

OpenVMS 사용자 설명서



i n v e n t

2002년 6월

이 설명서에서는 Compaq *OpenVMS* 운영 체제 사용 방법을 설명합니다. 이 설명서에 있는 정보는 모든 OpenVMS 사용자를 위한 것으로 OpenVMS 운영 체제를 실행하는 모든 컴퓨터에 적용할 수 있습니다.

개정/업데이트 정보:

이 설명서는 OpenVMS 사용자 설명서 버전 7.3보다 우선합니다.

소프트웨어 버전:

OpenVMS Alpha Version 7.3-1
OpenVMS VAX Version 7.3

© 2002 Compaq Computer Corporation

Compaq, Compaq 로고, AlphaServer, OpenVMS, POLYCENTER, Tru64, VAX, VMS 및 DIGITAL 로고는 미국 및 기타 여러 나라에서 Compaq Information Technologies Group, L.P.의 등록 상표입니다.

UNIX 및 X/Open은 미국 및/또는 기타 여러 나라에서 The Open Group의 등록 상표입니다. 여기에서 언급한 다른 모든 제품은 해당 기업의 등록 상표일 수 있습니다.

기밀 컴퓨터 소프트웨어(Confidential computer software)인 이 소프트웨어를 소유, 사용 또는 복사하려면 Compaq으로부터 유효한 라이선스를 받아야 합니다. FAR 12.211 및 12.212에 의거하여 상용 컴퓨터 소프트웨어(Commercial Computer Software), 컴퓨터 소프트웨어 문서(Computer Software Documentation) 및 상업 품목을 위한 기술 데이터(Technical Data for Commercial Items)는 벤더의 표준 상업 라이선스로 미국 정부에 사용 허가되었습니다.

Compaq에서는 여기에 포함된 기술적 또는 편집상의 오류나 누락에 대해 책임을 지지 않습니다. 이 문서의 정보는 어떠한 보증도 없이 “있는 그대로” 제공되며 예고 없이 변경될 수 있습니다. Compaq 제품에 대한 보증은 해당 제품과 함께 제공되는 제한적 보증서에 명시되어 있습니다. 이 문서의 어떠한 내용도 이 제한적 보증 외에 어떠한 추가적 보증도 구성하지 않습니다.

ZK6489

Compaq OpenVMS 설명서는 CD-ROM에서 사용할 수 있습니다.

머리말.....	35
대상.....	35
문서 구조	35
관련 문서	38
고객 의견	39
추가 설명서 주문 방법	39
규칙.....	39

1. OpenVMS 운영 체제 시작하기

로그인.....	45
로그인 성공.....	46
로그인 오류.....	47
PC에서 로그인	48
계정의 암호 선택	49
초기 암호 얻기	50
초기 암호 변경	50
암호 제한	50
암호 유형	51
시스템 암호 입력	51
보조 암호 입력	52
여러 유형의 계정에 대한 암호 요구 사항.....	53
정보 메시지 읽기	54
메시지 생략.....	55
로그인 메시지.....	55
로그인 및 로그인 클래스 유형	56
대화형 로그인.....	56
비대화형 로그인	57
로그인 실패.....	58
시스템 암호 요청 터미널	59
로그인 클래스 제한.....	59
이동 제한	59
이동 제한 동안의 일괄 처리 작업.....	59
전화 접속 로그인 실패	60

차례

침입 방지 절차	60
암호 변경	61
사용자 암호 선택	61
작성된 암호 사용	62
작성된 암호: 단점	63
보조 암호 변경	63
로그인 시 암호 변경	64
암호 및 계정 만료 시간	65
암호 만료	65
보조 암호 사용	65
암호 변경 실패	66
계정 만료	66
암호 보호에 대한 지침	67
시스템 응답 인식	69
기본 작업	69
정보 시스템 메시지	69
시스템 오류 메시지	70
현재 프로세스 확인	71
시스템 도움말 보기	72
온라인 도움말 사용	72
특정 명령에 대한 도움말 보기	74
시스템 메시지에 대한 도움말 보기	74
시스템 로그아웃	75
계정 정보 얻기	75
원격 세션 종료	76
네트워크 연결 손실	76
시스템 보안의 손상 없이 로그아웃	77
네트워크	78
네트워크 노드	78
네트워크를 통해 프로그램 실행	78

2. 시스템과 상호 작용하는 DCL 사용

명령 입력	84
사용 모드	84
DCL 명령 유형	84
DCL 명령행	86
구문	87
명령 취소	88
기본값 사용	88
여러 행 명령 입력	89
DCL 명령 입력 규칙	90
매개 변수 입력	91
한정자 입력	92
명령 한정자	92
위치 한정자	92
매개 변수 한정자	93
충돌하는 한정자	93
한정자에서 사용할 수 있는 값	93
날짜 및 시간을 값으로 입력	95
절대 시간 표시 형식	95
변경 시간 표시 형식	96
조합 시간 형식	98
명령 다시 호출	100
Ctrl/B 누르기	100
화살표 키 사용	100
RECALL 명령 사용	100
DCL 명령행 편집	102
SET TERMINAL 명령	102
명령행 일부 삭제	103
터미널 키 정의	104
키 시퀀스	105

3. 파일을 사용하여 정보 저장

파일 이름 및 파일 지정 방법 이해	111
완전한 파일 지정 방법 제공	111
파일 지정 방법의 규칙	112
DCL 명령에서 사용하는 기본 파일 유형	114
언어 소스 프로그램용 기본 파일 유형	115
파일 버전	116
네트워크 노드 이름	116
DECnet-Plus 노드 전체 이름 지정	117
TCP/IP 이름 및 주소 지정	118
DECnet을 사용하여 원격 노드의 파일 액세스	118
TCP/IP를 사용하여 원격 노드의 파일 액세스	119
네트워크 파일 지정 방법 사용	119
액세스 제어 문자열 형식	121
파일 이름과 함께 와일드카드 사용	122
별표(*) 와일드카드 문자	122
백분율 기호(%) 와일드카드 문자	123
기타 파일 이름	124
Null 파일 이름 및 파일 유형	124
자기 테이프의 대체 파일 이름	125
파일 만들기 및 수정	126
파일 만들기	126
파일 복사	126
파일 연결	127
DECnet을 사용하여 원격 노드에서 해당 노드로 파일 복사	127
DECnet을 사용하여 사용자 노드에서 원격 노드로 파일 복사	127
TCP/IP를 사용하여 원격 시스템의 파일 복사	128
액세스 제어 문자열을 사용하여 파일 복사	128
파일 이름 바꾸기	128
파일 내용 표시	129
TYPE 명령 사용	129
표시 제어	129
원격 노드에서 파일 표시	129

와일드카드를 사용하여 파일 표시	130
여러 파일 표시	130
파일 삭제	131
PURGE 명령 사용	132
다른 사용자로부터 파일 보호	133
사용 제어 항목(ACL)	133
보호 유형	133
파일 인쇄	134
인쇄 작업 우선 순위	134
대기열 정보 표시	135
인쇄 양식	135
인쇄 작업 중지	136
다른 노드의 파일 인쇄	136
PRINT 명령 한정자	137
WWPPS 유틸리티(Alpha에만 해당)	139

4. 디렉토리를 사용한 파일 정리

디렉토리 구조	150
디렉토리 이해	152
디렉토리 생성	152
디렉토리 표시	153
디렉토리 삭제	154
기본값 설정	155
존재하지 않는 디렉토리를 기본값으로 설정	155
SHOW DEFAULT 명령	155
임시 기본값 사용	156
다른 사용자로부터 디렉토리 보호	157
와일드카드를 사용하여 디렉토리 구조 검색	158
줄임표 와일드카드 문자	158
하이픈(-) 하위 디렉토리 문자	159
UIC 형식 디렉토리 사용	160
UIC 디렉토리 와 와일드카드 사용	160
UIC 형식을 이름 형식으로 변환	161

5. 확장 파일 지정 방법

ODS-5 볼륨 구조	165
긴 파일 이름	165
파일 이름에 사용할 수 있는 더 많은 유효 문자	166
대/소문자 유지	168
와일드카드 사용	169
중첩 디렉토리 구조	171
디렉토리 이름 지정 구문	171
디렉토리 ID 및 파일 ID 약어	172
DCL의 확장 파일 지정 방법 분석 기능 사용	173
확장 파일 지정 방법을 사용할 수 있는 경우	174
확장 파일 이름 표시	177
DIRECTORY 명령	177
TYPE 명령	179
DELETE 명령	179
PURGE 명령	180
터미널에 확장 파일 이름 표시	181
혼합 환경에서 작업	182

6. 디스크 및 테이프 드라이브 사용

물리적 장치 이름	187
장치 정보 표시	188
논리적 장치 이름	189
일반 장치 이름	190
OpenVMS Cluster 장치 이름	191
볼륨 및 볼륨 집합	192
장치 관리	193
장치 할당	193
볼륨 초기화	194
볼륨 마운트	195
운영자 지원 요청	196
개인용 장치의 파일 액세스	198
볼륨 분리	198

7. Mail을 사용하여 다른 사람과 통신

우편 호출 및 종료.....	204
Mail 호출	204
Mail 종료	204
메시지 읽기.....	205
새 우편 읽기	205
이전 메시지 읽기	206
메시지 검색.....	207
메시지 보내기.....	208
네트워크를 통해 우편 보내기.....	209
네트워크 프로토콜 지정.....	209
노드 이름 지정	210
인터넷 우편 주소 사용	210
논리 노드 이름 사용	211
여러 사용자에게 메시지 보내기.....	212
개별 이름 사용	212
우편 그룹 만들기	212
우편 그룹에 메시지 보내기	214
Mail에서의 파일 조작	215
DDIF 파일 보내기	216
DCL에서 파일 보내기	216
메시지에서 파일 만들기.....	217
메시지에 파일 추가.....	218
메시지를 보내는 기타 방법	219
메시지에 응답.....	219
메시지 전달.....	219
메시지 구성.....	222
폴더 만들기.....	222
우편 하위 디렉토리 만들기	222
폴더로 메시지 이동.....	223
폴더 간의 메시지 복사	223
폴더 선택	223
폴더 삭제	224

차례

우편 파일 만들기 및 액세스	224
우편 메시지 개수 수정	225
메시지 삭제	226
삭제된 메시지 복구	226
우편 메시지 인쇄	227
우편 파일 보호	228
기본 보호	228
보안 단위	228
Mail에서 텍스트 편집기 사용	229
EVE 사용	229
/EDIT 한정자 키워드 사용	229
편집기 선택	230
명령 파일을 사용하여 우편 편집	230
선택한 편집기 무시	231
Mail 키패드 사용	232
키패드 키 재지정	233
추가 키 정의 할당	233
영구 키 정의 만들기	234
Mail 명령 요약	235
메시지 읽기	235
메시지 교환	236
메시지 제거	236
메시지 인쇄	237
메시지 구성	238
메시지 표시	239
Mail 환경 사용자 정의	239
종료 또는 제어 전송	241
우편 파일 압축	242
시스템 관리 명령	242
MIME 유틸리티	243
MIME 유틸리티 호출	243
MIME 유틸리티 초기화	244
선택적인 MIME 유틸리티 파일 만들기	245
MIME 유틸리티를 사용하여 MIME 인코딩 파일 추출	246

MIME 유틸리티를 사용하여 파일 인코딩	247
MIME 유틸리티 명령	248
오류 처리	250

8. EVE를 사용하여 텍스트 파일 편집

EVE 기능	254
도움말 보기	256
키보드 도움말 사용	256
EVE 도움말 사용	257
편집 세션 시작	258
명령 입력	260
명령 입력	260
정의된 키 사용	260
편집 내용 저장 및 EVE 종료	263
WRITE FILE 명령 사용	263
EXIT 명령 사용	263
QUIT 명령 사용	263
커서 이동	265
텍스트 입력	271
텍스트 추가	271
파일 포함	271
인쇄되지 않는 특수 문자	271
텍스트 입력을 위한 EVE 편집 키	272
텍스트 입력을 위한 EVE 명령	272
버퍼 모드 설정	273
텍스트 지우기 및 복원	275
텍스트 이동	279
텍스트 복사	284
상자 편집	285
텍스트 상자 선택	285
텍스트 상자 잘라내기/붙여넣기	286
SET BOX SELECT 명령	288
삭제 보류 사용	289

차례

삭제 보류로 선택 영역 지우기	289
삭제 보류로 지운 선택 영역 복원	289
텍스트 찾기 및 바꾸기	291
텍스트 찾기	292
대/소문자 구분 검색 설정	293
와일드카드 사용	295
검색에 공백 포함	295
텍스트에서 위치 표시	296
텍스트 바꾸기	296
명령행 한정자 사용	299
다른 위치에서 시작	300
작업 파일 사용	300
Main 버퍼 수정	301
EVE를 실행하는 다른 방법	302
검색 목록에서 EVE 실행	302
와일드카드를 사용하여 EVE 실행	302
와일드카드 디렉토리 이름을 사용하여 EVE 실행	303
여러 입력 파일을 사용하여 EVE 실행	303
저널링 및 복구	304
버퍼 변경 저널링 사용	304
EVE 형식화 명령	310
버퍼 사용	314
버퍼 정보 보기	317
버퍼 삭제	317
버퍼 상태 변경	318
Messages 버퍼 표시	319
여러 버퍼 편집	319
EVE로 파일 읽어오기	320
EVE에서 파일 쓰기	320
창 사용	321
특정 버퍼의 섹션 두 개 보기	323
두 개의 버퍼 편집	323
하위 프로세스 만들기	324
생성	324

9. 파일 정렬 및 병합

고성능 Sort/Merge	329
파일 정렬	332
키 정의	333
다중 키 필드	338
동일한 키 필드	339
비문자 데이터	340
출력 파일 조직	341
정렬 프로세스	342
조합 순서 지정	343
일괄 처리 작업으로 Sort 실행	347
명령 프로시저	347
입력 레코드 포함	347
파일 병합	349
정렬된 파일	350
동일한 키 필드	351
터미널에서 레코드 입력	352
Sort/Merge 지정 방법 파일 사용	353
정렬 또는 병합 작업 최적화	360
정렬 프로세스	361
레코드 및 필드 생략	362
작업 파일 할당	362
작업 집합 범위 수정	363
정렬/병합 한정자 요약	364
입력 파일 한정자	369
출력 파일 한정자	370
지정 방법 파일 한정자	374

10. 리소스에 대한 액세스 제어

프로세스의 권한 증명서 표시	390
객체의 보안 프로파일	392
보안 프로파일 수정	392
보호 코드 해석	394

차례

기본 파일 보호	396
기본 UIC 보호	396
기본 ACL 보호	396
파일 이름 바꾸기	397
명시적 파일 보호	397
네트워크에서 파일 액세스	398
액세스 제어 문자열	398
액세스 제어 문자열 보호	398
프록시 로그인 계정을 사용하여 암호 보호	399
일반 액세스 프록시 계정	401
계정 및 파일에 대한 액세스 감사	402
마지막 로그인 시간 보기	402
보안 경보를 트리거할 수 있는 이벤트	403
보안 감사 로그 파일	404
중요한 파일에 ACE 추가	404

11. 장치 및 파일의 논리 이름 정의

논리 이름 특성	407
시스템 정의 논리 이름 사용	408
논리 이름 만들기	409
DEFINE 명령 사용	409
파일 I/O용 명령 프로시저에 논리 이름 만들기	410
논리 이름 만들기 규칙	410
변환 속성	411
액세스 모드	412
논리 노드 이름 만들기	413
같은 객체에 대해 여러 논리 이름 만들기	415
논리 이름 삭제	416
논리 이름 변환	417
반복적 변환	417
시스템 기본값으로 채워진 누락 필드	418
논리 이름 변환의 기본 검색 순서	419
논리 이름 표시	420

검색할 논리 이름 표 지정	420
변환 속성 및 액세스 모드 표시.....	421
검색 목록 만들기 및 사용.....	422
와일드카드를 허용하는 명령과 함께 검색 목록 사용	423
SET DEFAULT 명령과 함께 검색 목록 사용	423
RUN 명령을 사용하여 검색 목록 사용	424
여러 검색 목록의 검색 순서.....	425
논리 이름 표 특성.....	426
논리 이름 표 디렉토리	426
디렉토리 표 구조 표시	427
기본 논리 이름 표.....	428
프로세스 논리 이름 디렉토리.....	429
프로세스 논리 이름 표	430
시스템 논리 이름 디렉토리	431
공유 논리 이름 표.....	433
공유 논리 이름 표의 기본 보호.....	437
공유 논리 이름 관리를 위한 특권 및 액세스 요구 사항.....	438
논리 이름 표 만들기.....	440
프로세스-개인 논리 이름 표 만들기.....	440
공유 논리 이름 표 만들기.....	440
클러스터 전체 논리 이름 표 만들기.....	441
특권 및 액세스 요구 사항	441
기본 보호 수정	442
논리 이름 표의 할당량 설정.....	442
논리 이름 변환 순서 수정	444
논리 이름 표 삭제.....	446
영구 프로세스 논리 이름	447
대화형 프로세스와 일괄 처리 프로세스의 등가 이름 차이점	447
영구 프로세스 논리 이름을 사용하여 파일 I/O 재지정	448

12. 기호, 명령 및 표현식 정의

기호.....	455
논리 이름 및 기호 비교.....	456
기호 사용	457
기호를 사용한 DCL 명령 표시.....	458
기호 약어	458
외부 명령 정의	458
기호 대체	459
기호 삭제	459
기호 표시	460
다른 기호로 기호 사용.....	461
기호 연결	461
문자열 할당에 기호 포함	462
기호를 사용한 데이터 저장 및 조작.....	463
문자열.....	464
문자열 정의.....	465
문자열 표현식.....	465
문자열 연산.....	466
문자열 비교.....	466
부분 문자열 대체	468
숫자 값 및 식 사용	470
숫자 지정	470
숫자의 내부 저장	471
산술 연산 수행	472
숫자 비교	473
숫자 오버레이 수행.....	474
논리 값 및 식 사용	476
논리 연산	476
논리식.....	476
논리 연산 결과	478
렉시칼 함수에서 반환한 값 사용	479
작업 순서	481
데이터 유형 평가	482

표현식에서 값 유형 변환	483
문자열을 정수로 변환	483
정수에서 문자열로 변환	484
기호 표 이해	485
로컬 기호 표	485
글로벌 기호 표	485
기호 표 검색 순서	486
기호 값 마스크	487
SET SYMBOL 명령	487
기호 범위 상태	487
기호 대체 이해	489
강제 기호 대체	490
기호 대체 연산자	491
명령 처리 3 단계	494
1 단계: 명령 입력 스캔	494
2 단계: 명령 구문 분석	494
3 단계: 표현식 평가	494
반복적인 대체	495
정의되지 않은 기호	497
기호 사용에 대한 대안: 자동 외부 명령	499
자동 외부 명령 사용	500
자동 외부 명령 제한	502

13. 명령 프로시저 소개

명령 프로시저 작성을 위한 기본 정보	505
기본 파일 유형	505
명령 작성	505
명령행 작성	505
명령행에서 레이블 사용	507
로컬 기호 테이블의 레이블	507
중복 레이블	507
명령 프로시저에서 주석 사용	508
명령 프로시저 작성 방법	509

차례

명령 프로시저 작성 단계	510
1 단계: 명령 프로시저 디자인	510
2 단계: 변수 할당 및 조건 테스트	512
3 단계: 루프 추가	515
4 단계: 명령 프로시저 종료	517
5 단계: 프로그램 논리 테스트 및 디버그	518
6 단계: 정리 작업 추가	521
7 단계: 명령 프로시저 완료	524
명령 프로시저 실행	526
다른 명령 프로시저에서 명령 프로시저 실행	526
원격 노드에서 명령 프로시저 실행	526
DCL 한정자 또는 매개 변수를 사용하여 명령 프로시저 실행	527
대화형으로 명령 프로시저 실행	529
명령 프로시저를 일괄 처리 작업으로 실행	529
디스크 및 테이프 볼륨에서 명령 프로시저 실행	532
명령 프로시저 종료 및 중단	533
종료 방법	533
종료 처리 루틴	534
오류 처리	535
기본 오류 동작	536
기타 오류 처리 방법	537
ON 명령	537
SET NOON 명령 사용	540
Ctrl/Y 중단 처리	541
명령 프로시저 중지	541
특권 있는 이미지 중지	542
Ctrl/Y 동작 루틴 설정	543
ON 명령 사용	543
Ctrl/Y 입력의 효과	543
Ctrl/Y 중단의 비활성화 및 활성화	547
Ctrl/Y 중단의 비활성화 및 활성화	547
조건 코드를 사용하여 명령 프로시저의 오류 감지	549
조건 코드 표시(\$STATUS)	549
조건 코드와 EXIT 명령	550

보안 수준 결정	551
성공적인 완료 테스트.....	551
\$STATUS를 설정하지 않는 명령 사용	552
로그인 명령 프로시저.....	553
시스템 로그인 명령 프로시저.....	553
개인 로그인 명령 프로시저	553
전용 계정의 로그인 명령 프로시저	554
확장 파일 지정 및 구문 분석 스타일	555
DCL 명령 매개 변수에 확장 파일 이름 사용	556
명령 프로시저 파일 지정	556
대/소문자 보존 및 \$FILE.....	558
앰퍼샌드 및 어포스트로피 대체.....	558

14. DCL을 사용한 고급 프로그래밍

명령 프로시저 입력	563
명령 프로시저에 데이터 포함 시 제한 사항.....	563
그 밖의 데이터 입력 방법.....	564
매개 변수를 사용하여 데이터 전달	565
매개 변수를 정수로 지정	565
매개 변수를 문자열로 지정	565
매개 변수를 기호로 지정	566
매개 변수를 null 값으로 지정	567
매개 변수를 사용하여 일괄 처리 작업에 데이터 전달.....	568
매개 변수를 사용하여 중첩된 명령 프로시저에 데이터 전달	569
데이터 프롬프트 표시.....	570
SYSS\$INPUT 논리 이름을 사용하여 데이터 입력	572
SYSS\$INPUT을 터미널로 재정의.....	572
SYSS\$INPUT을 별개의 파일로 정의	573
명령 프로시저 출력 수행	574
데이터 표시.....	574
명령 및 이미지에서 출력 재지정	575
명령 프로시저에서 데이터 반환.....	578
오류 메시지 재지정.....	579

차례

읽기 및 쓰기 파일(파일 I/O).....	581
OPEN 명령 사용	582
파일에 쓰기.....	585
고유한 이름의 파일 만들기	587
WRITE 명령 사용	588
데이터 지정.....	588
/SYMBOL 한정자 사용	590
/UPDATE 한정자 사용.....	590
READ 명령 사용	591
/END_OF_FILE 한정자 사용	592
/INDEX 및 /KEY 한정자 사용.....	593
/DELETE 한정자 사용.....	593
Close 명령 사용	594
파일 수정	595
레코드 업데이트.....	595
새 출력 파일 만들기.....	597
파일에 레코드 추가.....	600
파일 I/O 오류 처리	601
기본 오류 조치	602
실행 흐름 제어 방법.....	603
IF 명령 사용.....	603
THEN 명령 사용	604
ELSE 명령 사용	604
명령 블록 사용	605
GOTO 명령 사용	609
GOSUB 및 RETURN 명령 사용.....	611
새 명령 수준 만들기.....	613
CALL 명령 사용.....	613
Case 문 작성.....	617
레이블 나열.....	617
Case 문 작성.....	618
명령 블록 작성	619
루프 작성	620
PIPE 명령 사용	624

조건 명령 실행을 위한 PIPE 명령 사용.....	624
파이프라인 실행을 위한 PIPE 명령 사용	624
서브셸 실행을 위한 PIPE 명령 사용	627
백그라운드 실행을 위한 PIPE 명령 사용	627
입/출력 재지정을 위한 PIPE 명령 사용.....	627
PIPE 명령 중단	628
하위 프로세스 성능 향상	629

15. 렉시칼 함수를 사용하여 정보 얻기 및 조작

렉시칼 함수를 사용하는 이유.....	635
사용자 프로세스에 대한 정보 얻기	636
확인 설정 변경	638
기본 파일 보호 변경.....	640
시스템에 대한 정보 얻기	641
OpenVMS Cluster 노드 이름 확인	642
대기열 정보 얻기	642
프로세스 정보 얻기.....	643
F\$CONTEXT 렉시칼 함수	644
파일 및 장치에 대한 정보 얻기.....	646
장치 검색	646
디렉토리에서 파일 검색.....	647
파일의 이전 버전 삭제	647
논리 이름 변환	648
문자열 조작.....	650
문자열이나 문자가 있는지 확인.....	650
문자열의 일부 추출.....	651
출력 문자열 서식 지정	653
데이터 유형 조작	655
데이터 유형 변환	655
표현식 평가.....	656
기호의 존재 여부 확인	656

16. 프로세스 및 일괄 처리 작업의 이해

프로세스 컨텍스트 해석.....	659
분리된 프로세스 사용.....	663
하위 프로세스 사용	664
하위 프로세스를 사용하여 작업 생성.....	664
하위 프로세스를 사용하여 여러 작업 수행	664
하위 프로세스 만들기.....	665
하위 프로세스 종료.....	666
하위 프로세스 컨텍스트.....	667
가상 터미널의 연결이 끊긴 프로세스에 연결	669
터미널 연결 해제	669
연결이 끊긴 프로세스 제거	670
연결이 끊긴 프로세스 관리	670
일괄 처리 작업으로 작업	671
일괄 처리 작업 제출.....	671
일괄 처리 작업에 데이터 전달	674
일괄 처리 작업 출력 제어.....	675
일괄 처리 작업 특성 변경.....	677
SUBMIT 명령 한정자	678
일괄 처리 대기열의 작업 표시	679
일괄 처리 작업 삭제 및 중지	681
일괄 처리 작업 다시 시작.....	681
일괄 처리 작업 실행 동기화.....	682

A. 문자 집합

B. 주식 명령 프로시저

CONVERT.COM 명령 프로시저.....	703
REMINDER.COM 명령 프로시저	708
DIR.COM 명령 프로시저.....	712
SYS.COM 명령 프로시저.....	715
GETPARMS.COM 명령 프로시저.....	718
EDITALL.COM 명령 프로시저.....	721

MALEDIT.COM 명령 프로시저.....	724
FORTUSER.COM 명령 프로시저.....	726
LISTER.COM 명령 프로시저.....	731
CALC.COM 명령 프로시저.....	734
BATCH.COM 명령 프로시저.....	736
COMPILE_FILE.COM 명령 프로시저.....	741
 용어집.....	 745
 색인.....	 759

표 1.	39
표 2.	40
표 1-1.	46
표 1-2.	49
표 1-3.	52
표 1-4.	58
표 1-5.	70
표 1-6.	72
표 2-1.	82
표 2-2.	89
표 2-3.	95
표 2-4.	96
표 2-5.	96
표 2-6.	97
표 2-7.	97
표 2-8.	99
표 2-9. DCL 명령 입력 기능	105
표 2-10. DCL 명령 중단 기능	105
표 2-11. 명령 다시 호출 기능	106
표 2-12. 커서 위치 제어 기능	106
표 2-13. 화면 표시 제어 기능	108
표 3-1.	112
표 3-2.	114
표 3-3.	115
표 3-4.	135
표 3-5.	138
표 3-6.	140
표 3-7.	142
표 3-8. 일반적으로 관련된 언어 코드와 국가 코드	143
표 4-1.	154

표

표 5-1.	167
표 5-2. 와일드카드 및 일치 패턴 예.....	170
표 5-3. ODS-5 볼륨에서의 디렉토리 이름	171
표 5-4. 지원되지 않는 OpenVMS 구성 요소.....	175
표 6-1.	191
표 6-2.	193
표 6-3.	193
표 6-4.	194
표 6-5.	196
표 7-1.	206
표 7-2.	206
표 7-3.	208
표 7-4.	213
표 7-5.	214
표 7-6.	215
표 7-7.	219
표 7-8. MIME 유틸리티 옵션 파일	245
표 8-1. 커서를 이동하는 EVE 편집 키	265
표 8-2. 커서를 이동하는 EVE 명령	266
표 8-3.	272
표 8-4.	272
표 8-5. 텍스트 지우기 및 복원을 위한 EVE 편집 키	275
표 8-6. 텍스트 지우기 및 복원을 위한 EVE 명령	276
표 8-7.	279
표 8-8. 텍스트를 이동하는 EVE 편집 키	280
표 8-9. 텍스트를 이동하는 EVE 명령	281
표 8-10. 상자 편집을 위한 EVE 명령.....	286
표 8-11. SET BOX SELECT 명령	288
표 8-12. 버퍼에서 텍스트를 찾기 위한 EVE 명령	291
표 8-13.	296

표 8-14.	297
표 8-15.	298
표 8-16.	298
표 8-17. EDIT 명령행 한정자	299
표 8-18.	300
표 8-19. 버퍼 변경 저널링 및 복구를 위한 EVE 명령	305
표 8-20. 버퍼 변경 저널 파일 이름.....	306
표 8-21.	307
표 8-22.	308
표 8-23. EVE 편집 키 및 기능.....	310
표 8-24. EVE 텍스트 형식화 명령 및 기능.....	310
표 8-25. 버퍼 조작을 위한 EVE 명령.....	314
표 8-26.	317
표 8-27.	318
표 8-28. EVE 창에서 사용되는 키.....	321
표 8-29. EVE 창 명령	321
표 8-30.	323
표 9-1. 고성능 Sort/Merge : 동작의 차이점.....	330
표 9-2. /KEY 한정자 값	334
표 9-3.	342
표 9-4.	344
표 9-5.	352
표 9-6.	367
표 9-7.	370
표 9-8.	372
표 9-9.	376
표 9-10.	378
표 9-11.	379
표 9-12.	380
표 9-13.	381

표

표 9-14.	382
표 10-1.	394
표 10-2.	395
표 10-3.	403
표 11-1. 기본 논리 이름 표	428
표 11-2. 프로세스 논리 이름 디렉토리	429
표 11-3. 프로세스 논리 이름 표의 기본 논리 이름	430
표 11-4. 시스템 논리 이름 디렉토리의 기본 논리 이름	432
표 11-5. 시스템 논리 이름 표의 기본 논리 이름	435
표 11-6. 공유 논리 이름 표의 기본 보호	437
표 11-7. 공유 논리 이름 작업에 필요한 특권 또는 액세스 유형	438
표 12-1.	456
표 12-2.	464
표 12-3. 문자열 비교	467
표 12-4.	468
표 12-5. 숫자 비교	473
표 12-6.	475
표 12-7.	477
표 12-8.	477
표 12-9.	478
표 12-10.	478
표 12-11.	481
표 12-12.	482
표 12-13.	483
표 12-14.	484
표 12-15.	486
표 13-1.	510
표 13-2.	510
표 13-3.	511
표 13-4.	512

丑 13-5.	512
丑 13-6.	513
丑 13-7.	516
丑 13-8.	517
丑 13-9.	518
丑 13-10.	521
丑 13-11.	521
丑 13-12.	523
丑 13-13.	524
丑 13-14.	531
丑 13-15.	532
丑 13-16.	536
丑 13-17.	537
丑 13-18.	551
丑 13-19.	552
丑 14-1.	570
丑 14-2.	576
丑 14-3.	581
丑 14-4.	585
丑 14-5.	591
丑 14-6.	595
丑 14-7.	597
丑 14-8.	598
丑 14-9.	600
丑 14-10.	610
丑 14-11.	618
丑 14-12.	620
丑 15-1.	636
丑 15-2.	637
丑 15-3.	641

표

표 15-4.	646
표 15-5.	648
표 15-6.	650
표 15-7.	655
표 16-1.	668
표 16-2.	670
표 A-1. DEC 다국적 문자 집합	686
표 A-2. DCL 문자 집합	698

그림 4-1. 디렉토리 구조	150
그림 7-1. 우편 구성	225
그림 7-2. Mail 유틸리티 키패드	232
그림 8-1. EVE 키 — VT200, VT300 및 VT400 계열 터미널.....	261
그림 8-2. EVE 키 — VT100 계열 터미널.....	262
그림 9-1. 오름차순으로 정렬된 목록.....	333
그림 9-2. 목록의 레코드 필드	337
그림 9-3. 키 필드에 따라 정렬.....	337
그림 9-4. 기본 키 레코드로 정렬.....	338
그림 9-5. 다중 키 필드로 정렬.....	339
그림 9-6. 다중 키 필드로 정렬(오름차순 및 내림차순).....	339
그림 9-7. 동일한 키 필드로 정렬.....	340
그림 9-8.	350
그림 9-9.	351
그림 9-10. 지정 방법 파일을 사용한 출력.....	355
그림 13-1. ON 명령 동작.....	539
그림 13-2. Ctrl/Y 동작 다음의 실행 흐름	544
그림 13-3. 중첩된 프로시저에서의 Ctrl/Y	546
그림 16-1. 일괄 처리 작업 실행 동기화.....	683
그림 A-1.	697

머리말

대상

이 설명서는 Compaq *OpenVMS* 운영 체제의 모든 사용자를 위한 것입니다.

