 이름이 지정된 카운터 인터페이스 정의는 카피북 DFHNCASM에 제공됩니다. 이를 COPY DFHNCASM을 사용하여 상수 CSECT 영역 내에 포함하십시오. 기호 값은 NC_name DC F'nnn' 양식으로 정적 전자 상수로서 정의됩니다. 예를 들어, 다음과 같습니다.

```
NC_BROWSE_NEXT DC F'7'
```

대체 정의 세트는 카피북 DFHNCEQU에 기호와 동일한 값으로서 제공됩니다. 이러한 기호는 정적 상수와의 충돌을 피하기 위해 모두 NC_EQU_name 양식으로 되어 있습니다. 이러한 동일한 값은 함수 코드나 리턴 코드 비교에는 사용되지 않고, 예를 들면, 함수 코드 NC_ASSIGN을 =A(NC_EQU_ASSIGN)에 대한 참조로 대체할 수 있도록 주소 상수 값으로서 사용되어야 합니다.

이름 지정된 카운터 인터페이스에 대한 호출의 어셈블러 버전의 구문은 다음과 같습니다.

```
CALL DFHNCTR,(function,return_code,pool_selector,counter_name,      X
                  value_length,current_value,minimum_value,maximum_value,  X
                  counter_options,update_value,compare_min,compare_max),VL
```

CALL 매크로는 다음 예제에 표시된 대로 목록 표시의 끝을 설정하기 위해 VL 옵션을 지정해야 합니다.

```
CALL DFHNCTR,(NC_ASSIGN,RC,POOL,NAME,CTRLLEN,CTR),VL
```


C/C++

C/C++을 위한 이름 지정된 카운터 인터페이스 정의는 헤더 파일 DFHNCC에 제공됩니다. 기호 상수 이름은 대문자입니다. 함수 이름은 소문자로 된 dfhnctr입니다.

COBOL

COBOL의 이름 지정된 카운터 인터페이스 정의는 카피북 DFHNCCOB에 제공됩니다.

CICS가 지원하는 COBOL의 이전 버전은 이름에 밑줄을 허용하지 않습니다. 그러므로 카피북 DFHNCCOB에 제공된 기호 이름은 밑줄 대신에 하이픈을 사용합니다(예: NC-ASSIGN 및 NC-COUNTER-AT-LIMIT).

RETURN-CODE 특수 레지스터는 각 호출에 의해 설정됨을 유의하십시오. 이는 프로그램이 종료되기 전에 명시적으로 다시 설정하지 않는 한 프로그램 전체 리턴 코드에 영향을 미칩니다.

PL/I

PL/I을 위한 이름 지정된 카운터 인터페이스 정의는 포함 파일 DFHNCPLI에 제공됩니다.

구문

이 다이어그램은 이름 지정된 카운터 DFHNCTR 호출의 구문을 보여줍니다.

:

```
CALL DFHNCTR(function,return_code,pool_selector,counter_name,
             value_length,current_value,minimum_value,maximum_value,
             counter_options,update_value,compare_min,compare_max);
```

그림 135. DFHNCTR 호출 구문-PL/I 설명

참고:

- 이름 지정된 카운터를 가리키는 모든 함수는 최소한 처음 네 개의 매개변수를 필요로 하지만 나머지 매개변수는 선택적이고 뒤따르는 사용하지 않는 매개변수는 생략할 수 있습니다.

임베디드된 선택적 매개변수를 사용하지 않으려면 기본값을 지정하거나 매개변수 목록에 생략된 매개변수의 널(null) 주소가 포함되는지 확인하십시오. 선택적 매개변수를 생략하는 호출의 예제는 866 페이지의 『널(null) 매개변수를 갖는 DFHNCTR 호출 예제』의 내용을 참조하십시오.

- NC_FINISH 함수에는 처음 세 개의 매개변수만을 필요로 합니다.

function

다음 기호 상수 중 하나를 사용하여 32비트 정수로서 수행될 함수를 지정합니다.

NC_CREATE

초기값, 범위 한계 및 *current_value*, *minimum_value*, *maximum_value*, *update_value* 및 *counter_options* 매개변수에 지정된 기본 옵션을 사용하여 이름 지정된 새로운 카운터를 작성하십시오.

선택적 값 매개변수를 생략하면 생략된 값의 기본값을 사용하여 새로운 이름 지정된 카운터가 작성됩니다. 예를 들어, 모든 선택적 매개변수를 생략하면 카운터는 초기값 0, 최소값 0 및 높은 값의 최대값으로 작성됩니다(이 중 단어 필드는 X'FF'로 채워짐).

NC_ASSIGN

이름 지정된 카운터의 현재 값을 지정한 다음 다음 요청을 준비하기 위해 이를 증가시키십시오. 지정된 수가 카운터에 지정된 최대 수와 같으면 이는 최종적으로 최대값보다 큰 값 1로 증가됩니다. 이로 인해 이름 지정된 카운터에 대한 후속 NC_ASSIGN 요청은 카운터가 NC_REWIND 함수를 사용하여 재설정되거나 NC_WRAP 카운터 옵션(*counter_options* 매개변수 참조)에 의해 자동으로 리와인드될 때까지 실패합니다.

(NC_COUNTER_AT_LIMIT 사용)

이 작업에는 *compare_min* 및 *compare_max* 매개변수를 사용하여 이름 지정된 카운터의 현재 값에 대한 조건부 테스트를 포함할 수 있습니다.

서버는 호출 매개변수 목록에 이러한 필드를 지정하고 요청이 성공적이면 최소 및 최대값을 리턴합니다.

NC_ASSIGN 요청에서 *counter_options* 매개변수를 사용하여 NC_CREATE 요청이 설정한 카운터 옵션을 덮어쓸 수 있습니다.

update_value 매개변수를 사용하여 이름 지정된 카운터(기본 증분은 1)에 대한 이 요청에만 사용될 증분을 지정할 수 있습니다. 이를 통해 단일 요청으로 여러 수의 범위를 구할 수 있습니다. 자세한 정보는 *update_value* 매개변수의 설명을 참조하십시오.

이름 지정된 카운터는 현재 값이 지정된 후 *update_value*만큼 증가됨을 유의하십시오. 예를 들어, 다음과 같습니다.

현재 값이 109이면

*update_value*는 25를 지정합니다.

이름 지정된 카운터 서버는 109를 리턴하고 현재 값을 다음 NC_ASSIGN 요청을 위해 준비된 134로 설정하여 109 - 133 범위에서 숫자를 효율적으로 지정합니다. 이 증분은 0 사이의 임의의 값일 수 있고 한계는 이름 지정된 카운터에 설정된 최소값과 최대값으로 판별됩니다. 따라서 증분 한계

는 ((*maximum_value* 플러스 1) 마이너스 *minimum_value*)입니다. 0의 증분은 NC_ASSIGN이 비교 옵션을 제외하고 NC_INQUIRE와 똑같이 작동하게 만듭니다.

증분이 1보다 크고 이름 지정된 카운터가 최대 한계 근처이면 서버는 지정된 증분의 전체 값만큼 현재 수를 증분시킬 수 없습니다. 이 상황은 카운터를 증분하면 최대값 더하기 1을 초과하는 경우에 발생합니다. 이 상황에서 *counter_options* NC_NOREDUCE | NC_REDUCE 및 NC_NOWRAP | NC_WRAP를 통해 이름 지정된 카운터 서버가 취할 조치를 제어하십시오. 이러한 옵션의 운영에 대한 자세한 정보는 *counter_options* 매개변수를 참조하십시오.

NC_BROWSE_FIRST

지정된 이름보다 크거나 같은 이름이 있는 첫 번째 이름 지정된 카운터의 세부사항을 리턴하고 이에 따라 *counter_name* 필드를 갱신하십시오.

NC_BROWSE_NEXT

지정된 이름보다 큰 이름이 있는 다음 번 이름 지정된 카운터의 세부사항을 리턴하고 *counter_name* 필드를 이에 따라 갱신하십시오.

NC_DELETE

지정된 이름 지정된 카운터를 삭제하십시오. 서버는 매개변수 목록에서 이러한 필드를 지정하고 요청이 성공적이면 *current_value*, *minimum_value*, *maximum_value* 및 *counter_options*을 리턴합니다.

NC_FINISH

현재 MVS 태스크(TCB)와 지정된 풀의 이름 지정된 카운터 서버 간의 연결을 종료합니다. 동일 풀에 추가로 요청이 작성되면 새 연결이 설정됩니다.

이 기능은 특정 카운터에는 적용되지 않으므로 필요한 유일한 매개변수는 함수, 리턴 코드 및 풀 이름입니다.

이 함수는 연결을 종료해야 하는 특별한 이유가 있는 경우에만 사용하십시오(예: 서버의 시스템 종료를 허용할 때).

NC_INQUIRE

이름 지정된 카운터의 세부사항(*current_value*, *minimum_value*, *maximum_value* 및 *counter_options*)을 수정하지 않고 리턴합니다. 현재 값이 다음 번 NC_ASSIGN 호출 시에 리턴될 값입니다. 이름 지정된 카운터의 최대값이 이미 지정된 경우에는 서버는 최대값보다 큰 현재 값 1을 리턴합니다.

NC_REWIND

이름 지정된 카운터를 최소값으로 재설정합니다. 이 함수는 최대값이 허용하는 마지막 수가 지정되었을 때에만 유효하므로 카운터를

NC_COUNTER_AT_LIMIT 조건으로 남겨둡니다. NC_ASSIGN 호출이 서버가 이름 지정된 카운터의 마지막 수를 지정하게 하려면 NC_REWIND 함수를 사용하여 카운터를 재설정하십시오.

이 작업에는 *compare_min* 및 *compare_max* 매개변수를 사용하여 이름 지정된 카운터의 현재 값에 대한 조건 테스트를 포함할 수 있습니다.

매개변수 목록에서 이러한 필드를 지정하고 요청이 성공적이면 서버는 새 현재 값, 최소값 및 최대값을 리턴합니다.

옵션 매개변수 또는 *update_value* 매개변수가 이름 지정된 카운터가 한계에 도달했기 때문에 실패했던 NC_ASSIGN 요청에 지정된 경우에는 원래 NC_ASSIGN이 여전히 실패하는지 여부를 확인할 수 있도록 NC_REWIND 요청에도 동일한 매개변수 값을 지정해야 합니다. NC_REWIND 요청은 해당하는 NC_ASSIGN 요청이 성공할 때마다 리턴 코드 102 (NC_COUNTER_NOT_AT_LIMIT)로 비활성화됩니다.

NC_WRAP 옵션이 적용되거나 *update_value* 매개변수가 0이면 NC_REWIND는 NC_ASSIGN이 항상 이러한 조건으로 성공하기 때문에 비활성화됩니다. NC_WRAP 옵션에 대한 자세한 정보는 *counter_options* 매개변수를 참조하십시오.

NC_UPDATE

이름 지정된 카운터를 새 값으로 설정합니다. 이 작업에는 *compare_min* 및 *compare_max* 매개변수를 사용하여 이름 지정된 카운터의 현재 값에 대한 조건부 테스트를 포함할 수 있습니다.

update_value 매개변수에 새 값을 지정합니다. 새 값을 지정하지 않으면 이름 지정된 카운터는 변경되지 않고 남아 있습니다.

이 함수를 사용하여 유효한 *counter_options* 덮어쓰기 매개변수(또는 널(null) 주소)를 지정할 수 있지만 카운터 옵션은 아무런 효과가 없습니다. 널(null) 또는 NC_NONE을 *counter_options* 매개변수로 지정하십시오.

return_code

리턴 코드를 수신하기 위해 32비트 정수 필드를 지정합니다. 동일한 정보가 레지스터 15에도 리턴되고 COBOL 호출자를 위해 RETURN-CODE 특수 레지스터에 저장됩니다.

각 리턴 코드에는 해당하는 기호 상수가 있습니다. 이에 대한 세부사항은 867 페이지의 『리턴 코드』의 내용을 참조하십시오.

pool_selector

이름 지정된 카운터가 상주하는 풀을 식별하는 데 사용하는 8문자 풀 선택 매개변수를 지정합니다.

이 매개변수는 선택적입니다. 풀 선택 매개변수를 생략하면 8개 공백의 문자열(X'40') 문자가 가정됩니다.

*pool_selector*에 허용 가능한 문자는 A-Z 0-9 \$ @ # _이지만 첫 번째 문자는 숫자이거나 밑줄일 수 없습니다. 매개변수는 필요에 따라 8개의 문자를 후미의 공백으로 채워야 합니다. 옵션 테이블이 매개변수를 유효한 비공백 이름 지정된 카운터 이름으로 맵핑하는 경우 매개변수는 현재 리전의 기본 풀을 사용하기 위해 모두 공백일 수 있습니다.

사용 중인 이름 지정된 카운터 옵션 테이블에 따라서 풀 선택자 매개변수를 실제 풀 이름으로서나 옵션 테이블을 통해 실제 풀 이름으로 맵핑되는 논리적 풀 이름으로서 사용할 수 있습니다. 기본 옵션 테이블은 다음을 가정합니다.

- DFHNC(테이블 항목을 POOLSEL=DFHNC*와 일치시킴)로 시작하는 풀 선택 매개변수는 실제 풀 이름입니다.
- 기타 풀 선택 매개변수(모두 공백 포함)는 기본 풀 이름으로 맵핑됩니다.

참고: 호출 인터페이스를 위한 기본 풀 이름은 DFHNC001입니다. EXEC CICS API의 기본 풀 이름은 NCPLDFT 시스템 초기화 매개변수가 정의합니다.

DFHNCOPT 옵션 테이블에서 풀 선택 매개변수에 대한 정보는 853 페이지의 『이름 지정된 카운터 옵션 테이블』의 내용을 참조하십시오.

counter_name

필요에 따라 후미 공백으로 채워지는 이름 지정된 카운터의 이름을 포함하는 16바이트 필드를 지정합니다.

이름에 허용 가능한 문자는 A-Z 0-9 \$ @ # _이지만 첫 번째 문자는 숫자나 밑줄일 수 없습니다. 매개변수는 필요에 따라 16개의 문자를 후미의 공백으로 채워야 합니다.

다른 애플리케이션과의 충돌을 피하기 위해 최대 8바이트의 공통 접두부가 있는 이름을 사용하는 것이 좋습니다. CICS에 의해 내부적으로 사용되는 모든 이름 지정된 카운터에는 DFH로 시작하는 이름이 있습니다.

NC_BROWSE_FIRST 및 NC_BROWSE_NEXT 함수의 경우 실제 이름은 이 필드에 리턴되며 이는 이 용도를 위한 변수여야 합니다. 다른 모든 함수의 경우 이는 상수일 수 있습니다.

value_length

이름 지정된 각 카운터 값 필드의 길이를 표시하는 32비트 정수 필드를 지정합니다. 값은 부호 없는 형식(높은 순서 비트가 값의 일부임)이거나 양수의 부호 있는 형식(높은 순서 비트가 0 부호 비트용으로 예약됨)일 수 있습니다. 부호 없는 형식을 사용하려면 길이를 8 - 64비트에 해당하는 1 - 8 범위의 바이트로 지정하십시오. 부호 있는 형식으로 된 값을 사용하려면 길이를 7 - 63비트에 해당하는 -1 - -8 범위의 음수로서 지정하십시오. 이름 지정된 카운터 EXEC 인터페이스와의 호환성을 위해서 이는 전자 부호 있는 2진 값(COUNTER)으로 처리되는 카운터의 경

우 -4로 설정되어야 하고 이중 단어 부호 없는 2진 값(DCOUNTER)으로 처리되는 카운터의 경우 8로 설정되어야 합니다.

호출에 값이 없는 매개변수가 사용되는 경우에는 값 길이를 포함하여 모든 후미의 사용되지 않는 매개변수를 완전히 생략하거나 *value_length*를 0으로 지정할 수 있습니다.

입력값이 8바이트보다 짧으면 높은 순서 0 바이트를 사용하여 내부적으로 사용되는 전체 8바이트로 확장됩니다. 출력값이 짧은 값 필드에 리턴되면 낮은 순서 바이트의 지정된 수가 리턴되어 높은 순서 바이트를 무시합니다. 그러나 값이 너무 커서 필드 내에 정확하게 표현할 수 없는 경우에는 866 페이지의 『결과 오버플로우 확인』에 설명된 대로 경고 리턴 코드가 설정될 수도 있습니다.

current_value

다음에 사용할 변수를 지정합니다.

- 이름 지정된 카운터의 초기 순서 번호 설정
- 이름 지정된 카운터에서 현재 순서 번호 수신.

NC_CREATE 함수의 경우 이 매개변수는 입력(발신인) 필드이고 상수로서 정의될 수 있습니다. 기본값은 낮은 값(2진 0)입니다. 값은 다음 최소값과 최대값으로 지정되는 범위 내에 있거나 최대값보다 1이 클 수 있습니다. 이 경우 카운터는 사용되기 전에 NC_REWIND 함수를 사용하여 재설정되어야 합니다. 이 필드에는 부호 확인이 이뤄지지 않지만 부호 비트 세트가 있는 값은 일반적으로 카운터 한계와 일치하지 않으므로 서버에 의해 거부됩니다. 모두 부호가 있는 숫자로 구성된 범위가 있는 카운터의 경우 한계값에 있는 카운터에는 부호 비트 세트가 있고 이는 올바른 입력값으로 사용될 수 있습니다.

다른 모든 카운터 함수의 경우 이 매개변수는 출력(수신인) 필드이고 변수로서 정의되어야 합니다.

minimum_value

다음에 사용할 변수를 지정합니다.

- 이름 지정된 카운터의 최소값 설정
- 이름 지정된 카운터로부터 지정된 최소값을 수신.

NC_CREATE 함수의 경우 이 매개변수는 입력(발신인) 필드이고 상수로서 정의될 수 있습니다. 기본값은 낮은 값(2진 0)입니다.

다른 모든 함수의 경우 이 매개변수는 출력(수신인) 필드이고 변수로서 정의되어야 합니다.

maximum_value

다음에 사용할 변수를 지정합니다.

- 이름 지정된 카운터의 최대값 설정
- 이름 지정된 카운터로부터 지정된 최대값을 수신.

NC_CREATE 함수의 경우 이 매개변수는 입력(발신인) 필드이고 상수로서 정의될 수 있습니다. 0이 아닌 *value_length* 매개변수를 지정했지만 *maximum_value*를 생략하는 경우에는 *maximum_value*는 지정된 길이에 대해 높은 값(또는 부호가 있는 변수의 경우 가장 큰 양의 값)으로 기본적으로 높은 값으로 설정됩니다. *value_length* 매개변수가 생략되거나 0으로 지정된 경우에는 *maximum_value*는 기본적으로 높은 값의 8바이트로 설정됩니다. 그러나 최소값이 모두 낮은 값이고 최대값이 높은 값의 8바이트이면 최대값은 서버가 몇몇 예약된 값을 내부용으로 사용할 수 있도록 허용하기 위해 감소됩니다.

다른 모든 함수의 경우 이 매개변수는 출력(수신인) 필드이고 변수로서 정의되어야 합니다.

counter_options

래핑과 증분 감소를 제어하는 이름 지정된 카운터 옵션을 표시하기 위해 선택적 전자 필드를 지정합니다. 올바른 옵션은 기호 값 NC_WRAP 또는 NC_NOWRAP 및 NC_REDUCE 또는 NC_NOREDUCE에 의해 표시됩니다. 기본 옵션은 NC_NOWRAP 및 NC_NOREDUCE입니다.

NC_NOWRAP

서버는 NC_COUNTER_AT_LIMIT 조건으로 실패하는 NC_ASSIGN 요청에 대한 응답으로 이름 지정된 카운터를 다시 최소값으로 자동으로 되감기하지 않습니다. NC_NOWRAP을 강제 실행하고, NC_COUNTER_AT_LIMIT 조건에 이름 지정된 카운터가 있으면 NC_ASSIGN 함수는 카운터가 NC_REWIND 요청에 의해 재설정되거나 카운터 옵션이 NC_WRAP로 재설정될 때까지 작동하지 않습니다.

NC_WRAP

서버는 NC_COUNTER_AT_LIMIT 조건에 있는 카운터에 대한 NC_ASSIGN 요청에 대한 응답으로 NC_REWIND를 자동으로 수행합니다. 서버는 이름 지정된 카운터의 현재값을 최소값과 동일하게 설정하고, 새 현재 값을 호출자에게 리턴하고, 이름 지정된 카운터를 증가시킵니다.

NC_NOREDUCE

지정할 남아 있는 숫자 범위(현재값과 최대값 더하기 1 사이의 차이)가 *update_value* 매개변수에 지정된 증분보다 적으면 지정은 실패합니다 (NC_WRAP가 강제 실행되지 않는 한). NC_NOREDUCE는 NC_NOWRAP와 함께 NC_ASSIGN 요청이 NC_COUNTER_AT_LIMIT 조건으로 실패함을 의미합니다.

예를 들어, 현재 번호가 199 990이고 카운터 최대 수가 199 999로 정의되었을 때 요청이 갱신 값 15를 지정하면 증분으로 현재 번호가 200 000을 초과하게 되므로 NC_REQUEST는 실패합니다.

NC_REDUCE

지정할 남아 있는 숫자 범위(현재값과 최대값 더하기 1 사이의 차이)가 *update_value* 매개변수에 지정된 증분보다 적으면 증분은 감소되고 지정은 성공합니다. 이 경우 NC_ASSIGN 요청은 *update_value*에 지정된 것보다 적은 숫자 범위를 지정했고, 이름 지정된 카운터는 NC_COUNTER_AT_LIMIT 조건에 남아 있습니다. 후속 NC_ASSIGN 요청은 이름 지정된 카운터가 NC_REWIND 요청으로 재설정될 때까지 실패합니다.

NC_CREATE에 지정된 옵션은 이름 지정된 카운터와 함께 저장되고 다른 이름 지정된 카운터 함수의 기본값으로 사용됩니다. NC_ASSIGN, NC_REWIND 또는 NC_UPDATE 요청에서 옵션을 덮어쓸 수 있습니다. DFHNCTR 호출에 *counter_options*을 지정하고 싶지 않은 경우에는 기호 상수 NC_NONE(0과 같음)를 입력 매개변수로 지정하거나 널(null) 주소를 지정하십시오.

NC_CREATE, NC_ASSIGN, NC_REWIND 및 NC_UPDATE 함수의 경우 이 매개변수는 입력 필드입니다.

NC_DELETE, NC_INQUIRE 및 NC_BROWSE 함수의 경우 이 매개변수는 출력 필드이고 NC_CREATE에 지정된 옵션을 리턴합니다.

update_value

카운터를 갱신하는 데 사용할 값을 지정합니다. NC_ASSIGN의 경우, 이는 현재 카운터 값에 추가될 증분입니다(현재 번호가 지정된 후에). 1이 아닌 증분을 지정하는 것이 지정 작업에 어떤 영향을 미치는지에 대한 정보는 *function* 매개변수에서 NC_ASSIGN 옵션을 참조하십시오.

NC_UPDATE의 경우 이는 이름 지정된 카운터의 새 현재 값입니다.

compare_min

이름 지정된 카운터의 현재 값과 비교할 값을 지정합니다. 값을 지정하면 이 매개변수는 지정된 값보다 크거나 같은 이름 지정된 카운터의 현재 값에서 NC_ASSIGN, NC_REWIND 또는 NC_UPDATE 작업을 조건부로 만듭니다. 비교가 충족되지 않은 경우에는 작업이 범위를 벗어난 카운터 리턴 코드(RC 103)로 거부됩니다.

널(null) 주소를 지정하여 이 매개변수를 생략하면 서버는 비교를 수행하지 않습니다.

compare_max

이름 지정된 카운터의 현재 값과 비교할 값을 지정합니다. 값을 지정하면 이 매개변수는 지정된 값보다 작거나 같은 이름 지정된 카운터의 현재 값에서 NC_ASSIGN, NC_REWIND 또는 NC_UPDATE 작업을 조건부로 만듭니다. 비교가 충족되지 않은 경우에는 작업이 범위를 벗어난 카운터 리턴 코드(RC 103)로 거부됩니다.

이 매개변수에 높은 값(X'FF')을 지정하면 서버는 비교를 수행하지 않습니다. *value_length* 매개변수가 지정하는 모든 바이트에 (X'FF')를 지정해야 합니다.

compare_max 값이 compare_min 값보다 작으면 유효한 범위는 반올림될 것으로 가정됩니다. 이 경우 현재 값은 둘 중 하나의 비교를 충족하는 경우에는 범위 내에 있는 것으로 간주됩니다. 그렇지 않은 경우에는 두 비교를 모두 충족해야 합니다.

결과 오버플로우 확인

호출 인터페이스는 결과 필드의 지정된 크기에 맞지 않거나 부호 있는 변수가 사용될 때 부호 비트로 오버플로우하는 결과에 대해 검사합니다.

결과 필드(counter_value, minimum_value 또는 maximum_value)가 value_length를 음수 값으로 지정하여 부호 있는 변수로서 정의된 경우, 호출 인터페이스는 부호 비트로 오버플로우하는 결과를 검사합니다. 이 경우 조작은 정상적으로 완료하지만 리턴 코드 NC_RESULT_OVERFLOW가 설정됩니다. 특수한 사례로서, 한계 값에 있는 카운터에 대한 결과 값은 리턴 코드를 불필요하게 설정하는 것을 피하기 위해 이 양식의 오버플로우에 대해 검사되지 않습니다. 이것은 한계에 있고 그의 최대값이 최대 양수 값인 카운터에 대해 조회하는 경우, 이 리턴 코드를 유발하지 않고 현재 카운터 값으로 음수 숫자가 리턴될 수 있음을 의미합니다.

결과 필드(counter_value, minimum_value 또는 maximum_value)가 결과의 0이 아닌 전체 부분을 포함하기에는 너무 짧은 경우, 조작은 정상적으로 완료하지만 다음 리턴 코드 중 하나가 설정됩니다.

- 선행 파트가 정확하게 1인 경우 NC_RESULT_CARRY.
- 선행 파트가 1보다 큰 경우 NC_RESULT_TRUNCATED.

높은 값의 최대값을 갖는 4바이트 부호 없는 카운터가 그의 한계 값에 도달한 경우, 리턴 코드 NC_RESULT_CARRY가 설정되고 카운터 값은 0입니다.

널(null) 매개변수를 갖는 DFHNCTR 호출 예제

DFHNCTR 호출에서 선택적 매개변수를 생략하는 경우, 매개변수 목록이 누락된 매개변수에 대한 널(null) 주소를 갖고 빌드되는지 확인하십시오. 뒤에 오는 예제는 COBOL 프로그램에서 널(null) 주소를 설정된 일부 매개변수를 갖는 NC_CREATE 요청을 실행하는 방법을 보여줍니다.

생략된 매개변수에 대한 널(null) 주소를 갖는 DFHNCTR 호출: 이 예제에서, 호출에서 사용되는 매개변수는 다음과 같이 WORKING-STORAGE SECTION에서 정의됩니다.

호출 매개변수	COBOL 변수	필드 정의
function	01 NC-FUNCTION	PIC S9(8) COMP VALUE +1.
return_code	01 NC-RETURN-CODE.	PIC S9(8) COMP VALUE +0.
pool_selector	01 NC-POOL-SELECTOR	PIC X(8).
counter_name	01 NC-COUNTER-NAME	PIC X(16).
value_length	01 NC-VALUE-LENGTH	PIC S9(8) COMP VALUE +4.
current_value	01 NC-CURRENT-VALUE	PIC S9(8) VALUE +0.
minimum_value	01 NC-MIN-VALUE	PIC S9(8) VALUE +0.

호출 매개변수	COBOL 변수	필드 정의
<i>maximum_value</i>	01 NC-MAX-VALUE	PIC S9(8) VALUE -1.
<i>counter_options</i>	01 NC-OPTIONS	PIC S9(8) COMP VALUE +0.
<i>update_value</i>	01 NC-UPDATE-VALUE	PIC S9(8) VALUE +1.
<i>compare_min</i>	01 NC-COMP-MIN	PIC S9(8) VALUE +0.
<i>compare_max</i>	01 NC-COMP-MAX	PIC S9(8) VALUE +0.

널(null) 주소에 대해 사용되는 변수는 다음과 같이 LINKAGE SECTION에서 정의됩니다.

LINKAGE SECTION.

01 NULL-PTR USAGE IS POINTER.

설명된 대로 WORKING-STORAGE SECTION에서 지정된 데이터 이름과 LINKAGE SECTION에서 지정된 NULL-PTR 이름을 사용하면 다음 예제는 *value_length*, *current_value*, *minimum_value* 및 *counter_options*가 지정된 유일한 선택적 매개변수인 이름 지정된 카운터 서버에 대한 호출을 설명합니다. 다른 것은 기본값으로 허용되거나 후속 선택적 매개변수의 경우에 함께 생략됩니다.

NAMED-COUNTER SECTION.

```
*
  SET ADDRESS OF NULL-PTR TO NULLS.
*
  MOVE 1      TO NC-FUNCTION.
  MOVE 100    TO NC-MIN-VALUE NC-CURRENT-VALUE.
  MOVE NC-WRAP TO NC-OPTIONS.
  MOVE "DFHNC001" TO NC-POOL-SELECTOR.
  MOVE "CUSTOMER_NUMBER" TO NC-COUNTER-NAME.
  CALL 'DFHNCTR' USING NC-FUNCTION NC-RETURN-CODE NC-POOL-SELECTOR
                      NC-COUNTER-NAME NC-VALUE-LENGTH NC-CURRENT-VALUE
                      NC-MIN-VALUE NULL-PTR NC-OPTIONS.
```

DFHNCTR과 함께 COUNTER 대신 DCOUNTER를 사용하려는 경우, NC-CURRENT-VALUE, NC-MIN-VALUE 및 NC-MAX-VALUE 정의를 부호 있는 10진수 값(이전에 나열된 대로)에서 부호 없는 2진 정의(예: PIC 9(16) COMP)로 변경해야 합니다. 또한, 8을 NC-VALUE-LENGTH으로 이동하십시오.

리턴 코드

이름 지정된 카운터 호출 인터페이스는 3개의 경고 리턴 코드(1 - 3)를 갖는데, 이들은 그렇지 않으면 정상적으로 완료한 요청에 대한 결과 오버플로우를 표시합니다. 둘 이상의 경고 리턴 코드가 동일한 요청에 적용되는 경우, 가장 높은 적용 가능한 경고 리턴 코드가 설정됩니다.

나머지 리턴 코드는 심각도에 따라서 범위(100, 200, 300 및 400)로 나뉘어집니다. 0이 아닌 리턴 코드의 각 범위는 리턴 코드 범주를 설명하는 더미 리턴 코드로 시작하여 기호 이름을 사용하여 각 범위에 있는 값을 확인하기 쉽게 만듭니다.

다음 목록에서 숫자 리턴 코드 뒤에 기호 이름이 뒤따릅니다.

0 (NC_OK)

요청이 정상적으로 완료했습니다.

1 (NC_RESULT_OVERFLOW)

결과 값이 부호 비트로 오버플로우했습니다.

2 (NC_RESULT_CARRY)

결과 값이 오버플로우했고, 선행 파트가 정확하게 1입니다.

3 (NC_RESULT_TRUNCATED)

결과 값이 오버플로우했고, 선행 파트가 1보다 큼니다.

100 (NC_COND)

이 범위의 리턴 코드는 조건이 충족되지 않았기 때문에 조건부 함수가 성공하지 못했음을 표시합니다.

101 (NC_COUNTER_AT_LIMIT)

이 이름 지정된 카운터에 대한 이전 요청이 최대값을 얻었고 카운터가 지금 한계에 있기 때문에 NC_ASSIGN 함수가 거부됩니다. NC_REWIND 함수 호출이 실행되어 카운터를 다시 설정할 때까지 새 카운터 값을 지정할 수 없습니다.

102 (NC_COUNTER_NOT_AT_LIMIT)

이름 지정된 카운터가 한계 값에 있지 않기 때문에 NC_REWIND 함수가 거부됩니다. 이것은 다른 태스크가 이미 NC_REWIND로 카운터 다시 설정에 성공했을 때 발생할 수 있습니다.

103 (NC_COUNTER_OUT_OF_RANGE)

이름 지정된 카운터의 현재 값이 *compare_min* 및 *compare_max* 매개변수에 지정된 범위 안에 있지 않습니다.

200 (NC_EXCEPTION)

이 범위의 리턴 코드는 애플리케이션이 처리할 수 있는 예외 조건을 표시합니다.

201 (NC_COUNTER_NOT_FOUND)

이름 지정된 카운터를 찾을 수 없습니다.

202 (NC_DUPLICATE_COUNTER_NAME)

지정된 이름을 갖는 이름 지정된 카운터가 이미 존재하기 때문에 NC_CREATE 함수가 거부됩니다.

203 (NC_SERVER_NOT_CONNECTED)

선택된 풀에 대한 활성 연결이 존재하지 않기 때문에 NC_FINISH 함수가 거부됩니다.

300 (NC_ENVIRONMENT_ERROR)

이 범위의 리턴 코드는 환경 오류를 표시합니다. 이들은 일반적으로 몇 가지 외부 요인에 의해 유발되며 프로그램이 처리할 수 없을 수 있는 심각한 오류입니다.

301 (NC_UNKNOWN_ERROR)

서버가 인터페이스가 이해하지 못하는 오류 코드를 보고했습니다. 일반적으로, 이것은 인터페이스 로드 모듈 DFHNCIF가 서버 자체보다 더 낮은 유지보수 또는 릴리스 레벨에 있지 않는 한 불가능합니다.

302 (NC_NO_SPACE_IN_POOL)

이름 지정된 카운터 풀에 공간이 충분하지 않기 때문에 새 카운터를 작성할 수 없습니다.

303 (NC_CF_ACCESS_ERROR)

결합 기능에 액세스하는 데 사용되는 매크로에서 구조 결합 또는 연결 유실 같은 예기치 않은 오류가 발생했습니다. 애플리케이션 작업 로그의 DFHNC0441 메시지에서 추가 정보를 찾을 수 있습니다.

304 (NC_NO_SERVER_SELECTED)

현재 옵션 테이블을 사용하여 프로그램에서 지정된 풀 선택사항 매개변수를 올바른 서버 이름으로 해석할 수 없습니다.

305 (NC_SERVER_NOT_AVAILABLE)

적합한 이름 지정된 카운터 풀에 대해 인터페이스가 서버에 대한 연결을 설정할 수 없습니다. 애플리케이션 작업 로그의 AXM 서비스 메시지에서 추가 정보를 찾을 수 있습니다.

306 (NC_SERVER_REQUEST_FAILED)

요청의 서버 처리 중에 이상 종료가 발생했습니다. 애플리케이션 작업 로그 및 서버 작업 로그에 있는 메시지에서 추가 정보를 찾을 수 있습니다.

307 (NC_NAME_TOKEN_ERROR)

이름 지정된 카운터 인터페이스 모듈의 IEANTxx 이름/토큰 서비스 호출이 예기치 않은 리턴 코드를 제공했습니다.

308 (NC_OPTION_TABLE_NOT_FOUND)

풀 이름 해석에 필요한 DFHNCOPT 옵션 테이블 모듈을 로드할 수 없습니다.

309 (NC_OPTION_TABLE_INVALID)

옵션 테이블의 처리 중에 이름 지정된 카운터 인터페이스에서 알 수 없는 항목 형식을 만났습니다. 옵션 테이블이 올바르게 생성되지 않거나, DFHNCIF 인터페이스 로드 모듈이 옵션 테이블과 동일한 릴리스 레벨에 있지 않습니다.

310 (NC_USER_EXIT_NOT_FOUND)

주어진 풀 이름과 일치하는 옵션 테이블 항목이 사용자 엑시트 프로그램을 지정했지만, 사용자 엑시트 프로그램이 옵션 테이블로 링크 편집되지 않고 로드될 수 없습니다.

311 (NC_STRUCTURE_UNAVAILABLE)

이름 지정된 카운터 서버 목록 구조가 일시적으로 사용 불가능합니다. 예를 들어, 이 상황에 대한 한 가지 이유는 z/OS 시스템 관리 리빌드가 진행 중이라는 것입니다.

참고: 이름 지정된 카운터에 대한 EXEC CICS 인터페이스는 CALL 인터페이스를 내부적으로 사용하지만, 1초 동안 대기하고 요청을 재시도하여 이 리턴 코드를 애플리케이션으로부터 숨깁니다. EXEC CICS 인터페이스는 이 대기를 계속하고 성공할 때까지 재시도하며, 애플리케이션은 오류 응답이 아니라 시간 지연만을 갖는 결과를 갖습니다. CALL 인터페이스를 사용하는 애플리케이션에서 동일한 기술을 사용할 수 있습니다.

400 (NC_PARAMETER_ERROR)

이 범위의 리턴 코드는 일반적으로 호출하는 프로그램에서 코딩 오류의 결과인 매개변수 오류를 표시합니다.

401 (NC_INVALID_PARAMETER_LIST)

매개변수 목록이 다음 이유 중 하나 때문에 올바르지 않습니다.

- 너무 적은 매개변수(4개 미만 또는 NC_FINISH 함수의 경우 3개 미만)가 지정됩니다.
- 너무 많은 매개변수(9개 이상)가 제공됩니다.
- 매개변수 주소가 0입니다.
- 목록 끝 표시자가 누락되었습니다.

402 (NC_INVALID_FUNCTION)

함수 코드 매개변수가 지원되는 범위에 있지 않습니다.

403 (NC_INVALID_POOL_NAME)

풀 선택사항 매개변수가 허용되지 않는 문자나 삽입된 공백을 포함합니다.

404 (NC_INVALID_COUNTER_NAME)

counter_name 매개변수가 허용되지 않는 문자 또는 삽입된 공백을 포함합니다.

405 (NC_INVALID_VALUE_LENGTH)

값 길이 매개변수가 0 - 8의 범위에 있지 않습니다.

406 (NC_INVALID_COUNTER_VALUE)

지정된 카운터 값이나 증분 값이 카운터에 대한 최소 및 최대 한계와 일치하지 않습니다.

NC_CREATE 함수의 경우 *current_value* 매개변수 또는 NC_UPDATE 함수의 경우 *update_value*에 지정된 카운터 값은 지정된 최소값보다 작을 수 없으며 (최대값 + 1)보다 클 수 없습니다.

NC_ASSIGN 또는 NC_REWIND 함수에 대한 *update_value* 매개변수에 지정된 증분 값은 카운터의 총계 범위((최대값 - 최소값) + 1)보다 클 수 없습니다.

407 (NC_INVALID_COUNTER_LIMIT)

최대값이 최소 값보다 작습니다.

408 (NC_INVALID_OPTIONS)

counter_options 매개변수의 값이 올바르지 않습니다. 어떤 정의된 옵션에도 대응하지 않는 값이거나 상호 배타적인 옵션을 나타내는 값입니다.

이름 지정된 카운터 복구

이름 지정된 카운터는 결합 기능에서만 저장됩니다. 그러므로 이름 지정된 카운터를 사용하는 애플리케이션은 임의의 결합 기능 문제점의 가능한 영향을 처리하기 위한 복구 로직을 구현해야 할 수 있습니다.

이름 지정된 카운터 풀의 결합 기능 또는 목록 구조가 실패하지만 또 다른 기능이 사용 가능한 경우, 일반적으로 다른 기능에서 매우 빨리 이름 지정된 카운터의 목록 구조를 다시 작성할 수 있어야 합니다. 서버의 원래 인스턴스는 문제점을 발견하자마자 종료하고, 새 인스턴스가 일반적으로 ARM(사용 가능한 경우)에 의해 즉시 시작되며, 설치 정책이 재시작을 허용합니다. 풀의 목록 구조가 실패한 것으로 알려지는 경우, 새 서버는 새 인스턴스를 즉시 할당할 수 있어야 하며 풀은 수 초 안에 사용 가능해야 합니다.

그러나 결합 기능 전원 장애 같은 일부 상황에서는 MVS가 초기에 상황을 연결 유실로 인지하고 원래 기능이 재시작되었을 때까지 목록 구조가 실패했음을 판별할 수 없는 경우가 있습니다. 그런 상황에서는 운영자 명령을 실행하여 기존 구조의 삭제를 강제 실행하여 새 인스턴스가 즉시 할당될 수 있도록 함으로써 복구를 가속화할 수 있습니다.

새 구조가 작성되었을 때까지, 서버가 사용 불가능하기 때문에 카운터 값을 얻으려는 시도가 거부됩니다. 이것은 번호를 할당하는 대체 방법을 갖고 있지 않는 한 새 구조가 작성되고 있는 동안에는 그런 요청을 발행하는 애플리케이션이 사용 불가능함을 의미합니다.

이름 지정된 카운터 풀에 대한 목록 구조가 실패 때문에 다시 작성되는 경우, 목록 구조는 비어 있으며 애플리케이션은 즉시 이름 지정된 카운터를 더 이상 찾을 수 없음을 발견합니다. 이 상황에서의 표준 복구 기술은 다음과 같습니다.

1. 카운터 이름을 바탕으로 자원에 대해 애플리케이션이 큐에 넣기 명령(ENQ)을 실행하게 하십시오. 이것은 단 하나의 태스크가 각 이름 지정된 카운터를 수리하려고 시도하게 합니다.
2. 다른 태스크가 이미 이름 지정된 카운터를 다시 작성했는지 확인하십시오.

3. 이름 지정된 카운터가 다시 작성되지 않은 경우, 설명하는 다음 방법을 사용하여 다른 정보로부터 재구성할 적절한 값을 사용하여 다시 작성하십시오.
4. 자원에 대해 큐에서 해제 명령(DEQ)을 실행하여 이름 지정된 카운터를 해제하십시오.

큐에 넣기 및 큐에서 해제 프로세스는 시간을 소비하는 다중 태스크 및 동일한 처리를 중복하는 자원을 피하기 위해 사용됩니다. 그러나 이름 지정된 카운터를 다시 작성하기 위해 사용하는 프로세스가 충분히 단순한 경우 큐에 넣기 및 큐에서 해제 프로세스를 생략할 수 있습니다. 다른 태스크가 이미 이름 지정된 카운터를 수리한 경우, 카운터를 다시 작성하려는 모든 후속 시도는 중복 카운터 이름 표시와 함께 거부됩니다.

이름 지정된 카운터가 구조 실패 사이에서 보존될 필요가 없는 임시 정보(예를 들어, 그런 실패에도 살아남을 것으로 예상되지 않는 활성 프로세스의 고유 ID)에만 사용되고 있는 경우, 복구 기술은 최소한일 수 있습니다. 예를 들어, 표준 초기값을 사용하여 카운터를 다시 작성할 수 있습니다.

그러나 이름 지정된 카운터가 주문 번호 같은 영구 정보에 사용되는 경우, 다시 작성하려면 카운터 값을 재구성하기 위한 특정 애플리케이션이 필요할 수 있습니다. 예를 들어, 애플리케이션이 주문 파일에서 현재 사용되는 가장 높은 키를 찾을 수 있습니다. 활성 트랜잭션이 이미 카운터로부터 새 번호를 획득했을 수 있지만 아직 사용하지 않은 경우, 복구 프로세스에서 이를 허용해야 합니다. 지정되었지만 아직 기록되지 않은 값을 허용하는 두 가지 방법은 다음과 같습니다.

1. 애플리케이션이 카운터를 이미 지정되었을 수 있는 값으로 설정하지 않도록 충분히 큰 여백을 선택하여 마지막으로 사용된 값에 안전 여백을 추가하십시오.
2. 모든 카운터 값을 일시적인 것으로 취급하십시오. 카운터를 명백히 사용되지 않은 다음 값으로 복원하고, 카운터를 사용하는 애플리케이션에서 카운터 값이 이미 지정되었고 애플리케이션이 이를 사용하려고 시도하는 상황을 다루는 로직을 포함하십시오. 중복 키는 값(데이터베이스 또는 파일 키로서) 사용되는 시점에 중복 키 예외에 의해 발견될 수 있으며, 이 때 애플리케이션이 새 카운터 값을 얻고 다시 시도할 수 있습니다. 중복된 값을 사용하려는 원래 시도로부터 부작용이 발생하지 않도록 주의깊게 확인하십시오.

가장 높게 사용된 카운터 값을 찾고 일시적으로 다음 값을 지정하는 기술은 이름 지정된 카운터 서버가 사용 불가능할 때 번호를 지정하는 백업 방법으로도 사용될 수 있습니다. 그러나, 중복 키를 다루는 로직이 일반적으로 비정상적인 복구 상황에서만 연습되기 때문에 주의깊은 검증 및 테스트가 필요합니다.

기존 데이터 저장소로부터 카운터 값을 다시 작성하기 어려운 경우, 또 다른 가능성은 종종(예를 들어, 100 또는 1000개 번호당 한 번) 카운터 값이 파일의 레코드에 로깅될 수 있다는 점입니다. 복구 로직이 파일에 기록된 번호를 취하고 값이 로깅된 간격의 두 배 같은 안전 여백을 추가하여 이름 지정된 카운터에 대해 적합한 값을 다시 작성할 수 있습니다.

z/OS 릴리스 3 이상을 실행하는 시스템에서는, 시스템 관리 이중화를 사용하여 서로 다른 결합 기능에 이름 지정된 카운터의 이중화 사본을 유지보수할 수 있습니다. 이것은 카운터에 대한 액세스 유실 위험을 크게 줄이지만, 성능 및 자원에서 몇 가지 비용을 포함합니다. 아마도 운영상 오류 또는 소프트웨어 문제점 때문에 아직 구조를 잃는 몇 가지 이론적 위험이 있고, 결합 기능의 임의의 데이터가 영구적으로 간주될 수 없으므로 카운터 값을 재구성하는 일부 방법이 아직 필요할 수 있습니다.

제 14 부 인쇄 및 스푼 파일

프린터 제어 및 스푼 파일에 대한 이해 및 CICS 기능 사용.

제 44 장 인쇄 및 스푼 파일

CICS는 인쇄를 위한 특수 명령을 제공하지 않지만, 프린터에만 적용되는 터미널 제어 명령과 BMS에 옵션이 있으며, 일부 프린터의 경우에는 트랜지언트 데이터 또는 SPOOL 명령을 사용합니다.

보통 다른 유형의 일반 사용자 통신에서 발생하지 않는 두 가지의 인쇄 관련 문제가 있습니다.

1. 추가 형식화 고려사항이 있습니다(특히 3270 프린터에 해당되는).
2. 인쇄해야 하는 태스크에 프린터에 대한 직접 액세스 권한이 없을 수 있습니다.

또한 프린터의 고유한 두 가지 범주가 있으며, 범주는 다른 API(Application Programming Interface) 가지고 있습니다.

CICS 프린터

CICS에 대한 터미널로 정의되고 CICS에 의해 직접 관리되는 프린터. 상대적으로 짧은 문서의 요구 시 인쇄에 적합한, 사용자 가까이 있는 저속 디바이스입니다. 3289 및 3262는 보통 CICS 프린터로 접속됩니다.

비CICS 프린터

운영 체제 또는 다른 애플리케이션에 의해 관리되는 프린터. 이 프린터는 보통 즉시 사용 가능하지 않아도 되는 볼륨 인쇄에 적합한, 중앙 처리 사이트에 있는 고속 디바이스입니다. 이 프린터는 특수 연결, 관리 또는 공유가 필요한 다른 프린터나 고급 기능이 될 수도 있습니다.

CICS 프린터의 형식

프린터 터미널에 쓰기 위한 API(Application Programming Interface)는 기본적으로 포 시장치에 쓰기 위한 것과 동일합니다. 터미널 제어 명령(SEND)을 모든 CICS 프린터에 사용할 수 있고, 이들의 대부분은 BMS에서도 지원됩니다(SEND MAP, SEND TEXT 및 SEND CONTROL).

태스크가 프린터를 프린시펄 기능으로 가지고 있는 배열의 문제점은 887 페이지의 『CICS 프린터 사용』에서 설명됩니다.

912 페이지의 『BMS 자원 레벨』에서는 BMS가 지원하는 디바이스를 나열합니다. 외부 제어기 또는 LU 유형 4의 구성요소인 프린터의 경우 일괄처리 데이터 교환(BDI) 명령뿐만 아니라 터미널 제어 및 BMS를 사용할 수 있습니다. BDI 명령은 751 페이지의 제 32 장 『일괄처리 데이터 상호 교환 사용』에 설명되어 있습니다.

BMS와 터미널 제어 사용 간의 선택사항은 표시장치 터미널에서와 동일한 고려사항을 기반으로 합니다. 표시장치와 마찬가지로 프린터는 기능면이나 해당 기능의 구현면에서 서로 크게 차이가 나고, 이러한 차이점은 이들이 승인하는 데이터 스트림과 디바이스 제어에 반영되어 있습니다.

터미널 제어 명령을 사용할 때 애플리케이션 코드는 프린터에 필요한 방법으로 출력을 포맷해야 합니다. 라인 프린터 및 유사한 디바이스의 경우 포맷팅은 프로그램에 거의 영향을 미치지 않습니다. 그러나 고기능 프린터의 경우 데이터 스트림은 종종 매우 복잡합니다. 포맷팅에는 중요한 애플리케이션 코드가 필요하고 디바이스 종속성을 프로그램 로직에 소개합니다.

이러한 터미널의 일부의 경우, 코딩 노력은 BMS를 사용하여 대폭 줄어들고, 이는 프로그래머가 디바이스 데이터 스트림을 작성하거나 이해하는 노고를 줄여줍니다. BMS는 또한 동일 프로그램이 변경없이 여러 유형의 프린터 또는 프린터와 표시장치의 혼합을 지원할 수 있도록 애플리케이션 코드에서 대부분의 데이터 스트림 종속성을 제거합니다. BMS는 모든 디바이스 종속성을 제거하지 않고 형식에 몇몇 제한사항을 부과합니다. 이는 또한 추가 경로 길이와 관련되고, 양은 사용자가 만드는 별개의 BMS 요청 수, 요청의 복잡도 및 사용자 고유 프로그램에서 회피한 해당하는 경로 길이에 달려 있습니다.

인쇄된 출력에 대한 요청

CICS 인쇄 요청은 요청 화면에 표시된 내용을 동일한 제어 장치에서 사용 가능한 첫 번째 프린터에 복사하도록 CICS에 요구합니다. 관련된 오버헤드는 프린터 사용 가능 여부, 요청 터미널이 CICS에 원격인지 로컬인지 여부에 따라 다릅니다.

사용 가능한 프린터가 없고 요청이 원격 또는 로컬 디바이스에서 이루어진 경우 다음과 같습니다.

- CICS가 표시장치 터미널에 버퍼를 읽습니다. 여기에는 널(null)을 포함하여 화면의 모든 위치 전송이 수반됩니다.

로컬 디바이스에서 요청하는 경우, READ BUFFER 명령은 채널 속도로 발생하므로 대형 입력 메시지 크기는 응답 시간을 너무 많이 증가시키지 않고 회선을 독점하지 않습니다.

- 터미널 오류 프로그램이 메시지를 처리할 수 있도록 오류 태스크가 생성됩니다. 프린터를 사용할 수 있고 요청이 로컬 디바이스에서 이루어진 경우에는 이 단계가 필요 없습니다.
- 3270 인쇄 태스크(CSPP)가 첨부되어 사용 가능하면 프린터에 전체 버퍼를 기록합니다.

프린터를 사용할 수 있고 요청이 원격 디바이스에서 이루어진 경우, CICS가 요청 디바이스 버퍼 사본을 출력 디바이스 버퍼로 전송하도록 요구하는 아주 짧은 데이터 스트림을 제어 장치로 전송합니다.

CICS 3270 프린터

BMS가 제공하는 프린터의 추가 형식 제어의 대부분은 CICS 프린터, 3270 프린터의 특정 유형을 위한 것입니다. 3270 프린터는 3270 데이터 스트림을 승인하는 모든 프린터이고, 이는 3270 표시장치와 동등한 하드카피입니다.

여기에는 3270 표시장치의 표시장치 버퍼에 해당하는 페이지 버퍼가 있습니다. (3270 데이터 스트림의 소개 설명에 대해서는 757 페이지의 『3270 버퍼』의 내용을 참조하십시오.) 3270 프린터에 대해 먼저 설명하고, 더 단순한 비3270 프린터에 대해서는 884 페이지의 『비3270 CICS 프린터』까지 지연합니다.

3270 프린터는 버퍼 제어 주문과 인쇄 형식 주문의 두 개의 서로 다른 유형의 포매팅 지시사항을 승인합니다. 버퍼 제어 명령은 제어 장치가 수신할 때 실행되고 버퍼가 채워지는 방식을 관리합니다. 이들은 3270 표시장치 화면을 포맷하는 데 사용되는 것과 동일한 주문입니다. 763 페이지의 『데이터 스트림에서의 순서』에서 이미 중요한 사항의 일부를 설명했습니다. 예를 들어, SBA(세트 버퍼 주소)는 제어 장치에 버퍼에서 뒤따르는 데이터를 배치할 위치를 알려주고, SF(시작 주소)는 속성 바이트 및 가능한 필드 데이터 등등을 신호를 보냅니다. *IBM 3270 Information Display System Data Stream Programmer's Reference* 매뉴얼에서 전체 목록을 찾을 수 있습니다.

반대로, 인쇄 형식 주문은 수신될 때 실행되지 않지만 대신 데이터와 함께 버퍼에 저장됩니다. 이러한 주문 NL(새 행), FF(용지 공급) 등등은 인쇄 작업 중에만 해석되고, 이 기간 중에 인쇄된 출력의 형식을 제어합니다. 이들은 버퍼 위치를 차지하는 것 외에는 표시장치에 아무런 효과도 없습니다. 이들은 화면에 공백처럼 보입니다.

3270 프린터에 쓰고 있다면 버퍼 제어 주문이나 인쇄 형식 주문 또는 이들의 혼합으로 포맷할 수 있습니다. 767 페이지의 『아웃바운드 데이터 스트림 예제』에 버퍼 제어 주문으로 포맷하는 예제를 표시합니다. 이 동일한 데이터 스트림을 3270 프린터로 전송하는 경우에는 이는 767 페이지의 그림 128에 표시된 화면의 이미지를 인쇄합니다. 동일한 데이터 스트림을 표시장치와 프린터로 보낼 수 있도록 버퍼 제어 주문으로 인쇄된 출력을 포맷하기를 선택할 수도 있습니다.

반면, 동일한 스트림을 3270 프린터 및 비3270 프린터로 전송할 수 있도록 인쇄 형식 주문으로 포맷하도록 선택할 수도 있습니다. 인쇄 형식 주문은 많은 비3270 프린터에서의 포맷 제어와 같습니다. 이 선택사항에 대한 자세한 정보는 882 페이지의 『NLEOM 옵션』에서 NLEOM 옵션의 설명을 참조하십시오.

다음은 버퍼 제어 주문을 사용하는 767 페이지의 『아웃바운드 데이터 스트림 예제』에서 데이터 스트림과 동일한 인쇄된 출력을 생성하는 인쇄 형식 주문을 사용하는 데이터입니다.

표 44. 인쇄 제어 주문을 사용하는 데이터 스트림의 예제

바이트	내용	참고
1	X'FF'	프린터가 새 페이지에 공백을 만들게 하려면 "Formfeed"(FF) 주문
2-23	공백	첫 행에서 열 1-22를 차지하기 위한 22개의 공백.
24-33	자동차 레코드	행 1에서 다음 사용 가능한 열(23-32)에 나타나는 인쇄될 텍스트.
34	X'1515'	프린터를 세 번째 행의 시작부에 배치하기 위한 두 개의 연속된 "새 행"(NL) 주문.
35-80	직원 번호: _____ 태그 _____ 상태: __	행 3의 첫 번째 위치에서 시작하여 인쇄될 텍스트.
81	X'19'	인쇄를 중지하는 "메시지 끝"(EM) 인쇄 주문.

인쇄 형식 주문을 사용하면 필드 구조는 유실된다는 점을 유의하십시오. 프린터를 입력을 위해 사용하지 않으므로 이는 일반적으로 문제가 되지 않습니다. 그러나 인쇄 제어 주문으로 포맷하더라도 색상 또는 밑줄과 같은 속성을 텍스트의 영역에 지정하기 위해 버퍼 제어 주문 또한 사용해야 할 수도 있습니다.

CICS 3270 프린터 옵션

BMS의 경우 3270 프린터에 적용하는 특수 제어는 명령 옵션의 양식을 취합니다. PRINT, ERASE, L40, L64, L80, HONEOM, NLEOM, FORMFEED 및 PRINTERCOMP.

터미널 제어 명령에서 ERASE는 또한 옵션으로 표현되지만 다른 제어는 데이터 스트림에 직접 표현됩니다. IBM 3270 Data Stream Device Guide 및 IBM 3270 Data Stream Programmers Reference에서는 이를 인코딩하는 방법을 알려줍니다. 다음에 뒤 따르는 설명이 이들이 수행하는 작업에 대해 설명합니다.

PRINT 옵션과 인쇄 제어 비트

3270 표시장치나 프린터에 쓰면 디바이스 버퍼를 갱신합니다. 표시장치에서는 결과가 화면에 즉시 반영되는데, 이것은 버퍼로부터 구동됩니다. 그러나 프린터의 경우 "쓰기 제어 문자"에서 적절한 비트를 켜 때까지 인쇄가 발생하지 않기 때문에 시각적인 효과가 없을 수 있습니다.

WCC는 3270 데이터 스트림의 일부입니다. 758 페이지의 『출력 데이터 스트림』의 내용을 참조하십시오. BMS의 경우, SEND MAP, SEND TEXT 또는 SEND CONTROL

명령이나 SEND MAP과 함께 사용되는 맵에서 PRINT 옵션을 지정하여 인쇄 비트를 켭니다. 터미널 제어 SEND를 사용 중인 경우, CTLCHAR 옵션을 사용하여 인쇄 비트를 켜야 합니다.

터미널 쓰기가 모든 터미널 제어 SEND에서 발생하며, ACCUM 또는 PAGING 옵션을 사용 중인 아닌 한 모든 SEND MAP, SEND TEXT 또는 SEND CONTROL에서 발생합니다. ACCUM은 페이지가 가득 차거나 논리 메시지가 끝날 때까지 쓰기를 지연합니다. ACCUM을 사용할 때는 동일한 페이지에 대한 모든 SEND 명령에서 동일한 인쇄 옵션을 사용해야 합니다. PAGING은 또 다른 태스크에 대한 터미널 쓰기를 지연하지만, PAGING이 없는 경우와 같은 방식으로 생성됩니다.

인쇄 비트가 켜질 때까지 인쇄가 발생하지 않는다는 사실은 다중 쓰기를 갖는 스테이지에서 인쇄 버퍼를 빌드하고 이미 버퍼에 있는 데이터나 속성 바이트를 변경할 수 있게 합니다. 즉, 하드웨어를 사용하여 BMS의 ACCUM 옵션으로 얻는 효과의 일부를 달성할 수 있습니다. 그러나 NLEOM 옵션은 이 기능에 영향을 줍니다. 882 페이지의 『NLEOM 옵션』의 설명을 참조하십시오.

ERASE 옵션

3270 표시장치 버퍼처럼, 3270 프린터 버퍼는 지우는 쓰기 명령을 사용할 때만 지워집니다. BMS 및 터미널 제어 SEND 모두에 대해 ERASE 옵션을 지정하여 이를 수행합니다.

프린터가 대체 화면 크기 특성을 갖는 경우, 버퍼 크기는 표시장치의 경우와 같이 지우기 시간에 설정됩니다. 결국, 버퍼를 트랜잭션에 필요한 크기로 설정하고 이전 트랜잭션에서 남은 모든 버퍼 내용을 지우려면 트랜잭션의 첫 번째 터미널 쓰기가 지우기를 표시해야 합니다.

행 너비 옵션: L40, L64, L80, HONEOM

인쇄 비트 외에, 쓰기 제어 문자는 인쇄 중에 행 길이를 통제하는 비트 쌍을 포함합니다.

터미널 제어 명령을 사용 중인 경우 CTLCHAR 옵션을 사용하여 이들 비트를 설정합니다. BMS의 경우, 기본값은 HONEOM(“honor end-of-message”를 의미) 옵션에 의해 생성되는 비트입니다. 이 설정을 사용하면 프린터는 버퍼 제어 및 인쇄 형식 순서에 따라서만 형식화하여 버퍼의 첫 번째 EM(메시지 끝) 문자에서 인쇄를 중지합니다. 디바이스의 최대 너비(플래튼 너비)를 넘어서 인쇄하려고 시도하는 경우에만 프린터는 새 행으로 이동합니다.

그러나 40, 64 또는 80자(옵션 L40, L64 및 L80)에서 행 길이가 고정되도록 지정할 수도 있습니다. 그 경우, 프린터는 특정 인쇄 형식 주문을 무시하고, 지정된 행 크기에

도달할 때 새 행으로 이동하고, 전체 버퍼를 인쇄합니다. 무시되는 인쇄 형식 주문은 NL(줄 바꾸기), CR(캐리지 리턴) 및 EM(메시지 끝)입니다. 대신 그래픽으로 인쇄됩니다.

BMS에서 L40, L64 또는 L80을 사용하는 경우, 터미널 정의에서 페이지 너비에 대응하는 값만 사용하십시오(885 페이지의 『CICS 프린터의 특성 결정하기』 참조). 이유는 BMS가 페이지 크기를 바탕으로 버퍼 주소를 계산하고, 다른 페이지 너비를 사용하는 경우 이들 주소가 잘못되었다는 점입니다.

NLEOM 옵션

BMS는 일반적으로 SEND TEXT 또는 SEND MAP을 사용하는지와 관계 없이 3270 프린터를 포맷하기 위해 인쇄 형식 주문이 아니라 버퍼 제어 주문을 사용합니다. 그러나 NLEOM 옵션을 지정하여 BMS에 인쇄 형식 주문만을 사용하도록 말할 수 있습니다.

이를 수행하면 BMS는 데이터를 전적으로 공백과 NL(새 행) 문자로 형식하고 데이터 뒤에 EM(End-Of-Message)를 삽입합니다. NLEOM은 HONEOM을 암시합니다. (NLEOM 지원에는 표준 BMS가 필요합니다. 이는 최소 BMS에서는 사용할 수 없습니다.)

SCS 프린터와의 호환성을 유지하기 위해 이를 수행하고 싶을 수도 있습니다. (인쇄 형식 주문은 해당하는 SCS 제어 문자와 호환 가능합니다.) 다음 작동 차이는 NLEOM을 선택하거나 피할 수 있게 해줍니다.

빈 행

3270 프린터는 인쇄 중에 널(null) 행을 억제합니다. 즉, 데이터 필드가 없고 표시장치 화면에 공백으로 나타나는 행은 동일 맵이 프린터로 전송될 때 생략됩니다. BMS 하에서는 화면의 모든 행에 하나 이상의 필드를 배치하여 인쇄된 양식이 표시 양식과 정확히 같도록 강제할 수 있습니다. 그렇지 않으면 비어 있게 될 행에 단일 공백을 포함하는 필드를 사용하십시오. NLEOM을 지정해도 이와 같은 효과가 있습니다. BMS는 데이터가 있는지 없는지 여부와 관계없이 모든 행에 새 행 문자를 사용하기 때문입니다.

다중 전송

NLEOM을 사용하면, 연속된 쓰기의 데이터는 버퍼에 누적됩니다. 이는 위치 정보를 포함하지 않기 때문입니다. 그러나 BMS는 ACCUM 옵션을 사용하지 않는 한 NLEOM이 있는 각 SEND에서 데이터 끝에 EM(End-Of-Message) 문자를 추가합니다. 인쇄가 발생하면, 첫 번째 EM 문자는 인쇄를 중지하므로 NLEOM이 있는 첫 번째 SEND의 데이터(및 버퍼에서 해당 시점까지 지워지지 않은 데이터)만이 인쇄될 수 있습니다. 순 효과는 ACCUM 옵션을 사용하지 않는 한 NLEOM이 있는 여러 SEND 명령으로 채워진 버퍼를 인쇄할 수 없다는 점입니다.

페이지 너비

BMS는 항상 페이지에서 크기가 문자 위치의 수인 내부 버퍼를 사용하여 한 번에 하나의 출력 페이지를 빌드합니다. (BMS가 페이지 크기를 판별하는 방법에 대한 설명은 885 페이지의 『CICS 프린터의 특성 결정하기』의 내용을 참조하십시오.) 포맷하기 위해 버퍼 제어 주문을 사용 중이면 터미널 정의는 페이지 너비 40, 64, 80 또는 디바이스의 최대값(플래튼 크기)을 지정해야 합니다. 그렇지 않으면 출력이 제대로 포맷되지 않을 수도 있습니다. 반면, NLEOM을 사용 중이라면 터미널 정의는 최대 플래튼 크기의 페이지 너비를 지정할 수도 있습니다.

총 페이지 크기

버퍼 제어 주문을 사용 중이면 행 수와 페이지 너비의 제품은 버퍼 크기를 초과해서는 안 됩니다. 버퍼가 페이지의 이미지로 사용되기 때문입니다. 각 행의 오른쪽에서 사용하지 않는 위치는 널 문자로 표시됩니다. 그러나 NLEOM을 사용하는 경우에는 BMS는 페이지 크기를 버퍼 용량으로 제한하지 않습니다. BMS는 터미널에 정의된 페이지 크기에 따라 페이지를 빌드한 다음 가능한 경우 새 행 문자를 사용하여 스트림을 압축합니다. 결과로 나오는 스트림이 버퍼 용량을 초과하면 BMS는 이를 전송하기 위해 터미널에 여러 쓰기를 사용합니다.

FORMFEED

프린터가 양식의 맨 위로 진행 가능하도록 정의된 경우에(연관된 TYPETERM 정의에 FORMFEED 옵션을 사용하여) FORMFEED 옵션은 BMS를 버퍼의 시작부에 용지 공급 인쇄 형식 주문(X'0C')을 놓게 합니다. CICS는 이 기능을 사용하여 정의된 프린터의 용지 공급 요청을 무시합니다.

화면의 위치(1,1)를 사용하는 맵을 사용하여 SEND MAP을 실행하는 경우 순서를 겹쳐쓰고 용지 공급을 잃게 됩니다. 이는 NLEOM을 사용하는지 여부와 관계없이 발생합니다.

SEND CONTROL 명령에 FORMFEED 및 ERASE를 함께 사용하는 경우에는 결과는 NLEOM이 있는지 여부에 달려 있습니다. NLEOM이 없으면, SEND CONTROL FORMFEED ERASE는 널(null) 행의 전체 페이지가 뒤따르는 용지 공급 문자를 보냅니다. 프린터는 이러한 널(null) 행을 억제하여 이를 하나의 빈 행으로 대체합니다. NLEOM을 사용할 때는 동일한 명령이 페이지의 각 행에 대한 하나의 새 행 문자가 뒤따르는 용지 공급을 전송하므로 효과는 비3270 프린터에서처럼 전체 공백 페이지일 수 있습니다.

PRINTERCOMP 옵션

프린터에 SEND TEXT할 때는 페이지 크기에 영향을 미치는 하나의 추가 옵션이 있습니다. 이는 PRINTERCOMP 옵션이며 이는 개별 SEND TEXT 명령이 아니라 실

행 중인 트랜잭션과 연관된 PROFILE에 지정되어 있습니다. CICS가 제공하는 기본 프로파일에 PRINTERCOMP 값은 NO입니다.

PRINTERCOMP(NO) 아래에서, BMS는 3270 표시장치에 전송하는 것과 일관된 인쇄된 출력을 생성합니다. 표시장치의 경우 BMS는 각 SEND TEXT 명령의 텍스트 앞에 속성 바이트를 놓고 이 또한 속성 바이트로 각 행을 시작합니다 이러한 속성 바이트는 화면에 공간을 차지하므로 BMS는 PRINTERCOMP가 NO인 경우 이를 프린터를 위한 공백으로 대체합니다. PRINTERCOMP이 YES이면, BMS는 이러한 공백을 억제하여 사용자가 전체 프린터 너비와 버퍼의 모든 위치를 사용할 수 있게 해줍니다. 텍스트에 임베드한 새 행 문자는 PRINTERCOMP(NO)일 때에도 여전히 PRINTERCOMP(YES)로 인식됩니다.

가능한 경우 표시장치와의 호환성을 위해서 그리고 애플리케이션이 다른 프린터 유형을 사용하는 경우 일관된 결과를 보장하려면 PRINTERCOMP(NO)가 한 위치씩 사용 가능한 행 너비를 줄이더라도 이를 사용해야 합니다.

비3270 CICS 프린터

비3270 프린터는 SNA 문자 세트(SCS) 프린터 등과 같은 3270 데이터 스트림을 승인하지 않는 모든 프린터입니다. 비3270 프린터는 3270 제품군 디바이스일 수 있고, 3287 및 3262와 같은 많은 디바이스는 제어 장치에서 정의되는 방식에 따라 3270 프린터 또는 SCS(비3270) 프린터일 수 있습니다.

비3270 프린터에는 버퍼 페이지가 없으므로 버퍼 제어 주문을 이해하지 않습니다. 포맷팅은 전적으로 인쇄 제어 순서로 이뤄집니다. 3270 프린터와의 호환성을 위해 BMS는 메모리에 페이지의 이미지를 구성하여 이들을 포맷하고 항상 한 번에 하나의 전체 페이지를 출력합니다. 그러나 관련된 하드웨어 버퍼가 없는 것처럼 플랫론 너비를 초과하지 않는 한 어떤 크기 페이지라도 정의할 수 있습니다. BMS는 NLEOM 옵션을 사용하여 3270 프린터에 하는 것처럼 페이지를 인쇄하는 데 필요한 횟수만큼 전송합니다.

BMS는 이러한 프린터를 위해 공백과 NL(새 행) 문자로 포맷합니다. 터미널의 정의가 용지 공급 지원을 표시하는 경우에는 용지 공급(FF) 문자 또한 사용합니다.

BMS는 또한 터미널 정의에 HORIZFORM 옵션이 있고 맵에 HTAB 스펙이 있는 경우 포맷하기 위해 가로 탭을 사용합니다. 마찬가지로, 터미널 정의가 VERTICALFORM을 지정하고 맵이 VTAB를 포함하는 경우에는 세로 탭을 사용합니다. 탭 문자는 데이터 스트림을 상당히 단축할 수 있습니다. 탭이 사용된 경우, BMS는 현재 태스크 또는 몇몇 이전 태스크가 이미 프린터에 탭을 설정한 것으로 가정합니다. SCS 프린터에서 IBM 3270 Data Stream Device Guide에 설명한 대로 터미널 제어 SEND 명령으로 탭을 설정했습니다. 다른 비3270 프린터의 경우 적합한 디바이스 안내서를 참조해야 합니다.

SCS 프린터에 SEND TEXT하기 위해서는 BMS는 새 행(X'15')과 세트 속성(X'28')을 제외하고는 입력 데이터 스트림에서 비3270 제어 코드를 인지하지 않습니다. 다른 모든 문자는 표시장치 문자인 것으로 가정됩니다. 특히, BMS 하에서 투명성 제어 주문(X'35')을 사용하려고 시도하는 경우 데이터 스트림이 영향을 받을 수 있습니다. 이 제어 주문은 일반적으로 이를 뒤따르는 데이터를 무시되게 합니다(다음 바이트에 무시할 데이터의 길이가 포함됨). 그러나, BMS는 X'35' 제어 주문을 인지하지 않기 때문에 투명성 제어 주문을 뒤따르는 데이터를 마치 데이터 스트림의 정상 파트인 것처럼 처리합니다.

이 데이터를 제대로 처리할 수 없는 경우에는 BMS는 데이터 스트림에서 이를 제거할 수 있습니다. 예를 들어, 투명성 순서에 X'28' 문자가 발생하면 이는 세트 속성 제어 주문으로 오인됩니다. 이 경우 이를 뒤따르는 2바이트는 속성 설명으로 오인되고 3개의 바이트 모두가 데이터 스트림에서 제거될 수도 있습니다. X'0C' 문자(용지 공급)는 또한 데이터 스트림에서 제거될 책임이 있습니다. BMS에 의해 인식되고 변경될 수 있는 문자를 포함하는 투명성 순서를 포함하여 데이터 스트림을 전송하고 싶으면 권장되는 방법은 BMS가 아니라 터미널 제어 SEND 명령을 사용하는 것입니다.

SCS 입력

SCS 프린터는 프로그램 어텐션 키 양식으로 입력 기능이 제한되어 있습니다.

그러나 이러한 키는 769 페이지의 『3270 터미널에서의 입력』에서 설명된 어텐션 키와 같지 않습니다. 대신에 'APAK *nn*' 문자로 구성된 형식화되지 않은 데이터 스트림을 전송합니다. 여기서 *nn*은 2자리 PA 키 번호입니다(예를 들어, PA 키 1의 경우 'APAK 01').

'APAK' 트랜잭션을 정의하여 이러한 입력을 캡처할 수 있습니다(SCS 입력은 다른 PA 키 입력처럼 보이지 않기 때문에 APAK는 TASKREQ 속성 값이 아닌 트랜잭션 ID임). 이 트랜잭션에 의해 호출된 프로그램은 RECEIVE를 실행하여 눌린 PA 키와 입력의 숫자 위치를 판별할 수 있습니다.

CICS에서 프린터 사용

다음을 사용하여 CICS 프린터와 비CICS 프린터에 대한 프로그래밍 방법을 보여줄 수 있습니다.

CICS 프린터의 특성 결정하기

CICS 프린터의 둘 이상의 유형을 지원하는 프로그램을 쓰는 경우에는 특정 프린터의 특성을 판별해야 할 수도 있습니다. 이 목적을 위해 ASSIGN과 INQUIRE TERMINAL 명령을 사용할 수 있습니다.

프린터에 특정적인 몇몇을 포함하여 터미널에 적용되는 ASSIGN 옵션은 739 페이지의 표 37에 표시됩니다.

프린터에 적용되는 INQUIRE TERMINAL 옵션 및 터미널 정의에서 해당하는 매개변수는 표 45에 표시됩니다.

표 45. 프린터의 INQUIRE TERMINAL 옵션

INQUIRE 옵션	TERMINAL 또는 TYPETERM 정의의 소스	설명
PAGEHT	PAGESIZE(x,y)의 x	페이지당 행 수(대체 화면 크기 터미널의 경우 기본 크기를 반영함)
PAGEWD	PAGESIZE(x,y)의 y	행당 문자 수(대체 화면 크기 터미널의 경우 기본 크기를 반영함)
DEFPAGEHT	PAGESIZE(x,y)의 x	기본 모드에서 페이지당 행 수(대체 화면 크기 터미널만 해당)
DEFPAGEWD	PAGESIZE(x,y)의 y	기본 모드에서 행당 문자 수(대체 화면 크기 터미널만 해당)
ALTPAGEHT	ALTPAGE(x,y)의 x	대체 모드에서 페이지당 행 수(대체 화면 크기 터미널만 해당)
ALTPAGEWD	ALTPAGE(x,y)의 y	대체 모드에서 행당 문자 수(대체 화면 크기 터미널만 해당)
DEVICE	DEVICE	디바이스 유형(가능한 값은 시스템 프로그램 개발의 시스템 프로그래밍 명령 소개 참조)
TERMMODEL	TERMMODEL	터미널의 모델 번호(1 또는 2)

BMS 페이지 크기, 3270 프린터

BMS는 터미널 정의와 CICS 프린터의 페이지 크기를 판별하게 위해 실행 중인 트랜잭션의 프로파일 둘 모두를 사용합니다. 프로파일은 터미널에 대체 화면 크기 기능이 있을 때 기본 또는 대체 크기를 사용할지 여부를 판별하는 데 사용됩니다. CICS에서 기본 프로파일 화면의 "기본" 크기를 지정합니다. 다음 테이블은 사용된 값을 나열합니다.

표 46. BMS 페이지 크기를 정의하는 매개변수의 우선순위. BMS는 터미널 정의에 지정된 적합한 열에서 첫 번째 값을 사용합니다.

대체 크기를 사용하는 대체 화면 크기가 있는 터미널	기본 크기를 사용하는 대체 화면 크기가 있는 터미널	대체 화면 크기 기능이 없는 터미널
ALTPAGE	PAGESIZE	PAGESIZE
ALTSCREEN	DEFSCREEN	TERMMODEL
DEFSCREEN	TERMMODEL	(12,80)
TERMMODEL	(12,80)	
(12,80)		

"페이지"의 정의는 BMS에 고유합니다. 터미널 제어 SEND 명령을 사용하여 인쇄 중인 경우에는 프린터 형식에 의해 디바이스의 물리적 제한 내에 페이지를 구성하는 것을 정의합니다. 한 번에 얼마나 많은 데이터를 보낼 수 있는지를 판별하기 위해 버퍼 크기

를 알아야 한다면 ASSIGN 명령이 리턴하는 SCRNNHT 및 SCRNNWD 값으로부터 이를 판별할 수 있습니다.

다중 프린터 유형 지원

서로 다른 페이지 크기가 있는 프린터를 지원하기 위해 프로그램을 쓰고 있는 경우에는 프로그램을 벗어난 페이지 크기와 같은 디바이스 종속성을 유지하는 것이 항상 가능하지는 않습니다. 그러나 BMS는 두 가지 방법으로 이 문제를 돕습니다.

1. 맵을 일반적으로 참조하고 BMS가 사용자 태스크와 연관된 터미널에 설계된 맵을 선택하게 할 수 있습니다.
2. SEND TEXT를 사용 중인 경우에는 BMS는 텍스트를 수신 터미널의 페이지 크기에 따라 단어 경계에서 행으로 분리합니다. 각 페이지에서 헤더와 트레일러 텍스트를 요청할 수도 있습니다.

관련 개념:

933 페이지의 『디바이스 종속 맵』

맵 필드의 위치, 기본 속성, 기본 콘텐츠는 물리적 맵핑에만 나타나고 기호 맵에는 나타나지 않기 때문에 단일 프로그램을 사용하여 동일한 변수 정보를 포함하지만 화면의 배열마다 다른 상수 정보를 포함하는 맵을 빌드할 수 있습니다.

CICS 프린터 사용

다음 메소드를 사용하여 CICS 프린터에 인쇄할 수 있습니다. START 명령으로 인쇄, 트랜지언트 데이터로 인쇄 또는 BMS 라우팅으로 인쇄합니다.

이 태스크 정보

앞서 언급했듯이, 인쇄에서 자주 발생하는 두 번째 문제는 프린터의 소유권과 관계가 있습니다. 인쇄 요청은 종종 표시장치 터미널에 있는 사용자로부터 시작됩니다. 요청을 처리하고 인쇄된 출력을 생성하는 태스크는 사용자의 터미널과 연관되어 있으므로 출력을 프린터로 직접 전송할 수 없습니다.

태스크가 사용하려는 프린터를 소유하지 않는 경우에는 다음 메소드 중 하나를 사용하여 이를 소유하는 또 다른 태스크를 작성해야 작업을 수행할 수 있습니다.

1. START 명령으로 태스크를 작성합니다.
2. 태스크를 트리거하는 리전 내부 트랜지언트 데이터 큐에 씁니다.
3. BMS ROUTE 명령에서 출력을 프린터로 지시합니다.
4. 화면 복사만이 필요한 경우에는 ISSUE PRINT 명령을 사용하십시오.

START 명령으로 인쇄

인쇄 태스크를 작성하기 위한 첫 번째 기술은 인쇄하려는 태스크에서 START 명령을 실행하는 것입니다. 명령은 프린터를 TERMID 옵션에서 STARTed 태스크에 의해 필요로 하는 터미널로 이름을 지정하고, FROM 옵션에서 인쇄될 데이터 또는 이를 찾기 위한 지시사항을 전달합니다.

이 태스크 정보

START는 CICS가 프린시פל 기능이 전용 터미널인(해당 터미널이 사용 가능할 때) 태스크를 작성하게 합니다.

사용자가 반드시 제공해야 하는 STARTed 태스크에 의해 실행되는 프로그램은 RETRIEVE 명령을 사용하여 인쇄될 데이터를 검색한 다음 이를 SEND, SEND MAP 또는 SEND TEXT를 사용하여 터미널(프린터)에 씁니다. 예를 들어, 다음과 같습니다.

```
(build output in OUTAREA, formatted as expected by the STARTed task)
EXEC CICS START TRANSID(PRNT) FROM(OUTAREA) TERMID(PRT1)
      LENGTH(OUTLNG) END-EXEC.
```

그림 136. 인쇄하려는 태스크(프린터 PRT1에서)

```
EXEC CICS RETRIEVE INTO(INAREA) LENGTH(INLNG) END-EXEC.
(do any further data retrieval and any formatting required)
EXEC CICS SEND TEXT FROM(INAREA) LENGTH(INLNG) ERASE PRINT END-EXEC.
(repeat from the RETRIEVE statement until a NODATA condition arises)
```

그림 137. STARTed 태스크(트랜잭션 PRNT 실행)

프린터와 연관된 태스크는 또 다른 태스크가 현재 인쇄가 완료되기 전에 동일 프린터에 데이터를 전송하는 경우 여기에 전송된 모든 데이터가 소진될 때까지 반복됩니다. 이를 수행하면 CICS에 이전 태스크가 여전히 인쇄 중일 때 도착하는 출력할 새 태스크를 작성해야 하는 부담을 줄여줍니다. CICS는 처리되지 않은 START 요청이 있는 한 프린터에 새 태스크를 작성하므로 최종적으로 인쇄되는 내용이 변경되지는 않습니다.

트랜지언트 데이터로 인쇄

인쇄 태스크를 작성하기 위한 두 번째 방법은 트랜지언트 데이터와 관련됩니다. CICS 리전 내부 트랜지언트 데이터 큐는 "트리거"라는 특성을 가지도록 정의될 수 있습니다. 트리거가 있는 큐에서 항목 수가 트리거 값에 도달하면 CICS는 큐를 처리하기 위한 트랜잭션을 작성합니다.

이 태스크 정보

큐 정의는 CICS에 이 태스크가 실행하는 트랜잭션, 프린시פל 기능으로서 필요로 하는 터미널(있는 경우)이 무엇인지를 알려줍니다.

이 매커니즘을 사용하여 인쇄 데이터를 생성하는 태스크로부터 프린터를 소유하는 태스크로 가져올 수 있습니다. 트랜지언트 데이터 큐는 이 방법으로 출력을 지시하는 각 프린터에 정의됩니다. 인쇄하려는 태스크는 출력을 필수 프린터와 연관된 큐에 놓습니다 (WRITEQ TD 명령 사용). 큐와 프린터에 충분한 항목이 사용 가능하면 CICS는 큐를 처리하기 위한 태스크를 작성합니다. (이 용도의 경우 "충분한"의 트리거 레벨은 일반적으로 한 항목으로서 정의됩니다.) 트리거된 태스크는 큐에서 출력을 수신하고 (READQ TD 명령 사용) SEND, SEND MAP 또는 SEND TEXT 명령을 사용하여 이를 프린시플 기능(프린터)에 씁니다.

STARTed 프린터 태스크의 경우에서처럼 트리거되는 태스크가 실행하는 프로그램을 제공해야 합니다. CICS로 배포된 샘플 프로그램에는 "주문 큐 인쇄 샘플 프로그램"이라는 이러한 프로그램의 완전한 예제를 포함합니다. *CICS 4.1 샘플 애플리케이션 안내서*에서는 이 프로그램을 자세히 설명하지만 기본사항은 다음과 같습니다.

인쇄하려는 태스크(프린터 PRT1에서)::

이 예제는 프린터 PRT1에 사전 정의된 기호 대상인 'PRT1'이라고 불리는 트랜지언트 데이터 큐에 데이터를 쓰기 위한 구문을 보여줍니다.

```
(do any formatting or other processing required)
EXEC CICS WRITEQ TD QUEUE('PRT1') FROM(OUTAREA)
      LENGTH(OUTLNG) END-EXEC.
```

트리거되는 태스크::

이 예제는 인쇄 태스크가 동일 코드가 모든 큐(프린터)에 대해 작동할 수 있도록 하드 코딩된 값 대신 ASSIGN 명령을 사용하여 큐의 이름을 판별하는 방법을 보여줍니다. START 대응부처럼 이 태스크는 QZERO 조건을 발견할 때까지 읽기 및 전송 순서를 처음부터 끝까지 수행하여 큐가 비어 있음을 표시합니다.

```
EXEC CICS ASSIGN QNAME(QID) END-EXEC.
EXEC CICS READQ TD QUEUE(QID) INTO(INAREA) LENGTH(INLNG)
      RESP(RESPONSE) END-EXEC.
IF RESPONSE = DFHRESP(QZERO) GO TO END-TASK.
(do any error checking, further data retrieval and formatting required)
EXEC CICS SEND FROM(INAREA) LENGTH(INLNG) END-EXEC.
(repeat from READQ command)
```

이는 STARTed 태스크의 효율성 문제일 뿐이지만 트랜지언트 데이터에는 중요합니다. 그렇지 않으면 처리되지 않은 큐 항목이 특정 조건 하에 축적될 수 있습니다. (트리거를 사용하여 트랜지언트 데이터 큐를 처리하는 태스크의 작성에 대한 세부사항은 809 페이지의 『ATI(Automatic Transaction Initiation)』의 내용을 참조하십시오.)

이 기술을 사용하면 하나의 단위로 인쇄될 출력이 큐에 단일 항목이나 연속적 항목으로 나타나는지 확인해야 합니다. 큐 항목과 인쇄된 출력 간에는 고정된 관계가 없습니다. 패키지 배열은 엄밀하게 큐를 쓰는 프로그램과 이를 읽는 프로그램 사이에 있습니다.

다. 그러나, 태스크가 함께 인쇄되어야 하는 여러 항목을 쓰는 경우에는 완료되기 전에 다른 태스크가 큐에 쓰지 않도록 해야 합니다. 그렇지 않으면 여러 태스크에서 인쇄된 출력이 끼어들 수 있습니다.

TD 큐가 복구 가능으로 정의된 경우에는 CICS는 끼워넣기를 금합니다. 태스크가 복구 가능한 큐에 쓴 후에는 CICS는 첫 번째 태스크가 확약하거나 작성된 내용을 제거 할 때까지(SYNCPPOINT 사용 또는 태스크 종료 시) 쓰기를 원하는 다른 태스크를 지연합니다. 큐가 복구 불가능한 경우에는 이 함수를 스스로 수행해야 합니다. 한 가지 방법은 첫 번째 큐 항목을 쓰기 전에 ENQUEUE를 실행하고 마지막을 쓴 후에 DEQUEUE를 실행하는 것입니다. (트랜지언트 데이터 큐에 대한 설명은 807 페이지의 제 38 장 『트랜지언트 데이터 제어』의 내용을 참조하십시오.)

BMS 라우팅으로 인쇄

태스크는 또한 BMS 라우팅을 사용하여 해당 프린시פל 기능이 아닌 프린터에서 출력을 얻을 수 있습니다. 이 기술은 BMS 논리 메시지(ACCUM 또는 PAGING 옵션)에만 적용되므로 이미 논리 메시지를 빌드 중인 경우에 가장 적합합니다.

이 태스크 정보

라우팅된 메시지를 완료할 때 CICS는 라우팅 목록에 이름이 지정된 각 터미널에 대해 태스크를 작성합니다. 이 태스크는 터미널을 프린시פל 기능으로 가지고 있고, 페이지를 표시하기 위한 CICS 제공 트랜잭션인 CSPG를 사용하여 출력을 프린터로 전달합니다. 라우팅은 START 명령을 사용하는 것과 효과가 유사하지만 CICS는 인쇄를 수행하는 프로그램을 제공합니다. (라우팅에 대한 자세한 정보는 987 페이지의 『메시지 라우팅』의 내용을 참조하십시오.)

비CICS 프린터 사용

여기에 나열된 단계는 CICS 외부에서 관리되는 프린터에서 출력을 포맷하고, 출력을 전달하고, 출력이 인쇄 준비가 될 때 애플리케이션에 알리는 방법을 보여줍니다.

프로시저

1. 사용하려는 프린터를 제어하는 애플리케이션 또는 서브시스템이 필요로 하는 방법으로 출력을 포맷하십시오.
2. 출력을 해당 애플리케이션이 필요로 하는 양식으로 애플리케이션이나 프린터를 제어하는 서브시스템으로 전달하십시오.
3. 필요한 경우 출력이 인쇄 준비가 되었음을 애플리케이션에 알리십시오.

비CICS 프린터 포맷

CICS 외부에서 관리되는 일부 프린터의 경우 BMS로 출력을 포맷할 수 있습니다. 그러나 대부분의 프린터의 경우 프린터를 구동하는 애플리케이션의 형식 요구사항을 충족해야 합니다.

BMS로 출력 포맷은 『비CICS 프린터를 위한 프로그래밍』에 설명되어 있습니다. 프린터를 구동하는 애플리케이션의 형식 요구사항은 디바이스 형식이거나 애플리케이션이 지시하는 중간 양식일 수 있습니다. 기존의 행 프린터의 경우 포맷팅은 행 이미지를 생성하는 문제이고, 종종 캐리지 제어 문자를 추가합니다.

비CICS 프린터: 데이터 전달

인쇄 데이터는 일반적으로 데이터를 CICS 및 애플리케이션 둘 모두에 액세스 가능한 중간 파일에 배치하는 방법으로 CICS의 외부에 있는 애플리케이션에 전달됩니다. 파일 내의 형식뿐만 아니라 파일의 유형은 수신 애플리케이션이 결정합니다.

일반적으로 표 47의 첫 번째 열에 나열된 파일 유형 중 하나입니다. 테이블의 두 번째 열은 이러한 데이터를 작성하는 데 어떤 CICS 명령 그룹을 사용할 수 있는지를 보여줍니다.

표 47. 인쇄 데이터를 비CICS 프린터로 전송하기 위한 중간 파일

파일 유형	데이터를 쓰기 위한 메소드
스풀 파일	CICS 스푼 명령(SPOOLOPEN, SPOOLWRITE 등등.) 트랜지언트 데이터 명령(WRITEQ TD) 터미널 제어 및 BMS 명령(SEND, SEND MAP 등등.)
BSAM	CICS 스푼 명령(SPOOLOPEN, SPOOLWRITE 등등.) 트랜지언트 데이터 명령(WRITEQ TD)
VSAM	CICS 파일 제어 명령(WRITE)
DB2	EXEC SQL 명령
IMS	EXEC DLI 명령 또는 CALL DLI 문

비CICS 프린터를 위한 프로그래밍

VSAM, DB2 또는 IMS를 사용 중인 경우에는 사용할 수 있는 CICS 애플리케이션 프로그래밍 언어는 사용 중인 파일 유형에 의해 결정됩니다. 그러나 BSAM 및 스푼 파일의 경우에는 선택사항이 있습니다. 파일의 CICS 정의(또는 부재)는 어떤 명령을 사용하는지를 판별합니다.

파일은 다음과 같을 수 있습니다.

- 리전 외부 트랜지언트 데이터 큐(트랜지언트 데이터 큐에 대한 자세한 정보는 807 페이지의 제 38 장 『트랜지언트 데이터 제어』의 내용을 참조하십시오.)
- 순차적인 터미널의 출력의 절반(747 페이지의 제 31 장 『순차 터미널 지원 사용』 및 931 페이지의 『비3270 터미널에 대한 지원』의 내용을 참조하십시오.)
- 스푼 파일(895 페이지의 『JES에 대한 CICS 인터페이스』의 내용을 참조하십시오.)

트랜지언트 데이터 큐 정의와 순차 터미널 정의는 둘 모두 CICS 시작 JCL에서 연관된 데이터 정의(DD) 문을 가리키고 파일이 BSAM 파일인지 또는 스푼 파일인지 여부를 판별하는 것은 이 DD 문입니다. CICS 스푼 명령에 의해 작성된 파일은 사용 전에 정의를 필요로 하지 않고 정의상 스푼 파일입니다.

인쇄 애플리케이션이 BSAM 또는 스푼 파일 입력을 승인하는 경우에는 파일을 CICS에 정의하는 방법을 결정할 때 몇몇 요소를 고려해야 합니다.

시스템 정의

SPOOLOPEN 명령에 의해 작성된 파일은 CICS 또는 운영 체제에 정의될 필요가 없습니다. 반면, 트랜지언트 데이터 큐 및 순차적인 터미널은 사용 전에 둘 모두에 정의되어야 합니다.

태스크 사이에 공유

트랜지언트 데이터 큐로 정의된 파일은 모든 태스크 사이에 공유됩니다. 이를 통해 여러 태스크에서 인쇄 파일을 작성할 수 있지만 태스크가 함께 인쇄되어야 하는 큐(예를 들어, 단일 보고서의 인쇄 행)에 여러 레코드를 쓰는 경우에는 다른 태스크가 사용자의 태스크 사이에 레코드를 쓰는 것을 막기 위해서 인 큐 논리를 포함해야 합니다. 이는 888 페이지의 『트랜지언트 데이터로 인쇄』에서 리전 내부 큐에 설명된 것과 동일한 요구사항입니다. 그러나 리전 외부 트랜지언트 데이터의 경우 CICS는 복구성 솔루션을 제공하지 않고 프로그램은 배치 자체를 방지해야 합니다.

반대로, SPOOLOPEN에 의해 작성된 파일은 이를 작성한 태스크에 의해서만 작성될 수 있습니다. 이는 인터리빙 출력의 위험을 제거하고 또한 태스크 사이에 파일을 공유를 방지합니다.

순차적 터미널과 연관된 스푼 파일은 한 번에 하나의 태스크(터미널을 프린시 펄 기능으로 가지고 있는 태스크)에 의해서만 작성될 수 있습니다. 이는 또한 인터리빙을 방지하지만 태스크가 파일을 연속적으로 공유할 수 있게 허용합니다.

인쇄를 위해 릴리스

운영 체제가 BSAM 및 스푼 파일을 CICS에서 수신 애플리케이션으로 전달하기 위해서는 이 둘 모두가 닫혀 있어야 하므로 인쇄는 연관된 파일이 닫히기 전까지는 시작되지 않습니다. SPOOLOPEN에 의해 작성된 파일은 SPOOLCLOSE 명령으로 이미 닫히지 않는 한 태스크 종료 시에 자동으로 닫힙니다. 반대로, 리전 외부 트랜지언트 데이터 큐는 몇몇 태스크가 SET 명령을 사용하여 이를 명시적으로 닫기 전까지는 열려 있습니다. (나중에 사용하려는 경우에는 또 다른 SET를 사용하여 다시 열어야 합니다.) 따라서 트랜지언트 데이터는 추가 프로그래밍을 희생하여 처리하려는 파일의 릴리스에 대한 더 많은 제어 권한을 제공합니다.

순차적 터미널의 출력을 표시하는 파일은 CICS가 시스템 종료되기 전까지는 자동으로 닫히지 않고 인쇄를 위해 릴리스되지 않으므로 CICS는 이전에 이를 닫기 위한 기능을 제공하지 않습니다. BMS를 사용하기 위해 수행하는 것처럼, 순차적 터미널을 사용하여 데이터를 CICS 외부에서 제어되는 프린터로 전달하는 경우에는 이 제한사항에 대해서 알고 있어야 합니다.

형식화 파일을 순차 터미널로 정의하는 경우에는 BMS를 사용하여 출력을 포맷할 수

있습니다. 이 기능을 사용하면 CICS 표시장치 및 프린터 터미널에 사용하는 CICS 외부에서 관리되는 프린터에도 동일한 맵을 사용할 수 있습니다. 예를 들어, MVS 작업 항목 서브시스템(JES)이 관리하는 행 프린터입니다.

이 옵션을 선택하는 경우에는 BMS가 항상 터미널 정의의 페이지 크기를 사용하여 한 번에 한 페이지의 출력을 전송한다는 점과 순차 터미널의 출력을 표시하는 데이터 세트가 CICS 시스템 종료 전에는 릴리스되지 않는다는 점을 기억하십시오.

스풀 파일 한계

운영 체제는 순차 번호를 지정하여 스푼 파일을 식별합니다. 이 수에는 상한선이 있고, 이 수 뒤에는 재사용됩니다. 이 한계는 일반적으로 크지만 장 시간(CICS가 할 수 있는 것처럼) 실행되고 대량의 스푼 파일을 작성하는(CICS 아래에서 애플리케이션이 할 수 있는 것처럼) 작업은 한계를 초과할 수 있습니다. 대량의 스푼 파일을 생성하는 애플리케이션을 쓰는 경우에는 시스템 프로그램에 문의하여 시스템 한계 내에 있는지 확인하십시오. 새 스푼 파일이 각 SPOOLOPEN 문 및 스푼 파일로 정의되는 트랜지언트 데이터 큐의 각 열기에 작성됩니다.

인쇄 애플리케이션 알림

데이터를 CICS 외부의 인쇄 애플리케이션에 전달할 때에는 처리할 준비가 완료된 데이터가 있음을 애플리케이션에 알려야 합니다.

애플리케이션이 자동으로 실행되고 데이터를 찾아야 함을 알고 있으면 이를 수행할 필요가 없습니다. 예를 들어, MVS 작업 입력 시스템(JES)이 소유하는 프린터에서 인쇄하려면 해야 할 일은 적합한 라우팅 정보를 사용하여 스푼 파일을 작성하는 것입니다. JES는 나머지를 수행합니다.

그러나 종종 처리를 수행하기 위해 작업을 제출해야 합니다. 그렇지 않으면 실행 애플리케이션에 작업이 있음을 알리십시오.

CICS 태스크에서 일괄처리 작업을 제출하려면 작업의 JCL을 포함하는 스푼 파일을 작성해야 하고 이 파일을 JES 내부 리더로 지시해야 합니다. 891 페이지의 표 47에서 스푼 파일에 나열된 세 가지 방법 중 하나를 사용하여 파일을 작성할 수 있습니다. 비록 순차 터미널을 사용하더라도 작업은 앞서 설명된 대로 CICS가 시스템 종료하기 전까지는 실행되지 않습니다. 스푼 명령으로 작성된 파일의 경우 파일을 JES 내부 리더로 라우팅하는 정보는 SPOOLOPEN 명령에 지정됩니다. 트랜지언트 데이터 큐 및 순차 터미널의 경우 라우팅 정보는 파일의 첫 번째 레코드, "JOB 카드"에 나타납니다.

인쇄될 출력은 일괄처리 작업에 입력으로서 임베드되거나 작업이 승인하는 데이터 저장 영역의 양식을 통해 개별적으로 전달될 수 있습니다.

표시장치 화면 인쇄

인쇄 요구사항이 표시장치 화면을 프린터에 복사하는 것이면 이미 설명된 것 외에도 추가로 선택사항이 있습니다. 이들 중 일부는 터미널 하드웨어 자체가 제공하며 일부는 CICS가 제공합니다.

CICS 지원의 일부는 또한 하드웨어 기능에 달려 있고 따라서 옵션도 관련된 터미널의 유형에 달려 있고, 어떤 경우에는 CICS에 정의되는 방법에 달려 있습니다. 자세한 정보는 참조 -> 시스템 정의에서 TERMINAL 자원의 내용을 참조하십시오.

CICS 인쇄 키

이러한 첫 번째 옵션은 *CICS 인쇄 키*(로컬 복사 키라고도 불림)입니다. 이 옵션을 사용하면 터미널이 3270 표시장치 또는 3270 호환 모드에 있는 표시장치인 경우, 프로그램 어텐션 키를 눌러서 화면의 인쇄된 사본을 요청할 수 있습니다. 인쇄 키 지원은 CICS에서는 선택적입니다. 시스템 프로그래머는 이를 포함할지 여부 및 어떤 키가 지정되는지를 설명합니다. 기본값은 PA1입니다. 참조 -> 시스템 정의에서 결합된 정보 목록에 대한 DISPLAY 및 PRINT 옵션에서 PRINT 옵션을 참조하십시오.

인쇄 키는 표시장치 화면을 적합한 것으로 정의된 프린터들 중에서 사용 가능한 첫 번째 프린터로 복사합니다. 어떤 프린터가 적합한지는 다음과 같이 요청이 시작되는 표시장치 터미널의 정의에 따라 다릅니다.

- "프린터-어댑터" 기능 없이 정의된 z/OS Communications Server 3270 표시장치의 경우 터미널 정의의 PRINTER 및 ALTPRINTER 옵션에 이름 지정된 프린터가 적합합니다. PRINTER는 사용 가능한 경우 사용됩니다. ALTPRINTER는 두 번째 선택사항입니다. 둘 모두 사용 불가능한 경우에는 PRINTER가 사용 가능하게 될 때 요청이 실행을 위해 큐 처리됩니다.
- 3790의 3270 호환 모드와 3650 호스트 대화(3270) 논리 장치의 경우 동일한 선택사항이 적용됩니다.
- 프린터-어댑터 기능이 정의된 z/OS Communications Server 3270 표시장치의 경우 복사는 표시장치와 동일한 제어 장치에 있는 프린터로 제한됩니다. 제어 장치 내의 프린터 권한 부여 매트릭스는 프린터 적합성을 판별합니다.
- 프린터-어댑터 기능이 있는 3790의 3270 호환 모드 논리 장치의 경우 3790은 적합성을 판별하고 복사를 위한 프린터를 할당합니다.
- 프린터-어댑터 기능이 있는 3275의 경우, 인쇄 키는 3275 표시장치 버퍼에 현재 있는 데이터를 표시장치에 첨부된 3284에 인쇄합니다.

여기서 CICS는 위의 처음 세 번째 경우에서처럼 프린터를 명시적으로 선택하고, 프린터는 CICS 인쇄 키 요청에 "사용 가능"하려면 서비스에 있어야 하고 태스크에 첨부되지 않아야 합니다. 여기서 제어 장치 또는 서브시스템은 할당을 하고, 가용성 및 상태는 서브시스템이 결정합니다. 디바이스의 대괄호 상태는 일반적으로 사용 가능한지 여부를 판별합니다.

ISSUE PRINT 및 ISSUE COPY

애플리케이션은 **ISSUE PRINT** 및 **ISSUE COPY** 명령을 사용하여 화면을 사용자뿐만 아니라 프린터에 복사하기를 시작할 수 있습니다. **ISSUE PRINT**는 CICS 인쇄 키를 누르는 사용자를 시뮬레이트하고, 프린터 적합성과 가용성은 CICS 인쇄 키 요청에서와 동일합니다.

ISSUE COPY 명령을 사용하면 복사할 터미널을 소유하는 태스크와는 반대로, 프린터를 소유하는 태스크에서 화면을 복사할 수 있습니다. 이는 TERMID 옵션에 이름이 지정된 터미널의 버퍼를 태스크를 실행하는 프린시플 기능의 버퍼로 복사합니다. 복사 방법과 복사가 발생한 후의 인쇄 시작은 **ISSUE COPY** 명령의 CTLCHAR 옵션에 정의된 "복사 제어 문자"에 의해 제어됩니다. 이 제어 문자에서 비트 설정은 IBM 3270 Data Stream Device Guide의 내용을 참조하십시오. 버퍼가 복사되는 터미널과 프린터는 둘 모두 3270 논리 장치여야 하며 이들은 동일한 제어 장치에 있어야 합니다.

하드웨어 인쇄 키

일부 3270 터미널에는 또한 *하드웨어 인쇄 키*가 있습니다. 이 키를 누르면 화면이 표시장치와 같은 제어 장치에 있는 사용 가능하고 적합한 첫 번째 프린터로 복사됩니다. 이 기능은 전적으로 제어 장치에 의해 수행되며, 이의 구성 및 터미널 상태 정보는 적합성과 가용성을 결정합니다. 사용 가능한 프린터가 없는 경우에는 요청이 실패합니다. 사용자는 화면의 왼쪽 하단 구석에서 기호로 알림을 받게 되고 나중에 요청을 재시도해야 합니다.

BMS 화면 복사

CICS 및 하드웨어 인쇄 키 둘 모두는 화면 복사를 사전 정의된 적합한 프린터 세트로 제한하고, 둘 이상의 프린터가 적합한 경우에는 선택은 다른 태스크의 프린터 사용에 달려 있습니다. BMS 논리 메시지의 일부로서 작성된 화면의 경우 더 많은 일반 화면 복사 기능을 사용 가능합니다. 사용자는 논리 메시지, CSPG를 표시하기 위해 CICS 트랜잭션의 "페이지 복사" 옵션을 사용하여 이러한 화면을 인쇄할 수 있습니다. 페이지 복사를 사용하면 출력을 받을 특정 프린터의 이름을 지정할 수 있고, 이는 표시장치와 동일한 제어 장치에 있을 필요는 없습니다. CSPG에 대한 자세한 정보는 참조 > 시스템 정의에서 CSPG - 페이지 검색의 내용을 참조하십시오.

JES에 대한 CICS 인터페이스

CICS는 CICS 애플리케이션이 스푼 파일을 작성하고 검색할 수 있게 허용하는 JES(MVS의 Job Entry Subsystem 구성요소)에게 프로그래밍 인터페이스를 제공합니다.

스풀 파일은 JES에 의해 관리되고, 작성하는 작업과 디바이스에 의한 실제 처리 프로세스 사이에 저속 주변 디바이스(프린터, 펀치 및 플로터)에 관한 버퍼 출력에 익숙해져 있습니다. 카드 리더로부터의 입력 파일 또한 스푼 파일이고 디바이스와 데이터를 사용하는 작업 간의 버퍼의 역할을 합니다.

인터페이스는 5개의 명령으로 구성됩니다.

- SPOOLOPEN INPUT은 입력할 파일을 엽니다.
- SPOOLOPEN OUTPUT은 출력할 파일을 엽니다.
- SPOOLREAD는 입력 파일에서 다음 레코드를 검색합니다.
- SPOOLWRITE는 한 레코드를 출력 파일에 추가합니다.
- SPOOLCLOSE는 파일을 닫고 JES가 후속 처리할 수 있도록 릴리스합니다.

여기서 "입력" 및 "출력"은 여기에서 CICS 관점을 가리킵니다. 스푼이 하나의 작업에 출력하는 내용은 항상 또 다른 작업 또는 JES 프로그램에 대한 스푼 입력입니다.

이러한 명령은 JES의 JES2 또는 JES3 양식과 함께 사용될 수 있지만, 각각에 약간의 제한이 적용되기도 합니다(898 페이지의 『스풀 인터페이스 제한사항』 참조). JES 용어는 둘 모두를 가리킵니다. JES 원격 스푼링 통신 서브시스템(RSCS) 네트워크를 통해 연결된 다른 시스템과의 파일을 교환하기 위해 다른 제품의 요구사항을 지원할 수 있습니다.

스풀 명령을 사용하여 다음 유형의 작업을 수행할 수 있습니다.

- 인쇄 또는 JES에 의한 다른 처리를 위해 출력 파일을 작성합니다. JES는 고속 프린터 및 카드 리더를 포함하여 운영 체제의 "단위 레코드" 기능의 대부분을 관리합니다. 이러한 기능을 사용하려면 처리할 데이터를 스푼 파일에서 JES로 전달합니다. 그림 138의 내용을 참조하십시오.

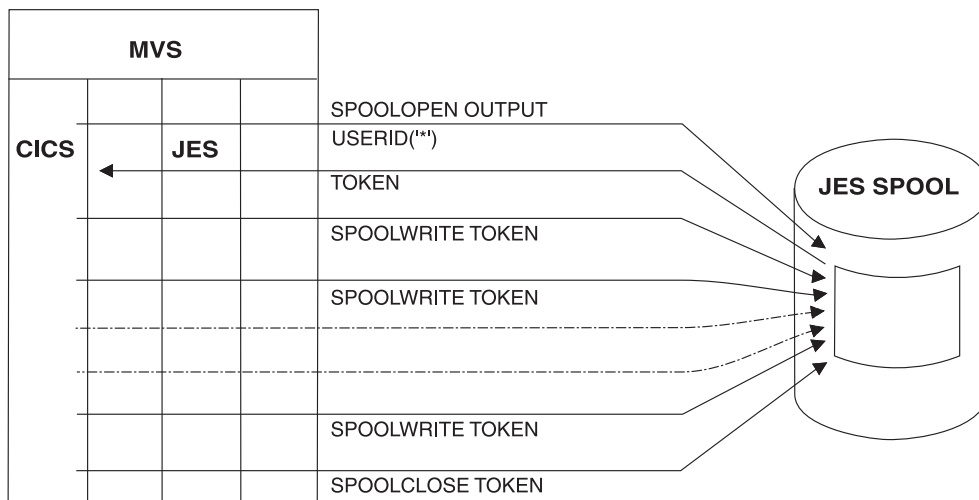


그림 138. 파일을 작성하고 이를 JES 스푼에 쓰기

- 일괄처리 작업을 MVS에 제출합니다. JES "터미널 리더"에 지시된 스푼 파일은 완벽한 작업으로 처리되고 실행됩니다.
- 데이터를 MVS 아래에서 실행하는 또 다른 작업(CICS 외부)으로 전달하기 위해 (출력) 파일을 작성합니다.
- 이러한 작업으로부터 전달된 데이터를 검색합니다. 그림 139의 내용을 참조하십시오.

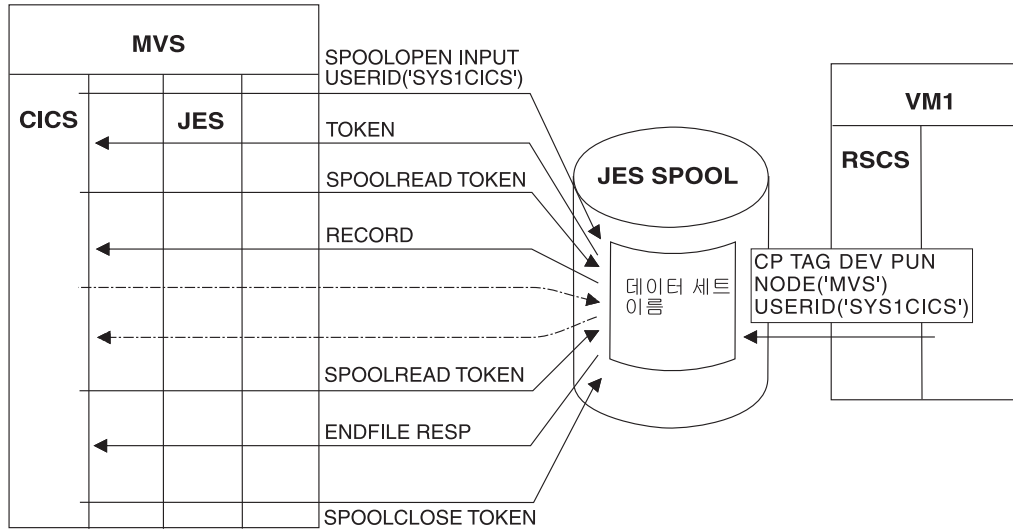


그림 139. JES 스푼로부터 데이터 검색

- CICS가 실행 중인 시스템이 아닌 데이터를 VM, VSE/ESA 또는 MVS 시스템 등과 같은 또 다른 운영 체제로 전달하기 위한 파일을 작성합니다. 그림 140의 내용을 참조하십시오.

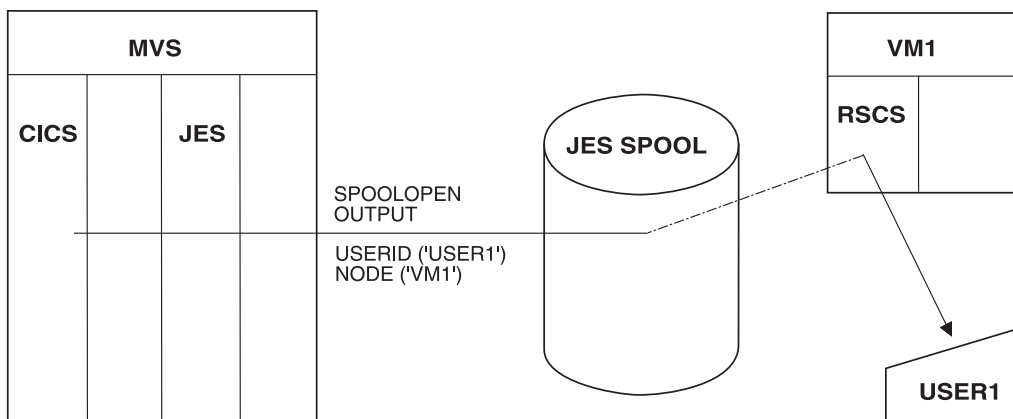


그림 140. 기록된 파일을 원격 대상으로 전송하기

JES에 대한 CICS 인터페이스 사용

JES에 CICS 인터페이스를 사용하려면, DFHSIT SPOOL=YES 시스템 초기화 매개변수를 사용자의 CICS 시동 JCL에서 정의해야 합니다.

이 태스크 정보

EXEC CICS SPOOLCLOSE, SPOOLOPEN, SPOOLREAD과 SPOOLWRITE 명령에 RESP 또는 NOHANDLE를 지정해야 합니다. RESP은 HANDLE CONDITION으로 일대일 응답을 지닙니다. RESP를 코딩하지 않으면 프로그램이 이상 종료됩니다. RESP2 옵션 또한 코딩할 수 있습니다.

INPUT 또는 OUTPUT을 위해 1000개 레코드보다 큰 SYSOUT 데이터 세트를 처리하는 트랜잭션은 나머지 CICS에 성능 영향을 미칠 수 있습니다. 이러한 트랜잭션을 피할 수 없는 경우에는 일반 시스템 성능을 주의깊게 평가해야 합니다. 나머지 CICS에 대한 영향을 받아들이지 않으면 속도 메커니즘을 도입해야 합니다.

JES 스푼 파일에 대한 모든 액세스는 하나의 논리적 작업 단위(LUW) 내에서 완료되어야 합니다. EXEC CICS SYNCPOINT 명령을 실행하면 내재적으로 열린 보고서를 위한 SPOOLCLOSE 명령이 실행됩니다.

스풀 인터페이스 제한사항

JES에는 애플리케이션을 설계할 때 고려해야 하는 내부 한계가 있습니다. 몇몇은 JES2에 적용되고, 몇몇은 JES3 및 몇몇은 둘 모두에 적용됩니다.

특히 다음과 같습니다.

- JES2는 단일 작업(예: CICS)이 작성할 수 있는 총 스푼 파일의 수에 상한선을 부과합니다. CICS가 실행 중에 이 한계를 초과하면 후속 SPOOLOPEN OUTPUT 명령은 ALLOCERR 조건으로 실패합니다.
- JES3는 이러한 한계를 명시적으로 부과하지 않지만 JES2와 JES3 둘 모두의 경우 작성된 각 파일의 몇몇 제어 정보는 CICS의 전체 실행을 위해 지속됩니다. 이러한 이유로 대량의 스푼 파일을 작성하는 것은 JES 자원에 스트레스를 줍니다. 이러한 애플리케이션을 설계하기 전에 시스템 프로그래머에 문의해야 합니다.
- 스푼 파일에는 처리될 때까지 다른 자원(버퍼, 큐 요소, 디스크 공간)이 필요합니다. 대상에서 처리하기 위해 장시간 기다릴 수 있는 대량의 파일을 생성하는 경우에는 시스템 직원에게 문의해야 합니다.
- 로컬 스푼 파일을 지정하고 OUTDESCR 피연산자가 NODE과 USERID 피연산자를 대체할 수 있게 하려면 노드(*)와 userid(*)를 코드화하십시오. NODE(*)를 다른 사용자 ID와 함께 사용하지 마십시오. NODE과 USERID 피연산자가 명시적 ID를 지정하면, OUTDESCR 피연산자는 그들을 대체할 수 없습니다.

- CICS에 의해 생성된 데이터 세트가 JES에 HELD 상태에 없도록 사용자의 시스템이 정의되는지 확인하십시오. EXEC CICS SPOOLOPEN INPUT 명령이 실행될 때 CICS은 HELD 상태에서 데이터 세트를 검색하지 않습니다.

출력 스푼 파일 작성

출력 스푼 파일을 작성하려면, 프로그램은 출력 데이터 세트를 할당하기 위해 SPOOLOPEN OUTPUT 명령을 실행하는 것을 시작됩니다. 레코드는 SPOOLCLOSE 명령이 송출 또는 처리를 위해 JES에 파일을 방출하는데 사용되는 SPOOLWRITE 명령을 사용하여 파일에 추가됩니다.

이 태스크 정보

SPOOLOPEN OUTPUT 명령의 NODE 및 USERID 옵션은 완료될 때 파일로 수행할 작업에 대해 JES에 말하고 해당하는 경우 포매팅과 다른 처리를 JES에 전달하기 위한 다른 옵션이 있습니다. SPOOLOPEN은 TOKEN 필드에서 고유한 토큰을 리턴하고, 이는 기록 중인 파일을 식별하기 위해 모든 후속 SPOOLWRITE 및 SPOOLCLOSE 명령에 사용되어야 합니다.

그 후에 태스크는 SPOOLOPEN OUTPUT 명령에 리턴된 토큰 값을 지정하는 SPOOLWRITE 명령을 사용하여 데이터를 파일에 넣습니다. 스푼 파일은 순차적이고 각 SPOOLWRITE는 파일에 한 레코드를 추가합니다. 파일이 완료되면 태스크는 파일을 식별하는 토큰으로 SPOOLCLOSE를 실행하여 전달 또는 처리를 위해 파일을 JES에 릴리스합니다.

태스크는 여러 출력 스푼 파일을 작성할 수 있고, 한 번에 둘 이상이 열려 있을 수 있습니다. 서로 다른 파일에서의 작업은 토큰에 의해 분리된 상태로 있습니다. 그러나 동일한 태스크에서 논리적 작업 단위(LUW) 전체에 걸쳐 스푼 파일은 태스크 중에 공유될 수 없습니다. 스푼 파일은 이를 연 태스크에 의해서만 기록될 수 있고, 해당 토큰이 SYNCPOINT 명령 또는 태스크 종료 전에 파일을 닫는 데 실패하는 경우에는 CICS는 이 시점에서 이를 자동으로 닫습니다.

노드가 원격 시스템이면 데이터 세트는 대상 사용자 ID에 대해 JES 스푼에서 큐 처리됩니다. 이 대상 사용자의 ID는 SPOOLOPEN OUTPUT USERID 매개변수에 지정되었습니다. 노드가 원격 VM 시스템이면 데이터는 동일한 USERID 매개변수에 지정된 ID의 VM RDR 큐에 큐 처리됩니다.

참고: 사용자가 제출하는 작업이 가능한 빨리 실행되기를 원하면 처음 다섯 문자에 /*E0F를 포함하는 레코드가 있는 스푼 파일을 종료해야 합니다. 이 문은 JES가 다른 레코드가 릴리스 전에 현재 버퍼를 채울 때까지 기다리는 대신에, 처리를 위해 파일을 릴리스하도록 합니다.

MVS 내부 리더 사용

USERID 매개변수를 사용하여 출력이 MVS 내부 리더에 작성되도록 지정할 수 있습니다. 이 용도로 CICS SPOOL 명령을 사용하려면 USERID(『INTRDR』)를 지정하고 또한 명시적인 노드 이름을 사용하십시오. NODE(*)를 사용하지 마십시오. INTRDR 은 내부 리더를 식별하는 IBM이 예약한 이름입니다.

USERID(『INTRDR』)를 지정하는 경우에는 SPOOLWRITE 명령이 쓴 출력 레코드는 JOB 문으로 시작하는 JCL 문이어야 합니다. 또한 SPOOLOPEN 명령에 NOCC 옵션을 지정해야 합니다. 시스템은 내부 리더의 출력 레코드를 주소 공간에 있는 버퍼에 놓습니다. 이 버퍼가 가득 차면 JES는 내용을 스푼에 놓습니다. 나중에 JES는 스푼에서 작업을 검색합니다. 스푼 파일의 이름 지정에 대한 자세한 정보는 902 페이지의 『스폴 파일 식별』의 내용을 참조하십시오.

입력 스푼 파일 읽기

스폴 파일 읽기를 위한 명령 순서는 이를 작성하는 것과 유사합니다. 파일을 선택하는 SPOOLOPEN INPUT 명령으로 시작합니다. 그런 다음 SPOOLREAD 명령으로 각 레코드를 검색합니다. 파일이 고갈되었거나 필요한 만큼 읽은 경우에는 SPOOLCLOSE 명령을 사용하여 처리를 종료합니다.

이 태스크 정보

CICS는 출력 파일을 열 때처럼 특정 파일을 열 때 이를 식별할 수 있는 토큰을 제공하고 이 토큰을 파일에 대한 모든 후속 명령에서 사용할 수 있습니다.

출력 스푼 파일처럼 입력 스푼 파일은 이를 연 태스크에 독점적입니다. 첫 번째 태스크가 이를 닫기 전까지는 다른 태스크는 이를 사용할 수 없습니다. 파일은 이를 연 것과 동일한 논리적 작업 단위(LUW)에서 읽어야 하고 CICS는 SYNCPOINT 명령 시나 태스크가 이를 수행하지 않은 경우 태스크 종료 시에 이를 자동으로 닫습니다. 그러나 태스크 또는 또 다른 태스크가 처음부터 다시 읽을 수 있는 방식으로 파일을 닫을 수 있습니다.

출력 파일과는 달리, 태스크에는 한 번에 단 하나의 스푼 파일만이 입력을 위해 열려 있을 수 있습니다. 게다가, 단 하나의 CICS 태스크에는 지정된 시간 동안 입력을 위해 하나의 파일만이 열려 있을 수 있습니다. 입력 스푼 파일의 이 단일 스레딩에는 여러 프로그래밍 연관성이 있습니다.

- 스폴 파일을 읽는 태스크는 가능한 짧은 시간 동안 열려 있어야 하고, CICS가 태스크 종료 처리의 일환으로서 이를 닫게 하는 대신에 명시적으로 닫아야 합니다. 개별 레코드의 처리가 긴 경우 파일을 또 다른 저장영역의 양식으로 전송하고 싶을 수도 있습니다.

- 또 다른 태스크가 스푼 파일을 읽고 있으면, SPOOLOPEN INPUT 명령은 SPOLBUSY 조건으로 실패합니다. 이는 오류가 아닙니다. 잠시 기다린 후 다시 시도하십시오.
- 여러 입력 파일을 읽는 경우에는 입력 스레드를 독점하고 이를 필요로 하는 다른 태스크를 잠그는 것을 피하기 위해 태스크를 닫고 다음 번을 여는 사이에 태스크를 잠시 지연해야 합니다.

원격 애플리케이션은 CICS 트랜잭션용인 모든 파일을 CICS가 상주하는 시스템에 특정 사용자 이름으로 라우팅해야 합니다. 이를 수행하려면 VM 시스템이 사용하는 CP 명령의 예제는 897 페이지의 그림 139의 내용을 참조하십시오. 그림은 또한 데이터를 검색하는 데 사용하는 EXEC CICS SPOOL 명령을 표시합니다.

CICS 트랜잭션은 SPOOLOPEN 명령을 실행하여, USERID 매개변수와 선택적으로는 기록기 이름 내의 출력 클래스에 기록기 이름을 지정합니다. 정상 응답은 다음과 같습니다.

1. 이 외부 기록기를 위한 입력이 없습니다.
2. 단일 스레드가 사용 중입니다.
3. 파일이 검색을 위해 할당되었고 CICS가 리턴하는 "토큰"으로 식별됩니다. 토큰은 데이터 세트를 검색하기 위한 모든 SPOOL 명령에 포함되어야 합니다.

(1)과 (2)의 경우에 트랜잭션은 자체를 재시작하여 적합한 간격 후에 SPOOLOPEN을 재시작해야 합니다.

(3)의 경우에는 트랜잭션은 SPOOLREAD 명령으로 파일을 검색하고 사용자의 경로를 릴리스하기 위해 가능한 빨리 다른 SPOOLCLOSE로 진행해야 합니다. 이는 특히 입력 경로가 단일 스레드이므로 JES로부터의 입력에 중요합니다. 인터페이스를 사용하는 둘 이상의 트랜잭션이 있을 때에는 파일은 다른 기록기 이름 또는 단일 기록기 이름 내의 다른 클래스를 사용하여 차별화될 수 있습니다. 게다가, 트랜잭션이 종료하는지 또는 SPOOLCLOSE와 후속 SPOOLOPEN 사이의 짧은 기간 동안 기다린다는 점을 확인해야 합니다. 이를 수행하지 않으면 하나의 트랜잭션이 다른 트랜잭션이 인터페이스를 사용하는 것을 막을 수 있습니다.

JES 엑시트

JES2 및 JES3 둘 모두 수신 파일을 검사하는 방법을 제공합니다. JES2의 경우, TSO/E 대화식 데이터 전송 기능 심사 및 알림 엑시트가 사용됩니다. JES3와 동등한 것은 Validate Incoming Netdata File 엑시트입니다.

JES에 대한 CICS 인터페이스를 사용하여 읽힐 파일이 올바르게 처리되는지 확인하려면 이러한 엑시트를 구성하는 설치의 사용을 검토해야 합니다.

스풀 파일 식별

입력 스푼 파일은 SPOOLOPEN INPUT 명령에서 USERID 및 CLASS 옵션으로 식별됩니다.

입력에서 USERID는 JES 외부 기록기의 이름입니다. 외부 기록기는 동일한 대상이나 처리가 있는 스푼 파일의 그룹을 표시하는 JES 시작 시에 JES에 정의된 이름입니다. JES가 자체적으로 처리하는 파일의 경우, 외부 기록기는 일반적으로 특정 하드웨어 디바이스(예: 프린터)와 연관됩니다. 이러한 기록기의 이름은 JES용으로 예약되어 있습니다.

애플리케이션 간의 파일 전송의 경우, CICS 태스크가 스푼 파일을 읽을 때 발생하므로 유일한 이름 지정 요구사항은 수신인(CICS 태스크)이 발신인이 사용한 이름을 알고, 수신인의 운영 체제에 있는 다른 애플리케이션이 동일 이름을 다른 용도로 사용하지 않는다는 점을 알아야 합니다. CICS 태스크가 의도되지 않은 스푼 파일을 읽지 않도록 하기 위해서, CICS에서는 사용자가 지정하는 외부 기록기 이름이 처음 네 문자에서 자체 z/OS Communications Server 애플리케이션 ID와 일치해야 합니다. 결과적으로, 파일을 CICS로 지시하는 작업 또는 시스템은 이를 CICS 애플리케이션 ID의 처음 네 문자로 시작하는 외부 기록기 이름에 전송해야 합니다.

JES는 1문자 CLASS 값으로 특정 외부 기록기의 파일을 분류합니다. SPOOLOPEN INPUT 명령에 클래스를 지정하는 경우에는 사용자가 이름 지정하는 외부 기록기의 해당 클래스에 첫 번째(가장 오래된) 파일을 얻게 됩니다. 클래스를 생략하면 해당 기록기의 클래스에 가장 오래된 파일을 얻습니다. 발신인은 클래스를 지정합니다. 발신인이 클래스를 지정하지 않으면 A가 사용됩니다.

출력에서 NODE 및 USERID 값 둘 모두를 사용하여 SPOOL 파일의 대상을 식별합니다. CICS가 실행 중인 MVS 시스템에서 z/OS Communications Server에 해당 시스템이 알려져 있으므로 NODE는 운영 체제의 이름(예: MVS, VM)입니다.

USERID의 의미는 운영 체제에 따라 변합니다. VM에서, 이는 특정 사용자입니다. MVS에서는, JES 외부 기록기이거나 또 다른 JES 대상, TSO 사용자 또는 해당 시스템을 실행 중인 또 다른 작업일 수 있습니다. 이러한 대상이 JES 내부 리더이고 이는 일반적으로 예약명 INTRDR이 있습니다. 작업을 MVS 시스템에 제출하고 싶은 경우에는 스푼 파일을 내부 리더에 쓸 수 있습니다. 이 파일은 카드 리더 또는 TSO를 통해 제출된 작업과 동일한 양식과 순서로 작업을 실행하는 데 필요한 모든 JCL 문을 포함해야 합니다.

다음 예제는 내부 리더를 위해 SPOOLOPEN을 사용하는 COBOL 프로그램을 표시합니다. 이 예제에서는 캐리지 제어를 위한 첫 번째 문자의 사용을 막기 위해 NOCC 옵션을 지정하고 JCL 레코드 포맷을 사용해야 합니다.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01  OUTPUT-FIELDS.
    03  OUTPUT-TOKEN    PIC X(8)  VALUE LOW-VALUES.
    03  OUTPUT-NODE     PIC X(8)  VALUE 'MVSESA31'.
    03  OUTPUT-USERID   PIC X(8)  VALUE 'INTRDR  '.
    03  OUTPUT-CLASS    PIC X      VALUE 'A'.
PROCEDURE DIVISION.
EXEC CICS SPOOLOPEN OUTPUT
      TOKEN(OUTPUT-TOKEN)
      USERID(OUTPUT-USERID)
      NODE(OUTPUT-NODE)
      CLASS(OUTPUT-CLASS)
      NOCC
      PRINT
      NOHANDLE
END-EXEC.

```

그림 141. 내부 리더를 위해 *SPOOL* 명령을 사용하는 *COBOL* 프로그램의 예제

OUTDESCR은 매개변수의 문자열 주소를 포함하는 필드의 주소로 설정될 포인터 변수를 MVS JCL의 OUTPUT 문에 지정합니다.

다음 예제는 OUTDESCR 피연산자를 사용하는 COBOL 프로그램을 표시합니다.

```

WORKING-STORAGE SECTION.
01 F.
02 W-POINTER USAGE POINTER.
02 W-POINTER1 REDEFINES W-POINTER PIC 9(9) COMP.
01 RESP1 PIC 9(8) COMP.
01 TOKENWRITE PIC X(8).
01 ....
01 W-OUTDIS.
02 F PIC 9(9) COMP VALUE 43.
02 F PIC X(14) VALUE 'DEST(A20JES2 )'.
02 F PIC X VALUE ' '.
02 F PIC X(16) VALUE 'WRITER(A03CUBI)'.
02 F PIC X VALUE ' '.
02 F PIC X'11' VALUE 'FORMS(BILL)'.
LINKAGE SECTION.
01 DFHCOMMAREA PIC X.
01 L-FILLER.
02 L-ADDRESS PIC 9(9) COMP.
02 L-OUTDIS PIC X(1020).
PROCEDURE DIVISION.
    EXEC CICS GETMAIN SET(W-POINTER) LENGTH(1024)
        END-EXEC.
    SET ADDRESS OF L-FILLER TO W-POINTER.
    MOVE W-POINTER1 TO L-ADDRESS.
    ADD 4 TO L-ADDRESS.
    MOVE W-OUTDIS TO L-OUTDIS.
    EXEC CICS SPOOLOPEN
        OUTPUT
        PRINT
        RECORDLENGTH(1000)
        NODE('*')
        USERID('*')
        OUTDESCR(W-POINTER)
        TOKEN(TOKENWRITE)
        RESP(RESP1)
        NOHANDLE
    END-EXEC.
    EXEC CICS SPOOLWRITE
        .
        .
        .

```

참고:

1. GETMAIN 명령을 코딩하는 것이 중요합니다.
2. L-FILLER는 호출 프로그램이 전달하는 매개변수가 아닙니다. L-FILLER의 BLL은 그러면 SET ADDRESS에 의해 대체됩니다. 그러면 GETMAIN 요청을 사용하여 확보된 영역의 주소는 L-ADDRESS인 L-FILLER에 의해 지시된 첫 번째 단어(따라서 자신을 가리킴)로 이동됩니다. 그러면 L-ADDRESS는 주소 바로 뒤의 영역(L-OUTDIS)을 가리키기 위해 더하기 4만큼 변경됩니다. 그런 다음 L-OUTDIS는 OUTDESCRIPTOR DATA로 채워집니다. 그러므로 W-POINTER는 OUTDESCR 데이터를 가리키는 포인터가 있는 영역을 가리킵니다.

SPOOL 명령의 예제

다음 예제는 COBOL, PL/I, C 및 ASSEMBLER를 사용한 SPOOL 명령을 보여줍니다.

COBOL

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01  RESP          PIC 9(8) BINARY.
01  RESP2         PIC 9(8) BINARY.
01  TOKEN         PIC X(8).
01  OUTLEN        PIC S9(8) BINARY VALUE +80.
77  OUTPRT        PIC X(80) VALUE
    'SPOOLOPEN FUNCTIONING'.
01  PARMSPTR      POINTER.
01  PARMS-AREA.
    03  PARMSLEN   PIC S9(8) BINARY VALUE 14.
    03  PARMSINF   PIC X(14) VALUE
        'WRITER(MYPROG)'.
    03  PARMADDR   POINTER.
PROCEDURE DIVISION.
    SET PARMSPTR TO ADDRESS OF PARMS-AREA
    SET PARMADDR TO PARMSPTR
    SET PARMSPTR TO ADDRESS OF PARMADDR
    EXEC CICS SPOOLOPEN OUTPUT
        NODE ('*')
        USERID ('*')
        RESP(RESP) RESP2(RESP2)
        OUTDESCR(PARMSPTR)
        TOKEN(TOKEN)
    END-EXEC
    EXEC CICS SPOOLWRITE
        FROM(OUTPRT)
        RESP(RESP) RESP2(RESP2)
        FLENGTH(OUTLEN)
        TOKEN(TOKEN)
    END-EXEC
    EXEC CICS SPOOLCLOSE
        TOKEN(TOKEN)
        RESP(RESP) RESP2(RESP2)
    END-EXEC.
```

PL/I

```

DCL
  RESP FIXED BIN(31),
  RESP2 FIXED BIN(31),
  TOKEN CHAR(8),
  OUTLEN FIXED BIN(31) INIT(80),
  OUTPRT CHAR(80) INIT('SPOOL OPEN FUNCTIONING'),
  PARMADDR POINTER,
  PARMSPTR POINTER;
DCL
  1 PARMS,
    2 PARMSLEN FIXED BIN(31) INIT(14),
    2 PARMSINF CHAR(14) INIT('WRITER(MYPROG)')
      ALIGNED;
PARMADDR=ADDR(PARMS);
PARMSPTR=ADDR(PARMADDR);
EXEC CICS SPOOL OPEN OUTPUT NODE('*') USERID('*')
      TOKEN(TOKEN) OUTDESCR(PARMSPTR) RESP(RESP)
      RESP2(RESP2);
EXEC CICS SPOOL WRITE FROM(OUTPRT) FLENGTH(OUTLEN)
      RESP(RESP) RESP2(RESP2) TOKEN(TOKEN);
EXEC CICS SPOOL CLOSE TOKEN(TOKEN) RESP(RESP)
      RESP2(RESP2);

```

C

```

#define PARMS struct _parms
PARMS
{
  int   parms_length;
  char  parms_info[200];
  PARMS * pArea;
};
PARMS ** parms_ptr;
PARMS parms_area;
char userid[8] = "*";
char node[8] = "*";
char token[8];
long rcode1, rcode2;
/* These lines will initialize the outdescr area and
set up the addressing */
parms_area.parms_info[0] = '\0';
parms_area.pArea = &parms_area;
parms_ptr = &parms_area.pArea;
/* And here is the command with ansi carriage controls
specified and no class*/
EXEC CICS SPOOL OPEN OUTPUT
      NODE ( node )
      USERID ( userid )
      OUTDESCR ( parms_ptr )
      TOKEN ( token )
      ASA
      RESP ( rcode1 )
      RESP2 ( rcode2 );

```

ASSEMBLER

```

OUTPRT  DC      CL80'SPOOLOPEN FUNCTIONING'
PARMSPTR EQU    6
RESP    DC      F'0'
RESP2   DC      F'0'
TOKEN   DS      2F
OUTPTR   DC      A(PARMSLEN)
PARMSLEN DC     F'14'
PARMSINF DC     C'WRITER(MYPROG)'
        LA      PARMSPTR,OUTPTR
EXEC CICS SPOOLOPEN OUTPUT OUTDESCR(PARMSPTR)
        NODE('*') USERID('*') RESP(RESP)
        RESP2(RESP2) TOKEN(TOKEN)
EXEC CICS SPOOLWRITE FROM(OUTPRT)
        TOKEN(TOKEN) RESP(RESP) RESP2(RESP2)
EXEC CICS SPOOLCLOSE TOKEN(TOKEN) RESP(RESP)
        RESP2(RESP2)

```


제 15 부 BMS(Basic Mapping Support)

BMS(Basic Mapping Support)를 사용하여 터미널 디바이스 제어.

제 45 장 BMS(Basic Mapping Support)

BMS(Basic Mapping Support)는 CICS 프로그램과 터미널 디바이스 사이의 API(Application Programming Interface)입니다.

BMS는 이러한 목적의 두 가지 명령 세트 중 하나입니다. 다른 하나인 터미널 제어는 애플리케이션 개발에서 터미널 지원에서 설명합니다.

많은 애플리케이션의 경우, BMS에는 몇 가지 장점이 있습니다. 먼저, BMS는 애플리케이션에서 디바이스 종속성을 제거합니다. 디바이스 독립 출력 명령을 해석하고, 특정 터미널에 대해 디바이스 종속 데이터 스트림을 생성합니다. 또한 수신되는 디바이스 종속 데이터를 디바이스 독립적 형식으로 변환합니다. 이 기능은 복잡한 디바이스 데이터 스트림을 학습해야 하는 필요성을 없애줍니다. BMS는 애플리케이션이 아니라 터미널 정의에서 디바이스 정보를 판별하므로, 다양한 디바이스에 대해 동일한 프로그램을 사용할 수 있도록 허용합니다.

두 번째로, BMS는 애플리케이션 논리로부터 형식 준비 및 디자인을 구분하여, 서로의 영향을 줄입니다. 이 두 기능 모두 새 프로그램을 쉽게 작성하고 기존 코드를 쉽게 유지보수할 수 있도록 합니다.

관련 태스크:

937 페이지의 『BMS 맵핑된 출력 전송』

BMS 맵핑된 출력을 전송하려면 먼저 다음 단계를 수행하여 출력 데이터를 맵 구조로 이동해야 합니다.

관련 정보:

916 페이지의 『맵 작성』

BMS는 맵 정의에 대해 세 가지의 어셈블러 언어 매크로 지시사항(매크로)을 제공합니다. 이 맵핑 정의 방법은 여전히 광범위하게 사용되며, 여기에 설명된 방법입니다.

943 페이지의 『SEND MAP 명령 사용』

이 목록은 SEND MAP 명령이 BMS에 알려주는 내용을 보여줍니다.

953 페이지의 『맵핑된 데이터 수신』

형식화된 화면은 출력만큼 입력에서도 중요합니다. 데이터 입력 애플리케이션은 명백한 한 가지 예지만, 대부분의 다른 애플리케이션도 부분적으로나마 형식화된 입력을 사용하기도 합니다.

BMS 지원 레벨

BMS 지원에는 최소, 표준 및 전체의 세 가지 레벨이 있습니다. 대부분의 설치에는 전체 BMS를 사용합니다. 사용자의 설치가 전체 BMS를 사용하는 경우에는 BMS의 모든 기능을 사용할 수 있습니다. 설치가 최소 또는 표준 BMS를 사용하는 경우에는 사용자의 레벨을 넘어서는 레벨을 필요로 하는 기능에 유의하십시오.

기능은 여기에 요약되어 있고 최소 BMS에 없는 기능을 다룰 때마다 다시 언급됩니다.

최소 BMS

최소 BMS는 이 예제와 단순 맵핑된 결과 및 맵핑된 입력에서 설명된 모든 것을 포함하여 3270 터미널의 모든 기본 기능을 지원합니다.

참고: 최소 BMS에는 본질적으로 표준 또는 전체 BMS보다 짧은 경로 길이를 가지고 있습니다. 이는 더 큰 버전에 포함되어 있고 이것이 제공하는 기능을 넘어서는 기능을 필요로 하지 않는 명령에서 "빠른 경로"로서 호출됩니다. 특히, ACCUM, PAGING, SET, OUTPARTN, ACTPARTN, LDC, MSR 또는 REQID 옵션 없이 SEND MAP 및 SEND CONTROL 명령에 사용되고, 프린시플 기능이 정의가 외부 포맷을 포함하지 않는 3270 표시장치 또는 프린터일 때 RECEIVE MAP 명령에 사용됩니다. CICS 추적 테이블을 보면 특정 BMS 요청이 빠른 경로를 사용했는지 여부를 알 수 있습니다. 빠른 경로가 사용되면 추적 테이블에는 BMS 입력 및 엑시트 코드의 중복 입력이 포함됩니다.

표준 BMS

표준 BMS는 다음을 추가합니다.

- 3270 이외의 터미널에 대한 지원
- 텍스트 출력 명령
- 특수 하드웨어 기능에 대한 지원: 파티션, 논리 디바이스 코드, 자기 슬롯 리더, 외부 포맷 등등
- SEND 명령에 대한 추가 옵션: NLEOM 및 FMHPARM

표준 BMS는 다음 터미널을 지원합니다.

- 순차 터미널(카드 리더, 라인 프린터, 테이프 또는 디스크로 구성됨)
- TWX 모델 33/35
- 1050
- 2740-1(버퍼 수신 없음), 2740-2, 2741
- 2770
- 2780
- 2980, 모델 1, 2 및 4

- 3270
- 3600(3601) LU
- 3650(3653 및 3270 호스트 대화 LU)
- 3650 해독기 LU
- 3767/3770 대화식 LU
- 3770 일괄처리 LU
- 3780
- LU 유형 4

전체 BMS

전체 BMS는 다음의 경우에 필요합니다.

- BMS 출력을 사용자 고유의 터미널에 직접 보내지 않고 전송할 때(SET 및 PAGING 옵션 및 BMS 라우팅)
- 여러 BMS SEND 명령을 사용하여 점증적으로 빌드된 메시지(ACCUM 및 PAGING 옵션)

몇몇 CICS 플랫폼은 BMS의 모든 기능을 지원하지는 않습니다. BMS는 z/OS용 CICS Transaction Server 및 VSE용 CICS Transaction Server에서 완전히 지원됩니다. 그러나 플랫폼 간에는 같은 레벨이라도 일반적으로 런타임 환경에서의 차이로 인한 차이점이 있습니다. 이들은 *CICS 제품군: API* 구조에서 기능별로 자세히 설명됩니다. 애플리케이션이 결국 다른 플랫폼으로 이동하거나 그래픽 사용자 인터페이스가 다른 플랫폼으로 분배될 가능성이 있는 경우에는 해당 매뉴얼을 참조해야 합니다.

BMS 출력 예제

이 예제는 BMS가 형식화된 화면을 작성하는 방법을 보여줍니다. 이는 프로그램의 데이터 항목 목록을 취하고 이를 사전 정의된 형식에 따라 화면에 표시합니다.

BMS는 프로그램이 제공하는 변수 데이터를 상수 데이터와 병합하여 형식(제목, 변수 필드의 레이블, 이러한 필드의 기본값)에 형식화된 화면을 작성합니다. 이는 현재 쓰고 있는 터미널에 대한 데이터 스트림을 빌드하여 이 병합된 데이터를 지정된 화면 위치에 적합한 속성(색상, 강조표시 등등)과 함께 표시합니다. 데이터 스트림에 대해서는 어떤 것도 알 필요가 없고, 필요한 CICS 명령을 쓰기 위해 포맷에 대해 많이 알 필요가 없습니다.