시스템 관리자는 효율적인 컴퓨터 작업 환경을 만들고 유지하는 관리 작업을 수행합니다. 따라서 시스템 관리자로서 시스템 관리 개념 및 절차를 이해하려면 *OpenVMS System Manager's Manual*을 참조하십시오.

문서 구조

각 장에서 컴퓨터 작업 수행의 개념 및 절차를 설명합니다. 각 장의 앞부분에는 기본적인 정보가 먼저 제시되고 다소 복잡한 개념 및 절차는 뒤에서 설명합니다.

시작하기

OpenVMS 운영 체제를 사용하여 시작하려면 다음 장을 참조하십시오.

- 1장
OpenVMS 운영 체제 시작하기에서는 시스템 로그인 및 로그아웃 방법, 암호 변경 방법 및 도움말 보는 방법을 설명합니다.
- 2장
시스템과 상호 작용하는 DCL 사용에서는 DCL(DIGITAL Command Language) 사용 방법을 설명합니다.
- 3장
파일을 사용하여 정보 저장에서는 파일에 대해 설명하고 파일을 사용하여 정보를 저장하는 방법을 설명합니다. 그리고 파일 만들기, 복사, 이름 바꾸기, 표시, 삭제, 보호 및 인쇄 보기가 포함되어 있습니다.
- 4장
디렉토리를 사용한 파일 정리에서는 디렉토리를 사용하여 파일을 구성하고 관리하는 방법을 설명합니다.

- 5장

확장 파일 지정 방법에서는 ODS-5를 사용하는 OpenVMS Alpha 시스템에 필요한 확장 파일 지정 방법 환경을 설명합니다.

- 6장

디스크 및 테이프 드라이브 사용에서는 개인적인 용도로 사용하기 위해 테이프나 디스크를 예약하는 방법을 설명합니다. 사용자 그룹에서 공유하는 장치와는 달리 개인 장치는 시스템 관리자가 설치하거나 유지 관리하지 않을 수 있습니다.

- 7장

Mail을 사용하여 다른 사람과 통신에서는 Mail 유틸리티(MAIL)를 사용하여 사용자 시스템 또는 OpenVMS 네트워크용 DECnet을 통해 사용자 시스템에 연결된 다른 컴퓨터에서 다른 사용자와 통신하는 방법을 설명합니다. 그리고 보기 메일 메시지, 메일 메시지 읽기/보내기/회신/전달/구성에 대한 단계별 지시 사항, 메일 명령 요약 및 MIME 유틸리티 사용 방법에 대한 지시 사항이 들어 있습니다.

텍스트 및 레코드 조작

텍스트 파일 편집 및 레코드 정렬에 대한 자세한 내용은 다음 장을 참조하십시오.

- 8장

EVE를 사용하여 텍스트 파일 편집에서는 OpenVMS 운영 체제에 포함된 대화형 텍스트 편집기인 EVE를 설명합니다. 그리고 EVE를 사용하여 새 파일을 만들어 편집하거나 기존 파일을 편집하는 방법을 설명합니다. 또한 EVE 명령이 요약되어 있습니다.

- 9장

파일 정렬 및 병합에서는 정렬/병합 유틸리티(SORT/MERGE)를 사용하여 한 개 이상의 입력 파일에서 레코드를 정렬하는 방법이나 정렬된 파일을 병합하는 방법을 설명합니다. 그리고 Sort/Merge 명령 한정자가 요약되어 있습니다.

보안 확인

보안에 대한 자세한 내용은 다음 장을 참조하십시오.

- 10장

리소스에 대한 액세스 제어에서는 보호된 객체에 대한 액세스 제어 및 원격 시스템의 데이터 액세스 등 일반적인 보안 문제를 설명합니다.

논리 이름 및 기호

논리 이름 및 기호에 대한 자세한 내용은 다음 장을 참조하십시오.

- 11장

장치 및 파일의 논리 이름 정의에서는 파일, 디렉토리 및 장치를 나타내기 위해 논리 이름을 만들고 사용하는 방법을 설명합니다. 또한 시스템에서 만든 논리 이름이 요약되어 있습니다.

- 12장

기호, 명령 및 표현식 정의에서는 기호를 사용하여 명령, 문자열, 숫자 데이터를 나타내는 방법을 설명합니다. 그리고 기호를 표현식에 결합하여 기호가 나타내는 값을 조작하는 방법을 설명합니다.

프로그래밍

프로그램 작성 및 프로그래밍 함수에 대한 자세한 내용은 다음 장을 참조하십시오.

- 13장

명령 프로시저 소개에서는 **DCL** 명령 및 **DCL** 명령에서 사용하는 데이터 행이 들어 있는 파일인 명령 프로시저를 작성하는 데 사용되는 기본 기술을 설명합니다.

- 14장

DCL을 사용한 고급 프로그래밍에서는 명령 프로시저 작성에 사용되는 고급 기술을 설명합니다. 그리고 명령 프로시저 내에서 **PIPE** 명령을 대화형으로 사용하는 방법을 설명합니다.

- 15장

렉시칼 함수를 사용하여 정보 얻기 및 조작에서는 사용자 프로세스나 시스템 환경에 대한 정보를 얻기 위해 명령 프로시저 내에서 렉시칼 함수를 사용하는 방법을 설명합니다.

프로세스 관리

프로세스 관리에 대한 자세한 내용은 다음 장을 참조하십시오.

- 16장

프로세스 및 일괄 처리 작업의 이해에서는 시스템과 상호 작용할 수 있도록 OpenVMS 운영 체제에서 만든 환경인 프로세스를 설명합니다. 그리고 하위 프로세스, 프로그램 및 일괄 작업 사용 방법과 시기를 설명합니다.

참조 단위

다음 정보는 참조용으로 제공됩니다.

- 부록 A

문자 집합에서는 DEC 다국적 문자 집합 및 DCL 문자 집합을 설명합니다.

- 부록 B

주석 명령 프로시저에는 13장, 14장 및 15장에서 설명한 개념 및 기술을 보여주는 완전한 명령 프로시저가 들어 있습니다.

- 용어집

용어집에서는 이 설명서에 사용된 용어 목록을 제공합니다. 용어집 용어가 텍스트에서 처음으로 사용되면 강조 표시됩니다.

관련 문서

OpenVMS 제품 및 서비스에 대한 자세한 내용을 보려면 다음 World Wide Web 주소를 액세스하십시오.

<http://www.openvms.compaq.com/>

고객 의견

Compaq에서는 이 설명서에 대한 귀하의 의견을 환영합니다. 다음 주소 중 한 곳으로 의견을 보내주십시오.

표 1

인터넷	openvmsdoc@compaq.com
우편	Compaq Computer Corporation OSSG Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

추가 설명서 주문 방법

추가 설명서 주문 방법에 대한 자세한 내용은 다음 World Wide Web 주소를 방문하십시오.

<http://www.openvms.compaq.com/>

규칙

이 설명서에서 언급하는 OpenVMS는 Compaq OpenVMS와 같습니다.

VMScluster 시스템은 이제 OpenVMS Cluster 시스템이라고 칭합니다. 특별히 지정하지 않는 한, 이 문서에서 OpenVMS Cluster 또는 클러스터라고 언급하는 것은 VMScluster를 의미합니다.

이 설명서에서 DECwindows 및 DECwindows Motif는 OpenVMS 소프트웨어용 DECwindows Motif를 말합니다.

그리고 다음 규칙도 이 설명서에서 사용됩니다.

표 2

Ctrl/x	Ctrl/x 같은 시퀀스는 다른 키나 포인팅 장치 단추를 누르고 있는 동안 Ctrl이라는 레이블이 붙은 키를 누르고 있어야 한다는 의미입니다.
PF1 x	PF1 x 같은 시퀀스는 먼저 PF1이라는 레이블이 붙은 키를 눌렀다 놓은 다음 다른 키나 포인팅 장치 단추를 눌렀다 놓아야 한다는 의미입니다.
Return	보기에서 키 이름은 키보드에서 키를 눌러야 한다는 의미입니다. 이 문서의 HTML 버전에서는 대괄호로 표시되어 있습니다.
...	보기의 가로 줄임표는 다음 중 하나를 나타냅니다. - 특정 문에 추가되는 선택적 인수를 생략한 경우 - 앞에 나온 항목이 한 번 이상 반복될 수 있는 경우 - 추가 매개 변수, 값 또는 기타 정보가 입력될 수 있는 경우
.	세로 줄임표는 코드 보기나 명령 형식에서 항목을 생략한다는 의미입니다. 설명하고 있는 주제에서 특정 항목이 중요하지 않은 경우 해당 항목을 생략할 수 있습니다.
()	명령 형식 설명에서 괄호는 옵션을 두 개 이상을 선택할 때 괄호로 묶어야 한다는 의미입니다.
[]	명령 형식 설명에서 대괄호는 선택적 요소를 나타냅니다. 옵션을 전혀 선택하지 않거나 한 개를 선택 또는 모두 선택할 수 있습니다. 하지만 OpenVMS 파일 지정 방법의 디렉토리 이름 구문이나 할당문의 부분 문자열 지정 방법 구문에서 대괄호는 선택 사항이 아닙니다.
[[]]	명령 형식 설명에서 대괄호 안의 항목을 분리하는 세로 막대는 옵션을 전혀 선택하지 않거나 하나 이상을 선택할 수 있다는 의미입니다.
{ }	명령 형식 설명에서 중괄호는 필요한 요소를 나타냅니다. 나열된 옵션 중 하나를 선택해야 합니다.

표 2

(계속)

굵은 텍스트	이 텍스트 유형은 새 용어 소개나 인수 이름, 속성 또는 이유를 나타냅니다.
기울임꼴 텍스트	기울임꼴 텍스트는 중요한 정보, 전체 설명서 제목 또는 변수를 나타냅니다. 변수는 시스템 출력(내부 오류 <i>number</i>), 명령행 (<i>PRODUCER=name</i>), 텍스트 명령 매개 변수(여기서 <i>dd</i> 는 장치 유형의 사전 정의된 코드를 나타냄)에 표시될 수 있는 여러 가지 정보를 포함합니다.
대문자 텍스트	대문자 텍스트는 명령, 루틴 이름, 파일 이름 또는 시스템 권한의 약어를 나타냅니다.
고정 폭 텍스트	고정 폭 텍스트는 코드 보기와 대화형 화면 표시를 나타냅니다. C 프로그래밍 언어에서 고정 폭 텍스트는 키워드, 독립적으로 컴파일된 외부 함수 및 파일의 이름, 구문 요약 그리고 보기에 소개된 변수나 증명서에 대한 참조 등의 요소를 식별합니다.
-	명령 형식 설명이나 명령행 또는 코드행의 끝에 있는 하이픈은 명령이나 문이 다음 행에 계속됨을 나타냅니다.
숫자	다른 설명이 없는 한 텍스트의 모든 숫자는 십진수로 처리됩니다. 십진수가 아닌 기수(이진, 8진수 또는 16진수)는 명시적으로 표시됩니다.

머리말

1 OpenVMS 운영 체제 시작하기

OpenVMS는 대화형 가상 메모리 운영 체제입니다. 컴퓨터에 로그인해 있는 동안 사용자와 시스템이 **DCL(DIGITAL Command Language)**을 사용하여 대화할 수 있습니다. 사용자가 DCL을 사용하여 키보드로 명령을 입력하면 시스템이 읽고 번역합니다. 시스템은 해당 명령을 실행하거나 사용자가 입력한 내용을 해석할 수 없는 경우 화면상에 **오류 메시지**를 표시하여 응답합니다. 이 장에서는 OpenVMS 운영 체제와 상호 작용하기 위해 알아야 하는 다음과 같은 기본 정보에 대해 설명합니다.

- 로그인
- PC에서 로그인
- 계정의 암호 선택
- 정보 메시지 읽기
- 로그인 및 로그인 클래스 유형
- 로그인 실패
- 암호 변경
- 암호 및 계정 만료 시간
- 암호 보호에 대한 지침
- 시스템 응답 인식
- 시스템 도움말 보기
- 시스템 로그아웃
- 시스템 보안의 손상 없이 로그아웃
- 네트워크

이 장에 언급된 모든 명령에 대한 전체 설명을 보려면 *OpenVMS DCL Dictionary* 및 온라인 도움말을 참조하십시오.

로그인

로그인은 시스템에 액세스하고 자신을 인증된 사용자로 확인시키는 과정으로 구성됩니다. 로그인하면 시스템은 명령을 입력할 수 있는 환경을 만듭니다. 이런 환경을 **프로세스**라고 합니다.

OpenVMS 운영 체제의 로그인 및 로그아웃 방법은 사이트의 시스템 설정 방법에 따라 다릅니다. 이 절에서는 운영 체제의 로그인 및 로그아웃에 대한 일반적인 설명을 제공합니다. 사이트 특유의 절차에 대해서는 시스템 관리자에게 문의하십시오.

운영 체제와 상호 작용하려면 사용자 **계정**에 로그인해야 합니다. 계정은 사용자가 로그인할 때 시스템에서 사용자를 식별하는 이름이나 숫자입니다. 이 이름이나 숫자를 통해 시스템은 사용자 파일이 저장된 위치와 사용자가 다른 파일에 대해 가지는 액세스 권한을 알게 됩니다.

시스템 관리자(또는 설치 시 시스템 사용을 인증한 사람)는 일반적으로 계정을 설정하고 사용자의 필요에 따라 권한을 부여합니다. 계정에 설정된 액세스 권한 및 사용 권한의 유형에 따라 시스템 성능이나 다른 사용자에게 영향을 줄 수 있는 파일, 이미지 또는 유틸리티를 액세스할 수 있는지 여부가 결정됩니다.

계정을 액세스하려면 사용자 이름과 암호를 입력해야 합니다. 시스템 관리자는 일반적으로 사용자에게 사용자 이름과 초기 암호를 제공합니다. 사용자 이름은 사용자를 시스템에 식별시키고 다른 사용자와 구별합니다. 암호는 사용자를 보호합니다. 암호를 비밀로 유지하면 다른 사용자가 해당 사용자 이름으로 시스템 리소스를 사용할 수 없습니다.

시스템에 로그인하려면 다음 절차를 따릅니다.

표 1-1

단계	작업
1	시스템에서 사용자 이름을 묻는 프롬프트를 표시합니다. Username: 사용자 이름을 입력하고 Enter를 누릅니다. 대략 30초 내에 입력해야 합니다. 그렇지 않으면 시스템에서 시간이 초과됩니다. 시간 초과가 일어나면 로그인 절차를 다시 시작해야 합니다. 시스템에서는 사용자가 입력하는 대로 사용자 이름을 화면에 표시합니다. 예를 들면 다음과 같습니다. Username: CASEY 시스템에서 사용자 암호를 묻는 프롬프트를 표시합니다. Password:
2	암호를 입력하고 Enter를 누릅니다. 시스템은 암호를 표시하지 않습니다. 이를 “에코 없음”이라고 하기도 합니다.
3	시스템 관리자가 사용자 계정을 어떻게 설정했는지에 따라 보조 암호를 입력하거나 자동으로 생성된 암호를 사용해야 할 수 있습니다(51페이지의 “암호 유형” 참조).

로그인 성공

로그인하면 시스템에서 화면의 왼쪽 여백에 달러 기호(\$)를 표시합니다. 달러 기호는 기본 DCL **프롬프트**이며 시스템이 사용할 준비가 된 것을 나타냅니다. 다음 보기에서는 성공적인 로그인을 보여 줍니다.

```
Username: CASEY
Password:
      Welcome to OpenVMS on node MARS
      Last interactive login on Friday, 11-DEC-2002 08:41
      Last non-interactive login on Thursday, 10-DEC-2002 11:05
$
```

로그인 오류

사용자 이름이나 암호를 잘못 입력하거나 암호가 만료된 경우 시스템에서는 User authorization failure라는 메시지를 표시하고 사용자는 로그인되지 않습니다. 실수를 한 경우 **Enter**를 누르고 다시 시도합니다. 암호가 만료된 경우에는 암호를 변경해야 합니다. 시스템은 자동으로 **Set Password:** 프롬프트를 표시합니다. 이런 경우 암호를 변경하는 데 대한 자세한 내용은 61페이지의 “암호 변경”을 참조하십시오. 로그인하는 데 다른 문제가 있는 경우 사용자 계정을 설정한 사람에게 도움을 요청하십시오.

PC에서 로그인

과거에는 모니터와 키보드로 구성된 **비디오 터미널**로 호스트 컴퓨터에 연결했습니다. 모든 컴퓨팅 기능은 OpenVMS 운영 체제를 실행하는 호스트 컴퓨터에 있으며 주로 중앙 컴퓨터실에 설치됩니다. 오늘날은 일련의 독립적인 컴퓨팅 기능을 가진 개인용 컴퓨터(PC)나 워크스테이션에서 작업하는 것이 훨씬 일반적입니다. 이런 경우에는 터미널 에뮬레이션 프로그램을 통해 OpenVMS를 실행하는 호스트 컴퓨터에 연결합니다.

터미널 에뮬레이션 프로그램을 사용하면 TCP/IP 네트워크, 인터넷 또는 인트라넷을 통해 OpenVMS 시스템에 연결할 수 있습니다. 사용자와 운영 체제 간의 상호 작용이 터미널 에뮬레이션 프로그램에서 제공하는 인터페이스를 사용하여 PC 모니터에 표시됩니다. 이런 방식으로 OpenVMS에 연결하려면 터미널 에뮬레이션 프로그램을 시작하고 연결할 시스템을 선택한 다음 이 장에서 설명하는 대로 OpenVMS 운영 체제에 로그인합니다.

계정의 암호 선택

안전한 암호를 선택하려면 다음 지침을 따르십시오.

- 암호에 숫자와 문자를 모두 포함시킵니다. 문자만 포함하는 6자 암호도 상당히 안전하지만 문자와 숫자를 모두 포함하는 6자 암호가 훨씬 안전합니다.
- 6~10자를 포함하는 암호를 선택합니다. 적당한 길이의 암호가 훨씬 안전합니다. 최대 32자의 암호를 선택할 수 있습니다.
- 사전이나 모국어에서 암호를 선택하지 않습니다.
- 제품 이름이나 자신의 자동차 모델과 같이 사용자나 사용자의 컴퓨터 사이트와 쉽게 연관될 수 있는 단어를 선택하지 않습니다.
- 매번 새 암호를 선택합니다. 이전 암호를 다시 사용하지 않습니다.

시스템 관리자나 보안 관리자가 10자 이하의 암호나 반복하는 암호를 허용하지 않는 등의 추가 제한을 설정할 수도 있습니다.

다음 표에서는 안전한 암호와 위험도가 높은 암호(다른 사람이 쉽게 추측할 수 있는 단어)의 예를 보여 줍니다.

표 1-2

안전한 암호	위험도가 높은 암호
의미 없는 음절: aladaskgam eojfuvcue joxtyois	개인과 관련이 많은 단어: 자신의 이름 사랑하는 사람의 이름 애완 동물의 이름 동네 이름 자가용 이름
혼합 문자열: 492_weid \$924spa zu_\$rags	업무와 관련된 용어: 회사 이름 특정 프로젝트 부서 이름

초기 암호 얻기

일반적으로 시스템에서 자신의 계정이 만들어졌음을 알게 될 때 암호가 필요한지 여부를 알게 됩니다. 사용자 암호를 사용할 경우 일반적으로 시스템 관리자가 사용자의 첫 로그인을 위한 특정 암호를 할당합니다. 이 암호는 사용자 계정의 사용 방법에 대한 다른 정보와 함께 시스템 사용자 인증 파일(UAF)에 저장됩니다.

다른 사람이 쉽게 추측할 수 있는 암호는 피하는 것이 좋습니다. 계정을 만드는 사람에게 추천하기 어려운 암호를 지정하도록 요청하십시오. 지정되는 암호에 대한 조정 권한이 없는 경우 자신의 이름과 같은 암호를 지정 받을 수도 있습니다. 그럴 경우 로그인한 후 즉시 암호를 변경하십시오. 이름이나 성을 암호로 사용하는 것은 너무 잘 알려진 방법이므로 보안 측면에서 바람직하지 않습니다.

계정이 만들어질 때 암호의 최소 길이와 새 암호를 선택할 수 있는지 여부 또는 시스템에서 사용자 대신 암호를 생성하는지 여부도 알게 됩니다.

초기 암호 변경

암호를 변경하려면 계정이 만들어진 후 곧바로 계정으로 로그인합니다. 계정이 만들어진 순간에서 일정 시간이 경과한 후 처음 로그인하면 사용자의 계정으로 다른 사용자가 로그인하여 시스템에 손상을 줄 기회를 얻을 수 있습니다. 마찬가지로 암호를 변경하지 않고 그대로 두거나 암호를 변경할 수 없는 경우 시스템은 취약한 상태로 남습니다. 일어날 수 있는 손상은 대체로 사용되는 다른 보안 조치에 따라 다릅니다. 암호 변경에 대한 자세한 내용은 61페이지의 “암호 변경”을 참조하십시오.

암호 제한

시스템은 다음과 같이 암호가 적합한지를 검사합니다.

- 시스템은 자동으로 새 암호를 시스템 사전과 비교합니다. 이렇게 하여 암호가 모국어 단어가 아닌지 확인합니다.
- 이전 암호의 기록 목록을 유지하고 각각의 새 암호를 이 목록과 비교하여 암호를 다시 사용하지 않는지 확인합니다.
- 시스템 관리자가 사용자의 UAF 레코드에 지정하는 최소 암호 길이를 사용하도록 합니다.

시스템에서는 시스템 사전에 있는 암호, 이전에 사용한 암호 및 사용자의 UAF에 지정된 최소 암호 길이보다 짧은 암호는 거부합니다.

암호 유형

OpenVMS 운영 체제에서 인식하는 암호 유형은 다음과 같습니다.

- **사용자 암호**

대부분의 계정에 필요합니다. 사용자 이름을 입력하면 암호를 묻는 프롬프트가 나타납니다. 계정에 기본 및 보조 암호가 모두 필요한 경우 두 가지 암호를 입력해야 합니다.

- **시스템 암호**

특정 터미널에 대한 액세스를 제어하며 보안 관리자의 재량에 따라 요구됩니다. 시스템 암호는 일반적으로 전화 접속 및 공용 터미널 회선과 같이 무단 사용의 대상이 될 수 있는 터미널에 대한 액세스를 제어하기 위해 필요합니다.

- **기본 암호**

기본 및 보조 암호를 모두 요구하는 계정에 대해 입력해야 할 두 암호 중 첫 번째 암호입니다.

- **보조 암호**

기본 및 보조 암호를 모두 요구하는 계정에 대해 입력해야 할 두 암호 중 두 번째 암호입니다. 보조 암호는 사용자 계정에 추가 보안 수준을 제공합니다. 일반적으로 기본 사용자는 보조 암호를 알지 못하므로 이 암호를 제공할 감독자나 다른 핵심 직원이 있어야 합니다. 특정 애플리케이션의 경우 계정이 사용 중인 동안에도 감독자가 여전히 남아 있도록 결정할 수도 있습니다. 따라서 보조 암호를 사용하면 로그인 제어가 쉬워지고 로그인 후 작업 수행을 용이하게 합니다.

보조 암호는 시간을 소모하고 불편할 수 있으므로 최대 보안이 필요한 사이트인 경우에만 적합합니다. 이중 암호가 적합한 계정의 예로는 데이터베이스에 대한 긴급 복구를 허용하기 위해 정상 액세스 제어를 건너뛰는 경우를 들 수 있습니다.

시스템 암호 입력

보안 관리자는 사용자가 사용하도록 지정된 하나 이상의 터미널에 로그인하기 위해 시스템 암호를 지정해야 하는지 여부를 알려 줍니다. 보안 관리자에게 현재 시스템 암호, 변경 빈도 및 변경된 새 시스템 암호를 얻는 방법을 문의하십시오.

시스템 암호를 지정하려면 다음과 같이 합니다.

표 1-3

단계	작업
1	터미널이 인식 문자(일반적으로 벨)로 응답할 때까지 Enter 키를 누릅니다.
2	시스템 암호를 입력하고 Enter 를 누릅니다. 프롬프트가 표시되지 않고 시스템은 사용자가 입력하는 문자를 보여 주지 않습니다. 올바른 시스템 암호를 지정하지 못하더라도 시스템은 그 점을 알려 주지 않습니다. 해당 터미널에 시스템 암호가 필요한지 모를 경우 처음엔 시스템이 제대로 작동하지 않는 것으로 생각할 수 있습니다. 시스템이 응답하지 않는 경우 잘못된 암호를 입력한 것으로 간주하고 다시 시도합니다.
3	올바른 시스템 암호를 입력하면 시스템 알림 메시지(있는 경우)와 Username: 프롬프트가 표시됩니다. 예를 들면 다음과 같습니다. MAPLE - A member of the Forest Cluster Unauthorized Access is Prohibited Username:

보조 암호 입력

보안 관리자는 사용자 계정이 만들어질 때 해당 계정에 보조 암호를 사용해야 하는지 여부를 결정합니다. 사용자 계정이 기본 및 보조 암호를 요구할 경우 로그인할 때 두 암호가 필요합니다. 보안 관리자가 사용자의 **UAF**에 지정하는 최소 암호 길이는 두 암호에 모두 적용됩니다.

단일 암호 로그인에서와 마찬가지로 시스템에서는 전체 로그인에 제한된 시간을 할당합니다. 시간 내에 보조 암호를 입력하지 못하면 로그인 기간이 만료됩니다.

다음 보기에서는 기본 및 보조 암호를 필요로 하는 로그인을 보여 줍니다.

```
WILLOW - A member of the Forest Cluster
Welcome to OpenVMS on node WILLOW

Username: RWOODS
Password: Enter
Password: Enter
Last interactive login on Friday, 11-DEC-2002 10:22
$
```


여러 유형의 계정에 대한 암호 요구 사항

OpenVMS 시스템에서는 다음과 같은 네 가지 유형의 사용자 계정을 사용할 수 있습니다.

- 사용자나 보안 관리자가 정기적으로 변경하는 암호로 보안되는 계정으로 가장 일반적인 계정 유형입니다.
- 항상 암호를 필요로 하지만 사용자가 암호를 변경할 수 없는 계정으로 암호를 잠금(UAF에서 LOCKPWD 플래그 설정)으로써 보안 관리자는 암호에 대한 모든 변경 내용을 제어합니다.
- **제한된 계정**은 사용자의 시스템 사용을 제한하며 때로 암호를 요구하기도 합니다.
- **개방 계정**은 암호를 필요로 하지 않습니다. 개방 계정에 로그인할 경우 시스템은 암호를 묻는 프롬프트를 표시하지 않으며 사용자는 암호를 입력할 필요가 없으며 곧바로 명령을 입력할 수 있습니다. 개방 계정을 사용하면 누구나 시스템에 액세스할 수 있으므로 최소 보안이 요구되는 사이트에서만 사용할 수 있습니다.

정보 메시지 읽기

컴퓨터에 직접 연결된 터미널에서 로그인할 경우 OpenVMS 시스템은 다음 보기와 같은 정보 시스템 메시지를 표시합니다.

```
WILLOW - A member of the Forest Cluster [1]

Unlawful Access is Prohibited

Username: RWOODS
Password:
    You have the following disconnected process: [2]
Terminal Process name Image name
VT320: RWOODS (none)
Connect to above listed process [YES]:NO
    Welcome to OpenVMS on node WILLOW [3]
    Last interactive login on Wednesday, 11-DEC-2002 10:20 [4]
    Last non-interactive login on Monday, 30-NOV-2002 17:39 [5]
    2 failures since last successful login [6]

    You have 1 new mail message.[7]

$
```

위의 보기에서 다음을 확인하십시오.