참고: 단순히 하기 위해 이 섹션은 표시장치 화면에 대한 것이지만 대부분은 프린터에도 동등하게 적용됩니다. 877 페이지의 제 44 장 『인쇄 및 스폴 파일』에서는 표시장치와 프린터의 차이에 대해 설명하지만 인쇄에 적용되는 추가 고려사항에 대해서도 다

룹니다. 게다가, 예제와 설명은 BMS가 3270의 기능을 지원하도록 설계되었기 때문에 표준 3270 터미널을 가정합니다. 다른 터미널은 931 페이지의 『비3270 터미널에 대한 지원』에서 설명됩니다.

맵이라는 형식을 이를 사용하는 프로그램과 별개로 정의합니다. 프로그램을 수정하지 않고 고도 필드를 옮기고, 속성을 변경하고, 상수 텍스트를 변경할 수 있습니다. 변수 데이터를 추가하거나 제거하는 경우에는 영향 받은 필드를 사용하는 프로그램을 변경해야 합니다.

이의 작동 방법에 대한 기본은 비정형의 단순 예제에 설명되어 있습니다. 실제 세상에서 요구사항은 항상 더 복잡하지만 이는 너무 복잡한 세부사항 없이도 기본사항을 제공합니다. CICS 샘플 애플리케이션 중에는 보다 실제적이고 완전한 BMS 예제가 있습니다. 이러한 프로그램은 CICS 분배 테이프에 소스 양식으로 포함되어 있습니다. 샘플 애플리케이션 안내서에서 자세한 정보를 볼 수 있습니다.

이 예제는 비용 판매가 완료되기 전에 고객의 잔고를 확인하는 백화점에서 사용되는 트랜잭션을 위한 코드를 써야 한다고 가정합니다. 이 트랜잭션은 고객의 계정이 열려 있는지 확인하고, 계정의 상태에 따라 현재 구매가 허용 가능한지를 확인하는 것이 하는 일의 전부이기 때문에 "빠른 확인"이라고 불립니다. 이 트랜잭션의 출력 파트의 프로그램은 계정 번호를 얻고, 응답으로 그림 142에 표시된 화면을 생성합니다.

```
QCK                      Quick Customer Account Check
Account:  00000005
Name:     Thompson      Chris
Max charge: $500.00
```

그림 142. 일반 "빠른 확인" 출력 화면

프로그램은 고객의 레코드를 계정 파일에서 검색하기 위해 입력된 계정 번호를 사용합니다. 이 레코드의 정보를 사용하여 맵에 계정 번호와 고객 이름을 입력하고, 신용 한도, 미결제 잔액 및 마지막 청구 기간 후에 게시된 구매로부터 허용된 최대 비용을 계산합니다. 금액이 음수로 나오면 0 값을 표시하고 설명 메시지를 추가하도록 되어 있습니다. 또한 비용 카드가 그림 143에 표시된 메시지와 함께 분실, 도난 또는 취소된 경우 점원에게 경고해야 합니다.

```
QCK                      Quick Customer Account Check
Account:  00000005
Name:     Thompson      Chris
Max charge:  $0.00
STOLEN CARD - SECURITY NOTIFIED
```

그림 143. 경고 메시지가 있는 "빠른 확인" 출력 화면

이 메시지는 점원의 주의를 끌기 위해 강조표시됩니다.

먼저 화면을 정의해야 합니다. 916 페이지의 『맵 작성』에서 이 특정 맵에 대해 이를 수행하는 방법을 설명합니다. 그러나 우선은, 이 프로세스의 출력 중 하나가 그림 144에 있는 데이터 구조와 같은 것으로 가정해 보겠습니다. (COBOL을 사용하여 예제를 코딩하므로 COBOL 코딩된 버전의 구조를 보여줍니다. 그러나 BMS는 구조를 CICS가 지원하는 어떤 언어로나 생성합니다.) 맵 작성 프로세스는 이 소스 코드를 프로그램으로 복사할 수 있는 라이브러리에 저장합니다.

```
01 QCKMAPO.
02 FILLER PIC X(12).
02 FILLER PICTURE X(2).
02 ACCTNOA PICTURE X.
02 ACCTNOO PIC X(7).
02 FILLER PICTURE X(2).
02 SURNAMEA PICTURE X.
02 SURNAMEO PIC X(15).
02 FILLER PICTURE X(2).
02 FNAMEA PICTURE X.
02 FNAMEO PIC X(10).
02 FILLER PICTURE X(2).
02 CHGA PICTURE X.
02 CHGO PIC $,$$0.00
02 FILLER PICTURE X(2).
02 MSGA PICTURE X.
02 MSGO PIC X(30).
```

그림 144. "빠른 확인"의 기호 맵

이 구조의 데이터 이름은 맵 정의에서 나옵니다. 프로그램이 어떤 방식으로든 변경해야 하는 필드에 이름을 지정합니다. 이 예제에서 이 범주는 계정 번호, 성, 이름, 최대 비용 및 설명 메시지를 표시하는 필드를 포함합니다. "빠른 고객 계정 확인" 및 "계정" 등과 같이 변경되지 않는 필드 레이블 또는 화면 제목을 포함하지 않습니다.

화면에 이름을 지정하는 각 필드는 데이터 구조에 몇몇 필드를 생성합니다. 이는 맵에 지정하는 이름에 추가되는 1문자 접미부로 구별됩니다. 두 개는 여기에 나타납니다. 필드 속성 바이트의 "A" 접미부 및 출력 데이터를 위한 "O" 접미부입니다. 색상 및 강조 표시와 같은 특수 디바이스를 사용하기 위해 맵을 작성 중이었거나 출력뿐만 아니라 입력을 위해 맵을 사용 중이었다면 더 많이 있을 수 있습니다. 이러한 다른 필드에 대해서는 940 페이지의 『표시장치 특성 설정』 및 953 페이지의 『맵핑된 데이터 수신』에서 설명합니다.

이 특정 연습을 위한 키 필드는 "O" 접미부가 있습니다. 이들은 화면에 표시하고 싶은 데이터를 놓는 곳입니다. 데이터가 표시되는 방법을 변경하고 싶은 경우에는 "A" 서브 필드를 사용합니다. 이 예제에서는 고객이 의심스러운 카드를 사용 중인 경우 MSGA를 사용하여 메시지를 강조표시합니다.

여기에는 예제에 필요한 코드의 아웃라인이 있습니다. 맵을 작성하여 생성되는 데이터 구조(그림 144)에서 복사해야 하고 세 번째 행의 COPY QCKSET 문이 이를 수행합

니다. (원래대로 계정 레코드 포맷에 복사 문을 사용할 수도 있습니다. 여기서는 내용을 볼 수 있도록 부분적으로 확장된 상태로 표시합니다.)

```

WORKING-STORAGE SECTION.
C COPY IN SYMBOLIC MAP STRUCTURE.
01 COPY QCKSET.
01 ACCTFILE-RECORD.
    02 ACCTFILE-ACCTNO          PIC S9(7).
    02 ACCTFILE-SURNAME        PIC X(15).
    02 ACCTFILE-FNAME          PIC X(10).
    02 ACCTFILE-CREDIT-LIM     PIC S9(7) COMP-3.
    02 ACCTFILE-UNPAID-BAL     PIC S9(7) COMP-3.
    02 ACCTFILE-CUR-CHGS       PIC S9(7) COMP-3.
    02 ACCTFILE-WARNCODE       PIC X.

PROCEDURE DIVISION.

EXEC CICS READ FILE (ACCT) INTO (ACCTFILE-RECORD) RIDFLD (CKNO)
    ... END-EXEC.
MOVE ACCTFILE-ACCTNO TO ACCTNO0.
MOVE ACCTFILE-SURNAME TO SURNAME0.
MOVE ACCTFILE-FNAME TO FNAME0.
COMPUTE CHGO = ACCTFILE-CREDIT-LIM - ACCTFILE-UNPAID-BAL
    - ACCTFILE-CUR-CHGS.
IF CHGO < ZERO, MOVE ZERO TO CHGO
    MOVE 'OVER CHARGE LIMIT' TO MSGO.
IF ACCTFILE-WARNCODE = 'S', MOVE DFHMBRY TO MSGA
    MOVE 'STOLEN CARD - SECURITY NOTIFIED' TO MSGO
    EXEC CICS LINK PROGRAM('NTFYCOPS') END-EXEC.
EXEC CICS SEND MAP ('QCKMAP') MAPSET ('QCKSET') END-EXEC.
EXEC CICS RETURN END-EXEC.

```

그림 145. BMS 출력 예제

맵 작성

BMS는 맵 정의에 대해 세 가지의 어셈블러 언어 매크로 지시사항(매크로)을 제공합니다. 이 맵핑 정의 방법은 여전히 광범위하게 사용되며, 여기에 설명된 방법입니다.

그러나 더 쉽게 맵 작성 프로세스를 만들기 위해 표시장치 터미널의 기능을 이용하는 맵을 작성하기 위한 다른 제품도 있습니다. 이 제품들은 일반적으로 더 적은 프로그래머의 노력으로, BMS 매크로와 같은 출력을 생성합니다.

이 제품 중 하나가 SDF II(Screen Definition Facility II)입니다. SDF II는 진행되는 동안 모양과 사용성을 테스트하여, 표시장치에서 직접 화면을 빌드할 수 있도록 합니다. *Screen Definition Facility II* 일반 소개 파트 1 및 *Screen Definition Facility II* 일반 소개 파트 2에서 SDF II 대해 자세히 볼 수 있습니다.

BMS 맵을 정의하는데 사용된 세 개의 어셈블러 매크로는 다음과 같습니다.

DFHMDF

화면 또는 페이지의 개별 필드를 정의합니다.

DFHMDI

단일 맵을 필드 콜렉션으로 정의합니다.

DFHMSD

단일 맵을 맵핑 세트로 그룹화합니다.

이 프로세스의 설명은 개별 필드를 정의하는 방법을 알려주는 것으로 시작합니다. 그런 다음 필드가 전체 맵으로 가고, 맵이 맵핑 세트(어셈블리 장치)로 가는 방법을 설명하는 것으로 진행됩니다. BMS는 거의 모든 유형을 지원하지만, 주로 3270 유형 터미널에 맞게 디자인됩니다. 3270 용어에 대한 정보는 757 페이지의 제 34 장 『터미널의 3270 제품군』의 내용을 참조하십시오.

관련 개념:

911 페이지의 제 45 장 『BMS(Basic Mapping Support)』

BMS(Basic Mapping Support)는 CICS 프로그램과 터미널 디바이스 사이의 API(Application Programming Interface)입니다.

맵 필드 정의: DFHMDF

매크로를 코딩하려고 시도하기 전에 화면의 레이아웃을 설계해야 합니다. 이를 수행한 후에는 DFHMDF 매크로를 사용하여 화면(페이지)에서 각 필드를 정의합니다.

이 태스크 정보

여기에서는 다음을 표시합니다.

- 화면에서 필드의 위치
- 필드의 길이
- 기본 내용(프로그램에서 이를 항상 제공하려고 하지 않는 한)
- 운영자가 필드에 입력할 수 있는지 여부와 입력하는 내용, 커서가 거기에 멈추는지 여부, 문자의 강도 및 수정된 데이터 태그의 초기 상태를 관리하는 필드 표시장치 속성
- 몇몇 터미널의 경우 색상, 밑줄, 강조 표시 등과 같은 확장된 표시장치 속성
- 해당 내용이나 속성을 수정한 경우 프로그램에서 필드를 가리킬 때 사용하는 이름

애플리케이션이 참조하는 필드에는 필드 이름이 할당되어야 합니다. 필드 이름과 필드 이름을 형성할 때 사용할 수 있는 문자는 다음 규칙을 준수해야 합니다. (이러한 규칙은 현재 지원되는 컴파일러와 어셈블러에 적용됨을 유의하십시오.)

사용된 문자는 어셈블러 보통 기호의 이름에 유효해야 합니다. 이 문자 세트는 알파벳 문자 A - Z(대문자 또는 소문자), \$, #, @, 숫자 0 - 9 및 밑줄(_) 문자로 구성됩니다.

이 규칙에는 하나의 예외가 있습니다. 하이픈(-) 문자는 다음과 같은 경우 필드 이름에 사용될 수 있습니다.

- 맵핑 세트는 COBOL에 작성된 애플리케이션에 의해서만 사용됩니다.
- 맵핑 세트는 상위 레벨 어셈블러를 사용하여 생성됩니다.

필드 이름의 첫 번째 문자는 알파벳순이어야 하지만 다른 문자는 앞서 설명된 문자 세트 구성될 수 있습니다.

또한 필드 이름에 사용되는 문자는 맵을 사용하여 애플리케이션의 프로그래밍 언어가 지원하는 문자 세트를 준수해야 합니다. 예를 들어, 애플리케이션 언어가 COBOL이면 @ 문자 또는 이전 버전에서는 밑줄을 사용할 수 없습니다. 이러한 문자 세트에 대한 자세한 정보는 적합한 언어 참조 매뉴얼을 참조하십시오.

DFHMDF 매크로를 사용하면 필드 이름의 길이가 1부터 30자일 수 있습니다. DFHMDF 는 몇몇 추가 문자 중 하나를 정의된 이름에 첨부하여 추가 변수 이름을 파생하여 기호 설명 맵을 생성합니다. 그러므로 이러한 파생된 이름은 길이가 최대 31자일 수 있습니다. 어셈블러, PL/I 및 C 언어는 모두 31자 이상의 변수 이름을 지원합니다. 그러나 COBOL 언어는 최대 30자까지 허용합니다. 즉, 맵에 사용되는 필드 이름은 COBOL 애플리케이션의 경우 29자를 초과해서는 안 됩니다. 예를 들어, 다음 필드 정의는 COBOL 을 제외하고 모든 언어에 유효합니다.

```
ThisIsAnExtremelyLongFieldName DFHMDF LENGTH=10,POS=(2,1)
```

그리고 다음 필드 정의는 COBOL에만 유효합니다.

```
Must-Not-Exceed-29-Characters DFHMDF LENGTH=10,POS=(2,1) "
```

필드 정의의 모든 옵션이 여기에 설명되지는 않습니다. 나머지는 *CICS 애플리케이션 프로그래밍 참조서*에서 설명됩니다.

그림 146에서는 shows the 914 페이지의 그림 143에서 고려한 맵의 필드 정의를 보여줍니다.

```
DFHMDF POS=(1,1),LENGTH=3,ATTRB=(ASKIP,BRT),INITIAL='QCK'
DFHMDF POS=(1,26),LENGTH=28,ATTRB=(ASKIP,NORM),                X
INITIAL='Quick Customer Account Check'
DFHMDF POS=(3,1),LENGTH=8,ATTRB=(ASKIP,NORM),INITIAL='Account:'
ACCTNO DFHMDF POS=(3,13),LENGTH=7,ATTRB=(ASKIP,NORM)
DFHMDF POS=(4,1),LENGTH=5,ATTRB=(ASKIP,NORM),INITIAL='Name:'
SURNAME DFHMDF POS=(4,13),LENGTH=15,ATTRB=(ASKIP,NORM)
FNAME DFHMDF POS=(4,30),LENGTH=10,ATTRB=(ASKIP,NORM)
DFHMDF POS=(5,1),LENGTH=11,ATTRB=(ASKIP,NORM),INITIAL='Max charge:'
CHG DFHMDF POS=(5,13),ATTRB=(ASKIP,NORM),PICOUT='$,$$0.00'
MSG DFHMDF LENGTH=20,POS=(7,1),ATTRB=(ASKIP,NORM)
```

그림 146. BMS 맵 정의

1. **POS**(위치) 매개변수는 맵의 왼쪽 상단 구석, 위치 (1,1)에 상대적으로 필드의 행과 열 위치를 나타냅니다. 이는 반드시 존재해야 합니다. 모든 필드가 필드 속성 바이트로 시작함을 기억하십시오. POS는 이 바이트의 위치를 정의합니다. 필드의 내용은 오른쪽으로 즉시 따라붙습니다.
2. **LENGTH** 옵션은 필드의 문자가 얼마나 긴지를 알려줍니다. 길이는 속성 바이트를 포함하지 않으므로 각 필드는 해당 LENGTH 값보다 하나 이상의 열을 차지합니다. 예를 들어, 이 맵의 첫 번째 필드의 경우에는 속성 바이트는 행 1, 열 1에 있고 표시장치 데이터는 열 2-4에 있습니다. 필드는 최대 256문자일 수 있고 한 행에서 다른 행으로 줄 바꾸기할 수 있습니다. (맵이 화면보다 작은 경우 줄 바꾸기하는 필드에 유의하십시오. 자세한 정보는 946 페이지의 『출력 화면 빌드』에서 맵 외부 참조하십시오.)
3. **ATTRB**(속성) 옵션은 761 페이지의 『3270 필드 속성』에서 설명한 필드의 필드 속성을 설정합니다. 이는 필수는 아닙니다. BMS는 (ASKIP, NORM)—자동 건너뛰기 보호, 정상 강도, 수정된 데이터 태그 오프(이를 생략한 경우)의 기본값을 사용합니다. 각 확장 속성에는 다른 옵션이 있고, 이들 중 어느 것도 이 맵에 사용되지 않았습니다. 이들은 940 페이지의 『표시장치 특성 설정』에서 설명됩니다.
4. 필드의 **INITIAL** 값은 필요하지 않습니다. 이를 'QCK' 등과 같은 상수값이 있는 레이블과 제목 필드에 사용하고 프로그램이 값을 항상 제공할 필요가 없도록 필드에 기본값을 지정하려고 할 때 사용하십시오.
5. 필드 CHG의 정의에서 **PICOUT** 옵션은 BMS에게 필드에 어떤 PICTURE 절을 생성할지를 알려줍니다. 이를 사용하면 데이터를 맵에 직접 이동할 때 COBOL 또는 PL/I의 기능을 직접 편집할 수 있습니다. PICOUT을 생략하고 숫자(NUM) 속성 또한 생략하는 경우에는 BMS에서는 문자 데이터를 가정합니다. 915 페이지의 그림 144에서는 CHG의 PICOUT 옵션과 다른 필드에서 이의 부재의 영향을 보여줍니다. BMS는 사진으로부터 길이를 추론하므로 PICOUT을 사용하는 경우에는 LENGTH 옵션을 생략할 수 있습니다.
6. **GRPNAME** 및 **OCCURS** 옵션은 보다 복잡한 문제를 위한 것이므로 단순 예제에는 나타나지 않습니다. GRPNAME을 사용하면 프로그램 내에서 처리하기 위해 맵 필드를 소분류할 수 있고 OCCURS를 사용하면 프로그램에서 배열로서 취급할 수 있도록 맵 필드처럼 인접을 정의할 수 있습니다. 이러한 옵션은 927 페이지의 『복합 필드 사용』에서 맵에 대한 몇몇 추가 정보 뒤에 설명됩니다.

맵 정의: DFHMDI

맵에서 모든 필드가 정의된 후에 BMS에 DFHMDI 매크로를 앞에 놓는 방법으로 단일 맵을 형성한다고 알립니다.

이 태스크 정보

이 매크로는 BMS에 다음을 알려줍니다.

- 맵의 이름
- 행과 열에서 크기
- 화면에 나타나는 위치(한 화면에 여러 맵을 넣을 수 있음)
- 3270 확장 표시장치 속성을 사용하는지 여부, 사용하는 경우 어느 것을 사용하는지
- DFHMDI 매크로에서 특정 값을 지정하지 않은 필드를 위한 이러한 확장 속성의 기본값
- 맵 전송과 연관된 디바이스 제어(알람을 울리거나, 키보드를 잠금 해제하는지 여부 등)
- 맵이 지원하는 디바이스 유형, 여러 유형의 디바이스에 대해 여러 버전의 맵을 작성하려는 경우(933 페이지의 『디바이스 종속 맵』 참조)

맵 이름과 크기는 DFHMDI 매크로에 대한 중요한 정보이지만, 문서 용도의 경우 기본 값으로 설정하는 대신에 다른 옵션을 명시적으로 지정해야 합니다. 이 예제의 DFHMDI 매크로는 다음과 같을 수 있습니다.

```
QCKMAP DFHMDI SIZE=(24,80),LINE=1,COLUMN=1,CTRL=ALARM
```

맵 QCKMAP를 이름 지정했습니다. 이는 SEND MAP 명령에서 사용하는 ID입니다. 이는 24행 길이, 80열 너비이고, 표시장치의 첫 번째 행의 첫 번째 열에서 시작됩니다. 또한 맵이 표시될 때 알람을 울리기를 원함을 표시했습니다.

맵핑 세트 정의: DFHMSD

맵핑 세트를 정의하는 DFHMSD 매크로를 사용하여 맵을 작성합니다.

이 태스크 정보

맵은 맵핑 세트라는 그룹으로 어셈블링됩니다. 일반적으로 단일 트랜잭션이나 여러 관련 트랜잭션이 사용하는 모든 맵을 그룹화합니다. (926 페이지의 『맵을 맵핑 세트로 그룹화』에서 맵 그룹화의 이유에 대해 자세히 설명합니다.) 맵핑 세트는 둘 이상의 맵을 포함할 필요가 없습니다. 말하자면 이 단순 예제에서는 맵핑 세트는 "빠른 확인" 맵으로만 구성됩니다.

한 DFHMSD 매크로는 맵핑 세트에서 모든 맵 정의의 앞에 놓입니다. 이는 다음을 제공합니다.

- 맵핑 세트의 이름
- 출력, 입력 또는 둘 모두를 위해 맵을 사용 중인지 여부
- 개별 맵에 대해 DFHMDI 매크로에 지정하지 않은 맵 특성의 기본값
- 필드나 맵 정의에 지정하지 않은 확장 속성의 기본값

- 물리적 또는 기호 맵을 현재 어셈블리에 작성 중인지 여부(924 페이지의 『물리적 및 기호 맵핑 세트』 참조)
- 맵을 사용하는 프로그램의 프로그래밍 언어
- 맵을 빌드하는 데 사용되는 저장영역에 대한 정보

다음은 이 예제의 시작부에 필요로 하는 DFHMSD 매크로입니다.

```
QCKSET DFHMSD TYPE=MAP,STORAGE=AUTO,MODE=OUT,LANG=COBOL,TIOAPFX=YES
```

이 맵핑 세트 정의는 BMS에 여기에 있는 맵이 출력에만 사용되고, 이를 사용하는 프로그램은 COBOL로 작성되었음을 알려줍니다. 이는 이름 QCKSET를 맵핑 세트에 지정합니다. TIOAPFX=YES는 각 기호 맵의 시작부에 12바이트 "접두부" 필드를 포함 시킵니다. (915 페이지의 그림 144에서 두 번째 행에서 효과를 볼 수 있습니다.) 명령 언어 프로그램에서 항상 이 필드가 필요하고, 종종 기본값이 생략된 것처럼 이를 명시 적으로 지정해야 합니다. MAP 및 STORAGE는 937 페이지의 『BMS 맵핑된 출력 전송』에 설명되어 있습니다.

어셈블러에게 맵핑 세트에서 마지막 맵의 끝에 도달했음을 알려려면 맵 정의의 끝에 DFHMSD 매크로가 필요합니다.

```
DFHMSD TYPE=FINAL
```

BMS 매크로 쓰기

BMS 매크로는 어셈블러 언어 문이기 때문에 어셈블러 언어 구문 규칙을 따라야 합니다.

다음 규칙 세트는 작동하지만 실제 규칙보다 더 제한적입니다. 어셈블러 언어 구문 규칙의 전체 세트의 경우 어셈블러 H 버전 2 애플리케이션 프로그래밍 언어 참조를 참조 하십시오.

1. 이름을 열 1에서 시작하십시오. 맵과 맵핑 세트 이름은 최대 7자 길이일 수 있습니다. 필드 이름의 최대 길이(DFHMDI 매크로)는 프로그래밍 언어에 달려 있습니다. BMS는 필드 이름에 한 문자의 접두부를 추가하여 레이블을 작성합니다. 이러한 레이블은 프로그램으로 복사되기 때문에 대상 언어가 허용하는 것보다 길어 서는 안됩니다. 맵 필드 이름의 한계는 COBOL의 경우 29자이고, PI/I 및 어셈블 러 H의 경우 30자이고, 어셈블러 F의 경우 7자입니다. C 및 C++의 경우 맵이 내부 데이터 오브젝트로서 프로그램으로 복사되는 경우 30이고 외부 데이터 오브젝트인 경우에는 6입니다(맵 복사에 대한 자세한 정보는 937 페이지의 『맵의 저장영역 획득 및 정의』의 내용을 참조하십시오.)
2. 열 10부터 매크로 ID를 시작하거나 이름이 8개 위치를 초과하는 경우에는 이 ID 와 이름 사이에 하나의 공백을 남기십시오. 필드 정의의 경우 ID는 항상 DFHMDI 입니다. 맵 정의의 경우 DFHMDI이고, 맵핑 세트를 시작하고 끝내는 맵핑 세트 매크로의 경우에는 DFHMSD입니다.

3. 필드 설명의 나머지는 값이 뒤따르는 키워드로 구성됩니다(위치 매개변수의 POS 처럼). 때로 키워드에는 값이 없지만, 값이 있는 경우에는 등호(=)가 항상 키워드를 값과 구분합니다.
4. 매크로 ID 뒤에 하나의 공백을 남긴 다음에 키워드를 시작하십시오. 이들은 임의의 순서로 나타날 수 있습니다.
5. 하나의 쉼표(공백 아님)으로 키워드를 분리하지만 마지막 키워드 뒤에는 쉼표를 넣지 마십시오.
6. 키워드는 열 71을 통해 확장할 수 있습니다. 더 많은 공간이 필요한 경우에는 행에 완전히 잘 맞는 마지막 키워드를 뒤따르는 쉼표 뒤에서 멈추고 다음 행의 16열에서 재개하십시오.
7. 초기 값(INITIAL, XINIT 및 GINIT 키워드)은 새 행에서 시작하더라도 맞지 않을 수 있기 때문에 규칙에 대한 예외입니다. 2바이트 문자가 관련된 경우에는 예외입니다. 초기 값 자체의 첫 번째 문자 뒤의 어느 위치에서나 이를 분리할 수 있습니다. 이 방법으로 분리할 때에는 71까지 모든 열을 사용하고 다음 행의 16열에서 계속하십시오. 2바이트 문자 세트(DBCS) 데이터는 일반 1바이트(SBCS) 데이터보다 표현하기가 더 복잡합니다. DBCS 초기값이 있는 경우에는 12 단계를 참조하십시오.
8. 초기값을 작은따옴표로 둘러싸십시오. 텍스트 내에 작은따옴표가 필요한 경우에는 2개의 연속하는 작은따옴표(어셈블러가 추가 하나를 제거함)를 사용하십시오. 어셈블러에서는 앰퍼샌드 또한 특별한 중요성이 있고 동일한 기술을 사용합니다. 하나를 원하는 곳에 두 개의 앰퍼샌드를 사용하면 어셈블러가 추가 하나를 제거합니다.
9. 매크로에 둘 이상의 행을 사용하면 *마지막을 제외하고* 모든 행의 72열에 문자(공백을 제외한 아무것이나)를 하나 두십시오.
10. 맵에 설명을 원하면 단일 매크로를 구성하는 행 사이가 아니라 매크로 사이에 주석 행을 사용하십시오. 주석 행은 열 1에 별표가 있고 열 72에 공백이 있습니다. 설명은 열 2-71 사이 아무 곳이나 나타날 수 있습니다.
11. INITIAL 매개변수의 값과 설명을 제외하고는 대문자를 사용하십시오.
12. DBCS를 포함하는 초기값의 경우. 전적으로 DBCS인 초기 데이터가 있는 경우에는 데이터에 GINIT 키워드를 사용하고 키워드 PS=8 또한 지정하십시오. 데이터에 DBCS 및 SBCS 문자 둘 모두가 포함되는 경우, 즉, 이들이 혼합된 경우에는 INITIAL을 사용하고 SOSI=YES를 지정하십시오. (유지보수하고 있는 코드에서 찾을 수도 있으므로 세 번째 대안, XINIT를 설명해야 합니다. 그러나 XINIT는 사용하기가 훨씬 어렵고 데이터가 완전히 유효성 검증되지 않았으므로 가능한 경우 GINIT 및 INITIAL을 사용해야 합니다. XINIT는 순수하거나 혼합된 DBCS에 사용될 수 있습니다. PS=8이 있는 XINIT는 GINIT의 규칙을 따르고, SOSI=YES

가 있는 XINIT는 INITIAL의 규칙을 따릅니다(적어도 대부분은). 주요한 차이점은 XINIT에서는 데이터를 16진으로 표현하지만 GINIT 및 INITIAL에서는 일반 문자를 사용한다는 점입니다.)

이는 DBCS 초기값을 쓰는 방법입니다.

- 일반 INITIAL 매개변수에서처럼 데이터를 작은따옴표로 묶습니다.
- 상수에서 각 DBCS 문자에 두 개의 일반 문자를 사용하고(XINIT가 있는 16진수 숫자의 두 쌍) 각 SBCS 문자마다 하나씩(XINIT가 있는 한 쌍)을 사용합니다.
- 각 DBCS 문자열을 바로 앞에 있는 SO(Shift-Out) 문자로 구분하고 바로 뒤에는 SI(Shift-In) 문자로 구분합니다. SO는 16진 X'0E'이고 대부분의 키보드에 '<'로 나타나고 SI는 X'0F'(>)입니다. (PS=8이 있는 XINIT는 예외입니다. SO/SI 대괄호가 함축되고 이를 키로 입력하지 않습니다) 예를 들어, 이들 모두는 전적으로 DBCS인 동일한 초기값을 정의합니다. (우선 LENGTH 값을 무시하십시오. 이에 대해서 우선 설명합니다.)

```
GINIT='<D1D2D3D4D5>',PS=8,LENGTH=10
INITIAL='<D1D2D3D4D5>',SOSI=YES,LENGTH=12
XINIT='C4F1C4F2C4F3C4F4C4F5',PS=8,LENGTH=10
XINIT='0EC4F1C4F2C4F3C4F4C4F50F',SOSI=YES,LENGTH=12
```

- SBCS 및 DBCS 순서는 INITIAL(및 SOSI=YES가 있는 XINIT)와의 어떤 조합에서도 서로를 따를 수 있습니다. 이전 예의 DBCS 문자열 앞에 'ABC'를 추가하고 'def'가 문자열을 뒤따르면 다음과 같습니다.

```
INITIAL='ABC<D1D2D3D4D5>def',SOSI=YES,LENGTH=18
XINIT='C1C2C30EC4F1C4F2C4F3C4F4C4F50F848586',SOSI=YES,LENGTH=18
```

- 초기값의 길이를 계산하려면 이들을 일반 문자나 16진 쌍으로 표현하는지와 관계없이 각 DBCS 문자는 2를 세고, 각 SBCS 문자를 1을 셈하십시오. GINIT(및 PS=8이 있는 XINIT)를 사용할 때에는 SO 및 SI 문자를 세지는 않지만 INITIAL(및 SOSI=YES가 있는 XINIT), 각 SO와 각 SI에 하나씩을 추가합니다. (이전 예제에서 동일한 상수에 서로 다른 LENGTH 값을 참고하십시오.) 모든 경우에는 LENGTH 값은 256을 초과하지 않아야 합니다.
- GINIT 및 INITIAL의 경우 상수가 한 행에 맞지 않으면 앞서 설명된 것과는 약간 다른 "확장된" 연속 규칙을 사용하십시오. 확장된 연속의 경우 초기값 내에 전체 문자(SBCS 문자, DBCS 쌍 또는 DBCS 문자열을 끝내는 SI) 뒤에 중지할 수 있습니다. DBCS 문자열의 가운데에 있다면 SI(한 행에서 SO와 SI는 균형을 이뤄야 함)를 추가하십시오. 그런 다음 연속 문자를 사용하여 열 72까지 행을 채우십시오. 행에 있는 마지막 의미있는 문자와 다른 한 모든 문자가 가능합니다.

DBCS 문자열 내에서 중지한 경우에는 다음 행의 16 열에 SO 문자를 놓고 17에서 재개하십시오. 그렇지 않으면 16에서 재개하십시오.

```
GXMPL1 DFHMDF POS=(02,21),LENGTH=20,PS=8,GINIT='<D1D2D3D4D5D6>*****
        <D7D8D9D0>'
IXMPL1 DFHMDF POS=(02,21),LENGTH=23,PS=8,INITIAL='abc<D1D2D3D4>ABC**
        DEFGHIJ'
```

XINIT로 확장 연속을 사용할 수 없습니다. 922 페이지의 7 단계에 설명된 규칙을 사용하십시오.

- LENGTH 스펙이 사용자가 제공하는 초기값의 길이를 초과하면 값은 GINIT(또는 PS=8를 가진 XINIT)를 사용한 경우 LENGTH 값에 오른쪽에 DBCS 공백이 채워집니다. INITIAL을 사용한 경우에는 채우기 문자는 상수의 마지막 파트가 SBCS인 경우 SBCS이고, 마지막 파트가 DBCS인 경우 DBCS 공백입니다. SOSI=YES가 있는 XINIT를 사용하는 경우에는 채우기 문자는 항상 SBCS 공백입니다.

맵 어셈블링

코딩을 시작하기 전에 맵핑 세트를 어셈블하고 링크 편집해야 합니다.

이 태스크 정보

일반적으로 두 개의 서로 다른 형식으로 맵핑 세트를 작성하려면 두 번 어셈블링해야 합니다. DFHMSD 매크로의 TYPE 옵션은 어셈블러에 특정 어셈블리에서 생성할 양식을 알려줍니다.

물리적 및 기호 맵핑 세트

링크 편집이 뒤따르는 TYPE=MAP 어셈블리를 사용하여 물리적 맵핑 세트라는 로드 모듈을 생성할 수 있습니다. TYPE=DSECT 어셈블리는 집합적으로 기호 맵핑 세트라고 불리는 일련의 데이터 구조를 생성하는 데 사용됩니다.

물리적 맵핑 세트에는 인코딩된 양식으로 된 형식 정보가 포함됩니다. CICS는 실행 시에 상수 필드에 이를 사용하고 프로그램에서 변수 데이터를 병합하는 방법을 판별하는 데 사용합니다.

일반적으로 물리적 맵핑 세트는 애플리케이션과 동일한 라이브러리에 저장되고 프로그램이 PROGRAM 자원 정의를 필요로 하는 것과 마찬가지로 CICS 내에서 MAPSET 자원 정의를 필요로 합니다.

TYPE=DSECT 어셈블리의 결과물은 LANG 옵션에 지정된 소스 언어로 코딩된 일련의 데이터 구조입니다. 기호 입력 맵이라는 입력에 사용되는 각 맵의 구조가 있고, 기호 출력 맵이라는 출력에 사용되는 각 맵의 구조가 있습니다.

기호 맵핑 세트는 컴파일(어셈블리) 시에 사용됩니다. 이를 프로그램으로 복사하면 여기에서 이름으로 맵의 필드를 가리키고 물리적 맵핑 세트에 지정된 양식으로 변수 데이터를 전달할 수 있습니다. 이미 COBOL(915 페이지의 그림 144 참조)에서 기호 출력 맵의 예제를 보여주었고 이를 예제 코드에서 사용했습니다. 기호 맵핑 세트는 일반적으

로 설치 시에 프로그램으로 복사되는 소스 코드에 대해 정의하는 라이브러리에 저장됩니다. 멤버 이름은 일반적으로 맵핑 세트 이름과 동일하지만 꼭 동일할 필요는 없습니다.

프로그램을 컴파일하거나 어셈블리하기 전에 TYPE=DSECT 어셈블리가 필요합니다. 물리적 맵핑 세트는 실행 시간 전까지는 사용되지 않으므로 테스트할 준비가 될 때까지는 TYPE=MAP 어셈블리 및 링크 편집을 늦출 수 있습니다. 그러나 결국에는 둘 모두를 수행해야 하므로 많은 설치는 이를 자동으로 실행하기 위한 카탈로그화된 프로시저를 제공합니다. 이러한 프로시저는 맵핑 세트의 소스 파일을 복사하고 TYPE=MAP을 사용하여 한 번 처리한 다음 TYPE=DSECT를 사용하여 다시 한 번 처리합니다. 어셈블러 프로시저에서 SYSPARM 옵션을 사용하여 특정 어셈블리에서 TYPE 값을 대체할 수도 있습니다. 맵 어셈블리가 있는 SYSPARM의 설명은 어셈블러 H 버전 2 애플리케이션 프로그래밍 언어 참조 매뉴얼을 참조하고, 맵 어셈블리에 대한 자세한 정보는 183 페이지의 제 10 장 『맵핑 세트 및 파티션 세트 설치』의 내용을 참조하십시오.

참고:

1. 기호 맵핑 세트가 특정 언어로 코딩된다고 해서 다른 언어로 코딩된 프로그램에서 동일 맵을 사용할 수 없다는 것은 아닙니다. 필요한 각 LANG 값에 대해 TYPE=DSECT를 사용하여 어셈블링하고 결과물을 다른 라이브러리 또는 다른 이름으로 저장하는 것에 유의하십시오. LANG 값은 한 번만 수행해야 하는 TYPE=MAP 어셈블리에 영향을 미치지 않습니다.
2. 기호 맵에 영향을 미치는 방법으로 기존 맵을 수정하는 경우에는 이를 사용하는 모든 프로그램을 재컴파일(리어셈블)해야 합니다. 그래야만 컴파일러가 새 물리적 구조에 해당하는 기호 구조를 사용할 수 있습니다. 이름이 없는 맵 필드의 변경사항은 기호 맵에 영향을 미치지 않지만, 이름이 있는 필드의 추가, 삭제, 재배열 및 길이 변경사항은 영향을 미칩니다. 재배열이란 DFHMDF 매크로를 가리킵니다. 화면에서 필드의 순서는 기호 맵에 영향을 미치지 않습니다. 그렇지만 DFHMDF 매크로를 화면의 필드와 같은 순서로 하는 것이 보다 효율적입니다. 따라서 맵 정의에서 DSATTS 옵션을 변경하십시오. 이 옵션은 프로그램을 사용하여 변경할 수 있는 확장 속성을 가리킵니다. 재컴파일하는 것이 항상 가장 안전합니다.

SDF II 대체

IBM 라이선스가 있는 프로그램 화면 정의 기능 II를 사용하는 경우에는 어셈블리 및 링크 편집 단계가 필요하지 않습니다. SDF II는 대화식 맵 작성 프로세스의 마지막 단계에서 기호 맵핑 세트와 물리적 맵핑 세트 둘 모두를 작성합니다.

SDF II는 MVS(프로그램 5665-366) 또는 VM(5664-307) 하에서 실행될 수 있습니다. CICS/BMS 프로그램을 위한 화면 정의 기능 II Primer, 화면 정의 기능 II 일반 소개 파트 1 및 화면 정의 기능 II 일반 소개 파트 2를 참조하십시오. 화면 정의 기능 II 일반 정보 및 CICS/BMS 프로그램의 화면 정의 기능 II Primer에서 더 많은 정보를 찾을 수 있습니다.

맵을 맵핑 세트로 그룹화

맵핑 세트의 모든 물리적 맵핑은 함께 어셈블링되므로 이들은 단일 로드 모듈을 구성합니다. BMS는 태스크에서 맵핑 세트의 첫 번째 사용 시에 발행되는 단일 로드 요청으로 이들 모두에 대한 액세스를 얻습니다.

다른 세트의 맵을 요청하지 않는 한 추가 로드가 필요하지 않습니다. 이 경우에는 BMS는 이전 맵핑 세트를 릴리스하고 새로운 맵핑 세트를 로드합니다. 첫 번째 맵핑 세트로 돌아가면 결과적으로 다시 로드됩니다. 로딩과 삭제는 반드시 I/O와 관련되지는 않지만 맵을 맵핑 세트로 그룹화할 때에는 경로 길이를 고려해야 합니다. 일반적으로 맵이 함께 사용되는 경우에는 이들은 동일 맵핑 세트에 있어야 합니다. 함께 사용되지 않은 맵들은 서로 다른 맵핑 세트에 있어야 합니다.

한 세트에서 맵 수의 한계는 9 998이지만 주어진 로드 모듈의 크기를 이상적인 상태로 유지해야 합니다. 따라서 자주 사용되지 않는 맵을 지정된 프로세스에서 일반적으로 사용되는 맵과 별도로 보관할 수도 있습니다.

마찬가지로, 맵핑 세트의 모든 기호 맵은 단일 기호 구조에 있습니다. 이는 938 페이지의 『BASE 및 STORAGE 옵션』에 설명된 대로 맵을 사용하는 동안 필요로 하는 저장영역의 양에 영향을 미칩니다. 프로그래밍 언어에 따라서 상위 레벨 이름에 영향을 미칠 수도 있고 맵 분리 또는 결합의 원인일 수도 있습니다.

애플리케이션 데이터 구조(ADS)

BMS 매크로에 의해 생성된 기호 맵은 또한 애플리케이션 데이터 구조(ADS)로도 알려져 있습니다.

z/OS용 CICS Transaction Server, 버전 1.2 이상에서 생산된 물리적 맵핑에는 출력 로드 모듈의 ADS 설명자도 포함됩니다. 이는 사용자의 프로그램이 컴파일 시간에 관련 DSECT 또는 복사본을 포함하도록 요청하지 않고, BMS 애플리케이션 업무 데이터 구조(SEND과 RECEIVE MAP 요청에서 데이터를 위한 애플리케이션에서 사용된 구조)에 대한 해석을 허용하기 위해 제공됩니다.

ADS 설명자는 맵에 대한 일반 정보를 가진 헤더 및 ADS(맵핑 정의 매크로에서 모든 지정 필드에 해당)에 나타나는 모든 필드를 위한 필드 디스크립터를 포함합니다. DFHMAPDS에서 오프셋 필드로부터 맵핑 세트에 위치할 수 있습니다.

ADS 설명자는 모든 맵을 위해 생성됩니다. DSECT=ADSIADSL 옵션을 지정하여 ADS의 장기간 또는 단형을 맵핑하기로 선택할 수 있습니다. 기본값은 ADS, 짧은(일반) 양식입니다. ADS의 정식 양식은 4-바이트 경계에 모든 필드를 정렬시키고, WebSphere MQ과 같은, 타 제품과의 일부 인터페이스에 필요합니다.

z/OS용 CICS Transaction Server, 버전 1.2 이전 CICS 릴리스로 생성된 맵핑 세트에는 ADS 설명자가 포함되지 않습니다.

ADS 설명자의 형식은 다음 카피북에 포함됩니다.

표 48. ADS 설명자 카피북

언어	카피북
어셈블러	DFHBRARD
C	DFHBRARH
PL/I	DFHBRARL
COBOL	DFHBRARO

ADS 설명자에 대한 추가 정보에 대해서 *CICS 외부인터페이스 안내서*의 내용을 참조하십시오.

맵을 리어셈블해야 하지만 소스에 대한 액세스가 없는 경우에는 유틸리티 프로그램 DFHBMSUP이 z/OS용 CICS Transaction Server, 버전 1.2(이상)에서 제공되어 맵핑 세트 로드 모듈에서 BMS 매크로 소스를 다시 작성합니다.

애플리케이션 코딩에는 영향을 주지 않는 이러한 기능을 설명하는 DFHBMSUP에 대한 자세한 정보는 *CICS 운영 및 유틸리티 안내서*의 BMS 매크로 생성 유틸리티 프로그램(DFHBMSUP)을 참조하십시오.

복합 필드 사용

BMS는 복합 필드 GRPNAME 및 OCCURS를 정의하기 위해 DFHMDF 매크로에 두 개의 옵션을 제공합니다.

이 태스크 정보

지금까지 보여준 기호 맵은 각 이름 지정된 맵 필드의 고정된 필드 세트에 구성되어 있습니다(915 페이지의 그림 144에서 A 및 O 하위 필드, 등등). 이러한 필드는 대부분 일반적인 것이지만 BMS는 두 개의 공통 프로그래밍 상황에 맞는 약간 다른 구조를 생성하는 필드 정의를 위한 두 개의 옵션을 제공합니다.

컴포지트 필드: GRPNAME 옵션

GRPNAME 옵션은 기호 저장영역 정의를 생성하고 특정 필드를 하나의 그룹 이름으로 결합하는 데 사용됩니다. 그룹에 속하는 각 필드에는 동일한 그룹 이름이 지정되어야 합니다.

이 태스크 정보

때로 표시장치에서 단일 필드 내의 서브필드를 참조해야 합니다. 예를 들어, 다음과 같이 이 화면에 나타나는 날짜 필드가 있을 수 있습니다.

03-17-92

화면에는 하나의 필드이지만(숫자 "0" 바로 앞에 하나의 속성 바이트) 프로그램에서 월, 일 및 연도 구성요소를 별도로 조작할 수 있어야 합니다.

DFHMDF 매크로의 GRPNAME 옵션을 사용하는 "그룹 필드"를 사용하여 이를 수행할 수 있습니다. 이를 작성하려면 각 구성요소 서브필드의 DFHMDF 매크로를 코딩하십시오. 각 정의는 GRPNAME 옵션에서 동일한 그룹 필드의 이름을 지정합니다. 예를 들어, 행 10의 시작 부분부터 시작하여 이전 날짜를 그룹 필드로 정의하려면 다음과 같이 쓸 수 있습니다.

```
MO      DFHMDF POS=(10,1),LENGTH=2,ATTRB=BRT,GRPNAME=DATE
SEP1    DFHMDF POS=(10,3),LENGTH=1,GRPNAME=DATE,INITIAL='- '
DAY     DFHMDF POS=(10,4),LENGTH=2,GRPNAME=DATE
SEP2    DFHMDF POS=(10,6),LENGTH=1,GRPNAME=DATE,INITIAL='- '
YR      DFHMDF POS=(10,7),LENGTH=2,GRPNAME=DATE
```

이러한 정의는 기호 출력 맵에 다음을 생성합니다.

```
02 DATE.
03 FILLER PICTURE X(2).
03 MOA    PICTURE X.
03 MOO    PIC X(2).
03 SEP1   PIC X(1).
03 DAO    PIC X(2).
03 SEP2   PIC X(1).
03 YRO    PIC X(2).
```

그룹 필드를 사용할 때 몇몇 규칙을 관찰해야 합니다.

- 단 하나의 속성 바이트만이 있습니다. 이는 전체 그룹 필드에 선행하고 전체 필드에 적용됩니다. 이를 첫 번째 서브필드(여기에서는 MO)의 DFHMDF 매크로에서 단 한번만 지정하십시오.
- 단 하나의 속성 바이트만이 있기 때문에 커서는 그룹 필드가 단일 필드인 것처럼 행동합니다. 이 예제에서 커서는 월의 마지막 위치에서 일의 첫 위치로, 또는 일에서 연도로 이동하지 않고, 하이픈을 건너뛰니다. 이는 그룹이 실제로 하드웨어에 관한 단일 필드이기 때문입니다. 이는 구성요소 서브필드에 대한 프로그램 액세스를 위해서만 세분화됩니다.
- 첫 번째 뒤의 서브필드에는 속성 바이트가 없지만, 예제에 표시된 대로 마치 있는 것처럼 POS 옵션을 정의하십시오. 즉, POS는 서브필드가 시작되기 전에 하나의 문자를 가리키고, 우리 예제에서처럼, 이전 서브필드의 마지막 문자를 겹쳐쓸 수 있습니다.
- 이 예제에서는 모든 구성요소 서브필드가 인접하지만 꼭 그럴 필요는 없습니다. 서브필드 간의 갭에 다른 필드를 정의하지 않는 한 갭이 있을 수 있습니다. 그룹 필드는 첫 번째 서브필드부터 마지막까지 모든 열에 걸쳐 있고, 서브필드가 화면에 나타나는 순서대로 구성요소 DFHMDF 매크로를 놓아야 합니다. 그룹은 해당 이름을 지정하지 않는 첫 번째 DFHMDF 매크로로 끝납니다.

- 예제의 SEP1 및 SEP2 서브필드에 했던 것처럼 서브필드를 참조할 의도가 없더라도 모든 서브필드에 필드 이름을 지정해야 합니다.
- 그룹 필드나 해당 구성요소에 OCCURS 옵션(다음 섹션에서 설명됨)을 사용할 수 없습니다.

반복된 필드: OCCURS 옵션

OCCURS 옵션은 필드에 표시된 항목 수가 맵에 생성됨을 지정하고, 맵 정의가 필드를 매트릭스 또는 배열에 항목으로서 주소 지정 가능한 방식으로 생성됨을 지정합니다.

이 태스크 정보

때로 화면에는 프로그램의 배열로서 처리하려는 일련의 동일한 필드가 포함됩니다. 예를 들어, 점원이 새 고객에게 사용되지 않은 전화 번호를 지정할 때 사용할 40개의 표시장치를 작성해야 한다고 가정해 보겠습니다. (이 아이디어는 고객에게 몇몇 선택사항을 주는 것입니다.) 또한 최근까지 사용되었던 번호를 강조 표시하여 고객에게 이전 소유자에게 걸려오는 전화가 있을 수 있음을 경고하고 싶을 수도 있습니다.

단일 필드 정의가 있는 전화 번호를 표시하는 화면의 부분을 정의할 수 있습니다.

```
TELNO DFHMDF POS=(7,1),LENGTH=9,ATTRB=NORM,OCCURS=40
```

이 명령문은 40개의 연속하지만 별개인 표시장치 필드를 생성하며, 위치(7,1)에서 시작하여 많은 행을 필요로 하므로 여러 행을 건너 진행할 수 있습니다(여기에서는 5개). 화면 폭을 고르게 나누는 길이(속성 바이트를 추가하여)를 선택했으므로 우리의 숫자는 세로 열에 나타나고 행 경계 전체에 분할되지 않습니다. 사용자가 지정하는 속성 및 초기 값 또한 각 필드에 적용됩니다.

기호 맵에서 이러한 필드의 설명은 COBOL에서 다음처럼 보입니다.

```
02 TELNOG OCCURS 40.
03 FILLER PICTURE X(2).
03 TELNOA PICTURE X.
03 TELNOO PIC X(9).
```

이 구조를 사용하면 다음과 같이 프로그램(또는 다른 소스)의 배열에서 맵을 채울 수 있습니다.

```
PERFORM MOVENO FOR I FROM 1 THROUGH 40.
...
MOVENO.
MOVE AVAIL-NO (I) TO TELNOO (I).
IF DAYS-SINCE-USE (I) < 90, MOVE DFHMBRY to TELNOA (I).
```

(DFHMBRY는 필드 강도를 밝게로 설정하기 위해 CICS 제공하는 상수입니다. 942 페이지의 『속성 값 정의: DFHBMSCA』에서 보다 자세히 설명합니다.)

OCCURS 필드의 레이블은 CICS가 지원하는 서로 다른 언어마다 약간씩 다르지만 기호는 동일합니다.

OCCURS 옵션이 작성하는 배열의 각 요소는 단일 맵 필드입니다. 일련의 필드를 반복해야 하는 경우에는(즉, 구조의 배열) OCCURS를 사용할 수 없습니다. 프로그램에서 이러한 배열을 사용하려면 OCCURS 없이 모든 필드를 개별적으로 정의하여 필요한 물리적 맵핑을 생성해야 합니다. 그런 다음 결과로 나오는 기호 맵을 수정하여 개별 필드 정의를 배열의 요소가 반복해야 하는 구조인 배열로 교체할 수 있습니다. 개정된 기호 맵이 원본과 동일한 필드 구조가 있는지 확인해야 합니다. 대안은 이러한 배열을 직접 정의할 수 있는 SDF II를 사용하는 것입니다.

블록 데이터

BMS는 블록 데이터 형식이라고 하는 기호 맵에 대한 대체 형식을 제공하며, 이는 특정 상황에서 유용할 수 있습니다. 블록 데이터 형식에서, 기호 출력 맵은 터미널로 이동하는 화면 또는 페이지의 이미지입니다.

기호 출력 맵에는 각각의 이름 지정된 맵 필드에 대한 전형적인 필드 속성(A)과 출력 값(O) 서브필드가 있지만, 각 맵 필드에 대한 서브필드는 기호 맵 구조에서의 해당 이동이 화면에서의 위치에 해당되는 것처럼 필드 필드로 구분됩니다. 길이 서브필드가 없고, 기호 커서 위치지정은 결과로 사용할 수 없습니다.

예를 들어, 914 페이지의 그림 143 의 “빠른 확인” 화면에 대한 기호 맵은 블록 데이터 형식에서 다음과 유사합니다(맵 80 열 너비 가정). 동일한 맵 정의의 보통 “필드 데이터” 형식(915 페이지의 그림 144에 있는)과 이 화면을 비교해 보십시오.

```
01 QCKMAPO.
02 FILLER PIC X(12).          <---TIOAPFX still present
02 FILLER PICTURE X(192).     <---Spacer
02 ACCTNOA PICTURE X.         <---Position (3,13)
02 ACCTNOO PIC X(7).
02 FILLER PICTURE X(72).     <---Spacer
02 SURNAMEA PICTURE X.       <---Position (4,13)
02 SURNAMEO PIC X(15).
02 FNAMEA PICTURE X.         <---Position (4,30),
02 FNAMEO PIC X(10).
02 FILLER PICTURE X(52).     <---Spacer
02 CHGA PICTURE X.           <---Position (5,13)
02 CHGO PIC $,$$0.00
02 FILLER PICTURE X(139).    <---Spacer
02 MSGA PICTURE X.           <---Position (7,1).
02 MSGO PIC X(30).
```

그림 147. 블록 데이터 형식에서 “빠른 확인”에 대한 기호 맵

프로그램에서 필드 속성만 설정할 수 있습니다. BMS는 맵에서 DSATTS 옵션을 무시하고, 블록 데이터 형식으로 확장 속성에 대한 서브필드를 생성하지 않습니다. 입력에 대해서는 블록 데이터를 사용할 수 있습니다. 입력 맵은 출력 맵과 구조에서 동일합니다. 플래그(F)가 필드 속성(A) 서브필드를 바꾸고, 입력(I)은 필드 형식에서와 같이 출력(O) 서브필드를 바꾼다는 것이 예외입니다.

블록 데이터 형식은 애플리케이션이 빌드되었거나 화면에 표시해야 할 프린터 페이지 이미지에 대한 액세스를 가지고 있는 경우에 유용할 수 있습니다. 그러나 대부분의 상황에서, 보통의 필드 데이터 형식은 더 큰 기능과 유연성을 제공합니다.

비3270 터미널에 대한 지원

비3270 터미널에 대한 지원에는 표준 BMS가 필요합니다. 최소 BMS는 3270 표시장치와 프린터만 지원합니다.

최소 BMS는 3270 표시장치와 프린터만 지원합니다. 이 범주에는 3178, 3290, 8775 및 5520, LU 유형 2 및 LU 유형 3 디바이스, 3270 데이터 스트림을 허용하는 기타 터미널이 포함됩니다. *IBM 3270 Information Display System 데이터 스트림 프로그램 래퍼의 참조서 매뉴얼*에 전체 목록이 포함되어 있습니다.

표준 BMS는 SCS 프린터(3270 데이터 스트림을 사용하지 않는 3270 제품군 프린터)와 935 페이지의 표 49에 나열된 모든 터미널 유형으로 3270 지원을 확장합니다. BMS 및 SCS 데이터 스트림에 대한 자세한 정보는 884 페이지의 『비3270 CICS 프린터』의 내용을 참조하십시오.

이러한 터미널 유형 간의 기능상 차이점으로 인해 각각에 대해 BMS가 똑같은 방법으로 작동하게 하는 것은 불가능합니다. 이어지는 절에서는 특정 기능에 대한 하드웨어 기초가 부족한 디바이스에서 BMS 사용 시 제한사항을 설명합니다.

비3270 디바이스에 대한 출력 고려사항

BMS는 출력 데이터 스트림의 디바이스 종속 콘텐츠를 논리 콘텐츠와 분리하므로 BMS 출력 작성 시 고려해야 하는 3270 디바이스와 비3270 디바이스 간의 차이점은 몇 가지에 불과합니다.

3270 디바이스와 비3270 디바이스 간의 주요 차이점은 3270은 필드 중심이라는 점인데, 기타 디바이스는 대부분 그렇지 않습니다. 따라서 비3270 터미널로 전송된 출력 필드와 연관된 필드 속성과 확장 속성이 없습니다. BMS는 필드별로 출력을 올바른 위치에 배치할 수 있지만 필드 구조는 데이터 스트림에 반영되지 않습니다. BMS는 일부 터미널에서 일부 필드 속성을 에물레이트할 수도 있지만(예를 들어, 강조표시된 필드에 밑줄 표시) 수정된 데이터 태그가 없고 필드에 입력에 대한 보호도 없습니다.

터미널이 지원하지 않는 출력에 속성을 지정하는 경우 BMS는 속성을 무시합니다. 누락된 기능이 없어도 출력을 이해할 수 있으면 속성에 대해 걱정하지 않아도 됩니다.

입력 시 차이점

BMS의 많은 기능은 수정된 해당 필드만 필드별로 읽는 기능에 좌우되므로 필드 구조의 부재는 출력 작업보다 입력 작업에 더 많은 영향을 줍니다. 하드웨어가 화면에 해당 위치와 함께 필드별로 입력을 제공하지 않으면 사용자가 해당 정보를 제공해야 합니다.

다음 두 가지 방법으로 이를 수행할 수 있습니다. 첫 번째는 맵핑 정의의 FLDSEP 옵션에 1 - 4자 길이의 필드 분리문자 순서를 정의하는 것입니다. 입력의 각 필드 사이에 이 순서를 배치하고 화면이나 페이지에 나타나는 것과 동일한 순서로 입력 필드를 제공하십시오. 데이터가 포함된 마지막 필드까지 화면의 모든 필드를 제공해야 합니다. 필드에 입력이 없을 경우 종료 필드 분리문자 순서로만 구성될 수 있습니다. 하드카피 디바이스에서는 종이 이동으로 인해 입력이 출력을 오버레이할 수 없습니다. 이러한 터미널을 에뮬레이트하는 표시장치에서는 일반적으로 동일한 기술이 사용됩니다. 입력 필드는 필드 분리문자 순서로 분리되어 순서대로 용도에 맞게 예약된 영역에 입력됩니다.

두 번째 방법은 입력에 제어 문자를 포함하는 것입니다. 맵에서 FLDSEP 옵션을 생략하면 BMS가 제어 문자를 사용하여 "페이지"의 데이터 위치를 계산하고 그에 따라 맵핑합니다. BMS가 인식하는 제어 문자는 다음과 같습니다.

NL	줄 바꾸기	X'15'
IRS	교환 레코드 분리문자	X'1E'
LF	줄 바꾸기	X'25'
FF	용지 공급	X'0C'
HT	수평 탭	X'05'
VT	수직 탭	X'0B'
CR	캐리지 리턴	X'0D'
RET	TWX에서 리턴	X'26'
ETB	텍스트 블록 종료	X'26'
ESC	이스케이프(2780의 경우)	X'27'

RECEIVE MAP 명령을 사용하여 이런 유형의 데이터를 읽는 경우 실제 3270 입력과 몇 가지 차이점이 있습니다.

- 플래그 바이트(F 서브필드)가 설정되지 않습니다. 널(null)을 포함합니다. 운영자가 필드를 지웠는지 또는 커서가 필드에 남아 있었는지 여부를 판별할 수 없습니다.
- 입력 시 필드가 리턴되도록 출력에 수정된 데이터 태그를 사전 설정할 수 없습니다.

비3270 터미널용 특수 옵션

BMS는 데이터 스트림을 단축하는 디바이스 기능을 이용할 수 있도록 비3270 디바이스에 형식화 옵션을 추가로 제공합니다.

이러한 옵션은 다음과 같습니다.

- 수직 및 수평 탭. 디바이스가 지원하는 경우 수평 및 수직 탭 순서를 사용하여 출력을 배치할 수 있습니다. 탭 문자는 HTAB 및 VTAB 옵션을 통해 맵핑 세트 정의에 정의됩니다. 다음 수평 탭에 배치할 경우 데이터에 HTAB 문자를 포함시키십시오. 데이터에 VTAB 문자를 제공하여 다음 수직 탭에 배치하십시오. BMS는 이러한 문자를 특정 프린시플 기능에 필요한 탭 순서로 변환합니다.

BMS 출력에서 탭을 사용하기 전에 동일한 터미널에서 태스크 또는 일부 이전 태스크가 필수 위치에 탭을 설정했어야 합니다. 이는 일반적으로 731 페이지의 제 26 장 『데이터 전송 명령 사용』에 설명된 터미널 제어 SEND 명령을 사용하여 수행됩니다.

- 외부 형식화. 일부 논리 디바이스는 형식 정보를 저장하고 형식화 프로세스에 참여할 수 있습니다. 따라서 BMS는 훨씬 적은 데이터(본질적으로 기호 맵 콘텐츠를)를 전송하고 물리적 및 기호 맵을 논리 디바이스에 병합하는 작업을 위임할 수 있습니다. 세 부사항은 1016 페이지의 『아웃보드 형식화』의 내용을 참조하십시오.
- NLEOM(줄 바꾸기, 메시지 끝). 표준 BMS는 3270 버퍼 제어 순서가 아닌 공백 및 줄 바꾸기(NL) 문자를 사용하여 BMS가 출력을 형식화도록 요청하는 옵션도 제공합니다. 이 기술은 882 페이지의 『NLEOM 옵션』에 설명된 대로 프린터의 페이지 너비 설정에 있어 더 많은 유연성을 제공합니다.

디바이스 종속 맵

맵 필드의 위치, 기본 속성, 기본 콘텐츠는 물리적 맵핑에만 나타나고 기호 맵에는 나타나지 않기 때문에 단일 프로그램을 사용하여 동일한 변수 정보를 포함하지만 화면의 배열마다 다른 상수 정보를 포함하는 맵을 빌드할 수 있습니다.

작성하는 프로그램이 특성이 다른 여러 개의 디바이스를 지원해야 하는 경우 디바이스 종속 맵이 아주 편리합니다. 이름은 동일하지만 속성과 레이아웃이 다른 여러 개의 맵을 정의하여(각각 접미부가 다름) 여러 개의 디바이스를 지원하도록 하십시오.

예를 들어, "빠른 갱신" 트랜잭션을 사용하는 일부 직원은 지금까지 가정한 대로 3270 모델 2를 사용하고 나머지는 3개의 행과 40개의 열만 있는 특수 목적 터미널을 사용한다고 가정하십시오. 큰 화면에 맞게 설계한 형식은 작은 화면에는 적합하지 않지만 재배열할 경우 정보는 적합합니다.

QUP Quick Account Update:
Current charge okay; enter next
Acct: _____ Charge: \$ _____

그림 148. 작은 화면용 "빠른 갱신"

필요한 맵핑 정의는 다음과 같습니다.

```

QUPSET  DFHMSD TYPE=MAP,STORAGE=AUTO,MODE=INOUT,LANG=COBOL,SUFFIX=9
QUPMAP  DFHMDI SIZE=(3,40),LINE=1,COLUMN=1,CTRL=FRECKB
        DFHMDF POS=(1,1),LENGTH=24,ATTRB=(ASKIP,BRT),                X
        INITIAL='QUP Quick Account Update'
MSG      DFHMDF LENGTH=39,POS=(2,1),ATTRB=(ASKIP,NORM)
        DFHMDF POS=(3,1),LENGTH=5,ATTRB=(ASKIP,NORM),                X
        INITIAL='Acct:'
ACCTNO   DFHMDF POS=(3,11),LENGTH=6,ATTRB=(UNPROT,NUM,IC)
        DFHMDF POS=(3,18),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
        DFHMDF POS=(3,20),LENGTH=7,ATTRB=(ASKIP,NORM),INITIAL='Charge:'
CHG       DFHMDF POS=(3,29),LENGTH=7,ATTRB=(UNPROT,NORM),PICIN='$$$$0.00'
        DFHMDF POS=(3,37),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
        DFHMSD TYPE=FINAL

```

그림 149. 맵핑 정의

이름을 가진 필드는 동일한 이름과 동일한 길이를 갖고 맵핑 정의에 동일한 순서로 나타나기 때문에 이 버전의 맵을 어셈블링하여 생성된 기호 맵핑 세트는 953 페이지의 『입출력(I/O) 예제』에 표시된 기호 맵핑 세트와 동일합니다. (화면에 동일한 순서로 나타나지 않아도 됩니다. 이름 지정된 필드의 정의를 모든 맵에서 동일한 순서로 유지한다면 재배열할 수 있습니다.) 오직 하나의 기호 맵 사본만 필요하며 동일한 코드를 사용하여 맵을 빌드할 수 있습니다.

CICS는 트랜잭션이 실행되고 있는 터미널에 대한 TYPETERM 자원 정의의 ALTSUFFIX 옵션에 코드화된 값을 사용할 물리적 맵핑을 선택합니다. 트랜잭션 PROFILE 자원 정의에 SCRNSZE(ALTERNATE)도 지정해야 합니다. TYPETERM 및 PROFILE 자원 정의에 대한 정보는 *CICS 자원 정의 안내서*를 참조하십시오.

이 기술을 사용하여 특수 목적으로 사용되는 표준 터미널을 구별할 수 있습니다. 예를 들어, 영어 사용자와 프랑스어 사용자가 둘 다 애플리케이션을 사용한 경우 두 세트의 물리적 맵핑(프랑스어로 된 상수가 있는 물리적 맵핑과 영어로 된 물리적 맵핑)을 작성할 수 있습니다. 각각에 접미부를 지정하고 영어 접미부를 영어 터미널의 정의에 있는 ALTSUFFIX 값으로 지정하고 프랑스어 터미널에는 프랑스어 접미부를 지정하십시오. 맵을 사용하는 트랜잭션은 대체 화면 크기를 지정한 PROFILE을 가리킵니다. 그러면 맵을 전송할 때 BMS가 터미널과 일치한 접미부를 가진(즉, 해당 언어로 된) 버전을 선택합니다.

디바이스 종속 맵을 제공하는 또 다른 방법은 BMS가 터미널 유형을 기반으로 접미부를 생성하고 SEND MAP 또는 RECEIVE MAP을 실행할 때 현재 실행에서 터미널과 일치시키는 데 사용할 물리적 맵핑을 선택하도록 허용하는 것입니다.

디바이스 종속 자원

디바이스 종속 지원(DDS)은 디바이스 종속 맵을 가능하게 하는 설치 기능입니다.

맵핑 세트를 어셈블할 때 맵에 대해 TERM 옵션에 터미널 유형을 지정하십시오. 이렇게 하면 어셈블러가 해당 터미널 유형에 대한 문자가 접미부로 붙은 MAPSET 이름

아래에 물리적 맵핑 세트를 저장합니다. 또한 JCL 또는 링크 편집 NAME문을 사용하여 맵핑 세트가 저장되는 멤버 이름을 제어할 수 있습니다. DDS가 활성화된 상태에서 SEND MAP 또는 RECEIVE MAP을 실행하는 경우, BMS는 MAPSET 옵션에 사용자가 제공하는 이름에 1자 접미부를 추가합니다. 터미널 정의를 기반으로 접미부를 선택하므로 주어진 실행에 대한 터미널에 해당하는 물리적 맵핑을 로드합니다.

BMS는 일반적인 터미널 유형에 사용되는 접미부를 정의합니다. 예를 들어, 화면 크기가 24행 80열인 3270 모델 2에는 문자 'M'이 지정됩니다. 표 49에 표시된 표준 유형 중 하나이면 TYPETERM 정의를 통해 유형을 판별합니다.

표 49. BMS용 터미널 코드

코드	터미널 또는 논리 장치
A	CRLP(카드 판독기 입력, 라인 프린터 출력)
B	자기 테이프
C	순차 디스크
D	TWX 모델 33/35
E	1050
F	2740-1, 2740-2(버퍼 수신 없음)
G	2741
H	2740-2(버퍼 수신 있음)
I	2770
J	2780
K	3780
L	3270-1 표시장치(40자 너비)
M	3270-2 표시장치(80자 너비), LU 유형 2
N	3270-1 프린터
O	3270-2 프린터, LU 유형 3
P	모든 대화식 LU, 3767/3770 해독기 LU, 3790 전 기능 LU, SCS 프린터 LU
Q	2980 모델 1 및 2
R	2980 모델 4
U	3600(3601) LU
V	3650 호스트 대화식(3653) LU
W	3650 해독기 LU
X	3650 호스트 대화식(3270) LU
Y	3770 일괄처리 LU, 3770 및 3790 일괄처리 데이터 교환 LU, LU 유형 4
공백	3270-2(TERM 생략 시 기본값)

설치는 추가 터미널 유형(예: 앞서 설명한 미니어처 화면)도 정의할 수 있습니다. 시스템 프로그래머는 터미널 유형에 ID를 지정하고 터미널에 대한 TYPETERM 정의의 ALTSUFFIX 옵션에 이를 지정하여 이것을 수행합니다. 이러한 터미널에 대한 맵을 작

성하는 경우 TERM 옵션을 사용하는 대신 SUFFIX 옵션에 이 ID를 지정하십시오. 맵을 사용하는 트랜잭션은 대체 화면 크기를 지정하는 PROFILE도 가리켜야 하므로 ALTSUFFIX가 사용됩니다.

DDS를 사용할 경우 BMS가 물리적 맵핑 선택에 사용하는 규칙은 다음과 같습니다.

- BMS는 해당 정의가 ALTSUFFIX와 ALTSCREEN을 둘 다 지정하고 트랜잭션 PROFILE이 대체 크기를 요구하거나 기본 크기와 대체 크기가 같기 때문에 트랜잭션에 대한 화면 크기가 대체 크기인 경우 맵핑 세트 이름에 터미널 정의의 ALTSUFFIX 값을 추가합니다.
- 이러한 조건이 충족되지 않거나 BMS가 해당 접미부를 가진 맵을 찾을 수 없는 경우에는 터미널 정의의 터미널 유형에 해당하는 접미부를 가진 맵을 찾으려고 시도합니다.
- BMS가 해당 맵도 발견할 수 없는 경우에는 접미부가 없는 맵을 찾습니다(공백 접미부는 이를 사용할 수 있는 모든 터미널에 적합한 범용 맵을 표시함).

DDS를 사용하지 않을 경우 BMS는 항상 접미부가 없는 맵을 먼저(유일하게) 찾습니다.

디바이스 종속 지원은 시스템 프로그래머가 시스템 초기화 테이블에 설정한 BMS 설치 옵션입니다. 이를 이용하기 전에 시스템에 포함되어 있는지 확인하십시오. 하나의 디바이스 유형만 지원하긴 하지만 존재 여부를 알아야 합니다.

시스템에서 DDS를 사용하면, 하나의 디바이스 유형만 지원하지만 접미부가 붙은 맵핑 세트 작성 시 효율성 이점이 있습니다. 그 이유는 BMS가 범용 맵핑 세트(공백 접미부)를 기본값으로 설정하기 전에 존재하지 않는 맵핑 세트를 로드하려고 시도하지 못하게 하기 때문입니다.

반면에 DDS를 사용하지 않으면, BMS가 범용 접미부(공백)를 찾고 접미부가 붙은 맵을 찾는 데 실패하기 때문에 맵에 접미부를 붙이는 것은 불필요합니다.

터미널 정보 찾기

BMS의 전체 디자인 및 특히 디바이스 종속적인 지원 때문에 일반적으로 터미널의 형식에 대해서 자세히 알 필요는 없습니다. 그러나 프린시פל 기능의 특성을 알아야 하는 경우에 ASSIGN 및 INQUIRE 명령을 사용할 수 있습니다.

예를 들어, 터미널이 특정 확장 속성을 지원하는지, 어떤 자국어가 사용 중인지, 화면 크기 등등을 알 수 있습니다. 이러한 정보 유형은 사용자가 터미널과 통신하기 위해 BMS 또는 터미널 제어를 사용 중인지 여부에 적용됩니다. 터미널 제어의 경우 좀 더 자주 필요하므로 적용되는 옵션은 739 페이지의 제 29 장 『터미널 정보 찾기』에 설명되어 있습니다.