- [1] 알람 메시지는 노드 및 관련된 OpenVMS Cluster 이름을 식별합니다. 또한 불법 액세스가 금지되어 있음을 무단 사용자에게 경고할 수도 있습니다. 시스템 관리자나 보안 관리자가 이 메시지의 외관과 내용을 조정할 수 있습니다.
- [2] 연결이 끊긴 프로세스 메시지에서는 사용자가 마지막으로 로그인에 성공한 후 특정 시간에 연결이 끊겼지만 여전히 사용할 수 있음을 알려 줍니다. 사용자는 연결이 끊기기 전의 상태로 이전 프로세스에 다시 연결할 수 있습니다.

시스템에서는 다음과 같은 조건에서만 연결이 끊긴 메시지를 표시합니다.

- 중단이 발생한 터미널이 가상 터미널로 설정된 경우
- 사용자 터미널이 연결이 끊길 수 있는 터미널로 설정된 경우
- 최근 세션 동안에 해당 터미널을 통한 중앙 처리 장치(CPU)에 대한 연결이 로그아웃하기 전에 끊어진 경우

일반적으로 보안 관리자는 사용자가 다시 연결할 수 있도록 해야 하는데 이렇게 해도 시스템 보안에 특별한 문제를 야기하지 않기 때문입니다. 그러나 보안 관리자는 터미널의 설정을 변경하고 시스템에서 가상 터미널을 사용할 수 없게 하여 이 기능을 비활성화할 수 있습니다. 가상 터미널을 설정하고 다시 연결하는 데 대한 자세한 내용은 *OpenVMS System Manager's Manual*을 참조하십시오.

- [3] 환영 메시지에서는 실행 중인 OpenVMS 운영 체제의 버전 번호와 로그인한 노드의 이름을 나타냅니다. 시스템 관리자는 다른 메시지를 선택하거나 메시지 전체를 생략할 수도 있습니다.
- [4] 마지막으로 성공한 대화형 로그인 메시지에서는 마지막으로 완료된 로컬, 전화 접속 또는 원격 로그인의 시간을 제공합니다. 시스템은 상위 프로세스가 이런 유형 중의 하나였던 하위 프로세스의 로그인은 계산하지 않습니다.
- [5] 마지막으로 성공한 비대화형 로그인 메시지에서는 마지막으로 완료된 비대화형(일괄 처리 또는 네트워크) 로그인의 시간을 제공합니다.
- [6] 로그인 실패 횟수 메시지에서는 로그인에서 실패한 시도 횟수를 나타냅니다. 틀린 암호만 로그인 실패로 계산됩니다. 사용자의 주의를 끌기 위해 메시지가 표시된 후 벨이 울립니다.
- [7] 새 우편 메시지에서는 읽지 않은 우편 메시지가 있는지 여부를 나타냅니다.

메시지 생략

보안 관리자는 노드 이름과 운영 체제 식별을 포함하는 알람 및 환영 메시지를 생략할 수 있습니다. 로그인 절차는 운영 체제에 따라 다르므로 이런 정보 없이 로그인하면 훨씬 어렵습니다.

마지막 로그인 성공 및 실패 메시지는 옵션입니다. 보안 관리자가 이 메시지를 한꺼번에 활성화 또는 비활성화할 수 있습니다. 중간 수준이나 높은 수준의 보안이 필요한 사이트에서는 이런 메시지에서 침입 시도를 나타낼 수 있으므로 이 메시지를 표시합니다. 또한 이런 메시지는 시스템에서 로그인을 모니터링하고 있다는 것을 나타냄으로써 있을 수 있는 불법 사용자를 억제할 수 있습니다.

로그인 메시지

로그인할 때마다 시스템에서는 성공한 마지막 로그인 및 로그인 실패 횟수에 대한 값을 다시 설정합니다. 계정을 대화형으로 액세스하고 로그인 시도에서 잘못된 암호를 지정하지 않는 경우 성공한 마지막 비대화형 로그인 및 로그인 실패 메시지가 표시되지 않을 수 있습니다.

로그인 및 로그인 클래스 유형

로그인은 대화형이나 비대화형 중 하나가 될 수 있습니다. 대화형으로 로그인할 경우 사용자 이름과 암호를 입력합니다. 비대화형 로그인에서는 시스템이 사용자의 식별과 인증을 수행합니다. 즉, 사용자 이름과 암호를 묻는 메시지를 표시하지 않습니다.

대화형 및 비대화형 로그인과 함께 OpenVMS 운영 체제에서는 여러 로그인 클래스를 인식합니다. 시스템에 로그인한 방법에 따라 사용자가 속하는 **로그인 클래스**가 결정됩니다. 시스템 관리자는 시간 또는 요일뿐만 아니라 사용자의 로그인 클래스에 따라 시스템에 대한 액세스를 제어합니다.

대화형 로그인

대화형 로그인에는 다음과 같은 로그인 클래스를 포함합니다.

- 로컬

중앙 프로세서에 직접 연결된 터미널이나 중앙 터미널과 직접 통신하는 터미널에서 로그인합니다.

- 전화 접속

모뎀과 전화선을 사용하여 컴퓨터 시스템에 연결하는 터미널로 로그인합니다. 시스템에서 사용하는 터미널에 따라 처음에 몇 가지 추가 단계를 실행해야 할 수도 있습니다. 사용자 사이트의 보안 관리자가 필요한 자세한 사항을 제공할 수 있습니다.

- 원격

DCL 명령 **SET HOST**를 입력하여 네트워크를 통해 노드에 로그인합니다. 예를 들어, 원격 노드 **HUBBUB**에 로그인하려면 다음 명령을 입력합니다.

```
$ SET HOST HUBBUB
```

노드 **HUBBUB**의 계정에 액세스할 수 있는 경우 로컬 노드의 해당 계정에 로그인할 수 있습니다. 노드 **HUBBUB**의 장치에 액세스하고 있으면서도 물리적으로는 여전히 로컬 노드에 연결한 채로 남을 수 있습니다.

원격 세션에 대한 자세한 내용은 76페이지의 “원격 세션 종료”를 참조하십시오.

비대화형 로그인

비대화형 로그인은 다음을 포함합니다.

- 네트워크 로그인

사용자가 원격 노드에서 디렉토리의 내용을 표시하거나 디렉토리에 저장된 파일을 다른 노드로 복사하는 것과 같은 네트워크 작업을 시작할 때 시스템에서는 네트워크 로그인을 수행합니다. 사용자의 현재 시스템과 원격 시스템은 같은 네트워크의 노드이어야 합니다. 파일 지정 방법에서 사용자는 대상 노드를 식별하고 원격 노드에 대한 사용자 이름과 암호를 포함하는 액세스 제어 문자열을 제공합니다.

예를 들어, 원격 노드 **PARIS**에 계정이 있는 사용자 **GREG**가 다음 명령을 입력하는 경우 네트워크 로그인이 발생합니다.

```
$ DIRECTORY PARIS"GREG 8G4FR93A"::WORK2:[PUBLIC]*.*;*
```

이 명령은 디스크 **WORK2**의 공용 디렉토리에 있는 모든 파일의 목록을 표시합니다. 또한 암호 **8G4FR93A**도 표시합니다. 노드 **PARIS**의 프록시 계정을 사용하면 더욱 보안된 방법으로 같은 작업을 수행할 수 있습니다. **프록시 로그인**의 예를 보려면 399페이지의 “프록시 로그인 계정을 사용하여 암호 보호”를 참조하십시오.

- 일괄 처리 로그인

사용자가 제출한 일괄 처리 작업이 실행될 때 시스템은 일괄 처리 로그인을 수행합니다. **작업**을 구축하기 위한 인증은 작업이 제출될 때 결정됩니다. 시스템이 작업을 실행할 준비가 되면 작업 컨트롤러에서 사용자 계정에 로그인하는 비대화형 프로세스를 만듭니다. 작업이 로그인할 때 암호가 필요하지 않습니다.

로그인 실패

로그인은 여러 이유로 실패할 수 있습니다. 암호 중 하나가 변경되었거나 계정이 만료되었을 수 있습니다. 그럴 권한이 없는데도 네트워크나 모뎀을 통해 로그인하려 했을 수도 있습니다. 다음 표에서는 일반적인 로그인 실패의 원인을 요약합니다.

표 1-4

실패 표시	원인
터미널이 응답하지 않습니다.	잘못된 터미널, 시스템 암호가 필요한 터미널 또는 전원이 꺼진 터미널입니다.
모든 터미널이 응답하지 않습니다.	시스템이 다운된 상태입니다.
시스템 암호를 입력할 때 터미널이 응답하지 않습니다.	시스템 암호가 변경되었습니다.
시스템 메시지:	
“User authorization failure”	사용자 이름이나 암호의 입력 오류입니다. 계정이나 암호가 만료되었습니다.
“Not authorized to log in from this source”	사용자의 특정 로그인 클래스(로컬, 전화 접속, 원격, 대화형, 일괄 처리 또는 네트워크)가 인증되지 못했습니다.
“Not authorized to log in at this time”	사용자가 현재 시간 또는 오늘 로그인할 수 없습니다.
“User authorization failure”(몇 알 수 없는 사용자 실패 발생)	터미널에서 사용자의 사용자 이름을 사용하여 명백한 침입 시도가 수행되었고 시스템에서 사용자의 사용자 이름을 사용한 해당 터미널의 모든 로그인을 임시적으로 비활성화되었습니다.

다음 절에서 로그인 실패의 원인에 대해 더 자세히 설명합니다.

시스템 암호 요청 터미널

사용하려는 터미널에서 시스템 암호를 요청하는데 이런 요구 사항에 대해 모를 경우 로그인할 수 없습니다. 시스템 암호를 입력하기 전까지는 모든 로그인 시도가 실패합니다.

시스템 암호를 알고 있는 경우 51페이지의 “시스템 암호 입력”에 설명한 단계를 수행합니다. 그래도 로그인할 수 없으면 시스템 암호가 변경되었을 수 있습니다. 시스템 암호를 모르고 있는 것이 문제인 것으로 생각되면 다른 터미널로 로그인하거나 새 시스템 암호를 요청합니다.

로그인 클래스 제한

사용자가 사용자의 UAF 레코드에 금지된 로그인 클래스를 시도하면 로그인이 실패합니다. 예를 들어, 보안 관리자는 사용자가 네트워크를 통해 로그인할 수 없도록 제한할 수 있습니다. 이럴 경우 사용자가 네트워크 로그인을 시도하면 이 소스에서 로그인할 권한이 없다는 메시지가 나타납니다.

보안 관리자는 로컬, 원격, 전화 접속, 일괄 처리 또는 네트워크와 같은 로그인 클래스를 포함하거나 배제하도록 사용자 로그인을 제한할 수 있습니다.

이동 제한

로그인 문제의 또 다른 원인은 사용자의 이동 제한을 지키지 않은 것입니다. 시스템 관리자나 보안 관리자는 시간이나 요일에 따라 시스템에 대한 액세스를 제어할 수 있습니다. 이러한 제한은 로그인 클래스에 대해 부여됩니다. 보안 관리자는 모든 로그인 클래스에 같은 작업 시간 제한을 적용하거나 로그인 클래스에 따라 다른 제한을 부여할 수 있습니다.

해당 로그인 클래스에 대해 금지된 시간에 로그인을 시도하면 로그인이 실패합니다. 시스템은 사용자에게 지금 시간에 로그인할 권한이 없다고 알립니다.

이동 제한 동안의 일괄 처리 작업

이동 제한이 일괄 처리 작업에 적용될 경우 제출 작업 중 허용된 작업 시간 이외의 시간에 예약된 작업은 실행되지 않습니다. 시스템은 이후의 사용할 수 있는 허용된 작업 시간에 그런 작업을 자동으로 다시 제출하지 않습니다. 마찬가지로 작업을 시작하고 허용된 시간을 초과하여 실행하려 하면 작업 컨트롤러에서는 할당된 작업 이동 시간에 도달할 때 완료되지 않은 작업을 중단합니다. 이런 작업 종료 동작은 모든 작업에 적용됩니다.

전화 접속 로그인 실패

보안 관리자는 전화 접속 로그인 동안 연결이 자동으로 끊어지기 전에 올바른 암호를 입력할 수 있도록 사용자에게 제공하는 기회의 수를 제어할 수 있습니다.

로그인이 실패한 뒤 다시 로그인하려면, **Enter** 키를 누릅니다. 성공하거나 한계에 도달할 때까지 계속할 수 있습니다. 연결이 끊어지면 접속 회선에 재접속하여 다시 시작할 수 있습니다.

일반적으로 전화 접속 로그인 실패의 수를 제한하는 이유는 무단 사용자가 시행 착오를 통해 암호를 알아내는 것을 방지하기 위한 것입니다. 무단 사용자에게는 이미 전화 접속 회선의 익명성이라는 이점이 있습니다. 물론 각 전화 접속에 대해 시도 횟수를 제한하더라도 이러한 종류의 침입 시도를 항상 막을 수 있는 것은 아닙니다. 단지 무단 사용자는 다시 연결하여 다른 로그인을 시작하는 것이 요구됩니다.

침입 방지 절차

누군가 사용자의 사용자 이름으로 같은 터미널에 여러 번 실패한 로그인 시도를 하면 시스템은 **침입 시도**가 진행 중인 것처럼 응답합니다. 즉, 시스템에서는 누군가 사용자의 사용자 이름을 사용하여 무단으로 액세스하려고 하는 것으로 판단합니다.

보안 관리자의 결정에 따라 침입 방지 조치는 시스템의 모든 사용자에게 대해 사용될 수 있습니다. 보안 관리자는 특정 시간 동안 허용되는 암호 시도의 수를 제어합니다. 침입 방지 전술이 트리거되면 올바른 암호를 사용하더라도 지정된 시간동안 터미널에 로그인할 수 없습니다. 보안 관리자가 로그인을 다시 시도하려면 얼마나 기다려야 하는지 사용자에게 알려 주거나 사용자가 다른 터미널로 이동하여 로그인을 시도할 수 있습니다.

침입 방지로 로그인할 수 없다고 생각되고 개인적으로 다른 로그인 실패를 경험하지 않았다면 즉시 보안 관리자에게 문의하십시오. 또한 침입 시도에 대한 의심을 입증하거나 부인하기 위해 다른 로그인을 시도하여 마지막 로그인 이후의 로그인 실패 횟수를 보여 주는 메시지를 확인하여야 합니다. 시스템에서 로그인 메시지를 제대로 표시하지 않는 경우 보안 관리자는 인증 유틸리티(AUTHORIZE)를 사용하여 사용자 UAF 레코드의 데이터를 검사할 수 있습니다. 보안 관리자는 신속한 작업을 통해 누군가 다른 터미널에 로그인하려고 하는 것을 알아 낼 수 있습니다.

암호 변경

정기적으로 암호를 변경하면 시스템 보안이 향상됩니다. 암호를 변경하려면 **DCL** 명령 **SET PASSWORD**를 입력합니다.

시스템 관리자는 사용자가 직접 암호를 선택하게 하거나 사용자의 암호를 변경할 때 자동 암호 작성기를 사용하도록 요청할 수 있습니다. 사용자 고유의 암호를 선택하는 경우 암호는 길이와 적합성에 대한 시스템 제한 사항을 따라야 합니다(50페이지의 “암호 제한” 참조).

지정된 기간에 암호를 변경할 수 있는 횟수에는 제한이 없습니다.

다음 보기에서는 너무 짧은 암호를 선택한 경우를 보여 줍니다.

```
$ SET PASSWORD
Old password:
New password:
%SET-F-INVPWDLEN, password length must be between 12 and 32
characters; password not changed
```

사용자 암호 선택

시스템 관리자가 자동 암호 작성기 사용을 요청하지 않는 경우 **SET PASSWORD** 명령을 사용하면 다음과 같이 새 암호를 입력하라는 메시지를 표시한 다음 확인을 위해 새 암호를 다시 입력하라는 메시지를 표시합니다.

```
$ SET PASSWORD
New password:
Verification:
```

새 암호를 똑같이 두 번 입력하지 못하면 암호가 변경되지 않습니다. 위의 두 단계를 마쳐도 알람 메시지는 없습니다. 명령은 사용자 암호를 변경하고 **DCL** 프롬프트로 전환합니다.

보안 관리자가 암호 작성기를 요청하지 않더라도 시스템 보안을 강화하기 위해 암호 작성기를 사용하는 것이 좋습니다.

작성된 암호 사용

시스템 보안 관리자의 결정에 따라 시스템에서 사용자 대신 암호를 자동으로 작성해야 하는 경우 DCL 명령 SET PASSWORD를 입력하면 시스템은 사용자가 선택할 수 있는 암호 목록을 제공합니다. 시스템이 자동으로 작성된 암호를 사용하도록 설정되어 있지 않은 경우 /GENERATE 한점자와 함께 SET PASSWORD 명령을 지정하면 사용할 수 있습니다. 문자 순서가 모국어 단어와 유사하여 기억하기 쉽지만 아주 특이하여 외부인은 추측하기 어렵습니다. 시스템 작성 암호는 길이가 다양하기 때문에 추측하기가 훨씬 더 어렵습니다.

주

암호 작성기는 기본 음절 규칙을 사용하여 단어를 생성하지만 언어에 대한 실제 지식은 없습니다. 따라서 부적절한 단어를 만들 수도 있습니다.

다음 보기에서 시스템은 임의의 순서의 문자로 구성된 암호 목록을 자동으로 작성합니다. 이 보기에서 사용자의 최소 암호 길이는 UAF 레코드에 8자로 설정되어 있습니다.

```
$ SET PASSWORD
Old password: [1]

reankuna      rean-ku-na      [2]
cigtawdpau    cig-tawd-pau
adehecun      a-de-he-cun
ceebatorai    cee-ba-to-rai
arhoajabad    ar-hoa-ja-bad

Choose a password from this list, or press Enter to get a new list [3]
New password: [4]
Verification: [5]
$ [6]
```

이 보기에서 다음을 확인하십시오.

- [1] 사용자는 기존 암호를 올바르게 지정하고 **Enter** 키를 누릅니다.
- [2] 시스템은 8자에서 10자까지 길이를 가진 다섯 개의 선택할 수 있는 암호 목록으로 응답합니다. 일반적으로 발음하기 쉬운 것이 기억하기 쉬우므로 가장 좋습니다.

OpenVMS VAX 시스템에서는 여기서 위에서와 같이 같은 단어를 음절로 나눈 표시 방법이 각 암호 선택 항목의 오른쪽에 표시됩니다.
- [3] 시스템에서는 사용자가 새 암호를 묻는 프롬프트에 대한 응답으로 **Enter** 키를 눌러 새 목록을 요청할 수 있다는 것을 알려 줍니다.
- [4] 사용자는 처음의 다섯 가지 암호에서 하나를 선택하거나 **Enter** 키를 누릅니다.

- [5] 시스템은 이 암호가 자동 암호 작성기에서 제공한 것임을 인식하고 확인 프롬프트를 표시합니다. 사용자는 새 암호를 다시 입력하고 **Enter**를 누릅니다.
- [6] 시스템은 암호를 변경하고 **DCL** 프롬프트로 응답합니다.

작성된 암호: 단점

작성된 암호를 사용하면 다음 두 가지 단점이 있습니다.

- 자신이 선택한 암호를 기억하지 못할 수도 있습니다. 그러나 목록의 모든 암호 선택 항목이 마음에 들지 않거나 기억하기 쉽지 않다고 생각되면 항상 다른 목록을 요청할 수 있습니다.
- 명령이 표시하는 화면에서 암호 선택 항목이 노출될 수 있습니다. 계정을 보호하려면 혼자 암호를 변경하십시오. 비디오 터미널에서 변경할 경우 명령이 완료된 후 화면에서 암호 선택 항목의 표시를 지우십시오. 인쇄 터미널을 사용할 경우 모든 하드카피 출력을 적절하게 제거하십시오.

이런 방법으로 암호를 보호하는 데 실패했음을 나중에 알게 될 경우 즉시 암호를 변경합니다. 계정이 노출된 시간에 대한 사용자의 판단이나 사이트 정책에 따라 사용자의 계정을 통해 보안이 파괴되었을 수 있음을 보안 관리자에게 알려야 합니다.

보조 암호 변경

보조 암호를 변경하려면 **DCL** 명령 **SET PASSWORD/SECONDARY**를 사용합니다. 기본 암호를 변경하는 절차에서와 같이 기존 보조 암호와 새 보조 암호를 입력하라는 메시지가 표시됩니다. 보조 암호를 제거하려면 새 암호와 확인을 위한 프롬프트가 표시될 때 **Enter** 키를 누릅니다.

기본 및 보조 암호를 별도로 변경할 수 있지만 둘 다 같은 암호 유효 기간을 공유하므로 같은 빈도로 변경될 수 있습니다.

로그인 시 암호 변경

현재 암호가 만료되지 않은 경우에도 시스템에 로그인할 때 사용자 이름에 /NEW_PASSWORD 한정자를 포함시켜 암호를 변경할 수 있습니다. 사용자 이름 뒤에 /NEW_PASSWORD 한정자를 입력하면 로그인한 직후 새 암호를 설정하라는 메시지가 표시됩니다. 다음 보기에서는 로그인할 때 암호를 변경하는 방법을 보여 줍니다.

```
WILLOW - A member of the Forest Cluster

Username: RWOODS/NEW_PASSWORD
Password:
    Welcome to OpenVMS on node WILLOW
    Last interactive login on Tuesday, 7-NOV-2002 10:20
    Last non-interactive login on Monday, 6-NOV-2002 14:20

Your password has expired; you must set a new password to log in
New password:
Verification:
```

암호 및 계정 만료 시간

시스템 관리자는 암호나 계정 자체가 특정 날짜 및 시간에 자동으로 만료되도록 사용자 계정을 설정할 수 있습니다. 암호 만료 시간은 사용자가 정기적으로 암호를 변경하도록 시스템 보안을 향상시킵니다. 계정 만료 시간은 계정이 필요한 동안에만 사용될 수 있도록 하는데 도움이 됩니다.

암호 만료

암호 만료 시간이 다가오면 미리 경고 메시지를 받습니다. 이 메시지는 만료 날짜 5일 전에 처음 나타나고 그 후의 로그인 때마다 나타납니다. 이 메시지는 새 우편 메시지 바로 아래에 표시되며 터미널에 벨 소리를 내어 사용자의 주의를 끕니다. 이 메시지에서는 다음과 같이 암호가 만료될 것임을 나타냅니다.

```
WARNING - Your password expires on Thursday 11-DEC-2002 15:00
```

암호가 만료되기 전에 암호를 변경하지 못하면 로그인할 때 다음과 같은 메시지가 표시됩니다.

```
Your password has expired; you must set a new password to log in  
New password:
```

시스템은 새 암호를 입력하라는 메시지를 표시하거나 자동 암호 생성이 설정되어 있는 경우 이 목록에서 새 암호를 선택하라고 요청합니다. **Ctrl/Y**를 눌러 로그인을 중단할 수 있습니다. 다음에 로그인하려 할 때 시스템에서는 암호를 변경하라는 메시지를 다시 표시합니다.

보조 암호 사용

계정에 보조 암호가 사용될 경우(51페이지의 “암호 유형” 참조), 보조 암호는 기본 암호와 같은 시간에 만료됩니다. 두 암호를 모두 변경하라는 메시지가 표시됩니다. 기본 암호를 변경하고 보조 암호를 변경하기 전에 **Ctrl/Y**를 누르면 로그인할 수 없습니다. 시스템은 암호 변경을 기록하지 않습니다.

암호 변경 실패

시스템 관리자가 사용자에게 로그인 시 만료된 암호를 변경하도록 설정하지 않은 경우, 암호가 만료된 후 로그인하면 다음과 같은 최종 경고가 한 번 표시됩니다.

```
WARNING — Your password has expired; update immediately with  
SET PASSWORD!
```

이 상황에서 암호를 변경하지 않거나 변경하기 전에 시스템이 실패하는 경우 다시 로그인할 수 없습니다. 다시 액세스하려면 시스템 관리자에게 문의하십시오.

계정 만료

사용자 계정이 제한된 시간 동안에만 특정 목적을 위해 필요한 경우 사용자 계정을 만드는 사람은 해당 계정이 지속하는 기간을 지정할 수 있습니다. 예를 들어, 대학에서 학생 계정은 일반적으로 한 번에 단일 학기에만 권한이 부여됩니다.

만료 계정은 로그인이 자동으로 거부됩니다. 계정 만료 이전에 경고 메시지를 미리 받지 못하므로 자신의 계정 지속 기간을 사전에 알고 있어야 합니다. 계정 만료 기간은 **UAF** 레코드에 있으며 이 레코드는 **SYSPRV** 권한이나 그와 동등한 권한을 가진 사용자(일반적으로 시스템 관리자나 보안 관리자)만 **OpenVMS** 인증 유틸리티(**AUTHORIZE**)를 사용하여 액세스하고 표시할 수 있습니다.

계정이 만료한 후에 로그인을 시도하면 인증 실패 메시지가 표시됩니다. 연장하려면 사이트에 정의된 절차를 따릅니다.

암호 보호에 대한 지침

올바른 암호 사용을 포함하는 무단 시스템 액세스의 원인에는 비밀리에 발견하는 것보다는 암호의 소유자가 암호를 노출하는 경우가 더 많습니다. 암호를 아무에게도 알려 주지 말아야 합니다.

다음 규칙을 따르면 암호를 가장 잘 보호할 수 있습니다.

- 쉽게 추측할 수 없도록 상당히 긴 암호를 선택합니다. 사전에 있는 모국어 단어를 사용하지 마십시오. 암호에 숫자를 포함시키는 것이 좋습니다. 또는 시스템에서 자동으로 암호를 생성하도록 합니다.
- 암호를 적어 놓지 마십시오.
- 자신의 암호를 다른 사용자에게 제공하지 마십시오. 다른 사용자가 암호를 알게 된 경우 즉시 변경합니다.
- 전자 우편 메시지의 본문과 같은 파일에 암호를 포함시키지 마십시오. 다른 사람이 암호를 알려 주면 신속하게 이 정보를 삭제합니다.

실제 암호와 함께 표시되는 문자열을 통해 다른 사람이 파일에 있는 암호를 쉽게 찾을 수 있습니다. 예를 들어, 두 개의 콜론("::") 뒤에 오는 인용 부호는 항상 액세스 제어 문자열에서 사용자 이름과 암호 뒤에 옵니다. 시스템에 침입하려는 사람이 이 문자열에 대해 부적절하게 보호된 파일을 검색하여 사용자의 암호를 얻을 수 있습니다. 또한 다음과 같이 텍스트 파일에 단어 “password”를 사용하면 암호를 노출시킬 수도 있습니다.

My password is GOBBLEDYGOOK.

- 다른 시스템의 계정에 같은 암호를 사용하지 마십시오.
권한이 없는 사용자가 사용자의 계정이 있는 모든 시스템에서 하나의 암호를 시도할 수 있습니다. 암호를 처음 노출하는 계정에 적은 정보가 들어 있더라도 다른 계정에서 더 많은 정보나 더 많은 권한을 제공할 수 있으므로 결국 훨씬 큰 보안 파괴를 초래할 수 있습니다.
- 이미 켜져 있는 터미널에 로그인하기 전에 사용 가능한 경우 **Break** 키를 눌러 **보안 터미널 서버** 기능을 호출합니다. 이는 특히 공용 터미널실에서 작업할 때 유용합니다.
- 3~6달마다 암호를 변경하십시오. 암호를 공유하지 마십시오. 암호를 공유하려면 매달 변경하십시오.

- 노출된 것으로 의심할 만한 이유가 있는 경우 즉시 암호를 변경합니다. 그런 경우 보안 관리자에게 보고합니다.
- 사용하지 않을 터미널은 로그오프합니다.
권한이 없는 사용자가 암호 절도 프로그램을 로드하는 것과 같은 악의적인 목적으로 이 터미널을 사용할 수 있습니다.
- 마지막 로그인 메시지를 항상 확인합니다. 로그인 실패 횟수에 이상한 것이 있는지 주의를 기울이십시오. 로그인하는 동안 비정상적인 실패를 발견할 경우 암호를 즉시 변경하고 보안 관리자에게 알립니다.

시스템 응답 인식

시스템은 다음과 같은 하나 또는 여러 방법으로 사용자가 입력하는 명령에 응답합니다.

- 명령 실행. 일반적으로 시스템 프롬프트(기본적으로 달러 기호)가 표시되면 명령이 제대로 실행된 것입니다.
- 명령을 실행하고 수행한 작업을 메시지로 사용자에게 알림
- 명령 실행에 실패한 경우 오류에 대해 알림
- 사용자가 제공하지 않은 값(기본값) 제공

기본 작업

기본값은 사용자가 직접 제공하지 않는 경우 운영 체제에서 제공하는 값입니다. 예를 들어, **PRINT** 명령의 한정자로 매수를 지정하지 않으면 시스템은 기본값 1을 사용합니다. 운영 체제는 명령 한정자와 매개 변수를 포함한 여러 영역에서 기본값을 제공합니다. 운영 체제가 특정 명령에서 사용하는 기본값은 *OpenVMS DCL Dictionary*의 각 명령 항목에 설명되어 있습니다.

정보 시스템 메시지

시스템은 일부 명령에 대한 응답으로 시스템이 수행한 작업에 대한 정보를 시스템 메시지에 표시합니다. 예를 들어, **PRINT** 명령을 사용하면 시스템은 인쇄 작업에 할당된 작업 ID 번호와 해당 작업이 입력된 인쇄 대기열의 이름을 표시합니다.

```
$ PRINT MYFILE.LIS
Job MYFILE (queue SCALE_PRINT, entry 210) started on SYS$PRINT
```

모든 명령이 정보 메시지를 표시하지는 않습니다. 명령의 성공적인 완료는 일반적으로 DCL 프롬프트로 이동되는 것으로 표시됩니다. 완료되지 않는 경우 항상 **오류 메시지**가 표시됩니다.

시스템 오류 메시지

명령을 잘못 입력하면 시스템은 다음 보기와 같이 시스템 메시지를 표시하고 올바른 명령 문자열을 입력하라는 메시지를 표시합니다.

```
$ CAPY )
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
\CAPY\
$
```

세 부분으로 나누어진 코드 형식입니다.

DCL-W-IVVERB

각 부분의 의미는 다음과 같습니다.

표 1-5

DCL	오류를 입력한 OpenVMS 장치 또는 구성 요소 이름입니다. 이 보기에서 메시지는 기본 명령 인터프리터 인 DCL에서 온 것입니다.
W	경고를 나타내는 심각도 수준입니다. 다른 심각도 수준에는 S (성공), I (정보), E (오류) 및 F (치명적 또는 심각한 오류)가 있습니다.
IVVERB	메시지 유형입니다. 메시지는 OpenVMS 시스템 메시지의 mnemonic IVVERB 나 74페이지의 “시스템 메시지에 대한 도움말 보기”에 설명된 도움말 메시지 유틸리티(MSGHLP)를 사용하여 식별할 수 있습니다.

시스템에서 사용자가 요청한 기능을 수행할 수 없는 경우 명령 실행 동안 시스템 오류 메시지도 나타날 수 있습니다. 예를 들어, **PRINT** 명령을 올바르게 입력했지만 지정된 파일이 없는 경우 **PRINT** 명령에서는 다음과 같은 메시지로 오류에 대해 알려 줍니다.

```
$ PRINT NOFILE.DAT
%PRINT-E-OPENIN, error opening CLASS1:[MAYMON]NOFILE.DAT; as input
-RMS-E-FNF, file not found
$
```

첫 번째 메시지는 **PRINT** 명령에서 온 것입니다. 이 메시지에서는 지정된 파일을 열 수 없다는 것을 알려 줍니다. 두 번째 메시지는 첫 번째 메시지의 이유 즉, 파일이 없다는 것을 나타냅니다. **RMS**는 OpenVMS 파일 처리 소프트웨어인 레코드 관리 서비스(RMS)를 나타냅니다. 파일 처리와 관련된 오류 메시지는 일반적으로 **OpenVMS RMS** 메시지입니다.

현재 프로세스 확인

프로세스가 제대로 수행되고 있지 않은 것 같으면 **Ctrl/T**를 누릅니다. **Ctrl/T**를 누르면 현재 프로세스에 대한 한 행의 통계 정보가 표시됩니다. 통계 정보에는 노드와 사용자 이름, 현재 시간, 현재 프로세스, **중앙 처리 장치(CPU)** 사용 현황, 페이지 폴트 수, I/O 동작 수준 및 여러 CPU 관련 페이지에 나열되는 **메모리** 사용 현황이 포함됩니다.

대화형 터미널 세션 동안에 **Ctrl/T**를 누르면 현재 명령, 명령 프로시저 또는 이미지를 일시적으로 중단하고 통계를 표시합니다. **Ctrl/T**를 누르면 화면 상의 문자는 중단되지만 프로시저나 편집 세션에는 영향을 주지 않습니다. 예를 들어, **MCCARTHY**라는 사용자가 노드 **GREEN**에서 **EVE** 편집기를 사용하는 중에 **Ctrl/T**를 누르면 **EVE** 메시지 창에 다음 줄이 표시됩니다.

```
GREEN::MCCARTHY 13:45:02 EVE      CPU=00:00:03.33 PF=778 IO=295 MEM=315
```

화면을 새로 고치려면 **Ctrl/W**를 누릅니다.

Ctrl/T는 기본적으로 사용할 수 없습니다. 시스템이 실행 중인데도 **Ctrl/T**에서 통계 정보를 표시하지 않는 경우 **DCL** 명령 **SET CONTROL=T**로 **Ctrl/T**를 사용 가능하도록 설정할 수 있습니다. **DCL** 수준(달러 기호(\$) 프롬프트)에서 명령을 입력한 다음 **Ctrl/T**를 다시 누릅니다. **Ctrl/T**는 프로그램이나 **SET NOCONTROL=T**와 같은 명령으로 사용할 수 없도록 설정하지 않는 한 프로세스 지속 기간 동안 계속 사용됩니다. **Ctrl/T**가 화면에 표시되려면 터미널이 **BROADCAST** 모드로 설정되어 있어야 합니다. **BROADCAST** 모드는 브로드캐스트 메시지(예: **MAIL** 및 **REPLY**로 발행된 메시지)를 수신할 수 있는지 여부를 제어합니다.

터미널을 **BROADCAST** 모드로 설정하려면 **DCL** 프롬프트에서 **DCL** 명령 **SET TERMINAL/BROADCAST**를 입력합니다.

시스템 도움말 보기

운영 체제에 로그인되어 있을 때 **HELP** 명령을 사용하여 시스템 사용과 사용 가능한 명령에 대한 정보를 얻을 수 있습니다. 또한 74페이지의 “시스템 메시지에 대한 도움말 보기”에서와 같이 **HELP/MESSAGE** 명령을 입력하여 시스템 메시지에 대한 도움말을 볼 수 있습니다.

온라인 도움말 사용

OpenVMS 명령 및 유틸리티에 대한 도움말을 보려면 다음 절차를 따릅니다.

표 1-6

단계	작업
1	DCL 프롬프트에서 HELP 를 입력하고 Enter 를 누릅니다. 도움말에서는 항목 목록과 Topic? 프롬프트를 표시합니다.
2	특정 항목에 대한 정보를 보려면 프롬프트 뒤에 항목 이름을 입력하고 Enter 를 누릅니다..
3	하위 항목에 대한 정보를 보려면 프롬프트 위에 항목 이름을 입력하고 Enter 를 누릅니다. 도움말에서 해당 항목에 대한 정보를 표시합니다.
4	SHOW USERS 항목과 하위 항목의 목록을 다시 표시하려면 Subtopic? 프롬프트에 물음표(?)를 입력합니다. 나열된 모든 하위 항목을 보려면 별표(*)를 입력합니다.
5	다른 항목에 대한 정보를 보려면 Enter 를 누릅니다. Topic? 프롬프트가 표시됩니다.
6	도움말을 종료하려면 DCL 프롬프트로 이동될 때까지 Enter 를 누릅니다.

다음 보기에서는 **SHOW USERS** 명령에 대한 도움말을 조회하기 위해 입력하는 명령을 보여 줍니다.

```
$ HELP
HELP
.
. (HELP message text and subtopics)
.

Topic? SHOW USERS

SHOW

USERS

Displays the user name and node name (in a VAXcluster environment)
of interactive, subprocess, and batch users on the system.

Format

SHOW USERS [username]

Additional information available:

PARAMETER QUALIFIER
/BATCH      /CLUSTER  /FULL        /INTERACTIVE  /NETWORK  /NODE
/OUTPUT     /SUBPROCESS
Examples

SHOW USERS Subtopic? EXAMPLES

SHOW

USERS

Examples
.
. (SHOW USERS Examples message text and subtopics, if any)
.
SHOW USERS Subtopic?
SHOW Subtopic?
Topic?
$
```

특정 명령에 대한 도움말 보기

정보가 필요한 명령이 있으면 **HELP**와 그 명령 이름을 입력합니다. 예를 들어, **SHOW USERS** 명령에 대한 도움말을 보려면 다음 명령을 입력합니다.

```
$ HELP SHOW USERS
```

명령 이름이나 지정할 시스템 항목을 모를 경우 매개 변수로 **HINTS**라는 단어와 함께 **HELP** 명령을 입력합니다. **HINTS** 텍스트에 나열된 각 작업 이름은 관련 명령 이름이나 시스템 정보 항목의 목록과 연관이 있습니다.

*OpenVMS DCL Dictionary*에 **HELP** 명령에 대한 자세한 내용이 있습니다.

시스템 메시지에 대한 도움말 보기

시스템 메시지에 대한 온라인 도움말을 보려면 도움말 메시지 유틸리티(**MSGHLP**)를 사용합니다. 마지막 명령의 실행에 대한 정보를 표시하려면 다음을 입력합니다.

```
$ HELP/MESSAGE
```

메시지 증명서나 메시지 텍스트의 단어를 포함시켜 특정 메시지에 대한 정보를 표시할 수도 있습니다. 예를 들면 다음과 같습니다.

```
$ HELP/MESSAGE BADACP
```

메시지 상태 코드를 입력하여 메시지 및 그에 대한 설명도 액세스할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ HELP/MESSAGE/STATUS=%X00038090
```

메시지 상태 코드를 모를 경우 다음과 같이 **SHOW SYMBOL** 뒤에 **\$STATUS** 글로벌 기호를 입력하면 메시지 상태 코드를 볼 수 있습니다.

```
$ SHOW SYMBOL $STATUS
$STATUS == "%X00038090"
```

도움말 메시지 유틸리티를 사용하면 메시지 데이터베이스를 사용자 메시지로 업데이트하거나 기존 메시지 설명에 주석을 추가할 수 있습니다. 메시지 데이터베이스에서 메시지 하위 집합을 추출하여 사용자 정의 메시지 문서를 만들고 인쇄할 수도 있습니다. 도움말 메시지 유틸리티 사용에 대한 자세한 내용은 *OpenVMS System Messages: Companion Guide for Help Message Users*를 참조하십시오.

시스템 로그아웃

시스템 사용을 마쳤으면 항상 **로그아웃**합니다. 이렇게 하면 무단 사용자가 사용자의 계정과 시스템에 액세스하는 것을 방지합니다. 또한 더 이상 필요하지 않은 리소스를 다른 사용자가 사용할 수 있으므로 시스템 리소스를 합리적으로 사용하도록 합니다.

로그아웃하려면 다음과 같이 **DCL 프롬프트**에서 **LOGOUT**을 입력합니다.

```
$ LOGOUT
```

시스템은 다음 메시지와 같이 시스템을 로그아웃되었음을 확인하는 메시지를 표시합니다.

```
$ LOGOUT
HARRIS logged out at 11-DEC-2002 12:42:48.12
```

로그아웃하려면 **DCL 프롬프트(\$)**에 있어야 합니다. 프로그램을 컴파일 또는 실행하거나 텍스트 편집기(예: **EDT** 또는 **EVE**)를 사용하거나 유틸리티(예: **Mail**)를 실행하고 있는 동안에는 **LOGOUT** 명령을 입력할 수 없습니다. 먼저 프로그램, 편집기 또는 유틸리티를 종료해야 합니다. 시스템에서 **DCL 프롬프트**를 표시하면 로그아웃할 수 있습니다.

계정 정보 얻기

터미널 사용 시간(경과된 시간), 사용한 컴퓨터 시간(CPU 사용 시간) 및 기타 계정 정보를 보려면 다음과 같이 **DCL 프롬프트**에서 **LOGOUT/FULL**을 입력합니다.

```
$ LOGOUT/FULL
```

시스템에서는 다음과 같은 정보를 표시합니다.

```
SIMPSON logged out at 11-DEC-2002 12:42:48.12
```

```
Accounting information:
```

Buffered I/O count:	8005	Peak working set size:	212
Direct I/O count:	504	Peak virtual size:	770
Page faults:	1476	Mounted volumes:	0
Charged CPU time:	0 00:00:50.01	Elapsed time:	0 02:27:43.06

원격 세션 종료

다음 두 가지 방법으로 원격 세션을 종료할 수 있습니다.

- 원격 시스템의 로그아웃 절차(예: OpenVMS 시스템인 경우 LOGOUT 명령 사용)를 사용합니다.
- **Ctrl/Y**를 두 번 누르면 호스트 시스템에서 원격 세션을 중단할지 묻는 프롬프트가 표시됩니다. 원격 세션을 중단하려면 **YES (Y)**를 누릅니다. 이 방법은 원격 노드에 실행 중인 시스템 유형과 관계없이 사용할 수 있습니다.

원격 세션을 종료하면 시스템은 “%REM-S-END, control Entered to node NODENAME:”라는 메시지를 표시하고 원격 노드에 연결하는 시스템 프로세스로 들어갑니다.

네트워크 연결 손실

원격 시스템에 대한 **TCP/IP** 네트워크 연결이 손실되면 **TCP/IP**에서는 회선 장애와 같은 오류가 있는 경우 복구하지 않고 데이터를 배달하려고 하는 네트워크 기술의 특징인 **최적 배달** 프로토콜을 사용합니다.

원격 시스템에 대한 **DECnet** 네트워크 연결이 손실되면 **DECnet**에서는 통신을 다시 설정하려고 할 때 사용자 데이터를 다시 전송합니다. **DECnet**에서 미리 정의된 시간 초과 기간 내에 통신을 다시 설정할 수 없으면 원격 시스템 연결이 종료되고 “Path lost to partner”라는 메시지를 표시합니다.

시스템 보안의 손상 없이 로그아웃

세션을 로그아웃하면 시스템 리소스를 보존하고 사용자 파일을 보호합니다. 터미널을 온라인 상태로 두는 것은 내부 침입을 초래하는 가장 큰 원인 중의 하나입니다. 터미널이 온라인 상태이고 사무실이 개방되어 있으면 결국 암호와 권한을 도난 당하게 되고 자신과 그룹의 다른 사람들의 파일을 보호할 수 없게 되기 쉽습니다. 사용자 계정을 통해 액세스할 수 있는 모든 파일은 누구나 쉽게 전송할 수 있습니다. 내부자가 악의적으로 사용자 파일과 사용자에게 쓰기 권한이 있는 다른 모든 파일의 이름을 바꾸거나 삭제할 수 있습니다. 사용자에게 특히 파일 또는 모든 범주에 대한 권한과 같이 특수한 권한이 있는 경우 사용자가 악의적으로 중대한 손상을 줄 수 있습니다.

작업하는 시스템이 지정된 기간 동안 작동하지 않으면 자동으로 잠기는 시스템이 아닌 경우 잠시라도 사무실을 떠날 때는 로그아웃하는 것이 좋습니다. 원격 로그인을 수행한 경우 각 노드를 로그아웃해야 합니다.

보안 관리자가 사용자에게 로그아웃할 때 전화 접속 회선 연결을 끊도록 요청할 수도 있습니다. 전화 접속 회선 연결을 끊는 것은 다음과 같은 이점이 있습니다.

- 다른 사람이 열린 액세스 회선을 이용할 수 없습니다. 다른 사람이 회선에 액세스하려면 액세스 번호를 알아야 하며 직접 다시 접속해야 합니다.
- 사용하는 전화 접속 회선이 공용 장소에 있거나 다른 사람이 뒤이어 터미널을 사용할 수 있는 장소에 있는 경우에 특히 중요합니다.
- 필요한 전화 접속 회선 수를 줄여 리소스를 절약합니다.

네트워크

컴퓨터 시스템이 서로 연결될 때 **네트워크**를 형성합니다. OpenVMS 네트워크의 운영 체제는 서로 통신하고 정보와 리소스를 공유할 수 있습니다. 네트워크의 각 시스템은 네트워크 **노드** 또는 **호스트**라고 하고 고유한 이름이나 주소로 식별됩니다. **호스트**와 **노드**는 번갈아 사용할 수 있으며 네트워크에 연결된 시스템을 의미합니다.

OpenVMS에서는 네트워크 프로토콜을 선택할 수 있습니다. 단일 네트워크에서 Compaq OpenVMS용 TCP/IP 서비스 제품이나 Compaq DECnet 제품을 사용할 수도 있고 두 제품 모두 사용하는 환경을 만들 수도 있습니다. Compaq의 OpenVMS를 위한 기본 네트워크 전략은 산업 표준 네트워크 프로토콜 제품군인 TCP/IP입니다.

네트워크 노드

네트워크 노드에 로그인하면 네트워크의 다른 노드와 통신할 수 있습니다. 로그인한 노드는 **로컬 노드**라고 하고 네트워크의 다른 노드는 **원격 노드**라고 합니다. 원격 노드의 계정에 액세스할 수 있으면 로컬 노드에 연결된 상태로 로컬 노드에서 원격 노드의 해당 계정에 로그인하여 작업을 수행할 수 있습니다.

57페이지의 “비대화형 로그인”에서 원격 노드에 로그인하는 방법을 설명합니다. 원격 노드에서 수행할 수 있는 추가 작업은 이 설명서의 해당 장에서 설명합니다.

네트워크를 통해 프로그램 실행

TCP/IP 및 DECnet 소프트웨어에서 제공하는 지원을 바탕으로 **프로그램**을 로컬에서 실행하는 것처럼 네트워크를 통해 실행할 수 있습니다. 네트워크 소프트웨어는 운영 체제 내에 통합되어 있으므로 원격 파일을 액세스하는 프로그램을 쉽게 작성할 수 있습니다. 어플리케이션에서 원격 파일을 액세스하려면 파일 지정 방법에 원격 노드의 이름과 필요한 액세스 제어 정보를 포함하기만 하면 됩니다.

모든 TCP/IP 또는 DECnet 구현의 공통적인 기능인 작업 간 통신을 사용하면, 같거나 다른 운영 체제에서 실행되고 있는 두 어플리케이션이 사용된 프로그램 언어와 관계 없이 서로 통신할 수 있습니다. 네트워크 어플리케이션의 예로는 분산 처리 어플리케이션, 트랜잭션 처리 어플리케이션, 서버에 대한 연결을 제공하는 어플리케이션 등을 들 수 있습니다.

주

이 설명서의 원격 작업에 대한 예에서 프록시 계정은 사용자가 원격 시스템에서 작업을 수행할 수 있도록 합니다. 프록시 계정은 사용자가 원격 시스템에 액세스할 수 있는 방법 중 하나입니다. 원격 시스템 액세스 방법에 대한 자세한 내용은 *OpenVMS System Manager's Manual*을 참조하십시오.

OpenVMS 운영 체제 시작하기

네트워크

2 시스템과 상호 작용하는 DCL 사용

DCL(DIGITAL Command Language)은 운영 체제가 특정 작업을 수행하도록 지시하는 일련의 영어와 유사한 지시 사항입니다. DCL은 작업 수행을 위해 운영 체제와 상호 작용하는데 사용되는 200여개의 명령과 기능을 제공합니다. DCL 명령을 사용하여 다음과 같은 작업을 수행할 수 있습니다.

- 시스템에 관한 정보 얻기
- 파일 작업
- 디스크, 자기 테이프 및 기타 장치 작업
- 작업 환경 수정
- 프로그램 개발 및 실행
- 보안 제공 및 효율적인 자원 사용 보장

다음 표에서는 몇 가지 일반적인 컴퓨터 작업 수행에 사용되는 DCL 명령이 나열되어 있습니다.

표 2-1

명령	작업
COPY	지정된 파일의 복사본 만들기
COPY/FTP	TCP/IP 네트워크를 통한 호스트 간의 파일 전송
CREATE	파일 또는 디렉토리 만들기
DELETE	지정된 파일을 지우고 디렉토리에서 제거
DIRECTORY	디렉토리 내용(파일 목록) 표시
EDIT	텍스트 파일 내용 보기 및 변경
LOGOUT	세션 종료
PRINT	지정된 파일을 인쇄를 위해 프린터로 보내기
RENAME	지정된 파일의 이름 또는 위치 변경
SET	화면 상의 시스템 표시 방법 제어
SHOW	시스템의 상태 표시
TYPE	지정된 파일 내용을 화면에 표시

이 장에서는 DCL 사용 방법에 대해 설명하며 포함된 정보는 다음과 같습니다.

- DCL 명령 입력
- DCL 명령행
- DCL 명령 입력 규칙
- 매개 변수 입력
- 한정자 입력
- 날짜 및 시간을 값으로 입력
- 명령 다시 호출
- DCL 명령행 편집
- 터미널 키 정의
- 키 시퀀스

사용자 로컬 환경의 차이

이 설명서에서는 표준 DCL 명령만 다룹니다. 사이트의 시스템 관리자가 로컬 환경을 지원하기 위해 다음과 같이 시스템을 사용자 정의할 수 있습니다.

- 다른 명령 인터프리터 사용
- 일부 표준 DCL 명령의 기본 작업 변경
- 일부 DCL 명령 사용 불가능
- DCL 프롬프트 등의 일부 시스템 기본값 변경
- **확장 파일 지정 방법**을 사용하여 환경 구성

이 장에서 설명하는 명령, 한정자, 매개 변수에 대한 추가 정보는 *OpenVMS DCL Dictionary*와 온라인 도움말을 참조하십시오.

명령 입력

DCL 명령을 입력하려면 DCL 프롬프트(\$)에서 명령을 입력하고 **Enter** 키를 누릅니다. DCL은 일반적으로 대/소문자를 구분하지 않으므로 대문자 또는 소문자로 명령을 입력할 수 있습니다.¹

다음 보기에서는 DCL 명령인 **SHOW TIME**을 다음과 같이 입력합니다.

```
$ SHOW TIME
```

현재 날짜와 시간이 표시되고 DCL 다른 명령을 입력할 준비할 수 있는 상태를 나타내는 프롬프트가 다시 나타납니다.

```
11-DEC-2002 15:41:43  
$
```

사용 모드

DCL을 다음 두 가지 모드로 사용할 수 있습니다.

- 대화형

대화형 모드는 터미널에서 명령을 입력합니다. 한 명령의 실행이 끝나야 다른 명령을 입력할 수 있습니다.

- 일괄 처리

일괄 처리 모드는 시스템에서 명령 실행을 위해 프로세스를 만듭니다. **일괄 처리 작업**은 개별 사용자 프로세스로서 운영 체제에 실행을 위해 제출되는 명령 프로시저 또는 프로그램입니다. 일괄 처리 실행을 위한 명령 프로시저를 제출한 후 계속해서 터미널을 대화형으로 사용할 수 있습니다.

일괄 처리 작업과 네트워크 프로세스는 일괄 처리 모드에서 DCL을 사용합니다. 프로세스에 대한 정보는 16장을 참조하십시오.

DCL 명령 유형

DCL 명령을 입력하면 DCL 인터프리터가 명령을 읽어들이어 변환합니다. 명령 인터프리터가 명령에 응답하는 방식은 입력된 명령의 유형에 의해 결정됩니다. 다음과 같은 세 가지 유형의 DCL 명령이 있습니다.

1. 대/소문자 구분에 대한 내용은 5장을 참조하십시오.

- 내장된 명령

이러한 유형의 DCL 명령은 DCL 인터프리터에 내장되어 있으며 내부에서 실행됩니다.

- 프로그램 호출 명령

이 유형의 DCL 명령은 명령을 내부에서 실행하기 보다는 명령을 실행하는 다른 프로그램을 호출합니다. 명령 실행을 위해 호출된 프로그램을 **명령 이미지**라고 합니다. 명령 이미지는 대화형 프로그램인 **유틸리티**이거나(예: Mail), 비대화형 프로그램(예: COPY)일 수 있습니다.

- 외부 명령

이미지를 실행하는 기호를 **외부 명령**이라고 합니다. 외부 명령은 명령 인터프리터에서 DCL 명령으로 인식하지 않는 이름을 가진 이미지들을 실행합니다. 기호에 대한 자세한 내용은 12장을 참조하십시오.

DCL 명령행

DCL에는 다른 언어와 마찬가지로 사용 규칙과 어휘가 있습니다. DCL은 단어(어휘)와 단어 순서(구문 또는 형식)로 구성됩니다. 다음 절에서는 이 두 요소와 유효한 DCL 명령 구성 방법에 대해 설명합니다.

다음 보기에서는 DCL 명령행의 일반 형식과 부분들을 보여 줍니다.

```
$ PRINT/COPIES = 5  GROCERY.LIS  Enter
[1]  [2]  [3]      [4]      [5]      [6]
```

다음 목록에서는 DCL 명령행의 각 요소를 설명합니다.

[1] DCL 프롬프트

기본 DCL 프롬프트는 달러 기호(\$)입니다. 대화형으로 DCL을 사용하는 경우 명령을 입력할 수 있는 상태가 되면 프롬프트가 표시됩니다.

[2] DCL 명령

이 부분은 명령의 이름을 지정합니다. 내장된 명령, 프로그램 호출 명령 또는 외부 명령을 사용할 수 있습니다. 이 보기에서 DCL 명령은 PRINT입니다.

[3] 한정자

한정자는 명령 작업을 수정합니다. 일부 한정자는 명령 전체를 수정하고, 어떤 한정자는 특정 **명령 매개 변수**를 수정합니다. 어떤 한정자는 값을 받아들이 수 있습니다. 한정자 앞에는 항상 슬래시(/)를 붙입니다. 이 보기에서는 한정자는 /COPIES입니다.

[4] 값

값은 한정자를 수정하며 등호(=)를 앞에 붙이는 경우가 많습니다. 파일 지정 방법, 문자열, 숫자 또는 **DCL 키워드**를 값으로 사용할 수 있습니다. 키워드는 지정된 형식으로 사용하도록 예약된 단어입니다.

이 보기에서는 값은 5(5개의 복사본)입니다.

[5] 매개 변수

매개 변수는 명령 작업 대상을 지정합니다. 명령 내에서 지정된 순서대로 매개 변수를 사용해야 합니다. 매개 변수 값으로는 파일 지정 방법, 대기열 이름, 논리 이름 등이 있습니다.

[6] Enter 키

Enter 키는 DCL 명령행을 종료하고 명령을 처리할 준비가 되었음을 시스템에 알립니다.

다음 항목들도 DCL 명령행에 사용할 수 있습니다.

- 레이블

레이블은 명령 프로시저의 행을 식별합니다. 13장과 14장에 설명된 명령 프로시저 내에서만 레이블을 사용하십시오.

- 키워드

키워드는 지정된 형식에서만 사용하도록 정의된 단어입니다. 지정하려는 특정 DCL 명령 설명에 나열된 대로 키워드를 사용해야 합니다. 예를 들어, **system**, **owner**, **group**, **world**는 SET SECURITY 명령의 /PROTECTION 한정자를 위한 DCL 키워드입니다. DCL 키워드도 값을 가질 수 있습니다.

- 와일드카드 문자

와일드카드 문자는 별표(*), 백분율 기호(%), 줄임표(...) 및 하이픈(-)입니다. 와일드카드는 파일 이름, 파일 유형 디렉토리 이름 또는 파일 지정 방법의 버전 번호 안이나 해당 위치에서 사용하여 주어진 필드에 대해 모두를 나타낼 수 있습니다. 파일 및 디렉토리의 와일드카드 사용에 대한 내용은 3장, 4장 및 5장을 참조하십시오.

구문

말을 할 때 단어 순서에 따라 의미가 결정되는 것처럼 DCL의 경우에도 특정 단어 순서 또는 형식으로 올바른 명령행 요소를 배치해야 합니다.

다음 두 보기는 일반적인 DCL 명령에 사용되는 구문 또는 형식입니다.

command/qualifier=value=keyword

command parameter/qualifier

DCL 명령을 입력할 때 명령행에 반드시 입력해야 하는 매개 변수들이 있습니다. 사용자가 필수 매개 변수를 입력하지 않으면 빠진 정보를 입력하라는 프롬프트가 표시됩니다. 밑줄(_)로 시작하는 줄은 시스템에서 사용자의 응답을 기다리고 있다는 것을 의미합니다.

선택 매개 변수인 경우에는 프롬프트에서 **Enter** 키를 눌러 생략할 수 있습니다. 필수 매개 변수를 입력한 후 하나 이상의 나머지 매개 변수와 추가 한정자를 입력할 수 있습니다.

슬래시(/) 또는 콜뱅이 기호(@)를 포함하는 매개 변수는 반드시 인용부호(“”)로 묶어야 합니다.

다음 보기에서 **TYPE** 명령은 파일 지정이 필요합니다. 파일 지정은 **TYPE** 명령의 필수 매개 변수이기 때문에 입력하지 않으면 입력하라는 프롬프트가 표시됩니다.

```
$ TYPE
_File:    WATER.TXT
```

명령 취소

명령 프롬프트 다음에 Ctrl/Z를 누르면 명령이 취소되고 DCL 프롬프트가 다시 표시됩니다.

기본값 사용

기본값 항목은 명령행에 지정할 필요가 없습니다. **기본값**을 사용하여 작업이 수행되는 경우에는 사용자가 명령을 입력할 때 값 또는 기능을 명시적으로 지정하지 않더라도 특정한 값이 명령에 할당되거나 해당 명령과 관련된 특정 기능이 수행됩니다. 일반적으로 사용자가 예상하거나 일반적인 것으로 생각되는 값과 기능이 기본값으로 사용됩니다.

명령 매개 변수와 한정자를 비롯한 몇 가지 부분에 기본값이 제공됩니다. 매개 변수 기본값에 대한 내용은 이 설명서의 각 DCL 명령을 설명한 부분을 참조하십시오. 한정자 기본값에 대한 설명은 한정자 입력을 참조하십시오.

복사본의 수를 **PRINT** 명령의 한정자로 지정하지 않으면 DCL은 기본값으로 1을 사용합니다. 다음 보기에서는 **PRINT** 명령행에 /COPIES 한정자를 포함시켜 기본값을 무시하고 여러 개의 파일 복사본이 인쇄됩니다.

```
$ PRINT/COPIES=4 MYFILE.TXT
```

여러 행 명령 입력

두 행 이상되는 명령을 입력하는 경우에는 아래와 같은 방법으로 다음 행에서 명령을 계속 입력할 수 있습니다.

표 2-2

단계	작업
1	명령행을 하이픈(-)으로 끝내고 Enter 키를 누릅니다. 밑줄(_) 다음에 DCL 프롬프트인 달러 기호(\$)가 표시됩니다.
2	이 프롬프트 다음에 나머지 명령행을 입력합니다. 밑줄로 시작하는 줄은 시스템에서 사용자의 응답을 기다리고 있다는 것을 의미합니다.

다음 사항을 주의하십시오.

- 명령 이름, 매개 변수 등 사이에 적절한 공백을 포함시켜야 합니다.
- 하이픈 다음에 **Enter** 키를 누르면 공백이 추가되지 않습니다.
- 1024개 문자 제한을 초과하지 않는 선에서 명령을 입력할 경우 사용할 수 있는 연속된 행의 수에는 제한이 없습니다.
- 하이픈을 지정하지 않고 긴 명령행을 입력할 수 있습니다. 시스템에서 자동으로 다음 행으로 바꿉니다. 그러나 하이픈을 사용하여 명령행의 부분들을 나누면 명령행을 읽기가 더 쉬워집니다.

다음 보기에서는 여러 행 명령 입력 방법을 보여 줍니다.