BMS에 고유한 ASSIGN 옵션도 있지만 ACCUM 옵션을 사용할 때 보다 자주 필요하므로, 이들은 나중에 978 페이지의 『누적 처리에 대한 ASSIGN 옵션』에 설명됩니다.

BMS 맵핑된 출력 전송

BMS 맵핑된 출력을 전송하려면 먼저 다음 단계를 수행하여 출력 데이터를 맵 구조로 이동해야 합니다.

이 태스크 정보

기호 맵핑 세트를 어셈블한 후에는 코딩할 준비가 됩니다. 916 페이지의 『맵 작성』에서 사용된 예제는 애플리케이션에서 맵으로 데이터를 가져오는 방법을 설명합니다. 이 프로세스는 지금 보다 자세하게 설명되고, 수행해야 하는 모든 단계를 설명하고, 가지고 있는 옵션에 대해 자세히 설명해 줍니다.

맵핑된 출력을 생성하려면 다음 단계를 수행해야 합니다.

프로시저

1. 맵을 빌드할 저장영역을 획득하십시오.
2. 이 저장영역의 구조를 정의할 수 있도록 기호 맵핑 세트를 복사하십시오.
3. 이를 초기화하십시오.
4. 출력 데이터를 맵 구조로 이동하십시오.
5. 필드 속성을 설정하십시오.
6. SEND MAP 명령을 사용하여 맵을 화면에 쓰고 필요한 디바이스 제어 정보를 추가하십시오.

다음에 수행할 작업

관련 개념:

911 페이지의 제 45 장 『BMS(Basic Mapping Support)』

BMS(Basic Mapping Support)는 CICS 프로그램과 터미널 디바이스 사이의 API(Application Programming Interface)입니다.

맵의 저장영역 획득 및 정의

맵핑된 출력을 작성하는 첫 번째 단계는 프로그램이 BMS에 전달하는 변수 맵 데이터를 배열하는 저장영역을 제공하기 위한 것입니다.

이 태스크 정보

맵 구조를 작업 중인 저장영역에 배치하면, CICS는 자동으로 할당합니다. (CICS는 프로그램의 각 실행별 작업 중인 저장영역의 개인용 사본을 할당하므로 295 페이지의 『프

로그래밍 저장영역』에 설명된 대로 한 태스크의 데이터를 다른 태스크의 데이터와 혼동하지 않을 수 있습니다.) 작업 중인 저장영역을 사용하려면 해당 용도로 제공된 언어 명령문을 사용하여 기호 맵핑 세트를 복사하십시오.

```
COPY          in COBOL and assembler
%INCLUDE      in PL/I
#include       in C and C++
```

작업 중인 저장영역은 COBOL에 WORKING-STORAGE SECTION에 있으며 CICS 어셈블러 프로그램에 PL/I, C, C++, 및 DFHEISTG로 되어 있는 자동 저장영역입니다. 예를 들어, 다음과 같습니다.

```
WORKING-STORAGE SECTION.
...
01 COPY QCKSET.
...
```

또는 필요에 따라 CICS GETMAIN 명령을 사용하여 맵핑 세트 저장영역을 구하고 해제할 수도 있습니다. (GETMAIN은 787 페이지의 『저장영역 제어』에 설명되어 있습니다.) 이 경우에는 맵을 포인터 변수가 지시하는 저장영역으로 복사하십시오(어셈블러의 DSECT에서 PL/I, C 및 C++로 된 저장영역을 기반으로 하는 COBOL의 LINKAGE SECTION). GETMAIN으로부터 리턴 시에 프로그래밍 언어의 기능에 따라 SET 옵션에 리턴된 주소를 사용하여 저장영역을 데이터 구조와 연관시킬 수 있습니다.

916 페이지의 그림 145에서 예제로 작업 중인 저장영역을 사용했지만 GETMAIN을 사용할 수도 있었습니다. 그랬다면 코드는 다음과 같이 변경될 것입니다.

```
LINKAGE SECTION.
...
01 COPY QCKSET.
...
PROCEDURE DIVISION.
...
MOVE LENGTH OF QCKMAPO TO LL.
EXEC CICS GETMAIN SET(ADDRESS OF QCKMAPO)
        LENGTH(LL) END-EXEC.
...
```

GETMAIN 명령에서 필요한 길이는 이름에 "O" 문자가 접미사로 붙어 있는 변수의 길이입니다. COBOL, PL/I, C 및 C++에서는 이전 예제에서처럼 언어 기능을 사용하여 이 길이를 판별할 수 있습니다. 어셈블러에서는 이는 "L" 접미부가 붙어 있는 맵 이름이 레이블인 EQUate 문에 정의되어 있습니다.

BASE 및 STORAGE 옵션

DFHMSD 맵핑 세트 정의 매크로에 대한 두 개의 옵션인 BASE 및 STORAGE=AUTO는 맵의 저장영역이 정의되는 방법에 영향을 미칩니다. (STORAGE 옵션에는 항상 AUTO 값이 있습니다.) 둘 중 하나를 사용하거나 둘 모두를 사용하지 않을 수 있으므로 세 번째 가능성이 있습니다.

몇몇 맵을 포함하는 맵핑 세트에 아무런 옵션도 지정하지 않은 경우에는 맵의 기호 구조는 서로 오버레이할 수 있도록 정의됩니다. STORAGE=AUTO를 지정하는 경우에는 그렇지 않습니다. 각각이 별도의 공간을 차지합니다. 따라서 STORAGE=AUTO에는 더 많은 저장영역이 필요합니다.

그러나 단일 프로그램에서 서로 오버레이하는 맵을 사용할 때에는 이를 직렬로 사용하거나 프로그래밍하여 저장영역을 재사용할 수 있도록 보충해야 합니다. 저장영역이 중요한 문제이지 않는 한 STORAGE=AUTO는 프로그래밍을 단순화하고 오류의 위험성을 줄입니다.

PL/I, C 및 C++에서, STORAGE=AUTO에는 맵을 자동 저장영역(CICS가 할당하는 저장영역)으로서 정의하는 추가적인 효과가 있습니다. STORAGE=AUTO가 없으면 이러한 컴파일러는 기본 저장영역을 가정합니다. 이로 인해 일반적으로 추가 GETMAIN의 오버헤드가 발생합니다. BMS는 BASE 옵션이 있는 또 다른 이름을 지정하지 않는 한 BMSMAPBR 이름을 연관된 포인터 변수에 지정합니다.

세 번째 가능성인 BASE를 사용하여 여러 맵핑 세트에서 모든 맵에 동일한 저장영역을 사용할 수 있습니다. 이의 효과는 프로그래밍 언어마다 약간씩 다르지만 본질적으로 동일한 BASE 값이 있는 맵핑 세트의 모든 맵은 서로 오버레이합니다. COBOL에서, BASE=xxxx는 01 레벨(즉, 각 개별 맵)이 REDEFINES xxxx 절을 포함하게 합니다. PL/I, C 및 C++에서 이는 각 맵을 포인터 변수 xxxx를 기반으로 하는 저장영역으로 지정합니다. BASE는 프로그래밍 언어가 어셈블러일 때에는 사용할 수 없습니다.

출력 맵 초기화

출력 빌드를 시작하기 전에 맵 저장영역이 널로 초기화되었는지 확인하십시오. 그러면 이전 프로세스에 의해 남아 있는 데이터가 부주의하게 사용되는 일이 없게 됩니다.

이 태스크 정보

이 동일 맵을 사용하거나 오버레이된 맵을 사용하여 입력 데이터를 읽은 경우에는 이 데이터를 먼저 처리했거나 저장했는지 확인해야 합니다. 입력과 출력 맵 간의 관계는 956 페이지의 『기호 입력 맵』에서 설명되고 입력에 사용한 것과 동일한 맵을 사용하는 것이 965 페이지의 『맵핑된 입력 후 맵핑된 출력 전송』에서 설명됩니다.

널(X'00')을 구조로 이동하여 초기화합니다. 기호 맵 구조는 O 문자가 접미부로 붙은 명명된 전체 맵 영역을 가리킬 수 있도록 정의되어 있습니다. 915 페이지의 그림 144에서 이를 볼 수 있으며 실제로 명령문은 다음과 같습니다.

```
MOVE LOW-VALUES TO QCKMAP0.
```

"빠른 확인" 예제에서 맵을 빌드한 영역을 지웁니다. 입력과 출력 둘 모두에 맵을 사용한 경우에는 입력을 편집할 때 한 번에 하나의 필드씩 맵을 지우는 것이 쉬울 수 있습니다(963 페이지의 『입력 오류 처리』 참조).

CICS GETMAIN 명령어를 사용하여 맵 저장영역을 구할 때 초기화하는 또 다른 방법은 INITIMG 옵션을 사용하는 것입니다.

변수 데이터를 맵으로 이동

확득한 맵의 저장영역을 가지고, 저장영역에 맵 구조의 관계를 설정했고, 초기화하면 드디어 출력을 작성할 준비가 된 것입니다. 여기에는 데이터 자체와 해당 표시장치 속성의 두 개의 파트가 있습니다. 먼저, 데이터에 대해 설명한 다음 속성에 대해 설명합니다.

일반적인 경우에는 출력 표시는 몇몇 상수 또는 기본 데이터(물리적 맵핑이 제공)와 몇몇 변수 데이터(프로그램이 제공)로 구성됩니다. 프로그램이 제공하려는 각 필드의 경우 데이터를 이름이 O 문자가 접미부로 붙어 있는 맵에 지정된 이름인 기호 맵의 필드로 이동합니다. 예제는 BMS 출력 예제의 코드를 참조하십시오.

필드에 값을 제공하지 않은 경우에는, 즉, 초기화될 때처럼 널(null)로 남겨두면 BMS는 보통 맵에 지정된 초기값을 사용합니다(있는 경우). 상수(즉, 이름 없는 필드)는 또한 맵에 지정된 초기값을 얻습니다. 그러나 SEND MAP 명령에서 DATAONLY 및 MAPONLY 옵션은 프로그램 및 맵 데이터가 병합되는 방법을 수정합니다. 이러한 옵션을 945 페이지의 『기호 및 물리적 맵핑 병합』에서 설명하고 정확한 규칙을 946 페이지의 『출력 화면 빌드』에서 요약합니다.

표시장치 특성 설정

표시장치 속성은 출력 데이터의 두 번째 구성요소입니다. 이름 지정된 필드에 대해 BMS는 프로그램에 지정된 값 맵핑을 덮어쓰기 위해 사용할 수 있는 서브필드를 생성합니다.

BMS 출력 예제에서 맵 필드에 대한 3270 필드 속성이 ATTRB 옵션으로 정의되는 방법 및 필드의 이름이 지정된 경우 프로그램이 맵 값을 덮어쓰도록 BMS가 "A" 서브필드를 생성하는 방법에 대해 알아보았습니다. (속성에 대한 정보는 761 페이지의 『3270 필드 속성』의 내용을 참조하십시오.)

모든 3270 디바이스가 필드 속성을 지원하기 때문에 BMS는 항상 A 서브필드를 제공합니다. 많은 3270이 941 페이지의 표 50에 표시되는 일부 확장 속성도 포함합니다. BMS는 필드 속성을 집합적으로 지원하는 것처럼 이 속성 각각도 개별적으로 지원합니다. 속성 값은 DFHMDF 필드 정의에 지정하고 필드 이름을 지정하는 경우 BMS가 기호 맵에 서브필드를 생성하여 프로그램에 맵이 지정하는 값을 덮어쓸 수 있습니다. 각 확장 속성 유형에 대해 별도의 서브필드가 있습니다.

DFHMDI 또는 DFHMDS의 DSATTS 옵션에 필요한 속성을 지정하여 확장 속성에 대한 서브필드를 요청할 수 있습니다. MAPATTS 옵션에 확장 속성 목록도 포함해야 합니다(해당 속성 유형이 DFHMDF 매크로에 표시되지 않는 경우도 포함).

표 50. BMS 속성 유형. 열은 속성 유형, 연관된 MAPATTS 및 DSATTS 값 이름, 기호 맵의 연관된 서브필드 접미부를 표시합니다.

속성 유형	MAPATTS, DSATTS 값	서브필드 접미부
필드 속성	없음(기본값)	A
색상	COLOR	C
강조표시	HIGHLIGHT	H
아웃라인	OUTLINE	U
배경 투명도	TRANSP	T
유효성 검사	VALIDN	V
더블바이트 문자 기능	SOSI	M
프로그램화된 기호	PS	P

참고: 프로그램화된 기호를 사용하는 경우, 디바이스에서 영구적으로 로드되는 기호 세트를 선택한 경우를 제외하고는 적합한 기호 세트가 디바이스로 먼저 전송되었는지 확인해야 합니다. 터미널 제어 SEND 명령을 사용하여 이를 수행할 수 있습니다(731 페이지의 제 26 장 『데이터 전송 명령 사용』 참조). IBM 3270 정보 표시장치 시스템 데이터 스트림 프로그래머용 참조서 매뉴얼에서 전송할 항목에 대해 설명합니다.

적용되는 속성 유형은 실행 시의 프린시플 기능의 기능에 따라 달라집니다. 터미널이 소유하지 않는 속성 값을 지정하면 BMS가 이를 무시합니다. 다른 터미널 유형을 지원하는 경우에도 다른 기술을 사용하여 동일한 시각적 명확성을 얻어야 할 수도 있습니다. 739 페이지의 제 29 장 『터미널 정보 찾기』에서 설명하는 대로 ASSIGN 및 INQUIRE 명령으로 사용 중인 터미널 유형을 찾을 수 있습니다. 프로그램을 터미널 유형에 대해 독립적으로 유지하기 위한 프로비전도 BMS에 있습니다. 933 페이지의 『디바이스 종속 맵』의 내용을 참조하십시오.

속성 변경

맵 정의의 필드 속성은 MAPATTS 및 DSATTS 옵션을 사용하여 변경할 수 있습니다.

다음은 이것의 작동 방법에 대한 예제입니다. "빠른 확인" 애플리케이션에 있는 터미널에 색상 및 강조표시 기능이 있다고 가정해 보십시오. 허용된 최대 비용 필드를 화면의 나머지 부분과 다른 색상으로 표시하기로 결정할 수도 있습니다. 직원의 가장 큰 관심사이기 때문입니다. 또한 경고 메시지를 빨간색으로 표시할 수도 있습니다. 경고가 나타나면 직원이 이를 인지하는 것이 중요하기 때문입니다. 그리고 카드 분실의 이유로 직원의 주의를 얻고 싶을 때에는 프로그램에서 속성을 변경하여 메시지를 깜박이게 만들 수 있습니다. 이러한 기능을 추가하려면 맵 정의를 다음과 같이 변경해야 합니다.

```
QCKMAP DFHMDI SIZE=(24,80),...,X
MAPATTS=(COLOR,HIGHLIGHT),COLOR=GREEN,HIGHLIGHT=OFF,DSATTS=HIGHLIGHT
```

MAPATTS 옵션은 BMS에 맵 또는 프로그램에서 색상과 강조표시를 지정하도록 지시합니다. DSATTS에 나열된 속성은 MAPATTS에도 포함되어야 하기 때문입니다.

COLOR 및 HIGHLIGHT 값은 색상이 지정되지 않은 필드는 녹색이어야 하고 지정되지 않은 경우 강조표시가 꺼져 있어야 함을 표시합니다.

변경해야 하는 유일한 필드 정의는 녹색이 *아니거나* 강조표시된 것입니다.

```
CHG      DFHMDF POS=(5,13),LENGTH=8,ATTRB=(ASKIP,NORM),PICOUT='$,$$0.00',    X
          COLOR=NEUTRAL
MSG      DFHMDF LENGTH=20,POS=(7,1),ATTRB=(ASKIP,NORM),COLOR=RED
```

COLOR=NEUTRAL을 지정하면 터미널에서 필드가 흰색으로 표시됨을 의미합니다.

DSATTS 옵션은 실행 시에 몇몇 필드의 강조표시를 변경하고 싶으므로 이를 수행할 수 있도록 기호 맵에 "H"-접미부 서브 필드를 생성해야 함을 BMS에 지시합니다. 각 명명된 필드는 추가로 서브필드를 갖게 됩니다. 예를 들어, 메시지 필드는 915 페이지의 그림 144에서 현재 세 개의 행에서 다음으로 확장됩니다.

```
02 FILLER PICTURE X(2).
02 MSGH   PICTURE X.
02 MSGA   PICTURE X.
02 MSGO   PIC X(30).
```

깜박임을 생성해야 하는 프로그램 명령문은 다음과 같습니다.

```
MOVE DFHBLINK to MSGH.
```

일반적으로 BMS는 사용자가 이를 제공하는 경우 프로그램에서 속성 값을 얻고 사용자가 제공하지 않는 경우, 즉, 프로그램 값을 초기화된 것처럼 널(null)로 두는 경우에는 속성 값을 맵에서 얻습니다. 그러나 946 페이지의 『출력 화면 빌드』에서 '값이 나오는 위치'에서 설명된 대로 SEND MAP 명령에서 MAPONLY 및 DATAONLY 옵션은 속성 값뿐만 아니라 필드 데이터에 영향을 미칩니다.

속성 값 정의: DFHBMSCA

속성 값을 설정하는 데 필요한 1바이트 값은 3270 하드웨어가 정의하는 비트 조합입니다. CICS는 DFHBMSCA라는 소스 코드를 제공하고 이는 모든 속성에 일반적으로 사용되는 값을 정의하고 각 조합에 의미있는 이름을 지정합니다.

DFHBMSCA를 프로그램에 복사할 수 있습니다. 941 페이지의 『속성 변경』에서 DFHBLINK 프로그램이 예제입니다. DFHBLINK를 정의하려면 DFHBMSCA를 다음과 같이 작업 중인 저장영역으로 복사하십시오.

```
WORKING-STORAGE SECTION.
...
01 COPY DFHBMSCA.
```

DFHBMSCA의 버전은 각 프로그래밍 언어에 제공됩니다. 값 이름은 모든 버전에서 동일합니다. DFHBMSCA에 포함되지 않은 속성 조합이 필요한 경우에는 *3270 Data Stream Programmer's Reference*의 내용을 참조하십시오. 값을 자주 사용하는 경우에는 이를 포함하기 위해 DFHBMSCA를 수정할 수 있습니다.

어셈블러 언어에서만 값은 EQUates로 정의되므로 MVC 지시사항 대신에 , MVI를 사용하십시오.

SEND MAP 명령 사용

이 목록은 SEND MAP 명령이 BMS에 알려주는 내용을 보여줍니다.

SEND MAP 명령은 BMS에 다음을 알려줍니다.

- 사용할 맵(MAP 옵션) 및 해당 맵을 찾을 수 있는 위치(MAPSET 옵션)
- 맵의 변수 데이터를 찾을 수 있는 위치(FROM 옵션) 및 이를 맵의 값과 병합하는 방법(MAPONLY 및 DATAONLY)
- 데이터 스트림에 포함할 디바이스 제어 및 기타 제어 옵션
- 맵 정의에서 위치를 덮어쓰고 싶은 경우 커서를 둘 위치(CURSOR 옵션)
- 메시지가 완료되었는지 점증적으로 빌드되었는지 여부(ACCUM 옵션)
- 형식화된 출력으로 수행할 작업(TERMINAL, SET 및 PAGING 옵션)

MAP 및 MAPSET 옵션은 자체 설명적이며 단순 SEND MAP에 선행하는 프로그래밍 단계를 설명할 때 나머지 대부분을 다룹니다. 마지막 두 개의 주제에는 950 페이지의 『출력 배치 옵션: TERMINAL, SET, PAGING』에서 설명하는 BMS 논리 메시지 기능에 대한 지식이 필요합니다.

그 이전에는 기본값을 가정합니다. 즉, 각 SEND MAP은 하나의 메시지를 작성하고 우리는 이 메시지를 우리 소유의 터미널에 전송합니다.

관련 개념:

911 페이지의 제 45 장 『BMS(Basic Mapping Support)』

BMS(Basic Mapping Support)는 CICS 프로그램과 터미널 디바이스 사이의 API(Application Programming Interface)입니다.

SEND MAP 제어 옵션

BMS SEND 명령에는 많은 제어 옵션이 있습니다. 몇몇은 BMS의 특정 디바이스 또는 특수 기능에만 적용되고 연관된 디바이스 지원이나 기능에 도달하기 전까지는 이에 대한 설명을 유예합니다. 그러나 다음 디바이스 제어 옵션이 일반적으로 적용됩니다.

- 터미널에 디바이스 버퍼가 있는 경우 출력을 여기에 쓰기 전에 **ERASE**, **ERASEAUP** 및 **FRSET**는 모두 디바이스 버퍼의 내용을 수정합니다. ERASE는 전체 버퍼를 널(X'00')로 설정합니다. 터미널에 대체 화면 크기 기능이 있는 경우에는 ERASE는 또한 버퍼 크기를 설정합니다. 그러므로 태스크에서 첫 번째 SEND MAP은 일반적으로 버퍼를 정리하고 버퍼 크기를 선택하기 위해 ERASE 옵션을 지정합니다. 대체 화면 크기에 대한 자세한 정보는 758 페이지의 『출력 데이터 스트림』에서 3270 쓰기 명령을 참조하십시오.

ERASEAUP(모든 비보호 필드 지우기)은 보호되지 않은(즉, 운영자가 변경할 수 있는 필드) 버퍼의 모든 필드의 내용을 널(null)로 설정합니다. 이는 945 페이지의 『기호 및 물리적 맵핑 병합』에서 DATAONLY 옵션에 설명한 것처럼 데이터 입력에 유용합니다.

FRSET(필드 재설정)은 버퍼에서 모든 필드의 수정된 데이터 태그를 끕니다. 761 페이지의 『3270 필드 속성』에서 이 옵션에 대해 자세히 설명합니다.

- **FREEKB**(무료 키보드)는 출력이 터미널로 전송될 때 키보드를 잠금 해제합니다. 일반적으로 이를 표시장치 터미널에서 수행하고 싶어합니다.
- **ALARM**은 터미널에 소리가 나는 알람이 있는 경우 이를 울립니다.
- **FORMFEED, PRINT, L40, L64, L80** 및 **HONEOM**은 인쇄에만 관련되고 880 페이지의 『CICS 3270 프린터 옵션』에 설명되어 있습니다. **NLEOM**은 또한 주로 인쇄에서 사용되고 동일한 섹션에 설명되어 있습니다. **NLEOM**에는 표준 **BMS**가 필요합니다.

이러한 옵션 중 일부는 맵 자체에 지정될 수도 있습니다. 특히 3270 쓰기 제어 문자로 표현되고 DFHMDI 또는 DFHMSD 매크로의 CTRL 옵션으로 코딩되는 옵션 PRINT, FREEKB, ALARM, FRSET, L40, L64, L80, HONEOM 등입니다.

참고: CTRL 옵션은 항상 그룹으로서 취급되므로 이들 중 일부를 SEND MAP 명령에 포함하는 경우에는 BMS는 맵 정의에서 이들 모두의 값을 무시하고 이들을 명령에만 사용합니다. 앞서 언급한 것처럼 디바이스 제어 옵션을 SEND CONTROL 명령을 사용하여 맵 데이터에서 별개로 보낼 수도 있습니다. SEND MAP에서 사용할 수 있는 모든 옵션을 SEND CONTROL에서 사용할 수 있으나 NLEOM 등과 같이 데이터와 명백하게 관련된 옵션은 예외입니다.

기타 BMS SEND 옵션: WAIT 및 LAST

태스크가 BMS 또는 터미널 제어 SEND 명령을 사용하여 터미널에 쓰면 CICS는 일반적으로 전송을 스케줄한 다음 태스크를 다시 실행할 준비가 되게 합니다. 실제 전송은 나중에 터미널 유형, 액세스 방법 및 시스템에서의 기타 활동에 따라 발생합니다. 프로그램이 제어를 다시 얻기 전에 전송이 완료되었는지 확인하려면 WAIT 옵션을 사용하십시오.

WAIT는 처리와 태스크의 출력 전송 간 겹침을 방지하기 때문에 응답 시간을 약간 늘릴 수 있습니다. (그러나 겹침은 후속 SEND, RECEIVE 또는 태스크 종료까지만 발생합니다. CICS는 다른 터미널 조작을 시작하기 전에 기존의 터미널 조작을 완전히 종료하기 때문입니다.)

LAST 옵션을 사용하여 일부 터미널의 응답 시간을 약간 개선할 수 있습니다. LAST는 전송 중인 출력이 태스크의 마지막 출력임을 표시합니다. 이 지식을 사용하면 CICS가 데이터의 전송을 태스크 종료 시에 발생하는 z/OS Communications Server 대괄

호 끝 플로우와의 데이터 전송을 결합할 수 있습니다.

기호 및 물리적 맵핑 병합

지금까지는 모든 표시장치가 몇몇 상수 데이터(물리적 맵핑이 제공)와 몇몇 변수 데이터(프로그램이 제공하고 기호 맵에 따라 구조화됨)로 구성되었다고 가정했습니다. 그러나 종종 이러한 구성요소 중 하나 이상이 누락되었습니다.

MAPONLY 옵션

예를 들어, 메뉴 맵은 프로그램이 제공하는 데이터를 필요로 하지 않을 수도 있습니다. 이러한 경우에는 SEND MAP 명령의 FROM 옵션 자리에 MAPONLY 옵션을 코딩하십시오. 그런 다음 BMS는 물리적 맵핑의 모든 정보를 사용하여 상수(이름이 없는) 및 이름 지정된 필드 모두의 초기값을 전송합니다. 기호 맵핑 세트를 항상 MAPONLY를 사용하여 맵을 전송하는 프로그램에 복사할 필요가 없습니다. 실제로, 모든 프로그램이 이 방법으로 세트에 있는 모든 맵을 사용하는 경우에는 TYPE=DSECT 맵핑 세트를 건너뛸 수 있습니다.

MAPONLY는 또한 입력 전용 맵을 화면으로 가져오는 방법이기도 합니다.

DATAONLY 옵션

반대의 상황 또한 가능합니다. 프로그램은 모든 데이터를 제공할 수 있고 맵으로부터 상수나 기본값을 필요로 하지 않습니다. 이는 많은 상황에서 맵의 두 번째 및 후속 표시장치 시에 발생합니다. 운영자가 동일한 형식으로 표시되는 일련의 레코드를 통해 찾아보는 데이터 입력 애플리케이션, 조회 애플리케이션 및 입력에서 오류를 발견한 후에 다시 표시되는 화면의 경우입니다.

DATAONLY 옵션을 사용하여 이를 표시하는 경우 BMS는 이 상황을 이용합니다. 여전히 BMS에게 정보 배치를 위해 어떤 맵과 맵핑 세트를 사용 중인지를 알려주어야 하지만 BMS는 기호 맵에 널이 아닌 속성이나 데이터 값이 있는 필드만을 전송합니다. 다른 필드와 속성값은 변경되지 않은 상태로 남아 있습니다.

SEND CONTROL 명령

데이터를 전혀 전송할 필요가 없지만 디바이스 제어를 전송해야 하는 경우도 있습니다. 예를 들어, 화면을 지우거나 알람을 울려야 할 수도 있습니다. SEND CONTROL 명령을 사용하여 이를 수행하여 필요한 옵션을 나열하십시오.

데이터 입력 애플리케이션에서 프로그램을 고려하십시오. 처음 시작될 때 입력 필드, 연관된 레이블, 화면 표제 및 지시사항으로 화면을 포맷하기 위해 데이터 입력 맵을 표시합니다. 이 첫 번째 SEND MAP 명령은 MAPONLY를 지정합니다. 프로그램이 변수 데이터를 전송하지 않기 때문입니다. 그 후에 프로그램은 한 데이터 입력 세트를 승인합니다. 입력이 올바른 경우 프로그램은 이를 정리하고 또 다른 입력을 요청합니다. 여

전히 다른 변수 데이터를 전송할 필요가 없습니다. 해야 할 일은 화면에서 입력을 지우고 키보드를 잠금 해제하여 운영자가 다음 레코드를 입력하도록 신호를 보내는 것입니다.

```
EXEC CICS SEND CONTROL ERASEAUP FREEKB END-EXEC
```

이를 수행합니다. (이 디바이스 제어 옵션 및 기타 디바이스 제어 옵션에 대한 설명은 943 페이지의 『SEND MAP 제어 옵션』의 내용을 참조하십시오.)

오류가 있으면 프로그램은 운영자에게 문제점을 수정하는 방법을 알려주기 위해 변수 데이터를 전송해야 합니다. 이는 오류가 있는 필드의 속성을 변경하여 강조표시하고 이러한 용도로 제공된 필드에 메시지를 전송합니다. 여기에서는 이 맵이 이미 화면에 있기 때문에 프로그램은 DATAONLY 옵션을 사용합니다. (963 페이지의 『입력 오류 처리』에서 입력 오류 처리에 대해 자세히 설명합니다.)

특히 응답 시간이 중요한 경우 데이터 입력 상황에 있는 것처럼 적용할 때 MAPONLY, DATAONLY 및 SEND CONTROL을 사용해야 합니다. MAPONLY는 경로 길이를 저장하고, DATAONLY는 아웃바운드 데이터 스트림의 길이를 줄이고, SEND CONTROL은 둘 모두를 수행합니다.

출력 화면 빌드

물리적 맵핑 정의 옵션, SEND MAP 옵션, 프로그램 데이터 및 병합 옵션의 상호작용은 SEND MAP이 순서대로 된 후에 화면에 나타나는 내용을 판별하기 위한 규칙 요약으로 인해 충분히 복잡적입니다.

화면(버퍼)의 내용은 다음을 기준으로 판별됩니다.

- SEND MAP 명령 전에 있었던 내용
- SEND MAP 명령에서 전송된 필드(필드 속성, 확장 속성 및 표시장치 데이터)
- 이러한 필드 요소의 여러 값이 나오는 위치

이 순서로 가능성을 논의합니다.

시작하는 것

SEND MAP 명령에서 발생하는 첫 번째 일은 ERASE 옵션이 있는 경우 맵의 크기나 기점과 관계없이 전체 화면(버퍼)이 널(null)로 지워진다는 것입니다. 대체 화면 크기 기능이 있는 터미널에서 758 페이지의 『출력 데이터 스트림』에서 3270 쓰기 명령에 설명된 대로 화면 크기 또한 설정됩니다. 화면은 필드가 정의되지 않고 표시장치 데이터가 없는 상태로 포맷되지 않은 상태에 있습니다. ERASEAUP이 존재하면 화면에서 보호되지 않는 모든 필드는 지워지지만 모든 필드의 필드 구조 및 속성과 보호된 필드의 내용은 변경되지 않습니다.

ERASE 및 ERASEAUP은 SEND MAP 데이터가 버퍼로 로드되기 전에 인지됩니다. 이러한 옵션 모두가 SEND MAP에 나타나지 않는 경우에는 화면 버퍼는 운영자가 어

편 내용이든지 수정한 상태로 이전 쓰기 작업 후와 동일한 상태로 시작합니다. 일반적으로 필드의 위치(즉, 속성 바이트) 및 해당 속성은 변경되지 않지만 비보호 필드의 데이터 내용은 다를 수 있습니다. 게다가, 운영자가 CLEAR 키를 사용한 경우에는 전체 버퍼는 널(null)로 지워지고 화면은 ERASE 옵션을 포함했을 때처럼 포맷되지 않은 상태에 있습니다.

전송된 내용

두 번째로, BMS는 맵의 범위 내의 버퍼에서 이러한 위치만을 변경합니다. 이 영역 밖에서는 버퍼의 내용은 변경되지 않지만 맵 외부의 영역은 맵 외부에서 설명된 대로 모양이 변경될 수 있습니다.

맵 영역 내에서 전송된 내용은 DATAONLY 옵션이 있는지에 달려 있습니다. 이 옵션이 없는 일반적인 경우에는 BMS는 맵의 모든 필드의 모든 구성요소(필드 속성, 확장 속성, 표시장치 데이터)를 보냅니다. 이는 POS 피연산자에 지정된 위치에 필드를 작성하고 POS로부터 LENGTH 필드에 지정된 바이트 수를 덮어씁니다. 표시장치 데이터의 끝과 다음 속성 바이트(POS 값) 사이의 버퍼 위치는 변경되지 않은 상태로 남아 있습니다. 다른 맵을 사용한 마지막 쓰기 작업 후에 ERASE하지 않은 경우에는 이러한 사이 공간에 필드(속성 바이트)가 있거나 또는 없을 수도 있습니다.

이러한 필드 요소의 값은 다음 섹션에 설명된 대로 프로그램, 맵 또는 기본값에서 나옵니다.

DATAONLY가 있는 경우, BMS는 프로그램이 제공하는 이러한 필드 및 이들의 구성요소만을 전송합니다. 다른 화면 데이터는 변경되지 않습니다.

값이 나오는 위치

화면 내용을 결정짓는 값은 소스, 프로그램, 맵, 하드웨어의 기본값 등 네 개의 소스 또는 화면의 이전 내용에서 나옵니다.

BMS는 각 맵 필드의 각 구성요소를 개별적으로 고려하고, 제공된 프로그램으로부터 값을 구합니다.

- MAPONLY 옵션은 사용되지 않았습니다.
- 필드에는 맵에 이름이 있으므로 기호 출력 맵에는 데이터를 가져오는 위치인 서브필드의 해당하는 세트를 포함할 수 있습니다. 필드 속성 값은 이름이 A가 접미부로 붙어 있는 맵 필드 이름인 프로그램 서브필드에서 나옵니다. 표시장치 데이터는 0이 접미부로 붙은 동일한 이름의 서브필드에서 나오고 확장 속성 값은 속성을 식별하는 문자가 접미부로 붙은 동일한 이름의 서브필드에서 나옵니다. 941 페이지의 표 50의 내용을 참조하십시오. 확장 속성의 경우에는 기호 맵이 해당하는 서브필드를 포함하기 위해서는 속성 이름은 DSATTS 사이에도 나타나야 합니다.
- 값이 존재합니다. "존재"의 정의는 필드 구성요소에 따라 약간씩 다릅니다.

- 필드 속성 바이트의 경우 값은 널(X'00')이거나 또는 입력 작업(X'80', X'02' 또는 X'82')으로부터 남은 값 중 하나여서는 안됩니다.
- 확장 속성 바이트의 경우 값은 널(null)이어서는 안됩니다.

참고: BMS는 터미널이 지원하는 것으로 정의된 확장 속성 값만을 전송합니다. 다른 확장 속성의 값은 최종 데이터 스트림에서 누락됩니다.

- 표시장치 데이터의 경우 데이터의 첫 번째 문자는 널(null)이어서는 안됩니다.

이러한 조건이 하나라도 충족되지 않으면 다음 단계는 DATAONLY가 있는지 여부에 따라 달라집니다. DATAONLY가 있으면, BMS는 여기서 프로세스를 중지하고 프로그램으로부터 얻은 데이터만을 전송합니다. BMS는 프로그램에 의해 변경되지 않은 구성요소가 화면에서 변경되지 않는 방식으로 이를 수행합니다. 특히, 확장 속성 값은 새 값을 지정하거나 하드웨어 기본값을 요청하지 않는 한 변경되지 않습니다. (X'FF' 값은 사용자가 하드웨어 기본값을 얻기 위해 X'F0'를 지정하는 백그라운드 투명도를 제외하고는 모든 확장 속성에 대해 하드웨어 기본값을 요청합니다.)

DATAONLY가 없고, 이전 조건 중의 하나가 충족되지 않으면 BMS는 다음과 같이 맵에서 데이터를 구합니다.

- 필드 속성의 경우 필드의 ATTRB 옵션에서 값을 구합니다. 값이 없는 경우에는 BMS는 (ASKIP,NORM)의 ATTRB 값을 가정합니다.
- 확장 속성의 경우 BMS는 다음에서 값을 구합니다.
 - DFHMDF 필드 정의에서 해당하는 옵션
 - 여기에 지정되지 않은 경우에는 값은 DFHMDI 맵 정의에서 해당하는 옵션에서 구합니다.
 - 여기에도 없는 경우에는 값은 DFHMSD 맵핑 세트 정의에서 해당하는 옵션에서 구합니다.

(값이 어디에도 지정되지 않은 경우에는 BMS는 이를 전송하지 않으므로 이로 인해 3270이 하드웨어 기본값을 사용하게 됩니다.)

- 표시장치 데이터의 경우 맵의 초기값(INITIAL, XINIT 또는 GINIT 옵션)에서 구합니다. 초기값이 없는 경우에는 필드는 널(null)로 설정됩니다.

맵 외부

맵은 화면 또는 프린터 페이지와 동일한 크기일 필요가 없습니다. 애플리케이션은 화면 영역의 일부만을 사용할 수 있거나 사용자는 출력을 증진적으로 빌드하고 싶을 수도 있고 둘 모두에 해당할 수도 있습니다.

BMS 논리 메시지를 사용하면 여러 맵에서 화면을 빌드할 수 있으므로 이를 단일 터미널 쓰기를 사용하여 전송할 수 있습니다. 968 페이지의 『BMS 논리 메시지』에서 설명하는 ACCUM 옵션을 사용하여 이를 수행할 수 있습니다. ACCUM을 사용하지 않고도 터미널이 버퍼가 있는 3270 같은 디바이스인 경우에는 여러 맵에서 화면을 빌드

할 수 있습니다. 첫 번째 명령 뒤에 지우지 않고 화면(버퍼)의 서로 다른 여러 영역에 쓰기된 여러 SEND MAP 명령을 사용하여 이를 수행할 수 있습니다. 각 SEND MAP 은 출력을 야기하고 표시장치에 일시적인 "플래시"를 생성할 수도 있습니다. 이와 같은 이유와 함께 추가 I/O의 경로 길이를 제거하기 위해서, 이러한 콤포지트 화면에 논리 메시지를 사용하는 것을 선호할 수도 있습니다.

전송된 맵 외부에서 버퍼의 내용은 앞서 언급된 ERASE 및 ERASEAUP의 영향을 제외하고는 변경되지 않습니다. 일반적으로 이는 화면의 해당하는 영역이 변경되지 않음을 의미합니다. 그러나 맵 외부의 화면 위치는 맵 내의 필드에서 해당 속성을 구할 수 있습니다. 맵의 경계를 넘어서 및 문제의 위치 전에 필드를 정의하지 않은 경우(다른 맵 사용) 이것이 발생합니다. 이 위치 외부를 관리하는 맵의 내부에 있는 필드 속성을 변경하는 경우에는 내용은 변경되지 않더라도 위치의 모양이 변경될 수도 있습니다.

GDDM 및 BMS 사용

버퍼 오버레이 기술의 한 가지 사용법은 BMS와 그래픽 데이터 표시장치 관리자(GDDM) 출력의 혼합을 포함하는 화면을 작성하는 것입니다.

일반적으로 BMS 출력을 먼저 쓰고 그 다음에 GDDM이 이어집니다. GDDM 출력을 위해 BMS 맵에 공간을 남겨 놓거나, 구멍을 원하는 위치에 필드가 없는 맵을 작성하여 표시장치에 "그래픽 홀"을 작성할 수 있습니다. 이러한 맵은 "널 맵"이라고 불리고 크기(높이와 너비)는 홀의 크기에 해당합니다.

그래픽을 BMS 출력과 결합하기 위해 GDDM을 사용하는 경우에는 BMS가 사용 중일 수도 있는 프로그램된 기호 세트를 GDDM이 손상시키지 않도록 GDDM PSRSRV 호출을 포함해야 합니다.

커서 위치 지정

커서 위치 지정은 입력에 대한 맵을 사용할 때 중요합니다. 일반적으로 원하는 필드의 ATTRB 값에 IC("insert cursor")를 포함하여 맵 정의에 커서의 초기 위치를 설정합니다.

SEND MAP 명령에서 CURSOR 옵션은 맵을 표시할 때 필요한 경우 이 스펙을 덮어 쓸 수 있습니다. CURSOR(value)를 지정하면 BMS는 화면의 해당 절대 위치에 커서를 배치합니다. 계수는 첫 번째 행과 열(0 위치)에서 시작하고 행에서 계속 진행됩니다. 따라서 80열 표시의 네 번째 열, 세 번째 행에 커서를 놓으려면 CURSOR(163)을 코딩합니다.

값 없이 CURSOR를 지정하면 기호 커서 위치 지정("symbolic cursor positioning")을 수행함을 BMS에 알립니다. 커서를 배치하려는 필드의 길이 서브필드를 (-1)로 설정하여 배치합니다. 길이 서브필드는 출력 전용 맵에서 정의되지 않으므로, 기호 커서 위치 지정을 사용하려면 맵을 INOUT으로 정의해야 합니다. (길이 서브필드는 958 페이지의

지의 『형식화된 화면 입력』에서, INOUT 맵은 953 페이지의 『맵핑된 데이터 수신』에서 설명합니다.) 이 방법으로 둘 이상의 필드를 표시하면 BMS는 찾은 첫 번째 항목을 사용합니다.

기호 커서 위치 지정은 특히 터미널 운영자가 잘못된 데이터를 입력할 때 입력-출력 맵에서 유용합니다. 필드 유효성을 검증하는 경우 오류의 길이를 -1로 설정하면 BMS는 다시 표시할 때 첫 번째 오류에 커서를 배치합니다. 962 페이지의 『맵핑된 입력 처리』에서는 이 방법을 보여줍니다.

SEND CONTROL 명령을 사용하거나 CURSOR의 절대값을 지정하여 커서를 배치할 수 있습니다. SEND CONTROL에서 CURSOR를 생략하면 커서가 이동되지 않습니다.

올바르지 않은 데이터 및 기타 오류 전송

SEND MAP 및 SEND CONTROL 명령에서 발생할 수 있는 대부분의 예외 조건은 고급 BMS 옵션(논리 메시지, 파티션, 특수 디바이스)에만 적용됩니다. 그러나 올바르게 표시되지 않은 데이터를 터미널에 전송할 수도 있습니다.

BMS는 색상과 같은 확장 속성을 해당 속성으로 지원하도록 정의되지 않은 터미널에 전송하려고 하지 않아도, 기호 맵에서 속성 및 데이터 값의 유효성을 검사하지 않습니다.

SEND MAP 및 SEND CONTROL 명령에서 발생할 수 있는 오류 조건에 대한 자세한 정보는 참조 -> 애플리케이션 개발에서 CICS API 명령의 내용을 참조하십시오.

올바르지 않은 데이터의 효과는 특정 터미널 및 올바르게 표시되지 않은 데이터 특성 모두에 따라 달라집니다.

- 올바르게 표시되지 않은 데이터는 제어 순서로 해석될 수 있으므로, 디바이스가 데이터를 승인해도 잘못된 결과가 생성됩니다.
- 화면에 오류 지시자가 나타날 수 있습니다.
- ATNI 이상 종료가 발생할 수 있습니다. 태스크에 ATNI를 알리는 지점은 WAIT 옵션을 지정했는지 여부에 따라 달라집니다(943 페이지의 『SEND MAP 제어 옵션』 참조).

출력 배치 옵션: TERMINAL, SET, PAGING

출력 배치 옵션은 BMS가 형식화된 출력 스트림에서 수행해야 하는 작업을 지정합니다. 현재 설명하는 유일한 배치 옵션은 TERMINAL로, 이는 태스크의 프린시플 기능에 출력을 전송합니다. TERMINAL은 다른 배치를 지정하지 않는 경우 기본값입니다. 그러나 다른 옵션(SET 및 PAGING)도 있습니다.

1. BMS는 터미널에 전송하는 대신, 태스크에 형식화된 출력 스트림을 리턴할 수 있습니다. SET 배치 옵션을 사용하여 이 작업을 요청합니다. 전송을 지연하거나 특별

요구사항을 만족하도록 데이터 스트림을 수정하여 수행할 수도 있습니다. 937 페이지의 『맵의 저장영역 획득 및 정의』에서는 SET를 사용하는 방법과 시점을 설명합니다.

2. 나중에 터미널로 전달하기 위해 CICS 임시 저장영역에서 출력을 저장 및 관리하도록 BMS에 요청할 수 있습니다. 이 PAGING 옵션은 메시지가 둘 이상의 화면 또는 페이지를 포함할 수 있음을 함축하며, 화면 용량을 초과하는 메시지를 표시장치 터미널로 전송하려는 경우에 유용합니다. BMS는 완성되었음을 표시할 때까지 임시 저장영역에 전체 메시지를 저장합니다. 그런 다음, 운영자에게 터미널에서 출력 페이지를 표시하기 위한 기능을 제공합니다. 운영자 제어가 필요하지 않고 때때로 TERMINAL만으로도 충분해도 표시장치 및 프린터에서 PAGING을 사용할 수 있습니다.

PAGING을 사용하는 경우 출력은 설명한 대로 간접적으로 프린시פל 기능으로 이동합니다. 또한 전체 BMS는 다른 터미널 또는 여러 터미널이나 사용자 장치 외에도 또는 이를 대신하는 위치로 메시지를 전송할 수 있는 기능(라우팅)을 제공합니다. 전제조건을 설명한 후에 987 페이지의 『메시지 라우팅』에서 라우팅에 대해 설명합니다.

참고: PAGING 및 SET와 관련 옵션 모두에 전체 BMS가 필요합니다. TERMINAL은 최소 및 표준 BMS에서 사용 가능한 유일한 배치입니다.

SET 사용

BMS 메시지에 대해 SET 배치를 지정하는 경우, BMS는 출력을 형식화하고 디바이스 종속 데이터 스트림 형식으로 이를 리턴합니다. 일반적으로 리턴된 데이터 스트림은 연속적으로 터미널로 보내지지만 터미널 I/O는 발생하지 않습니다.

몇 가지 이유로 BMS는 데이터 스트림을 전송하지 않고 형식화하도록 요청됩니다. 다음 중 하나를 수행할 수 있습니다.

- 특수한 기능이 포함된 디바이스의 요구사항 또는 CICS에서 명시적으로는 지원되지 않는 제한사항을 충족하도록 데이터 스트림을 편집하십시오.
- 표준 3270 기능 또는 특수한 디바이스 특성을 기반으로 데이터 스트림을 압축하십시오.
- CICS에 직접 연결되지 않은 터미널로 데이터 스트림을 전송하십시오. 예를 들어, APPC 링크로 CICS에 연결된 시스템에 접속된 3270 터미널에 데이터를 전달할 수 있습니다. SET로 데이터를 형식화하고 결과 페이지를 링크 상의 파트너 프로그램으로 전송할 수 있습니다. 터미널이 사용자의 프린시פל 기능과 다른 유형인 경우, 적절한 유형의 터미널을 지정하고 SET를 사용하여 해당 터미널로 라우팅하여 996 페이지의 『SET로 라우팅』에서 설명하는 대로 적절한 형식화를 가져올 수 있습니다.

BMS는 SET 옵션에 이름이 지정된 포인터 변수를 페이지 목록 주소로 설정하여 형식화된 출력을 리턴합니다. 이 목록은 다음 형식으로 된 하나 이상의 4바이트 항목으로 구성되며 각각은 출력의 한 페이지에 해당합니다.

표 51. 페이지 목록 항목 형식

바이트	내용
0	터미널 유형(935 페이지의 표 49 참조)
1-3	형식화된 출력 페이지가 포함된 TIOA 주소

터미널 유형에 -1(X'FF')을 포함하는 항목은 페이지 목록 끝을 표시합니다. 이 목록의 주소는 24비트 길이만 가능합니다. 프로그램이 31비트 주소를 사용하는 경우, 24비트의 주소를 주소로 사용하기 전에 2진의 0으로 앞을 채워서 풀워드로 확장해야 합니다.

각 TIOA(터미널 입출력(I/O) 영역)가 이 영역에 대한 표준 형식입니다.

표 52. TIOA 형식

필드 이름	위치	길이	내용
TIOASAA	0	8	정보를 보관하는 CICS 저장영역(8바이트)
TIOATDL	8	2	반자 2진 형식의 필드 TIOADBA 길이
(이름없음)	10	2	예약된 필드
TIOADBA	12	TIOATDL	형식화된 출력 페이지
(이름없음)	TIOATDL + 12	4	페이지 제어 영역, SEND TEXT MAPPED 명령에 필요(사용되는 경우)

일부 BMS 명령은 페이지를 여러 개 작성하기 때문에 BMS가 목록을 사용하여 페이지를 리턴합니다. SEND MAP은 그렇지 않지만 SEND TEXT는 가능합니다. 게다가 라우팅 환경을 작성한 경우라면 BMS는 대상에서 각 터미널 유형에 대해 별도의 논리적 메시지를 빌드하고 한 개의 BMS 명령으로 여러 개의 다른 터미널 유형에 대한 페이지를 가져옵니다. 터미널 유형은 페이지가 속하는 메시지를 알려줍니다. (지정된 유형의 페이지는 항상 순서대로 표시됩니다.) 라우팅하지 않으면 터미널 유형은 항상 프린시플 기능의 유형입니다.

ACCUM 옵션을 사용하지 않으면 페이지를 작성하는 BMS 명령 리턴에서 이를 사용할 수 있습니다. 그렇지만 ACCUM을 사용하면 페이지에서 사용 가능한 공간이 사용 가능해질 때까지 BMS가 대기합니다. BMS는 페이지가 준비되었음을 프로그램에 신호하기 위해 RETPAGE 조건을 켭니다. HANDLE CONDITION 명령을 사용하거나 BMS 명령의 응답(EIBRESP 또는 RESP 옵션으로 리턴된 값)을 테스트하여 RETPAGE를 발견할 수 있습니다.

BMS는 목록을 재사용하기 때문에 BMS에서 페이지 목록을 리턴할 때마다 해당 목록의 정보를 캡처해야 합니다. 페이지 내용이 아니라 해당 정보만 저장해야 합니다. BMS는 페이지 자체를 재사용하는 것이 아니라 페이지의 저장영역을 해당 제어에서 태스크의 저장영역으로 이동합니다. 그렇게 하면 저장영역에 대해 작업할 때 페이지용으로 사용되었던 저장영역을 다시 사용할 수 있게 됩니다. 이 경우, FREEMAIN 명령의 DATA 또는 DATAPOINTER 옵션은 TIOASAA가 아니라 TIOATDL 필드를 지시해야 합니다.

맵핑된 데이터 수신

형식화된 화면은 출력만큼 입력에서도 중요합니다. 데이터 입력 애플리케이션은 명백한 한 가지 예지만, 대부분의 다른 애플리케이션도 부분적으로나마 형식화된 입력을 사용하기도 합니다.

입력에서 BMS는 대략적으로 출력 순서의 역순으로 작업을 수행합니다. 즉, 데이터 스트림에서 디바이스 제어 문자를 제거하고 입력 필드를 데이터 구조로 이동하여 이름으로 이들을 처리할 수 있습니다.

맵은 입력에서만, 출력에서만(이미 다른 사례임) 또는 둘 다에서 사용할 수 있습니다. 입력 전용 맵은 상대적으로 드물지만, 이러한 맵이 나타나는 경우 차이를 확인하고자 입력-출력 맵의 특수 사례로 다룹니다.

관련 개념:

911 페이지의 제 45 장 『BMS(Basic Mapping Support)』

BMS(Basic Mapping Support)는 CICS 프로그램과 터미널 디바이스 사이의 API(Application Programming Interface)입니다.

입출력(I/O) 예제

이 예제는 "빠른 갱신"이라는 단순 데이터 입력 프로그램에 대한 맵 정의를 보여줍니다.

입력 구조 세부사항을 설명하기 전에 913 페이지의 『BMS 출력 예제』의 "빠른 검사" 예제를 재검토하도록 합시다. 계정에 새 요금이 게시되면 야간 실행 간에 반복해서 한도까지 고객에게 요금을 부과하려는 것이 정책에 반한다고 가정하십시오. 일일 누계를 보존하여 "빠른 검사" 처리를 기능 보장하는 새 트랜잭션을 원합니다.

또한 입/출력 둘 다에 동일한 화면을 사용하려고 하므로 고객당 하나의 화면 입력만 있습니다. "빠른 갱신"이라는 새 트랜잭션에서 직원은 계정 번호와 요금을 둘 다 동시에 입력합니다. 정상 응답은 다음과 같습니다.

QUP Quick Account Update
Current charge okay; enter next
Account: _____
Charge: \$ _____

그림 150. Normal “quick update” response

트랜잭션을 거부하면 화면에 대한 입력 정보를 남겨 두므로 직원은 문제점에 대한 설명과 함께 입력된 내용을 볼 수 있습니다.

```

QUP                Quick Account Update
Charge exceeds maximum; do not approve
Account:    482554
Charge:    $ 1000.00

```

그림 151. “빠른 갱신” 오류 응답

(여기서 다시, 용이한 설명을 위해 맵을 짧게 유지하도록 지나치게 단순화했습니다.)

이 연습에 필요한 맵핑 정의는 다음과 같습니다.

```

QUPSET  DFHMSD TYPE=MAP,STORAGE=AUTO,MODE=INOUT,LANG=COBOL,TERM=3270-2
QUPMAP  DFHMDI SIZE=(24,80),LINE=1,COLUMN=1,CTRL=FREEKB
        DFHMDF POS=(1,1),LENGTH=3,ATTRB=(ASKIP,BRT),INITIAL='QUP'
        DFHMDF POS=(1,26),LENGTH=20,ATTRB=(ASKIP,NORM),                X
        INITIAL='Quick Account Update'
MSG      DFHMDF LENGTH=40,POS=(3,1),ATTRB=(ASKIP,NORM)
        DFHMDF POS=(5,1),LENGTH=8,ATTRB=(ASKIP,NORM),                X
        INITIAL='Account:'
ACCTNO   DFHMDF POS=(5,14),LENGTH=6,ATTRB=(UNPROT,NUM,IC)
        DFHMDF POS=(5,21),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
        DFHMDF POS=(6,1),LENGTH=7,ATTRB=(ASKIP,NORM),INITIAL='Charge:'
CHG      DFHMDF POS=(6,13),ATTRB=(UNPROT,NORM),PICIN='$$$$0.00'
        DFHMDF POS=(6,21),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
        DFHMSD TYPE=FINAL

```

그림 152. 입출력(I/O) 맵에 대한 맵핑 정의

이 입출력(I/O)에 대한 맵 필드 정의가 출력 전용 “빠른 검사” 맵에 대한 맵 필드 정의와 유사함을 알 수 있습니다(화면 콘텐츠 변경을 허용할 경우). 참고할 차이점은 다음과 같습니다.

- DFHMSD 맵핑 세트 정의의 MODE 옵션은 INOUT으로, 이 맵핑 세트의 맵이 입력과 출력 둘 다에 사용됨을 표시합니다. INOUT은 BMS가 맵핑 세트의 모든 맵에 대해 입력은 물론 출력에 대해서도 기호 구조를 생성하게 합니다. 이것이 입력 전용 맵인 경우 MODE=IN이고 BMS는 입력 구조만 생성합니다.
- 입력을 원하는 필드(ACCTNO 및 CHG)는 물론 출력을 전송하는 필드(MSG)에도 이름을 지정합니다. 입력 전용 맵에서와 마찬가지로 기호 맵에서 공간 절약을 위해 상수 필드에는 이름을 지정하지 않습니다.
- 입력 필드(ACCTNO 및 CHG)는 운영자가 데이터를 입력할 수 있도록 비보호입니다(UNPROT).
- IC(삽입 커서)가 ACCTNO에 대해 지정됩니다. 맵이 처음 표시될 때 운영자가 입력해야 하는 첫 번째 항목을 준비할 수 있도록 계정 번호 필드의 시작에 커서를 위치시킵니다. (맵을 전송할 때 이 배치를 덮어쓸 수 있습니다. IC는 기본 위치만 제공합니다.)
- ACCTNO 필드 바로 뒤에는 단일 공백으로 구성된 상수 필드가 있으며 CHG 필드 뒤에 비슷한 필드가 있습니다. 이를 “스토퍼” 필드라고 합니다. 일반적으로 일부 다

른 필드가 뒤에 바로 오지는 않는 각 입력 필드 뒤에 배치됩니다. 운영자가 사용자가 제공한 공간 이상으로 사용하지 않는 화면 영역에 데이터를 입력하지 못하게 합니다.

스토퍼 필드를 "자동 건너뛰기"로 정의하면, 운영자가 이전 입력 필드를 채운 후 커서는 다음 비보호 필드로 점프합니다. 운영자가 필드에서 필드로 이동하기 위해 커서를 움직이지 않아도 되므로 대부분의 입력 필드가 고정 길이인 경우 이는 편리합니다.

스토퍼 필드를 "자동 건너뛰기"가 아닌 "보호"로 정의하면, 운영자가 필드 끝을 넘어 입력을 시도할 경우 키보드가 잠깁니다. 이 선택사항은 운영자에게 오버플로우를 즉시 알리므로 대부분의 입력 필드가 가변 길이인 경우(일반적으로 커서 진행 키를 사용해야 함) 선호됩니다. 어느 것을 선택하든지 운영자가 일치된 인터페이스를 보도록 가능한 애플리케이션 전반에서 동일한 선택사항을 사용해야 합니다.

- CHG 필드에는 PICIN 옵션이 있습니다. PICIN은 기호 맵에서 편집 마스크를 생성하며(COBOL 및 PL/I에 유용함) 필드 길이를 나타냅니다. PICIN 사용에 대한 세부사항은 *CICS 애플리케이션 프로그래밍 참조서*를 참조하십시오.

그림 153은 이 INOUT 맵핑 정의에서 비롯된 기호 맵핑 세트를 표시합니다.

```

01 QUPMAPI.
  02 FILLER PIC X(12).
  02 FILLER PICTURE X(2).
  02 MSGL    COMP PIC S9(4).
  02 MSGF    PICTURE X.
  02 FILLER REDEFINES MSGF.
    03 MSGA    PICTURE X.
  02 MSGI    PIC X(40).
  02 ACCTNOL  COMP PIC S9(4).
  02 ACCTNOF  PICTURE X.
  02 FILLER REDEFINES ACCTNOF.
    03 ACCTNOA  PICTURE X.
  02 ACCTNOI  PIC X(6).
  02 CHGL    COMP PIC S9(4).
  02 CHGF    PICTURE X.
  02 FILLER REDEFINES CHGF.
    03 CHGA    PICTURE X.
  02 CHGI    PIC X(7) PICIN '$,$$0.00'.
01 QUPMAPO REDEFINES QUPMAPI.
  02 FILLER PIC X(12).
  02 FILLER PICTURE X(3).
  02 MSGO    PIC X(40).
  02 FILLER PICTURE X(3).
  02 ACCTNO  PICTURE X(6).
  02 FILLER PICTURE X(3).
  02 CHGO    PIC X.
  
```

그림 153. "빠른 갱신"에 대한 기호 맵

QUPMAPO에서 시작하는 이 구조의 두 번째 파트는 화면으로 데이터를 다시 전송하는 데 필요한 구조인 기호 출력 맵입니다. 재정의한 필드를 제외하고 **MODE=INOUT** 대신 **MODE=OUT**을 지정한 경우 예상한 것과 거의 동일합니다. 비교는 914 페이지의 그림 142의 내용을 참조하십시오. 주요 차이점은 필드 속성 (A) 서브필드가 누락된 것처럼 보인다는 것인데, 이에 대해 바로 설명하겠습니다.

기호 입력 맵

레이블 QUPMAPI 아래, 구조의 첫 번째 파트는 새로운 항목입니다. 기호 입력 맵으로, QUPMAP 맵으로 형식화된 화면에서 데이터를 읽는 데 필요한 구조입니다.

맵에서 이름 지정된 각 필드는 세 개의 서브필드를 포함합니다. 기호 출력 맵과 같이 각 서브필드는 맵 필드와 이름이 동일합니다(목적용을 나타내는 문자가 접미부로 추가됨). 입력과 관련된 접미부 및 서브필드는 다음과 같습니다.

- L** 맵 필드에서 입력 길이입니다.
- F** 플래그 바이트로, 운영자가 필드를 지울 것인지 여부와 커서가 해당 위치에 있는지 여부를 표시합니다.
- I** 입력 데이터 자체입니다.

입력 및 출력 구조가 정의되어 필드로 다른 필드를 오버레이합니다. 즉, 지정된 맵 필드의 입력(I) 서브필드는 항상 대응하는 출력(O) 서브필드와 동일한 저장영역을 차지합니다. 마찬가지로 입력 플래그(F) 서브필드는 출력 속성(A) 서브필드를 오버레이합니다. (구현상의 이유로 서브필드 정의 순서는 언어에 따라 약간 달라질 수 있습니다. COBOL에서 A 서브필드의 정의는 INOUT 맵에서 입력 구조로 이동되었지만, 출력 전용 맵에서와 같이 계속 출력에 적용됩니다. 어셈블리의 경우 입력 및 출력 서브필드 정의는 각 맵 필드에서 교차 배치됩니다.)

BMS는 더미 필드를 사용하여 다른 파트에서 나타나지 않는 서브필드의 구조 중 한 파트에 공간을 남깁니다. 예를 들어, 출력 전용 맵이어도 입력 맵의 길이(L) 서브필드에 대응하도록 출력 맵에 항상 2바이트 필드가 있습니다. 확장 속성의 출력 서브필드가 있는 경우(예: 색상 또는 강조표시) BMS는 이와 일치하도록 입력 맵에 더미 필드를 생성합니다. 914 페이지의 그림 142 및 955 페이지의 그림 153 모두에서 이러한 필드의 예제(COBOL에서 FILLER)를 볼 수 있습니다.

입력 및 출력 맵 구조에서 필드의 대응 관계는 데이터 입력 트랜잭션에서 작업할 때나 오류가 있는 입력을 수신해서 운영자로부터의 정정을 요청해야 하는 경우에 입력에 대한 맵을 사용하고 동일한 형식으로 다시 쓰는 프로세스에서 매우 편리합니다.

맵핑된 입력 프로그래밍

맵핑된 입력에 필요한 프로그래밍은 데이터를 반대 방향으로 이동하는 경우를 제외하고 맵핑된 출력의 프로그래밍과 유사합니다. 맵핑된 출력과 같이 맵을 정의하고 먼저 어셈블합니다.

터미널에서 읽는 하나 이상의 프로그램에서 다음을 수행합니다.

1. 기호 맵핑 세트가 대응하는 저장영역을 가져옵니다.
2. 이 저장영역의 구조를 정의하도록 기호 맵핑 세트를 복사합니다.
3. RECEIVE MAP 명령으로 입력 데이터를 형식화합니다.
4. 입력을 처리합니다.

맵핑된 출력에 대해 "빠른 갱신"으로 트랜잭션이 호출되고 대부분의 다른 트랜잭션이 작업을 수행하면 937 페이지의 『BMS 맵핑된 출력 전송』에서 앞서 설명한 단계를 계속 진행합니다. 맵핑된 입력에 대한 몇 가지 고려사항과 단축키는 965 페이지의 『맵핑된 입력 후 맵핑된 출력 전송』에서 설명합니다.

RECEIVE MAP 명령 사용

RECEIVE MAP 명령은 터미널 입력 데이터를 형식화하고 애플리케이션에서 액세스 가능하도록 설정합니다.

명령은 다음을 지정합니다.

- 입력 데이터 스트림 형식화에 사용할 맵(즉, 화면에서의 형식과 프로그램에서 예상하는 데이터 구조(MAP 옵션))
- 이 맵을 찾는 위치(MAPSET 옵션)
- 입력을 가져오는 위치(TERMINAL 또는 FROM 옵션)
- 변환을 대문자로 비활성화할 것인지 여부(ASIS 옵션)
- 형식화된 입력 데이터를 배치하는 위치(INTO 및 SET 옵션)

MAP 및 MAPSET 옵션은 사용할 맵을 BMS에 알리고 SEND MAP 명령에서 수행하는 작업을 그대로 수행합니다.

BMS는 FROM 옵션을 사용하지 않는 한, 태스크(해당 프린시פל 기능)와 연관된 터미널에서 형식화할 입력 데이터를 가져옵니다. FROM은 상대적으로 특이한 상황에서 사용되는 TERMINAL(기본값)의 대안입니다(967 페이지의 『다른 입력 형식화』 참조).

또한 BMS는 일부 경우에 소문자 입력을 자동으로 대문자로 변환합니다. 960 페이지의 『대문자 변환』에서 변환을 제어하는 방법을 설명합니다.

RECEIVE MAP의 INTO 또는 SET 옵션으로 형식화된 입력을 배치하는 위치를 BMS에 지시합니다.

화면의 데이터 외에도 RECEIVE MAP 명령은 운영자가 마지막으로 커서를 사용한 위치와 전송을 수행하는 키를 알려줍니다. 이 정보는 RECEIVE MAP 명령 완료 시 EIB에서 사용 가능합니다. EIBAID는 전송 키("어텐션 ID" 또는 AID)를 식별하고 EIBCURSR은 커서의 마지막 위치를 알려줍니다.

맵핑된 입력에 대한 저장영역 가져오기

RECEIVE MAP 명령을 실행할 때 BMS에는 입력 맵 구조를 빌드할 저장영역이 필요합니다. 프로그램의 작업 중인 저장영역에서 또는 **GETMAIN** 명령을 사용하여 이 공간을 직접 제공할 수 있습니다.

이는 출력 맵을 빌드할 저장영역을 할당하는 경우와 동일한 선택사항이며, 동일한 방법으로 이를 사용합니다(세부사항 및 예제는 937 페이지의 『맵의 저장영역 획득 및 정의』 참조). 어느 경우든 **RECEIVE** 명령의 **INTO** 옵션을 코드화하고 형식화된 입력을 배치할 변수의 이름을 지정하십시오. 예를 들어, "빠른 갱신"의 경우 필요한 명령은 다음과 같습니다.

```
EXEC CICS RECEIVE MAP('QUPMAP') MAPSET('QUPSET')
      INTO(QUPMAPI) END-EXEC.
```

일반적으로 수신 변수는 기호 입력 맵이 정의한 영역이며, 이전에 표시된 대로 BMS는 문자 "I"가 접미부로 붙은 맵 이름을 지정합니다. 그러나 원하면 몇 가지 다른 변수를 지정할 수 있습니다.

입력 작업의 경우, 저장영역 획득을 위한 세 번째 선택사항이 있습니다. **SET** 옵션을 코드화하는 경우, BMS는 **RECEIVE** 명령 시점에 저장영역을 획득하고 **SET** 옵션에 이름 지정된 포인터 변수의 주소를 리턴합니다. 따라서 "빠른 갱신"에서는 다음과 같이 **RECEIVE MAP** 명령을 코드화할 수 있습니다.

```
LINKAGE SECTION.
...
01 QUPMAP COPY QUPMAP.
...
PROCEDURE DIVISION.
...
EXEC CICS RECEIVE MAP('QUPMAP') MAPSET('QUPSET')
      SET(ADDRESS OF QUPMAPI) END-EXEC.
...
```

이런 방법으로 얻은 저장영역은 **FREEMAIN**을 실행하여 해제하지 않는 한 태스크가 종료될 때까지 남아 있습니다(787 페이지의 『저장영역 제어』 참조).

형식화된 화면 입력

CICS는 일반적으로 수정된 읽기 명령을 사용하여 3270 화면을 읽습니다. CICS는 터미널 제어 **RECEIVE** 명령에 **BUFFER** 옵션을 제공하는데, 이 옵션을 사용하여 3270 화면의 전체 콘텐츠를 캡처할 수 있습니다.

전송된 데이터는 운영자가 전송을 야기하기 위해 수행한 내용에 따라 다릅니다.

- **ENTER** 키 또는 기능 키
- **CLEAR**, **CNCL** 또는 **PA** 키("축약형 읽기" 키)
- 필드 선택: 커서 선택, 형광펜 발견 또는 트리거 필드

알아야 할 경우 발생한 이벤트를 알 수 있습니다. 이렇게 하는 방법은 960 페이지의 『어텐션 ID 사용』에 설명되어 있습니다. 또한 769 페이지의 『3270 터미널에서의 입력』에서 3270 입력 작업에 대한 자세한 내용을 찾을 수 있습니다.

축약형 읽기 키는 어텐션 ID(키 자체의 ID)만 전송합니다. 필드 데이터가 수신되지 않으며 맵핑할 내용이 없습니다. 이러한 이유로 축약형 읽기 키는 966 페이지의 『MAPFAIL 및 기타 예외 조건』에 설명된 대로 MAPFAIL 조건을 야기할 수 있습니다. 필드 선택 기능은 필드 데이터를 전송하지만 대부분의 경우 ENTER 및 기능 키와 동일한 데이터가 아닙니다(이에 대해서는 이어지는 단락에서 설명함). 이러한 기능을 사용할 계획인 경우 예외는 1010 페이지의 『특수 하드웨어에 대한 지원』의 내용을 참조하십시오.

대부분의 애플리케이션은 ENTER 키 또는 기능 키를 통해 전송되도록 설계됩니다. 이 중 하나가 전송에 사용되면, 수정된 화면의 모든 필드와 해당 필드만 전송됩니다.

관련 개념:

771 페이지의 『3270 터미널에서 읽기』

3270에는 읽기 버퍼와 수정된 읽기의 두 개의 기본 읽기 명령이 있습니다.

931 페이지의 『비3270 터미널에 대한 지원』

비3270 터미널에 대한 지원에는 표준 BMS가 필요합니다. 최소 BMS는 3270 표시 장치와 프린터만 지원합니다.

수정된 데이터

3270 화면 필드는 필드 속성 바이트의 비트 중 하나인 "수정된 데이터 태그"(MDT)가 켜져 있는 경우에만 수정된 것으로 간주됩니다.

수정은 761 페이지의 『3270 필드 속성』에 설명되어 있습니다. 터미널 하드웨어는 운영자가 필드를 어떤 식으로든 변경하는 경우(데이터 입력, 이미 있는 데이터 변경 또는 지우기) 이 비트를 켭니다. 필드의 ATTRB 값 중의 MDT를 포함하여 맵을 보낼 때 프로그램이 이를 켤 수도 있습니다. 운영자가 이를 변경하지 않더라도 특정 필드의 데이터가 리턴되기를 원하는 경우 이를 수행합니다.

해당하는 길이(L) 서브필드를 보면 특정 맵 필드로부터 입력이 있었는지 여부를 알 수 있습니다. 길이가 0이면 해당 필드에서 아무런 데이터도 읽히지 않은 것입니다. 연관된 입력(I) 서브필드에 모두 널(null)(X'00')이 포함됩니다. BMS는 입력 맵핑 작업을 수행하기 전에 전체 입력 구조를 널(null)로 설정하기 때문입니다. 수정된 데이터 태그가 꺼져 있거나(즉, 필드가 태그가 꺼진 상태로 전송되었고 운영자가 이를 변경하지 않았음) 운영자가 필드를 지운 경우에 길이가 0입니다. 관심이 있는 경우 플래그(F) 서브필드를 조사하여 이러한 두 개의 상황을 구분할 수 있습니다. 필드에 널(null)이 포함되지만 MDT가 켜져 있는 경우(즉, 운영자가 필드를 지워서 변경한 경우) 높은 순서 비트가 있는 것입니다. 플래그 서브필드에 대한 자세한 정보는 962 페이지의 『커서 찾기』의 내용을 참조하십시오.

길이가 0이 아니면 데이터가 필드에서 읽혀진 것입니다. 운영자가 일부를 입력했거나 원래 있던 내용을 변경했거나 필드가 MDT가 켜진 상태로 전송되었습니다. 해당하는 입력(I) 서브필드에서 데이터 자체를 찾을 수 있습니다. 길이 서브필드는 얼마나 많은 문자가 전송되었는지를 알려줍니다. 3270 터미널은 널이 아닌 문자만을 전송하므로, BMS는 얼마나 많은 데이터가 필드에 키 지정되었는지를 압니다. 필드 정의의 JUSTIFY 옵션에 별도로 지정하지 않는 한, 문자 필드는 오른쪽에 공백으로 채워지고, 숫자 필드는 왼쪽에 0으로 채워집니다. ATTRB=NUM이 있는 숫자임을 표시하지 않는 한, BMS는 필드에 문자 데이터가 포함된 것으로 가정합니다.

대문자 변환

CICS는 일부 환경에서는 소문자 입력 문자를 대문자로 자동 변환합니다. 트랜잭션 및 터미널 정의는 트랜잭션이 발생하는지 여부를 함께 판별합니다.

이러한 스펙이 상호작용하는 방법은 PROFILE의 UCTRAN 옵션과 CICS 자원 정의 안내서의 TYPETERM 자원 정의를 참조하십시오.

터미널 입력으로 시작된 태스크의 첫 번째 RECEIVE에서 제외하고 RECEIVE MAP 명령에서 ASIS 옵션을 사용하여 이 변환을 비활성화할 수 있습니다(첫 번째 RECEIVE는 RECEIVE MAP(FROM 없음) 또는 터미널 제어 RECEIVE임). CICS가 이 입력을 이미 읽고 변환했으므로 변환을 비활성화하기엔 너무 늦었습니다(270 페이지의 『태스크 시작 방법』에 설명된 대로 이로 인해 태스크가 호출됨). 따라서 정의상 태스크당 최대 하나의 RECEIVE MAP(FROM 없음)이 발생하는 의사 대화식 트랜잭션 순서에서는 ASIS를 완전히 무시합니다. 같은 이유로 ASIS를 FROM 옵션과 함께 사용할 수 없습니다(967 페이지의 『다른 입력 형식화』 참조).

어텐션 ID 사용

어텐션 ID는 많은 애플리케이션에서 입력의 일부이며 입력을 올바르게 해석하는 데도 필요합니다.

예를 들어, "빠른 갱신" 트랜잭션에서는 직원이 트랜잭션을 종료할 수 있도록 몇 가지 방법이 필요한데 아직 이를 제공하지 않았습니다. PF12를 누르면 트랜잭션 제어를 나가게 되는 규칙을 설정한다고 가정하십시오. 그리고 나서 RECEIVE MAP 명령 뒤에 다음을 코드화합니다.

```
IF EIBAID = DFHPF12,
  EXEC CICS SEND CONTROL FREEKB ERASE END-EXEC
  EXEC CICS RETURN END-EXEC.
```

이는 다음에 실행되어야 하는 트랜잭션을 지정하지 않고 트랜잭션을 종료하므로 운영자가 제어를 되찾습니다. RETURN에 선행하는 SEND CONTROL 명령은 키보드를 잠금 해제하고 화면을 지우므로 운영자는 다음 요청을 입력할 준비가 됩니다.

다양한 어텐션 키에 해당하는 16진 값이 DFHAID라는 카피북에 정의되어 있습니다. 이러한 정의를 사용하려면 사전 정의된 속성 바이트 조합을 사용하기 위해 DFHBMSCA를 복사하는 것과 동일한 방법으로 작업 중인 저장영역에 DFHAID를 복사하십시오(942 페이지의 『속성 값 정의: DFHBMSCA』 참조). DFHAID 카피북의 콘텐츠는 CICS 애플리케이션 프로그래밍 참조서 매뉴얼에 나열되어 있습니다.

HANDLE AID 명령 사용

사용된 어텐션 키를 식별하기 위해 HANDLE AID 명령을 사용할 수 있습니다.

HANDLE AID는 어텐션 ID(AID)가 표시장치에서 수신될 때 제어를 전달할 레이블을 지정하기 위해 사용됩니다.

제한사항: 이 명령은 COBOL, PL/I, 어셈블러 언어 애플리케이션(AMODE(64) 어셈블러 언어 애플리케이션은 제외)으로만 지원됩니다. 지원되는 다른 모든 고급 언어로는 지원되지 않습니다.

HANDLE AID는 다른 HANDLE 명령과 유사하게 작동합니다. 적용되는 첫 번째 RECEIVE 명령 이전에 실행하며, HANDLE AID에 이름 지정된 키가 사용되는 경우 후속 RECEIVE 완료 시 프로그램이 분기되도록 합니다.

다음 예제는 『이스케이프』 코드에 대한 대안을 보여줍니다.

```
EXEC CICS HANDLE AID PF12(ESCAPE) END-EXEC.
...
EXEC CICS RECEIVE MAP('QUPMAP') MAPSET('QUPSET') ...
...
ESCAPE.
EXEC CICS SEND CONTROL FREEKB ERASE END-EXEC
EXEC CICS RETURN END-EXEC.
```

HANDLE AID는 동일한 프로그램의 RECEIVE 명령에만 적용됩니다. 키에 대한 스펙은 키에 대해 새 레이블 이름을 지정하여 동일 프로그램의 다른 HANDLE AID가 이를 대체할 때까지 계속 적용되거나, 레이블 없이 키 이름을 지정하여 종료합니다. RECEIVE 명령의 RESP, RESP2 또는 NOHANDLE 옵션은 HANDLE AID 스펙의 효과에서 해당 명령을 면제하지만, 이 스펙은 다른 경우에 계속 적용됩니다.

입력 운영 동안 수신된 AID에 대해 HANDLE가 활성화된 경우, 제어는 발생하는 예외 상태와 해당 예외에 대해 HANDLE CONDITION이 활성화 상태인지 여부에 관계 없이 HANDLE AID에 지정된 레이블로 이동합니다. 따라서, HANDLE AID는 HANDLE CONDITION으로 이에 대해 확인하는 경우 예외 상태를 마스크할 수 있습니다. AID나 예외 상태, 또는 둘 다에 대해 대체 테스트를 사용하는 것이 바람직할 수 있습니다. AID에 대해 EIBAID를 확인하고, RESP 옵션을 사용하거나 예외에 대해 EIBRESP를 확인할 수 있습니다. 이 상황은 MAPFAIL 상태에 특히 중요합니다. 966 페이지의 『MAPFAIL 및 기타 예외 조건』의 내용을 참조하십시오.

커서 찾기

일부 애플리케이션에서는 운영자가 전송 시 커서를 마지막으로 위치시킨 위치를 알아야 합니다. 발견하는 두 가지 방법이 있습니다.

맵이 CURSLOC=YES를 지정한 경우, BMS는 커서가 마지막으로 위치한 맵 필드의 플래그 서브필드에서 일곱 번째(X'02') 비트를 설정합니다. 이는 물론 이름을 지정한 맵 필드에 커서가 위치한 경우에만 작동합니다.

또한 플래그 서브필드는 커서 존재와 필드 지움 둘 다를 표시하는 데 사용되므로 특히 이를 찾고 있는 경우에는 개별적으로 비트를 테스트해야 합니다(필드 지움의 경우 X'80' 비트, 커서의 경우 X'02' 비트). 비트를 테스트하기 곤란한 언어를 사용 중인 경우 조합을 테스트할 수 있습니다. X'80' 또는 X'82' 값은 지움을 표시합니다. X'02' 또는 X'82'는 커서를 표시합니다. 참조 -> 애플리케이션 개발에서 BMS 매크로 매뉴얼에 설명된 DFHBMSCA 정의는 이 조합을 모두 포함합니다.

EIB의 EIBCPOSN 필드에서도 커서 위치를 판별할 수 있습니다. 이는 화면에서 절대 위치이며 왼쪽 상단(위치 0)부터 계수하여 행을 가로지릅니다. 따라서 너비 40자 화면에서 41 값은 두 번째 행, 두 번째 열에 커서를 둡니다. 프로그램이 화면의 필드 배치와 터미널 유형에 민감해지므로 가능하면 이 방법을 피하십시오.

맵핑된 입력 처리

이 예제는 "빠른 갱신" 프로그램을 사용하여 입력 서브필드 사용 방법을 보여줍니다.

입력 서브필드 사용 방법을 설명하기 위해 "빠른 갱신"으로 돌아갑니다. 입력을 가진 후에는 계속하기 전에 이에 대한 몇 가지 검사를 수행해야 합니다. 먼저, 요금을 입력해야 하는데 양의 숫자여야 합니다(즉, 입력 길이는 0보다 커야 함).

```
IF CHGL = 0, MOVE -1 TO CHGL
MOVE 1 TO ERR-NO
ELSE IF CHGI NOT > ZERO OR CHGI NOT NUMERIC,
MOVE DFHUNIMD TO CHGA,
MOVE -1 TO CHGL
MOVE 2 TO ERR-NO.
```

여기와 다음의 'MOVE -1'문은 949 페이지의 『커서 위치 지정』에 설명된 대로 맵을 다시 표시할 때 오류 상태의 첫 번째 필드에 커서를 둡니다. 메시지 번호는 메시지 영역에 놓일 메시지를 알려줍니다. 1 - 6이며, 1은 "요금 입력"이고 6은 "요금 한도 초과"입니다. 중요도에 따라 대략 오름차순으로 이러한 검사를 수행하여 가장 기본적인 오류가 메시지를 수신하는 오류인지 확인합니다. 검사가 끝나면 ERR-NO가 0일 경우 모든 것이 정상임을 알 수 있습니다.

요금은 물론 계정 번호도 입력해야 합니다. 요금 조건이 어떤든 계정 번호가 있으면 고객의 계정 레코드를 검색하여 계정이 존재하는지 확인할 수 있습니다.


```

IF ACCTNOL = 0, MOVE -1 TO ACCTNOL
MOVE 3 TO ERR-NO
ELSE EXEC CICS READ FILE (ACCT) INTO (ACCTFILE-RECORD)
      RIDFLD (ACCTNOI) UPDATE RESP(READRC) END-EXEC
IF READRC = DFHRESP(NOTFOUND), MOVE 4 TO ERR-NO,
MOVE DFHUNIMD TO ACCTNOA
MOVE -1 TO ACCTNOL
ELSE IF READRC NOT = DFHRESP(NORMAL) GO TO HARD-ERR-RTN.

```

여기까지 도달하면 오류로 인해 계속할 수 없을 때까지 검사를 계속합니다. 운영자가 좋은 계정 번호(문제가 없는 계정 번호)를 주었는지와 계정에 비해 요금이 너무 많지 않은지 확인해야 합니다.

```

IF ERR-NO NOT > 2
IF ACCTFILE-WARNCODE = 'S', MOVE DFHMBRY TO MSGA
MOVE 5 TO ERR-NO
MOVE -1 TO ACCTNOL
EXEC CICS LINK PROGRAM('NTFYCOPS') END-EXEC
ELSE IF CHGI > ACCTFILE-CREDIT-LIM - ACCTFILE-UNPAID-BAL
      - ACCTFILE-CUR-CHGS
MOVE 6 TO ERR-NO
MOVE -1 TO ACCTNOL.
IF ERR-NO NOT = 0 GO TO REJECT-INPUT.

```

입력 오류 처리

처리할 운영자 입력이 있을 때마다 데이터가 올바르지 않을 가능성이 거의 항상 있으며 코드에서 이러한 만일의 사태에 대비해야 합니다.

일반적으로 입력이 틀린 경우 수행해야 할 작업은 다음과 같습니다.

- 운영자에게 오류를 알리십시오. 모든 오류를 한꺼번에 진단하십시오. 한 번에 하나씩 오류를 제공하면 운영자에게 성가신 일입니다.
- 운영자가 정정사항을 제외하고 다시 입력할 필요가 없도록 이미 입력된 데이터를 저장하십시오.
- 운영자가 정정한 후 입력을 재확인하십시오.

오류에 플래그 지정

"빠른 갱신" 트랜잭션에 대한 이전 코드에서는 메시지 필드를 사용하여 오류(첫 번째 오류)를 설명했습니다. 필드에 강조표시할 데이터가 있으면 오류가 발생한 모든 필드를 강조표시하고 BMS가 첫 번째 잘못된 필드에 커서를 위치시키도록 길이 서브필드를 -1로 설정했습니다. 다음과 같이 동일한 맵을 사용하여 이 정보를 전송합니다.

```

REJECT-INPUT.
MOVE LOW-VALUES TO ACCTNOO CHGO.
EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET') FROM(QUPMAPO)
      DATAONLY END-EXEC.

```

DATAONLY 옵션을 지정함에 유의하십시오. 맵의 상수 파트가 여전히 화면에 있고 거기서 재작성 시 포인트가 없기 때문에 이렇게 할 수 있습니다. 수신한 입력을 다시 전

송하지 않기 위해 ACCTNOO 및 CHGO 출력 필드를 지웠으며 다른 속성 조합을 사용하여 ACCTNO 필드를 밝게 했습니다(DFHBMERY 대신 DFHUNIMD). DFHUNIMD는 필드를 강조표시하고 수정된 데이터 태그를 설정한 상태로 두므로 운영자가 필드를 변경하지 않고 다시 전송하는 경우 계정 번호가 재전송됩니다.

좋은 입력 저장

운영자가 올바른 데이터와 올바르지 않은 데이터를 입력하면 정정사항을 제외하고 운영자가 다시 입력할 필요가 없도록 올바른 데이터를 저장해야 합니다. 한 가지 용이한 기술은 화면에 데이터를 저장하는 것입니다. 이를 위해 추가로 아무것도 수행하지 않아도 됩니다. 필드에서 MDT가 설정되면, 화면을 몇 번 읽든지 관계없이 운영자가 필드와 처음으로 접촉할 때 설정 상태를 유지합니다. 화면을 지우고 SEND에서 FRSET 옵션을 사용하여 명시적으로 설정 해제하거나 속성 서브필드를 태그가 해제된 값으로 설정할 때까지 태그는 설정 해제되지 않습니다.

화면에 데이터를 저장할 경우의 단점은 운영자가 CLEAR 키를 사용하면 모든 데이터가 유실된다는 점입니다. 태스크가 대화식인 경우, 오류 정보를 전송하고 정정을 요구하기 전에 입력을 프로그램의 안전한 영역으로 이동하여 이러한 위험을 피할 수 있습니다. 구성요소 태스크가 터미널과의 상호작용에 걸쳐 있지 않은 의사 대화식 순서에서는 오류를 발견하는 태스크가 기존 입력을 정정된 입력을 처리하는 태스크로 전달하는 경우 마찬가지입니다. 임시 저장영역에 쓰거나 많은 다른 방법으로 태스크를 종료하는 RETURN 명령에서 COMMAREA를 통해 데이터를 전달할 수 있습니다(가능성은 314 페이지의 『트랜잭션 사이에 데이터 공유』 참조).

프로그램 또는 임시 저장영역 큐에 데이터를 저장하면 CLEAR 키 문제점을 방지하는 것 이외에 인바운드 전송 시간도 줄어드는데, 오류 정정 주기에서 변경된 필드만 전송하기 때문입니다(이미 전송되고 정정되지 않은 필드가 다시 나타나지 않게 하려면 오류 정보 전송 시 FRSET을 지정해야 함). 최초 이후 사용자가 필드 또는 관련 필드를 변경한 경우에만 감사해야 하기 때문에 필드 감사 반복도 방지할 수 있습니다.

그러나 이러한 이점으로 인해 추가 프로그래밍과 복잡도가 발생하므로 이러한 선택을 정당화하려면 회선 시간 또는 감사 경로 길이 절약이 상당해야 하며 오류 가능성이 높습니다. 코드를 추가하여 새 입력을 기존 입력과 병합해야 하며, MDT를 설정 해제한 경우에는 길이 및 플래그 서브필드를 둘 다 확인하여 운영자가 맵 필드를 수정했는지 여부를 판별해야 합니다. 새 데이터가 있는 필드의 길이는 0이 아닙니다. 데이터가 있었다가 나중에 지워진 필드는 플래그 서브필드에서 최상위 비트를 갖습니다.

좋은 절충안은 데이터를 양쪽으로 저장하는 것입니다. 운영자가 화면을 지우면 저장된 데이터를 사용하여 화면을 새로 고치십시오. 그렇지 않으면 화면에서 수신되는 데이터를 사용하십시오. 병합 논리는 필요하지 않지만 운영자가 의도하지 않은 CLEAR 동안 시간을 낭비하지 않게 하십시오.

감사와 전송을 최소화한 "빠른 갱신" 코드의 경우, "아무것도 안함" 접근방식을 선택하고 화면에 대한 정보를 저장합니다.

재확인

마지막 요구사항은 입력 데이터를 재확인하는 것입니다. 태스크가 대화식인 경우 이는 정정된 입력을 수신(필요하면 병합)한 후 코드의 감사 섹션 반복을 의미합니다. 의사 대화식 순서에서는 일반적으로 실패한 트랜잭션을 반복합니다. 예제에서는 정정된 데이터를 새 데이터와 구별하기 어려운 방법으로 데이터를 저장했기 때문에 정정된 데이터에 대해 동일한 트랜잭션을 실행하도록 배열하기만 하면 됩니다. 따라서 다음과 같습니다.

```
EXEC CICS RETURN TRANSID('QUPD') END-EXEC.
```

여기서 'QUPD'는 "빠른 갱신" 트랜잭션의 ID입니다.

맵핑된 입력 후 맵핑된 출력 전송

트랜잭션이 초기 입력을 처리한 후 다음 단계는 트랜잭션 출력을 준비하고 전송하는 것입니다.

일반적으로 출력이 맵핑될 경우에는 937 페이지의 『BMS 맵핑된 출력 전송』에 설명된 단계를 따릅니다. 그러나 맵 빌드를 위한 저장영역 획득은 이미 수행한 입력 맵핑의 영향을 받습니다. 출력 맵과 입력 맵이 다르지만 동일한 맵핑 세트 또는 서로 오버레이되도록 정의된 맵핑 세트에 있으면 입력 맵핑 프로세스 동안 저장영역 획득을 이미 완료했습니다. 출력 맵과 입력 맵이 서로 오버레이되는 경우에는 여전히 필요한 맵 입력을 저장하고 출력 맵 빌드를 시작하기 전에 출력 구조를 널(null)로 지워야 합니다. 이것이 곤란하면 서로 오버레이되지 않도록 맵을 정의할 수 있습니다(이와 관련한 선택사항은 938 페이지의 『BASE 및 STORAGE 옵션』 참조).

또한 트랜잭션은 동일한 맵을 출력에 입력으로 사용하도록 요구할 수 있습니다. 이미 살펴보았듯이 이는 입력 오류를 처리하는 코드에서 일상적이며 "빠른 갱신"과 같은 단순 트랜잭션에서도 마찬가지입니다. 1화면 데이터 입력 트랜잭션은 또 다른 일반적인 예입니다.

화면에 맵이 이미 있는 새 데이터를 전송하는 경우, DATAONLY 옵션을 사용하여 전송을 줄일 수 있으며 SEND CONTROL 명령만 필요합니다. 이러한 옵션에 대한 설명은 945 페이지의 『기호 및 물리적 맵핑 병합』의 내용을 참조하십시오.

그러나 "빠른 갱신" 트랜잭션의 경우, "이동" 응답으로 메시지 필드를 채워야 합니다(요금으로 파일을 갱신하여 처리를 완료해야 합니다).

```
MOVE 'CURRENT CHARGE OKAY; ENTER NEXT' TO MSGO
ADD CHGI TO ACCTFILE-CUR-CHGS
EXEC CICS REWRITE FILE('ACCT') FROM (ACCTFILE-RECORD)....
```

다음 입력을 위해 화면이 준비되도록 입력 필드를 지워야 합니다. 화면(ERASEAUP 옵션은 비보호 필드를 모두 지움)과 출력 구조(출력 서브필드는 입력 서브필드를 오버레이하고 입력 데이터는 여전히 거기에 있기 때문) 둘 다에서 이렇게 해야 합니다.

```
MOVE LOW-VALUES TO ACCTNOO CHGO.
EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET') FROM(QUPMAPO)
DATAONLY ERASEAUP END-EXEC.
```

마지막으로, CICS에 제어를 리턴하고 다음 입력에 대해 동일한 트랜잭션이 실행되도록 지정할 수 있습니다.

```
EXEC CICS RETURN TRANSID('QUPD') END-EXEC.
```

MAPFAIL 및 기타 예외 조건

터미널에서 전송되는 사용 가능한 데이터가 없거나 전송된 데이터가 형식화되지 않은 경우 MAPFAIL 예외 조건이 발생합니다.

운영자가 CLEAR 키 또는 PA 키 중 하나를 사용한 경우 RECEIVE MAP에서 MAPFAIL 조건이 발생합니다. 다음 조건이 둘 다 적용될 때 운영자가 화면으로부터 ENTER 또는 PF 키를 사용하는 경우에도 발생합니다.

- 맵에 정의된 필드에 수정된 데이터 태그가 설정되어 있지 않습니다(이는 운영자가 아무것도 입력하지 않았고 사용자는 태그가 이미 설정된 필드를 전송하지 않았으므로 읽기 시 데이터가 리턴되지 않음을 의미함).
- 커서가 맵에 정의된 이름 지정된 필드에 없거나 맵이 CURSLOC=YES를 지정하지 않았습니다.

운영자가 ENTER를 너무 빨리 쉽게 누르거나 『축약형 읽기』 키를 실수로 누를 수 있습니다. 사용자 친화적이 되도록 오류가 발생한 트랜잭션을 종료하지 않고 MAPFAIL 조건 이후에 화면을 새로 고칠 수 있습니다.

MAPFAIL 조건은 현재 또는 이전 태스크에서 SEND MAP 또는 동등한 명령을 사용하여 처음 형식화하지 않고 RECEIVE MAP을 실행하는 경우에도 발생하며 전송한 맵과 다른 맵을 사용하는 경우 발생할 수 있습니다. 이런 상황은 논리 오류를 표시하거나 트랜잭션이 시동 단계임을 의미할 수 있습니다. 예를 들어, 빠른 갱신 예제는 시작하기(즉, 운영자가 트랜잭션 사용을 시작할 수 있도록 화면에 비어 있는 맵 가져오기)를 허용하지 않습니다. 별도의 트랜잭션을 사용하여 이를 수행할 수 있지만 MAPFAIL 이후에 화면을 새로 고쳐야 하는 코드를 이용할 수 있습니다. 필수 코드는 다음과 같습니다.

```
IF RCV-RC = DFHRESP(MAPFAIL)
  MOVE 'PRESS PF12 TO QUIT THIS TRANSACTION' TO MSGO
  EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET')
  FROM(QUPMAPO) END-EXEC.
```

이 코드는 운영자에게 이스케이프 방법을 상기시키는데, 이를 수행하기 위한 시도가 MAPFAIL의 원래 원인일 수 있기 때문입니다. 이 메시지를 전송하지 않거나 맵에서 기본값이었다면 다음과 같이 MAPONLY 옵션을 사용할 수 있습니다.

```
EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET') MAPONLY END-EXEC.
```

MAPFAIL이 발생하면 그렇지 않은 경우처럼 입력 맵 구조가 널(null)로 지워지지 않으므로 프로그램 논리가 이 지우기에 따라 달라지는 경우에는 이 조건을 테스트해야 합니다.

다른 예외 조건의 경우와 마찬가지로 HANDLE CONDITION 명령을 실행하여 MAPFAIL을 인터셉트할 수 있습니다. 그러나 HANDLE CONDITION 명령을 실행하고 수신하는 AID에 대해 HANDLE AID도 활성화되어 있으면 MAPFAIL에 대해 지정된 레이블이 아니라 AID에 대해 지정된 레이블로 제어가 이동합니다. 추가 설명은 961 페이지의 『HANDLE AID 명령 사용』의 내용을 참조하십시오. 이 상황에서는 EIBRESP도 테스트하지 않는 한 HANDLE을 실행했다라도 MAPFAIL을 인식하지 못합니다.

EOC 조건

EOC는 BMS 사용 시 자주 발생하는 다른 조건입니다. 체인 끝(EOC) 지시자가 z/OS Communications Server에서 리턴된 요청/응답 단위에 설정된 경우 발생합니다. EOC는 오류를 표시하지 않으며 BMS 기본 조치는 이 조건을 무시하는 것입니다.

다른 입력 형식화

RECEIVE MAP 명령을 사용하여 형식화하는 데이터가 일반적으로 터미널에서 오긴 하지만 터미널에서 오지 않았거나 간접적으로 온 데이터도 형식화할 수 있습니다.

예를 들어, 입력을 수신하고 일부 파트를 조사할 때까지 사용할 맵을 모를 수 있습니다. 이는 파티셔닝이나 논리 디바이스 코드와 같은 특수 하드웨어 기능을 사용하는 경우와 일정한 논리 상황에서도 발생할 수 있습니다. 또한 중간 프로세스(맵핑 없음)를 통해 형식화된 화면에서 읽고 나중에 트랜잭션에 전달된 데이터를 형식화해야 할 수 있습니다.

RECEIVE MAP 명령의 FROM 옵션이 이러한 상황을 해결합니다. FROM은 데이터를 이미 읽었고 고유 입력 스트림에서 입력 맵 구조로의 변환만 필요함을 BMS에 알립니다.

입력을 이미 읽었기 때문에 FROM을 사용하는 경우 길이를 지정해야 합니다. BMS는 일반적으로 그렇듯이 액세스 방법에서 이 정보를 가져올 수 없기 때문입니다. 데이터가 원래 다른 태스크의 RECEIVE 명령에서 비롯된 경우, RECEIVE MAP FROM 명령의 길이는 원래 해당 RECEIVE가 생성한 길이여야 합니다.

동일한 이유로, FROM을 사용하는 경우에는 ASIS 옵션을 사용하여 대문자로의 변환을 비활성화할 수 없습니다. 또한 BMS는 RECEIVE FROM 명령 뒤에 EIBAUD 및 EIBCUSR을 설정하지 않습니다.

그리고 마지막으로, BMS는 입력이 비릇된 디바이스를 모르고 현재 프린시פל 기능이 있다고 가정합니다(입출력(I/O)이 발생하지 않더라도 프린시פל 기능 없이 RECEIVE FROM을 사용할 수 없음). 데이터가 다른 유형의 디바이스에서 비릇된 경우, 입력 데이터 스트림을 올바른 변환하려면 비슷한 프린시פל 기능을 가진 트랜잭션에서 맵핑을 수행해야 합니다.

참고: 3270 의미에서 입력 데이터가 형식화되지 않기 때문에 BUFFER 옵션이 있는 터미널 제어 RECEIVE로 데이터 읽기를 맵핑할 수 없습니다. 그러한 입력을 RECEIVE MAP FROM하려고 시도하면 MAPFAIL이 발생합니다.

BMS 논리 메시지

배치 옵션은 SEND MAP 명령 및 출력 페이지 사이의 대응 관계에 영향을 주지 않습니다.

전체 BMS의 두 번째 기능, ACCUM 옵션도 사용하지 않는 한, 각 SEND MAP 명령에 대해 하나의 페이지를 가져옵니다. ACCUM을 사용하면 둘 이상의 맵을 사용하여 페이지를 단편적으로 빌드할 수 있으며 PAGING과 같이 메시지에서 페이지를 초과할 수 있습니다. 페이지 구분이나 출력을 특정 페이지 또는 화면 용량에 맞게 조정하는 작업에 대해서는 걱정하지 않아도 됩니다. BMS에서는 사용자가 원하는 경우 페이지 구분에 대한 제어를 사용자에게 제공하여 자동으로 이를 처리합니다. 누적 페이지 빌드에 대한 세부사항은 973 페이지의 『누적 출력 — ACCUM 옵션』에서 제공합니다.

둘 이상의 페이지로 구성된 출력 메시지 또는 여러 다른 맵으로 구성된 단일 페이지를 작성하는 즉시 BMS에서 누적 맵핑이라고 하는 작업을 수행합니다. PAGING은 다중 페이지를 함축하고 ACCUM은 다중 페이지 및 컴포지트 페이지 모두를 함축합니다. 따라서 이러한 옵션이 처음 나타날 때 BMS는 누적 맵핑 모드로 이동하고 논리 메시지를 시작합니다. SEND 명령과 메시지 사이의 일대일 대응 관계가 종료되고 후속 SEND MAPS는 현재 논리 메시지에 추가됩니다. 메시지 내 개별 페이지는 완료되면 즉시 처분되지만, 모두 동일한 논리 메시지에 속해 있으며, BMS에 종료하라고 지시할 때까지 계속됩니다.

논리 메시지 빌드

논리 메시지를 구성할 때 여러 규칙을 준수해야 합니다.

- 한 번에 하나의 논리 메시지만 빌드할 수 있습니다. 이 메시지 경로를 지정하는 경우 BMS는 내부적으로 둘 이상의 논리 메시지를 작성할 수 있지만, 내용 측면에서

는 하나뿐입니다. 메시지를 완성하고 처분한 후에 원하는 경우 다른 옵션을 사용하여 동일한 태스크에서 다른 메시지를 빌드할 수 있습니다.

- 메시지 관리 관련 옵션은 메시지를 빌드하는 모든 명령에서 동일해야 합니다. 다음과 같습니다.
 - 배치 옵션: PAGING, TERMINAL 또는 SET
 - 페이지 구성을 제어하는 옵션: ACCUM은 모든 명령에 존재하거나 모든 명령에 없어야 함
 - CICS 임시 저장영역에서 메시지 ID: REQID 옵션 값.

메시지 중간에 옵션을 전환하면 INVREQ 조건 또는 REQID의 경우 IGREQID 조건이 나타납니다.

- ERASE, ERASEAUP, NLEOM, FORMFEED 옵션은 페이지에 기여하는 BMS 명령 모두에서 사용될 때 적용됩니다.
- 페이지에 대한 CURSOR, ACTPARTN, MSR 옵션 값은 지정된 경우 최근 SEND MAP 명령에서, 지정되지 않은 경우 맵에서 가져옵니다.
- 최근 SEND MAP 명령의 3270 쓰기 제어 문자(WCC)가 사용됩니다. WCC는 임의의 WCC가 지정될 때마다 ALARM, FREEKB, PRINT, FRSET, L40, L64, L80, HONEOM 옵션에서 어셈블됩니다. 그렇지 않으면, 맵의 동일한 옵션에서 빌드됩니다. 명령의 옵션은 맵의 옵션과 혼합되지 않습니다.
- 메시지를 빌드하는 데 사용된 모든 명령의 FMHPARM이 포함됩니다.
- SEND MAP 및 SEND CONTROL 명령 모두를 사용하여 이전에 언급한 옵션이 일관되는 한, 논리 메시지를 빌드할 수 있습니다. 또한 SEND TEXT 및 SEND CONTROL 명령 조합으로 논리 메시지를 빌드할 수도 있습니다. (SEND TEXT는 텍스트 출력을 형식화할 때 SEND MAP의 대안입니다(981 페이지의 『SEND TEXT 명령』 참조).) 그러나 파티션 또는 논리 디바이스 코드를 사용하지 않는 한, 동일한 메시지에서 SEND MAP 및 SEND TEXT를 혼합할 수 없습니다(각각 1001 페이지의 『파티션 지원』 및 1010 페이지의 『논리 디바이스 구성요소』 참조).

또한 SEND TEXT의 특수한 두 개 양식이 있습니다. 여기에서는 결합된 맵핑 및 텍스트 출력이 허용되지만 다른 제한사항이 적용됩니다. 자세한 정보는 986 페이지의 『SEND TEXT MAPPED 및 SEND TEXT NOEDIT』의 내용을 참조하십시오.

- 논리 메시지를 빌드하는 동안, 터미널과 계속 통신할 수 있습니다. 라우팅도 수행하지 않는 한, 터미널에 쓰기 위해 BMS 명령을 사용할 수 없습니다. 그러나 BMS RECEIVE MAP 명령 및 터미널 제어 SEND와 RECEIVE 명령을 사용할 수 있습니다.

SEND PAGE 명령

논리 메시지를 완료한 경우, SEND PAGE 명령으로 BMS에 알립니다.

ACCUM 옵션을 사용한 경우, SEND PAGE는 현재 페이지를 완료하고 이전 페이지에 대해 이미 수행한 것처럼 사용자가 설정한 배치 옵션에 따라 BMS가 이 페이지를 처리하도록 합니다. 사용자의 배치가 TERMINAL이면, 이 마지막 페이지는 사용자의 프린시플 기능에 기록됩니다. SET인 경우에는 프로그램으로 리턴되고 PAGING이면 임시 저장영역에 기록됩니다. 사용자의 배치가 PAGING인 경우, BMS는 또한 전체 메시지의 전달을 프린시플 기능에 배열합니다. 『RETAIN 및 RELEASE』에 설명된 것처럼, SEND PAGE 명령의 옵션에 따라 수행 방법이 좌우됩니다.

SYNCPOINT 명령 또는 태스크의 끝은 명시적이기 보다 암시적으로 논리 메시지를 종료합니다. 기능하면, BMS는 사용자가 SYNCPOINT 또는 RETURN 전에 SEND PAGE를 실행한 것처럼 작동하지만, ACCUM 옵션을 사용한 경우 출력의 마지막 페이지는 손실됩니다. 결국, 명시적으로 SEND PAGE를 코딩해야 합니다.

또한 어떤 이유로 전송하지 않을 것을 결정한 경우 불완전한 논리 메시지를 삭제할 수도 있습니다. SEND PAGE 대신 PURGE MESSAGE 명령을 사용합니다. PURGE MESSAGE는 BMS가 CICS 임시 저장영역에 이미 쓴 페이지를 비롯하여 현재 논리 메시지 및 연관된 제어 블록을 삭제하도록 합니다. 원하면, 동일한 태스크에서 나중에 다른 논리 메시지를 작성할 수 있습니다.

RETAIN 및 RELEASE

SEND PAGE 명령의 옵션은 BMS에 터미널로 페이지를 전달하는 시점과 방법을 알립니다.

PAGING 배치로 논리 메시지를 완성하면 BMS는 전체 논리 메시지를 전달할 준비를 합니다. 그러면 임시 저장영역에 메시지가 누적됩니다. 페이지 표시 또는 인쇄는 SEND PAGE 명령 직후에 인라인으로 수행할 수 있지만 이 목적에서는 별도의 태스크를 스케줄하는 것이 더 일반적입니다. 두 경우 모두 CICS는 필요한 프로그램을 제공합니다. 이 프로그램에서는 앞뒤로 페이지를 이동하거나 특정 페이지를 표시하는 등, 터미널 운영자가 메시지 표시를 제어할 수 있습니다. 별도의 태스크를 사용하는 경우 트랜잭션 코드 CSPG에서 의사 대화식으로 실행됩니다. 표시가 인라인인 경우 메시지를 작성한 태스크 내에서 작업이 수행되고(동일한 CICS 제공 프로그램), 결과적으로 통신이 가능해 집니다.

SEND PAGE 명령에서 RETAIN 또는 RELEASE를 지정하거나 둘 다 지정하지 않고서 메시지를 전송하는 방법과 시점을 표시합니다. 가장 일반적인 방법이자 기본값은 둘 다 지정하지 않는 것입니다. 그러면 CICS는 메시지를 표시하고 제어를 태스크로 되돌리도록 CICS 제공 트랜잭션 CSPG에서 스케줄을 지정합니다. CSPG 트랜잭션은 터미널에서 실행 대기 중인 다른 항목과 함께 큐에 배치되며, 이는 터미널이 사용 가능해지면 우선순위로 실행됩니다. 일반적으로 대기 중인 다른 태스크가 없으면 CSPG는 태스크 작성이 끝나면 즉시 실행됩니다.

참고: 터미널은 CICS에서 CSPG를 자동으로 실행하기 위해 자동 트랜잭션 초기화를 허용하는 방식으로 정의되어야 합니다(연관된 TYPETERM 정의에서 ATI(YES)). 그렇지 않으면 운영자는 트랜잭션 코드 CSPG를 입력하거나 페이지 명령 중 하나를 입력하여 RELEASE 및 RETAIN 모두 지정하지 않을 때 프로세스를 시작해야 합니다.

RELEASE 옵션은 비슷하게 작동하지만, SEND PAGE RELEASE 이후에 태스크는 제어를 다시 얻지 못합니다. 대신, BMS는 메시지의 첫 번째 페이지를 터미널에 즉시 전송합니다. 그러면 CICS RETURN을 최상위 레벨 프로그램에서 실행한 경우와 같이 태스크를 종료하고 터미널에서 CSPG 트랜잭션을 시작하므로 운영자가 나머지 페이지를 표시할 수 있습니다. CSPG 코드는 RELEASE 및 RETAIN 모두 지정하지 않은 경우와 같이 의사 대화식으로 실행되고 원래 태스크는 이전인 경우 의사 대화식으로 남아 있습니다.

RELEASE와 두 옵션을 사용하지 않는 방법 사이에는 두 가지 기타 차이점이 있습니다.

- RELEASE를 통해 운영자는 CSPG를 사용하여 메시지를 표시한 후에 터미널에서 다음 입력에 대한 트랜잭션 ID를 지정할 수 있습니다.
- RELEASE에서는 터미널 운영자가 다중 트랜잭션에서 출력 체인을 생성할 수도 있습니다(972 페이지의 『터미널 운영자 페이지: CSPG 트랜잭션』 참조).

SEND PAGE RETAIN을 사용하면 BMS가 메시지를 즉시 전송할 수 있습니다. 이 프로세스를 완료하면 SEND PAGE 명령 직후 태스크에 제어를 재개합니다. 터미널이 표시장치인 경우 BMS는 메시지를 통해 CSPG 트랜잭션이 수행하는 것과 동일한 페이지의 운영자 기능을 제공합니다(단, 태스크의 프레임워크 내). 이 목적으로 BMS가 사용하는 코드는 RECEIVE 명령을 실행하여 운영자의 표시장치 요청을 가져오고, 이로 인해 태스크가 통신 가능해집니다.

참고: SEND PAGE 명령 처리 중 오류가 발생하면 논리 메시지는 완전하다고 간주되지 않으며, 표시되지 않습니다. BMS는 오류 이후 제어를 다시 얻는 준비를 하지 않는 한, 정리 처리를 통해 메시지를 버립니다. 이 경우 PURGE 명령으로 논리 메시지를 삭제하거나 SEND PAGE를 재시도할 수 있습니다. 실패의 원인이 되는 오류를 수정하지 않는 한, 재시도하지 마십시오.

AUTOPAGE 옵션

SEND PAGE 명령의 옵션은 BMS에 터미널로 페이지를 전달하는 방법을 알립니다.

표시장치 터미널의 경우 CSPG를 사용하여 터미널 운영자가 요청할 때 한 번에 하나의 페이지를 전송하려고 합니다. 프린터의 경우 차례대로 페이지를 전송하려고 합니다. SEND PAGE 명령에서 AUTOPAGE 또는 NOAUTOPAGE 옵션을 통해 이를 제어합니다. NOAUTOPAGE를 사용하면 터미널 운영자가 페이지의 표시를 제어할 수 있습니다. AUTOPAGE는 디바이스에서 메시지를 승인하는 즉시 오름차순으로 페이지를 전송합니다. 둘 다 지정하지 않으면 BMS는 터미널 정의에 적합한 항목을 판별합니다.

참고: 프린시פל 기능이 프린터인 경우 때때로 PAGING 대신 TERMINAL의 배치를 사용할 수 있습니다. 프린터로의 후속 전송이 표시장치에서와 같이 차례로 오버레이하지 않기 때문입니다. TERMINAL은 특히, ACCUM이 필요하지 않은 경우 오버헤드가 적으므로, 논리 메시지 작성을 방지할 수 있습니다.

터미널 운영자 페이징: CSPG 트랜잭션

CICS에서 제공하는 페이징 트랜잭션, CSPG를 사용하면 터미널의 사용자가 페이지 검색 요청을 입력하여 논리 메시지의 개별 페이지를 표시할 수 있습니다. 시스템 초기화 테이블에서 CSPG가 지원하는 검색 및 기타 요청에 대한 트랜잭션 ID를 정의합니다. 때때로 프로그램 기능 키를 사용하여 운영자의 노력을 최소화할 수 있습니다.

검색은 순차적으로 진행되거나(다음 페이지 또는 이전 페이지) 무작위로 진행될 수 있습니다(특정 페이지, 첫 번째 페이지, 마지막 페이지). 페이지 검색 외에도 CSPG는 다음 요청을 지원합니다.

페이지 복사

현재 표시장치의 페이지를 다른 터미널로 복사합니다. BMS는 터미널이 BMS에서 지원되는 유형인 경우 대상 터미널에서 페이지 크기가 서로 다른거나 형식화 특성이 서로 다르면 페이지를 다시 형식화합니다.

메시지 조회

CSPG로 터미널에 표시하기 위해 대기 중인 메시지를 나열합니다. 목록은 BMS에서 지정한 메시지 ID 및 라우팅된 메시지인 경우 메시지 제목(발신인이 제공한 경우)을 포함합니다.

메시지 제거

논리 메시지를 삭제합니다.

페이지 체인

메시지를 표시하기 위해 시작한 후에 현재 CSPG 트랜잭션을 일시중단하고 하나 이상의 기타 트랜잭션을 실행한 다음, 원래 CSPG 표시를 재개합니다. 중간에 개입된 트랜잭션은 자체적으로 BMS 또는 터미널 출력을 생성할 수 있습니다. 이 출력이 PAGING 및 RELEASE 또는 RETAIN 옵션으로 작성된 BMS 논리 메시지인 경우 이 메시지는 원래 메시지와 "체인"으로 연결되었으며 운영자는 이 메시지와 다른 메시지 사이를 전환할 수 있습니다.

자동 페이지로 전환

NOAUTOPAGE 표시 모드에서 AUTOPAGE 모드로 전환합니다. 키보드 및 하드카피 출력을 결합한 터미널의 경우 이를 통해 운영자가 특정 페이지 검사에 기반하여 메시지를 제거하거나 인쇄할 수 있습니다.

페이지를 검사하는 프로세스는 메시지를 제거할 수 있음을 운영자가 알릴 때까지 계속됩니다. CSPG에서는 이전에 언급한 대로 이 용도로 특정 요청을 제공합니다. SEND PAGE 명령이 OPERPURGE 옵션을 포함한 경우 이 요청은 메시지를 삭제하고 CSPG에서 제어를 되돌리는 유일한 방법입니다.

그러나 OPERPURGE가 없으면 CSPG 요청이 아닌 터미널의 입력이 메시지를 삭제하고 CSPG를 종료하라는 요청으로 해석됩니다. RETAIN 옵션으로 메시지가 표시되면 표시를 종료하는 비CSPG 입력을 통해 태스크가 실행을 재개할 때 BMS 또는 터미널 제어 RECEIVE에 액세스할 수 있습니다. CSPG 트랜잭션에 대한 자세한 정보는 다음을 참조하십시오. CSPG - 페이지 검색(CICS 제공된 트랜잭션)

논리 메시지 복구

PAGING의 배치로 작성된 논리 메시지는 작성 및 전달 사이에서 CICS 임시 저장영역에 보존됩니다.

BMS는 SEND 명령의 2자 REQID와, 그 뒤에 있는 고유성 유지를 위한 6자리 숫자에서 메시지에 대한 임시 저장영역 큐를 구성합니다. REQID를 지정하지 않는 경우, BMS는 두 개 별표(**) 값을 사용합니다.

임시 저장영역은 회복 가능한 자원이 될 수 있으므로, PAGING 배치가 있는 논리 메시지는 CICS 이상 종료 후에 복구될 수 있습니다. 실제로, CICS는 임시 저장영역의 복구성은 일반 큐 이름을 근거로 하므로, 메시지에 대한 REQID 선택사항으로 메시지 중 일부를 복구 가능하도록 만들고 다른 메시지는 그렇지 않도록 만들 수 있습니다. 논리 메시지가 복구 가능한 상태는 문제점 해결 및 지원에서 복구 처리에 대한 문제점 해결에 설명되어 있습니다.

라우팅된 메시지는 복구에 적합하며, 프린시פל 기능에 대해 작성된 메시지도 마찬가지입니다. 라우팅은 987 페이지의 『메시지 라우팅』에 설명되어 있습니다.

누적 출력 — ACCUM 옵션

ACCUM 옵션을 사용하면 임의 숫자의 SEND MAP 명령과 페이지 크기 미만 맵의 출력을 누적해서 빌드할 수 있습니다.

이 옵션이 없으면, 각 SEND MAP 명령은 한 페이지(배열이 PAGING인 경우) 또는 전체 메시지(TERMINAL 또는 SET인 경우)에 대응합니다. 그러나 ACCUM이 있으면 BMS는 출력을 형식화하지만 한 페이지에 맞춤 이상으로 누적했거나 사용자가 논리 메시지를 종료할 때까지 출력을 폐기하지 않습니다. 원하는 경우 페이지 구분을 가로채거나 BMS가 자동으로 처리하도록 둘 수 있습니다.

페이지 크기는 터미널 정의에 있는 PAGESIZE 또는 ALTPAGE 값에 의해 판별됩니다. PAGESIZE는 트랜잭션이 실행 중인 PROFILE이 기본 화면 크기를 지정하는 경

우에 사용되며, ALTPAGE는 대체 화면 크기를 표시하는 경우에 사용됩니다. (화면 크기와 달리, 페이지 크기는 ERASE 명령에 포함할 수 있는 DEFAULT 및 ALTERNATE 옵션의 영향을 받지 않습니다.)

플로팅 맵: BMS가 ACCUM을 사용하여 맵을 위치시키는 방법

맵은 *float*할 수 있습니다. 즉, 맵은 이미 동일한 페이지와 페이지의 에지에 쓴 맵에 관련하여 위치될 수 있습니다.

플로팅 맵은 여러 화면 크기를 지원하거나 헤더, 세부사항 행 및 트레일러 외부에서 조금씩 페이지를 빌드해야 하는 경우 프로그램 로직을 저장합니다.

이를 수행할 수 있도록 하는 BMS 옵션은 다음과 같습니다.

- JUSTIFY
- HEADER 및 TRAILER
- LINE 및 COLUMN 옵션에 대한 상대 값(NEXT 및 SAME)

ACCUM 옵션으로 컴포지트 화면을 빌드하는 경우, 특정 맵의 화면에서 위치는 다음에 의해 판별됩니다.

- 전송 시 화면에 남아 있는 공간
- 맵 정의에서 JUSTIFY, LINE 및 COLUMN 옵션 값

페이지의 남은 공간은 다시 다음 사항의 영향을 받습니다.

- 이미 현재 페이지에 위치된 맵
- “오버플로우 처리”에 참여하는지 여부. 즉, 페이지 구분 시 발생하는 처리. 참여하는 경우, 맵핑 세트에서 트레일러 맵의 크기도 요인입니다.

나열하려고 하는 배치 규칙은 FIRST 및 LAST의 JUSTIFY 값이 무시되어도 ACCUM을 지정하지 않은 경우에도 적용됩니다. 그러나 ACCUM이 없으면, 각 SEND MAP은 별도의 페이지에 해당되므로 페이지의 남은 공간은 항상 전체 페이지입니다.

페이지 중단: BMS 오버플로우 처리

맵핑된 논리 메시지를 빌드할 때, 페이지 중단이 발생할 때, 즉 방금 전송한 맵이 현재 페이지에 맞지 않을 때 알리도록 BMS가 알리도록 요청할 수 있습니다.

이 기능은 ACCUM으로 컴포지트 페이지를 형성하는 경우에 유용합니다. 이 기능을 사용하여 현재 페이지의 맨 아래에 트레일러 맵을 놓고 다음 페이지의 맨 위에 헤더 맵을 놓을 수 있고, 페이지에 번호를 매기는 등의 작업을 수행할 수 있습니다.

BMS는 다음 중 하나의 경우에 페이지 중단에서 프로그램에 제어를 부여합니다.

- OVERFLOW 상태에 대해 레이블 이름을 지정하여 HANDLE CONDITION 명령을 실행했거나,

- SEND MAP 명령에 RESP 또는 NOHANDLE 옵션과 함께 NOFLUSH 옵션을 지정합니다.

이 조치 중 하나에 두 가지 효과가 있습니다.

- 페이지에 남아 있는 공간의 계산이 변경됩니다. 전송 중인 맵 자체가 트레일러 맵이 아니면, BMS는 사용자가 결국 현재 페이지에서 트레일러 맵을 원한다고 가정합니다. 따라서 동일한 맵핑 세트에서 가장 큰 트레일러에 대한 공간으로 예약합니다. (가장 큰 트레일러 맵은 가장 많은 행이 있는 TRAILER 옵션을 포함하는 맵입니다.) 페이지 중단에 관심이 없는 경우(또는 트레일러 맵을 전송하는 경우), BMS는 페이지의 진짜 끝을 사용하여 현재 맵이 맞는지 판별합니다.
- 맵이 현재 페이지에 맞지 않으면 제어 플로우가 변경됩니다. 이러한 상태가 발견될 때, BMS는 OVERFLOW 상태를 발생시킵니다. 그런 다음 오버플로우를 발생한 SEND MAP 명령을 처리하지 않고 태스크로 제어를 리턴합니다. HANDLE CONDITION 명령을 사용한 경우 이 명령에 이름 지정된 위치로 제어가 이동합니다. NOFLUSH를 사용하면, 제어는 보통과 같이 SEND MAP 이후의 명령문으로 이동하므로, EIB에서 RESP 값 또는 EIBRESP를 테스트해야 합니다.

프로그램이 오버플로우 이후 제어를 가져오는 경우 다음을 수행해야 합니다.

- 현재 페이지에서 원하는 트레일러 맵을 추가합니다. BMS는 방금 사용한 맵핑 세트에서 가장 많은 행이 있는 맵에 대해 공간을 유지합니다. 이것이 예약할 올바른 행 수가 아니거나, 몇 개의 맵핑 세트를 사용 중인 경우, 적용될 수 있는 맵핑 세트에서 더미 맵을 포함하여 적절한 양을 보장할 수 있습니다. 더미 맵은 TRAILER를 지정해야 하고 예약할 행 수를 포함해야 합니다. SEND MAP 명령에서 이를 사용하지 않아도 됩니다.
- 다음 페이지의 맵 위에서 원하는 헤더 맵을 씁니다.
- 첫 번째 위치에 오버플로우를 야기한 맵을 다시 전송합니다. 오버플로우가 발생할 때 데이터 및 맵 이름을 추적해야 합니다. BMS는 이 정보를 자동으로 저장하지 않습니다. 오버플로우를 발생할 수 있는 여러 SEND MAP 명령을 가지고 있으면, NOFLUSH를 사용하는 경우보다 HANDLE CONDITION OVERFLOW를 사용하는 경우 다시 실행해야 하는 것을 판별하는 데 필요한 프로그램 로직이 한층 복잡합니다.

OVERFLOW가 켜지면, 나머지 공간 계산에 계속 오버플로우 규칙을 사용하여도, 출력이 현재 페이지에 맞지 않을 때 BMS가 사용자 프로그램에 제어를 리턴하는 것을 일시중단시킵니다. BMS가 헤더 또는 트레일러가 아닌 맵을 이름 지정하여 첫 번째 SEND MAP을 처리할 때까지 OVERFLOW는 계속 유지됩니다. 이로서 OVERFLOW에 대해 HANDLE CONDITION을 사용하지 않도록 설정하거나 사용자의 응답 코드 테스트를 변경하지 않고 트레일러와 헤더를 전송할 수 있고, 일반 출력으로 리턴하는 즉시 오버플로우 로직을 복원시킵니다. (원래 오버플로우를 발생한 맵을 다시 전송하는 것은 보통 오버플로우 상태를 끄는 이벤트입니다.)

오버플로우를 인터셉트하지 않으면, BMS는 페이지 중단이 발생하는 시기를 프로그램에 알리지 않습니다. 그 대신, 설정한 배치 종료일에 해당하는 현재 페이지를 처리하고, 오버플로우의 원인이 되는 맵에 대해 새 페이지를 시작합니다.

맵 배치 규칙

맵은 기본적으로 화면에서 맨 위에서 맨 아래 방향으로 배치됩니다. 전체 플로우를 맨 위에서 아래로 유지보수하는 경우, 공간이 허용된다면 맵을 나란히 배치할 수도 있습니다.

지정된 SEND MAP ACCUM 명령에 대해 정확한 규칙은 다음과 같습니다.

1. 맵이 시작되는 가장 높은 행은 다음과 같이 결정됩니다.
 - a. 맵핑 정의에 JUSTIFY=FIRST가 있는 경우, 페이지에 이미 있는 유일한 맵이 오버플로우 처리 중에 페이지에 배치된 헤더인 경우를 제외하고는 BMS는 즉시 새 페이지로 이동합니다(978 페이지의 5 단계). 이 경우, BMS는 1c단계로 계속 진행합니다.
 - b. 맵이 JUSTIFY=LAST를 지정하면 BMS는 페이지에 맞게 허용된 맨 마지막 행에서 맵을 시작합니다. 맵이 트레일러 맵이거나 오버플로우를 인터셉트하지 않는 경우 또는 이미 오버플로우 처리 중인 경우에는 BMS가 페이지의 모든 공간을 사용합니다. 그렇지 않으면 BMS는 가장 큰 트레일러 맵에 대한 림을 계속 유지할 수 있기 때문에 맵을 페이지에서 낮게 배치합니다. 맵이 시작 행을 사용하여 수직으로 들어 맞는 경우, 처리는 977 페이지의 3단계에서 계속됩니다(JUSTIFY=LAST인 경우 LINE 옵션은 무시). 그렇지 않으면 오버플로우가 발생합니다(978 페이지의 5단계).

참고: JUSTIFY=BOTTOM은 ACCUM을 사용하는 출력 조작에 대해 JUSTIFY=LAST와 동일합니다. (ACCUM을 사용하지 않는 경우 및 입력 맵핑에 대해서는 차이가 있습니다. CICS 애플리케이션 프로그래밍 참조서를 참조).

- c. 수직 JUSTIFY 값이 없는 경우(또는 JUSTIFY=FIRST로 인한 임의의 오버플로우 처리가 완료된 이후) LINE 피연산자가 확인됩니다. LINE에 대한 절대 값이 지정되면 해당 행이 페이지에 가장 최근에 배치된 맵의 시작 행 또는 그 아래에 있는 경우, 해당 행이 사용됩니다. 값이 그 위치 위에 있는 경우 BMS는 978 페이지의 5단계에서 새 페이지로 이동합니다.

LINE=NEXT인 경우, 처음으로 완료된 사용되지 않은 행(현재 페이지에서 있는 모든 맵 아래)이 사용됩니다. LINE=SAME인 경우, 가장 최근에 전송된 맵의 시작 행이 사용됩니다.

2. 이제 BMS는 잠정적인 시작 행이 지정되면 맵이 화면에 수직으로 맞는지 확인합니다. 여기에서 다시 BMS는 맵이 트레일러 맵이거나 오버플로우를 인터셉트하지 않거나 또는 이미 오버플로우를 처리 중인 경우 남아 있는 모든 공간을 사용합니다.

그렇지 않은 경우, BMS를 사용하려면 맵이 맞으면서도 가장 큰 트레일러 맵에 사용할 공간이 남아 있어야 합니다. 맵이 수직으로 맞지 않으면 BMS가 새 페이지를 시작합니다(978 페이지의 5단계).

3. 그런 다음 BMS는 맵이 제안된 시작 행을 가정하여 수평으로 맞는지 확인합니다. 수평 패지에서 LEFT 및 RIGHT의 JUSTIFY 옵션 값이 사용됩니다. LEFT가 기본값이며 이는 COLUMN 값이 맵의 왼쪽을 참조하는 것을 나타냅니다. COLUMN의 숫자 값은 페이지 왼쪽부터 계수하여 맵의 왼쪽 에지가 시작되어야 하는 위치를 나타냅니다. COLUMN=NEXT는 시작 행의 왼쪽에서 사용되지 않은 첫 번째 열에서 맵을 시작합니다. COLUMN=SAME은 맵의 왼쪽 열이 JUSTIFY=LEFT로 지정한 화면에 가장 최근에 배치되었으며 이는 트레일러 맵의 헤더가 아님을 나타냅니다.

JUSTIFY=RIGHT는 COLUMN 값이 맵의 오른쪽 에지를 참조하는 것을 나타냅니다. 숫자 값은 오른쪽부터 계수하여 맵의 오른쪽 에지가 시작되어야 하는 위치를 나타냅니다. COLUMN=NEXT는 오른쪽부터 첫 번째로 사용 가능한 열을 의미하며 COLUMN=SAME은 JUSTIFY=RIGHT를 포함하며 트레일러의 헤더가 아닌 가장 최근에 배치된 맵의 오른쪽 열입니다.

맵이 수평으로 맞지 않으면 BMS가 시작 행을 맵이 맞지 않거나 또는 오버플로우가 발생하는 행에 도달할 때까지 한 번에 한 행씩 아래로 조정합니다. 처리는 시작 행에 대한 각 조정 후에 수직 검사(976 페이지의 2단계)로 재개됩니다.

4. 맵이 맞으면 다음 규칙을 사용하여 BMS가 이를 현재 페이지에 추가하고 사용할 수 있는 공간을 갱신합니다.
 - 맵의 첫 행 위의 행은 전혀 사용할 수 없습니다.
 - 맵이 JUSTIFY=LEFT를 지정하면 페이지의 왼쪽 에지부터 맵의 가장 오른쪽 열까지의 모든 열은 맵의 맨 위에서 아무것도 없는 페이지(이 맵이나 이전 맵에서)의 마지막 행까지의 모든 행에서 사용할 수 없습니다.
 - 맵이 JUSTIFY=RIGHT를 지정하면 페이지의 오른쪽 에지에서 맵의 왼쪽 에지 사이의 모든 열은 맵의 맨 위에서 아무것도 없는 페이지의 마지막 행까지의 모든 행에서 사용할 수 없습니다.

978 페이지의 그림 154에서는 나머지 공간이 각 새 맵이 배치될 때 어떻게 주는지 보여줍니다.

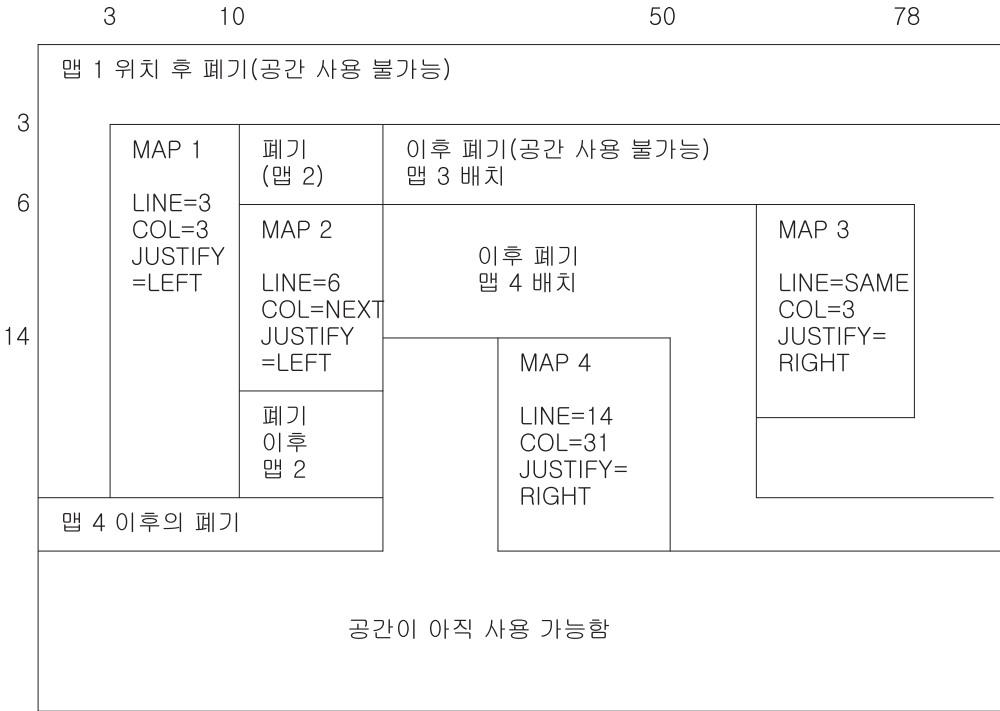


그림 154. 공간이 사용 불가능하게 되는 것을 보여주는 페이지에 대한 맵의 연속 배치

5. 현재 맵이 페이지에 맞지 않으면 BMS가 프로그램으로 제어를 리턴해야 하는지 여부를 결정합니다. 오버플로우 시에 제어를 요청했고 아직 오버플로우를 처리하지 않고 있다면 BMS는 974 페이지의 『페이지 중단: BMS 오버플로우 처리』에서 설명하는 대로 제어를 리턴합니다. 그렇지 않으면, BMS가 작성한 폐기 옵션에 따라 현재 페이지를 폐기하고 새 페이지를 시작한 후 976 페이지의 1단계에서 맞지 않았던 맵 처리를 재개합니다.

누적 처리에 대한 ASSIGN 옵션

복잡한 화면을 빌드할 경우의 복잡성을 쉽게 관리할 수 있도록, CICS는 누적 처리에 특별히 관련되는 ASSIGN 명령 옵션을 제공합니다.

다음 옵션이 누적 처리에 적용됩니다.

- MAPCOLUMN
- MAPHEIGHT
- MAPLINE
- MAPWIDTH

모두 최근에 전송된 맵에 적용됩니다. MAPHEIGHT 및 MAPWIDTH는 크기(행 및 열 수)이고 MAPLINE 및 MAPCOLUMN은 맵의 기점(상단 왼쪽 구석의 위치)입니다.

컴포지트 화면의 입력

여러 맵에서 빌드된 화면으로부터 맵핑된 입력을 읽을 수 있지만 제한사항이 있습니다.

첫 번째, RECEIVE MAP 명령에서 하나의 맵만 지정할 수 있는 반면 화면은 몇 개의 맵으로 작성되었을 수 있습니다.

두 번째, BMS는 입력에 대한 플로팅 맵의 위치를 지정하는 방법을 알 수 없어서 RECEIVE MAP 명령의 맵이 비어 있는 화면에 쓰여진 것으로 간주합니다. NEXT 또는 SAME의 LINE 또는 COLUMN 값은 입력에서 출력과 다르게 해석됩니다. JUSTIFY=LAST도 함께 무시되므로, 화면의 맨 아래에 맵을 위치시키고 거기에서 데이터를 읽으려면 JUSTIFY=BOTTOM을 사용해야 합니다.

관련 참조:

 참조 -> 애플리케이션 개발에서 RECEIVE MAP

성능 가이드

사용자가 보는 애플리케이션의 파트에 대한 전체적인 효율성에 대해 세 개의 구성요소가 있습니다. 프로세서 경로 길이, 통신 회선 활용도 및 사용자 시간입니다.

온라인 시스템이 진화함에 따라, 역점은 필요한 경우 다른 요인의 희생으로 가능한 사용자가 쉽고 빠르게 수행하도록 하는 것으로 꾸준히 이동되었습니다. 또한 프로세서 가격이 비싸짐에 따라, 디자이너는 프로그래밍 및 유지보수 노력을 줄이고 사용자 인터페이스가 더 양호하게 되도록 하기 위해 주기를 소비하려고 했습니다.

경로 길이 및 행 시간을 줄이려면 제안사항을 따르십시오. 절약이 추가 프로그래밍 노력 또는 덜 일반적인 디자인을 정당화하는지 여부와 해당 익스텐트를 스스로 판단하십시오.

경로 길이 최소화

일반적으로 단일 CICS 명령에서 실행된 명령어 수는 이를 호출한 애플리케이션에 있는 명령어의 수와 비교하여 큼니다. 지정된 태스크의 경로 길이는 다른 어떤 것보다 CICS 명령의 수와 유형에 더 많이 의존하고 명령은 튜닝하기에 가장 쉬운 영역입니다.

명령은 유형에 따라 다르고 지정된 명령의 경로 길이는 환경에 따라 상당히 달라질 수 있습니다.

BMS의 경우 일부 권장사항은 다음과 같습니다.

- 실용적일 때 단일 명령으로 화면(페이지)을 빌드하십시오. 적당한 양의 추가 프로그래밍이 동일한 기능을 달성할 때는 ACCUM 기능을 사용하여 컴포지트 화면 빌드를 피하고, 비정상적인 상황을 제외하고, 946 페이지의 『출력 화면 빌드』에서 맵 외부에 설명된 대로 여러 실제 쓰기를 통한 컴포지트 화면 빌드를 피하십시오.

- 한 번에 사용자가 조사할 수 있는 것보다 더 많은 출력을 생성하지 마십시오. 특히 몇몇 트랜잭션 조회는 특정 입력값을 위한 출력의 여러 페이지를 생성합니다. 이 경우에는 사용자는 일반적으로 초기 출력을 대충 페이지를 넘겨보는 대신에 검색을 좁히고 조회를 반복합니다. 너무 길어서 절대 보지 않게 될 출력을 생성하는 경로 길이를 피하려면 페이지를 합리적인 페이지 수로 제한하고, 사용자에게 마지막 페이지에 내용이 더 있음을 알리고, 사용자가 요청하는 경우 해당 지점부터 검색을 재시작하는 데 필요한 정보를 저장할 수 있습니다. 추가 프로그래밍은 최소입니다. 재시작 데이터를 저장하기 위한 방법은 314 페이지의 『트랜잭션 사이에 데이터 공유』의 내용을 참조하십시오.
- 가능한 경우, BMS "단축 경로"에 있는 명령을 사용하십시오. (적합한 명령과 터미널은 912 페이지의 『BMS 지원 레벨』에서 최소 BMS를 참조하십시오.)
- BMS 포맷이나 다른 함수를 필요로 하지 않는 매우 단순한 입력을 위한 터미널 제어 명령을 출력으로 사용합니다. 경로 길이가 결정적이면 터미널 제어를 전체적으로 사용하고 싶을 수도 있습니다. 그러나, 유연성, 초기 프로그래밍 노력 및 유지보수성의 관점에서 터미널 제어에 대한 BMS의 장점은 중요하고 일반적으로 경로 길이의 단점을 능가합니다.

메시지 길이 줄이기

인바운드와 아웃바운드 메시지 둘 모두의 길이를 줄이기 위해 3270 하드웨어의 장점을 이용할 수 있습니다. 터미널과 프로세서 간의 링크에서 대역폭이 강요되면 메시지가 짧으면 전체적으로 더 나은 응답을 받게 됩니다.

그러나, 지정된 전송의 시간은 그 당시에 해당 링크를 사용하는 다른 사용자에게 달려 있으므로, 직접적으로 개선되지 않을 수도 있습니다. 여기에 3270 데이터스트림의 길이를 줄이기 위한 몇몇 가능성이 있습니다.

- 화면을 전송할 때 불필요하게 MDT를 켜지 마십시오. 연관된 입력 필드가 입력에 전송되게 하기 때문입니다. 일반적으로 사용자가 필드에 입력할 때 하드웨어가 이를 수행하므로 태그를 설정할 필요가 없습니다. 태그는 화면이 아무리 여러 번 전송되더라도 프로그램이 명시적으로 끄기 전까지는 켜진 상태로 남아 있습니다(FRSET, ERASEAUP 또는 ERASE를 사용하거나 태그가 꺼진 상태에서 속성 바이트를 덮어쓰는 방법으로). 프로그램을 사용하여 이를 설정해야 하는 유일한 때는 화면에 있는 데이터를 사용자가 수정하지 않는 필드에 저장하고 싶을 때나 오류가 있는 필드를 강조표시하고 사용자가 필드를 변경하는지와 관계없이 필드를 리턴하고 싶을 때입니다. 이 경우 강조표시와 마찬가지로 MDT를 켜야 합니다.
- 화면의 입력을 재전송하고 싶지 않을 때 FRSET을 사용하여 MDT를 재설정하십시오. 즉, 이를 저장한 경우에는 사용자가 동일 화면의 후속 전송 시에는 이를 변경할 필요가 없습니다.
- 입력에서 공백은 전송되지만 널(null)은 전송되지 않으므로 화면을 보낼 때에는 입력 필드를 공백으로 초기화하지 마십시오. 따라서 널로 초기화하는 경우에는 데이터 스

트림은 수정된 각 필드에서 사용하지 않는 위치만큼 줄어듭니다. 입력을 맵핑하는 경우에는 화면에서의 모양은 같지만 프로그램으로 리턴된 데이터 또한 동일합니다.

- 단일 화면 데이터 항목 작업의 경우 ERASEAUP을 사용하여 화면을 다시 전송하는 대신에 화면에서 데이터를 지우십시오.
- 화면을 갱신하는 중이라면 변경된 데이터만을 전송하십시오. 특히 오류가 있는 필드를 강조표시하거나 화면에 메시지를 추가할 때처럼 변경이 크지 않은 경우입니다. BMS에서는 데이터 스트림을 줄이고 경로 길이를 줄이기 위해 DATAONLY 옵션을 사용할 수 있습니다. (945 페이지의 『기호 및 물리적 맵핑 병합』에서 DATAONLY 옵션을 참조하십시오.) 실제로, 필드를 강조표시하려면 새 속성 바이트만을 전송하면 됩니다. 필드 데이터는 화면에 방해받지 않은 채로 남아 있습니다.
- 터미널 제어 명령을 사용 중인 경우에는 공백과 널(null)이 있는 간격 대신에 SBA(Set Buffer Address) 및 RA(Repeat-To-Address) 순서로 포맷하십시오. (BMS는 사용자를 위해 이를 자동으로 수행합니다.)

텍스트 출력

텍스트 출력을 터미널에 보내고 있고 후속 입력을 위해 화면을 형식화하지 않는 경우, 맵을 작성하지 않아도 됩니다. BMS는 이러한 목적을 위해 명확하게 **SEND TEXT** 명령을 제공합니다.

SEND TEXT를 사용할 때, BMS는 경로가 지정되는 터미널에 대해 적절한 너비 및 깊이의 페이지로 출력을 분리합니다. 행은 단어 경계에서 분리되고 원하면 헤더 및 트레일러 텍스트를 각 페이지에 추가할 수 있습니다. 페이지 크기는 다른 BMS 출력의 경우와 같이 판별됩니다(969 페이지의 『SEND PAGE 명령』 참조).

SEND TEXT 명령

여러 유형의 형식화가 수행되는 것을 제외하고, SEND TEXT 명령은 SEND MAP과 아주 유사합니다. FROM 옵션에서 형식화될 텍스트의 위치를 지정하고 LENGTH 옵션에서 해당 길이를 지정합니다.

맵핑된 출력에 적용되는 다음과 같은 거의 모든 옵션은 텍스트 출력에도 적용됩니다.

디바이스 제어사항

FORMFEED, ERASE, PRINT, FREEKB, ALARM, CURSOR.

형식화 옵션

NLEOM, L40, L64, L80, HONEOM.

배치 옵션

TERMINAL, PAGING, SET.

페이지 형식화 옵션

ACCUM.

일반적으로, 이 옵션은 SEND MAP 명령에서와 같이 SEND TEXT 명령에서 동일한 의미를 갖습니다. SEND TEXT 명령 자체는 표준 BMS를 요구합니다. 맵핑된 환경에서 전체 BMS를 요구하는 ACCUM, PAGING 및 SET와 같은 옵션은 텍스트 환경에서도 전체 BMS를 요구합니다.

SEND MAP 컨텍스트에서 맵과 연관된 기능에 해당되는 SEND TEXT의 옵션도 있습니다. HEADER, TRAILER, JUSTIFY, JUSFIRST 및 JUSLAST입니다. 작동 방법은 『텍스트 페이지』에서 설명합니다.

SEND TEXT로 전달하지 않는 두 개의 SEND MAP 옵션은 ERASEAUP 및 NOFLUSH입니다. 텍스트는 최소한으로 필드를 사용하므로 ERASEAUP이 적용되지 않고, BMS는 텍스트 출력에서 OVERFLOW 상태를 발생하지 않으므로 NOFLUSH가 적용되지 않습니다.

텍스트 논리 메시지

SEND TEXT 명령에 ACCUM 또는 PAGING 옵션이 있으면 SEND MAP 명령에서와 마찬가지로 논리 메시지를 빌드함을 BMS에 알립니다.