```
$ COPY/LOG FORMAT.TXT,FIGURE.TXT,ARTWORK.TXT -
_$ SAVE.TXT
```

DCL 명령 PIPE를 사용하여 단일 DCL 명령에서 여러 명령문을 처리하는 복잡한 명령을 만들 수 있습니다. 예를 들면, 같은 DCL 명령행에서 다음 중 한 개 이상의 작업을 실행할 수 있습니다.

- 명령 순서 파이프 처리
- 입/출력 재지정
- 여러 행 명령 및 조건 명령 실행
- 배경 처리

자세한 내용은 624페이지의 “PIPE 명령 사용”과 *OpenVMS DCL Dictionary:N-Z*의 PIPE 명령을 참조하십시오.

DCL 명령 입력 규칙

DCL 명령을 입력할 때 다음 규칙이 적용됩니다. DCL 명령에서의 확장 파일 이름 사용에 대한 정보는 5장을 참조하십시오.

- 대문자와 소문자를 조합하여 사용할 수 있습니다. DCL 인터프리터에서 소문자를 대문자로 변환합니다. 매개 변수와 한정자의 경우 인용 부호(“”)로 묶지 않는 한 대문자와 소문자가 구별되지 않습니다.
- 첫 번째 매개 변수와 명령 이름 사이에 최소한 하나의 공백 또는 탭을 넣으십시오.
- 이전 매개 변수 또는 한정자와 각 추가 매개 변수 사이에 최소한 하나의 공백 또는 탭을 넣으십시오.
- 각 한정자는 슬래시(/)로 시작하십시오. 슬래시는 구분 기호 역할을 하며 공백이나 탭을 앞에 붙일 필요가 없습니다.
- 매개 변수나 한정자 값에 공백이나 탭이 포함된 경우에는 그 값을 인용 부호로 묶으십시오.
- null 문자(<NUL>)는 인용 부호로 묶더라도 DCL 명령행에서 지정할 수 없습니다.
- 각 명령행에는 요소(매개 변수, 한정자 및 한정자 값)를 127개까지 포함시킬 수 있습니다. 한 명령의 각 요소는 255자를 초과하지 않아야 합니다. 모든 기호¹와 렉시칼 함수²가 값으로 변환된 후에는 전체 명령이 1024자를 넘으면 안 됩니다.
- 명령을 간략화된 이름으로 축약시킬 수 있습니다. 단, 한 시스템에서 정의된 다른 명령과 구별되는 고유한 이름이어야 합니다. 고유성을 확인하는 데는 처음 네 자만 사용됩니다.

다음 두 명령은 같은 명령입니다.

```
$ PRIN/COPI=2 FORMAL_ART.TXT  
$ PRINT/COPIES=2 FORMAL_ART.TXT
```

명확성을 높이고 명령 프로시저의 상위 호환성을 보장하기 위해 명령 프로시저에서는 명령을 축약하지 마십시오. 명령 프로시저에서의 명령 사용에 대한 자세한 내용은 13장과 14장을 참조하십시오.

1. 기호는 간략화된 방식으로 정보를 시스템에 전달하는 데 사용됩니다. 이에 대한 내용은 12장에서 설명합니다.
2. 5장에 설명된 렉시칼 함수는 시스템에서 정보를 얻습니다.

매개 변수 입력

파일 지정은 가장 일반적인 유형의 매개 변수입니다. DCL 명령은 입력 파일 지정 방법(명령에 의해 작업이 이루어지는 파일)과 출력 파일 지정 방법(명령에 의해 만들어지는 파일)을 받아들일 수 있습니다.

명령행에서 매개 변수를 지정할 때 다음 규칙이 적용됩니다.

- 명령 설명에서 대괄호([])는 선택 항목을 나타냅니다. 예를 들면, 다음 명령에는 파일 지정을 입력하지 않아도 됩니다.

DIRECTORY [file-spec]

- 명령 설명에서 대괄호로 묶지 않은 것은 모두 필수 항목입니다. 예를 들면, 다음 명령에는 **장치 이름**을 반드시 입력해야 합니다.

SHOW PRINTER device-name

- 일반적으로 출력 파일 매개 변수가 입력 파일 매개 변수 뒤에 나옵니다.
- 매개 변수는 한 항목일 수도 있고 일련의 항목들일 수도 있습니다. 일련의 항목을 입력하는 경우에는 쉼표(,) 또는 더하기 기호(+)를 사용하여 항목을 구분합니다. 쉼표 또는 더하기 기호의 앞이나 뒤에 공백이나 탭 문자를 임의의 수만큼 넣을 수 있습니다. 더하기 기호를 구분 기호가 아니라 연결 기호로 인식하는 명령도 있다는 점을 주의하십시오.

보기

다음 보기에서는 입력 파일 **LISTS.TXT**를 출력 파일 **FORMAT.TXT**에 복사하는 방법을 보여 줍니다.

```
$ COPY LISTS.TXT FORMAT.TXT
```

다음 보기에서는 파일 지정 목록을 매개 변수로 입력하는 방법을 보여 줍니다.

```
DELETE file-spec[,...]
```

다음 보기에서는 매개 변수 목록을 지정하는 방법을 보여 줍니다. 여기서는 세 개의 파일이 네 번째 파일에 복사됩니다. 세 개의 파일 지정 **PLUTO.TXT**, **SATURN.TXT** 및 **EARTH.TXT**가 첫 번째 매개 변수를 구성합니다. **PLANETS.TXT**는 두 번째 매개 변수입니다. 첫 번째 매개 변수를 구성하는 **PLUTO.TXT**, **SATURN.TXT** 및 **EARTH.TXT** 파일 지정 사이에는 공백이 없습니다. 입력할 때 공백을 삽입했더라도 표시되지 않습니다.

```
$ COPY PLUTO.TXT, SATURN.TXT, EARTH.TXT PLANETS.TXT
```

한정자 입력

다음 세 유형의 한정자가 있습니다.

- 명령
- 위치
- 매개 변수

한정자 이름을 간략하게 축약할 수 있습니다. 단, 동일 명령의 다른 모든 한정자 이름과 구별되는 고유한 이름이어야 합니다. 그러나 명령 프로시저의 경우 상위 호환성을 보장하려면 명령과 한정자를 축약하여 사용하지 마십시오.

명령마다 기본 한정자가 있으므로 명령 기본값과 다른 경우가 아니면 한정자를 지정할 필요가 없습니다. 다음 부분에서는 한정자의 유형과 한정자 기본값에 대해 설명합니다. 특정 명령에 대한 기본값 정보는 *OpenVMS DCL Dictionary*에 포함되어 있습니다.

명령 한정자

명령 한정자는 명령을 수정하고 명령행의 어느 위치에나 표시할 수 있습니다. 그러나 명령 이름 다음에 한정자를 넣는 것이 좋습니다. 여러 개의 한정자를 지정하는 경우에는 명령 이름 뒤에 다른 명령 한정자와 함께 넣어야 합니다.

다음 보기에서 /QUEUE는 명령 한정자입니다. SATURN.TXT 파일과 EARTH.TXT 파일이 인쇄 대기열 LN03_PRINT에 대기합니다.

```
$ PRINT/QUEUE=LN03_PRINT SATURN.TXT,EARTH.TXT
```

위치 한정자

위치 한정자는 명령이나 매개 변수를 수정할 수 있으며 명령 문자열의 어느 위치에 두느냐에 따라 의미가 달라집니다. 위치 한정자를 명령 뒤, 첫 번째 매개 변수 앞에 넣으면 전체 명령 문자열에 영향을 주지만 매개 변수 뒤에 넣으면 해당 매개 변수에 대해서만 영향을 줍니다.

다음 보기에서 첫 번째 PRINT 명령은 SPRING.SUM과 FALL.SUM 두 파일의 복사본을 두 개 만들도록 요청하고, 두 번째 PRINT 명령은 SPRING.SUM 파일은 두 개의 복사본을, FALL.SUM은 한 개의 복사본을 만들도록 요청합니다.

```
$ PRINT/COPIES=2 SPRING.SUM,FALL.SUM  
$ PRINT SPRING.SUM/COPIES=2,FALL.SUM
```


매개 변수 한정자

매개 변수 한정자는 입력 파일, 출력 파일 등 특정 유형의 매개 변수에만 사용할 수 있습니다.

예를 들면, **BACKUP** 명령은 입력 및 출력 파일 지정에만 적용되는 몇 가지 매개 변수 한정자를 받아들입니다.

다음 보기에서는 입력 파일과 함께 사용하는 경우에만 지정할 수 있는 **/CREATED** 및 **/BEFORE** 한정자는 백업 작업을 위해 특정 입력 파일을 선택합니다. 별표(*)는 파일 이름 대신 사용한 와일드카드 문자입니다. **BACKUP** 명령은 2002년 12월 11일 이전에 만들어진 **.TXT** 유형의 파일을 모두 선택합니다.

```
$ BACKUP *.TXT/CREATED/BEFORE=11-DEC-2002 NEWFILE.TXT
```

충돌하는 한정자

하나의 명령행에서 두 개 이상의 모순되는 한정자를 사용할 경우에는 가장 오른쪽의 한정자가 다른 한정자들보다 우선 적용됩니다.

일부 명령은 동일한 명령행에서 지정할 수 없는 모순되는 한정자를 포함합니다. 호환되지 않는 한정자를 사용하면 명령 인터프리터에서 오류 메시지를 표시합니다.

다음 보기에서는 한정자가 충돌하는 경우를 보여 줍니다. **PRINT** 명령은 명령행에서 가장 오른쪽에 있는 한정자인 **/COPIES=2**와 **/NOBURST** 한정자만 받아들입니다.

```
$ PRINT MYFILE/COPIES=3/BURST/COPIES=2/NOBURST EARTH.TXT
```

한정자에서 사용할 수 있는 값

한정자는 키워드, 파일 지정, 문자열 및 숫자 값을 받아들일 수 있습니다. 한정자의 값을 입력할 때는 등호(=) 또는 콜론(:)을 사용하여 한정자와 값을 구분합니다.

일부 한정자 키워드의 경우에는 추가 정보가 필요합니다. 이러한 경우에는 콜론이나 등호를 사용하여 키워드와 값을 구분합니다.

값을 요구하는 키워드를 여러 개 지정하려면 목록을 괄호로 묶고 등호나 콜론을 사용하여 키워드와 값을 구분합니다.

보기

다음 보기에서는 어느 명령이나 유효합니다.

```
$ PRINT/COPIES=3 MYFILE.DAT
```

```
$ PRINT/COPIES:3 MYFILE.DAT
```

한정자 입력

다음 보기에서는 추가 정보가 필요한 한정자를 보여 줍니다. 콜론 또는 등호(=)를 사용하여 키워드 “PROTECTION”와 값을 구분합니다.

```
$ SET SECURITY/PROTECTION:GROUP:RW MYFILE.DAT
```

```
$ SET SECURITY/PROTECTION=GROUP=RW MYFILE.DAT
```

다음 보기에서는 여러 개의 키워드가 필요한 한정자를 보여 줍니다. 각 한정자에 여러 개의 값이 필요합니다.

```
$ SET SECURITY/PROTECTION=(OWNER=RWD,GROUP=RW) myfile.dat
```

```
$ SET SECURITY/PROTECTION=(OWNER:RWD,GROUP:RW) myfile.dat
```

날짜 및 시간을 값으로 입력

PRINT/AFTER 명령과 같은 특정 명령 및 한정자는 날짜와 시간 값을 받아들입니다. 다음 중 하나의 형식으로 이러한 값들을 지정할 수 있습니다.

- 절대 시간
- 변경 시간
- 조합 시간(절대 시간 표시 형식과 변경 시간 표시 형식을 함께 사용)

절대 시간 표시 형식

절대 시간은 특정한 날짜 또는 하루 중 특정한 시간입니다. 절대 시간 표시 형식은 다음과 같습니다.

[dd-mmm-yyyy][:hh:mm:ss.cc]

각 필드는 다음과 같습니다.

표 2-3

dd	일: 1부터 31까지의 정수
mmm	월: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
yyyy	연도: 정수
hh	시: 0부터 23까지의 정수
mm	분: 0부터 59까지의 정수
ss	초: 0부터 59까지의 정수
cc	1/100초: 0부터 99까지의 정수

절대 시간을 지정할 때는 다음 규칙을 적용합니다.

- 날짜나 시간을 오른쪽에서 잘라버릴 수 있습니다.
- 날짜와 시간을 모두 지정하는 경우에는 사이에 콜론을 넣으십시오.
- 날짜에는 최소한 하나의 하이픈이 포함되어야 합니다.

날짜 및 시간을 값으로 입력

- 문장 부호를 사용하여 필드를 구분하기만 하면 날짜와 시간 내의 어느 필드나 생략할 수 있습니다.
- 자르거나 생략한 날짜 필드에는 현재 날짜가 해당 필드의 기본값으로 사용됩니다.
- 자르거나 생략한 시간 필드에는 0이 기본값으로 사용됩니다.
- 현재 또는 미래 시간이 필요한 명령에 이전 시간을 지정하면 현재 시간이 사용됩니다. 절대 시간을 다음 키워드 중 하나로 지정할 수도 있습니다.

표 2-4

TODAY	현재 일, 월, 연도의 00:00:00.0 시
TOMORROW	내일 00:00:00.00 시
YESTERDAY	어제 00:00:00.00 시

다음 표는 절대 시간 지정 방법의 예를 보여 줍니다.

표 2-5

시간 지정 방법	결과
11-DEC-2002:13	2002년 11월 오후 1시
11-DEC	금년 12월 11일이 시작되는 자정
15:30	오늘 오후 3:30
19-	현재 연, 월의 19일 자정
19-:30	이달 19일 오전 12:30

변경 시간 표시 형식

변경 시간은 현재 날짜와 시간부터 미래 시간까지의 시간 간격입니다. 변경 시간의 일반 형식은 다음과 같습니다.

"+[dddd-][hh:mm:ss.cc]"

각 필드는 다음과 같습니다.

표 2-6

dddd	날짜 수, 0부터 9999까지의 정수
hh	시간, 0부터 23까지의 정수
mm	분, 0부터 59까지의 정수
ss	초, 0부터 59까지의 정수
cc	1/100초, 0부터 99까지의 정수

절대 시간, 변경 시간 또는 조합 시간으로 표현할 수 있는 값의 한정자인 경우에는 변경 시간을 조합 시간의 일부인 것처럼 지정해야 합니다. 예를 들어, 변경 시간 값이 현재 시간으로부터 5분이면 “+5”가 아니라 “0-0:5”를 사용하십시오.

변경 시간을 지정할 때는 다음 규칙을 적용합니다.

- 변경 시간을 오른쪽에서 잘라낼 수 있습니다.
- 날짜 수를 지정하는 경우에는 하이픈을 넣습니다.
- 필드를 구분하는 문장 부호를 넣기만 하면 시간 내의 필드를 생략할 수 있습니다.
- 시간 필드를 생략하면 기본값은 0입니다.

다음 표는 변경 시간 지정 방법의 예를 보여 줍니다.

표 2-7

시간 지정 방법	결과
“+3-”	지금부터 3일(72시간)
“+3”	지금부터 3시간
“+:30”	지금부터 30분
“+3-:30”	지금부터 3일 30분
“+15:30”	지금부터 15시간 30분

조합 시간 형식

절대 시간과 변경 시간을 함께 사용하려면 변경 시간을 더하거나 빼 절대 시간을 지정합니다. 다음 중 한 형식을 사용하십시오.

```
"[absolute time][+delta time]"  
[absolute time][-delta time]
```

절대 시간과 변경 시간 값의 변수 필드와 기본 필드는 앞에서 설명한 것과 동일합니다.

조합 시간을 지정할 때는 다음 규칙을 적용합니다.

- 변경 시간 값 앞에 더하기 기호나 빼기 기호를 붙입니다. 빼기 기호는 키보드의 하이픈 키입니다.
- 변경 시간 값 앞에 더하기 또는 빼기 기호가 붙어 있는 경우에는 전체 시간 지정을 인용 부호로 묶습니다.
- 현재 날짜 및 시간으로부터의 변경 시간을 지정하는 경우에는 절대 기간 값을 생략합니다.
- 가능하면 빠짐없이 날짜와 시간 정보를 지정하십시오.

다음 표는 조합 시간 지정의 예를 보여 줍니다.

표 2-8

시간 지정 방법	결과
“+5”	지금부터 5시간
“-1”	현재 시간 빼기 1시간. 빼기 기호(-)는 음의 오프셋을 나타냅니다. 뒤에 하이픈이 붙어 있지 않기 때문에 1은 하루가 아니라 1시간으로 해석됩니다.
“+:5”	지금부터 5분
“-:5”	현재 시간 빼기 5분
“-1-00”	현재 시간 빼기 1일. 빼기 기호(-)는 음의 오프셋을 나타냅니다. 1 뒤에 붙은 하이픈(-)은 날짜와 시간 필드를 구분합니다.
“31-DEC+:5”	금년 12월 31일 오전 12:05. 콜론 앞 부분의 절대 시간 지정의 기본값은 금년 12월 31일 자정입니다. 더하기 기호(+)는 양의 오프셋을 나타냅니다.
31-DEC:-00:10	금년 12월 30일 오후 11:50. 콜론 앞 부분의 절대 시간 지정의 기본값은 금년 12월 31일 자정입니다. DEC: 뒤의 빼기 기호(-)는 음의 오프셋을 나타냅니다.

명령 다시 호출

DCL 프롬프트에서 긴 명령행을 다시 입력하지 않고 이전에 입력한 명령행을 다시 호출할 수 있습니다. 명령이 표시되면 다시 실행하거나 편집할 수 있습니다.

OpenVMS VAX 시스템에서는 호출 버퍼에 이전에 입력한 명령이 20개까지 저장됩니다.

OpenVMS Alpha 시스템에서는 호출 버퍼에 이전에 입력한 명령이 254개까지 저장됩니다.

다음 중 한 가지 방법을 사용하여 이전에 입력한 명령을 표시할 수 있습니다.

- Ctrl/B 누르기
- 위쪽 및 아래쪽 화살표 키 사용
- RECALL 명령 사용

Ctrl/B 누르기

Ctrl/B를 한 번 누르면 이전 명령행이 다시 호출됩니다. Ctrl/B를 반복해서 누르면 이전 행 전에 입력했던 명령행에서 저장된 마지막 명령행까지 다시 호출할 수 있습니다.

화살표 키 사용

위쪽 및 아래쪽 화살표 키를 사용하여 바로 전의 명령과 저장된 각 명령을 연속적으로 다시 호출할 수 있습니다. 화살표 키를 반복해서 눌러 다른 명령으로 이동합니다.

RECALL 명령 사용

이전에 입력한 명령행을 조사하려면 RECALL/ALL 명령을 입력합니다. 사용할 수 있는 명령들을 검토한 후 RECALL과 원하는 명령의 번호를 입력하면 특정 명령행을 다시 호출할 수 있습니다.

표시하려는 명령행의 처음 문자들을 RECALL 뒤에 입력하는 방법도 있습니다. 이전 명령행을 가장 최근에 입력한 것부터 검색하여 사용자가 입력한 문자로 시작하는 첫 번째 명령행이 표시됩니다.

보기

다음 보기에서는 **RECALL/ALL**을 입력하면 이전 명령들이 표시되는 것을 보여 줍니다.

```
$ RECALL/ALL
1 SET DEFAULT DISK2:[MARSHALL]
2 EDIT ACCOUNTS.COM
3 PURGE ACCOUNTS.COM
4 DIRECTORY/FULL ACCOUNTS.COM
5 COPY ACCOUNTS.COM [ .ACCOUNTS]*
6 SET DEFAULT [ .ACCOUNTS]
```

다음 보기에서는 네 번째 명령행을 다시 호출하는 방법을 보여 줍니다.

```
$ RECALL 4
```

Enter 키를 누르면 **DCL** 프롬프트에서 위 목록의 네 번째 명령이 표시됩니다. **RECALL** 명령 자체는 호출 버퍼에 저장되지 않습니다.

다음 보기에서는 바로 전에 입력한 명령행 **EDIT ACCOUNTS.COM**을 다시 호출하는 방법을 보여 줍니다.

```
$ RECALL E
```

Enter 키를 누르면 다음 명령행이 표시됩니다.

```
$ EDIT ACCOUNTS.COM
```

주

OpenVMS 화면 관리 소프트웨어를 사용하는 유틸리티 또는 어플리케이션을 실행하고 있는 경우에는, **Ctrl/B**와 위쪽/아래쪽 화살표 키를 사용하여 명령 다시 호출을 수행할 수 있지만 그렇게 하려면 행 편집이 가능해야 합니다. 이 기능이 있는 유틸리티로는 **Mail**, **OpenVMS Debugger**, **Show Cluster**, **System Dump Analyzer(SDA)**, **EVE** 편집기 등이 있습니다.

호출 버퍼의 내용을 지우려면 **RECALL** 명령과 함께 **ERASE** 한정자를 입력합니다.

예를 들면 다음과 같습니다.

```
$ RECALL/ERASE
```

보안상의 이유로 암호가 포함된 명령을 입력한 후에는 호출 버퍼의 내용을 지우는 것이 좋습니다.

DCL 명령행 편집

DCL 명령 수준에서는 여러 가지 키와 키 시퀀스를 사용하여 입력한 내용을 변경할 수 있습니다. 터미널 유형에 따라 차이는 있지만 대부분의 터미널에는 **행 편집기**와 함께 사용할 수 있는 표준 기능 키가 있습니다.

행 편집이 가능한지 보려면 **SHOW TERMINAL** 명령을 입력합니다.

다음 보기에서는 행 편집이 가능합니다.

```
$ SHOW TERMINAL
```

```
Terminal: _VTA2138: Device_Type: VT200_Series Owner: ROHBA
```

```
Physical terminal: _TNA2114:
```

```
Remote Port Info: Host: 16.32.216.68 Port: 1409
```

```
Input: 9600 LFill: 0 Width: 80 Parity: None
Output: 9600 CRfill: 0 Page: 24
```

```
Terminal Characteristics:
```

Interactive	Echo	Type_ahead	No Escape
Hostsync	TTsync	Lowercase	Tab
Wrap	Scope	No Remote	Eightbit
Broadcast	No Readsyc	No Form	Fullldup
No Modem	No Local_echo	No Autobaud	Hangup
No Brdcstmbx	No DMA	No Altypeahd	Set_speed
No Commsync	Line Editing	Overstrike editing	No fallback
No Dialup	No Secure server	Disconnect	No Psthru
No Syspassword	No SIXEL Graphics	No Soft Characters	Printer port
Numeric Keypad	ANSI_CRT	No Regis	No Block_mode
Advanced_video	Edit_mode	DEC_CRT	DEC_CRT2
No DEC_CRT3	No DEC_CRT4	No DEC_CRT5	No Ansi_Color
VMS Style Input			

SET TERMINAL 명령

SET TERMINAL 명령을 사용하여 터미널의 DCL 명령행 편집 방법을 변경할 수 있습니다.

기본적으로 **SET TERMINAL** 명령을 사용하여 변경한 사항은 현재 세션에만 적용됩니다.

로그인할 때마다 터미널을 설정하려면 **LOGIN.COM** 파일에 **SET TERMINAL** 명령을 포함시키면 됩니다.

행 편집을 활성화하려면 **SET TERMINAL/LINE_EDIT** 명령을 입력합니다.

```
$ SET TERMINAL/LINE_EDIT
```

삽입 및 겹쳐쓰기 모드

삽입 모드 또는 겹쳐쓰기 모드로 명령행을 편집할 수 있습니다. 삽입 모드에서는 입력하는 문자가 **커서**의 왼쪽에 삽입됩니다. 겹쳐쓰기 모드에서는 입력하는 문자가 커서가 가리키는 문자 위에 겹쳐져서 쓰여집니다.

단일 명령행의 편집 모드를 변경하려면 **Ctrl/A**를 누릅니다. **Ctrl/A**는 토글 역할을 합니다. 세션의 편집 모드를 변경하려면 **SET TERMINAL/INSERT** 또는 **SET TERMINAL/OVERSTRIKE** 명령을 입력합니다.

텍스트 배치

SET TERMINAL/WRAP 명령을 사용하면 화면의 한 행에 들어가는 글자 수보다 더 많은 글자를 입력할 경우 텍스트가 다음 행으로 이어져서 표시됩니다.

SET TERMINAL/NOWRAP 명령을 사용하면 화면의 한 행에 들어가는 글자 수보다 더 많은 글자를 입력할 경우 그 행의 마지막 글자 위에 겹쳐져서 입력됩니다.

커서가 표시된 명령행만 편집할 수 있습니다. 텍스트 줄 바꿈을 사용하는 경우에는 위쪽 화살표를 사용하여 이전 명령행로 커서를 옮겨 편집할 수 없습니다. 이전 행으로 커서를 이동하려면 **Delete** 키를 사용하여 현재 행의 모든 문자를 삭제하십시오.

명령행 일부 삭제

백스페이스 키를 누르면 커서가 역방향으로 이동하고 그 자리의 문자가 지워집니다. 행 편집이 가능한 경우에는 **Ctrl/U**를 사용하여 그 행 시작부터 현재 커서 위치까지의 문자를 삭제할 수 있습니다. 행 편집이 불가능한 경우에는 **Ctrl/U**를 사용하면 한 행 전체가 취소됩니다. 그 행을 무시하고 DCL 프롬프트가 다시 표시됩니다.

터미널 키 정의

키 정의는 사용자가 특정 터미널 키에 대해 지정하는 문자열입니다. **DEFINE/KEY** 명령을 사용하십시오. 키를 정의한 후에는 문자열을 입력하는 대신 해당 키를 누르면 됩니다. 키 정의에는 일반적으로 명령행 전체 또는 일부가 포함됩니다. 키 정의를 사용하여 키보드를 사용자 정의하면 보다 적은 수의 키를 사용하여 **DCL** 명령을 입력할 수 있습니다. 정의된 키를 누르면 명령을 정의할 때 **/TERMINATE** 한정자를 사용하였는지 여부에 따라 터미널에 명령이 표시되거나 명령이 실행됩니다.

기본적으로 터미널은 숫자 키패드 모드로 설정됩니다. 숫자 키패드의 키를 다시 정의하려면 **SET TERMINAL** 명령을 사용하십시오. 자세한 내용은 *OpenVMS DCL Dictionary*의 **SET TERMINAL/APPLICATION_KEYPAD**, **SET TERMINAL/NUMERIC** 및 **DEFINE/KEY** 명령을 참조하십시오.

키 시퀀스

DCL 명령을 입력하는 방법 외에 특정 키 시퀀스를 사용하여 작업을 수행할 수 있습니다. 키 시퀀스는 시스템에서 다른 명령을 처리하는 동안 주의를 끌기 위한 방법입니다.

키 시퀀스를 입력하려면 **Ctrl** 키를 누른 채로 두 번째 키를 눌렀다 놓으십시오. 다음 표는 키 시퀀스를 기능별로 정리한 것입니다.

표 2-9

DCL 명령 입력 기능

키 시퀀스	기능
Ctrl/Z 및 F10	터미널에서 입력된 데이터에 파일 끝을 알리는 기능
Enter	현재 행 처리를 위해 시스템에 보내는 기능. 이미 로그인되어 있지 않은 경우에는 Enter 키를 누르면 로그인 시퀀스가 시작됩니다.

표 2-10

DCL 명령 중단 기능

키 시퀀스	기능
Ctrl/T	터미널 출력을 잠시 중단하고 현재 프로세스에 대한 통계 정보 행을 표시하는 기능. 사용자 노드, 사용자 이름, 시간, 실행중인 이미지의 이름, 현재 터미널 세션 중에 사용되는 시스템 자원에 대한 정보가 표시됩니다. Ctrl/T를 사용하여 시스템이 작동 중인지 확인할 수도 있습니다. 시스템이 잠시 응답을 하지 않거나 터미널이 NOBROADCAST로 설정된 경우에는 Ctrl/T를 눌러도 정보가 표시되지 않습니다. Ctrl/T를 사용하려면 시스템 로그인 명령 프로시저 또는 개인 로그인 명령 프로시저에서나 대화형으로 SET CONTROL=T 명령을 먼저 입력해야 합니다.

표 2-10

DCL 명령 중단 기능(계속)

키 시퀀스	기능
Ctrl/Y, Ctrl/C 및 F6	명령 처리를 중단하는 기능. SET NOCONTROL=Y 명령을 사용하여 Ctrl/Y를 사용 불가능 상태로 만들 수 있습니다. 대부분의 상황에서 Ctrl/Y는 DCL 프롬프트를 반환합니다. 실행 중인 프로그램은 계속 활성화된 상태입니다. 내장된 명령을 입력한 다음 CONTINUE 명령을 사용하여 프로그램을 계속할 수 있습니다. CONTINUE 명령을 입력한 후에는 Ctrl/W를 눌러 화면을 새로 고치도록 하십시오.

표 2-11

명령 다시 호출 기능

키 시퀀스	기능
Ctrl/B 및 위쪽 화살표	이전에 입력한 명령을 20개(VAX) 또는 254개(Alpha)까지 다시 호출하는 기능
아래쪽 화살표	호출 버퍼의 다음 행을 표시하는 기능

표 2-12

커서 위치 제어 기능

키 시퀀스	기능
백스페이스	입력된 마지막 문자를 삭제하는 기능
Ctrl/A 및 F14	겹쳐쓰기 모드와 삽입 모드 사이를 전환하는 기능. SET TERMINAL/LINE_EDITING 명령을 사용하여 설정한 기본 모드가 각 줄이 시작될 때 다시 설정됩니다.
Ctrl/D 및 왼쪽 화살표	커서를 한 자 왼쪽으로 이동하는 기능
Ctrl/E	커서를 행 끝으로 이동하는 기능
Ctrl/F 및 오른쪽 화살표	커서를 한 자 오른쪽으로 이동하는 기능
Ctrl/H 및 F12	커서를 행 시작으로 이동하는 기능

표 2-12

커서 위치 제어 기능(계속)

키 시퀀스	기능
Ctrl/I 및 탭	커서를 터미널 상의 다음 탭 정지 지점으로 이동하는 기능. 한 행의 8번째 문자마다 탭 정지 지점이 제공됩니다. 탭 설정은 일반적으로 사용자가 수정할 수 있는 하드웨어 터미널의 특징입니다. 탭 키는 행 편집이 사용 불가능 상태일 때도 작동합니다.
Ctrl/J	커서 왼쪽의 단어를 삭제하는 기능
Ctrl/K	현재 행을 다음 수직 탭 정지 지점으로 옮기는 기능
Ctrl/L	커서를 다음 페이지 시작으로 이동하는 기능. 행 편집이 가능할 때는 이러한 Ctrl/L 사용이 무시됩니다.
Ctrl/R	현재 명령을 반복하고 Ctrl/R을 누를 때 커서가 있던 위치에 커서를 그대로 두는 기능
Ctrl/U	커서 왼쪽에 있는 현재 입력 행의 모든 텍스트를 삭제하는 기능
Ctrl/V	일부 행 편집 기능 키를 사용할 수 없도록 하는 기능. 예를 들어, Ctrl/V를 누른 다음 Ctrl/D를 누르면 커서가 왼쪽으로 한 자 이동하는 대신 Ctrl/D가 생성됩니다. Ctrl/D는 DCL 수준의 행 종료자입니다. Ctrl/V와 함께 사용하는 경우 행 종료자가 아닌 문자는 아무런 영향도 주지 않습니다. Ctrl/H와 Ctrl/J를 예로 들 수 있습니다. 그러나 Ctrl/U와 같은 특정 제어 키는 행 편집 기능을 그대로 보유합니다.
Ctrl/X	현재 행을 취소하고 미리 입력 버퍼의 데이터를 삭제하는 기능
F7, F8, F9, F11	Compaq에 의해 예약된 기능

표 2-13

화면 표시 제어 기능

키 시퀀스	기능
Ctrl/O	터미널에 출력을 표시하는 것을 일시 중단했다 다시 계속하는 것을 번갈아 하는 기능. Ctrl/O는 Output off과 Output on으로 표시됩니다.
Ctrl/S	Ctrl/Q를 누를 때까지 터미널 출력을 일시적으로 중단하는 기능
Ctrl/Q	Ctrl/S에 의해 일시 중단되었던 터미널 출력을 재개하는 기능
Ctrl/W	화면 표시를 다시 실행하는 기능

3

파일을 사용하여 정보 저장

파일은 정보가 들어 있는 시스템 객체입니다. 이 정보는 컴퓨터에서 이해하는, 즉 기계가 읽을 수 있는 **데이터**가 될 수 있으며 사용자가 입력하고 조작하는 텍스트가 될 수도 있습니다. 파일 내용은 문서나 프로그램, 주소 목록의 텍스트가 될 수 있습니다. 텍스트 파일을 온라인으로 표시하거나 인쇄하여 해당 내용을 검사할 수 있습니다.

이미지 또는 **실행 가능 이미지**라고도 하는 프로그램은 읽을 수 있는 형식의 지시 사항 및 데이터가 들어 있는 파일입니다. 일부 프로그램은 **DCL** 명령과 연결되어 있습니다. 예를 들어, **DCL** 명령 **COPY**를 입력하면 시스템에서 **SYS\$SYSTEM:COPY.EXE** 프로그램을 실행합니다. 일부 프로그램은 **DCL** 명령 **RUN** 다음에 프로그램 이름을 입력하면 시작됩니다.

이미지 파일은 운영 체제나 사용자가 제공할 수 있으며 일반적으로 **.EXE**라는 파일 유형을 갖습니다. 이미지 파일은 **ASCII** 문자로 구성되어 있지 않으므로 **DCL** 명령 **TYPE**, **PRINT** 또는 **EDIT**를 사용하여 이미지 파일을 검사할 수 없습니다. 텍스트 파일에는 알파벳, 구두점, 숫자 및 기타 특수 기호를 나타내는 표준 방법인 **ASCII** 문자가 들어 있습니다.

이 장에서는 **OpenVMS** 네트워크용 **TCP/IP** 또는 **DECnet**을 통해 로컬로 파일을 만들고 조작하는 방법을 설명합니다. 이 장에는 다음 정보가 포함되어 있습니다.

- 파일 이름 및 파일 지정 방법 이해
- 파일 이름을 사용하여 와일드카드 사용
- 기타 파일 이름
- 파일 만들기 및 수정
- 파일 내용 표시
- 파일 삭제
- 다른 사용자로부터 파일 보호
- 파일 인쇄

자세한 내용은 다음을 참조하십시오.

- 5장, 확장 파일 지정 방법을 사용하는 환경에서 파일 이름에 대한 정보
- *OpenVMS DCL Dictionary* 및 온라인 도움말, 이 장에 설명되어 있는 명령
- *OpenVMS System Manager's Manual*, 원격 노드 액세스에 대한 정보
- *Compaq TCP/IP Services for OpenVMS User's Guide*, **TCP/IP** 사용자 유틸리티 및 명령 사용에 대한 정보
- *DECnet-Plus for OpenVMS Networking Manual*, **DECnet** 네트워크에 대한 정보
- *DECnet-Plus for OpenVMS Introduction and User's Guide*, **DECnet Phase V** 네트워크에 대한 정보

파일 이름 및 파일 지정 방법 이해

파일은 사람이 읽을 수 있는 데이터와 기계가 읽을 수 있는 데이터를 저장하기 위해 OpenVMS 운영 체제에서 사용하는 단위입니다. 파일을 만들거나 이름을 지정하는 경우 시스템에서 파일을 찾고 식별하는 데 사용할 수 있는 정보를 제공해야 합니다.

파일 이름은 **파일 이름**과 **파일 유형**으로 구성됩니다. 이름과 유형은 마침표(.)로 구분됩니다. 또한, 파일에는 **버전 번호**가 있습니다. 한 파일에 대해 여러 버전이 있을 수 있습니다. 버전 번호를 지정하지 않으면 시스템에서 파일의 가장 높은 기존 버전 번호를 사용합니다. 파일을 편집할 때 시스템은 원래 버전은 수정하지 않고 새 **출력 파일**을 만듭니다. 기본적으로 출력 파일은 원본과 동일한 이름과 파일 유형을 갖지만 동일한 이름의 기존 파일보다 하나 더 높은 버전 번호를 갖습니다.

파일 지정 방법은 파일 이름, 파일 유형 및 버전 번호로 구성됩니다.

완전한 파일 지정 방법 제공

파일은 네트워크의 특정 컴퓨터나 노드, 해당 볼륨의 특정 디렉토리에서 해당 컴퓨터에 연결된 특정 장치 또는 장치 집합(볼륨)에 있습니다. 완전한 파일 지정 방법의 역할은 다음과 같습니다.

- 파일을 찾고 식별하기 위해 시스템에서 사용하는 액세스 경로를 정확하게 설명합니다.
- 파일이 있는 디렉토리와 파일이 상주하는 네트워크 노드를 포함할 수 있습니다.
- 네트워크 파일 지정 방법으로도 알려져 있습니다.

완전한 파일 지정 방법의 모든 요소를 포함할 필요는 없습니다. 하지만 기본 구성 요소와 결합하는 경우 시스템에서 올바른 파일을 찾고 식별할 수 있도록 파일 지정 방법을 자세하게 지정해야 합니다.¹

시스템 기본값을 무시하거나 네트워크를 통해 파일 작업을 수행하려면 완전한 파일 지정 방법을 제공해야 합니다. 완전한 파일 지정 방법의 형식은 다음과 같습니다.

node::device:[root.][directory]file-name.file-type;version

1. 레코드 관리 서비스(RMS)는 파일 처리 및 관리에서 어플리케이션을 보조하는 OpenVMS 기능입니다. RMS는 파일 지정 방법 구문 분석 규칙을 유지 관리합니다. RMS에서 부분적인 파일 지정 방법에 기본값을 적용하는 방법에 대한 자세한 내용은 *Guide to OpenVMS File Applications*를 참조하십시오.

구성 요소는 다음과 같습니다.

표 3-1

Node	네트워크 노드나 호스트 이름입니다. TCP/IP나 DECnet을 지원하는 시스템에만 적용됩니다. 자기 테이프에 저장된 파일에는 적용되지 않습니다. 사용자가 로그인한 동일한 시스템에 있는 파일을 지정하는 데는 사용할 수 없습니다.
Device	OpenVMS 운영 체제를 실행 중인 컴퓨터에 연결된 디스크나 테이프 드라이브 또는 기타 주변 장치를 언급하는 데 사용되는 용어입니다. 각 장치에는 해당 유형과 위치를 나타내는 고유한 이름이 있습니다. 디스크는 ODS-2(기본값) 또는 ODS-5(OpenVMS Alpha 전용)로 포맷할 수 있습니다.
Directory	파일이 저장된 디렉토리의 이름입니다. 대괄호([]) 또는 꺾쇠 괄호(< >)를 사용하여 디렉토리를 구분합니다. 자기 테이프에 저장된 파일에는 적용되지 않습니다.
File name	파일의 이름입니다.
File type	파일의 구조나 유형을 식별합니다.
Version	파일의 버전 번호입니다. 버전은 10진수로 식별되며 새로운 파일 버전이 만들어질 때마다 1씩 증가합니다. 버전 번호를 사용자가 지정하지 않으면 시스템에서 자동으로 할당합니다.

파일 지정 방법의 규칙

다음 규칙을 사용하여 파일 지정 방법의 요소를 지정하십시오.

- 의미있는 파일 이름을 지정합니다. ODS-2 디스크를 사용하는 OpenVMS Alpha 및 OpenVMS VAX 시스템의 경우 파일 이름은 문자 A-Z(대문자 또는 소문자), 숫자 0-9, 밑줄(_), 하이픈(-), 물결표(~) 및 달러 기호(\$)에서 선택하여 39자까지 지정할 수 있습니다.
- 일부 이전 버전의 OpenVMS에서는 모든 형식의 파일 지정 방법에서 하이픈을 허용하지 않으므로 파일 이름의 첫 문자로 하이픈을 사용하지 마십시오.

- 파일 유형은 마침표(.)로 시작합니다. ODS-2 디스크를 사용하는 Alpha 및 VAX 시스템에서 파일 유형은 문자 A-Z(대문자 또는 소문자 형식으로 지정할 수 있음), 숫자 0-9, 밑줄(_), 하이픈(-) 및 달러 기호(\$)에서 선택하여 39자까지 지정할 수 있습니다.
- 버전 구성 요소는 세미콜론(;) 또는 마침표(.)로 시작합니다. 시스템에서 파일 지정 방법을 표시할 때 버전 구성 요소에 대해 세미콜론을 표시합니다.
- 자기 테이프의 파일을 참조하는 데에는 디렉토리 필드를 사용하지 마십시오. 디렉토리는 디스크의 파일에만 적용됩니다.
- 시스템이 네트워크의 일부인 경우 그리고 파일이 사용자가 로그인한 노드가 아닌 다른 노드에 있는 경우에만 노드 이름을 포함합니다.
- ODS-2 디스크를 사용하는 OpenVMS Alpha 및 OpenVMS VAX 시스템의 경우 사용자 파일 디렉토리(UFD) 이름 또는 하위 디렉토리 이름은 문자 A-Z(대문자 또는 소문자 형식으로 지정할 수 있음), 숫자 0-9, 밑줄(_), 하이픈(-) 및 달러 기호(\$)에서 선택하여 39자까지 지정할 수 있습니다. 하이픈으로 시작하는 하위 디렉토리 이름은 허용되지 않습니다.
- OpenVMS Alpha 버전 7.2 이상에서 디렉토리 및 루트 구성 요소의 모든 하위 디렉토리에서 총 문자 수(대괄호 및 구분 마침표 제외)는 512자를 초과할 수 없습니다. 또한, 디렉토리가 <directory-name>.DIR;1 형식의 파일로 저장된다는 점을 고려하여 UFD 및 하위 디렉토리 이름에는 파일 이름, 유형 및 버전 구성 요소와 동일한 제약 조건이 있습니다.
- 확장 파일 지정 방법을 지원하는 시스템과 지원하지 않는 시스템으로 구성된 환경에서는 이름이 좀 더 제한된 시스템의 기능인 파일 및 디렉토리는 해당 시스템에서 액세스할 수 없다는 것을 기억하십시오.

자세한 내용은 *Guide to OpenVMS File Applications*를 참조하십시오.

주

확장 파일 지정 방법을 사용하는 환경에서는 파일에 대한 이러한 규칙이 다릅니다. 확장 파일 이름에 대한 자세한 내용은 5장을 참조하십시오.

DCL 명령에서 사용하는 기본 파일 유형

특정 명령의 경우, 파일 유형을 생략하면 시스템에서 기본값을 적용합니다. 다음 표에서는 DCL 명령에서 사용하는 좀 더 일반적인 기본 파일 유형 중 몇 가지를 설명합니다.

표 3-2

파일 유형	내용
.CLD	명령 설명 파일
.COM	명령 프로시저 파일
.DAT	데이터 파일
.DIF	DIFFERENCES 명령으로 만든 출력 파일
.DIR	디렉토리 파일
.DIS	Mail 유틸리티의 메일 그룹 파일
.EXE	링커로 만든 실행 가능 프로그램 이미지 파일
.HLB	도움말 텍스트 라이브러리 파일
.HLP	도움말 라이브러리의 입력 소스 파일
.INI	초기화 파일
.LIS	언어 컴파일러 또는 어셈블러로 만든 목록 파일, PRINT 및 TYPE 명령의 기본 입력 파일
.LOG	일괄 처리 작업 출력 파일
.MAI	메일 메시지 파일
.PS	포스트스크립트 형식 파일
.SYS	시스템 이미지 파일
.TJL	DECTPU 및 ACL 편집기로 만든 저널 파일
.TLB	텍스트 라이브러리 파일
.TMP	임시 파일
.TPU	EVE 편집기용 명령 파일
.TPU\$JOURNAL	EVE 편집기로 만든 저널 파일
.TXT	텍스트 라이브러리용 입력 파일 또는 Mail 유틸리티 출력 파일

언어 소스 프로그램용 기본 파일 유형

다음 표에서는 일부 고급 언어 소스 프로그램용 기본 파일 유형을 나열합니다.

표 3-3

파일 유형	내용
.ADA	Compaq Ada 컴파일러용 입력 소스 파일
.BAS	BASIC 컴파일러용 입력 소스 파일
.B32	VAX BLISS-32 컴파일러용 입력 소스 파일
.C	Compaq C 컴파일러용 입력 소스 파일
.COB	OpenVMS VAX 시스템의 VAX COBOL 컴파일러 및 OpenVMS Alpha 시스템의 Compaq COBOL 컴파일러용 입력 소스 파일
.FOR	Compaq Fortran(OpenVMS VAX 시스템용 Compaq Fortran은 이전에는 VAX Fortran이었음)용 입 력 소스 파일
.M64	OpenVMS Alpha의 MACRO-64 어셈블러용 입력 소스 파일
.MAP	Linker 유틸리티로 만든 메모리 할당 맵
.MAR	OpenVMS Alpha의 VAX MACRO 어셈블러 또는 MACRO-32 컴파일러용 입력 소스 파일
.MLB	MACRO 어셈블러용 매크로 라이브러리
.MSG	메시지 텍스트를 지정하는 소스 파일
.OBJ	언어 컴파일러 또는 어셈블러로 만든 객체 파일
.OLB	객체 모듈 라이브러리
.OPT	LINK 명령에 대한 입력 옵션 파일
.PAS	Pascal 컴파일러용 입력 소스 파일
.PLI	PL/I 컴파일러용 입력 소스 파일
.STB	Linker 유틸리티로 만든 기호 테이블 파일
.UPD	VAX MACRO 소스 프로그램용 변경 사항에 대한 업데이 트 파일이면서 SUMSLP 유틸리티에 대한 입력

파일 버전

파일 이름과 파일 유형 이외에도 각 파일에는 버전 번호가 있습니다. 버전 번호는 파일 버전을 구분하는 1에서 32,767까지의 십진수입니다. 파일을 만들면 시스템에서는 이 파일에 버전 번호 1을 할당합니다.

여러 버전의 같은 파일이 있을 수 있습니다. 버전 번호를 지정하지 않으면 시스템에서 해당 파일의 가장 높은 기존 버전 번호를 사용합니다. 버전 번호 0을 지정하면 시스템에서 가장 높은 기존 버전을 사용합니다. 파일의 새 버전을 만드는 명령, 어플리케이션 또는 텍스트 편집기(예: EVE)를 사용하여 파일을 수정하는 경우 파일 이름은 그대로 있지만 버전 번호는 1씩 증가합니다.

버전 번호 앞에 세미콜론이나 마침표를 붙이십시오. 시스템에서 파일 지정 방법을 표시할 때 파일 버전 번호의 앞에 세미콜론을 표시합니다.

0이나 음수 버전 번호를 지정하여 상대적인 방식으로 파일 버전을 참조할 수 있습니다. 0을 지정하면 파일의 최신(가장 큰 번호) 버전을 찾습니다. -1을 지정하면 최신 버전 바로 이전 버전을 찾고 -2는 그 이전 버전을 찾는 식입니다. 파일의 최초(가장 작은 번호) 버전을 찾으려면 버전 번호로 -0을 지정하십시오. 버전 번호가 32767보다 큰 파일은 만들 수 없습니다. 버전 번호가 32767보다 큰 새 파일을 만들려고 하면 오류 메시지를 받게 됩니다.

CREATE/DIRECTORY, SET DIRECTORY 및 SET FILE 명령에 대해

/VERSION_LIMIT 한정자를 사용하면 파일의 버전 번호를 제어할 수 있습니다. 버전 제한을 초과하면 시스템에서 가장 낮은 버전 파일을 자동으로 삭제합니다. 예를 들어, 버전 제한이 5인데 파일의 여섯 번째 버전(ACCOUNTS.DAT;6)을 만드는 경우 시스템에서는 파일의 첫 번째 버전(ACCOUNTS.DAT;1)을 삭제합니다. 파일의 버전 제한을 보려면

DIRECTORY/FULL 명령을 입력하십시오. 버전 제한은 File attributes: 필드에 나열되어 있습니다.

네트워크 노드 이름

노드는 컴퓨터 네트워크의 일부인 개별 컴퓨팅 시스템입니다. 사용자 시스템이 네트워크의 일부인 경우 사용자가 로그인할 때 액세스한 노드가 로컬 노드입니다. 네트워크의 다른 노드는 원격 노드입니다. 원격 노드의 파일을 지정하려면 노드 이름을 사용합니다.

노드 지정 방법의 형식은 다음과 같습니다.

```
node["access-control-string"]::
```


파일 지정 방법의 일부로 노드 이름을 입력하는 경우 다음 규칙을 따르십시오.

- 노드 이름은 1-6개의 영숫자를 포함할 수 있으며 적어도 하나의 영문자를 포함해야 합니다. 예를 들면 다음과 같습니다.

AFTP1

F2OTR2

MYNODE

- 액세스 제어 문자열 사용 여부에 상관없이 노드 이름 다음에는 항상 이중 콜론(::)이 와야 합니다.
- 노드 이름을 지정할 때 0-42자의 **액세스 제어 문자열**을 포함시킬 수 있습니다. 액세스 제어 문자열에는 원격 노드로 보낼 로그인 정보가 들어 있습니다. 액세스 제어 문자열에 대한 자세한 내용은 121페이지의 “액세스 제어 문자열 형식”을 참조하십시오.
필요한 이중 콜론은 액세스 제어 문자열 다음에 옵니다.
- 노드 이름 대신에 논리 이름을 사용할 수 있습니다. 논리 노드 이름에 대한 자세한 내용은 11장을 참조하십시오.

DECnet-Plus 노드 전체 이름 지정

OpenVMS 시스템에서 노드 **전체 이름**을 지정할 수 있습니다. 하지만 전체 노드 이름을 인식하도록 **DECnet-Plus** 소프트웨어를 설치해야 합니다.

유효한 전체 노드 이름은 255자까지 포함할 수 있으며 다음 문자를 제외한 모든 문자를 포함할 수 있습니다.

- 공백
- 탭
- 문자: 쉼표(,), 인용 부호(“ ”), 슬래시(/), 감탄 부호(!), 등호(=), 더하기 기호(+), 골뱅이 기호(@), 어포스트로피(?), 괄호(()), 이중 콜론(::)
- 첫 문자나 마지막 문자로 단일 콜론(:)

전체 노드 이름을 인용 부호(“ ”)로 묶으면 일치하지 않는 인용 부호를 제외한 모든 문자를 포함할 수 있습니다. 노드 이름 내에 인용 부호가 있는 경우 인용 부호가 이중으로 있어야 하며 인용 부호를 포함하는 전체 문자열도 인용 부호로 묶어야 합니다.

OpenVMS 소프트웨어에서 노드 이름 구문에 대해 몇 가지 규칙을 적용하지만 유효한 노드 이름의 실제 집합은 사용자 시스템에서 실행되는 DECnet 소프트웨어에 의해 제약을 받습니다. 전체 이름에 대한 자세한 내용은 DECnet-Plus 설명서를 참조하십시오. 유효한 문자 코드를 비롯한 구문 규칙은 *DECnet-Plus DECdns Management Guide*에 자세하게 설명되어 있습니다.

다음 보기에서는 노드 이름에 인용 부호가 있기 때문에 전체 문자열이 인용 부호 안에 있습니다.

```
"MARY:.UNIVERSITY." "SCIENCE LAB" ""
```

유효한 전체 노드 이름에 대한 다른 보기는 다음과 같습니다.

```
MYNODE
```

```
MASSACHUSETTS:.BUSINESS.YOURNODE
```

```
A.B;C
```

TCP/IP 이름 및 주소 지정

TCP/IP의 경우 다른 설명이 없는 한, 명령행에서 호스트를 지정할 때마다 해당 호스트 이름, 정규화된 도메인 이름 또는 IP 주소를 사용할 수 있습니다. 호스트의 상대적 이름은 정규화된 도메인 이름을 포함하지 않는 간단한 이름입니다. 즉, 한 개 이상의 마침표(.)를 포함하지 않습니다. TCP/IP 구문 규칙에 대해서는 *Compaq TCP/IP Services for OpenVMS User's Guide*를 참조하십시오.

DECnet을 사용하여 원격 노드의 파일 액세스

원격 노드의 파일을 액세스하는 경우 DECnet은 원격 노드에서 로그인합니다. 이렇게 하려면 시스템에서는 해당 노드의 로그인 정보를 필요로 합니다. 액세스 제어 문자열을 사용하여 시스템에 제공할 수 있습니다. 액세스 제어 문자열을 생략하면 원격 노드로 전송된 로그인 정보는 다음과 같이 결정됩니다.

- 사용자에게 대한 프록시 로그인 계정이 원격 노드에 있으면 시스템에서 해당 계정을 사용하여 사용자를 로그인시킵니다. 프록시 로그인 계정을 사용하면 선택한 사용자가 노드에 로그인할 수 있습니다.
- 프록시 로그인 계정이 없으면 시스템에서는 로컬 시스템 관리자가 지정한 해당 노드의 기본 DECnet 계정을 사용합니다.

사용자가 액세스 제어 문자열을 포함시키면 시스템에서는 이 문자열을 사용하여 사용자를 원격 노드로 로그인시킵니다. 파일 지정 방법의 나머지 부분은 원격 노드로 전달되어 원격 노드에서 해석됩니다.

파일 지정 방법의 일부로 로컬 노드를 지정하면 파일이 사용자 로컬 노드에 있더라도 시스템에서는 네트워크를 통해 사용자를 로그인시켜 파일 작업을 수행합니다. 원격 시스템을 액세스하는 다른 방법에 대한 자세한 내용은 *OpenVMS System Manager's Manual*을 참조하십시오.

주

이 장의 나머지 부분의 노드 이름을 지정하는 보기에서 항상 액세스 제어 문자열을 포함하는 것은 아닙니다. 이러한 보기에서는 프록시 계정을 사용하면 사용자가 원격 시스템에서 작업을 수행할 수 있기 때문입니다.

TCP/IP를 사용하여 원격 노드의 파일 액세스

OpenVMS용 Compaq TCP/IP 서비스는 FTP(파일 전송 프로토콜)를 제공하여 네트워크를 통해 다른 호스트 간에 파일을 액세스하고 전송할 수 있게 해줍니다. FTP를 사용하려면 OpenVMS용 Compaq TCP/IP 서비스에 액세스할 수 있는 OpenVMS 시스템의 사용자 계정과 원격 FTP 호스트의 사용자 계정이 필요합니다. TCP/IP에서 계정 및 암호를 지정하지 않고 원격 호스트에 연결하는 것을 허용하는 경우도 있습니다. 이 기능이 활성화되어 있지 않은 경우에는 원격 호스트에 대한 사용자 인증 정보를 제공해야 합니다. FTP 명령 사용에 대한 자세한 내용은 *Compaq TCP/IP Services for OpenVMS User's Guide*를 참조하십시오.

네트워크 파일 지정 방법 사용

네트워크 파일 지정 방법에는 다음과 같은 세 가지 형식이 있습니다.

- 기본
- 외부
- 작업

각 형식에서 노드 지정 방법은 액세스 제어 문자열을 포함할 수 있습니다. 자세한 내용은 해당 제품의 *DECnet User's Manual* 및 *Compaq TCP/IP Services for OpenVMS User's Guide*를 참조하십시오.

기본 파일 지정 방법

파일의 기본 형식은 다음과 같습니다.

```
node::device:[directory]filename.type;version
```

외부 파일 지정 방법

외부 파일 지정 방법은 OpenVMS 구문에 맞지 않는 파일입니다. 외부 파일 지정 방법을 제공하기 위해 사용되는 형식은 다음과 같습니다.

```
node::"foreign-file-spec-string"
```

다음 보기에서는 파일 이름에 유효한 파일 이름 문자로 인식되지 않는 물음표(?)가 있으므로 파일 이름을 인용 부호(" ")로 묶어야 합니다. 또한, 사용자가 액세스하는 원격 노드의 운영 체제에서 인식할 수 있는 형식이어야 합니다.

```
$ COPY BOSTON:: "TEST?.DAT" *
```

작업 지정 방법 문자열

작업 지정 방법 문자열은 원격 노드에서 실행되는 프로그램을 식별합니다. 프로그램 내에서 작업 지정 방법 문자열을 사용하여 이 프로그램이 원격 노드의 다른 프로그램과 통신할 수 있게 할 수 있습니다. 작업 지정 방법 문자열을 표시하는 데 사용되는 형식은 다음과 같습니다.

```
node::"task-spec-string"
```

다음 지정 방법은 원격 노드 BOSTON의 프로그램 TEST2를 식별합니다.

```
BOSTON:: "TASK=TEST2 "
```

주

UNIX 시스템에서 파일을 복사하는 경우 몇 가지 제한이 있습니다. 자세한 내용은 *OpenVMS Record Management Utilities Reference Manual*을 참조하십시오.

액세스 제어 문자열 형식

액세스 제어 문자열은 원격 노드에서 로그인할 수 있는 계정을 지명합니다. 액세스 제어 문자열이 있는 노드 이름의 형식은 다음과 같습니다.

`node"access-control-string":`

액세스 제어 문자열을 인용 부호(“”)로 묶고 그 다음에 이중 콜론(:)을 붙입니다.

OpenVMS 시스템에서 액세스 제어 문자열은 사용자 이름, 한 개 이상의 공백이나 탭, 암호 순서로 구성됩니다. 액세스 제어 문자열에 대한 자세한 내용은 10장을 참조하십시오.

다음 보기에서 BOSTON은 네트워크 노드 이름입니다. “HIGGINS ETUHCARAP”은 액세스 제어 문자열입니다.

- HIGGINS는 BOSTON 노드의 사용자 이름입니다.
- ETUHCARAP은 해당 이름과 연결된 암호입니다.

```
$ DIR BOSTON"HIGGINS ETUHCARAP"::WEASEL2:[BORIS]ACCOUNTS.DAT
```

파일 이름과 함께 와일드카드 사용

한 번에 한 파일이 아닌 여러 파일에 DCL 명령을 적용하려면 **와일드카드 문자**를 사용하십시오. 입력한 파일 지정 방법의 일부와 일치하는 모든 파일에 이 명령이 적용됩니다.

이 장의 여러 보기는 파일 작업에서의 와일드카드 문자 사용을 보여 줍니다. DCL 명령에서 와일드카드 문자 사용은 개별 명령에 따라 다릅니다.

많은 DCL 명령과 함께 사용할 수 있는 두 가지 와일드카드는 별표(*)와 백분율 기호(%)입니다. 두 가지 모두 디렉토리 이름, 파일 이름 및 파일 유형에서 와일드카드 문자로 사용할 수 있습니다. 디렉토리와 함께 사용되는 와일드카드에 대한 자세한 내용은 158페이지의 “와일드카드를 사용하여 디렉토리 구조 검색”을 참조하십시오. 버전 구성 요소에서 별표(*)는 사용할 수 있지만 백분율 기호는 사용할 수 없으며 와일드카드와 숫자를 혼합할 수 없습니다.

OpenVMS 버전 7.2 이상을 실행 중인 Alpha 시스템에서 물음표(?)를 백분율 기호(%) 대신 사용할 수 있습니다.

확장 파일 지정 방법을 사용하는 환경에서 작업하는 경우 추가 와일드카드 옵션에 대한 자세한 내용은 5장을 참조하십시오.

별표(*) 와일드카드 문자

다음 사항을 검색하려면 별표(*) 와일드카드 문자를 사용하십시오.

- 디렉토리, 파일 이름 및 파일 유형 필드에서 전체 필드(또는 전체 필드 중 일부)
- 버전 번호 필드의 일부가 아닌 전체 버전 번호 필드

다음과 같은 경우 별표(*) 와일드카드 문자를 사용하십시오.

- 개별적으로 이름을 지정하지 않고 많은 파일을 조작하려는 경우
- 선택한 파일을 좀 더 구체적인 그룹으로 제한하려는 경우
- 디렉토리 지정 방법의 경우

보기

다음 보기에서 파일 지정 방법은 [FROGMAN] 디렉토리에서 모든 파일의 모든 버전을 선택합니다.

```
$ PRINT [FROGMAN]*.*;*
```

다음 보기에서는 파일 유형 .DAT가 있는 현재 기본 디렉토리의 해당 파일만 표시됩니다.

```
$ TYPE *.DAT;*
```

이 보기에서 명령은 [FROGMAN]에서 한 단계 아래의 하위 디렉토리에 있는 파일 유형 .DAT가 있는 모든 파일을 선택합니다.

```
$ DIRECTORY [FROGMAN.*]*.DAT
```

다음 보기에서 와일드카드 문자는 디렉토리 지정 방법에 표시됩니다.

```
$ TYPE [*.*.*]AVERAGE.*;*
```

이 파일 지정 방법은 현재 기본 디스크에서 두 번째 단계의 하위 디렉토리에 있는 파일 유형의 AVERAGE라는 모든 파일의 모든 버전을 선택합니다. 예를 들어, 이 파일 지정 방법은 [A.B.C]AVERAGE.DAT를 선택하지만 [X.Y]AVERAGE.DAT는 선택하지 않습니다.

백분율 기호(%) 와일드카드 문자

백분율 기호(%) 와일드카드 문자를 파일 지정 방법의 단일 문자에 대한 대체로 사용하십시오. 디렉토리, 파일 이름 및 파일 유형 필드에서 백분율 기호를 사용할 수 있습니다. 하지만 버전 번호 필드나 ANSI 자기 테이프 파일 지정 방법에는 백분율 기호를 사용할 수 없습니다. 백분율 기호는 필드에서 한 문자의 위치를 대체하므로 대체할 하나의 문자가 있어야 합니다. 필요한 개수만큼 그리고 다른 와일드카드 문자와 조합하여 백분율 기호를 지정할 수 있습니다.

다음 보기에서는 이름이 DISTRICT이고 그 다음에 한 문자가 있는 모든 .DAT 파일의 최신 버전을 표시합니다.