텍스트 논리 메시지에는 맵핑된 논리 메시지와 동일한 규칙이 적용됩니다(968 페이지의 『논리 메시지 빌드』 참조). 특히, SEND TEXT 및 SEND CONTROL 명령을 둘 다 사용하여 메시지를 빌드할 수 있지만, 별도로 언급된 경우를 제외하고 SEND MAP에서 혼합할 수 없습니다. 또한 맵핑된 메시지와 동일한 방법으로 메시지를 종료할 수 있습니다(968 페이지의 『BMS 논리 메시지』 참조).

텍스트 페이지

SEND TEXT를 사용하는 페이지 구성은 SEND MAP을 사용하는 페이지 구성과 약간 다릅니다.

먼저, 단일 SEND TEXT 명령은 화면 또는 프린터 페이지 용량보다 더 많은 출력을 생성할 수 있습니다(SEND MAP의 경우 이러한 조건은 발생하지 않음). BMS는 전체 메시지를 전송합니다. 즉, 논리 기능을 사용하지 않고 다중 페이지 메시지를 프린터에 전달할 수 있습니다. 그러나 표시장치에서는 동일한 기술을 사용할 수 없습니다. BMS가 전체 메시지를 전달해도 구성요소 화면은 메시지를 서로 오버레이하므로 너무 빨라서 메시지를 읽을 수 없습니다.

ACCUM을 지정하면 BMS는 자동으로 출력을 여러 페이지로 구분합니다. 두 번째 차이점은 SET의 배치를 지정하지 않는 한, 태스크가 페이지 구분을 제어하지 못한다는 점입니다. 대신, 현재 페이지에 추가 공간이 없으면 BMS는 새 페이지를 시작합니다. 헤더 및 트레일러(있는 경우)를 자동으로 추가하고 오버플로우 조건을 발생시키지 않습니다. 단일 SEND TEXT 명령으로 페이지를 생성했거나 여러 명령으로 메시지를 단편적으로 빌드한 경우 이 조건이 적용됩니다. 태스크에 페이지 구분을 알리는 유일한 상

황은 배치가 SET인 경우입니다. 이 경우 BMS는 RETPAGE 조건을 발생시켜 하나 이상의 페이지가 완성되었음을 알립니다(951 페이지의 『SET 사용』 참조).

다음은 ACCUM을 사용하여 BMS가 텍스트의 페이지를 빌드하는 방법에 대한 세부사항입니다.

1. 모든 메시지는 페이지 1에서 시작하며, 처음에는 비어 있습니다.
2. HEADER 옵션을 지정하면 BMS는 모든 페이지를 헤더 텍스트로 시작합니다. BMS는 헤더 또는 트레일러(사용자가 원하는 경우)에서 페이지 번호를 지정합니다. (헤더 형식 및 페이지 번호 지정은 984 페이지의 『헤더 및 트레일러 형식』에서 설명합니다.)
3. 자리맞춤 옵션 중 하나를 지정한 경우(JUSTIFY, JUSFIRST, JUSLAST), BMS는 표시된 행에서 텍스트를 시작합니다. JUSFIRST는 헤더 다음의 첫 번째 행 또는 헤더가 없는 경우 맨 위 행에서 텍스트를 시작합니다. JUSTIFY=n은 n번째 행에서 텍스트를 시작하고 JUSLAST는 가장 낮은 행 및 트레일러(있는 경우) 모두를 현재 페이지에 맞춤 수 있도록 가장 낮은 행에서 텍스트를 시작합니다. 현재 페이지의 내용 때문에 BMS가 여기서 지정한 자리맞춤 옵션을 인식하지 못하면 BMS는 6 단계에서 먼저 새 페이지로 이동합니다.

자리맞춤은 각 SEND TEXT 명령에 대한 데이터의 시작에만 적용됩니다. 데이터 길이 때문에 추가 페이지가 필요한 경우 BMS는 여기서 처음으로 사용 가능한 위치에서 텍스트를 계속합니다.

4. 자리맞춤을 지정하지 않으면 BMS는 첫 번째 사용 가능한 위치에서 텍스트를 시작합니다. 메시지의 첫 번째 SEND TEXT에서 이 선택사항은 JUSFIRST와 동일하게 작동합니다. 그러면 텍스트는 현재 논리 메시지의 이전 SEND TEXT부터 텍스트 다음에 하나의 문자를 이어 입력합니다. (중간에 삽입되는 문자는 3270 터미널의 경우 속성 바이트이고 이외에서는 공백입니다.)
5. BMS는 시작 위치를 판별한 후 계속해서 페이지를 구분하고 984 페이지의 『텍스트 행』에서 설명한 대로 공간이 없거나 데이터가 없어질 때까지 데이터를 여러 행으로 구분합니다. 트레일러를 지정한 경우 트레일러의 요구사항으로 인해 사용 가능한 공간이 줄어듭니다. 공간보다 먼저 데이터가 소진되면 이때 처리가 종료됩니다. SEND PAGE 또는 PURGE MESSAGE 명령으로 작업을 마쳤음을 표시하면 메시지가 완성됩니다.
6. 현재 페이지에 텍스트가 맞지 않으면 트레일러 텍스트(있는 경우)를 맨 아래 추가하여 완성하고, 맵핑된 논리 메시지에서도 같이 설정한 배치 옵션(TERMINAL, PAGING 또는 SET)에 따라 배치합니다. 트레일러는 헤더와 같이 옵션이며, TRAILER 옵션으로 지정합니다(984 페이지의 『헤더 및 트레일러 형식』 참조).
7. 그런 다음, BMS는 새 페이지로 이동하고 남은 데이터를 사용하여 2 단계부터 반복합니다.

텍스트 행

BMS는 올바른 규칙을 사용하여 터미널에서 표시할 텍스트를 여러 행으로 구분합니다.

텍스트를 여러 행으로 구분하는 경우 BMS는 다음 규칙을 사용합니다.

1. 보통 각 행은 공백으로 보이는 항목으로 시작합니다. 3270 디바이스에서 화면 또는 인쇄된 페이지의 나머지 행을 차지하는 필드의 속성 바이트에 해당합니다. 다른 디바이스의 경우 공백 또는 캐리지 제어 문자입니다.

출력을 작성하는 태스크가 PRINTERCOMP(YES)를 지정하는 PROFILE에서 실행 중이고 출력 디바이스가 3270 프린터인 경우 예외가 발생합니다. 이 경우 각 행의 시작 위치에서 문자는 예약되지 않습니다. 883 페이지의 『PRINTERCOMP 옵션』의 내용을 참조하십시오.

2. BMS는 모든 공백을 포함하여 문자에 대한 텍스트 문자를 복사하고 이때 행 끝에서 두 가지 예외가 발생합니다.
 - 행이 단어 중간에 끝나면 BMS는 현재 행을 공백으로 채우고 잘린 단어를 다음 행에서 처음 사용 가능한 위치에 배치합니다. 이 목적에서 "단어"란, 공백을 포함하지 않는 연속된 문자열입니다.
 - 두 개의 단어를 하나의 공백으로 구분하고 첫 번째 단어는 공백의 여지 없이 현재 행에 맞는 경우 공백이 제거되고 다음 행은 두 번째 단어의 시작으로 시작됩니다.
3. 대상 터미널이 프린터인 경우 줄 바꾸기(NL) 문자와 기타 인쇄 형식 순서 및 공백을 임베드하여 형식을 제어할 수 있습니다. NL 및 탭은 특히 열 형식 데이터에 유용하며, BMS는 이러한 문자를 필터링하거나 해석하지 않습니다. 그러나 인쇄 형식 순서는 표시를 형식화하지 않습니다. 인쇄 형식 순서 사용에 대한 자세한 정보는 879 페이지의 『CICS 3270 프린터』의 내용을 참조하십시오.
4. 또한 출력에서 set 속성(SA) 순서 시퀀스를 포함할 수도 있습니다. (각 항목은 데이터 스트림에서 단일 문자의 속성을 설정합니다(767 페이지의 『설정 속성 순서』 참조).) SA 시퀀스가 정확히 3바이트 길이이고 올바른 속성 유형을 표시하지 않으면 BMS는 태스크를 이상 종료시킵니다. 그러나 속성을 지원하지 않는 터미널에 대해 올바른 SA 시퀀스를 사용하면 BMS는 SA 시퀀스를 제거하고 메시지를 전송합니다. SA 순서가 있는 속성 세트는 후속 순서로 덮어쓰거나 다른 SEND TEXT 명령(기본값으로 다시 설정함)을 사용할 때까지 그대로 남아 있습니다.

텍스트에서 SA 이외의 3270 순서를 포함해서는 안 됩니다. BMS는 이 순서를 표시 데이터로 처리하고 의도한 대로 형식화하지 않습니다. 터미널 오류가 발생할 수도 있습니다.

헤더 및 트레일러 형식

텍스트 메시지의 페이지에 헤더나 트레일러를 배치하려면 설정된 형식의 데이터 블록을 가리키십시오.

텍스트 메시지의 페이지에 헤더를 배치하려면 HEADER 옵션에서 다음 형식의 데이터 블록을 가리키십시오.

L	L	P	C	PNFLD
<———— TEXT ————>				

트레일러에는 동일한 형식을 사용하되, TRAILER 옵션으로 가리키십시오. 여기서,

LL 헤더(트레일러) 데이터의 길이입니다(4바이트의 LL, P 및 C 문자는 제외). LL 은 반자 2진 양식으로 표현해야 합니다.

P 페이지 번호 대체 문자입니다(아래 PNFLD 참조). 페이지 번호를 원하지 않으면 공백을 사용하십시오.

C 예약된 1바이트 필드입니다.

TEXT 각 출력 페이지 맨 위(맨 아래)에 배치될 헤더(트레일러)입니다. 다중 행을 원하면 줄 바꾸기 문자(X'15')를 사용하여 행 바꾸기가 발생해야 하는 위치를 표시하십시오.

PNFLD

헤더(트레일러) 텍스트 내 페이지 번호 필드입니다. 출력 페이지에 번호를 매기려면, 그렇지 않으면 헤더(트레일러) 텍스트에 나타나지 않는 문자를 선택하십시오. 이 문자를 페이지 번호가 나타날 위치에 배치하십시오. 가져올 예상 페이지 번호 수(32,767은 BMS에서 허용하는 최대값임)에 따라 1 - 5개의 인접 위치를 사용할 수 있습니다. 위 P 필드에 동일한 문자를 배치하여 BMS에 대체할 위치를 알려십시오. P에는 X'0C', X'15', X'17', X'26' 또는 X'FF'를 사용하지 마십시오. 이러한 값은 다른 용도로 예약되어 있습니다. 페이지 번호 매기기를 원하지 않으면 P에 공백(X'40')을 배치하십시오.

논리 메시지를 빌드하는 경우에는 페이지 구분이 발생할 때 존재하도록 각 SEND TEXT 명령에서 HEADER 및 TRAILER 옵션을 반복해야 하며, 메시지를 종료하는 SEND PAGE 명령에 트레일러를 다시 지정해야 합니다.

다음은 페이지에 번호를 매기는 헤더에 대한 COBOL 정의 예제이며, 99까지 숫자를 비워둡니다.

```
EXEC CICS SEND TEXT FROM (OUTPUT-AREA)
      HEADER(HEADER-TEXT) PAGING ACCUM END-EXEC.
```

여기서,

```

01 HEADER-TEXT
02 HEADER-LL      PIC S9(4) COMP VALUE +11.
02 HEADP          PIC X VALUE '@'.
02 FILLER          PIC X VALUE LOW-VALUE.
02 HEADING        PIC X(11) VALUE 'PAGE NO. @@'.

```

SEND TEXT를 사용하여 빌드된 화면은 터미널 운영자로부터의 광범위한 입력을 위해 설계되지 않습니다. 그러나 화면의 필드 구조가 적합하고 운영자가 예상 내용을 알고 있거나 확인할 수 있으면 어텐션 ID를 해석하고 단순 입력(예: CSPG 트랜잭션에서 페이지 표시를 제어하는 데 사용되는 입력)을 읽을 수 있습니다. (새 필드는 메시지를 구성한 각 SEND TEXT 명령을 사용하여 전송된 텍스트의 첫 번째 문자에서는 물론 각 라인에서도 시작됩니다. 정의된 필드는 비보호, 영숫자 및 정상 강도이므로 운영자가 필드에 입력할 수 있습니다.) 일반적으로 이 상황에서는 터미널 제어 RECEIVE가 사용됩니다. 화면의 필드 구조와 일치하는 필드 구조를 가진 맵을 빌드할 수 있는 경우에만 RECEIVE MAP을 사용할 수 있습니다.

SEND TEXT MAPPED 및 SEND TEXT NOEDIT

BMS는 이미 형식화된 출력에 대해 BMS의 메시지 전달 기능 중 일부를 사용할 수 있는 SEND TEXT 명령의 두 가지 특수 양식을 제공합니다. SEND TEXT MAPPED는 BMS에서 이전에 빌드했으며 SET 옵션으로 캡처한 디바이스 종속 데이터의 페이지를 전송합니다. SEND MAP 또는 SEND TEXT 명령을 사용하여 원래 페이지를 빌드할 수 있습니다.

SEND TEXT NOEDIT는 유사하지만 프로그램 또는 BMS 이외의 방법으로 빌드된 디바이스 종속 출력의 페이지를 전송하는 데 사용됩니다.

TERMINAL의 배치를 사용하여 프린시פל 기능에 이러한 페이지를 개별적으로 전달하거나 PAGING 옵션으로 빌드된 논리 메시지에 포함할 수 있습니다. 논리 메시지에서 이러한 양식은 각 BMS SEND가 별도의 페이지를 표시하는 한(즉, ACCUM 옵션을 사용하지 않음), 보통 SEND TEXT 명령 또는 SEND MAP 명령과 혼합될 수 있습니다. 동일한 논리 메소드에서 맵핑된 출력과 텍스트를 혼합할 경우 일반적인 제한사항은 이때 적용되지 않습니다. 이미 페이지가 구성되었기 때문입니다.

또한 이 명령은 라우팅 환경에서 사용할 수도 있습니다. 고유한 터미널로 라우팅하거나 전송하는 경우 데이터 스트림이 대상에 적합한지 확인해야 합니다. BMS는 대상에서 지원하지 않는 3270 속성을 제거하는 것 외에 전송 전에 확인하지 않습니다.

페이지 형식화 옵션(ACCUM, JUSTIFY, JUSFIRST, JUSLAST, HEADER, TRAILER)은 이러한 명령에 적용되지 않습니다. 페이지는 이미 정의를 통해 형식화되어 빌드되었기 때문입니다.

MAPPED 및 NOEDIT 양식의 기본적인 차이점은, SEND TEXT MAPPED는 SET 옵션에서 리턴하는 페이지 끝에 BMS가 추가하는 4바이트 페이지 제어 영역(PGA)을

사용한다는 점입니다. 이 영역은 사용할 쓰기 명령과 쓰기 제어 문자, 페이지에서 사용하는 확장 속성과, 페이지가 양식 피드, SCS 형식의 데이터, 14비트 또는 16비트 버퍼 주소, 구조화된 필드, FMH를 포함하는지 여부를 BMS에 알립니다. 이를 통해 BMS는 한 디바이스에서 빌드된 페이지를 하드웨어 특성이 다른 디바이스로 전달할 수 있습니다. 페이지 복사나 라우팅 작업에 필요할 수도 있기 때문입니다. SEND TEXT NOEDIT를 사용하여 명령 자체에서 이러한 유형의 정보를 지정합니다. BMS에서 작성된 출력에서는 SEND TEXT MAPPED, 다른 방법으로 형식화된 출력에서는 NOEDIT를 사용해야 합니다. SEND TEXT MAPPED 또는 SEND TEXT NOEDIT를 사용하여 출력에 구조화된 필드를 포함할 수 없습니다. 이러한 출력에서는 터미널 제어 SEND를 사용해야 합니다.

SEND TEXT MAPPED 명령의 LENGTH 옵션은 페이지를 빌드할 때 리턴되는 TIOATDL 값에서 설정해야 합니다. 이 경우 PGA를 포함하지 않습니다. 그러나 SEND TEXT MAPPED로 나중에 사용하기 위해 페이지를 복사하는 경우 페이지 자체와 PGA를 복사해야 합니다(총 TIOATDL + 4바이트).

관련 정보:

951 페이지의 『SET 사용』

BMS 메시지에 대해 SET 배치를 지정하는 경우, BMS는 출력을 형식화하고 디바이스 종속 데이터 스트림 형식으로 이를 리턴합니다. 일반적으로 리턴된 데이터 스트림은 연속적으로 터미널로 보내지지만 터미널 I/O는 발생하지 않습니다.

『메시지 라우팅』

BMS의 메시지 라우팅 기능은 사용자 태스크의 프린시פל 기능이 아닌 다른 터미널에 메시지를 전송할 수 있도록 합니다. 태스크는 심지어 프린시פל 기능을 가지고 있지 않아도 됩니다.

메시지 라우팅

BMS의 메시지 라우팅 기능은 사용자 태스크의 프린시פל 기능이 아닌 다른 터미널에 메시지를 전송할 수 있도록 합니다. 태스크는 심지어 프린시פל 기능을 가지고 있지 않아도 됩니다.

라우팅은 이 터미널의 직접 제어를 태스크에 부여하지 않지만, 그 대신 각 대상에 대한 태스크의 스케줄링이 메시지를 전달하도록 합니다. 이 태스크는 CICS 제공 트랜잭션 CSPG(PAGING 배치로 메시지가 사용자 자신의 터미널에 전달되도록 하기 위해 사용되는 것과 동일한)를 실행합니다. 따라서 라우팅된 메시지를 수신하는 표시장치 터미널의 운영자는 CSPG 요청을 사용하여 메시지를 봅니다. (CSPG에 대한 자세한 정보는 972 페이지의 『터미널 운영자 페이지: CSPG 트랜잭션』의 내용을 참조하십시오.)

메시지 라우팅은 애플리케이션 메시지 전환 및 브로드캐스팅과, 인쇄에도 유용합니다(887 페이지의 『CICS 프린터 사용』 참조). 이는 터미널 사용자가 다른 터미널 및 사용

자에게 메시지를 전송할 수 있는 CICS 제공 트랜잭션 CMSG에 대한 기초입니다. CMSG 설명과 이 CMSG로 수행할 수 있는 작업에 대해서는 *CICS 제공된 트랜잭션*에서 CMSG 메시지 전환을 참조하십시오.

메시지 경로를 지정하려면, ROUTE 명령 실행으로 시작합니다. 이 명령은 메시지를 전송할 위치, 전달할 시간, 오류에 대해 수행할 사항, 기타 세부사항을 BMS에 알려줍니다. 그런 다음 메시지를 빌드합니다. 메시지는 맵핑된 메시지나 텍스트 메시지가 될 수 있습니다. 그러나 논리 메시지여야 하고(즉, ACCUM 또는 PAGING이 존재함), 배치는 TERMINAL이 아니라 PAGING 또는 SET여야 합니다. PAGING은 한층 일반적인 선택이며 뒤에 나오는 설명에서 가정합니다. 996 페이지의 『SET로 라우팅』의 라우팅 컨텍스트에 SET에 대한 설명이 있습니다.

ROUTE 명령은 SEND PAGE 명령으로 메시지를 종료할 때까지 적용되므로, 이러한 상황이 발생할 때까지 다른 명령을 실행하면 안됩니다. (메시지를 빌드하는 동안 ROUTE를 실행하면 올바르지 않은 요청 응답이 발생합니다. 논리 메시지를 시작하기 전에 두 번째 ROUTE를 실행하는 경우 첫 번째 ROUTE를 바꿉니다.) 또한 PURGE MESSAGE로 메시지를 종료할 수도 있습니다(메시지를 전송하지 않을 것을 결정한 경우). PURGE MESSAGE는 ROUTE 명령으로 설정된 라우팅 환경이 사용자의 논리 메시지와 함께 버려지도록 합니다.

메시지 대상

운영자의 특정 클래스 요청, 특정 운영자에 대한 이름 지정 또는 특정 터미널 이름 지정과 같은 여러 가지 방법으로 라우팅된 메시지의 대상을 지정할 수 있습니다.

- ROUTE 명령의 OPCLASS 옵션을 사용하여 운영자의 특정 클래스가 메시지를 수신하도록 요청할 수 있습니다. RACF 사용자 정의 또는 CICS 시작 테이블(SNT) 항목에서 운영자와 클래스가 연관됩니다.
- ROUTE 명령의 LIST 옵션으로 지시하는 라우팅 목록을 사용하여 메시지를 수신하는 특정 운영자의 이름을 지정할 수 있습니다. 운영자는 RACF 정의 또는 시작 테이블(SNT) 항목에도 지정되는 3자의 OPIDENT 값으로 식별됩니다.
- 메시지를 수신하는 특정 터미널의 이름을 지정할 수 있습니다. 이는 라우팅 목록으로 수행됩니다. 터미널은 4자의 TERMID 값으로 식별되며 적용되는 터미널 유형에 대해서는 2자의 논리 디바이스 코드로 식별됩니다.

참고: 라우팅된 메시지 대상을 지정하기 위해 특정 기능에 사인온한 운영자의 ID 또는 운영자 클래스 값을 알아야 하는 경우, ASSIGN 명령에 OPID 또는 OPCLASS 옵션을 지정하여 이를 찾을 수 있습니다.

적합한 터미널

특정 대상에 대해 메시지를 올바로 형식화하려면 BMS가 형식화하는 터미널의 특성을 알아야 합니다. 운영자 또는 운영자 클래스로 지정하는 대상에도 적용됩니다.

따라서 라우팅 목록 처리의 첫 단계는 대상을 메시지가 전달 가능한 터미널 목록으로 변환하는 것입니다. 이 "적합한 터미널" 목록은 *ROUTE* 명령 실행 시에 터미널 네트워크 상태가 포함된 *OPCLASS* 스펙과 라우팅 목록의 정보를 결합합니다.

나중에 메시지 전달이 준비되면 *BMS*는 이 목록을 사용하여 메시지를 수신하는 터미널을 판별합니다. 메시지를 수신하려면 터미널이 목록에 있어야 하지만 있다고 해도 모두 전달되는 것은 아닙니다. 터미널에 연관된 운영자 제한사항이 있을 수도 있고 전달은 나중에 수행하는 것이기 때문에 상태 또는 터미널 네이처가 변경될 수도 있습니다.

두 시기 모두에서 목록이 빌드되고 전달 시에 *BMS*는 고유의 해당 *CICS* 리전(라우팅 태스크가 실행 중이거나 실행되었던)에 설치된 터미널 정의로만 제한되며 예상한 모든 터미널 정의가 없을 수도 있습니다. 우선, 자동 설치된 터미널은 *ROUTE* 시(목록 포함에서 제외) 또는 전송 시도 시(전달 방지) 모두에서 로그인되지 않았을 수 있습니다.

다중 리전 환경에서 한 개의 리전에 인식된 터미널이 다른 리전에서는 인식되지 않을 가능성도 있습니다. 이는 *CICS* 자원 정의 안내서의 *TYPETERM* 정의 속성에서 설명하는 것처럼 정의된 방식에 따라 다릅니다. 특히, 터미널 정의가 리전을 소유하는 리전에서 *SHIPPABLE*로 이를 지정하여 리전 사이에서 공유되는 경우, 터미널은 배송이 시작되도록 하는 무엇인가가 발생하기 전까지는 다른 모든 리전에 정의되지 않습니다. 이는 일반적으로 터미널이 트랜잭션을 불확실한 리전으로 첫 번째로 라우팅할 때 발생합니다. 따라서 이 리전의 *ROUTE*는 이런 이벤트가 처음 발생하기 전에 터미널을 포함할 수 없습니다.

다음 섹션은 *BMS*가 적합한 터미널 목록을 빌드하는 방법에 대해 설명합니다. 이는 *ROUTE* 명령 실행 시에 발생합니다.

OPCLASS로만 지정되는 대상

운영자 클래스(*OPCLASS* 옵션)는 지정하지만 라우팅 목록은 지정하지 않은 경우, *BMS*는 로컬 시스템에서 모든 터미널 정의를 스캔합니다. 이 조건을 모두 충족시키는 모든 터미널은 적합한 터미널 목록에 포함됩니다.

- 터미널은 *BMS*에서 지원되는 유형입니다.
- 터미널은 터미널로 라우팅되지 않은 라우팅된 메시지는 수신할 수 있습니다(터미널 정의에서 *ROUTEDMSG* (ALL)).
- 운영자는 터미널에서 사인온합니다.
- 운영자는 *OPCLASS* 목록의 운영자 클래스 중 하나에 속합니다.

결과 항목은 *OPCLASS* 목록의 운영자 클래스 중 하나에 속하는 운영자가 사인온할 때와 사인온하는 경우에만 전달이 수행되도록 표시됩니다. (이 운영자는 *ROUTE* 시에 사인온한 운영자일 필요는 없습니다.)

OPCLASS 및 LIST 생략

운영자 클래스 또는 라우팅 목록을 지정하지 않으면 BMS는 처음 두 조건을 충족시키는 모든 터미널을 위 목록에 추가하고 전달 시에 운영자 제한사항을 설정하지 않습니다. 이는 많은 터미널이 라우팅된 모든 메시지를 수신할 자격이 되는 네트워크에서는 선택하면 안되는 선택사항일 것입니다.

라우팅 목록 제공

라우팅 목록을 제공하면 BMS가 터미널 정의를 스캔하지 않고 사용자의 제공을 사용하여 자체 목록을 빌드합니다. 항목 각각은 다음과 같이 처리됩니다. 처리에는 항목이 사용되거나 건너뛰는지 여부 및 이유를 설명하기 위해 목록 항목에 상태 플래그를 설정하는 작업도 포함됩니다.

- 항목에 터미널 ID는 있지만 운영자 ID는 없다면 터미널은 적합한 목록이 정의된 경우 BMS에서 지원되는 유형이며 라우팅된 메시지를 받을 수 있는 해당 목록으로 이동합니다. BMS가 터미널 정의를 찾지 못하면 "항목 건너뛰"를 설정하고 "올바르지 않은 터미널 ID" 비트(X'C0')를 라우팅 목록 항목의 상태 플래그에 설정합니다. 터미널이 있지만 BMS에서 지원되지 않거나 라우팅된 메시지를 받도록 허용되지 않은 경우 "항목 건너뛰" 및 "BMS가 지원하지 않는 터미널" 비트가 설정됩니다(X'A0').

참고: 라우팅된 메시지를 수신하는 터미널의 적합성은 터미널 정의에서 ROUTEDMSGS 옵션으로 제어합니다. 터미널이 모든 라우팅된 메시지를 수신하거나, 터미널이나 운영자 이름을 사용하여 라우팅된 메시지만 수신 또는 라우팅된 메시지를 수신하지 않도록 하는 세 개의 값을 사용할 수 있습니다. 라우팅 목록과 OPCLASS도 지정한 경우, BMS는 사용자가 나열한 클래스 중 하나에 속하는 운영자가 터미널에 사인온하는지 여부를 확인합니다. 사인온하지 않으면 BMS가 "운영자가 사인온하지 않음" 비트(X'10')를 항목의 상태 플래그에 설정하여 이를 사용자에게 통보하지만 여하튼 터미널은 포함합니다. 운영자 클래스를 지정하는 경우조차도 목록 항목에 연관된 운영자의 제한사항은 없습니다.

- 항목에 터미널 및 운영자 ID 모두 포함된 경우, 터미널 ID는 운영자 ID가 없는 것처럼 동일하게 확인되고 동일한 오류가 발생할 수 있습니다. 터미널이 이 테스트를 패스하면 적합한 목록으로 이동합니다. 그렇지만 이름 지정된 운영자가 동일한 터미널에 사인온하는 경우에만 메시지가 전달되도록 항목은 표시됩니다.

운영자가 ROUTE 명령 실행 시 터미널에 사인온하지 않으면 BMS는 "운영자가 사인온하지 않음" 비트(X'10')를 상태 플래그에 켜서 이를 사용자에게 통보하지만 터미널은 사인온 상태와 상관없이 전달 목록으로 이동합니다. (OPCLASS는 운영자 ID가 있는 경우 완벽하게 무시됩니다.)

- 항목에 운영자 ID만 포함된 경우, BMS는 운영자가 사인온한 터미널에서 터미널 정의를 찾을 때까지 이를 검색합니다. (운영자는 추가 터미널에 사인온했을 수도 있지만 BMS는 이를 무시합니다.) 터미널이 BMS에서 지원되는 유형이 아니거나 터미널

이 라우팅된 메시지를 수신할 수 없으면 BMS는 "항목 건너뛰" 및 "운영자가 지원되지 않는 터미널에 사인온" 비트(X'88')를 상태 플래그에 설정합니다. 또한, 라우팅 목록에 터미널 ID도 입력합니다. 터미널이 적합한 경우, 이전에 설명한 대로 BMS는 터미널 및 운영자 ID 모두를 지정한 것처럼 항목을 처리합니다.

운영자가 어디에도 사인온하지 않은 경우, BMS는 "항목 건너뛰" 및 "운영자가 사인온하지 않음" 비트(X'90')를 상태 플래그에 설정합니다.

라우팅 목록 형식

BMS에는 라우팅 목록에 대한 고정된 형식이 필요합니다. 목록의 각 항목은 16바이트 길이입니다.

표 53. 표준 라우팅 목록 항목 형식

바이트	내용
0-3	터미널 또는 논리 장치 ID(후행 공백을 포함하고 4자) 또는 공백
4,5	LDC 지원을 사용하는 논리 장치의 경우 LDC 니모닉(2자) 또는 공백
6-8	운영자 ID 또는 공백
9	경로 항목에 대한 상태 플래그
10-15	예약됨, 공백을 포함해야 함

터미널 또는 운영자 ID가 각 항목에 있어야 합니다. 논리 디바이스 구성요소(LDC)도 동반될 수 있습니다. LDC에 대한 자세한 정보는 1010 페이지의 『논리 디바이스 구성요소』에서 LDC 및 라우팅을 참조하십시오.

라우팅 목록의 항목은 보통 순차적으로 나옵니다. 그러나 모두 인접하지는 않습니다. 목록에서 불연속성이 나타나면 다음 그룹의 첫 번째 항목을 가리키는 8바이트 체인 항목으로, 마지막 그룹을 제외하고 연속된 항목의 각 그룹을 종료합니다. 이 항목은 다음과 비슷합니다.

표 54. 라우팅 목록 체인 항목 형식

바이트	내용
0,1	2진 반자 형식에서 -2(X'FFFE')
2,3	예약됨
4-7	연속된 항목의 다음 그룹에서 첫 번째 항목의 주소

전체 목록의 끝은 -1의 반자 값(X'FFFF')을 포함하는 2바이트 항목으로 표시합니다.

목록은 원하는 만큼 여러 그룹으로 구성될 수 있습니다. 총 대상 수에 대한 상한이 있지만, 많은 변수에 따라 달라집니다. 이 상한을 초과하면 BMS는 이상 종료 코드 ABMC로 태스크를 이상 종료시킵니다.

ROUTE 명령에서 리턴하면 BMS는 조건 코드를 표시하여 목록의 오류를 알립니다.

RTESOME

라우팅 목록에서 하나 이상의 항목을 사용할 수 없어서 건너뛴을 의미합니다. 기본 조치는 성공적으로 처리된 대상을 사용하여 라우팅 작업을 계속하는 것입니다.

RTEFAIL

목록의 대상을 사용할 수 없으므로 라우팅 환경이 설정되지 않음을 의미합니다. 기본 조치는 태스크로 제어를 리턴하는 것입니다. 이 조건은 순차적으로 테스트해야 합니다. 라우팅 환경이 없으면 후속 BMS SEND 명령이 프린시플 기능으로 이동하지만, 이는 의도된 동작이 아니기 때문입니다.

RTESOME 및 RTEFAIL에서 반영하는 일반 정보 외에도 BMS는 상태 플래그(9번째 바이트)를 설정하여 목록의 각 항목에서 수행한 작업을 알립니다. 널값(X'00')은 항목이 모두 올바른 의미를 의미합니다. 최상위 비트는 항목을 사용하는지, 아니면 건너뛰는지 여부를 나타내고, 다른 비트는 수행된 내용을 정확히 알립니다. 다음은 각 비트의 의미입니다.

ENTRY SKIPPED(X'80')

항목을 사용하지 않았습니다. 이 비트가 설정되면 다른 비트도 이유를 표시하기 위해 설정됩니다.

INVALID TERMINAL IDENTIFIER(X'40')

항목에서 이름 지정된 터미널에 대한 터미널 정의가 없습니다. 항목을 건너뛴니다.

TERMINAL NOT SUPPORTED UNDER BMS(X'20')

라우팅 목록 항목에 이름 지정된 터미널이 BMS에서 지원되는 유형이 아니거나 라우팅된 수신 메시지에서 제한됩니다. 항목을 건너뛴니다.

OPERATOR NOT SIGNED ON(X'10')

항목에서 이름 지정된 운영자가 사인온되지 않았습니다. 다음 조건 중 하나로 이 플래그가 설정됩니다.

- 운영자 ID 및 터미널 ID 모두 지정했으며, 운영자가 터미널에 사인온하지 않았습니다. 항목을 건너뛰지 않습니다.
- 터미널 ID 없이 운영자 ID를 지정했으며, 운영자가 터미널에 사인온하지 않았습니다. 항목을 건너뛴니다.
- OPCLASS가 ROUTE 명령에 지정되었으며 라우팅 목록 항목에 터미널 ID가 지정되었고 터미널에 사인온한 운영자에게 지정된 운영자 클래스가 없습니다. 항목을 건너뛰지 않습니다.

OPERATOR SIGNED ON AT UNSUPPORTED TERMINAL(X'08')

라우팅 목록 항목에서 운영자 ID만 지정했으며, 해당 운영자가 BMS에서 지원하지 않는 터미널에 사인온했거나 라우팅된 메시지를 수신할 권한이 없습니다. 항목을 건너뛴니다. 터미널 이름이 항목의 터미널 ID 필드에서 리턴됩니다.

INVALID LDC MNEMONIC(X'04')

다음 조건 중 하나로 이 플래그가 설정됩니다.

- 라우팅 목록에 지정된 LDC 니모닉이 이 터미널에 대해 정의되지 않았습니다. 즉, 터미널은 LDC를 지원하지만 LDC 목록이 없거나 해당 LDC 목록이 확장되었지만 이 항목을 포함하지 않습니다.
- 이 LDC 항목의 디바이스 유형이 LDC를 포함하는 라우팅 목록의 첫 번째 항목 디바이스 유형과 다릅니다(1010 페이지의 『논리 디바이스 구성요소』의 LDC 및 라우팅에서 설명한 대로 하나의 LDC 디바이스 유형만 허용됨).

항목을 건너뛵니다.

참고: CICS에서는 표준 라우팅 목록 항목을 정의하는 소스 코드와 상태 플래그 비트 조합을 테스트하는 데 필요한 값을 제공합니다. 멤버가 DFHURLDS인 COPY 또는 INCLUDE를 통해 프로그램에 이 코드를 삽입할 수 있으며, 같은 방식으로 BMS 어텐션 ID 또는 속성 바이트 정의를 포함할 수 있습니다.

메시지 전달

메시지는 명령에서 스케줄링 옵션(INTERVAL, TIME, AFTER, AT)에 따라 **ROUTE** 명령을 실행하고 얼마가 지난 후에 전달할 수 있습니다. 즉시, 시간 간격이 경과한 후 또는 하루 중 특정 시점에 전달을 요청할 수 있습니다.

예정된 시간에 도달하면 BMS는 적합한 터미널 목록의 모든 터미널에 메시지를 전달하려고 합니다. 특정 터미널에 메시지를 전달하려면 다음 모든 조건을 만족해야 합니다.

- 터미널은 BMS에서 지원하는 유형이자, ROUTE 명령을 처리할 때와 동일한 유형으로 정의되어야 합니다. 메시지의 작성 및 전달 사이에 지연 시간이 긴 경우 특정 TERMID로 정의한 터미널이 특히 자동 설치 환경에서 특성을 변경하거나 지울 수 있습니다. 3270 터미널은 ROUTE 명령을 실행할 때 존재하는 동일한 확장 속성을 그대로 보유하지 않아도 됩니다. BMS는 전달 시 데이터 스트림에서 지원되지 않는 속성을 제거하기 때문입니다.
- 터미널은 서비스 상태이고 사용 가능해야 합니다(즉, 프린시פל 기능으로 터미널에서 실행 중인 태스크가 없어야 함).
- 터미널은 자동 트랜잭션 초기화에 적합해야 합니다. 또는 터미널 운영자가 CSPG 트랜잭션으로 메시지 전달을 요청해야 합니다.

참고: 여러 메시지가 특정 터미널에 전달되도록 누적된 경우 운영자가 특정 순서로 메시지를 본다고 보장할 수 없습니다. 실제로 CSPG 트랜잭션에서 운영자는 일부 상황에서 전달 순서를 제어할 수 있습니다. 특정 순서의 페이지가 필요한 경우 하나의 메시지로 전송해야 합니다.

- 전달 목록 항목이 특정 운영자로 또는 특정 클래스의 운영자로 전달을 제한하는 경우 터미널에 사인온한 운영자의 자격이 유효해야 합니다. (이러한 제한사항을 적용하는 OPCLASS 및 LIST 스펙은 988 페이지의 『메시지 대상』의 내용을 참조하십시오.)
- 제거 지연은 다음 절에서 설명한 대로, 만료되어서는 안 됩니다.

전달할 수 없는 메시지

BMS가 적합한 터미널에 메시지를 전달할 수 없으면 정기적으로 계속 시도합니다.

다음 조건 중 하나가 나타날 때까지 전달을 재시도합니다.

- 터미널 상태가 변경되어 메시지를 보낼 수 있습니다.
- 대상 터미널 운영자에 의해 메시지가 삭제됩니다.
- 제거 지연 시간이 경과합니다.

제거 지연은 전달 스케줄이 지정된 경우 메시지 전달에 허용된 시간입니다. 이 간격이 경과하면 메시지를 버립니다. 제거 지연은 시스템 전체에 적용되는 값으로 시스템 초기화 테이블에서 PRGDLY 옵션으로 설정합니다. 이 항목의 사용은 선택사항입니다. 시스템 프로그래머가 PRGDLY를 0으로 설정하면 메시지는 무기한으로 보존됩니다.

BMS가 이 방식으로 메시지를 제거하는 경우 ERRTERM에 지정한 터미널로 오류 메시지를 전송합니다. (특정 터미널 이름 없이 ERRTERM을 사용하는 경우 원래 메시지를 작성한 태스크의 프린시פל 기능으로 메시지를 전송합니다. ERRTERM도 함께 생략하면 메시지는 전송되지 않습니다.)

복구 가능 메시지

PAGING 배치를 사용하는 라우팅된 메시지의 작성 및 전달 사이에, BMS는 보통의 PAGING 메시지 경우에서와 같이 CICS 임시 저장영역에서 메시지를 저장합니다. 따라서, 라우팅되지 않은 메시지의 경우와 같이 REQID 옵션 값을 선택하여 라우팅된 메시지를 복구 가능하도록 만들 수 있습니다.

두 개 이상의 터미널 유형으로 라우팅하는 경우, BMS는 유형마다 별도의 논리 메시지를 빌드하고(적절한 디바이스 종속 데이터 스트림을 사용하여) 유형마다 별도의 임시 저장영역 큐를 사용합니다.

참고: 두 개의 메시지 형식이 가능한, 대체 화면 크기 기능이 있는 터미널 대상의 경우, BMS는 태스크가 메시지를 작성하는 프로파일이 기본 크기를 지정하는 경우 기본 크기를 선택하고, 프로파일이 대체 크기를 지정하는 경우 대체 크기를 선택합니다.

그러나 논리 메시지가 복구 가능한지 여부를 계속 선택할 수 있도록 모든 논리 메시지가 동일한 REQID 값을 사용합니다.

BMS는 또한 메시지를 수신하기에 적절한 터미널 목록을 저장하고 전달이 발생했는지 여부를 추적하기 위해 임시 저장영역을 사용합니다. 특정 유형의 적절한 모든 터미널이 메시지를 수신한 경우, BMS는 연관된 논리 메시지를 삭제합니다. 모든 대상이 전달을 수신하거나 제거 지점이 만기되는 경우, BMS는 메시지에 대한 모든 정보를 지워서, 마스터 터미널 운영자 메시지 큐에 대상별로 전달할 수 없는 메시지 수를 보고합니다.

관련 정보:

973 페이지의 『논리 메시지 복구』

PAGING의 배치로 작성된 논리 메시지는 작성 및 전달 사이에서 CICS 임시 저장영역에 보존됩니다.

메시지 ID

원하면 라우팅된 메시지에 제목을 지정할 수 있습니다. 제목은 메시지 자체의 일부는 아니지만, BMS가 CICS 임시 저장영역에서 메시지에 대해 유지보수하는 다른 정보와 함께 포함됩니다.

제목은 운영자나 터미널에 대해 여러 메시지가 누적될 수 있는 상황에서 유용합니다. 제목을 사용하면 운영자가 메시지 표시 순서를 제어할 수 있기 때문입니다. (CSPG - 페이지 검색 CICS 제공된 트랜잭션에서 CSPG 명령의 “query” 옵션을 참조하십시오.)

제목을 지정하려면, 반자 2진 길이 필드가 앞에 있는 제목으로 구성되는 데이터 영역을 가리키도록 ROUTE 명령의 TITLE 옵션을 사용하십시오. 길이에는 2바이트 길이 필드가 포함되고 최대값이 64이므로, 제목 자체는 62자까지 가능합니다. 예를 들어, 다음과 같습니다.

```
01 MSG-TITLE.
   02 TITLE-LENGTH    PIC S9(4) COMP VALUE +19.
   02 TITLE-TEXT      PIC X(17) VALUE 'MONTHLY INVENTORY'.
...
EXEC CICS ROUTE TITLE(MSG-TITLE)....
```

그림 155. 제목 지정

라우팅에 대한 프로그래밍 고려사항

대부분의 경우 경로가 지정되지 않은 메시지를 수행하는 것과 동일한 방식으로 라우팅된 메시지를 빌드합니다. 그러나 BMS는 대상 사이에서 터미널 유형마다 별도의 논리 메시지를 빌드하기 때문에, 차이가 있습니다.

라우팅 및 페이지 오버플로우

다른 유형의 터미널은 다른 페이지 기능을 가지고 있으므로, 여러 유형에 대해 다른 시간에 페이지 오버플로우가 발생할 수 있습니다. SEND MAP 명령을 사용 중이고 오

버플로우를 인터셉트하는 경우, 프로그램은 BMS가 ROUTE에 대한 응답으로 작성 중인 각 논리 메시지의 페이지마다 오버플로우가 발생할 때 제어를 가져옵니다.

오버플로우 발생 시 페이지에 번호를 매기거나 페이지 종속 처리를 수행하려면, 각 터미널 유형에 대해 개별적으로 정보를 추적해야 합니다. 이러한 목적으로 보존되는 데이터 영역을 오버플로우 제어 영역이라고 합니다. ROUTE 명령 이후와, 오버플로우를 발생할 수 있는 BMS 명령 이전에 DESTCOUNT 옵션과 함께 ASSIGN 명령을 실행하여 필요한 이 영역의 수(즉, ROUTE 명령에 나타난 여러 터미널 유형 수)를 지시할 수 있습니다. 이 때 ASSIGN DESTCOUNT를 실행하면, BMS가 빌드하는 논리 메시지의 수가 리턴됩니다.

오버플로우가 발생할 때, 논리 메시지 오버플로우가 발생한 대상을 판별하기 위해 동일한 명령을 사용할 수 있습니다. 이 때 ASSIGN DESTCOUNT는 메시지들 사이에 BMS가 이 ROUTE 명령에 대해 빌드 중인 메시지의 상대 번호를 리턴합니다. 오버플로우 제어 영역을 사용 중인 경우 이 번호는 사용할 메시지를 알려줍니다. 이 때 ASSIGN PAGENUM을 사용하면, BMS는 또한 오버플로우가 발생한 페이지의 번호도 리턴합니다.

다양한 터미널 유형에 대해 다양한 오버플로우 위치의 복잡함을 처리하기 위해, 라우팅 환경에서 수행해야 하는 처리는 다음과 같습니다.

- ASSIGN DESTCOUNT로 오버플로우가 발생한 논리 메시지를 판별합니다(아주 단순한 오버플로우 처리를 수행 중이지 않은 경우).
- 비라우팅 환경에서 수행하는 것처럼(974 페이지의 『페이지 중단: BMS 오버플로우 처리』 참조), 현재 페이지에 대한 트레일러 맵과 다음 페이지의 헤더를 차례로 전송합니다. OVERFLOW 상태가 강제 실행되는 동안, SEND MAP 명령은 오버플로우가 발생한 논리 메시지에만 적용됩니다. (사용자는 중간 페이지인 논리 메시지에서 이 명령을 원하지 않고, BMS는 동일한 페이지 용량을 갖도록 발생하는 다른 터미널 유형에 대해 어떤 가정도 하지 않습니다.)
- 비라우팅 환경에서 수행한 것처럼, 오버플로우를 발생한 명령을 다시 실행하십시오. 그러나 수행한 후에는 오버플로우가 발생하지 않을 때까지 오버플로우에 대해 다시 테스트하고 전체 프로세스를 반복해야 합니다. 이 프로시저는 트레일러 및 헤더와, 빌드 중인 논리 메시지 각각으로 오버플로우를 발생한 맵을 가져오도록 합니다.

SET로 라우팅

라우팅 환경에서 SET의 배치(disposition)를 지정할 때, 페이지는 완료되는 대로 프로그램에 리턴되므로 라우팅 목록의 대상으로 어떤 메시지도 전송되지 않습니다.

그러나 ROUTE 명령은 대상 및 터미널 유형 사이에 해당되는 대상과 터미널 유형을 판별하기 위해 일반적인 방식으로 처리됩니다. BMS는 보통과 같이 유형마다 별도의 논리 메시지를 빌드하고 어떤 터미널 유형에 대해서도 완료될 때마다 프로그램에 페이지를 리턴합니다. BMS는 PAGING의 배치에 대해 수행하는 대로 OVERFLOW 및

RETPAGE 상태를 발생합니다. 결국, SET가 있는 ROUTING을 사용하면 프린시פל 기능의 터미널 유형이 아닌 다른 터미널 유형에 대해 메시지를 형식화할 수 있습니다.

메시지 라우팅으로 대화 인터리빙

라우팅할 메시지를 빌드하는 동안, BMS SEND 명령과 RECEIVE MAP 및 터미널 제어 명령을 사용하여 프린시פל 기능과 대화할 수 있습니다.

이와 같은 SEND 명령은 PAGING 또는 SET 명령이 아닌 다른 TERMINAL의 배치 옵션을 가지고 있어야 하며 ACCUM을 지정하면 안됩니다. 연관된 입력 및 출력은 터미널이 메시지 대상 중 하나인 경우에도 직접 처리되며 논리 메시지를 방해하지 않습니다.

라우팅 없이, 968 페이지의 『논리 메시지 빌드』에 명시된 대로 BMS SEND를 사용할 수 없습니다.

MAPPINGDEV 기능

최소 BMS 기능은 태스크의 프린시פל 기능이 TCTTE(터미널 제어 테이블 항목)에 정의된 기능과 상태의 입력과 출력 맵핑 작업을 수행하는 맵핑 디바이스임을 가정합니다.

BMS 기능을 사용하는 트랜잭션의 프린시פל 기능에는 BMS가 지원하는 디바이스 유형이 있어야 합니다. 그러나 MAPPINGDEV 기능은 프린시פל 기능이 아닌 디바이스의 맵핑 작업을 수행할 수 있도록 허용하는 최소 BMS의 확장입니다. MAPPINGDEV 요청이 완료되면 맵핑된 데이터는 애플리케이션으로 돌아갑니다. BMS에는 MAPPINGDEV 디바이스와의 통신이 없습니다.

RECEIVE MAP 명령 및 SEND MAP 명령에 MAPPINGDEV 옵션을 지정할 수 있지만 다른 BMS 명령에는 지정할 수 없습니다.

MAPPINGDEV 옵션에 지정된 TERMID는 BMS가 지원하는 3270 제품군에서 디바이스를 표시해야 합니다. 디바이스가 파티션된 경우에는 기본 상태에 있을 것으로 가정합니다. 외부 포매팅은 무시됩니다.

데이터는 최소 BMS와 같은 방식으로 맵핑되고 맵핑 세트 정의를 변경하거나 맵핑 세트를 재생성할 필요가 없습니다.

MAPPINGDEV 옵션이 있는 SEND MAP

MAPPINGDEV 옵션이 있는 SEND MAP 명령은 또한 SET 옵션을 지정해야 합니다. (SET 옵션은 BMS에 맵핑된 출력 데이터 스트림을 포함하는 저장영역의 주소를 설정하는 포인터를 제공합니다.)

저장영역 보호가 활성화되면 데이터는 트랜잭션 정의의 TASKDATAKEY 옵션에 지정된 키에서 저장영역으로 리턴됩니다. 저장영역은 트랜잭션 정의의 어떤 TASKDATALOC 옵션을 지정했는지에 따라 행의 위나 아래에 있습니다.

저장영역은 태스크 관련 사용자 저장영역에 있지만 TIOA(Terminal Input/Output Area) 형식으로 되어 있습니다. 애플리케이션은 DFHTIOA 카피북을 사용하여 저장영역을 참조할 수 있습니다. 오프셋 8에 있는 TIOATDL 필드에는 저장영역에서 오프셋 12에서 TIOADBA에서 시작하는 데이터 스트림의 길이를 포함합니다. TIOATDL에 배치된 길이 값은 4바이트 페이지 제어 영역의 길이를 포함하지 않고, 이는 데이터 스트림에 사용된 확장 속성 등과 같은 정보를 포함하고 DFHPGADS 카피북을 사용하여 참조될 수 있습니다.

일반적으로 저장영역에는 데이터 스트림보다 큰 길이가 있습니다. 저장영역은 출력 데이터 스트림의 정확한 길이가 판별되기 전에 할당되기 때문입니다. 이 저장영역은 SEND TEXT MAPPED 명령에서 사용할 수 있는 양식으로 되어 있습니다.

MAPPINGDEV 옵션 없이 SET 옵션을 사용하는 데 친숙한 경우에는 데이터 스트림이 페이지 목록에 의해 간접적으로 애플리케이션으로 리턴된다는 점을 알고 있습니다. 그러나 MAPPINGDEV가 지정되면 데이터 스트림을 포함하는 저장영역에 대한 직접 포인터가 애플리케이션으로 리턴됩니다.

SEND MAP MAPPINGDEV 명령이 처리를 완료하면 저장영역은 애플리케이션의 제어 하에 있고 애플리케이션이 FREEMAIN 요청을 사용하여 이를 해제하지 않는 한 트랜잭션의 끝까지 할당된 상태로 남아 있습니다. 장기 실행 트랜잭션의 경우 CICS는 태스크가 종료될 때 이러한 영역을 자유롭게 하므로 FREEMAIN 요청을 사용하여 이러한 저장영역을 릴리스하는 것이 바람직합니다.

MAPPINGDEV 옵션이 있는 RECEIVE MAP

RECEIVE MAP 명령에서 MAPPINGDEV 옵션을 사용할 때 FROM 옵션을 지정해야 합니다. BMS는 터미널 제어 RECEIVE 명령을 통해 리턴되는 데이터 스트림(즉, 일반 입력 3270 데이터 스트림)과 일관된 형식화된 3270 입력 데이터 스트림을 제공하기 위해서는 FROM 옵션이 필요합니다. 유일한 차이점은 이 정보는 터미널 제어에 의해 입력 데이터 스트림에서 제거되므로 AID와 입력 커서 주소로 시작하지 않는다는 점이지만, MAPPINGDEV 옵션이 지정된 경우 AID 값과 입력 커서 위치를 지정할 수 있는 RECEIVE MAP 명령의 옵션이 있습니다. 데이터 스트림에 AID와 입력 커서 주소가 포함된 경우에는 이들은 BMS에 의해 무시됩니다.

어떤 옵션도 지정되지 않은 경우에는 BMS는 입력 데이터 작업이 ENTER 키에 의해 종료되었고 적합한 AID 값을 EIBAID 필드로부터 애플리케이션으로 리턴한다고 가정합니다. BMS는 또한 입력 커서가 홈 주소에 배치되었음을 가정하고 EIBCPASN 필드에서 애플리케이션으로 0값을 리턴합니다.

RECEIVE MAP 명령의 새 AID 옵션을 사용하면 애플리케이션이 지정된 경우 ENTER의 기본값을 덮어쓰는 AID 값을 지정할 수 있습니다. 애플리케이션이 제공했거나 BMS에 의해 기본적으로 제공되었는지 여부와 관계 없이, 사용자가 설정한 AID 값은 가능한 경우 제어를 애플리케이션이 실행한 이전 HANDLE AID 요청에 의해 등록된 루틴으로 전달합니다.

RECEIVE MAP 명령의 새 CURSOR 옵션을 사용하면 애플리케이션이 지정된 경우 기본값 0을 덮어쓰는 입력 커서 위치를 지정할 수 있습니다. 애플리케이션이 제공했거나 BMS에 의해 기본적으로 제공되는지 여부와 관계 없이, 입력 커서 값은 CURSLOC=YES로 맵을 정의할 때 커서 위치 처리에 사용됩니다.

최소 BMS RECEIVE MAP 명령에서처럼 맵핑된 데이터는 INTO 또는 SET 옵션에 의해 애플리케이션으로 리턴됩니다. 어떤 옵션도 지정되지 않은 경우에는 CICS 변환기는 맵 이름에 'I' 문자를 첨부하여 기본 INTO 옵션을 적용하려고 시도합니다.

MAPPINGDEV 옵션과 함께 SET 옵션을 사용할 때는 BMS가 맵핑된 입력 데이터 스트림을 포함하는 저장영역의 주소와 함께 설정하는 포인터 변수를 제공해야 합니다. 데이터는 태스크 관련 사용자 저장영역에 리턴됩니다. 저장영역 보호가 활성화되면 데이터는 트랜잭션 정의의 TASKDATAKEY 옵션이 지정하는 키에서 저장영역에 리턴됩니다. 저장영역은 트랜잭션 정의의 TASKDATALOC 옵션에 따라 행의 위나 아래에 있습니다.

RECEIVE MAP MAPPINGDEV 명령이 처리를 완료하면 저장영역은 SET 옵션에 의해 리턴되고 애플리케이션의 제어 하에 있고 애플리케이션이 FREEMAIN 요청을 사용하여 이를 해제하지 않는 한 트랜잭션의 끝까지 할당된 상태로 남아 있습니다. 장기 실행 트랜잭션의 경우 CICS는 태스크가 종료될 때 이러한 영역을 자유롭게 하므로 FREEMAIN 요청을 사용하여 이러한 저장영역을 릴리스하는 것이 바람직합니다.

관련 참조:

 참조 -> 애플리케이션 개발에서 RECEIVE MAP

 참조 -> 애플리케이션 개발에서 SEND MAP

샘플 어셈블리 MAPPINGDEV 애플리케이션

이 예제는 사용자에게 MAPPINGDEV 기능과 연관된 키워드를 코드화하는 방법을 보여주기 위해 FILEA 운영자 명령 샘플에 대한 변경된 버전을 사용합니다.

1000 페이지의 그림 156는 FILEA 운영자 명령어 샘플 프로그램의 수정이고 이름이 DFH\$AGA인 동일한 맵핑 세트를 사용합니다.

이 애플리케이션은 MAPPINGDEV 기능과 연관된 키워드를 코딩하는 방법을 보여주기 위한 용도이고 이 기능을 테스트하기 위한 수단으로서 사용됩니다. MAPPINGDEV 기능을 사용하는 애플리케이션의 권장 디자인으로 제공되지 않습니다.

```

DFH$AMNX CSECT
*
      DFHREGS
DFHEISTG DSECT
OUTAREA DS 0CL512
        DS CL8
OUTLEN  DS H
        DS H
OUTDATA DS CL500
INLEN   DS H
INAREA  DS CL256
PROOF   DS CL60
        COPY DFH$AGA
        COPY DFHBMSCA
DFH$AMNU CSECT
      EXEC CICS HANDLE AID PF3(PF3_ROUTINE)
*
      XC  DFH$AGAS(DFH$AGAL),DFH$AGAS
      MVC MSGO(L'APPLMSG),APPLMSG
      EXEC CICS SEND MAP('DFH$AGA') FROM(DFH$AGAO) ERASE
            MAPPINGDEV(EIBTRMID) SET(R6)
      MVC  OUTAREA(256),0(R6)
      MVC  OUTAREA+256(256),256(R6)
      EXEC CICS SEND TEXT MAPPED FROM(OUTDATA) LENGTH(OUTLEN)
*
      EXEC CICS RECEIVE INTO(INAREA) LENGTH(INLEN)
            MAXLENGTH(MAXLEN)
*
      EXEC CICS RECEIVE MAP('DFH$AGA') SET(R7) LENGTH(INLEN)
            MAPPINGDEV(EIBTRMID) FROM(INAREA)
            CURSOR(820) AID(=C'3')
*
      XC  PROOF,PROOF
      MVC PROOF(25),=C'You just keyed in number '
      MVC PROOF+25(6),KEYI-DFH$AGAI(R7)
FINISH  DS 0H
      EXEC CICS SEND TEXT FROM(PROOF) LENGTH(60) ERASE FREEKB
      TM  MSGF-DFH$AGAI(R7),X'02'
      BNO RETURN
      XC  PROOF,PROOF
      MVC PROOF(33),=C'Input cursor located in MSG field'
      EXEC CICS SEND TEXT FROM(PROOF) LENGTH(60) ERASE FREEKB
*
*      THE RETURN COMMAND ENDS THE PROGRAM.
*
RETURN  DS 0H
      EXEC CICS RETURN
*
PF3_ROUTINE DS 0H
      XC  PROOF,PROOF
      MVC PROOF(30),=C'RECEIVE MAP specified AID(PF3)'
      B   FINISH
MAXLEN  DC H'256'
APPLMSG DC C'This is a MAPPINGDEV application'
      END

```

그림 156. MAPPINGDEV 애플리케이션의 ASM 예제

파티션 지원

일부 IBM 표시장치를 사용하면 한 화면을 서로 독립된 화면인 것처럼 쓰기용 및 읽기용 영역으로 분할할 수 있습니다. 이런 영역을 파티션이라고 하며 특수한 하드웨어를 사용할 수 있도록 해주는 BMS의 기능은 총괄하여 파티션 지원이라고 합니다.

파티션에는 표준 BMS가 필요합니다.

3270 제품군의 멤버인 IBM 3290 표시장치 및 IBM 8775가 파티셔닝을 지원하는 주요 디바이스의 예입니다. 디바이스 매뉴얼(3290에 대해서는 *IBM 3290 정보 표시장치 패널 설명 및 참조* 및 *IBM 8775 표시장치 터미널 구성요소 설명*)을 참조하여 파티션된 디바이스의 전체 기능에 대해 이해하십시오. 그렇지만 주요 기능은 다음과 같습니다.

- 물리적 화면을 1-8개의 겹쳐지지 않는 직사각형 영역으로 나눠 배열할 수 있습니다. 영역은 서로 독립적입니다. 운영자는 개별적으로 해당 영역을 지우고 키보드 상태(잠금 또는 잠금 해제)도 개별적으로 유지보수되며 동시에 해당 영역에 쓰고 읽을 수도 있습니다.
- 지정된 시간에 한 개의 파티션만 활성화됩니다. 활성 파티션은 커서가 포함되는 한 개의 파티션입니다. 운영자는 이 파티션에서 키 입력만 가능하며 커서는 파티션 경계에서 줄바꿈됩니다. 데이터를 전송하는 키가 해제되면(Enter 키 또는 프로그램 기능 키 중 하나) 데이터는 활성 파티션에서만 전송됩니다.
- 1006 페이지의 『활성 파티션 판별』의 설명에 따라 운영자는 "jump" 키를 사용하여 원할 때마다 활성 파티션을 변경할 수 있으며 프로그램도 마찬가지로 변경할 수 있습니다.
- BMS도 지정된 SEND에서 한 개 파티션에만 쓸 수 있지만 사용자는 여러 개의 SEND를 발행할 수 있고 활성 파티션에 쓸 필요는 없습니다.
- 파티션 구성은 디바이스에 데이터 스트림으로 전송되어 각 새 태스크에 대해 또는 태스크 내에서 파티션을 변경할 수 있습니다. 파티션을 정의하는 BMS 구성체는 파티션 세트라고 하며 1003 페이지의 『파티션 정의』에서 설명합니다.
- 기본 상태(파티션이 없는)의 터미널을 사용할 수도 있으며 파티션 배열 변경에 사용한 동일한 명령으로 파티션된 상태에서 기본 상태로 전환할 수 있습니다.
- 화면 영역의 파티션 방법을 지정할 때 화면이 구동되는 하드웨어 버퍼 공간도 분할할 수 있습니다. 파티션된 디바이스에서 버퍼 용량은 일반적으로 화면 용량보다 크기 때문에 일부 파티션에 추가 버퍼 공간이 지정될 수 있습니다. 파티션에 할당된 화면 영역은 뷰포트라고 하며 버퍼 저장영역은 표시 공간이라고 합니다.

BMS는 표시 공간을 파티션에 대한 페이지 크기로 사용하기 때문에 표시 공간에서 수용할 수 있을 만큼, 한 번에 모두 표시되지 않더라도 원하는 만큼의 데이터를 보낼 수 있습니다. 디바이스의 키를 사용하면 운영자는 파티션의 뷰포트를 수직으로 화면이동해서 전체 표시 공간을 볼 수 있습니다. 호스트의 개입 없이도 화면 이동이 가능합니다.

- 일부 파티션된 디바이스를 사용하여 다른 크기의 문자 세트에서도 선택할 수 있습니다. 1004 페이지의 『3290 문자 크기』에서 이에 대해 자세히 설명합니다.

파티션의 독립성에도 불구하고 표시장치는 여전히 CICS에 대한 단일 터미널입니다. 여러 개의 의사 대화식 트랜잭션 순서가 동일한 파티션 세트를 사용하는 경우 이들 사이에서 화면 공간을 협력하여 사용할 수 있더라도 한 번에 둘 이상의 태스크를 해당 프린시פל 기능으로 터미널에서 사용할 수 없습니다(1009 페이지의 『터미널 공유』 참조).

참고: 3290은 내부적으로 구성할 수 있기 때문에 CICS 또는 다른 모든 시스템에 대해 둘 이상의 논리 장치로 작동합니다. 이 정의는 해당 논리 터미널 중 하나에서 발생 가능한 파티셔닝과는 별개입니다.

파티션된 화면에 대한 사용

파티션된 화면은 특히 애플리케이션의 특정 유형에 유용합니다.

화면 이동

단일 화면에 맞는 것보다 많은 출력을 생성하는 트랜잭션의 경우, 화면 이동은 BMS 터미널 페이지의 대안입니다.

예를 들어, 단 하나의 파티션으로 구성되는 파티션 세트를 정의할 수 있습니다. 여기서 뷰포트는 전체 화면이고 표시 공간은 전체 버퍼입니다. 전체 버퍼에 단일 페이지로 기록할 수 있고, 운영자는 터미널 기능을 사용하여 데이터에서 화면 이동할 수 있습니다. 호스트와의 어떤 상호작용도 없으므로, 화면 이동 요청에 대한 응답 시간이 아주 짧습니다. 단, 버퍼의 기능으로 제한됩니다.

또한 화면의 일부만 화면 이동하고 고정 데이터에 대해 일부 파티션을 사용할 수도 있습니다.

데이터 항목

파티션된 화면에 대한 또 다른 좋은 사용은 “heads down” 데이터 항목입니다. 이 때 운영자의 생산성은 애플리케이션이 입력을 처리하고 다음에 키보드를 다시 열 수 있는 속도에 따라 결정됩니다.

파티션된 화면에서, 화면을 두 개의 동일한 항목 화면으로 나눌 수 있습니다. 운영자는 화면을 채우고 Enter를 누른 후, 데이터 항목 트랜잭션이 첫 번째 입력을 처리하는 동안 두 번째 화면을 채웁니다. 입력이 제대로 되었으면, 프로그램은 다음 항목에 대한 준비 시 이 입력을 지웁니다. 그렇지 않으면 운영자가 후속 작업을 손실하지 않으면서 정정할 수 있는 기회가 계속 제공됩니다. CICS 4.1 샘플 애플리케이션 안내서에는 이와 같은 데이터 항목 트랜잭션의 예제가 있습니다.

Lookaside

많은 온라인 조작에서, 운영자는 간혹 진행 중인 트랜잭션을 완료하기 위해 두 번째 트랜잭션을 실행해야 합니다. 주문 항목은 예제입니다. 이 예제에서, 운영자는 항목을 완료하기 위해 코드 또는 가격을 찾아보아야 할 수도 있습니다.

많은 조회가 비슷합니다. 초기 조회는 다시 히트 요약 목록을 가져옵니다. 운영자는 하나를 선택하고 추가 세부사항을 요청한 후, 세부사항을 위해 다른 것을 선택해야 할 수 있습니다. 이러한 경우, 파티션된 화면을 사용하여 운영자는 화면에서 나중에 필요한 첫 번째 태스크의 출력을 유지하면서 두 번째 태스크를 수행할 수 있습니다. *CICS 4.1 샘플 애플리케이션 안내서*에도 lookaside 트랜잭션 예제가 있습니다.

“도움말” 텍스트는 여전히 “lookaside”의 다른 예제입니다. 화면의 한 파티션을 이 텍스트에 할당하는 경우, 운영자는 기본 화면을 손실하지 않고 필요한 학습 정보를 가져올 수 있습니다.

데이터 비교

운영자가 동시에 두 개 이상의 데이터 세트를 비교해야 하는 애플리케이션은 파티션된 화면의 좋은 후보입니다. 파티션을 수행하면 나란히 비교할 수 있고, 화면 이동 기능은 상대적으로 큰 문서 또는 레코드를 비교할 수 있도록 만듭니다.

오류 메시지

화면을 파티션하고 오류 메시지 및 기타 설명 텍스트에 하나의 영역을 할당하는 경우 유용성을 개선할 수 있습니다. 이를 수행하는 경우, 운영자는 항상 메시지를 찾을 위치를 알고 있으며 기본 화면 영역에는 이와 같은 정보가 겹쳐쓰여지지 않습니다.

『파티션 정의』에 설명된 것처럼, 파티션 세트에서 지정하는 경우에 CICS는 자체 고유의 메시지를 이와 같은 파티션에 보냅니다.

파티션 정의

화면의 각 파티셔닝은 화면에 함께 표시하기 위한 화면 영역(파티션) 컬렉션인 파티션 세트로 정의됩니다.

맵핑 세트를 수행하는 것과 같이 어셈블러 매크로로 파티션 세트를 정의합니다. 이 매크로 중 두 가지는 DFHPSD 및 DFHPDI입니다.

파티션 세트 정의는 다음을 정의하는 DFHPSD(파티션 세트 정의) 매크로로 시작합니다.

- 파티션 세트의 이름
- 화면 크기(BMS는 파티션 뷰포트가 사용 가능한 전체 공백을 초과하지 않는지 확인합니다.)

- 기본 문자 셀 크기(『3290 문자 크기』에 셀 크기에 대한 설명이 있습니다.)
- 파티션 세트를 특정 화면 크기와 연관시키기 위해 사용되는 파티션 세트 접미부(1005 페이지의 『파티셔닝 설정』 참조)

초기 DFHPSD 매크로 다음에, DFHPDI 매크로로 각 파티션(화면 영역)을 정의합니다. DFHPDI는 다음을 지정합니다.

- 파티션 세트 내에서 파티션 ID.
- 화면에서 파티션이 위치되는 곳.
- 해당 뷰포트 크기(행 및 열 수).
- 뷰포트와 연관되는 표시 공백(할당된 버퍼 공간의 양). 행 및 열 수로도 표시됩니다. 화면 이동은 엄밀하게 세로로 이동하므로, BMS에서는 표시 공백의 너비가 뷰포트 너비와 일치해야 합니다.
- 사용할 문자 크기.
- 파티션 크기에 적합한 맵핑 세트를 선택하기 위해 사용되는, 파티션과 연관된 맵핑 세트 접미부.
- 파티션이 CICS 오류 메시지를 수신할 수 있는지 여부(BMS는 생성하는 특정 오류 메시지를 지정된 대로 파티션(있는 경우)에 전송합니다.)

옵션 TYPE=FINAL만 포함하여, 두 번째 DFHPSD 매크로로 파티션 세트를 종료합니다. 애플리케이션 코딩에는 영향을 주지 않는 이러한 기능을 설명하는 CICS 애플리케이션 프로그래밍 참조서에서 설명됩니다.

어셈블러 매크로이므로, 작성할 때 어셈블러 형식 규칙을 따라야 합니다. 어셈블러 언어에 익숙하지 않으면 921 페이지의 『BMS 매크로 쓰기』의 내용을 참조하십시오. 작성 후 파티션 세트를 어셈블하고 링크 편집해야 합니다. 결과로 생성되는 로드 모듈은 맵핑 세트와 동일한 라이브러리나, 설치 프로그램에서 선호하는 경우 별도의 라이브러리에 위치될 수 있습니다. 시스템 스태프 역시 PARTITION 정의로 시스템에 대해 각 파티션 세트를 정의해야 합니다.

3290 문자 크기

3290 하드웨어는 최대 8개까지 다른 크기의 여러 문자 세트를 사용할 수 있도록 합니다. 두 세트는 하드웨어와 함께 제공되고, 다른 세트는 터미널 제어 SEND 명령으로 로드될 수 있습니다.

각 문자는 화면에 직사각형 셀을 점유합니다. 셀 크기는 화면, 특히 화면의 특정 파티션에 맞는 행 및 열 수를 판별합니다. 파티션별로 셀 크기를 지정할 수 있기 때문입니다. 셀은 가로 및 세로로, 화소(pel) 단위로 측정됩니다. 허용되는 가장 작은 셀은 가로 6pel x 세로 12pel입니다. 3290 화면은 높이 750pel 너비 960pel입니다. 따라서 최소

셀 크기를 사용할 경우, 세로 62자(즉, 62행) 및 가로 160자(160 열)를 맞출 수 있습니다. (3290은 항상 사용자 셀 크기에 가장 잘 맞는 문자 세트를 선택하여, 이 문자를 셀 상단 왼쪽 구석에 위치시킵니다.)

파티션 크기는 파티션에 대해 지정하는 셀 크기(pel)를 기반으로 행 및 열 수로 표시됩니다. (옵션의 이름은 CHARSIZE이지만 실제 셀 크기입니다.) 파티션이 화면에 맞는지 확인하려면, 파티션이 화면에 겹쳐지거나 화면에 맞지 않는 경우 어셈블할 시기를 BMS가 알려주어도, pel 단위로 할당을 수행해야 합니다. 파티션 높이는 파티션의 행 수와 세로 CHARSIZE 차원의 곱이고, 파티션 너비는 열 수 및 가로 CHARSIZE 값의 곱입니다.

CHARSIZE 크기를 DFHPDI 파티션 정의에 지정하지 않는 경우, BMS는 DFHPSD 파티션 세트 정의에 지정된 기본값을 사용합니다. DFHPSD가 CHARSIZE를 지정하지 않은 경우, BMS는 설치될 때 터미널에 대해 설정된 기본값을 사용합니다. 모든 파티션에 대해 지정하지 않고 일부 파티션에 대해 셀 크기를 지정하는 경우, 설치 기본값과 사용자 선택사항을 혼합하지 않도록 파티션 세트에 대해서도 기본값을 지정해야 합니다.

파티셔닝 설정

TRANSACTION 정의의 PARTITIONSET 옵션에서 이름을 지정하여 특정 트랜잭션에 대해 로드할 파티션 세트를 BMS에 알릴 수 있습니다.