```
$ TYPE [JONES.TAXES.PROPERTY]DISTRICT%.DAT
```

표시 결과에는 DISTRICT1.DAT, DISTRICT2.DAT 및 DISTRICT3.DAT 파일이 포함됩니다. DISTRICT 뒤에 두 개 이상의 문자가 있으므로 DISTRICT4_5.DAT는 표시되지 않으며 DISTRICT.DAT도 표시되지 않습니다.

다음 보기의 파일 지정 방법은 유효합니다.

```
$ [MA*]INS%%A*.*J*.*
```

기타 파일 이름

다음 절에서는 OpenVMS 환경에서 지원되는 기타 유형의 파일 이름을 설명합니다.

Null 파일 이름 및 파일 유형

파일 이름이나 파일 유형과 같은 파일 지정 방법 구성 요소가 없는 경우 **DCL** 명령이나 유틸리티의 내장된 구문 분석 작업 중에 기본값으로 대체되기도 합니다. 예를 들어, **FORTRAN** 명령은 기본 파일 유형이나 **.FOR**를 사용합니다. 다음 명령은 **FORTRAN** 컴파일러가 **FILE.FOR** 파일을 컴파일하게 만듭니다.

```
$ FORTRAN FILE
```

또한, **DIRECTORY** 명령은 누락된 구성 요소를 별표 와일드카드로 대체합니다. 예를 들어, 다음 명령은 파일 유형에 상관 없이(마침표(.) 포함) 파일 이름 **FILE**이 있는 모든 파일을 표시합니다.

```
$ DIRECTORY FILE
```

파일은 **null(null 값)**인 이름을 갖거나 구분 기호 마침표(**null** 파일 유형이라고도 함)로만 구성된 파일 유형을 가질 수 있습니다. 예를 들어, 다음은 유효한 파일 이름입니다.

```
.TMP  
TEMP.
```

Null 파일 유형을 사용하여 파일 참조

다음과 같이 구분 기호 마침표로만 구성된 유형을 사용하여 파일을 참조할 수 있습니다.

```
$ DIRECTORY TEMP. !
```

파일 이름 구분 기호가 없으므로 **null** 파일 이름을 갖는 파일을 참조할 수는 없습니다. 파일 이름 없이 파일 참조는 항상 파일 이름이 누락된 것으로 해석됩니다.

디렉토리 유틸리티는 누락된 파일 이름을 자동으로 "*"로 대체하므로 다음 명령은 **.TMP** 파일만 표시하는 것이 아니라 **.TMP** 유형이 있는 모든 파일의 목록을 표시합니다.

```
$ DIRECTORY .TMP
```


자기 테이프의 대체 파일 이름

표준(ODS-2 호환 가능) 파일 이름뿐만 아니라 운영 체제에서는 ANSI 레이블이 붙은 자기 테이프의 대체 파일 명명 규칙을 지원합니다. 이 형식은 다음과 같습니다.

"filename";version

파일 이름은 ASCII "a" 문자 집합에서 1-17자를 포함할 수 있습니다. 이 문자 집합에는 숫자, 대문자, 공백 및 다음 문자가 포함됩니다.

!"% '()*+,-./:;<=>?&_

또한, 별표(*) 문자가 ANSI 자기 테이프 파일 이름에서 허용됩니다.

자세한 내용은 *Guide to OpenVMS File Applications*를 참조하십시오.

파일 만들기 및 수정

다음 절에서는 OpenVMS 환경에서 지원되는 도구와 명령을 사용하여 파일을 만들고 수정하는 방법을 설명합니다.

대화형 텍스트 편집기를 사용하여 텍스트 파일을 만들고 수정할 수 있습니다. EVE와 EDT는 OpenVMS 운영 체제에 포함된 두 가지 텍스트 편집기입니다. 다른 텍스트 편집기도 시스템에서 사용할 수 있습니다.

DCL 명령 CREATE, COPY 및 RENAME을 사용하여 파일을 만들고 수정할 수 있습니다. 다음 절에서는 이러한 명령을 사용하여 파일을 만들고 수정하는 방법을 설명합니다.

확장 파일 지정 방법을 사용하는 환경에서 작업하는 경우 해당 환경에서 파일을 만들고 복사하는 방법에 대한 자세한 내용은 5장을 참조하십시오.

파일 만들기

CREATE 명령은 텍스트 파일을 만듭니다. CREATE 명령을 사용하여 파일을 수정할 수는 없습니다. Enter 키를 누른 다음에는 단어를 수정하기 위해 이전 행으로 돌아갈 수 없습니다. CREATE 명령으로 만든 파일을 수정하려면 텍스트 편집기를 사용해야 합니다. Ctrl/Z를 누르면 파일의 끝임을 알려 주고 DCL 명령 수준으로 돌아가게 합니다.

다음 보기에서 TEST.TXT라는 파일은 CREATE 명령을 입력하고 텍스트 행을 입력하여 만든 것입니다.

```
$ CREATE TEST.TXT
this is a test
12345678
Ctrl/Z
```

파일 복사

다음은 복제하는 데 COPY 명령을 사용할 수 있습니다.

- 새 파일에서 기존 파일의 내용 복제
- 한 번에 여러 파일 복제
- COPY 명령과 함께 /SINCE 한정자를 사용하여 지정한 조건에 맞는 파일만 복제

보기

다음 보기에서는 FEES.DAT 파일을 RECORDS.DAT로 복사합니다.

```
$ COPY FEES.DAT RECORDS.DAT
```

다음 보기에서는 기본 디렉토리의 모든 .TXT 파일을 다른 디렉토리로 복사합니다.

```
$ COPY *.TXT;* [SAVETEXT]*.*;*
```

다음 보기에서는 1999년 12월 11일 이후에 수정된 [JONES.LICENSES.DOG] 디렉토리의 해당 파일만 기본 디렉토리로 복사합니다.

```
$ COPY /SINCE=11-DEC-1999/MODIFIED [JONES.LICENSES.DOG]*.* *
```

파일 연결

COPY 명령은 파일을 **연결**하는 데에도 사용할 수 있습니다. 예를 들어, 사용자 기본 디렉토리에서 FEES1.DAT를 FEES.DAT(새 버전의 FEES.DAT 구성)에 추가하려면 다음 명령을 입력합니다.

```
$ COPY FEES.DAT,FEES1.DAT FEES.DAT
```

FEES.DAT 뒤의 쉼표와 파일 이름 FEES1.DAT 사이에는 공백이 없습니다.

DECnet을 사용하여 원격 노드에서 해당 노드로 파일 복사

다른 노드에서 해당 노드로 파일을 복사하려면 COPY 명령을 사용합니다. 예를 들어, CHAOS 노드에 있는 DISK2:[PUBLIC] 디렉토리의 모든 파일의 최신 버전을 사용자 기본 디렉토리에 있는 동일한 이름의 파일로 복사하려면 다음 명령을 입력합니다.

```
$ COPY CHAOS::DISK2:[PUBLIC]*.* *
```

DECnet을 사용하여 사용자 노드에서 원격 노드로 파일 복사

사용자 노드에서 다른 노드로 파일을 복사하려면 COPY 명령을 사용합니다. 여러 시스템에서 파일을 복사하려고 하는 경우 보호 위반이나 DECnet 오류 메시지를 받으면 메일을 사용하여 파일을 복사하거나 액세스 제어 문자열을 사용할 수 있습니다.

다음 보기에서는 기본 디렉토리에 있는 모든 파일의 최신 버전을 CHAOS 노드의 DISK2:[STAFF_BACKUP] 디렉토리에 있는 동일한 이름의 파일로 복사합니다.

```
$ COPY *.* CHAOS::DISK2:[STAFF_BACKUP]
```

TCP/IP를 사용하여 원격 시스템의 파일 복사

TCP/IP는 FTP(파일 전송 프로토콜) 서비스를 사용하여 네트워크를 통해 다른 호스트 간에 파일을 액세스하고 전송합니다. 원격 호스트에서 로컬 호스트로 파일을 복사하려면 **GET** 명령을 사용합니다. 로컬 호스트에서 원격 호스트로 파일을 복사하려면 **PUT** 명령을 사용합니다. 이러한 명령을 사용하려면 원격 호스트와 함께 활성 **FTP** 세션이 있어야 합니다. 세션 중에는 얼마든지 **FTP** 명령을 입력할 수 있습니다. **FTP** 명령 사용에 대한 자세한 내용은 *Compaq TCP/IP Services for OpenVMS User's Guide*를 참조하십시오.

다음 보기에서는 **FEES.DAT** 파일을 **CHAOS** 노드의 **JONES** 계정으로 보냅니다.

```
$ MAIL/SUBJECT="Fee schedule" FEES.DAT CHAOS::JONES
```

액세스 제어 문자열을 사용하여 파일 복사

보호 위반을 받은 다음 파일을 복사하려면 액세스 제어 문자열을 사용하여 파일 지정 방법의 노드 이름을 사용할 수 있습니다(액세스 제어 문자열 형식 참조).

다음 보기에서 사용자는 사용자 이름 **SMITH**와 암호 **SPG96PRT**와 함께 **CHAOS** 노드에 계정이 있습니다. 사용자는 기본 디렉토리에 있는 모든 파일의 최신 버전을 **CHAOS**의 계정으로 복사합니다.

```
$ COPY *.* CHAOS"SMITH SPG96PRT"::DISK2:[STAFF_BACKUP]
```

파일 이름 바꾸기

파일에 새 이름을 지정하여 선택적으로 다른 디렉토리에 위치키려면 **RENAME** 명령을 사용합니다. 이름을 바꾸면 원본 파일이 더 이상 존재하지 않습니다. **RENAME** 명령을 사용하는 경우 입력 및 출력 위치는 같은 장치에 있어야 합니다.

다음 보기에서는 **FEES.DAT** 파일에 새 이름 **RECORDS.DAT**를 지정하고 기본 디렉토리에 서 **[SAVETEXT]** 디렉토리로 이동합니다.

```
$ RENAME FEES.DAT;4 [SAVETEXT]RECORDS.DAT
```

파일 내용 표시

다음 절에서는 OpenVMS 환경에서 지원되는 도구와 명령을 사용하여 파일 내용을 표시하는 방법을 설명합니다.

TYPE 명령 사용

화면에 파일 내용을 표시하려면 DCL 프롬프트에 **TYPE** 명령과 파일 이름을 입력합니다. 시스템에서 기본적으로 파일의 최신 버전을 표시하므로 파일 지정 방법에서 버전 번호를 지정하지 않아도 됩니다.

다음 보기에서는 **STAFF_VACATIONS.TXT** 파일의 최신 버전을 표시합니다.

```
$ TYPE STAFF_VACATIONS.TXT
```

표시 제어

TYPE 명령에 **/PAGE** 한정자를 지정하면 한 번에 한 화면씩 볼 수 있습니다. 시스템에서 다음 화면을 보려면 **Enter** 키를 누르라는 메시지를 표시합니다.

/READ_ONLY 한정자와 함께 대화형 텍스트 편집기(예: **EVE** 또는 **EDT**)를 호출하면 대화형 편집 명령을 사용하여 파일 내에서 이동할 수 있으며 특정 문자 시퀀스를 검색할 수 있습니다. **/READ_ONLY** 한정자를 사용하면 대화형 편집기를 종료하는 경우 수정된 버전의 파일을 만들 수 없습니다.

원격 노드에서 파일 표시

원격 노드에서 파일 내용을 표시하기 위해 **DECnet**을 사용하려면 파일 지정 방법에 노드 이름, 디스크, 디렉토리를 포함시킵니다.

다음 보기에서는 **COMPANY_HOLIDAYS.TXT**(원격 노드 **CHAOS**에 있음) 파일을 표시합니다.

```
$ TYPE CHAOS::DISK2:[PUBLIC]COMPANY_HOLIDAYS.TXT
```

원격 노드에서 파일 내용을 표시하기 위해 **TCP/IP**를 사용하려면 **FTP VIEW** 명령을 사용하고 파일 이름을 지정합니다. 파일이 현재 작업 디렉토리에 없으면 파일 지정 방법에 디렉토리 이름을 포함시킵니다. **FTP VIEW** 명령에 대한 자세한 내용은 *Compaq TCP/IP Services for OpenVMS User's Guide*를 참조하십시오.

와일드카드를 사용하여 파일 표시

별표(*) 와일드카드를 사용하여 특정 파일의 모든 버전을 표시할 수 있습니다.

다음 보기에서는 [JONES] 디렉토리에 있는 LOGIN.COM 파일의 모든 버전을 표시합니다.

```
$ TYPE [JONES]LOGIN.COM;*
```

다음 보기에서는 [JONES] 디렉토리에 있는 STAFF란 단어로 시작하는 모든 파일의 모든 버전 및 모든 파일 유형을 표시합니다.

```
$ TYPE [JONES]STAFF*.*;*
```

여러 파일 표시

TYPE 명령행에서 두 개 이상의 파일을 지정하면 시스템에서 사용자가 지정한 순서대로 파일을 표시합니다. 와일드카드 문자를 사용하면 시스템에서 영문 순으로 파일을 표시합니다.

파일 삭제

DELETE 명령은 디렉토리에서 파일을 제거하고 이 파일에서 사용하던 디스크 공간을 다른 파일이 사용할 수 있도록 릴리즈합니다. **DELETE** 명령을 사용하는 경우 각 파일 지정 방법에서 버전 번호를 지정하거나 버전 번호로 별표(*) 와일드카드 문자를 지정해야 합니다.

예를 들어, **POUND.LIS** 파일의 버전 **17**을 삭제하려면 다음 명령을 입력합니다.

```
$ DELETE POUND.LIS;17
```

POUND.LIS 파일의 버전 **16**과 **17**을 삭제하려면 다음 명령을 입력합니다.

```
$ DELETE POUND.LIS;16,;17
```

POUND.LIS 파일의 모든 버전을 삭제하려면 다음 명령을 입력합니다.

```
$ DELETE POUND.LIS;*
```

와일드카드 문자를 사용하여 여러 파일을 삭제하는 경우 **/CONFIRM** 한정자를 사용하여 각각의 삭제 여부를 확인할 수 있습니다. 마찬가지로 파일을 삭제할 때 파일 이름을 표시하려면 **DELETE** 명령과 함께 **/LOG** 한정자를 지정합니다.

다음 보기에서는 **/CONFIRM** 한정자가 지정되어 있으므로 하위 디렉토리 **[JONES.LICENSES.DOG]**에 있는 모든 파일의 삭제를 확인합니다.

```
$ DELETE/CONFIRM *.*;*
DISK1:[JONES.LICENSES.DOG]FEES.DAT;4, delete? [N]: Y
DISK1:[JONES.LICENSES.DOG]FEMALE.LIS;6, delete? [N]: Y
DISK1:[JONES.LICENSES.DOG]MALE.LIS;3, delete? [N]: N
DISK1:[JONES.LICENSES.DOG]POUND.LIS;17, delete? [N]: Y
```

다음 보기에서는 **/LOG** 한정자가 지정되어 있으므로 시스템에서 파일을 삭제한 다음 파일 이름을 표시합니다.

```
$ DELETE/LOG *.LIS;*_%DELETE-I-FILDEL, DISK1:[JONES.LICENSES.DOG]FEMALE.LIS;6 deleted (35 blocks)
_%DELETE-I-FILDEL, DISK1:[JONES.LICENSES.DOG]MALE.LIS;3 deleted (5 blocks)
_%DELETE-I-FILDEL, DISK1:[JONES.LICENSES.DOG]POUND.LIS;17 deleted (9 blocks)
```

PURGE 명령 사용

PURGE 명령은 기본 디렉토리나 지정된 다른 디렉토리에서 지정한 파일(또는 모든 파일)의 최신 버전을 제외한 모든 버전을 삭제합니다. 파일을 업데이트한 다음 이전 버전을 삭제하면 디스크에서 사용 가능한 디스크 공간을 좀 더 확보할 수 있습니다.

다음 보기에서는 기본 디렉토리에 있는 각 파일의 최신 버전 두 개를 제외한 모든 버전을 삭제합니다.

```
$ PURGE/KEEP=2
```

다른 사용자로부터 파일 보호

다음 절에서는 파일 보호 절차에 대한 개요를 제공합니다. 자세한 보안 정보는 다음을 참조하십시오.

- 4장, 디렉토리 보호에 대한 내용
- 10장, 파일 보호 변경에 대한 자세한 내용

사용 제어 항목(ACL)

다른 사용자가 자신의 파일에 액세스하는 것을 방지하려면 보호를 변경하거나 해당 파일의 사용 제어 항목(ACL)을 수정할 수 있습니다. 보호를 변경하거나 파일의 ACL을 수정하려면 파일을 소유하고 있어야 하고 파일에 대한 액세스 제어를 할 수 있거나 GRPPRV, SYSPRV, BYPASS 또는 READALL 특권이 있어야 합니다.

보호 유형

파일 보호에는 기본 및 명시적 보호라는 두 가지 유형이 있습니다. 파일이 만들어지면 일반적으로 이 파일에는 상위 디렉토리나 같은 보호가 있습니다. 이것이 기본 보호입니다.

CREATE/PROTECTION 명령을 사용하여 파일을 만들거나

SET SECURITY/PROTECTION 명령을 실행하여 기존 파일의 보호를 변경하는 경우는 명시적 파일 보호를 사용하는 것입니다.

파일을 완전하게 보호하려면 파일이 있는 디렉토리에 대한 보호 수준 이상을 적용해야 합니다.

파일 인쇄

파일을 인쇄하려면 **PRINT** 명령을 사용합니다. **PRINT** 명령은 인쇄 작업(인쇄할 모든 파일)을 **인쇄 대기열**이라는 인쇄할 작업 목록에 둡니다. **PRINT** 명령에서 명명된 파일의 파일 유형은 기본적으로 **.LIS**가 되거나 명시적으로 명명된 마지막 파일 유형이 됩니다. 시스템에서는 작업 이름, 대기열 이름, 작업 번호 및 작업 상태를 표시합니다.

기본적으로 작업 이름은 **PRINT** 명령에 있는 첫 번째 또는 유일한 파일 지정 방법의 이름입니다. 작업이 대기열로 전송된 다음에는 작업 번호를 사용하여 작업을 참조합니다. 작업이 대기된 후에 대기열에 선행하는 다른 작업이 없고 프린터가 물리적으로 인쇄할 준비가 된 경우에 이 작업이 인쇄됩니다.

다음 보기에서는 세 개의 파일을 포함하는 인쇄 작업을 기본 인쇄 대기열 **SYS\$PRINT**에 놓습니다.

```
$ PRINT POUND,MALE,FEES.DAT
Job POUND (queue SYS$PRINT, entry 202) started on SYS$PRINT
```

PRINT 명령의 기본 파일 유형은 **.LIS**이므로 **POUND.LIS**, **MALE.LIS** 및 **FEES.DAT** 파일이 대기됩니다. 작업 이름은 **POUND**이며 대기열 이름은 **SYS\$PRINT**, 작업 번호는 **202**입니다.

인쇄 작업 우선 순위

인쇄 대기열은 한 번에 하나의 작업만 실행할 수 있습니다. 인쇄 작업은 일정 **우선 순위**에 따라 인쇄가 예약되며 가장 높은 우선 순위를 갖는 작업이 먼저 인쇄됩니다. 같은 우선 순위의 작업이 두 개 이상인 경우 가장 작은 작업이 일반적으로 먼저 인쇄됩니다. 같은 우선 순위와 같은 크기의 작업은 해당 신청 시간에 따라 인쇄되도록 선택됩니다. 우선 순위는 시스템 관리자가 결정하거나 **PRINT** 명령에 **/PRIORITY** 한정자를 입력하여 결정할 수도 있습니다. 우선 순위 예약에 대한 자세한 내용은 *OpenVMS System Manager's Manual*을 참조하십시오.

대기열 정보 표시

기본 인쇄 대기열 **SYS\$PRINT**는 일반적으로 사이트 특유의 시스템 시작 프로시저의 일부로 시작됩니다. 다음 표에서는 대기열에 대한 정보를 표시하는 데 사용할 수 있는 명령을 보여 줍니다.

표 3-4

표시할 내용	입력할 명령
사이트의 대기열	SHOW QUEUE
인쇄 작업의 상태	SHOW ENTRY
다른 사용자가 대기시킨 작업	SHOW ENTRY/USERNAME= <i>username</i>
특정 작업에 대한 정보	SHOW ENTRY <i>job-name</i> SHOW ENTRY <i>entry-number</i>

다음 보기에서 **SHOW ENTRY** 명령은 대기 중인 인쇄 작업에 대한 정보를 표시하는 데 사용됩니다.

```
$ SHOW ENTRY
```

```

Entry  Jobname      Username      Blocks  Status
-----
  202   POUND          JONES         38     Pending
      On stopped printer queue SYS$PRINT)
```

인쇄 양식

인쇄 양식은 다음 기능을 수행합니다.

- 특정 페이지의 형식화 속성(예: 여백 및 페이지 길이)을 결정합니다.
- 양식에 지정된 용지 스톱에 따라 작업을 인쇄할 수 있는지 결정합니다.

인쇄 요청이 제한된 경우 **Compaq**에서 모든 대기열에 대해 **DEFAULT**라는 시스템 기본 양식을 제공하므로 특수 양식을 사용하지 않아도 됩니다. 시스템 관리자도 인쇄 양식을 만들 수 있습니다. 출력 형식을 지정해야 하거나 특정 인쇄 작업에서 특수 용지가 필요한 경우 시스템 관리자에게 문의하십시오.

인쇄 작업 중지

인쇄 작업을 중지하고 인쇄 대기열에서 삭제하려면 **DELETE/ENTRY** 명령에 대한 항목 번호 매개 변수를 입력합니다.

다음 보기에서는 항목 202를 삭제합니다.

```
$ DELETE/ENTRY=202
```

다른 노드의 파일 인쇄

DECnet이나 TCP/IP 서비스를 사용하면 다른 시스템의 파일을 인쇄할 수 있습니다.

시스템 관리자는 TCP/IP를 사용하여 원격 네트워크 호스트의 인쇄 대기열로 인쇄 작업을 전송하는 **DCL PRINT** 명령을 사용할 수 있게 해주는 **LPR(Line Printer Remote)**과 **LPD(Line Printer Daemon)** 네트워크 서비스와 함께 시스템을 구성할 수 있습니다. 원격 호스트는 LPR/LPD를 실행하는 UNIX 시스템이나 다른 OpenVMS 시스템이 될 수 있습니다. LPR/LPD 네트워크 서비스를 사용하여 다음 작업을 수행할 수 있습니다.

- 원격 네트워크 호스트에 연결된 프린터로 인쇄 작업 보내기
- 인쇄 대기열 상태 표시
- 인쇄 작업 취소
- UNIX 시스템의 사용자가 시작한 인쇄 작업을 로컬 OpenVMS 시스템 인쇄 대기열에서 받기
- SMTP 메일을 통해 “finished” 알림을 받기

LPR/LPD 명령을 사용하여 파일을 인쇄하는 방법을 설명하는 *Compaq TCP/IP Services for OpenVMS User's Guide*를 참조하십시오.

DECnet을 사용하여 다른 시스템의 파일을 인쇄하고 이 파일을 원격 노드로 복사하고 **PRINT** 명령에 대해 **/REMOTE** 한정자를 지정할 수 있습니다.

다음 보기에서는 **COMPANY_HOLIDAYS.TXT** 파일을 로컬 노드에서 원격 노드 **CHAOS**로 복사하고 인쇄를 위해 **CHAOS** 노드의 기본 시스템 인쇄 대기열 (**SYS\$PRINT**)로 이 파일을 대기시킵니다.

```
$ COPY COMPANY_HOLIDAYS.TXT CHAOS"JONES PANDEMONIUM"::DISK2:[JONES]*  
$ PRINT/REMOTE CHAOS::DISK2:[JONES]COMPANY_HOLIDAYS.TXT
```

액세스 제어 문자열은 사용자 JONES에게 CHAOS 노드의 [JONES] 디렉토리로 파일을 복사할 권한이 부여되었음을 표시합니다. 파일 지정 방법의 끝에 있는 별표(*) 와일드카드는 파일이 원격 노드로 복사될 때 파일 이름 COMPANY_HOLIDAYS.TXT를 복제할 것을 시스템에 지시합니다.

주

PRINT 명령에 대한 모든 한정자가 /REMOTE 한정자와 호환되는 것은 아닙니다. 예를 들어, 특정 인쇄 대기열에는 작업을 대기시킬 수 없습니다. 모든 작업은 기본 시스템 인쇄 대기열(SYS\$PRINT)로 보내집니다. /REMOTE와 호환되는 PRINT 명령 한정자 목록은 *OpenVMS DCL Dictionary*에서 DCL 명령 PRINT에 대한 /REMOTE 한정자 설명을 참조하십시오.

PRINT 명령 한정자

인쇄 작업은 PRINT 명령에 대한 한정자를 사용하여 다양한 방법으로 제어할 수 있습니다. 예를 들어, 인쇄할 사본 수를 지정하거나 인쇄 작업이 완료되면 시스템에서 알려 줄 것을 요청할 수 있습니다.

DECprint Supervisor 소프트웨어를 사용자 시스템에서 실행하고 있으면 이 설명서에 나온 한정자뿐만 아니라 /PARAMETER 한정자를 사용하여 가로로, 양면으로 또는 다른 방식으로 인쇄할 수 있습니다. 시스템에서 사용할 수 있는 인쇄 옵션의 목록은 시스템 관리자에게 문의하십시오.

다음 표에서는 PRINT 명령 한정자를 요약하여 나열합니다. PRINT 명령에 대한 자세한 내용은 *OpenVMS DCL Dictionary* 또는 온라인 도움말을 참조하십시오. ¹

표 3-5

인쇄 작업	인쇄 작업 명령 및 한정자
사본 수:	
작업별	PRINT/JOB_COUNT=n ¹
파일별	PRINT/COPIES=n ¹
지정한 파일만	file-spec/COPIES=n ¹
페이지 수	PRINT/PAGES= ¹
인쇄 기능:	
플래그 페이지	PRINT/FLAG= ¹
양식 유형(용지)	PRINT/FORM= ¹
특수 기능	PRINT/CHARACTERISTICS= ¹
두 칸 띄우기	PRINT/SPACE ¹
페이지 제목	PRINT/HEADER ¹
작업 실행 알림	PRINT/NOTIFY
작업 실행 지연:	
지정한 시간 동안	PRINT/AFTER
무기한	PRINT/HOLD
지연된 작업 릴리즈	SET QUEUE/ENTRY/RELEASE
인쇄 작업 표시	SHOW ENTRY
인쇄 작업 중지:	
작업 삭제	DELETE/ENTRY=job-number
현재 작업을 중지하고 대기열 의 다음 작업 인쇄 시작	STOP/ABORT
현재 작업을 중지하고 인쇄하 기 위해 다시 대기시킴	STOP/REQUEUE
대기열의 작업을 완료한 다음 보관	PRINT/RETAIN

1. SET QUEUE/ENTRY 명령의 병렬 한정자를 사용하면 이미 대기 중이지만 아직 인쇄되지 않은 인쇄 작업에 대해 이러한 작업을 지정할 수 있습니다.

WWPPS 유틸리티(Alpha에만 해당)

WWPPS는 포스트스크립트 프린터에서 다양한 언어 문자를 사용하여 텍스트 파일을 인쇄할 수 있게 해주는 유틸리티입니다. 포스트스크립트 인쇄 가능 파일 내에 글꼴 데이터를 포함시키면 프린터에 로컬 언어 글꼴이 없더라도 언어 문자를 인쇄할 수 있습니다.

주

포스트스크립트 인쇄 가능 파일에 글꼴 데이터를 포함시키면 파일 크기가 프린터 메모리에서 지원할 수 있는 크기 이상으로 증가할 수 있습니다. 이러한 경우 WWPPS에서는 오류 페이지를 인쇄된 출력의 끝에 추가하여 해당 파일 크기가 프린터 용량을 초과했음을 알려줍니다.

중국어, 한국어, 일본어 같은 언어를 인쇄하려면 최소 24MB 메모리가 있는 프린터를 사용할 것이 권장됩니다.

지원되는 언어

WWPPS는 다음 언어를 지원합니다.

- 키릴 자모(ISO8859-5)
- 그리스어(ISO8859-7)
- 히브리어(ISO8859-8)
- 일본어(Super DEC Kanji)
- 한국어(DEC Korean)
- 라틴어 1(ISO8859-1)
- 라틴어 2(ISO8859-2)
- 라틴어 4(ISO8859-4)
- 중국어 간체(DEC Hanzi)
- 태국어(TACTIS)
- 중국어 번체(Taiwanese EUC/DEC Hanyu)
- 터키어(ISO8859-9)
- 유니코드

문자를 처리할 때 WWPPS는 현재 로케일에서 해당 문자를 인쇄할 수 있는지 확인합니다. OpenVMS 설치 중에 OpenVMS RTL(Run-Time Library)용 Compaq C에서 로케일 설정을 제공합니다. 16비트 유니코드나 ISO 10646(USC-4) 형식의 파일을 제외하고는 영어 이외의 언어로 된 문자가 들어 있는 파일을 인쇄하려면 적절한 로케일을 설정해야 합니다. 해당 프로세스의 로케일 설정이 입력 파일에 적합하지 않으면 /LOCALE 한정자를 사용하여 인쇄 작업의 로케일을 구체적으로 설정할 수 있습니다.

지원되는 코드 집합

다음 코드 집합은 OpenVMS 시스템에서 지원됩니다.

표 3-6

코드 집합	코드 집합 이름
DECHANYU	중국어 번체용 DECHanyu(Plane 1 및 Plane 2에만 해당)
DECHANZI	중국어 간체용 DECHanzi
DECKOREAN	한국어용 DECKorean
GB18030	중국어 간체 및 번체용 GB18030-2000
ISO8859-1	ISO Latin-1
ISO8859-2	ISO Latin-2
ISO8859-5	ISO Latin-5
ISO8859-7	ISO Latin-7
ISO8859-8	ISO Latin-8
ISO8859-9	ISO Latin-9
SDECKANJI	일본어용 Super DEC Kanji
TACTIS	태국어용 TIS-620

이러한 모든 코드 집합은 WWPPS에서 지원되지만 각 코드 집합에 대한 글꼴은 한 번에 하나의 언어와 연결될 수 있습니다.

또한, WWPPS는 태국어를 제외하고는 이러한 모든 코드 집합에 대해 유니코드 문자 변환을 지원합니다. 유니코드 문자는 이러한 코드 집합 중 하나의 문자로 변환되고 해당 코드 집합을 지원하는 글꼴이 포스트스크립트 파일의 문자용으로 사용됩니다. 문자를 변환할 수 없는 경우 공백으로 인쇄됩니다.

WWPPS 호출

시스템 관리자가 WWPPS에 대한 외부 명령을 이미 설정했을 수도 있지만 아직 설정되어 있지 않은 경우 다음 행을 해당 LOGIN.COM에 추가하여 외부 명령을 설정할 수 있습니다.