이와 같이 수행하는데 이름 지정된 파티션 세트가 아직 터미널에 로드되지 않은 경우, BMS는 태스크에서 첫 번째 BMS SEND의 데이터에 파티션 정의를 추가합니다.

BMS가 파티션을 현재 상태에서 변경하지 않도록 지시하거나(PARTITIONSET=KEEP) 파티션 자체를 로드할 것을 표시할 수도 (PARTITIONSET=OWN) 있습니다. PARTITIONSET 값을 전혀 지정하지 않는 경우, BMS는 트랜잭션을 초기화할 때 터미널을 기본 상태(파티션 없음)로 설정합니다.

트랜잭션과 연관되는 PARTITIONSET 값이 무엇이든지, 태스크는 논리 메시지를 빌드하는 동안 명령을 실행할 수 없는 것을 제외하고, SEND PARTNSET 명령으로 거의 언제든지 새 파티션을 설정할 수 있습니다.

SEND PARTNSET는 어떤 것도 터미널에 즉시 전송하지 않습니다. 그 대신, 첫 번째 BMS SEND에서 TRANSACTION 정의의 PARTITIONSET 옵션에 이름 지정된 파티션 세트를 전송하는 것처럼, BMS는 데이터 또는 제어 정보를 전송하는 그 다음 BMS 명령과 함께 파티션 정보를 전송하는 것을 기억합니다. 따라서, 새 파티션에 의존하는 RECEIVE 또는 RECEIVE MAP을 실행하기 전에 SEND MAP, SEND TEXT 또는 SEND CONTROL 명령을 실행해야 합니다.

참고: 다음 상황에서 파티션이 예기치 않게 변경될 수 있습니다. CICS가 오류 메시지를 사용자 터미널에 전송해야 하고, 현재 파티션 세트에 오류 파티션이 포함되어 있지 않은 경우, CICS는 터미널을 기본 상태로 되돌리고, 화면을 지운 후 메시지를 씁니다. 이러한 이유로, 모든 파티션 세트에서 오류 메시지에 적합한 것으로 하나의 파티션을 지정하는 것이 아주 좋습니다.

BMS가 파티션 세트를 로드할 때, 디바이스 종속 지원이 적용되는 경우 터미널에 적절한 파티션 세트를 로드하기 위해 터미널 유형을 나타내는 문자를 요청된 이름 뒤에 접미부로 추가합니다. 터미널과 연관된 TYPETERM 정의의 ALTSUFFIX 옵션 값에서 접미부를 가져옵니다. 파티션 세트에 접미부를 추가하는 것은 맵핑 세트에 접미부를 추가하는 것과 유사하므로, 올바른 접미부가 있는 파티션 세트가 없는 경우 동일한 단계 순서가 사용됩니다(933 페이지의 『디바이스 종속 맵』 참조).

BMS SEND 명령에 대한 파티션 옵션

파티션된 화면에 쓸 때, 하나의 파티션에만 쓰며 명령의 영향은 해당 파티션으로 제한됩니다.

ERASE 및 ERASEAUP는 파티션 내에서만 지우며, FREEKB는 파티션이 활성화될 때만 키보드를 잠금 해제합니다.

맵 정의의 PARTN 옵션이나 SEND MAP의 OUTPARTN 옵션으로 전송 중인 파티션을 지정할 수 있습니다. OUTPARTN은 PARTN을 대체합니다. 어느 하나도 지정하지 않으면, BMS는 세트에서 첫 번째 파티션을 선택합니다.

파티션의 사용은 933 페이지의 『디바이스 종속 맵』에서 설명한 맵핑 세트 이름의 접미부에 영향을 줍니다. 맵핑 세트 접미부는 해당 절에 설명된 대로 판별되는 대신 파티션에 대한 MAPSFX 값에서 가져옵니다.

활성 파티션 판별

파티션으로 전송할 때, 커서를 해당 파티션이나 다른 파티션으로 이동할 수 있습니다. 맵핑 정의의 PARTN 옵션에서 ACTIVATE 값은 사용자가 쓰는 파티션에 커서를 놓습니다.

BMS SEND 명령에서 ACTPARTN을 지정하는 경우, 파티션(쓰고 있는 파티션일 필요는 없음)에 이름을 지정할 수 있고 ACTIVATE 스펙을 대체합니다. ACTIVATE 및 ACTPARTN은 활성 파티션에 대해 키보드를 잠금 해제하고 커서를 여기에 놓습니다. 어느 것도 존재하지 않으면, 커서는 이동하지 않으며 키보드는 잠금 해제되지 않습니다.

전송할 때 커서를 두어서 파티션을 활성화할 수 있어도 해당 주제에 마지막 단어가 없습니다. 운영자는 터미널에서 점프 키를 사용하여 커서를 다른 파티션으로 이동할 수 있기 때문입니다. 이와 같이 하면 터미널에서 다시 데이터를 수신하는 것이 복잡해 질 수 있지만 BMS는 도움말을 제공하며, 이에 대해 설명하려고 합니다.

BMS RECEIVE 명령에 대한 파티션 옵션

RECEIVE MAP 명령을 실행할 때, 맵 정의의 PARTN 옵션이나 RECEIVE MAP의 INPARTN 옵션으로 데이터를 예상하는 파티션(즉, 활성화할 것으로 예상하는 파티션)을 BMS에 알릴 수 있습니다.

INPARTN은 PARTN을 대체합니다. 사용자가 수행하고, 사용자가 이름 지정한 파티션과 다른 파티션을 운영자가 전송하는 경우, BMS는 사용자가 이름 지정한 파티션에 커서를 다시 위치시키고 키보드를 잠금 해제한 후 RECEIVE 명령을 반복합니다. 또한 운영자에게 올바른 파티션을 사용하도록 요청하는 메시지를 오류 파티션 (ATTRB=ERROR의 파티션)에 전송합니다. (오류 파티션이 없는 경우 어떤 메시지도 전송되지 않습니다.) 잘못된 파티션의 입력은 손실되지 않아도 버려집니다. 나중에 다시 읽을 수 있기 때문입니다. BMS는 세 번까지 수행합니다. 운영자가 네 번째 라운드 동안 지속되는 경우 BMS는 PARTNFAIL 상태를 발생립니다.

입력 파티션을 지정할 필요는 없습니다. 간혹 입력을 허용하는 유일한 파티션이 있으며, 간혹 동일한 맵이 모두에 적용됩니다. INPARTN 없이 RECEIVE MAP을 실행하고 맵에 PARTN 옵션이 없는 경우, BMS는 파티션의 데이터를 승인하고, 이 데이터를 명령에 이름 지정된 맵으로 맵핑합니다. INPARTN 옵션을 포함하는 ASSIGN 명령으로 이후에도 파티션을 판별할 수 있습니다(필요한 경우).

그러나 INPARTN은 첫 번째 BMS 운용 이후까지 설정되지 않으므로, 올바른 맵을 선택하기 위해 전송 중인 파티션을 알아야 하는 경우 다른 접근방식이 필요합니다. 이 상황에서, 맵핑되지 않은 데이터를 읽고 이 데이터를 전송한 파티션을 알려주는 RECEIVE PARTN 명령을 실행할 수 있습니다. 그런 다음 파티션을 일치시키는 맵을 사용하여, FROM 옵션과 파티션을 일치시키는 맵을 사용하여 RECEIVE MAP 명령을 실행합니다. FROM이 있는 RECEIVE MAP은 이미 읽은 데이터를 맵핑합니다(967 페이지의 『다른 입력 형식화』에 설명된 것처럼).

파티션에 대한 ASSIGN 옵션

INPARTN 옵션 외에도, 파티션된 터미널에 대한 프로그래밍에서 도움을 받을 수 있는 세 가지의 다른 ASSIGN 옵션이 있습니다.

PARTNS 옵션은 사용자의 태스크와 연관된 터미널이 파티션을 지원하는지 여부를 알려주고 PARTNSET 옵션은 현재 파티션 세트의 이름(설정하지 않은 경우 공백)을 리턴합니다. 네 번째 ASSIGN 옵션인 PARTNPAGE는 『파티션 및 논리 메시지』에서 자세히 설명하는 논리 메시지에만 적용됩니다.

파티션 및 논리 메시지

파티션이 설정된 터미널에 대해 BMS 논리 메시지를 빌드할 때, 메시지의 페이지를 여러 파티션으로 보낼 수 있습니다.

같은 파티션에서 혼합하지 않으면, 텍스트 출력을 일부 파티션으로 보내고 맵핑된 출력을 다른 파티션으로 보낼 수 있습니다. (이는 논리 메시지에서 텍스트와 맵핑된 출력을 혼합하는 것에 대해 보통 규칙의 예외입니다.)

출력이 표시될 때, 각 파티션의 첫 페이지가 초기에 표시됩니다. 페이지는 파티션 기준으로 번호가 매겨지며, 운영자가 특정 파티션에 입력하는 CSPG 명령은 해당 파티션에만 적용되며, 페이지 제거 명령은 예외입니다. 제거 명령은 모든 파티션에서 전체 논리 메시지를 삭제합니다.

메시지에 컨트리뷰션하는 각 BMS SEND에 대해, 출력이 이동할 파티션을 지정합니다. ACCUM을 사용하지 않는 경우, BMS는 해당 파티션에 대해 페이지를 빌드합니다. ACCUM을 사용 중이면, BMS는 출력을 해당 파티션에 대한 현재 페이지에 둡니다. 따라서 페이지 오버플로우는 파티션 기준으로 발생합니다. 오버플로우를 인터셉트 중인 데 오버플로우가 발생한 파티션이 확실하지 않은 경우, ASSIGN 명령의 PARTNPAGE 옵션을 사용하여 찾을 수 있습니다.

참고: BMS는 논리 메시지를 빌드할 때 페이지 크기와 파티션 ID 둘 다를 사용하므로, 파티션 메시지 중간을 변경할 수 없습니다.

파티션 사이에 페이지를 분배할 때 페이지 오버플로우를 처리하는데 필요한 bookkeeping은 라우팅 환경에서 필요한 것과 비슷합니다(995 페이지의 『라우팅 및 페이지 오버플로우』 참조). 특히, 다른 파티션에서 오버플로우를 야기할 수 있는 어떤 것을 수행하기 전에 하나의 파티션에 대해 오버플로우 처리를 완료하는지 확인해야 합니다. 이와 같이 하지 못하면 프로그램이 루프 상태가 되고 출력도 올바르게 않게 됩니다.

파티션 및 라우팅

여러 파티션에 쓰여진 논리 메시지를 라우팅할 수 없습니다. BMS는 라우팅 환경에서 BMS SEND 명령의 OUTPARTN 및 ACTPARTN 옵션을 무시합니다.

파티션을 지원하는 터미널에 보통의 메시지의 경로를 지정할 수 있지만, BMS는 메시지를 빌드하고 CSPG 트랜잭션은 기본(파티션되지 않은) 상태에서 터미널을 사용하여 이를 표시합니다.

또한 함께 파티션 및 논리 디바이스 코드를 사용할 수 없습니다. (LDC는 1010 페이지의 『논리 디바이스 구성요소』에서 설명됩니다.) 또한, 아웃바운드 형식과 함께 파티션을 사용할 수 있지만 GDDM과 함께 파티션을 사용할 수 없습니다(1016 페이지의 『아웃보드 형식화』 참조).

어텐션 ID 및 예외 상태

파티션된 터미널에는 CLEAR 키가 전체 화면을 지우는 것과 동일한 방법으로 활성 파티션을 지우는 CLEAR PARTITION 키가 있습니다. 사용자 프로그램 로직에서 이 추가 어텐션 ID에 대해 확인해야 할 수도 있습니다.

CLEAR PARTITION AID 값은 DFHAID에 포함됩니다(960 페이지의 『어텐션 ID 사용』 참조).

파티션과 연관되는 일부 새 예외 상태와, 이전 상태 중 일부를 가져오기 위한 새로운 방법이 있습니다. 새 예외 상태로는 INVPARTN(파티션 세트에 존재하지 않은 파티션의 이름 지정), INVPARTNSET(파티션 세트가 아닌 모듈의 이름 지정), PARTNFAIL(운영자가 전송을 시작한 파티션이 아닌 다른 파티션에서의 수신)이 있습니다. 적용되는 명령과 함께 모든 것이 참조 -> 애플리케이션 개발에서 BMS 매크로 매뉴얼에 설명되어 있습니다.

터미널 공유

각각에 별도의 파티션을 지정하여, 여러 프로세스 사이에 터미널을 공유할 수 있습니다. 터미널에서는 한 번에 여러 태스크를 진행할 수 없지만, 파티션된 터미널에서 여러 의사 대화식 트랜잭션 순서의 구성요소 태스크를 인터리브할 수 있습니다.

간단한 예제를 위해, 운영자가 두 파티션에서 데이터를 입력하도록 하여 기존 의사 대화식 데이터 항목 트랜잭션에 대한 응답 시간을 개선한다고 가정합니다(1002 페이지의 『데이터 항목』 참조). 동시에 두 개의 레코드에 대해 작동하도록 애플리케이션을 수정하거나, 해당 입력을 가져온 동일한 파티션으로 전송하도록 수정할 수 있습니다. 그럼 다음 각 파티션에서 독립적으로 실행할 수 있습니다.

TRANSACTION 정의(순서에 여러 트랜잭션이 있는 경우, 연루된 모든 트랜잭션)의 PARTITIONSET 옵션으로 파티션을 설정할 수 있습니다. 이전에 명시된 것처럼, BMS는 각 트랜잭션에 동일한 PARTITIONSET 값이 있으면 파티션을 다시 로드하지 않습니다. 또는, 예비 트랜잭션(예: 두 파티션 모두에서 첫 번째 항목 화면을 표시한 트랜잭션)으로 파티션을 설정하고 데이터 항목 트랜잭션에 대해 KEEP의 PARTITIONSET 값을 사용할 수 있습니다. 유사한 트랜잭션이나 다른 트랜잭션 사이에 파티션된 화면을 공유할 때마다, 하나의 트랜잭션이 다른 트랜잭션에 필요한 파티션 세트를 제거하지 않도록 해야 합니다. 또한 두 개의 다른 CICS 시스템이 동일한 화면을 공유할 수 있는 경우, BMS가 파티션을 다시 로드하지 않아야 할 때 다시 로드하지 못하도록 시스템은 공동으로 파티션 세트에 이름을 지정해야 합니다.

데이터 항목 트랜잭션 순서에서 다음 트랜잭션 ID를 지정하기 위해 RETURN 명령에 TRANSID 옵션을 사용하는 경우, 옵션이 파티션이 아니라 전체 터미널에 적용되기 때문에 또 다른 약간의 변경사항을 작성해야 합니다. 하나의 솔루션은 화면의 첫 번째 필드에 다음 트랜잭션 ID를 위치시키고(필드 정의에서 수정된 데이터 태그를 커서) RETURN에서 TRANSID를 제거하는 것입니다. 270 페이지의 『태스크 시작 방법』에 설명된 대로, CICS는 입력에서 다음 트랜잭션을 판별합니다.

특수 하드웨어에 대한 지원

파티션 외에도, BMS는 다른 특수 하드웨어 기능을 지원합니다.

BMS는 다음 기능을 지원합니다.

- 논리 디바이스 구성요소
- 10/63 자기 슬롯 리더
- 필드 선택 기능: 커서 선택, 라이트 펜, 트리거 필드
- 아웃보드 형식화

자기 슬롯 리더 및 아웃보드 형식화 둘 다에서는 표준 BMS가 필요합니다. 커서 선택 키, 라이트 펜 및 트리거 필드에 대한 지원은 최소에 포함됩니다.

논리 디바이스 구성요소

논리 디바이스 구성요소(LDC)는 BMS에서 지원되는 다른 특수 하드웨어 기능입니다. 파티션처럼 LDC에는 표준 BMS가 필요합니다.

LDC를 지원하는 터미널은 한 위치(논리 장치)에서 제어되는 여러 개의 작동 가능한 구성요소(논리 디바이스)로 구성되는 터미널입니다. 구성요소는 원격 작업 스테이션을 나타내는 프린터, 리더, 키보드, 표시장치이거나 워드 프로세싱 스테이션이나 통장 프린터와 같이 여러 개의 유사 디바이스일 수 있습니다. IBM 3601 논리 장치, 3770 일괄처리 논리 장치, 3770, 3790 일괄처리 데이터 상호 교환 논리 장치, LU 유형 4 논리 장치 모두 논리 디바이스 구성요소를 지원합니다.

논리 장치는 CICS에 대한 한 개의 엔티티이지만 독립적으로 읽고 쓰기가 가능한 구성요소로 구성되기 때문에 LDC 터미널에 대한 CICS API(Application Programming Interface)는 파티션된 터미널의 API와 유사하며 각 LDC는 파티션 세트에서 한 개의 파티션에 대응합니다. 그렇지만 서로 다른 점은 많기 때문에 특정 터미널 유형에 대한 CICS 지원은 CICS 매뉴얼을 참조해야 합니다. 다음의 섹션은 프로그래밍에 영향을 주는 주요 차이점에 대해 설명하며 자세한 내용은 다음과 같습니다.

- LDC 정의
- SEND 명령 옵션
- 논리 메시지
- 라우팅

논리 디바이스 구성요소 정의

터미널의 논리 디바이스 구성요소는 LDC 테이블이라고 부르는 목록으로 정의합니다. TERMINAL 정의의 TYPETERM 구성요소는 테이블을 지시하며 이는 논리 장치에 대해 개별적이거나 동일한 구성요소를 포함하는 몇 개의 논리 장치에서 공유 가능합니다.

테이블 자체는 DFHTCT TYPE=LDC(터미널 제어) 매크로로 정의됩니다. 두 매크로에 대한 설명은 *CICS* 자원 정의 안내서를 참조하십시오.

LDC 테이블에는 논리 장치의 각 논리 디바이스 구성요소에 대한 다음 정보가 포함됩니다.

- 2자의 논리 디바이스 ID입니다. 이 ID는 일반적인 표준 약어(예: 콘솔에 대해 CO, 자기 스트라이프 인코더에 대해서는 MS)이지만 꼭 약어일 필요는 없습니다.
- 1자의 디바이스 코드로 디바이스 유형(콘솔, 카드 리더, 워드 프로세싱 스테이션)을 나타냅니다. 코드는 디바이스 유형 및 매크로로 제공되는 기타 정보를 통해 CICS로 지정됩니다.
- BMS 페이지 크기입니다. 다른 논리 디바이스는 서로 다른 페이지 크기를 갖기 때문에 BMS는 논리 장치에 연관된 크기가 아니라 이 크기를 사용합니다.
- BMS 페이지 상태(AUTOPAGE 또는 NOAUTOPAGE); 971 페이지의 『AUTOPAGE 옵션』의 내용을 참조하십시오.

데이터를 논리 디바이스 구성요소로 전송

BMS 출력은 SEND MAP, SEND TEXT 또는 SEND CONTROL 명령 또는 맵핑 세트의 LDC 옵션에 이름을 지정하여 터미널의 특정 논리 디바이스 구성요소로 직접 전송합니다. 명령의 값이 맵핑 세트의 값을 덮어씁니다. LDC가 어디에도 표시되지 않으면 BMS는 터미널 유형별로 다른 기본값을 사용합니다.

LDC 및 논리 메시지

사용자 자체 터미널에 대해 BMS 논리 메시지를 빌드하면 페이지를 다른 파티션에 대한 논리 메시지로 직접 전송하는 것과 동일한 방법으로 다른 논리 디바이스 구성요소 사이에서도 메시지 페이지를 분배할 수 있습니다. BMS는 파티션에 대해 페이지를 개별적으로 누적하는 것과 동일하게 각 논리 디바이스 구성요소에 대해 페이지를 개별적으로 누적합니다(1007 페이지의 『파티션 및 논리 메시지』 참조). 텍스트와 맵핑된 출력을 한 개의 LDC로 전송하지 않는다면 메시지에 이 둘을 모두 포함할 수 있습니다. LDC가 페이지 오버플로우를 발생시키며 터미널 운영자 페이지 명령은 논리 디바이스 구성요소를 기반으로 운영됩니다.

모든 디바이스에 키보드가 있는 것은 아니기 때문에 페이지를 검색할 때 운영자(또는 디바이스 제어기의 사용자 코드)는 요청이 적용되는 LDC를 표시해야 합니다. 파티션의 경우, 메시지 영구 제거 요청은 모든 LDC에서 전체 메시지를 삭제합니다. 논리 디바이스에 대한 페이지 검색 세부사항은 *CICS* 제공된 트랜잭션의 CSPG - 페이지 검색을 참조하십시오.

페이지 오버플로우를 인터셉트하는 경우, ASSIGN 명령에 LDCMNEM 또는 LDCNUM 옵션을 지정하여 실행해서 오버플로우된 LDC를 판별할 수 있습니다. 둘 다 오버플로우된 디바이스를 식별할 수 있으며 첫 번째는 2자로된 이름을 사용하고 두 번째는 1바

이트의 숫자 ID를 사용합니다. 파티션된 디바이스를 사용하는 것처럼 ASSIGN PAGENUM로 오버플로우된 디바이스에 대한 페이지 번호를 판별할 수 있습니다.

LDC 및 LDC에 대해 고유한 페이지 오버플로우와 연관된 제한사항이 한 개가 있습니다. 오버플로우가 발생하면 현재 페이지에 대한 트레일러 맵과 오버플로우된 LDC에 대한 다음 페이지 헤더 모두를 전송해야 합니다. 실패하는 경우 BMS가 INVREQ(올바르지 않은 요청) 조건을 실행합니다.

LDC 및 라우팅

라우팅은 메시지가 LDC를 지원하는 모든 대상에 대해 동일한 구성요소 유형으로 전달되는 경우에 LDC 환경에서 지원됩니다. 다중 LDC 메시지를 라우팅할 수는 없습니다.

여러 가지 방법으로 LDC 값을 제공할 수 있습니다.

- ROUTE 명령에서 LDC 옵션을 사용하면 제공된 값이 다른 모든 소스를 덮어쓰고 LDC가 적용되는 적절한 모든 대상에 사용됩니다.
- 라우팅 목록 항목(ROUTE 명령이 아닌)에 LDC를 지정하면 해당 값이 연관된 대상에 대해 사용됩니다. (둘 다 지정하고 모두 동의하지 않으면 ROUTE 목록 값이 사용되며 불일치는 항목의 상태 플래그에 플래그로 표시됩니다.)
- 둘 다 지정하지 않으면 값은 BMS SEND 명령에서 LDC를 생략하는 경우의 라우팅 환경에서 값과 같이 터미널 및 시스템 LDC 테이블에서 판별됩니다. (SEND 명령의 값은 라우팅이 적용되는 경우 무시됩니다.)

10/63 자기 슬롯 리더

일부 IBM 표시장치 터미널은 자기 슬롯 리더(MSR)(작은 마그네틱 카드에서 데이터를 읽는 장치)를 선택적 기능으로 지원합니다. MSR에는 운영자 조치를 요청하는 프롬프트를 표시하기 위한 들을 수 있는 알람과 지시자가 있습니다. 일부 터미널은 MSR 자체를 제어하지만, IBM 8775 및 IBM 3643과 같은 다른 터미널에서는 사용자가 프로그램으로 기능을 제어할 수 있습니다.

CICS는 태스크의 프린시פל 기능에 MSR이 있는지 여부를 알려주는 ASSIGN 명령 옵션, MSR을 제공합니다.

BMS에서, BMS SEND 명령의 MSR 옵션을 사용하여 이러한 MSR의 상태를 제어할 수 있습니다. 이 옵션은 터미널에 전송된 데이터를 표시하는 것 외에, 제어 데이터의 4 바이트를 접속된 MSR로 전송합니다. BMS는 필요할 수 있는 제어 순서의 대부분을 포함하는 카피북인 DFHMSRCA를 제공합니다. *CICS 애플리케이션 프로그래밍 참조서 매뉴얼*은 제공되는 상수에 대해 설명하고 필요할 경우 목록을 확장할 수 있도록 제어 데이터의 구조를 설명합니다.

MSR로 전송하는 제어 순서는 디바이스의 다음 입력에 영향을 주므로, RECEIVE 명령이 실행될 때까지 효과가 없습니다. MSR의 입력은 디바이스 버퍼에 위치되고 키보

드 입력과 동일한 방법으로 전송됩니다. MSR 입력으로 전송이 발생하면, EIBAIID에서 어텐션 ID를 찾아보고 이를 발견할 수 있습니다. X'E6' 값은 MSR의 입력을 표시하고, X'E7'은 MSR 확장(존재할 수 있는 두 번째 MSR)의 입력에 신호를 보냅니다. MSR 입력에 대해 화면을 형식화하는 방법에 대한 정보와 이 디바이스에 대한 세부사항은 *IBM 3270 Information Display System Data Stream Programmer's Reference* 매뉴얼을 참조하십시오.

필드 선택사항 기능

BMS는 운영자가 화면에서 필드를 선택하여 입력하고 입력을 전송할 수 있도록 허용하는 여러 특수한 하드웨어 기능을 지원합니다.

다음 기능이 지원됩니다.

- 트리거 필드
- 커서 선택 가능 필드
- 라이트 펜 발견

트리거 필드 지원

트리거 필드는 8775와 같은 특정 유형의 터미널에 대한 특수 하드웨어 기능입니다. 트리거 필드로 정의된 필드는 운영자가 필드 밖으로 커서를 이동할 경우 터미널이 준비된 경우 해당 내용을 전송하도록 합니다.

필드는 운영자가 커서를 해당 필드로 이동시키거나, 데이터를 입력하거나, DELETE 또는 ERASE EOF 키를 사용할 때 준비됩니다. 필드에서 전송이 발생한 후, 또는 운영자가 ERASE INPUT 키를 사용한 경우, 또는 터미널로의 전송이 발생한 후에는 이 필드가 준비 해제됩니다. (파티션을 사용 중인 경우, 이러한 효과가 있도록 하려면 트리거 필드를 포함하는 파티션으로 전송해야 합니다.)

필드는 맵에서, 또는 프로그램 대체로, VALIDN 확장 속성을 TRIGGER 값으로 설정하여 트리거 필드로 정의합니다.

트리거 필드가 전송을 야기시킬 때 필드 자체만 전송됩니다. 다른 필드는 수정된 경우에도 전송되지 않습니다. X'7F' 값을 가지고 있는지 어텐션 ID를 확인하여 트리거 필드에 의해 야기된 전송을 발견할 수 있습니다.

호스트가 이전 전송을 처리하는 동안 운영자가 계속 데이터를 입력할 수 있도록, 유효성 검사 기능을 지원하는 터미널은 키보드를 버퍼링합니다. 운영자가 버퍼 용량을 초과하지 않거나 이전 오류 진단 이전에 많은 데이터를 입력하지 않도록, 이와 같은 입력을 처리하는 프로그램은 신속하게 반응해야 합니다.

전형적인 프로시저는 입력을 수신하는 프로그램이 트리거 필드 내용을 즉시 확인하도록 하는 것입니다. 올바르게, 프로그램은 운영자가 계속하도록 키보드 잠금을 해제합니다.

(FREEKB 옵션을 포함하는 BMS SEND 명령이 이를 수행합니다.) 필드에 오류가 있는 경우, 진단 메시지를 전송하는 것 외에도 저장된 키 스트로크를 버릴 수 있습니다. 다음 조치가 이를 수행합니다.

- ERASE, ERASEAUP 또는 ACTPARTN을 포함하거나 FREEKB를 잠그는 BMS SEND 명령
- 트리거 필드를 포함하는 파티션이 아닌 다른 파티션으로 보내지는 BMS SEND
- RECEIVE MAP, RECEIVE PARTITION 또는 터미널 제어 RECEIVE 명령
- 태스크 종료

트리거 필드에 대한 자세한 정보는 *IBM 3270 Information Display System Data Stream Programmer's Reference* 매뉴얼을 참조하십시오.

커서 및 펜 감지 가능 필드

BMS는 감지 가능 필드(일부 터미널에서 사용 가능한 다른 특수 하드웨어 기능)를 지원합니다. 감지 가능 필드에 대해 두 개의 하드웨어 메커니즘인 “커서 선택” 키와 라이트 펜이 있습니다.

터미널에는 키 또는 펜 중 하나(둘 다는 아님)가 있습니다. 둘 다 동일한 방식으로 작동하며, 키가 펜 뒤를 이었으므로, 키에 대해 설명합니다.

필드가 감지 가능하려면, 특정 필드 속성을 가지고 있어야 하며 지정 문자라고 하는 데이터의 첫 번째 문자에는 특정 값 중 하나가 포함되어야 합니다. 원하는 경우 지정 문자 다음에 다른 표시 데이터를 가질 수 있습니다.

감지 가능성을 통제하는 필드 속성 바이트의 비트 역시 밝기를 제어합니다. 고강도 (ATTRB=BRT) 필드는 지정 문자가 감지 가능 값 중 하나인 경우에 감지 가능합니다. 정상 강도 필드는 감지할 수도, 감지하지 못할 수도 있습니다. 이를 위해서는 ATTRB=DET를 지정해야 합니다. 표시되지 않는(ATTRB=DRK) 필드는 감지할 수 없습니다.

일반적으로, 맵핑 정의에서 지정하거나 프로그램 덮어쓰기로 속성 및 지정 문자를 지정할 수 있습니다. 그러나 DET는 입력 전용 맵에 나타날 때 특수 효과를 갖습니다. 이에 대해 곧 설명할 것입니다.

고강도 필드는 정의에 의해 감지 가능성에 대한 올바른 필드 속성을 가지고 있으므로, 터미널 운영자는 필드의 첫 번째 위치에 지정 문자를 키 입력하여 보호되지 않는 고강도 필드를 감지 가능하도록 만들 수 있습니다.

선택 필드

두 가지 유형의 발견 가능한 필드인 선택사항 및 어텐션 필드가 있습니다. 유형은 지정 문자에 의해 좌우됩니다.

선택 필드는 물음표(?) 또는 초과 부호(>)의 지정 문자로 정의됩니다. (?)은 운영자가 필드에 제시되는 것을 선택하지 않았음을 의미하고 (>)는 선택했음을 의미하는 것이 규칙입니다. 하드웨어는 이 규칙을 중심으로 디자인되지만, 강요되지 않으므로 다른 규칙(적합한 경우)을 사용할 수 있습니다. 지정자를 어느 하나의 값으로 초기화하고 수정된 데이터 태그를 어느 하나의 값을 사용하여 off 또는 on으로 초기화할 수 있습니다.

커서가 선택 필드에 있을 때 운영자가 커서 선택 키를 누를 때마다, 지정자는 하나의 값에서 다른 값으로 전환합니다(?는 >로 변경되고 >는 ?로 변경됩니다). 이전 상태에 관계없이 지정자가 >에서 ?로 이동할 때 MDT는 off가 되고 지정자가 ?에서 >로 이동할 때 on이 됩니다. 이로써 운영자는 (다시 필드 아래에서 커서 선택을 눌러서) 선택한 필드에 대한 자신의 생각을 변경할 수 있고 MDT의 상태에 대한 궁극적인 제어를 제공합니다. 다른 필드의 경우와 같이, MDT는 전송이 발생할 때 필드가 포함되는지 여부를 좌우합니다. 이 때 어떤 전송도 발생하지 않지만, 선택 필드는 자체적으로 전송을 야기하지 않습니다. 이는 어텐션 필드의 목적입니다.

어텐션 필드

어텐션 필드는 공백, 널(null) 또는 앰퍼샌드의 지정 문자로 정의됩니다.

BMS가 일부 상황에서 널(null)을 전송하지 않고, 필드의 첫 번째 위치를 널(null)로 대체할 수 없으므로, 데이터 스트림의 널(null)은 이 기능에서 공백과 같은 효과가 있습니다. (946 페이지의 『출력 화면 빌드』에서 값 제공 위치를 참조하십시오.) 선택 필드와는 달리, 어텐션 필드에서 커서로 커서 선택 키를 누를 때 전송이 발생합니다.

지정 문자가 앰퍼샌드이면, 커서 선택 키를 누르는 영향은 ENTER 키를 누르는 것과 같습니다. 그러나 지정자가 공백이거나 널(null)인 경우 전송되는 것은 MDT가 켜져 있는 모든 필드와 주소와, 커서의 위치, 그리고 X'7E'의 어텐션 ID입니다. 이 필드의 내용은 전송되지 않습니다. ENTER 키(또는 앰퍼샌드 지정자와 함께 커서 선택)를 사용하기 때문입니다. 어느 경우든지, MDT 비트가 켜져 있는 필드는 운영자가 변경했거나 MDT가 켜진 상태로 전송된 보통의 필드 또는 선택 필드가 될 수 있습니다.

발견할 수 있는 필드로부터 BMS 입력

공백이나 널 지정자가 있는 커서 선택 어텐션 필드에 의해 초래된 전송 후에 BMS는 해당하는 입력(I) 서브필드를 X'FF'로 설정하여 어떤 필드가 전송되었는지(즉, MDT가 켜져 있음)를 알려줍니다. 그렇지 않으면 첫 번째 위치는 X'00'으로 설정됩니다.

주의 필드가 단 한 번만 전송된 경우에는 이 값으로부터 또는 그렇지 않은 경우 커서의 위치로부터 어떤 어텐션 필드가 전송을 초래했는지를 알 수 있습니다.

전송이 앰퍼샌드 지정자가 있는 커서 선택 어텐션 필드에 의해 초래되었거나 ENTER 키 또는 함수 키에 의해 초래된 경우에는 I 서브필드에는 MDT가 켜져 있는 경우 필드의 내용을 포함하고 L 서브필드는 일반적으로 해당 길이를 반영합니다. DET 속성이 입력 맵의 필드에 지정된 경우에는 예외입니다(즉, MODE=IN 또는 MODE=INOUT,

DATA=FIELD). 이러한 맵을 명명하는 RECEIVE MAP 뒤에 이 I 서브필드에는 필드가 선택된 경우(즉, MDT가 켜짐) 길이가 1인 X'FF'를 포함하고 아닌 경우 널(null)(X'00')을 포함합니다. BMS는 일부가 전송되었더라도 필드에 다른 입력을 제공하지 않습니다.

결과적으로 발견 가능한 필드로부터 데이터를 받아야 하고 선택되었는지 여부를 알아는 경우에는 입력 맵에서 DET의 사용을 피해야 합니다. 출력과 입력에 별개의 맵을 사용하고 DET 속성은 출력 맵에만 지정하거나 또는 맵에서가 아니라 프로그램이 전송한 데이터 스트림에 DET 속성을 설정할 수 있습니다. 높은 강도 필드의 경우에는 BRT가 DET를 암시하므로 DET를 지정할 필요가 없습니다. BMS는 입력 맵에서 BRT로 지정된 필드의 데이터를 리턴합니다.

또한 데이터가 전송되는지를 확실히 해야 합니다. 전송의 원인이 ENTER 키, 기능 키 또는 앰퍼샌드 지정자 문자가 있는 어텐션 필드이면 필드 데이터가 전송됩니다. 원인이 공백 또는 널(null) 지정자가 있는 어텐션 필드이면 필드 데이터가 전송되지 않습니다.

발견 가능한 필드에 대한 자세한 정보는 *IBM 3270 정보 표시 시스템 데이터 스트림 프로그래머의 참조 매뉴얼*을 참조하십시오.

아웃보드 형식화

아웃보드 형식화는 호스트 프로세서와 접속된 서브시스템 사이의 라인 트래픽 양을 감소하기 위한 기술입니다. 감소는 네트워크에서 변수 데이터만 전송하는 것으로 이뤄집니다. 이 데이터는 서브시스템 내의 프로그램에 의해 상수 데이터(예: 물리적 맵핑)와 결합됩니다. 형식화된 데이터가 표시될 수 있습니다.

3650 호스트 통신 논리 장치, DPPX 및 DPS 버전 2의 8100 Series 프로세서, 또는 3174 제어 장치를 통해 접속된 터미널에 대해 아웃보드 형식화를 사용할 수 있습니다. 3650이 사용하는 맵은 사용되기 전에 3650 변환 정의 언어를 사용하여 다시 정의해야 합니다. 8100에 대해 사용되는 맵은 SDF II 유틸리티나 DPS 버전 2의 대화식 맵핑 정의 구성요소를 사용하여 8100에서 생성되어야 합니다.

호스트 프로세서의 프로그램이 많은 맵핑된 데이터를 서브시스템에 전송하면, 맵에서 변수 데이터만 전송하도록 BMS에 지시하여 트래픽을 감소할 수 있습니다. 그러면 서브시스템이 데이터를 수신할 때 맵핑 운영을 수행해야 합니다. BMS는 데이터를 형식화하기 위해 사용될 서브시스템 맵을 식별하는 정보로 변수 데이터 앞에 접두부를 붙입니다.

아웃보드 형식화를 지원하는 터미널은 해당되는 TYPETERM 정의에 OBFORMAT(YES)를 가지고 있습니다. 프로그램이 이와 같은 터미널에 대해 SEND MAP 명령을 실행하고, 지정된 맵 정의에 OBFMT=YES가 포함된 경우, BMS는 서브시스템이 데이터를 형식화하고 적절한 데이터 스트림을 생성할 것으로 가정합니다.

OBFMT=YES가 있는 맵을 아웃보드 형식화가 지원되지 않는 터미널로 전송하는 경우, BMS는 OBFMT 피연산자를 무시합니다.

아웃보드 형식화를 지원하는 일부 디바이스의 프로그래밍에 대한 자세한 정보는 751 페이지의 제 32 장 『일괄처리 데이터 상호 교환 사용』의 내용을 참조하십시오.

BMS: 성능 향상을 위한 디자인

BMS(Basic Mapping Support)를 사용하여 형식화된 데이터 스트림을 빌드하는 경우 여기 설명된 요인을 명심해야 합니다.

불필요하게 수정된 데이터 태그(MDT) 설정 방지

MDT는 READ MODIFIED 명령(복사 작업을 제외한 모든 작업에 CICS가 사용하는 명령)에서 필드가 전송되어야 하는지 여부를 판별하는 속성 바이트의 비트입니다.

필드에 대한 MDT는 일반적으로 사용자가 필드에 데이터를 입력하면 3270 하드웨어에서 설정됩니다. 그러나 맵에서 FSET를 지정하거나 태그가 설정된 덮어쓰기 속성 바이트를 전송하여 화면으로 맵을 전송할 때도 태그를 설정할 수 있습니다. 맵의 상수 필드에 대해 또는 레이블이 없고 맵을 수신하는 프로그램으로 전송되지 않는 필드에 대해 이런 방법으로 태그를 설정해서는 안 됩니다.

또한 일반적으로 일반 입력 필드에는 FSET를 지정하지 않아도 됩니다. 그 이유는 이미 언급한 대로 사용자가 데이터를 입력하는 필드에서 MDT가 자동으로 설정되기 때문입니다. 그러면 다음 RECEIVE 명령에 포함됩니다. 화면이 몇 번 전송되면 FRSET, ERASEAUP 또는 ERASE 옵션을 통해 또는 태그가 해제된 덮어쓰기 속성을 사용하여 프로그램이 명시적으로 설정 해제할 때까지 이러한 태그는 설정 상태를 유지합니다.

사용자가 화면에 입력하지 않은 정보를 입력 간에 저장할 수 있습니다. 이는 프로그램을 통해 MDT를 설정하기 위한 의도적인 이유입니다. 그러나 이 저장 기술은 적은 양의 데이터에만 적합하며, 전송 오버헤드가 수반되기 때문에 원격 터미널의 경우보다 로컬에 더 적합합니다. 예를 들어, 이 기술은 입력 필드에 대한 기본값을 저장하는 데 특히 유용합니다. 일부 애플리케이션에서는 사용자가 일부 필드에 기본값이 이미 포함된 화면을 완료해야 합니다. 기본값 변경을 원하지 않는 사용자는 해당 필드를 건너뛰면 됩니다. 입력을 처리하는 프로그램에 이러한 기본값이 무엇인지 알려야 합니다. 기본값이 항상 같으면 프로그램에서 상수로 제공할 수 있습니다. 그러나 가변적이고 이전 입력에 따라 달라지는 경우에는 화면을 작성하는 맵에서 FSET를 사용하여 MDT를 설정하여 화면에 저장할 수 있습니다. 그러면 화면을 읽는 프로그램이 필드를 변경하지 않는 사용자로부터는 기본값을 수신하고 필드를 변경하는 사용자로부터는 새 값을 수신합니다.

참고: CLEAR, PA1, PA2 또는 PA3 키를 누르면 저장된 값이 화면에 리턴됩니다.

FRSET를 사용하여 인바운드 트래픽 축소

입력 필드가 많은 화면이 있는 경우(여러 번 읽어야 함), 다음 읽기를 준비하여 화면에 다시 쓸 때 FRSET를 지정하여 입력 데이터 스트림의 길이를 줄일 수 있습니다. FRSET는 MDT를 설정 해제하므로 사용자가 다음 번에 다시 입력하지 않는 한 해당 쓰기 이전에 입력된 필드는 없습니다. 오류 주기가 많거나 다른 어떤 이유로 반복 전송이 있는 프로세스와 상대적으로 전체 화면을 처리하는 경우 이렇게 하면 실질적으로 절약할 수 있습니다. 그러나 후속 읽기에서는 **변경된 필드만** 전송되기 때문에 프로그램은 각 주기의 입력을 저장하고 새 데이터를 기존 데이터와 병합해야 합니다. MDT가 설정 상태를 유지하기 때문에 FRSET를 사용하지 않는 경우에는 이렇게 하지 않아도 되며, 입력되는 시기와 관계없이 모든 필드가 전송됩니다.

화면으로 공백 필드를 전송하지 않음

전체 공백으로 구성되거나 뒤 공백으로 오른쪽을 채운 화면으로 필드를 전송하는 것은 일반적으로 회선 용량을 낭비하는 것입니다. BMS에서 이렇게 하도록 요구하는 유일한 경우는 화면에서 현재 데이터가 포함된 필드를 지우거나 나머지 화면을 변경하지 않고 화면에 현재 있는 것보다 짧은 데이터로 필드를 바꾸어야 하는 경우입니다.

이는 BMS가 맵을 나타내는 데이터 스트림을 빌드할 때 공백(X'40')은 포함하지만 널(null)(X'00')은 생략하기 때문입니다. 따라서 출력 데이터 스트림이 짧아집니다. 필드의 후속 문자에 관계없이 BMS는 첫 번째 데이터 문자가 널(null)인 필드를 생략합니다.

BMS는 맵을 빌드하는 데 사용할 영역을 널(null)로 초기화할 것을 요구합니다. 널(null)(X'00')을 기호 맵 구조의 mapnameO 필드로 이동하여 이를 수행합니다. 자세한 정보는 939 페이지의 『출력 맵 초기화』의 내용을 참조하십시오. BMS는 속성 위치와 데이터의 첫 번째 위치에서 널(null)을 사용하여 맵에서 값에 대해 변경이 수행되지 않음을 나타냅니다. 프로그램 또는 TIOA에서 맵 영역을 재사용하는 경우 이런 방법으로 지우려면 특별한 주의를 기울여야 합니다.

올바로 CICS 영역 주소 지정

CICS 영역이 올바로 주소 지정되었는지 확인하는 몇 가지 방법이 있습니다. 다음을 확인하십시오.

- 4KB를 초과하는 LINKAGE SECTION 구조를 가진 각 COBOL 프로그램에 필수 정의와 두 개 이상의 인접 BLL 셀 설정이 있습니다.
- 모든 BLL 포인터가 01 레벨 항목인 영역을 가리킵니다.
- 호출 레벨 DL/I는 올바로 주소 지정된 PSB에만 사용됩니다.

가능하면 MAPONLY 옵션 사용

MAPONLY 옵션은 맵의 상수 데이터만 전송하고 프로그램의 가변 데이터는 병합하지 않습니다. 결과 데이터 스트림이 항상 짧은 것은 아니지만 조작성은 BMS에서 경로 길이

가 짧아집니다. 데이터 입력에 사용할 구조 화면을 전송할 때 종종 MAPONLY를 사용할 수 있습니다.

변경된 필드만 기존 화면으로 전송

예를 들어, 화면에 메시지가 추가되거나 입력 화면에서 두 개 이상의 필드가 강조표시되어 오류를 표시하는 경우 변경된 필드만 전송하는 것이 중요합니다. 이러한 상황에서는 DATAONLY 옵션을 사용하여 변경된 필드를 제외하고 널(null)로 구성된 맵을 전송해야 합니다. 속성 바이트만 변경된 필드의 경우, 해당 바이트만 전송하고 나머지 바이트는 널(null)로 전송해야 합니다. BMS는 이 입력을 사용하여 해당 필드만으로 구성된 데이터 스트림을 빌드하며, 화면의 다른 모든 필드는 변경되지 않은 채로 남아 있습니다.

이 조언을 무시하고 불필요하게 긴 데이터 스트림을 전송하기 쉽습니다. 예를 들어, 오류가 있는지 입력 화면을 검사하는 프로그램이 오류를 찾을 경우 두 개의 옵션이 있습니다.

- 입력 맵에 오류 정보(강조표시된 속성, 오류 메시지 등)를 추가하고 다시 전송할 수 있습니다.
- 오류와 메시지 필드만으로 구성된 전체 새 화면을 빌드할 수 있습니다.

전자는 코드화하기가 약간 쉽지만(두 개의 맵 영역을 갖거나 필드를 이동하지 않아도 됨) 출력 데이터 스트림에 올바른 입력 필드는 물론 오류와 메시지 필드도 포함되기 때문에 전송 시간이 훨씬 길어질 수 있습니다. 실제로, 입력에 비어 있거나 짧은 필드가 있으면 BMS가 누락된 문자를 공백이나 0으로 대체했을 수 있기 때문에 원래 입력 스트림보다 길어질 수 있습니다.

3270 하드웨어를 사용하는 경우, 터미널의 입력 스트림이 256바이트를 초과하면 터미널 제어 장치가 최대값인 256바이트인 개별 전송으로 자동 분할합니다. 이는 긴 입력 스트림에는 몇 개의 물리적 I/O 작업이 필요함을 의미합니다. 이는 애플리케이션에 투명하기는 하지만 추가 회선 및 프로세서 오버헤드를 야기합니다. 출력 스트림은 일반적으로 단일 전송으로 전송됩니다.

회선 트래픽을 줄이기 위한 데이터 입력 작업 디자인

사용자가 동일한 화면을 여러 번 완료해야 하는 경우가 자주 있습니다. 각 주기에서 데이터만 변경됩니다. 제목, 필드 레이블, 지시사항 등은 변경되지 않은 채로 남아 있습니다. 이런 상황에서는 입력이 허용되어 처리되면 SEND CONTROL ERASEAUP 명령(또는 짧은 확인 메시지만 포함하고 ERASEAUP 옵션을 지정하는 맵)으로 응답할 수 있습니다. 이렇게 하면 화면의 모든 비보호 필드(즉, 마지막 입력의 모든 입력 데이터)가 지워지고 MDT가 다시 설정됩니다. 보호 필드에 있는 레이블과 기타 텍스트는 변경되지 않고 화면은 다음 데이터 입력 주기를 위해 준비되며, 필요한 데이터만 전송되었습니다.

화면으로 전송된 데이터 압축

형식화되지 않은 데이터를 화면으로 전송하거나 BMS 외부에서 형식화된 화면을 작성하는 경우, 버퍼 주소 설정(SBA) 및 주소까지 반복(RA) 순서를 데이터 스트림에 삽입하여 데이터를 추가로 압축할 수 있습니다. SBA를 사용하면 화면에 데이터를 배치할 수 있으며, RA는 뒤에 오는 문자가 지정된 종료 주소까지 버퍼의 현재 위치에서 생성되게 합니다. SBA는 화면에서 뒤에 데이터가 오는 사용하지 않는 영역이 상당히 있을 때마다 유용합니다. RA는 화면에 긴 순서의 동일한 문자(예: 공백 또는 대시)가 있는 경우 유용합니다. 그러나 RA가 처리하는 속도는 3270 제어 장치의 모든 모델에서 균등하지 않음에 유의해야 합니다. 사용 전에 구성에 어떻게 적용되는지 확인해야 합니다.

CICS는 터미널로 출력이 전송되기 직전에 구동되는 엑시트를 제공합니다(XTC OUT). 이 엑시트에 SBA 및 RA 대체를 추가하여 일반 서브루틴을 사용하여 데이터 스트림을 압축할 수 있습니다. 여기에는 애플리케이션에서 압축 논리를 제거한다는 것과 BMS가 생성하는지 여부에 관계없이 모든 출력 데이터 스트림에 적용된다는 이중의 장점이 있습니다.

공백 대신 널(null) 사용

BMS 외부에서 널(null)은 출력 데이터 스트림에서 특별한 의미를 갖지 않는다는 점에 유의해야 합니다. 화면에서 공백 영역이 필요한 경우 공백이나 널(null)을 전송할 수 있습니다. 즉, 출력 스트림에서 동일한 공간을 차지합니다. 그러나 사용자가 공백 필드를 변경하고 나중에 읽을 가능성이 있는 경우에는 다시 전송되지 않으므로 널(null)을 사용하십시오.

널(null) 또는 공백이 필요하지 않게 하는 방법 사용

공백이어야 하는 화면의 큰 영역의 경우, 공백 또는 널(null)을 전송하지 말고 다른 방법을 고려해야 합니다(예: BMS 사용, 데이터 스트림에 SBA 및 RA 순서 직접 배치, ERASE 및 ERASEAUP 옵션 사용).

페이지 빌드 및 라우팅 작업

BMS 페이지 빌드 기능은 긴 메시지를 빌드 및 표시하고 다중 대상으로 메시지를 전송하며, 물리적 특성이 다른 여러 개의 디바이스에 맞게 단일 메시지를 형식화하기 위한 강력하면서도 유연한 도구를 제공합니다. 그러나 고기능 도구의 경우처럼 실질적인 오버헤드를 필요로 합니다.

자세한 정보는 716 페이지의 『유효 데이터 세트 운영』의 내용을 참조하십시오. 다음 경우 페이지 빌드 옵션(ACCUM)이 필요합니다.

- 길이가 출력 디바이스 용량을 초과하는 메시지 전송(멀티페이지 출력)
- 입력 터미널 이외의 다른 대상 사용

- 다중 맵에서 빌드된 페이지 전송
- BMS 페이지 복사 기능 사용

멀티페이지 출력 전송

많은 화면 크기 페이지로 구성된 대형 출력 메시지를 생성하는 트랜잭션은 시스템 자원을 많이 사용하는 경향이 있습니다. 먼저, 프로세서 활동, CSPG 트랜잭션 실행, 데이터 세트 I/O 활동을 수반하는 모든 페이지를 작성해야 합니다. 그리고 나서 임시 저장 영역에 페이지를 저장해야 합니다. 터미널 사용자가 메시지의 모든 페이지를 살펴보면, 많은 트랜잭션이 실행되어 페이지 요청을 처리합니다(각각 회선 및 프로세서 오버헤드가 필요함). 명백히 일부 오버헤드는 트랜잭션의 크기와 복잡도로 인해 발생하며 피할 수 없습니다. 실제로 몇 명의 사용자가 동시에 페이지 출력을 급속히 화면이동하는 경우에는 필요한 트랜잭션이 시스템을 독점할 수 있습니다.

사용자가 실제로 모든 페이지를 보아야 하고 앞뒤로 자주 화면이동해야 하는 경우에는 모든 페이지를 동시에 생성하고 『기존의』 CICS 페이지 서비스를 사용하여 제공하는 것이 훨씬 효율적입니다. 그러나 사용자가 몇 개의 페이지만 필요로 하거나 메시지에서 앞뒤로 얼마나 화면이동하려는지 쉽게 지정할 수 있는 경우에는 다음 두 가지 선택사항이 있습니다.

1. 먼저, 의사 대화식 트랜잭션을 구성하여 하나의 출력 화면만 생성하십시오. 이 트랜잭션이 처음 실행될 때 여러 페이지 출력의 첫 번째 페이지를 생성합니다. 출력 화면은 사용자가 다음에 원하는 페이지를 표시할 수 있도록 공간을 포함합니다. 트랜잭션은 항상 자신을 가리키도록 다음 트랜잭션 ID를 설정하므로 다음에 실행될 때 요청된 페이지를 표시할 수 있습니다.

사용자에게 CICS가 제공하는 일부 옵션(예: 한 페이지 앞으로, 한 페이지 뒤로, 선택된 페이지로 건너뛰기)과 애플리케이션 관련 일부 옵션(예: 출력의 다음 페이지를 시작할 데이터 세트 키)을 제공할 수 있습니다.

2. 대안은 ACCUM 옵션을 사용하여 멀티페이지 출력 메시지를 페이지 빌드하되, 메시지의 페이지 수를 제한하는 것입니다(예: 5개). 사용자는 일반 CICS 명령을 사용하여 서브세트 페이지를 페이지 이동합니다. 출력의 마지막 화면에서 추가 출력이 있다는 표시와 사용자가 다음 세그먼트를 볼 것인지 여부를 표시하기 위한 위치를 추가합니다. 첫 번째 예제에서와 마찬가지로 다음 트랜잭션 ID는 원래 트랜잭션으로 설정되므로 CICS가 페이지 명령을 수신하지 않으면 해당 트랜잭션을 호출합니다.

입력 터미널 이외의 다른 대상으로 메시지 전송

태스크와 연관된 입력 터미널이 아닌 다른 터미널로 메시지를 보내야 하는 경우 BMS 라우팅은 이를 수행하는 가장 효율적인 방법입니다. 이는 메시지를 다중 대상으로 전송해야 하거나 메시지에 다중 페이지가 포함된 경우 특히 그렇습니다. 라우팅은 메시지 수신인이 메시지에 액세스하기 위한 CICS 페이지 명령을 필요로 하는 경우 권장되는 방법입니다.

그러나 이전 조건이 어느 것도 적용되지 않는 경우에는 트랜잭션과 연관되지 않은 터미널로 출력을 전달하는 다른 두 가지 방법 중에서 선택할 수 있습니다.

1. START 명령을 TERMID 옵션과 함께 사용하여 기록할 터미널을 지정하고 FROM 옵션과 함께 사용하여 전송할 데이터를 지정할 수 있습니다. 자체 트랜잭션은 시작된 트랜잭션입니다. 메시지에 대해 RETRIEVE 명령을 실행한 후 자체 터미널로 전송합니다. START 명령에 대한 프로그래밍 정보는 참조 -> 애플리케이션 개발에서 START의 내용을 참조하십시오.
2. 마찬가지로, 특정 터미널로 대상 지정된 메시지를 리전 내부 트랜지언트 데이터 큐에 출력할 수 있습니다. 다음과 같이 트랜지언트 데이터 큐에 대한 정의를 지정해야 합니다.
 - TERMINAL로서의 대상
 - 터미널 ID
 - 트리거 레벨
 - 트랜잭션 이름

자체 트랜잭션은 트랜지언트 데이터 큐를 읽고 터미널로 메시지를 전송합니다. 큐가 빌 때까지 이 순서를 반복한 후 종료됩니다. 지정한 트리거 레벨은 지정된 수의 메시지가 큐에 배치될 때마다 호출됨을 의미합니다.

참고: 어떤 방법을 통해서든 라우팅 메시지와 연관된 오버헤드로 인해 ROUTE=ALL과 같은 기능을 주의하여 사용해야 합니다.

다중 맵에서 빌드된 페이지 전송

여러 맵을 사용하여 점차적으로 화면을 쉽게 빌드할 수 있긴 하지만 때로는 페이지 빌드 작업을 사용하지 않음으로써 상당한 오버헤드를 피할 수 있습니다(특히, 하나의 출력 화면만 있고 페이지링은 필요하지 않은 경우). 이에 대한 예제는 출력이 헤더 맵 뒤에 두 번째 맵과 함께 전송되는 가변수의 세부사항 세그먼트가 오고 마지막으로 세부사항 뒤에 트레일러 맵이 오는 애플리케이션입니다. 이러한 애플리케이션에 대한 평균 출력 화면이 8개의 (2행) 세부사항 세그먼트와 헤더 및 트레일러를 포함하며, 이 모두가 단일 화면에 맞는다고 가정하십시오. 페이지 빌드를 사용하여 이 화면을 작성하려면 11개의 BMS 호출(헤더, 세부사항, 트레일러, 페이지아웃)이 필요한 반면, 프로그램이 출력 화면을 내부적으로 빌드하는 경우에는 하나의 호출만 필요합니다.

BMS 페이지 복사 기능 사용

누적된 BMS 메시지를 구성하는 개별 메시지가 임시 저장영역에 저장되기 때문에 BMS를 사용하여 터미널 사용자는 개별 페이지를 다른 터미널에 복사할 수 있습니다. 그러나 복사 기능이 페이지 빌드를 사용하기 위한 유일한 이유인 경우에는 3274 제어 장치 복사 기능 또는 CICS 복사 키 기능을 대신 사용하도록 고려해야 합니다.

3274 복사 기능은 CICS의 참여와 전송을 필요로 하지 않으며 지금까지 가장 효율적인 방법입니다. BMS 복사 기능의 다른 유형이긴 하지만 CICS 복사 키 기능은 오버헤드가 있습니다(878 페이지의 『인쇄된 출력에 대한 요청』 참조). BMS 복사에 적용되지 않는 대상 제한사항도 있습니다.

제 16 부 부록

주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다. IBM은 다른 국가에서 이 책에 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 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 그러나 비IBM 제품, 프로그램 또는 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 라이선스까지 부여하는 것은 아닙니다. 라이선스에 대한 의문사항은 다음으로 문의하십시오.

150-945

서울특별시 영등포구 국제금융로 10, 3IFC,

한국 아이.비.엠 주식회사

대표전화서비스: 02-3781-7114

2바이트(DBCS) 정보에 관한 라이선스 문의는 한국 IBM 고객만족센터에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

Intellectual Property Licensing

Legal and Intellectual Property Law

IBM Japan, Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, 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 보증 없이 이 책을 "현상태대로" 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 책에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 변경된 사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통지 없이 언제든지 개선 및/또는 변경할 수 있습니다.

- (i) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및
- (ii) 교환된 정보의 상호 이용을 목적으로 정보를 원하는 프로그램 라이선스 사용자는 다음 주소로 문의하십시오.

150-945

서울특별시 영등포구 국제금융로 10, 3IFC,

한국 아이.비.엠 주식회사

대표전화서비스: 02-3781-7114

이러한 정보는 해당 조건(예를 들면, 사용료 지불 등)하에서 사용될 수 있습니다.

이 정보에 기술된 라이선스가 부여된 프로그램 및 프로그램에 대해 사용 가능한 모든 라이선스가 부여된 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이선스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

개인정보 보호정책 고려사항

SaaS(Software as a Service) 솔루션을 포함한 IBM 소프트웨어 제품(이하 "소프트웨어 오퍼링")은 제품 사용 정보를 수집하거나 일반 사용자의 사용 경험을 개선하거나 일반 사용자와의 상호 작용을 조정하거나 그 외의 용도로 쿠키나 기타 다른 기술을 사용할 수 있습니다. 대부분의 경우 소프트웨어 오퍼링은 개인 식별 정보를 수집하지 않습니다. IBM 소프트웨어 오퍼링 중 일부는 개인 식별 정보를 수집할 수 있도록 도와 줍니다. 이 소프트웨어 오퍼링이 쿠키를 사용하여 개인 식별 정보를 수집하는 경우, 이 오퍼링의 쿠키 사용에 대한 자세한 정보는 다음과 같습니다.

CICSplex SM 웹 사용자 인터페이스:

WUI 기본 인터페이스: 전개된 구성에 따라 이 소프트웨어 오퍼링은 세션 관리, 인증, 사용자 용이성 향상 또는 기타 사용법 추적이나 기능을 위해 각 사용자의 사용자 이름 및 개인 식별 정보를 수집하는 세션 및 지속적 쿠키를 사용할 수 있습니다. 이러한 쿠키는 사용 안함으로 설정할 수 없습니다.

WUI 데이터 인터페이스: 전개된 구성에 따라 이 소프트웨어 오퍼링은 세션 관리, 인증 또는 기타 사용법 추적이나 기능을 위해 각 사용자의 사용자 이름 및 개인 식별 정보를 수집하는 세션 쿠키를 사용할 수 있습니다. 이러한 쿠키는 사용 안함으로 설정할 수 없습니다.

WUI Hello World 페이지: 전개된 구성에 따라 이 소프트웨어 오퍼링은 개인 식별 정보를 수집하지 않는 세션 쿠키를 사용할 수 있습니다. 이러한 쿠키는 사용 안함으로 설정할 수 없습니다.

CICS 탐색기: 전개된 구성에 따라 이 소프트웨어 오퍼링은 세션 관리, 인증 및 싱글 사인온 구성을 위해 사용자의 사용자 이름 및 비밀번호를 수집하는 세션 및 지속적 환경 설정을 사용할 수 있습니다. 사인온 중에 선택란을 선택하여 사용자 비밀번호를 암

호화된 양식으로 저장하면 사용자의 명시적인 조치에 의해서만 사용으로 설정할 수 있지만 이러한 환경 설정은 사용 안함으로 설정할 수 없습니다.

이 소프트웨어 오퍼링을 위해 전개되는 구성이 귀하(고객)에게 쿠키 등의 기술로 일반 사용자의 개인 식별 정보를 수집하는 기능을 제공하는 경우, 통지 및 동의에 필요한 모든 요구사항을 포함하여 그러한 데이터 수집에 적용되는 모든 법률에 대해 직접 법적 자문을 구해야 합니다.

이러한 목적을 위해 쿠키를 비롯한 여러 기술을 사용하는 데 대한 자세한 정보는 IBM의 개인정보 보호정책(<http://www.ibm.com/privacy/kr/ko/>), IBM의 온라인 개인정보 보호정책(<http://www.ibm.com/privacy/details/kr/ko/>) 『쿠키, 웹 비콘 및 기타 기술』절과 『IBM 소프트웨어 제품 및 SaaS(Software-as-a-Service) 개인정보 보호정책』(<http://www-01.ibm.com/software/info/product-privacy/>)을 참조하십시오.

상표

IBM, IBM 로고 및 [ibm.com](http://www.ibm.com)[®]은 전세계 여러 국가에 등록된 International Business Machines Corp.의 상표 또는 등록상표입니다. 기타 제품 및 서비스 이름은 IBM 또는 타사의 상표입니다. IBM 상표의 최신 목록은 웹 사이트의 저작권 및 상표 정보 (www.ibm.com/legal/copytrade.shtml)에서 확인할 수 있습니다.

Java 및 모든 Java 기반 상표와 로고는 Oracle 및/또는 그 계열사의 상표 또는 등록상표입니다.

UNIX는 미국 또는 기타 국가에서 사용되는 Open Group의 등록상표입니다.

참고 문헌

z/OS용 CICS Transaction Server의 CICS 서적

일반

z/OS용 CICS Transaction Server 프로그램 디렉토리 - 기본, GI13-3375
z/OS용 CICS Transaction Server 프로그램 디렉토리 활성화 모듈 - 기본, GI13-3376
z/OS용 CICS Transaction Server 프로그램 디렉토리 활성화 모듈 - 개발자 평가판, GI13-3377
z/OS용 CICS Transaction Server 프로그램 디렉토리 활성화 모듈 - Value Unit Edition, GI13-3378
z/OS용 CICS Transaction Server 새로운 기능, GC34-7437
z/OS용 CICS Transaction Server CICS TS 버전 5.3으로 업그레이드, GC34-7436
z/OS용 CICS Transaction Server 설치 안내서, GC34-7414

CICS 액세스

CICS 인터넷 안내서, SC34-7416
CICS 웹 서비스 안내서, SC34-7452

관리

CICS 시스템 정의 안내서, SC34-7428
CICS 사용자 정의 안내서, SC34-7404
CICS 자원 정의 안내서, SC34-7425
CICS 운영 및 유틸리티 안내서, SC34-7420
CICS RACF 보안 안내서, SC34-7423
CICS 제공된 트랜잭션, SC34-7427

프로그래밍

CICS 애플리케이션 프로그래밍 안내서, SC43-2758
CICS 애플리케이션 프로그래밍 참조서, SC34-7402
CICS 시스템 프로그래밍 참조서, SC34-7429
CICS FEPI(Front End Programming Interface) 사용자 안내서, SC34-7412
CICS C++ OO 클래스 라이브러리, SC34-7405
CICS 분산 트랜잭션 프로그래밍 안내서, SC34-7410
CICS 비즈니스 트랜잭션 서비스, SC34-7403
CICS의 Java 애플리케이션, SC34-7417

진단

CICS 문제점 판별 안내서, GC34-7422
CICS 성능 안내서, SC34-7421
CICS 메시지 및 코드 제 1권, GC34-7418
CICS 메시지 및 코드 제 2권, GC34-7419
CICS 진단 참조서, GC34-7409
CICS 복구 및 다시 시작 안내서, SC34-7424
CICS 데이터 영역, GC34-7406
CICS 추적 항목, SC34-7430
CICS 디버깅 도구 인터페이스 참조서, GC34-7408

통신

CICS 상호 통신 안내서, SC34-7415
CICS 외부 인터페이스 안내서, SC34-7411

데이터베이스

CICS DB2 안내서, SC34-7407
CICS IMS 데이터베이스 제어 안내서, SC34-7413
CICS 공유 데이터 테이블 안내서, SC34-7426

z/OS용 CICS Transaction Server에 대한 CICSplex SM 서적

일반

CICSplex SM 개념 및 계획, SC34-7441
CICSplex SM 웹 사용자 인터페이스 안내서, SC34-7451

관리(Administration and Management)

CICSplex SM 관리, SC34-7438
CICSplex SM 운영 보기 참조서, SC34-7447
CICSplex SM 모니터 보기 참조서, SC34-7446
CICSplex SM 워크로드 관리, SC34-7444
CICSplex SM 자원 사용 관리, SC34-7443
CICSplex SM 비즈니스 애플리케이션 관리, SC34-7442

프로그래밍

CICSplex SM 애플리케이션 프로그래밍 안내서, SC34-7439
CICSplex SM 애플리케이션 프로그래밍 참조서, SC34-7440

진단

CICSplex SM 자원 테이블 참조서 제 1권, SC34-7449
CICSplex SM 자원 테이블 참조서 제 2권, SC34-7450

CICSplex SM 메시지 및 코드, GC34-7445

CICSplex SM 문제점 판별, GC34-7448

기타 CICS 서적

다음 서적에는 CICS에 대한 자세한 정보가 포함되어 있으나 z/OS용 CICS Transaction Server, 버전 5 릴리스 3 과 함께 제공되지는 않습니다.

Designing and Programming CICS Applications, SR23-9692

CICS Application Migration Aid Guide, SC33-0768

CICS 제품군: API Structure, SC33-1007

CICS 제품군: Client/Server Programming, SC33-1435

CICS 제품군: Interproduct Communication, SC34-6853

CICS 제품군: Communicating from CICS on System/390, SC34-6854

z/OS용 CICS Transaction Gateway 관리, SC34-5528

CICS 제품군: 일반 정보, GC33-0155

CICS 4.1 Sample Applications Guide, SC33-1173

CICS/ESA 3.3 XRF Guide, SC33-0661

기타 IBM 서적

다음 서적은 관련 IBM 제품에 관한 정보를 포함합니다.

MVS

z/OS MVS Initialization and Tuning Guide, SA22-7591

z/OS MVS Initialization and Tuning Reference, SA22-7592

z/OS MVS JCL Reference, SA22-7597

OS/390 V2R8.0 MVS System Commands, GC28-1781

SNA

Sessions between Logical Units, GC20-1868.

SQL

DB2 Universal Database for OS/390 and z/OS: Application Programming and SQL Guide, SC26-9933

DB2 Universal Database for OS/390 and z/OS: SQL Reference, SC26-9944.

기타 관련 서적

An Introduction to the IBM 3270 Information Display System, GA27-2739

3274 Control Unit Reference Summary, GX20-1878

Component Description: IBM 2721 Portable Audio Terminal, GA27-3029

IBM 2780 Data Transmission Terminal Component Description, GA27-3035
CICS/ESE 3.3 IBM 3270 Data Stream Device Guide, SC33-0232
IBM 3270 Data Stream Programmer's Reference, GA23-0059
IBM 4700/3600/3630 Guide, SC33-0233

내게 필요한 옵션

내게 필요한 옵션 기능은 거동 장애나 시각 장애 같은 신체적 장애가 있는 사용자가 소프트웨어 제품을 사용할 수 있도록 도와줍니다.

다음 중 한 가지 방법으로 CICS 시스템을 설정, 실행, 유지보수하는 데 필요한 대부분의 태스크를 수행할 수 있습니다.

- CICS에 로그인된 3270 에뮬레이터 사용
- TSO에 로그인된 3270 에뮬레이터 사용
- 3270 에뮬레이터를 MVS 시스템 콘솔로 사용

IBM Personal Communications는 장애를 가진 사람들을 위한 내게 필요한 옵션 기능을 3270 에뮬레이션에 제공합니다. 이 제품을 사용하여 CICS 시스템에서 필요한 내게 필요한 옵션 기능을 제공할 수 있습니다.

색인

[가]

가로 탭 884
 가상 색인 기능(VLF) 158
 가상 저장영역 305
 가상 저장영역 환경 304
 간격 제어 777
 간격 제어 명령 취소 777
 만기 시간 778
 요청 ID 지정 777
 태스크 시작 777
 태스크의 지연 처리 777
 DELAY 명령 778
 POST 명령 778
 START 명령 778
 간접 큐 809
 감지 가능 필드 1014
 개방형 트랜잭션 환경(OTE) 278
 CONCURRENCY(REQUIRED) 프로그램 286
 OPENAPI 프로그램 287
 제한사항 289
 개요
 채널을 사용한 동적 라우팅 353
 갱신 작업, BDAM 711
 결합 기능 데이터 테이블 690
 결합 기능 목록 구조
 현재 값 852
 고유 런타임 라이브러리 11
 공간, 표시 1001
 공백 필드 1018
 공유 데이터 테이블 689
 공유 저장영역 299
 교착 697
 교착상태 309
 방지 681
 구성요소
 다중, 대화식 329
 여러 개의 채널, 한 개의 구성요소 328
 한 개의 채널-여러 개의 프로그램 327
 구성요소 구조 653
 그룹 필드 927
 글로벌 사용자 엑시트 793

기능 관리 헤더
 설명 745
 기능 키 205, 246
 기능(PF) 키, CEBR 트랜잭션 234
 기능, EDF 201
 기본 임시 저장영역 295
 기본 저장영역 305
 기본값
 조건에 대한 조치 423
 기술, 프로그래밍 304
 기호 818, 832
 값 정의 837, 839
 문서 템플릿에 배치 832
 입력 맵 956
 출력 맵 956
 기호 대체 818
 기호 레지스터 DFHEIPLR 76
 기호 목록 818, 837, 839
 기호 설명 맵핑 세트
 프로그램에서 사용 163
 기호 참조 818, 832
 기호 커서 위치 지정 949
 기호 테이블 818

[나]

내용 유형 xi
 내용 유형 맵핑 xi
 널(null) 값, 사용 1020
 널(null) 매개변수, DFHNCTR 호출의 예제 866
 논리 디바이스 구성요소 1010
 논리 레벨, 애플리케이션 35, 366
 논리 레코드 표시 744
 논리 메시지에 대한 규칙 968
 논리 메시지, 규칙 968
 논리 장치(LU)
 기능 743
 논리적 작업 단위(LUW)
 복구 가능 자원 269
 사용된 동기점 421
 설명 269

[다]

다른 프로그램으로 데이터 전달 367
 다중 기본 레지스터 76
 단일 스레드 테스트 197
 단일 화면 모드, EDF 214
 대기, 운영 체제 311
 대기, 터미널 728
 대상 ID 752
 대체
 색인 673
 키 673
 대화 파트너 725
 대화식 디버깅
 CECI 트랜잭션 241
 CECS 트랜잭션 247
 CEDF 트랜잭션 201
 대화식 문제점 제어 시스템 450
 대화식 프로그래밍 267, 755
 데이터
 다른 프로그램으로 전달 367
 레코드 300
 정의 306
 채이닝 743
 초기화 306
 트랜잭션에 저장 292
 데이터 교환 블록 98
 데이터 끝 지시 문자 748
 데이터 변환 353
 및 채널 353
 SOAP 예제 359
 채널 사용 355
 필요한 이유 353
 데이터 세트 716
 비어 있음 673
 사용자 299
 순차 313
 일괄처리 데이터 상호 교환 751
 차단됨 686
 BDAM 686, 687
 CICS 애플리케이션에서 액세스 698
 데이터 세트 일시중단 813
 데이터 스트림
 버퍼 주소 설정 순서 1020

- 데이터 스트림 (계속)
 - 압축 1020
 - 인바운드 1017
 - 주소까지 반복 순서(SBA) 1020
 - RA 순서 1020
 - SBA 순서 1020
- 데이터 테이블
 - 결합 기능 690
 - 공유 689
- 데이터를 맵으로 이동 940
- 데이터의 채이닝 743
- 데이터, 맵으로 이동 940
- 데이터, 표시장치에서 읽기 957
- 독점 자원 309
- 동기점 지정 421, 422
- 동기화 조치
 - 저널 출력 419
- 동기화, DPL 458
- 동시 찾아보기 703
- 동적
 - 저장영역 305
 - 트랜잭션 라우팅 370
 - 트랜잭션 백아웃 프로그램 443
 - 프로그램 308
- 동적 링크 라이브러리 18
- 동적 저장영역, 확장 86
- 동적 프로그램 링크
 - 연관관계 413
- 동적 LIBRARY 145
- 동적 LIBRARY 자원 사용 145
- 동적으로 라우팅된 애플리케이션
 - 채널 및 컨테이너로 마이그레이션 363
- 디바이스 종속 맵 933
- 디바이스 종속 지원 934
- 디버깅 201

[라]

- 라우팅 목록 988
 - 세그먼트 체인 항목 형식 991
 - 표준 항목 형식 991
 - LIST 옵션 991
- 라우팅, 트랜잭션 452
- 라이브러리 색인(LLA) 158
- 라이트 펜 감지 가능 필드 1014
- 런타임 라이브러리
 - 고유 11
 - Language Environment 11

- 레벨, 애플리케이션 논리 366
- 레코드
 - 갱신 708
 - 길이 300
 - 삭제 711
 - 식별 687
 - 쓰기 708, 713
 - 읽기 698
 - 잠금 681
 - 잠금(RLS) 682
 - 저널 418
 - 찾아보기 698
 - 추가 713
 - BDAM 데이터 세트에 추가 715
 - ID 677
- 레코드 갱신 708
- 레코드 레벨 공유(RLS)
 - RLS 모드에서 파일에 액세스 676
- 레코드 삭제 711
- 레코드 설명 필드 716
- 레코드 쓰기 708, 713
- 레코드 읽기 698
- 레코드 잠금 682
- 레코드 추가 713
- 레코드의 공유 제어
 - VSAM RLS 682
- 레코드의 독점 제어
 - BDAM 711
 - VSAM 681
 - VSAM RLS 682
- 로깅 314
- 로드 라이브러리
 - 2차 익스텐트 지원 166
- 로컬 복사 키 894
- 리전 내부 큐 807
- 리전 내부 트랜지언트 데이터 298
- 리전 외부 큐 808
- 리전 외부 트랜지언트 데이터 299, 313
- 리전간 통신 470
- 링크 기능 인터페이스를 사용하여 JSON 변환 577
- 링크 팩 영역(LPA) 158
- 링크 편집 93, 99
- 링크에서 리턴된 컨테이너 발견 343

[마]

- 마이그레이션
 - 채널을 사용하기 위한 START 데이터 363
- 만기 시간
 - 지정 778
- 매개변수
 - 널(null) 866
- 맵
 - 기호 입력 956
 - 기호 출력 956
 - 데이터 이동 940
 - 디바이스 종속 933
 - 링크 편집(link-edit) 924
 - 세트 308, 926
 - 작성 916
 - 출력 초기화 939
 - 플로팅 974
 - BMS 914, 1018, 1020
- 맵의 링크 편집 924
- 맵핑 세트
 - 프로그램에서 기호 설명 맵핑 세트 사용 163
 - 16MB 라인 이상 로드 183
 - CSECT 추가 193
- 멀티맵 화면 1020
- 멀티스레드 테스트 197
- 멀티스레딩 276
- 멀티페이지 출력 1020
- 메시지 라우팅 987
- 메시지 라우팅으로 대화 인터리빙 997
- 메시지 제목 995
- 메시지, 전달할 수 없음 994
- 명령 레벨 해독기
 - 보안 고려사항 241
 - 호출 247
 - ENTER 키 246
- 명령 언어 변환기 96, 111
 - 옵션 111, 113
 - 행 번호 98
 - APOST 옵션 100
 - CBLCARD 옵션 100
 - CICS 옵션 100
 - COBOL2 옵션 100
 - COBOL3 옵션 100
 - CPSM 옵션 100
 - DBCS 옵션 100

명령 언어 변환기 (계속)

DEBUG 옵션 100
 DLI 옵션 100
 EDF 옵션 100
 EPILOG 옵션 100
 EXCI 옵션 100
 FLAG 옵션 100
 GDS 옵션 100
 GRAPHIC 옵션 100
 LEASM 옵션 100
 LENGTH 옵션 100
 LINECOUNT 옵션 100
 LINKAGE 옵션 100
 MARGINS 옵션 100
 NATLANG 옵션 100
 NOCBLCARD 옵션 100
 NOCPSM 옵션 100
 NODEBUG 옵션 100
 NOEDF 옵션 100
 NOEPILOG 옵션 100
 NOFEPI 옵션 100
 NOLENGTH 옵션 100
 NOLINKAGE 옵션 100
 NONUM 옵션 100
 NOOPSEQUENCE 옵션 100
 NOOPTIONS 옵션 100
 NOPROLOG 옵션 100
 NOSEQ 옵션 100
 NOSEQUENCE 옵션 100
 NOSOURCE 옵션 98, 100
 NOSPIE 옵션 100
 NOVBREF 옵션 100
 NOXREF 옵션 100
 NUM 옵션 100
 OPMARGINS 옵션 100
 OPSEQUENCE 옵션 100
 OPTIONS 옵션 100
 PROLOG 옵션 100
 QUOTE 옵션 100
 SEQ 옵션 100
 SEQUENCE 옵션 100
 SOURCE 옵션 98, 100
 SP 옵션 100
 SPACE 옵션 100
 SPIE 옵션 100
 SYSEIB 옵션 100
 VBREF 옵션 98, 100
 XOPTS 키워드 111

명령 언어 변환기 (계속)

XREF 옵션 100
 명령, SYNCPOINT 421
 모니터링 애플리케이션 성능 449
 모듈별 프로그램 308
 문서 817
 검색 845
 데이터 바꾸기 844
 데이터 추가 842
 및 CICS 웹 지원 817
 삭제 848
 작성 835
 재사용 845
 책갈피 842, 844
 코드 페이지 변환 823
 문서 템플릿 817
 기호 817, 818, 832
 기호 목록 818
 문서에 배치 835, 842, 844
 및 CICS 웹 지원 817, 823, 826
 설정 825
 엑시트 프로그램 830
 임베디드 명령
 #echo 832, 833
 #include 833
 #set 818, 832, 833
 임시 저장영역 큐 826
 캐싱 822
 엑시트 프로그램에서 지정 830
 통신 영역 830
 트랜지언트 데이터 큐 826
 파티션된 데이터 세트(PDS) 825
 CICS 파일 826
 CICS 프로그램 827, 829
 DFHDHDL 매크로 827, 829
 HFS 파일 826
 문서 템플릿의 기호 817
 문서 템플릿의 캐싱 822
 문서 템플릿의 통신 영역 830
 문자 세트 823
 물리적 맵핑 세트
 설치 187
 물리적 키 687

[바]

바인딩 658, 660

바인딩 파일

XML 바인딩 563
 반복 내용 555
 반복된 필드 929
 반이중 모드 725
 밝기 1014
 배수 효과 441
 변동 배열 538, 634
 변수, CECI/CECS 248
 변환 6, 93
 COBOL 42
 변환기
 동적 호출 100
 Language Environment 준수 컴파일러와
 통합된 93
 변환기 데이터 세트 96, 111
 변환기 동적 호출 100
 변환기 동적 호출에서 DDname 목록 111
 보안
 레코드 또는 필드 레벨 451
 프로그래밍 힌트 451
 CICS 정의의 자원 ID 451
 EDF 202
 SEC 시스템 초기화 옵션 451
 SPCOMMAND 자원 유형 451
 보조 임시 저장영역 295
 보조 추적 311
 복구
 동기점 421
 문제점 방지 301
 순차 터미널 지원 747
 자원 309
 복구 가능 자원 268
 독점 사용 268
 복사 가능
 BMS 1020
 복합 필드 927
 분산 애플리케이션 디자인 274
 뷰포트 1001
 브래킷 프로토콜, LAST 옵션 746
 브릿지(3270)
 ADS 설명자 926
 블록 분리 인수 687
 블록 참조 687
 블록화된 데이터 세트 686
 비대화식 프로그래밍 267
 비동기 저널 출력 419
 비동기 처리 452, 469

비어 있는 데이터 세트 673
 비정상 종료 복구 443
 비즈니스 서비스 653
 비터미널 트랜잭션
 EDF 217
 비CICS 프린터 877, 890

[사]

사용자
 데이터 세트 299
 저장영역 293
 추적 시작점 448
 사용자 대체 가능 모듈 793
 사용자 유지보수 테이블 689
 사용자 키 저장영역 789
 사전 인쇄 양식 813
 상대 레코드 번호(RRN) 673, 678
 상대 바이트 주소 지정(RBA)
 업그레이드 679
 상대 바이트 주소(RBA) 673, 678
 상주 모드(RMODE)
 CICS 애플리케이션을 위한 옵션 156
 상태 플래그 바이트, 라우팅 목록 991
 상표 1029
 상호 통신 452
 색인, 대체 673
 서버
 리전 455, 458
 프로그램 458
 서버공간 802
 서버루틴 306
 서비스 653
 서비스 요청자 애플리케이션 작성 648
 서비스 지향 구조 653
 서비스 처리 662
 선택 필드 1015
 세로 탭 884
 수신확인 728
 수정된 데이터 태그 959, 1017
 순차 터미널 지원 197, 747
 스레드 세이프 프로그램 278, 281
 ADDRESS CWA 281
 EXTRACT EXIT 281
 GETMAIN SHARED 281
 스텝, EXEC 인터페이스 117, 129
 스톱
 명령 5

스톱 (계속)
 파일 896
 시나리오
 다중 대화식 구성요소 329
 여러 개의 채널, 한 개의 구성요소 328
 한 개의 채널-여러 개의 프로그램 327
 한 개의 채널-한 개의 프로그램 327
 시스템 정보에 대한 액세스
 실행 인터페이스 블록(EIB) 6
 시스템 정보, 액세스 대상 5
 시스템 추적 시작점 448
 시작점, 추적 448
 식별
 BDAM 레코드 687
 신속한 서비스 전달 131, 134
 실행 단위 35
 실행 수정, EDF 219
 실행 인터페이스 블록 98
 실행 진단 기능 98, 102, 201
 싱글스레딩 276

[아]

아웃보드 형식화 1016
 애플리케이션 665, 667
 논리 레벨 366
 디자인 267
 비동기 처리 469
 사용 132
 상호 통신 고려사항 452
 설치 132, 143
 성능 향상을 위한 디자인 303
 시작하기 134
 작성 5
 전개 132
 테스트 197
 트랜잭션 라우팅 454
 BMS 맵핑 세트 사용 163
 DPL(Distributed Program Link) 455
 DTP(distributed Transaction
 Processing) 469
 function shipping 454
 XML로 변환 475
 애플리케이션 논리 레벨 35
 애플리케이션 데이터를 JSON으로 변환 646
 애플리케이션 디자인
 배수 효과 441

애플리케이션 디자인 고려사항
 자원 독점 제어 309
 애플리케이션 바인딩 667
 애플리케이션 번들 132, 665
 애플리케이션 설치
 어셈블리 언어 167
 자체 작업 스트림 사용 179
 C 174
 COBOL 169
 PL/I 174
 애플리케이션 시작점
 시작하기 134
 애플리케이션 테스트
 단일 스레드 테스트 197
 멀티스레드 테스트 197
 순차 디바이스 사용 197, 747
 순차 터미널 지원 197
 시스템 준비 199
 시스템 테이블 항목 준비 198
 애플리케이션 테이블 항목 준비 198
 회귀 테스트 197
 애플리케이션 패키징 665
 애플리케이션 프로젝트 132
 어셈블리 언어 71, 117
 다른 언어와 혼합 16, 84
 비준수 루틴 81
 애플리케이션 307
 작업 중인 저장영역 71
 제한사항 72
 프로그래밍 기술 72, 81, 84
 31비트 주소 지정 72
 64비트 레지스터 72
 CALL문 84
 DFHECALL 매크로 117
 Language Environment 요구사항 81
 어셈블리 언어의 HANDLE ABEND LABEL
 제한사항 72
 어셈블리 93
 어셈블리 애플리케이션 설치
 샘플 작업 스트림 167
 어셈블리 언어 71
 작업 중인 저장영역 71
 프로그래밍 기술 84
 64비트 주소 지정 모드 애플리케이션 88
 어셈블리, TYPE=DSECT 924
 어텐션 필드 1015
 어텐션 ID 960

언어 구조	운영 체제 대기 311	인터페이스 모듈
JSON으로 변환 579	원격 링크 프로그램	사용 166
WSDL로 변환 477	DPL 217	CPI 통신 116
언어 구조를 XML로 맵핑 563	EDF 217	EXEC 116
언어 구조에서 맵핑 생성 642	원격 트랜잭션, EDF 216	EXEC CICS 또는 EXEC DLI 명령을 사
언어 혼합 16	유틸리티 프로그램	용하는 프로그램 179
업그레이드 세트 673	JSON 지원 578	SAA 자원 복구 116
엑시트 프로그램	XML 지원 476	인터페이스 스텝, EXEC 117, 129
문서 템플리트 830	의사 대화식 프로그래밍 267	일괄처리 데이터 상호 교환 751
통신 영역 830	이름 지정된 카운터 851	대상 ID 752
연관관계 380	개요 851	확정 응답 752
권장되는 프로그래밍 기술 382	결합 기능 목록 구조 852	DEFRESP 옵션 752
사용자 ID 416	옵션 테이블 853	ISSUE WAIT 명령 753
안전하지 않은 프로그래밍 기술 382	이름 지정된 카운터 필드 852	NOWAIT 옵션 753
안전한 프로그래밍 기술 382	최대값 852	일괄처리 유틸리티
의심스러운 프로그래밍 기술 382	최소값 852	JSON 지원 578
BAPPL 410	카운터 이름 852	XML 지원 476
LOCKED 413	풀 852	일반 삭제 711
LU 이름 415	현재 값 852	일반 키 699
연관관계 그룹 410	CICS API 854	읽기 우선 큐 처리 727
연기된 저널 출력 420	DFHNC001 853	읽기 전용 컨테이너 350
영역, 동적 저장영역 305	DFHNCO 매크로 853	읽기 전용 DSA(RDSA) 159
예비 변환된 코드 178	이벤트	임시 데이터 813
예외 조건	모니터링 위치 449	임시 저장영역
설명 423	이상 종료 428	기본 295
HANDLE CONDITION 명령 428, 430	이상 종료 엑시트 기능 443	데이터 813
IGNORE CONDITION 명령 433	이상 종료 엑시트 루틴 445	보조 295
예외 추적 시작점 448	이상 종료 엑시트 프로그램 445	찾아보기 트랜잭션, CEBR 231
예제	이스케이프 시퀀스 818, 839	채널 및 컨테이너로 마이그레이션 363
다중 대화식 구성요소 329	이스케이프 해제 818, 839	임시 저장영역 큐 295
여러 개의 채널, 한 개의 구성요소 328	이탈 태스크 311	문서 템플리트 826
채널을 구성하는 CICS 클라이언트 프로그	인바운드	트랜잭션간 연관관계 397
램 349	데이터 스트림 1017	입력 데이터
채널을 사용하는 CICS 서버 프로그램	인쇄 877	체이닝 743
349	트랜지언트 데이터 888	입력 데이터 맵핑 957
채널, 기본 323	행 길이 881	입력 맵, 기호 956
컨테이너, 기본 323	BMS 라우팅으로 890	
한 개의 채널-여러 개의 프로그램 327	CICS API 고려사항 891	
한 개의 채널-한 개의 프로그램 327	START 명령 888	
BTS 활동에 비해 단순 클라이언트 프로그	인쇄 제어 비트 880	
램 350	인쇄 중 행 길이 881	
오버레이 308	인쇄 키 894	
옵션	인쇄 키, 하드웨어 894	
기능 키에서, EDF 221	인쇄 형식화 748	
HANDLE CONDITION 명령 430	인쇄된 출력, 요청 878	
외부 제어기 751	인터럽트 728	
외부 CICS 인터페이스(EXCI) 470	인터럽트 되돌리기 728	
요청/응답 단위(RU) 743		

[자]

자기 슬롯 리더, 10/63 1012
자동 태스크 초기화 270, 726
자원
경합 268
독점 사용 268
독점 제어 309
복구 가능 268, 309
엑세스의 순서 제어 783
제어 268

자원 백아웃 421
자원 액세스 순서, 제어 783
자원에 대한 경합 268
작업 단위 421
작업 세트 306
재진입성 276
저널
 레코드 300, 418
저널 유형 ID 421
저널 제어
 출력 동기화 419
저널 ID 421
저널링 313, 418
저장영역
 공유 가능 788
 사용자 293
 사용자 키 789
 정적 308
 주 305
 채널
 수명 344
 컨테이너
 수명 344
 프로그램 295
 CICS 키 789
저장영역 보호 787
저장영역 제어 787
저장영역, 동적 86, 305
전달할 수 없는 메시지 994
정보 포함 명령 833
정적 저장영역 308
정적 LIBRARY(DFHRPL) 145
정책
 시작하기 134
제거 지연 994
제목, 메시지 995
제어
 BDAM 독점 711
 VSAM 블록의 681
제어 전달, 리턴 예상(LINK) 366
제한사항
 어셈블러 언어 72
 31비트 주소 지정 72
 64비트 레지스터 72
 C 및 C++ 54
 COBOL 26, 37, 42
 PL/I 66
조건, 예외 423

조회 트랜잭션 271
주소 지정 모드(AMODE)
 CICS 애플리케이션을 위한 옵션 156
지원되는 컴파일러
 Language Environment 11
지원, JSON 578
지원, XML 476
지정 문자 1014
직접 터미널 997

[차]

참조 국소성 306
참조 세트 308
참조 수정 42
참조 유효성 306
찾아보기 718
 레코드 703
 DELAY 718
 SUSPEND 718
찾아보기 작업
 BDAM 707
채널
 개요 321, 322
 구성 348
 기본 예제 323
 데이터 변환 355
 동적 및 분산 라우팅 353
 링크에서 리턴된 컨테이너 발견 343
 마이그레이션 360
 동적으로 라우팅된 애플리케이션 363
 임시 저장영역 363
 LINK 명령 361
 RETURN 명령 362
 XCTL 명령 362
 범위 338, 339, 341
 설계 346
 수명 344
일반 시나리오
 다중 대화식 구성요소 329
 여러 개의 채널, 한 개의 구성요소 328
 한 개의 채널-여러 개의 프로그램 327
 한 개의 채널-한 개의 프로그램 327
읽기 전용 컨테이너 350
작성 332
장점 330
큰 COMMAREA 321

채널 (계속)
 프로그램에 전달된 컨테이너 발견 343
 현재 333, 338
 현재, 예제, LINK 명령 포함 333
 현재, 예제, XCTL 명령 포함 336
 BTS 활동과 비교하여 350
 CICS 변환기 338
 JCICS에서 사용 350
 LINK 명령 373
 LINK 및 XCTL 명령에서 295
 RETURN 명령 376
 RETURN 명령에 320
채널 구조 348
채널 기반 서비스 658, 662
채널 설계 346
채널 작성 332
채널을 사용하기 위해 START 데이터를 마이
 그레이션 363
채널을 사용한 동적 라우팅 353
채널의 범위
 개요 338
 예제, LINK 명령 포함 339
 예제, XCTL 명령 포함 341
책갈피 842, 844
채이닝 731
체크아웃, 프로그램 201
추가 32K COMMAREA 321
추적
 설명 448
 추적 시작점 448
출력 데이터, 채이닝 743
출력 맵 초기화 939
출력 맵, 기호 956
출력 맵, 초기화 939

[카]

카운터 이름
 이름 지정된 카운터 852
카피북 변환 115
커서 감지 가능 필드 1014
커서 위치 949
커서 위치 지정, 기호 949
커서 찾기 962
커서, 찾기 962
컨테이너
 개요 321, 322
 기본 예제 323

컨테이너 (계속)

- 링크에서 리턴되는 항목 발견 343
- 마이그레이션 360
 - 동적으로 라우팅된 애플리케이션 363
 - 임시 저장영역 363
 - LINK 명령 361
 - RETURN 명령 362
 - XCTL 명령 362
- 수명 344
- 읽기 전용 350
- 채널 설계 346
- 컨텍스트, BTS 또는 채널 351
- 프로그램에 전달된 컨테이너 발견 343
- JCICS에서 사용 350

컨텍스트

- 컨테이너, BTS 또는 채널 351

컴파일 93

컴포지트 655

컴포지트 와이어링 655

컴포지트 필드 927

큐

- 리전 내부 807

- 리전 외부 808

큐 처리

- 트랜지언트 데이터 807

큐에 넣기

- VSAM 내부 309

- VSAM 파일의 309

크기, 프로그램 304

큰 COMMAREA 321, 322, 327, 330, 350

큰 COMMAREA의 채널 321, 322, 327,

330, 350

클라우드 131

클라우드 애플리케이션 131

클라우드 인에이블먼트 134

클라이언트 리전 455

클라이언트 코드 페이지 823

키

- 대체(2차) 673

- 물리적 687

- 일반 699

- 하드웨어 인쇄 894

[타]

태스크 관련 사용자 엑시트 793

태스크 제어 781

- 자원 액세스 순서 783

탭, 가로 884

탭, 세로 884

터미널

- 경합 726

- 공유 252

- 대기 728

- 옵션 943

- 자원, 순차 747

터미널 라우팅 997

터미널 제어

- 기능 관리 헤더(FMH) 745

- 논리 레코드 표시 744

- 논리 장치에 대한 기능 743

- 대화 파트너 725

- 명령 725

- 반이중 모드 725

- 브래킷 프로토콜, LAST 옵션 746

- 인쇄 형식화 748

- 인터럽트 728

- 읽기 우선 큐 처리 727

- 입력 데이터 맵핑 957

- 입력 데이터의 체이닝 743

- 출력 데이터의 체이닝 743

- 테이블 사용자 영역(TCTUA) 319

- 파트너, 대화 725

- 프로토콜 중단 727

- 프로토콜, break 727

- 플립 플롭 모드 725

- 확정 응답 744

- FMH, 아웃바운드 746

- FMH, 인바운드 745

터미널 제어 명령 721

터미널에 대한 경합 726

터미널, 확장 데이터 스트림 187

통합 변환기 6, 93

통합 CICS 변환기 93

트랜잭션

- 교착상태 695

- 대화식 267

- 라우팅 452, 454

- 라우팅, 동적 370

- 비대화식 267

- 연관관계 364, 454, 777, 781, 788, 797

- 의사 대화식 267

트랜잭션 그룹

- 연관관계 410

트랜잭션 분리 787

트랜잭션 사이에 데이터 공유 314

트랜잭션 시스템 연관관계 381

트랜잭션 연관관계

- 트랜잭션 시스템 연관관계 381

- 트랜잭션간 연관관계 381

트랜잭션 ID

- CEBR 231

- CECI 247

- CEDF 트랜잭션 201

트랜잭션간 연관관계

- 관련사항 및 수명 410

- 글로벌 관련사항 412

- 터미널 관련사항 415

- BAPPL 관계 410

- userid 관련사항 416

발견 408

안전하지 않은 프로그래밍 기술 389

- 공유 저장영역 사용 390

- 태스크 수명 저장영역 사용 393

- CWA 389

- DEQ 사용 395

- ENQ 사용 395

- LOAD PROGRAM HOLD 사용

391

- WAIT EVENT 사용 394

안전한 프로그래밍 기술 384

- BTS 컨테이너 사용 388

- COMMAREA 384

- DEQ를 ENQMODEL과 함께 사용

387

- ENQ를 ENQMODEL과 함께 사용

387

- TCTUA 385

애플리케이션 생성기에 의해 초래됨 408

연관관계 수명 409

연관관계 트랜잭션 그룹 410

의심스러운 프로그래밍 기술

- 글로벌 사용자 엑시트 381

- 트랜지언트 데이터 400

- DELAY 및 CANCEL REQID 명령

405

- INQUIRE 및 SET 명령 381

- POST 명령 407

- RETRIEVE WAIT 및d START 명령

402

- START 및 CANCEL REQID 명령

403

임시 저장영역 397

임시 저장영역 데이터 공유 397

트랜잭션간 연관관계 (계속)
 프로그래밍 기술 382
 트랜잭션에 데이터 저장 292
 트랜지언트 데이터 888
 리전 내부 298
 리전 외부 299, 313
 큐 298, 400
 트랜지언트 데이터 제어
 큐 807, 808
 ATI(Automatic Transaction
 Initiation) 809
 트랜지언트 데이터 큐
 문서 템플릿 826
 트리거 필드 1013

[파]

파일 제어
 개요 673
 BDAM 데이터 세트 687
 파트너 범위 469
 파트너, 대화 725
 파트너, 범위 469
 파티션 1001
 파티션 세트
 설치 194
 16MB 라인 이상 로드 183
 파티션된 데이터 세트(PDS)
 문서 템플릿 825
 파티션, 정의 1003
 파티션, 활성 1006
 페이지 결합 306
 페이지 라우팅 조작 1020
 페이지 빌드 조작 1020
 페이지 오버플로우 996
 페이지 중단 974
 페이지징
 영향 축소 306
 펜 감지 가능 필드 1014
 표시 공간 1001
 표시장치
 레지스터, EDF 223
 화면 321
 표시장치 특성 940
 표시장치에서 데이터 읽기 957
 표시장치, 읽기 957
 프로그래밍 기술
 어셈블리 언어 72, 81, 84

프로그래밍 기술 (계속)
 어셈블리 언어 84
 일반 304
 C 및 C++ 54
 COBOL 26, 33, 37
 PL/I 65, 66, 67, 69
 프로그래밍 모델 267
 프로그래밍 제한사항
 어셈블리 언어 72
 C 및 C++ 54
 COBOL 26, 37, 42
 PL/I 66
 VS COBOL II 32
 프로그램
 문서 템플릿 827
 크기 304
 테스트 201
 프로그램 디자인
 대화식 267, 755
 비대화식 267
 의사 대화식 267
 프로그램 저장영역 295
 프로그램 제어
 다른 프로그램에 링크 366
 다른 프로그램으로 데이터 전달 367
 프로그램 논리 레벨 366
 프로그램에 링크, 리턴 기대 366
 프로그램에 전달된 컨테이너 발견 343
 프린시플 기능 270, 271
 프린터
 비CICS 877, 890
 3270 879
 옵션 880
 CICS 877
 특성 판별 885
 SCS 882
 플래그 바이트, 라우팅 목록 991
 플랫폼 665
 시작하기 134
 애플리케이션 바인딩 667
 플로팅 맵 974
 플립 플롭 모드 725
 필드
 공백 1018
 그룹 927
 반복 929
 복합 927
 컴포지트 927

필드 (계속)
 BMS 915

[하]

하위 엔클레이브 19
 현재 채널
 개요 333, 338
 예제, LINK 명령 포함 333
 예제, XCTL 명령 포함 336
 형식화된 화면에서 읽기 958
 형식화된 화면, 읽기 958
 호스트 코드 페이지 823
 호출 가능 서비스 12
 혼합된 주소 지정 모드 트랜잭션 372
 화면 복사, BMS 894
 화면 필드, 3270 959
 화면, 형식화된에서 읽기 958
 확장 읽기 전용 DSA(ERDSA) 159
 확장된 상대 바이트 주소 지정(XRBA)
 업그레이드 679
 확장된 상대 바이트 주소(XRBA) 678
 확장 응답 프로토콜
 터미널 제어 744
 활성 파티션 1006
 회귀 테스트 197
 회선 트래픽 축소 1019

[숫자]

10/63 자기 슬롯 리더 1012
 2진 데이터를 XML로 변환 568
 2차 익스텐트, CICS 로드 라이브러리 166
 31비트 모드 트랜잭션 372
 31비트 주소 지정
 어셈블리 언어 72
 COBOL 26
 3262 프린터 877
 3270 브릿지
 ADS 설명자 926
 3270 정보 표시 시스템 187
 3270 제품군
 강도 761
 기본 색상 761
 데이터 스트림 757
 데이터 스트림 순서 763
 데이터 스트림에서의 순서 763
 데이터 스트림, 아웃바운드 767

3270 제품군 (계속)

- 버퍼 757
- 보호 761
- 색상, 기본 761
- 속성, 확장됨 763
- 수정된 데이터 태그 761
- 쓰기 제어 문자 758
- 아웃바운드 데이터 스트림 767
- 어텐션 ID 770
- 인바운드 필드 형식 772
- 읽기 771
- 입력 위치 770
- 주의 키 770
- 터미널에 쓰기 758
- 터미널, 쓰기 758
- 표시장치 특성 760
- 필드 760
- 필드 속성 761
- 필드 형식, 인바운드 772
- 형식화되지 않은 모드 773
- 확장 속성 763
- AID 770
- MDT 761
- 3270 표시장치 731
- 3270 프린터 879
 - 옵션 880
- 3270 화면 필드 959
- 3289 프린터 877
- 3290 표시장치 1001
- 3601 논리 장치 1010
- 3770 일괄처리 논리 장치 1010
- 3770 일괄처리 데이터 상호 교환 논리 장치 1010
- 3790 일괄처리 데이터 논리 장치 1010
- 64비트 주소 지정 모드
 - 어셈블리 언어 88

A

- abend user task, EDF 221
- ABEND 명령 443
- ACCEPT문, COBOL 26
- ACCUM 옵션 880, 968
- ACK 728
- ACTPARTN 옵션 969, 1006
- ADDRESS COMMAREA 명령 368
- ADDRESS 명령 5
- ADDRESS 특수 레지스터 33

- ADS 설명자 926
- AFFINITY 속성 410
- AFFLIFE 속성 410
- AFTER 옵션 778
- ALARM 옵션 943
- ALLOCATE 명령 312
 - 대기 금지, NOSUSPEND 옵션 312
- ALLOCERR 조건 898
- ALTPAGE 값 973
- AMODE(64)
 - 어셈블리 언어 88
- AMODE(주소 지정 모드)
 - CICS 애플리케이션을 위한 옵션 156
- APAK 트랜잭션 885
- APCG 305
- API
 - DPL을 위한 서브세트 463
- APOST 옵션 100
- APPLCTN 132
- APPLDEF 132
- application/x-www-form-urlencoded 818
- argc 54
- argv 54
- ASIS 옵션 960
- ASKTIME 명령 777
- ASSIGN 명령 5, 739, 885
 - 옵션 739
- DESTCOUNT 옵션 996
- MAPCOLUMN 옵션 978
- MAPHEIGHT 옵션 978
- MAPLINE 옵션 978
- MAPWIDTH 옵션 978
- MSR 옵션 1012
- PAGENUM 옵션 996
- AT 옵션 778
- ATI 270, 726, 809
- ATI(Automatic Transaction Initiation) 809
- ATNI 454
- Atom 피드
 - XML 바인딩
 - 작성 563
- ATTENTION 키 728
- AUTOPAGE 옵션 971
- AZI6 454

B

- BAKR(Branch And Stack) 어셈블리 명령어 72
- BASE 옵션 939
- BDAM 716
 - 갱신 작업 711
 - 데이터 세트 686, 687
 - 독점 제어 711
 - 찾아보기 작업 707
- BDI 877
- BGAM 723
- BMS 877
 - 라우팅 890
- BMS 명령 721
- BMS(Basic Mapping Support)
 - 결과 예제 913
 - 경로 길이 최소화 979
 - 경로 길이, 최소화 979
 - 그룹 필드 927
 - 기호 맵 186, 924
 - 기호 설명 맵핑 세트 설치 189
 - 대문자 변환 960
 - 데이터 스트림 1017
 - 데이터를 맵으로 이동 940
 - 데이터, 맵으로 이동 940
 - 링크 편집(link-edit) 924
 - 매크로 916
 - 매크로를 쓰기 위한 규칙 921
 - 매크로, 쓰기 규칙 921
 - 맵 914, 1017, 1018, 1020
 - 맵 어셈블링 924
 - 맵 작성 916
 - 맵 준비 183
 - 맵의 저장영역 937
 - 맵핑 세트 926
 - 맵핑 세트 설치 183, 185
 - 맵핑 세트의 유형 186
 - 맵, 기호 924
 - 맵, 데이터 이동 940
 - 맵, 물리적 924
 - 맵, 어셈블링 924
 - 맵, 작성 916
 - 맵, 저장영역 937
 - 맵, 출력 초기화 939
 - 멀티맵 화면 1020
 - 메시지 길이 줄이기 980
 - 메시지 길이, 감소 980

BMS(Basic Mapping Support) (계속)

물리적 맵핑 186, 924
 물리적 맵핑 세트 설치 187
 물리적 맵핑 세트 어셈블링 및 링크 편집 187
 반복된 필드 929
 복사 기능 1020
 복합 필드 927
 성능 가이드 979
 수정된 데이터 태그(MDT) 1017
 애플리케이션에서 BMS 맵핑 세트 사용 163
 어셈블리, TYPE=DSECT 924
 올라르지 않은 데이터 950
 입력 데이터 맵핑 957
 전체 912
 지원되는 터미널 912
 최소 912
 출력 맵 초기화 939
 출력 맵, 초기화 939
 커서 위치 949
 커서 찾기 962
 커서, 찾기 962
 콤포지트 필드 927
 파티션 세트 설치 194
 페이지 라우팅 조작 1020
 페이지 빌드 조작 1020
 표시장치에서 데이터 수신 953
 표시장치, 데이터 수신 953
 프로그램에서 기호 맵핑 세트 사용 189
 플랫폼 전체에 걸친 지원 912
 필드 915
 필드, 그룹 927
 필드, 반복됨 929
 필드, 복합 927
 필드, 콤포지트 927
 화면 복사 894
 BMS 지원 레벨 912
 BMS(Basic Mapping Support)
 표준 912
 BMS의 기호 설명 맵핑 세트 189
 DFHASMVS 프로시저 187
 DFHLNKVS 프로시저 187
 DFHMAPS, 맵을 설치하기 위한 프로시저 193
 DFHMDF 매크로 916, 917
 DFHMMDI 매크로 916, 920
 DFHMSD 매크로 916, 920

BMS(Basic Mapping Support) (계속)

DFHPSD, 파티션 세트 정의를 위한 194
 EOC 조건 966
 GRPNAME 옵션 927
 MAPSET 자원 정의 924
 MDT 959
 OCCURS 옵션 929
 PROGRAM 자원 정의 924
 SEND MAP 명령 937
 TYPE=DSECT 어셈블리 924
 BMS에서 소문자 변환 960
 BOTTOM 명령, CEBR 트랜잭션 236
 BRACKET 옵션 746
 BROWSE TEMP STORAGE 옵션, CEDF 222
 BTS 활동 350
 BUILDCHAIN 743

C

C 및 C++ 53
 다른 언어와 혼합 16
 로케일 지원 61
 인수 59
 작업 중인 저장영역 53
 제한사항 54
 지원 53
 프로그래밍 기술 54
 EIB, 액세스 60
 JSON 스키마로 맵핑 613, 616
 XML 스키마로 맵핑 517, 521
 XPLink 61, 62, 63
 C 및 C++로 맵핑 517, 521, 613, 616
 C 및 C++의 로케일 지원 61
 C 언어 고려사항
 LENGTH 옵션 기본값 300
 struct, 기호 설명 맵핑 세트 186
 CALL DL/I 인터페이스, COBOL 30
 CALL문
 어셈블러 언어 84
 CANCEL 명령 777
 CBLCARD 옵션 100
 CDUMP 54
 CEBR 트랜잭션 231
 보안 고려사항 231
 임시 저장영역 찾아보기 231
 찾아보기 트랜잭션 231
 초기화 231

CEBR 트랜잭션 (계속)

트랜지언트 데이터 238
 표시장치 233
 BOTTOM 명령 236
 CEBR 초기화 231
 COLUMN 명령 236
 FIND 명령 236
 GET 명령 237
 LINE 명령 237
 PURGE 명령 237
 PUT 명령 237
 QUEUE 명령 237
 SYSID 명령 238
 TERMINAL 명령 238
 TOP 명령 238

CECI 트랜잭션

공유 저장영역 252
 기능 키 값 영역 246
 메시지 행 246
 명령 입력 242
 명령 입력 행 242
 명령행 242
 변경 수행 248
 변수 248
 본문 245
 상태 영역 242
 소개 241
 앰퍼샌드(&) 248
 정보 영역 245
 터미널 공유 252
 프로그램 제어 251
 화면 레이아웃 241
 about to execute command 244
 command execution complete 244
 command syntax check 243
 ENQ 명령 252

CECS 트랜잭션 247

CEDF 트랜잭션 201, 202

기능 201
 기능 키 205
 기능(PF) 키의 옵션 221
 단일 화면 모드 214
 보안 202
 본문 206
 비터미널 트랜잭션 217
 실행 수정 219
 원격 링크 프로그램 217
 원격 트랜잭션 216

- CEDF 트랜잭션 (계속)
- 의사 대화식 프로그램 216
 - 이중 화면 모드 216
 - 표시장치 205
 - 표시장치 겹쳐쓰기 219
 - 표시장치 레지스터 223
 - 프로그램 레이블 219
 - 헤더 206
 - 호출 202
 - abend user task 221
 - browse temporary storage 222
 - DPL 217
 - EDF 트랜잭션 202
 - invoke CECI 222
- CEE Bint, Language Environment HLL 사용자 엑시트 24
- CEEBXITA 사용자 엑시트 21
- CEECSTX 사용자 엑시트 21
- CEEDOPT CSECT 19
- CEEENTRY 매크로 81
- CEEHDLR 서비스 13
- CEEROPT CSECT 19
- CEEUOPT CSECT 19
- CEEWUCHA 샘플 사용자 조건 핸들러 32, 67
- CESF, GOODNIGHT 트랜잭션 749
- CHANNEL
- 옵션 320, 364, 370
 - LINK 명령 364
 - RETURN 명령 364
 - XCTL 명령 364
- CICS
- 테스트 환경 7
- CICS 값 데이터 영역 300
- CICS 덤프 유틸리티 프로그램 450
- CICS 문서의 코드 페이지 823
- CICS 번들 665
- CICS 영역 주소 지정 1018
- CICS 영역, 주소 지정 1018
- CICS 옵션 100
- CICS 웹 서비스
- 유니코드 561, 640
 - UTF-16 561, 640
- CICS 유지보수 테이블 689
- CICS 키 저장영역 789
- CICS 파일
- 문서 템플릿 826
- CICS 프로그램
- 문서 템플릿 827
 - DFHDHDLT 매크로 827, 829
- CICS 프린터 877
- 특성 판별 885
- CICS DATAKEY 옵션 293, 793
- CICS VAR 환경 변수 22
- CLASS 옵션 902
- CLEAR
- 키 1009
 - PARTITION AID 값 1009
 - PARTITION 키 1009
- CLEAR 키 321
- CLOCK 54
- CMT 689
- CNOTCOMPL 옵션 743
- COBOL 25
- 가변적 반복 내용 555
 - 글로벌 변수 42
 - 다른 언어와 혼합 16
 - 변환 42
 - 빈 행 42
 - 서브프로그램 호출 34, 37
 - 실행 단위 35
 - 예약어 테이블 26
 - 일괄처리 컴파일 44
 - 작업 중인 저장영역 25
 - 제한사항 26, 37, 42, 307
 - 중첩된 프로그램 47
 - 지원 25
 - 참조 수정 42
 - 컴파일러 옵션 26
 - 프로그래밍 기술 26, 33, 37
 - 프로그래밍 제한사항
 - VS COBOL II 32
 - 31비트 주소 지정 26
 - ADDRESS 특수 레지스터 33
 - CALL DL/I 인터페이스 30
 - CBLSHPOP 런타임 옵션 30
 - CICS 데이터 영역 주소 지정 33
 - DFHNCTR 호출의 예제 866
 - JSON 스키마로 맵핑 602, 607
 - REPLACE문 42
 - WITH DEBUGGING MODE 26
 - XML 스키마로 맵핑 505, 511
- COBOL 프로그램의 일괄처리 컴파일 44
- COBOL2 옵션 100
- COBOL3 옵션 100
- COBOL로 맵핑 505, 511, 602, 607
- COBOL의 글로벌 변수 42
- COBOL의 중첩된 프로그램 47
- CODEREG 피연산자 76
- COLUMN
- 명령, CEBR 트랜잭션 236
- COM 어셈블러 명령어 72
- COMMAREA 268, 292, 294, 306
- 옵션 320, 364, 367, 369
 - 채널 및 컨테이너로 마이그레이션 360
 - 동적으로 라우팅된 애플리케이션 363
 - LINK 명령 361
 - RETURN 명령 362
 - XCTL 명령 362
 - LINK 명령 364
- COMMAREA 현대화 322
- COMMAREA > 32K 321, 330, 350
- Communications Server 725
- CONNECT PROCESS 명령 466
- CONSISTENT 옵션
- READ 명령 699
- CONVERSE 명령 731, 747, 755
- COPY문 115
- CPI
- 참조 5
- CPI 통신 스텝 469
- CPI 통신 인터페이스 모듈, DFHCPLC 116
- CPI-C 452, 469
- CPSM 옵션 100
- CQRY 트랜잭션 271
- CSECT, 맵 어셈블리에 추가 193
- CSNAP 54
- CSPG 트랜잭션 890, 894, 970, 972
- CSPP 트랜잭션 271
- CTDLI 54
- CTEST 54
- CTLCHAR 옵션 880, 881
- CTRACE 54
- CURSOR 옵션 949, 969
- ACCUM 옵션 943
 - SEND MAP 명령
 - ACCUM 옵션 943
- CVDA 300, 451, 452
- CWA 315
- CWA(Common Work Area) 315
- 보호 315
- CWAKEY 매개변수 315

D

DATAONLY 옵션 943, 945, 1019
 DATAREG 피연산자 76
 DBCS 옵션 100
 DDS 934
 DEBKKEY 옵션 707
 DEBREC 옵션 687, 707
 DEBUG 옵션 100
 DEFRESP 옵션 752, 755
 터미널 제어 744
 DELAY 명령 777, 778
 DELETEQ TD 명령 807
 DELETEQ TS 명령 813
 DELETE문, COBOL 26
 DEQ 명령 781
 DEQUEUE 명령 890
 DESTCOUNT 옵션 996
 DESTID 옵션 752
 DESTIDLENG 옵션 752
 DFH3QSS 7
 DFHAID 54
 DFHAPXPO 19
 DFHASMVS 프로시저 187, 189, 195
 DFHBMSCA 54, 942
 DFHBMSCA 정의 962
 DFHBMSUP 927
 DFHCOMMAREA 26, 367
 DFHCPLC 469
 DFHCPLC, CPI 통신 인터페이스 모듈 116
 DFHCPLRR, SAA 자원 복구 인터페이스 모듈 116
 DFHDHTL 매크로 827, 829
 DFHDHTXD 830
 DFHDHTXH 830
 DFHDHTXL 830
 DFHDHTXO 830
 DFHDYPDS 360
 DFHEAG0, 어셈블러용 인터페이스 모듈 129
 DFHEAG, 어셈블러용 인터페이스 모듈 129
 DFHEAI0, 어셈블러용 인터페이스 모듈 117
 DFHEAI, 어셈블러용 인터페이스 모듈 117
 DFHEAP1\$, 어셈블러용 변환기 99
 DFHECP1\$, COBOL용 변환기 99
 DFHEDF 그룹 199
 DFHEDP1\$, C용 변환기 99
 DFHEGTAL 프로시저 164, 167
 DFHEIBLK 26

DFHEIEND 매크로 107, 109
 DFHEIENT 매크로 84, 107, 109
 기본값 76
 CODEREG 76
 DATAREG 76
 EIBREG 76
 DFHEIPLR 기호 레지스터 76
 DFHEIRET 매크로 84, 102, 106
 DFHEISTG 매크로 107, 109
 DFHEITAL 프로시저 164, 167
 DFHEIVAR 26
 DFHELII 117
 DFHELII, Language Environment 준수 컴파
 일라용 인터페이스 모듈 117
 DFHEPP1\$, PL/I용 변환기 99
 DFHEXEC 117
 DFHEXTAL 프로시저 164, 167
 DFHFCT 매크로 689
 DFHJS2LS
 카탈로그된 프로시저 586
 DFHLNKVS 프로시저 187, 195
 DFHLS2JS
 카탈로그된 프로시저 579
 DFHLS2SC
 카탈로그된 프로시저 477
 DFHLS2SC, CICS XML 지원 프로그램
 563
 DFHMAPS 프로시저 163, 191
 DFHMAPT
 HTML 템플릿 설치를 위한 프로시저
 192
 DFHMDF 매크로 916, 917
 표시장치 특성 940
 DSATTS 옵션 940
 MAPATTS 옵션 940
 DFHMDI 매크로 916, 920
 DFHMIRS 프로그램 458
 DFHMSCAN 유틸리티 프로그램 144
 DFHMSD 매크로 916, 920
 BASE 옵션 939
 STORAGE 옵션 939
 DFHMSD, 맵핑 세트를 어셈블링하기 위한 매크로 186
 DFHMSRCA 54, 1012
 DFHNC001
 기본 이름 지정된 카운터 풀 853
 DFHNCO 매크로
 이름 지정된 카운터 옵션 테이블 853

DFHNCOPT
 이름 지정된 카운터 옵션 테이블 853
 DFHNCTR
 널(null) 포인터를 갖는 예제 COBOL 호출
 866
 DFHPDI 매크로 1004
 DFHPEP 프로그램 443
 DFHPSD 매크로 194, 1003
 DFHRESP 변환기 함수 98, 424
 DFHSC2LS
 카탈로그된 프로시저 484
 DFHURLDS 991
 DFHVALUE 98
 DFHYITDL 프로시저 164, 176
 DFHYITEL 프로시저 164, 176
 DFHYITPL 프로시저 164, 174
 DFHYITVL 프로시저 164, 170
 DFHYXTDL 프로시저 164, 176
 DFHYXTEL 프로시저 164, 176
 DFHYXTPL 프로시저 164, 174
 DFHYXTVL 프로시저 164, 170
 DFHZITCL 프로시저 164, 170
 DFHZITPL 프로시저 164, 174
 DIB 98
 DISPLAY문, COBOL 26
 DLI 옵션 100
 DLL 18
 DL/I
 동기점 422
 참조 5
 DOCTEMPLATE 자원 정의 817, 825
 EXITPGM 속성 830
 FILE 속성 826
 HFSFILE 속성 826
 MEMBERNAME 속성 825
 PROGRAM 속성 827
 TDQUEUE 속성 826
 TSQUEUE 속성 826
 DOCTOKEN 835, 848
 DOCUMENT CREATE 명령 817, 835
 DELIMITER 옵션 837, 839
 DOCSIZE 옵션 845
 DOCTOKEN 835
 LISTLENGTH 옵션 837
 SYMBOLLIST 옵션 817, 818, 835,
 837, 839
 DOCUMENT DELETE 명령 848
 DOCTOKEN 848

DOCUMENT INSERT 명령 817, 842, 844, 845
 DOCUMENT RETRIEVE 명령 817, 845
 CLNTCODEPAGE 옵션 823, 845
 HOSTCODEPAGE 옵션 823
 DOCUMENT SET 명령 817
 DELIMITER 옵션 837, 839
 LISTLENGTH 옵션 837
 SYMBOL 옵션 817, 818, 837, 839
 SYMBOLLIST 옵션 817, 818, 837, 839
 DPL 217, 364, 422, 452, 455
 DPL(Distributed Program Link) 364
 독립적 동기점 458
 서버 리전 455, 458
 서버 프로그램 458
 예외 조건 466
 옵션 456
 클라이언트 리전 455
 프로그래밍 고려사항 463
 COMMAREA 옵션 458
 DPL API 서브세트 463
 REMOTENAME 옵션 458
 REMOTESYSTEM 옵션 458
 SYSID 옵션 458
 TRANSID 옵션 458
 DSA 305
 DSATTS 옵션 940
 DTP 452, 469
 DUMP TRANSACTION 명령 449
 DUPKEY 상태 703

E

ECBLIST 781
 EDF 98, 102, 201
 EDF 옵션 100
 EDF 표시장치에 겹쳐쓰기 219
 EDF 호출 202
 EDF의 프로그램 레이블 219
 EIB 98, 423, 726
 설명 6
 터미널 제어 피드백 740
 EIBCALEN 필드 368
 EIBCOMPL 필드 731
 EIBFN 필드 369
 EIBREG 피연산자 76
 EIB의 날짜 필드 6

EIB의 시간 필드 6
 ENDBR 명령 703
 ENQ 명령 312, 781
 ENQBUSY 조건 312
 ENQUEUE 명령 890
 EOC 조건 743, 966
 EODI 문자 748
 EODS 조건 743
 EPILOG 옵션 100
 EQUAL 옵션 699
 ERASE 옵션 881, 943, 969
 ERASEAUP 옵션 943, 969, 982
 ERDSA 159
 ESDS
 확장된 주소 지정으로 업그레이드 679
 ESDS(Entry-Sequenced Data Set) 673
 EXCI 102
 옵션 100
 통신 470
 CALL 470
 EXCI - ECI(External Call Interface) 452
 EXCI(External Call Interface) 452
 EXEC 인터페이스 모듈 116, 117, 129, 179
 EXEC 인터페이스 스텝 117, 129
 EXECKEY 294, 315
 EXECKEY 매개변수 789
 EYUVALUE 98

F

FEPI
 참조 5
 FEPI(Front End Programming Interface) 452
 FETCH 54
 FETCHABLE 옵션 69
 FIND 명령, CEBR 트랜잭션 236
 FLAG 옵션 100
 FLOAT 컴파일러 옵션 65
 FMH 745
 아웃바운드 746
 옵션 746
 인바운드 745
 FMHPARM 옵션 969
 FOR 옵션 778
 FORMATTIME 명령 777
 FORMFEED 옵션 883, 969
 FREE 명령 746

FREEKB 옵션 943
 FREEMAIN 명령 788
 FROM 옵션 943
 FRSET 옵션 943
 Function shipping 452
 function shipping 454
 function-ship 709

G

GDDM 946
 GDS 옵션 100
 GENERIC 옵션 677, 699
 GET 명령, CEBR 트랜잭션 237
 GETMAIN 명령 293
 CICSSTACKKEY 옵션 293, 793
 INITIMG 옵션 307, 788
 NOSUSPEND 옵션 788
 SHARED 옵션 292, 299, 788
 TASKSTACKKEY 옵션 293
 USERSTACKKEY 옵션 293, 793
 GETMAIN64 명령
 SHARED 옵션 299
 GOODNIGHT 트랜잭션, CESF 749
 GRAPHIC 옵션 100
 GRPNAME 옵션 927
 GTEQ 옵션 677, 699

H

HANDLE ABEND 434
 HANDLE ABEND 명령 428, 443
 HANDLE AID 명령 961
 HANDLE CONDITION ERROR 명령 432
 HANDLE CONDITION 명령 428, 435
 HFS 파일
 문서 템플릿 826
 HOLD 옵션 364
 HONEOM 옵션 881
 HTML 템플릿
 설치 192
 HTML 템플릿 설치 192

I

IBM Screen Definition Facility II(SDF II) 183
 IBMWRLKC 연계 편집기 입력 67

ICTL(Input Format Control) 어셈블러 명령어 72
 ICVR 매개변수 199
 ID
 VSAM 레코드 677
 IGNORE CONDITION 명령 428, 433
 IGREQID 조건 969
 IGYCCICS 26
 IGZWRLKA 32
 IMMEDIATE 옵션 369, 726, 746
 IMS.RESLIB(IMS 라이브러리) 179
 INBFMH 조건 743
 INITIMG 옵션 307, 788
 INPUTMSG 옵션 364, 370
 INQUIRE TERMINAL 명령 740, 885
 INQUIRE 명령 5
 INRTN 옵션 1007
 INTERVAL 옵션 778
 INVITE 옵션 726
 INVOKE SERVICE 658, 660, 662
 INVPARTN 상태 1009
 INVPARTNSET 상태 1009
 INVREQ 조건 969
 IPCS 450
 IRC 470
 iscics 7
 ISCINVREQ 454
 ISSUE ABORT 명령 752
 ISSUE ADD 명령 751
 ISSUE COPY 명령 728, 894
 ISSUE DISCONNECT 명령 728
 ISSUE END 명령 752
 ISSUE ERASE 명령 728, 752
 ISSUE NOTE 명령 752
 ISSUE PRINT 명령 894
 ISSUE QUERY 명령 751
 ISSUE RECEIVE 명령 751
 ISSUE REPLACE 명령 751
 ISSUE SEND 명령 751
 ISSUE WAIT 명령 752, 753

J

JCICS
 및 채널 350
 JCICS에서 채널 사용 350
 JES 5, 896

JES(작업 항목 서브시스템)
 스플러 명령 898
 입력 900
 종료 900
 JES 스플에서 데이터를 검색 896
 RESP 및 RESP2 옵션 898
 JOURNALNAME 421
 JOURNALNUM 421
 JSON
 언어 구조로 변환 586
 JSON 스키마 602, 607, 613, 616, 622, 627
 JSON으로 변환 579
 JSON 스키마에서 맵핑 생성 644
 JSON 지원 578
 JSON을 애플리케이션 데이터로 변환 647
 JTYPEID 421
 JUSFIRST 옵션 983
 JUSLAST 옵션 983
 JUSTIFY 옵션 983

K

Knowledge Center xi
 Knowledge Center 내용 유형 xi
 KSDS(Key-Sequenced Data Set) 673

L

L8 및 L9 모드 TCB 286, 287
 제한사항 289
 Language Environment 11
 덤프 대상 12
 동적 링크 라이브러리 18
 런타임 옵션 19, 22
 및 CICS LINK 19
 하위 엔클레이브의 19
 CBLPSHPOP 30
 CEEBOXITA 사용자 엑시트 21
 CEECSIX 사용자 엑시트 21
 메시지 대상 12
 어셈블러 언어 81
 언어 혼합 16
 이상 종료 처리 13
 PL/I 67
 저장영역 15
 조건 처리 13
 조건 핸들러, 사용자 작성 13

Language Environment (계속)
 지원 레벨 11
 지원되는 컴파일러 11
 지원, 레벨 11
 호출 가능 서비스 12
 AID 처리 13
 CEEBINT 24
 CICSVAR 환경 변수 22
 DLL 18
 HANDLE AID 명령 13
 HANDLE CONDITION 명령 13
 HLL 사용자 엑시트 24
 Language Environment의 조건 핸들러 13
 PL/I 67
 VS COBOL II 32
 Language Environment 런타임 옵션 19
 STORAGE 26
 Language Environment의 CBLPSHPOP 런타임 옵션 30
 LAST 옵션 746, 943
 브래킷 프로토콜 746
 LDC 1010
 LDCMNEM 옵션 1010
 LDCNUM 옵션 1010
 LEASM 옵션 100
 LENGERR 조건 731
 LENGTH 옵션 100, 300, 731
 LENGTHLIST 옵션
 작업 관련 저장영역의 여러 덤프 450
 LIBRARY 145
 LINE 명령
 CEBR 트랜잭션 237
 LINECOUNT 옵션 100
 LINK
 채널 및 컨테이너로 마이그레이션 361
 LINK PROGRAM 470
 LINK 명령 293, 304, 305, 366
 CHANNEL 옵션 364, 370
 COMMAREA 옵션 364, 367, 369
 IMMEDIATE 옵션 369
 INPUTMSG 옵션 364, 370
 TRANSID 옵션 369
 LINKAGE 옵션 100
 LIST 옵션 988
 LLA (Library Lookaside) 158
 LOAD 명령
 HOLD 옵션 364

LOCKED 연관관계 413
lookaside 트랜잭션 1003
LPA 158
LU 유형 4
 논리 레코드 표시 744
 디바이스 727
 일괄처리 데이터 상호 교환 752
LU(논리 장치)
 기능 743

M

MAPATTS 옵션 940
MAPCOLUMN 옵션 978
MAPFAIL 조건 959, 966
MAPHEIGHT 옵션 978
MAPLINE 옵션 978
MAPONLY 옵션 943, 945, 1018
MAPPED 옵션 986
MAPSET 옵션 943
MAPSET 자원 정의 924
MAPWIDTH 옵션 978
MARGINS 옵션 100
MASSINSERT 옵션 713, 716
maxOccurs
 JSON 스키마에서 634
 XML 스키마에서 538
MDT 959, 1017
MERGE문, COBOL 26
minOccurs
 JSON 스키마에서 634
 XML 스키마에서 538
MONITOR POINT 명령 449
MONITOR 명령 449
MSGINTEG 옵션 755
MSR 1012
MSR 옵션 969, 1012
MVS 서브공간 802
MVS 트랜잭션 372
MVS의 작업 항목 서비스시스템 구성요소 896
MXT 매개변수 310

N

NATLANG 옵션 100
NLEOM 옵션 879, 880, 882, 969
NOAUTOPAGE 옵션 971
NOBLCARD 옵션 100

NOCPSM 옵션 100
NODE 옵션 899
NODEBUG 옵션 100
NOEDF 옵션 100
NOEDIT 옵션 986
NOEPILOG 변환기 옵션 76
NOEPILOG 옵션 100
NOFEPI 옵션 100
NOFLUSH 옵션 974, 982
NOHANDLE 옵션 424, 431
NOJBUFSP 조건 312
NOLENGTH 옵션 100
NOLINKAGE 옵션 100
NONUM 옵션 100
NOOPSEQUENCE 옵션 100
NOOPTIONS 옵션 100
NOPROLOG 변환기 옵션 76
NOPROLOG 옵션 100
NOQUEUE 옵션 312
NOSEQ 옵션 100
NOSEQUENCE 옵션 100, 111
NOSOURCE 옵션 100
NOSPACE 조건 432
NOSPIE 옵션 100
NOSUSPEND 옵션 312
 GETMAIN 명령 788
 READ 명령 699
 READNEXT 명령 711
 READPREV 명령 711
 WRITE 명령 713
NOTRUNCATE 옵션 731
NOVBREF 옵션 100
NOWAIT 옵션 753
NOXREF 옵션 100
NUM 옵션 100
NUMREC 옵션 711
NUMSEGMENTS 옵션
 작업 관련 저장영역의 여러 덩어리 450

O

OCCURS 옵션 929
OO COBOL
 지원 25
OPCLASS 옵션 988
OPENAPI 287
 제한사항 289
OPEN문, COBOL 26

OPID 옵션 988
OPIDENT 값 988
OPMARGINS 옵션 100
OPSEQUENCE 옵션 100
OPSYN(Equate Operation Code) 어셈블러
 명령어 72
OPTIONS 옵션 100
OPTIONS(MAIN) 스펙 65
OTE, 개방형 트랜잭션 환경 278
OVERFLOW 상태 975

P

PA 키 885
PAGENUM 옵션 996
PAGESIZE 값 973
PAGING 옵션 880, 943, 950
PARTITIONSET 옵션 1005
PARTN 옵션 1007
PARTNFAIL 상태 1009
PARTNPAGE 옵션 1008
PERFORM DUMP 명령 449
PERFORM 명령 5
PF(프로그램 기능) 키 233
PLT 프로그램 793
PL/I 65
 다른 언어와 혼합 16
 제한사항 66
 패치된 프로시저 69
 프로그래밍 기술 65, 66, 67, 69
 FLOAT 컴파일러 옵션 65
 JSON 스키마로 맵핑 622, 627
 Language Environment 요구사항 67
 OPTIONS(MAIN) 스펙 65
 XML 스키마로 맵핑 527, 531
PL/I로 맵핑 527, 531, 622, 627
POP HANDLE 명령 428, 435, 443
POST 명령 777, 778
PRGDLY 옵션 994
PRINTERCOMP 옵션 884
PROGRAM 옵션 443
PROGRAM 자원 정의 924
PROLOG 옵션 100
PROTECT 옵션 755
PURGE MESSAGE 명령 970, 987
PURGE 명령, CEBR 트랜잭션 237
PUSH HANDLE 명령 428, 435, 443
PUT 명령, CEBR 트랜잭션 237

Q

QBUSY 조건 312
 QUERY SECURITY 명령 451
 NOLOG 옵션 452
 RESCCLASS 옵션 451
 RESID 옵션 451
 RESTYPE 옵션 451
 QUEUE 명령, CEBR 트랜잭션 237
 QUOTE 옵션 100
 QZERO 조건 809

R

RACF 451
 RBA(Relative Byte Address) 673, 678
 RDF 716
 READ 명령 708
 CONSISTENT 옵션 699
 NOSUSPEND 옵션 699
 REPEATABLE 옵션 699
 UNCOMMITTED 옵션 699
 READNEXT 명령 703
 NOSUSPEND 옵션 711
 READPREV 명령 703
 NOSUSPEND 옵션 711
 READQ TD 명령 312, 807
 READQ TS 명령 813
 ITEM 옵션 813
 READ문, COBOL 26
 RECEIVE MAP 명령 957
 ASIS 옵션 960
 RECEIVE PARTN 명령 1007
 RECEIVE 명령 726, 728, 731, 747, 755
 MAPFAIL 조건 966
 RELEASE 54
 RELEASE 명령
 HOLD 옵션 364
 RELEASE 옵션 971
 RELTYPE 키워드 686
 REMOTENAME 옵션 458
 REMOTESYSTEM 옵션 458
 RENT 속성 159
 REPEATABLE 옵션
 READ 명령 699
 REPLACE문 42
 REQID 옵션 777, 969, 994
 REQUIRED 286

RESCCLASS 옵션 451
 RESETBR 명령 703
 RESID 옵션 451
 RESP 값 423
 RESP 및 RESP2 옵션
 JES에 대한 인터페이스를 위해 898
 RESP 옵션 424, 425, 431
 NOHANDLE 비활성화 430
 RESP2 값 423, 424
 RESP2 옵션 424
 RESTYPE 옵션 451
 RETPAGE 조건 952, 983
 RETRIEVE 명령 777, 783
 RETURN
 채널 및 컨테이너로 마이그레이션 362
 RETURN 명령 294, 369
 CHANNEL 옵션 320, 364
 COMMAREA 옵션 320, 364
 IMMEDIATE 옵션 726, 746
 INPUTMSG 옵션 364, 370
 TRANSID 옵션 269
 REWRITE 명령 708
 REWRITE문, COBOL 26
 RIDFLD 옵션 686, 703
 RMODE 컴파일러 옵션 26
 RMODE(상주 모드)
 CICS 애플리케이션을 위한 옵션 156
 ROUTE 명령 987
 페이지 오버플로우 996
 LIST 옵션 988
 TITLE 옵션 995
 ROUTEDMSGS 옵션 988
 RPTOPTS 19
 RRDS(Relative Record Data Set) 673
 RRN(상대 레코드 번호) 673, 678
 RTEFAIL 조건 991
 RTESOME 조건 991
 RUWAPOL 15
 RU(요청/응답 단위) 743
 RVI 728

S

SAA 자원 복구 421
 SAA 자원 복구 인터페이스 모듈,
 DFHCPLRR 116
 SAM 723
 SCA 653

SCA (계속)
 개요 653
 와이어링 655
 우수 사례 657
 채널 기반 서비스 658
 컴포지트 655
 SCA 컴포지트 작성
 우수 사례 657
 XML 기반 서비스 660
 SCA 문제점 진단 663
 SCA 문제점 해결 663
 SCS
 프린터 882
 SCS 입력 885
 SDF II 916, 925
 SDF II(IBM Screen Definition Facility
 II) 183
 SEC 시스템 초기화 옵션 451
 SEGMENTLIST 옵션
 작업 관련 저장영역의 여러 덩어리 450
 SEND CONTROL 명령 877, 945
 SEND MAP 명령 877, 937
 ACCUM 옵션 968
 ALARM 옵션 943
 CURSOR 옵션 943, 949
 DATAONLY 옵션 943
 ERASE 옵션 943
 ERASEAUP 옵션 943, 982
 FREEKB 옵션 943
 FROM 옵션 943
 LAST 옵션 943
 MAPONLY 옵션 943
 MAPSET 옵션 943
 NOFLUSH 옵션 974, 982
 PAGING 옵션 943, 950
 SET
 명령 950
 SET 옵션 943
 TERMINAL 옵션 943, 950
 WAIT 옵션 943
 SEND PAGE 명령 422, 970, 987
 AUTOPAGE 옵션 971
 NOAUTOPAGE 옵션 971
 RELEASE 옵션 971
 SEND PARTNSET 명령 1005
 SEND TEXT 명령 877, 981
 MAPPED 옵션 986
 NOEDIT 옵션 986

SEND 명령 728, 731, 747, 755
 CNOTCOMPL 옵션 743
 CTLCHAR 옵션 880
 FMH 옵션 746
 INVITE 옵션 726
 LAST 옵션 746
 MSR 옵션 1012
 SEQ 옵션 100
 SEQUENCE 옵션 100
 Service Component Architecture
 개요 653
 진단 663
 SESSBUSY 조건 312
 SET
 명령 5
 SET DOCTEMPLATE NEWCOPY 명령
 822
 SET 옵션 943, 950
 SETLOCALE 54
 SHARED 옵션 292, 299
 GETMAIN 명령 788
 SHARED 옵션 293
 SIGNAL 상태 728
 SOA 653
 SORT문, COBOL 26
 SOURCE 옵션 100
 SP 옵션 100
 SPACE 옵션 100
 SPCOMMAND 자원 유형 451
 SPIE 옵션 100
 SPOLBUSY 조건 900
 SPOOLCLOSE 명령 896
 SPOOLOPEN
 예제 905
 SPOOLOPEN 명령 891, 896
 NODE 옵션 899
 TOKEN 옵션 899
 USERID 옵션 899
 SPOOLREAD 명령 896
 SPOOLWRITE 명령 896
 SQL
 참조 5
 START 데이터, 채널을 사용하기 위해 마이그
 레이션 363
 START 명령 777, 783, 888
 STARTBR 명령 703
 START문, COBOL 26
 STATIC 피연산자 76

STATREG 피연산자 76
 STOP문, COBOL 26
 STORAGE 런타임 옵션, Language
 Environment 26
 STORAGE 옵션 939
 struct, C/370 기호 설명 맵핑 세트 186
 SUSPEND 명령 781
 SVC99 54
 SYNCONRETURN 옵션 458, 466
 SYNCPOINT 명령 421, 422, 970
 ROLLBACK 옵션 443
 SYSEIB 옵션 100
 SYSID 명령, CEBR 트랜잭션 238
 SYSID 옵션 458
 SYSIDERR 454
 SYSIN 97
 SYSPARM, 맵핑 세트를 어셈블링하기 위한
 피연산자 186
 SYSPRINT 98
 SYSPUNCH 97
 SYSSTATE 매크로 76
 SYSTEM 54

T

TASKDATAKEY 옵션 293, 792
 TASKDATALOC 옵션 110, 293
 TCTUA 319, 792
 TCTUAKEY 319, 792
 TCTUALOC 319
 TERM 옵션 934
 TERMID 값 988
 TERMINAL
 옵션 950
 TERMINAL 명령, CEBR 트랜잭션 238
 TERMINAL 옵션 950
 TEST 컴파일러 옵션 26
 TIOATDL 값 987
 TITLE 옵션 995
 TOKEN 옵션 709, 899
 TOP 명령, CEBR 트랜잭션 238
 TRANGRP 410
 TRANISO 800
 TRANSACTION WORK ARER 292
 TRANSFORM DATATOXML 568
 TRANSFORM XMLTODATA 570
 TRANSID 옵션 369, 458
 TRUNC 컴파일러 옵션 26

TWA 292
 TWASIZE 옵션 292
 TYPE=DSECT 어셈블리 924

U

UMT 689
 UNCOMMITTED 옵션
 READ 명령 699
 UNTIL 옵션 778
 UOW 421
 연관관계 413
 UPDATE 옵션 708
 USERDATAKEY 옵션 293, 793
 USERID 옵션 899, 902

V

VBREF 옵션 100
 VLF(Virtual Lookaside Facility) 158
 VOLUME 옵션 752
 VOLUMELENG 옵션 752
 VS COBOL II
 지원 25
 프로그래밍 32
 Language Environment 32
 Language Environment 호출 가능 서비스
 12
 WORKING-STORAGE 제한 26
 VSAM
 데이터 세트 716
 큐에 넣기 309
 프로세서 오버헤드 716
 MASSINSERT 옵션 716

W

WAIT EVENT 명령 394, 777
 WAIT EXTERNAL 명령 394, 781
 WAIT JOURNALNUM 명령
 저널 출력과 동기화 418
 WAIT TERMINAL 명령 728
 WAIT 옵션 419, 420, 728, 943
 WAITCICS 명령 394, 781
 WITH DEBUGGING MODE 26
 WRITE JOURNALNAME 명령 312, 419
 WRITE JOURNALNUM 명령 312, 419
 저널 레코드 작성 418

WRITE 명령 713
 NOSUSPEND 옵션 713
 WRITEQ TD 명령 807
 WRITEQ TS 명령 813
 WRITE문, COBOL 26
 WRKAREA 매개변수 315
 WSDL
 언어 구조로 변환 484
 WSDL 문서
 가변 길이 559
 공백 559

X

X8 및 X9 TCB 62
 XCTL
 채널 및 컨테이너로 마이그레이션 362
 XCTL 명령 293, 294, 304, 305
 CHANNEL 옵션 364, 370
 COMMAREA 옵션 364, 367
 INPUTMSG 옵션 364, 370
 XML
 구문 분석 572
 데이터 유형 572
 데이터로 변환 475
 변환 572
 조회 571
 지원 475
 xsd:any 데이터 유형 573
 XML 구문 분석 572
 XML 기반 서비스 660, 662
 XML 바인딩 563
 XML 변환 572
 XML 스키마 505, 511, 517, 521, 527, 531
 가변 길이 559
 공백 559
 WSDL로 변환 477
 XML 스키마를 애플리케이션 데이터로 매핑
 566
 XML 조회 571
 XML 지원 476
 DFHLS2SC 563
 XML을 2진 데이터로 변환 570
 XOPTS 키워드 111
 XPCFTCH 63
 XPCREQ 글로벌 사용자 엑시트 455, 458
 XPCTA 63
 XPLink 61

XPLink (계속)
 글로벌 사용자 엑시트 63
 비XPLink 오브젝트 63
 TCB 62
 XRBA(확장된 상대 바이트 주소) 678
 XREF 옵션 100
 xsd:any 데이터 유형 573
 XTC OUT 엑시트 1020

Z

z/OS Communications Server 723

[특수 문자]

#echo 명령 832, 833
 #include 명령 833
 #set 명령 818, 832, 833
 + 32K COMMAREA 330, 350
 REQID 옵션 703



Licensed Materials – Property of IBM

SC43-2758-00