```
$ WWPPS :== $SYS$SYSTEM:WWPPS.EXE
```

DCL 프롬프트에서 WWPPS 유틸리티를 호출하려면 다음을 입력합니다.

```
$ WWPPS
```

WWPPS 유틸리티 명령

WWPPS 유틸리티 명령 다음 목록에는 WWPPS 유틸리티에서 사용할 수 있는 명령, 매개 변수, 한정자에 대한 설명이 있습니다. 각각의 설명 다음에 보기가 나옵니다.

EXIT

WWPPS 세션을 종료하고 DCL 명령 수준으로 돌아갑니다. Ctrl/Z이나 Ctrl/C를 눌러 WWPPS 세션을 종료할 수도 있습니다.

```
WWPPS> EXIT
```

HELP

WWPPS(World-Wide 포스트스크립트 인쇄 하위 시스템)에 대한 정보를 얻을 수 있습니다.

```
WWPPS> HELP PRINT
```

개별 명령이나 항목에 대한 정보를 얻으려면 **HELP** 명령 다음에 명령이나 항목 이름을 입력합니다.

```
HELP [topic]
```

PRINT

한 번에 하나의 텍스트 파일을 인쇄 가능한 포스트스크립트 파일로 변환한 다음 프린터 대기열로 보냅니다. 표준 글꼴이나 굵게 문자를 인쇄할 수 있습니다.

PRINT/QUEUE=queue-name [/qualifiers] file-spec

*file-spec*으로 지정한 텍스트 파일을 전송할 대기열 이름을 지정하려면 모든 **PRINT** 명령에서 **/QUEUE** 한정자가 필요합니다. 예를 들어, 다음 명령은 미국 영어로(**/LOCALE** 한정자로 지명) 인쇄할 **REPORT.TXT** 파일을 **PRT_QUEUE** 프린터 대기열로 전송합니다.

```
WWPPS> PRINT/QUEUE=PRT_QUEUE/LOCALE=EN_US_ISO8859-1 REPORT.TXT
```

PRINT 명령의 선택적 한정자는 다음과 같습니다.

- /COPIES

인쇄해야 할 사본의 수를 지정합니다. 기본 사본 수는 1입니다.

- /INDENTATION

왼쪽 여백에서 들여쓸 문자 수를 지정합니다. 기본값은 /INDENTATION=0(들여쓰지 않음)입니다. 허용되는 최대값은 /PAPER_SIZE와 /ORIENTATION에 대해 지정된 값이나 기본값에 따라 다릅니다.

표 3-7

/PAPER_SIZE	/ORIENTATION	/INDENTATION 최대값
LETTER	PORTRAIT	39
A4	PORTRAIT	38
LETTER	LANDSCAPE	65
A4	LANDSCAPE	67

- /LENGTH

행 수로 페이지 길이를 지정합니다. 기본 길이는 LETTER 크기의 경우 66행이며 A4 크기의 경우 68행입니다.

- /LOCALE

WWPPS가 입력 파일을 변환하는 로케일 설정을 지정합니다. 유니코드 형식 (UTF-20)의 텍스트 파일에는 /LOCALE을 지정하지 않아도 됩니다.

로케일은 다음 규칙에 따라 구성됩니다.

language_country_codeset.LOCALE

언어와 국가는 OSF 명명 규칙에서 정의한 대로 각각 두 개의 문자입니다. 사용 가능한 값은 /LOCALE 하위 항목을 참조하십시오. 예를 들어,

EN_US_ISO8859-1은 미국에서 사용되는 영어의 로케일을 나타냅니다.

기본적으로 WWPPS에서는 시스템 지정 로케일이나 프로세스 지정 로케일을 사용합니다. 시스템 지정 로케일이나 프로세스 지정 로케일이 없으면 기본값은 /LOCALE=C입니다.

시스템에 지정된 로케일을 표시하려면 다음 명령을 입력합니다.

```
$ LOCALE SHOW PUBLIC
```

표 3-8에서는 일반적으로 서로 관련된 언어 코드와 국가 코드를 나열합니다.

표 3-8

일반적으로 관련된 언어 코드와 국가 코드

언어 코드	언어	국가 코드	국가
CA	카탈로니아어	ES	스페인
ES	스페인어		
CS	체코어	CZ	체코
DA	덴마크어	DK	덴마크
DE	독일어	CH	스위스
		DE	독일
EL	그리스어	GR	그리스
EN	영어	GB	영국
		US	미국
FI	핀란드어	FI	핀란드
FR	불어	BE	벨기에
		CA	캐나다
		FR	프랑스
HE	히브리어	IL	이스라엘
IW	히브리어		
HU	헝가리어	HU	헝가리
IS	아이슬란드어	IS	아이슬란드
IT	이탈리아어	IT	이탈리아
JA	일본어	JP	일본
KO	한국어	KR	한국
LT	리투아니아어	LT	리투아니아

표 3-8

일반적으로 관련된 언어 코드와 국가 코드 (계속)

언어 코드	언어	국가 코드	국가
NL	네덜란드어	NL	네덜란드
NO	노르웨이어	NO	노르웨이
PL	폴란드어	PL	폴란드
PT	포르투갈어	PT	포르투갈
RU	러시아어	RU	러시아
SK	슬로바키아어	SK	슬로바키아
SL	슬로베니아어	SI	슬로베니아
SV	스웨덴어	SE	스웨덴
TH	태국어	TH	태국
ZH	중국어	HK	홍콩
		TW	대만
		CN	중국

OpenVMS 시스템에서 지원되는 코드 집합은 139페이지의 “WWPPS 유틸리티 (Alpha에만 해당)”의 지원되는 코드 집합에 나열되어 있습니다.

- **/ORIENTATION**

논리 페이지의 인쇄 출력 방향을 **PORTRAIT**(기본값)나 **LANDSCAPE**로 지정합니다.

- **/PAPER_SIZE**

용지 크기를 **LETTER**(기본값)나 **A4**로 지정합니다.

- **/RANGE**

인쇄할 페이지 범위 즉, 시작 페이지 번호 *m*과 마지막 페이지 번호 *n*을 지정합니다. 또는 페이지 범위를 인쇄하는 대신 **ODD**를 지정하여 홀수 페이지만 인쇄하거나 **EVEN**을 지정하여 짝수 페이지만 인쇄할 수도 있습니다. 기본적으로는 전체 문서가 인쇄됩니다.

- **/VERTICAL**

중국어, 한국어 및 일본어 더블바이트 문자에 대한 수직 쓰기 모드를 지정합니다. **/VERTICAL**이 지정되어 있으면 더블바이트 문자가 시계 반대 방향으로 90도 회전하고 행이 왼쪽에서 오른쪽으로 인쇄됩니다. 인쇄된 페이지를 시계 방향으로 90도 회전하면 문자를 오른쪽에서 왼쪽으로 수직 행으로 읽을 수 있습니다. 수직 모드에서 영어와 같은 언어의 싱글바이트 문자는 계속 왼쪽에서 오른쪽으로 수평으로 인쇄됩니다.

- **/WIDTH**

열에서 페이지 너비를 지정합니다. 유효한 값은 다음과 같습니다.

- 80(LETTER 크기 용지 및 PORTRAIT 방향)
- 132(LETTER 크기 용지 및 LANDSCAPE 방향)
- 78(A4 크기 용지 및 PORTRAIT 방향)
- 136(A4 크기 용지 및 LANDSCAPE 방향)

파일을 사용하여 정보 저장

파일 인쇄

4 디렉토리를 사용한 파일 정리

디렉토리는 파일의 이름과 위치를 포함하는 특별한 종류의 파일입니다. 예를 들면, 시스템 관리자가 사용자 계정을 만들 때 사용자 이름과 같은 이름의 디렉토리도 만들어집니다. 사용자 이름이 JONES라면 디렉토리는 [JONES]일 것입니다.

하위 디렉토리는 다른 디렉토리 또는 하위 디렉토리 파일 안에 있는 디렉토리 파일입니다. 하위 디렉토리를 사용하여 파일을 사용하기에 유용한 그룹으로 정리할 수 있습니다. 예를 들면, 메모를 포함하는 하위 디렉토리와 상태 보고서를 위한 하위 디렉토리를 만들 수 있습니다.

하위 디렉토리는 디렉토리와 마찬가지로 그 하위 디렉토리 안에 목록으로 작성된 파일의 이름과 포인터가 들어 있습니다. 한 하위 디렉토리 안에 다른 하위 디렉토리를 포함시키고 다시 그 하위 디렉토리에 다른 디렉토리를 포함시킬 수 있습니다. 이러한 구조(최상위 디렉토리와 하위 디렉토리)를 **계층 디렉토리 구조**라고 합니다.

일반적으로 사용자가 액세스하는 파일은 디스크에 저장됩니다. 각 디스크에는 **마스터 파일 디렉토리(MFD)**라는 주 디렉토리가 있고 MFD에는 **사용자 파일 디렉토리(UFD)** 목록이 포함되어 있습니다. UFD는 사용자의 최상위 디렉토리가 됩니다. 대부분의 경우 시스템 상의 각 사용자에 대해 하나의 UFD가 존재합니다. UFD에는 사용자 디렉토리에 목록으로 작성된 파일의 이름 및 파일에 대한 포인터가 포함되어 있습니다. 최상위 디렉토리는 대개 **프로세스 기본** 디렉토리입니다. 계정을 달리 수정하지 않는 한 로그인할 때 프로세스 기본 디렉토리가 자동으로 최상위 디렉토리가 됩니다.

전체 파일 지정의 장치(디스크) 및 디렉토리 구성 요소를 **파일 경로**라고 합니다. 파일 경로에 파일 이름과 파일 형식 및 버전을 결합하면 전체 파일 지정이 됩니다. 전체 파일 지정에는 파일을 찾는 데 필요한 모든 정보가 들어 있습니다.¹

부분 파일 지정에 대한 기본값 적용하는 방법에 대한 내용은 *Guide to OpenVMS File Applications*를 참조하십시오.

이 장에서는 디렉토리를 사용하여 파일을 정리하고 관리하는 방법에 대해 설명합니다. 다음과 같은 정보가 포함되어 있습니다.

- 디렉토리 구조
- 디렉토리 이해
- 기본값

1. 자기 테이프에 파일을 저장할 수도 있지만 자기 테이프에는 디렉토리 구조가 없습니다. 테이프에 저장된 파일에 액세스하려면 파일 정보만 포함하는 파일 지정을 사용합니다.

- 다른 사용자로부터의 디렉토리 보호
- 와일드카드를 사용한 디렉토리 구조 검색
- UIC 형식 디렉토리 사용

주

이 장에 나오는 보기 중에 노드 이름을 지정하는 보기에 액세스 제어 문자열이 없는 경우도 있습니다. 이는 사용자가 프록시 계정을 통해 원격 시스템 상에서 작업을 수행할 수 있기 때문입니다.

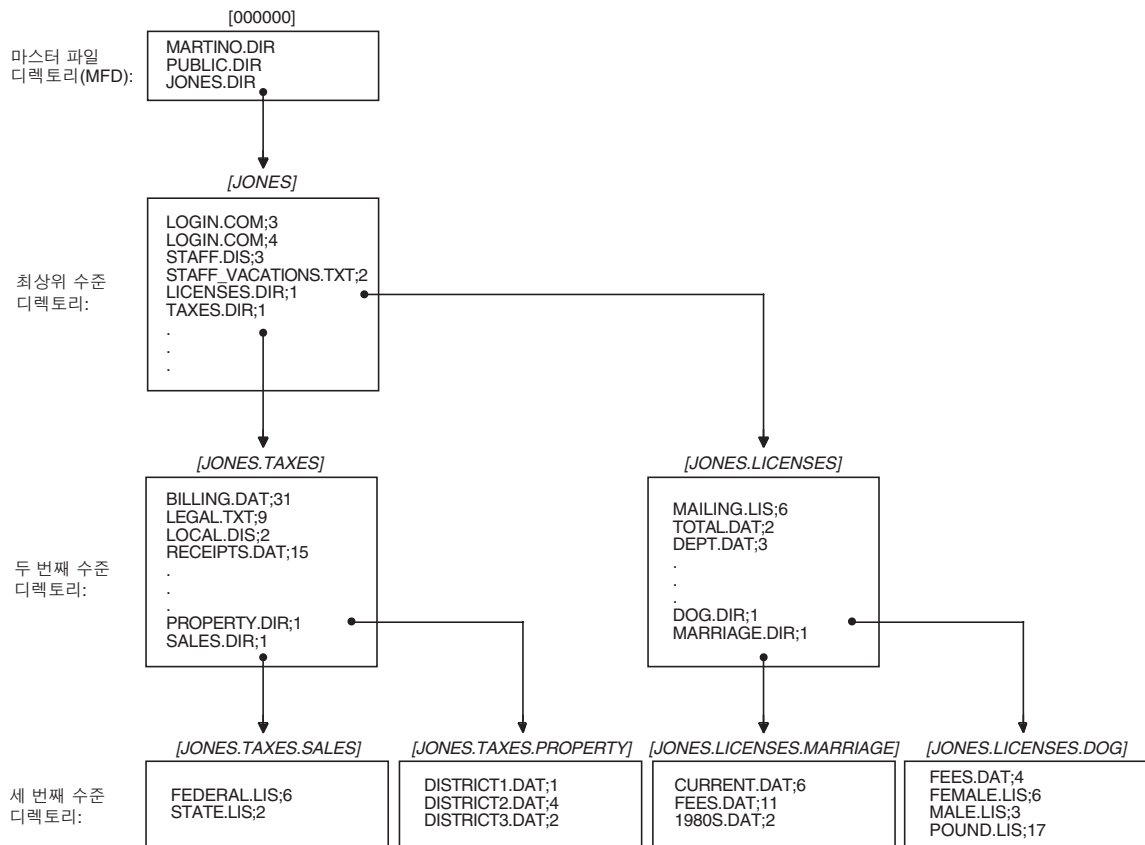
확장 파일 지정 환경에서 작업하는 경우 디렉토리 구조와 구문이 여기에 설명된 일반 구조와 다를 수 있습니다. 그와 같은 환경의 디렉토리 사용에 대한 내용은 5장을 참조하십시오.

디렉토리 구조

그림 4-1은 샘플 디렉토리 계층입니다. 구조의 최상위는 마스터 파일 디렉토리(MFD)이며 디렉토리 이름은 [000000]입니다. 이 샘플에서 MFD에는 MARTINO.DIR, PUBLIC.DIR 및 JONES.DIR를 비롯한 사용자 파일 디렉토리 항목이 있습니다. 최상위 디렉토리인 [JONES]는 [000000]에 있는 JONES.DIR;1이란 이름의 사용자 파일 디렉토리입니다.

이 장에 나오는 많은 보기들은 그림 4-1의 샘플 디렉토리 구조를 기반으로 한 것입니다.

그림 4-1 디렉토리 구조



ZK-1746-GE

이 디렉토리 구조에 대해 다음 사항에 주의하십시오.

- 사용자가 JONES인 것으로 가정합니다. 로그인하면 사용자의 기본 디렉토리인 [JONES]에 위치합니다.
- [JONES]에는 4개의 비디렉토리 파일이 있습니다.

LOGIN.COM;3

LOGIN.COM;4

STAFF.DIS;3

STAFF_VACATIONS.TXT;2

- [JONES]에는 또한 다음 2개의 디렉토리 파일이 있습니다.

LICENSES.DIR;1

TAXES.DIR;1

- 디렉토리 파일 LICENSES.DIR;1은 [JONES.LICENSES] 하위 디렉토리를 가리킵니다.
- TAXES.DIR;1은 하위 디렉토리 [JONES.TAXES]를 가리킵니다.
- [JONES.LICENSES] 하위 디렉토리에는 3개의 비디렉토리 파일과 2개의 디렉토리 파일이 있습니다.
- 디렉토리 파일 DOG.DIR;1은 하위 디렉토리 [JONES.LICENSES.DOG]를 가리킵니다.
- MARRIAGE.DIR은 [JONES.LICENSES.MARRIAGE] 하위 디렉토리를 가리킵니다.

디렉토리 이해

파일 지정의 디렉토리 구성 요소는 몇 개의 하위 디렉토리 이름이 뒤에 올 수 있는 UDF와 같은 최상위 디렉토리 이름으로 이루어집니다. 하위 디렉토리 이름은 마침표(.)로 구분합니다.

OpenVMS Alpha 버전 7.2의 이전 버전과 OpenVMS VAX의 모든 버전은 UFD와 7개 이하의 하위 디렉토리 이름을 포함하는 디렉토리 구성 요소를 지원합니다. OpenVMS Alpha 버전 7.2 이상은 하나의 디렉토리 구성 요소에 UFD와 하위 디렉토리를 합하여 255개의 이름을 지원합니다.

디렉토리 지정 형식은 다음과 같습니다.

[directory.subdirectory]

하나 이상의 하위 디렉토리 수준을 추가하려면 마침표와 각 하위 디렉토리를 위한 디렉토리 이름을 추가합니다. 다른 하위 디렉토리의 하위 디렉토리는 계층에서 한 수준 위의 하위 디렉토리 이름에 마침표와 하위 디렉토리 이름을 붙여 지정합니다.

OpenVMS Alpha 버전 7.2의 이전 버전, OpenVMS VAX 모든 버전, ODS-2 디스크를 사용하는 OpenVMS Alpha 시스템 상에서는 하위 디렉토리 이름이 39자를 넘으면 안됩니다.

ODS-5 디스크를 사용하는 OpenVMS Alpha 버전 7.2 이상에서는 하위 디렉토리가 <subdirectory-name>.DIR;1로 저장되므로 하위 디렉토리 이름이 파일 이름 제한에 의해 제한됩니다. 파일 지정의 디렉토리 및 루트 구성 요소의 총 문자 수는 구분 기호인 중괄호와 마침표를 제외하고 512개를 초과하지 말아야 합니다.

디렉토리 생성

디렉토리를 만들려면 CREATE/DIRECTORY 명령을 입력합니다. 현재 디렉토리 아래에 하위 디렉토리를 만들 경우에는 현재 디렉토리 이름을 지정하지 않고 마침표와 하위 디렉토리 이름을 입력할 수 있습니다.

다음 보기에서는 [JONES.TAXES] 디렉토리가 생성됩니다.

```
$ CREATE/DIRECTORY [JONES.TAXES]
```

다음 보기에서는 현재 기본 디렉토리가 [JONES]이고 하위 디렉토리 [JONES.LICENSES]가 생성됩니다.

```
$ CREATE/DIRECTORY [.LICENSES]
```

디렉토리 표시

디렉토리에 있는 파일들의 이름을 표시하려면 DCL 프롬프트에서 **DIRECTORY** 명령을 입력합니다. 하위 디렉토리에 있는 파일 목록을 표시하려면 **DIRECTORY** 명령을 입력하고 하위 디렉토리 이름 앞에 마침표를 붙여 입력합니다.

DIRECTORY 명령과 함께 특정 명령 한정자를 사용하는 경우에는 파일 이름 외의 정보를 검색할 수 있습니다. **DIRECTORY** 명령 한정자에 대한 자세한 내용은 *OpenVMS DCL Dictionary* 또는 온라인 도움말을 참조하십시오.

다음 보기는 [JONES] 디렉토리에 있는 파일 목록을 표시합니다. [JONES] 디렉토리에는 [JONES.LICENSES]와 [JONES.TAXES]라는 2개의 하위 디렉토리와 4개의 비디렉토리 파일인 STAFF.DIS, STAFF_VACATIONS.TXT 및 2개의 LOGIN.COM 버전이 포함되어 있습니다.

```
$ DIRECTORY
Directory DISK1:[JONES]

LICENSES.DIR;1
LOGIN.COM;3
LOGIN.COM;4
STAFF.DIS;3
STAFF_VACATIONS.TXT;2
TAXES.DIR;1
```

Total of 6 files.

다음 보기에서는 기본 디렉토리는 그대로 [JONES]이고 하위 디렉토리 [JONES.LICENSES]의 내용이 표시됩니다.

```
$ DIRECTORY [.LICENSES]

Directory DISK1:[JONES.LICENSES]

DEPT.DAT;3
DOG.DIR;1
MAILING.LIS;6
MARRIAGE.DIR;1
TOTAL.DAT;2

Total of 5 files.
```

디렉토리 삭제

디렉토리를 삭제하려면 다음 절차를 사용합니다.

표 4-1

단계	작업
1	디렉토리에 파일이 없는지 확인합니다. 디렉토리에 파일이 있는지 알아보려면 DIRECTORY 명령을 사용합니다. 디렉토리에 파일이 없으면 다음 메시지가 표시됩니다. <pre>%DIRECT-W-NOFILES, no files found</pre>
2	디렉토리가 파일을 포함하고 있으면, 그 파일을 다른 디렉토리로 복사하여 저장하거나 파일을 저장할 필요가 없다면 파일을 삭제합니다. 디렉토리가 하위 디렉토리를 포함하고 있으면, 그 하위 디렉토리를 확인하여 그 안에 포함된 파일을 복사하거나 삭제한 다음 하위 디렉토리를 삭제합니다.
3	삭제하려는 디렉토리의 한 수준 위로 디렉토리를 이동합니다. 디렉토리에 하위 디렉토리가 파일처럼 존재하는 것을 기억하십시오. 디렉토리를 삭제할 때 디렉토리를 선택하고 파일을 삭제합니다.
4	파일을 삭제하려면 디렉토리의 파일 보호를 변경해야 합니다. 마스터 파일 디렉토리의 디렉토리 파일을 삭제하려면 SYSPRV 특권이 필요합니다. 파일 보호에 대한 자세한 내용은 3장을 참조하십시오.
5	DELETE 명령을 사용하여 디렉토리를 삭제합니다.

다음 보기는 하위 디렉토리인 [JONES.LICENSES]를 삭제하는 방법을 보여 줍니다.

```
$ SET DEFAULT [JONES.LICENSES]
$ DIRECTORY
%DIRECT-W-NOFILES, no files found
$ SET DEFAULT [JONES]
$ SET SECURITY/PROTECTION=OWNER:D LICENSES.DIR
$ DELETE LICENSES.DIR;1
```

기본값 설정

기본 디렉토리를 변경하려면 **SET DEFAULT** 명령을 사용합니다. 다른 **SET DEFAULT** 명령을 입력하거나 로그아웃할 때까지는 새 기본값이 유효합니다. 하위 디렉토리를 기본값으로 설정하려면 하위 디렉토리 이름을 그보다 한 수준 위의 디렉토리 이름에 추가합니다.

다음 보기에서는 **[JONES]** 디렉토리가 기본값으로 설정되고 **[JONES]STAFF_VACATIONS.TXT**가 표시됩니다.

```
$ SET DEFAULT [JONES]
$ TYPE STAFF_VACATIONS.TXT
```

다음 보기에서는 **[JONES.TAXES]** 하위 디렉토리에 위치한 **BILLING.DAT**가 표시됩니다.

```
$ SET DEFAULT [JONES.TAXES]
$ TYPE BILLING.DAT
```

존재하지 않는 디렉토리를 기본값으로 설정

존재하지 않는 디스크나 디렉토리를 기본값으로 설정할 수 있습니다. 존재하지 않는 디렉토리를 기본값으로 설정하면 파일을 조작하려고 할 때 디렉토리가 존재하지 않는다는 메시지가 표시됩니다. 존재하지 않는 디스크나 디렉토리에 위치하고 있어서 원하는 작업을 수행할 수 없는 경우에는 존재하는 디스크나 디렉토리를 기본값으로 설정하십시오.

SHOW DEFAULT 명령

현재의 기본 디렉토리를 표시하려면 다음 보기와 같이 **SHOW DEFAULT** 명령을 입력합니다.

```
$ SHOW DEFAULT
DISK1:[JONES.TAXES]
$ SET DEFAULT [PUBLIC]
$ SHOW DEFAULT
DISK1:[PUBLIC]
```

SET DEFAULT 명령을 사용하여 기본 장치를 변경할 수 있습니다. 설정된 기본 장치는 다른 **SET DEFAULT** 명령을 입력하거나 로그아웃할 때까지 유효합니다. 또한 명령에 디렉토리를 넣지 않고 기본값을 설정하려는 장치를 지정할 수도 있습니다.

다음 보기에서는 기본 장치를 변경하는 방법을 보여 줍니다.

```
$ SHOW DEFAULT
  DISK1:[JONES]
$ SET DEFAULT DISK2:[GROUP]
$ SHOW DEFAULT
  DISK2:[GROUP]
```

다음 보기에서는 [JONES] 디렉토리가 기본값으로 DISK1과 DISK2에 존재합니다.

```
$ SHOW DEFAULT
  DISK1:[JONES]
$ SET DEFAULT DISK2:
$ SHOW DEFAULT
  DISK2:[JONES]
```

임시 기본값 사용

파일 목록을 입력하고 목록에 포함된 각 파일에 대해 전체 파일 지정을 하지 않으면 노드 이름, 장치 이름, 디렉토리 이름에 임시 기본값이 적용됩니다. 임시 기본값 대신 현재의 기본 디렉토리를 사용하려면 빈 대괄호를 사용합니다. 목록에 표시된 파일에 노드 이름이 포함된 경우에는 2개의 콜론을 사용하여 임시 기본값을 무시할 수 있습니다.

다음 보기에서는 A.LIS와 B.LIS가 [STATS] 디렉토리에서 [RESULTS] 디렉토리로 복사됩니다.

```
$ COPY [STATS]A.LIS,B.LIS [RESULTS]
```

시스템에서는 [STATS]A.LIS와 같이 목록 앞에 파일 지정을 사용하여 B.LIS 파일의 임시 기본 디렉토리도 [STATS]라는 것을 확인합니다.

다음 보기에서는 임시 기본 장치와 2개의 다른 디렉토리가 사용됩니다.

```
$ COPY BASE:[STATS]A.LIS,[TIME]B.LIS,C.LIS [RESULTS]
```

A.LIS, B.LIS 및 C.LIS 세 파일 모두 BASE 장치로부터 복사됩니다. A.LIS 파일은 [STATS] 디렉토리에서 복사되고 다른 두 파일은 [TIME] 디렉토리에서 복사됩니다.

다음 보기에서는 현재 기본 디렉토리는 [BETA]입니다. 이 명령은 [ALPHA]TEST.DAT 및 [BETA]FINAL.DAT를 [RESULTS] 디렉토리에 복사합니다.

```
$ COPY [ALPHA]TEST.DAT,[ ]FINAL.DAT [RESULTS]
```

다른 사용자로부터 디렉토리 보호

파일이 상주하는 디렉토리에 최소한 동일한 보호를 적용하지 않고는 파일을 완전히 보호할 수 없습니다. 예를 들면, 파일에 대한 읽기 권한만 있고 액세스 권한이 하나도 없는 사용자는 파일 내용에 액세스할 수 없지만 그 파일이 존재하는지를 볼 수는 있습니다. 반대로 파일에 대한 액세스 권한은 있지만 파일의 디렉토리(또는 상위 디렉토리 중 하나)에 대한 액세스 권한이 없는 사용자는 파일이 존재하는지를 볼 수 없습니다.

주

개인용 파일을 보호하려면 디렉토리 보호만으로는 부족합니다. 디렉토리 안에 있는 각 파일도 보호해야 합니다.

기본적으로 최상위 디렉토리는 UIC에 따른 보호(S:RWE,O:RWE,G:RE,W:E)를 받으며 ACL 보호는 받지 않습니다. 하위 디렉토리는 상위 디렉토리로부터 UIC에 따라 보호를 받습니다. 보호 코드에 대한 자세한 내용은 394페이지의 “보호 코드 해석”을 참조하십시오.

디렉토리를 만들 때 UIC에 따른 보호를 명시적으로 지정하려면 /PROTECTION 한정자를 CREATE/DIRECTORY 명령과 함께 사용합니다. 디렉토리가 만들어질 때까지는 디렉토리를 위한 ACL을 지정할 수 없습니다. 기존 디렉토리의 UIC에 따른 보호를 변경하려면 디렉토리 파일에 SET SECURITY/PROTECTION 명령을 적용합니다.

읽기 액세스가 아닌 실행 액세스 권한을 지정하여 디렉토리 액세스를 제한할 수는 있지만 금지할 수는 없습니다. 디렉토리에 대한 실행 액세스 권한이 있으면 그 디렉토리에 있는 파일을 조사하고 읽을 수 있습니다. 파일 지정 방법을 이미 알고 있는 경우 파일을 조사할 수는 있지만 디렉토리에 있는 파일의 목록을 표시할 수는 없습니다. 보안에 대한 추가 정보는 *OpenVMS Guide to System Security*를 참조하십시오.

와일드카드를 사용하여 디렉토리 구조 검색

디렉토리 구조의 어느 지점에서나 그 구조에 속한 다른 디렉토리 또는 하위 디렉토리를 참조할 수 있습니다. 디렉토리나 하위 디렉토리 이름을 지정하거나 줄임표(...)와 하이픈(-) 와일드카드 문자를 사용하는 방법이 있습니다. 와일드카드에 대한 추가 정보는 5장을 참조하십시오.

확장 파일 지정 방법 환경에서 작업하는 경우에는 5장의 와일드카드를 사용한 디렉토리 구조 검색 정보를 참조하십시오.

줄임표 와일드카드 문자

줄임표(...) 와일드카드 문자를 사용하여 디렉토리 계층을 따라 검색합니다. 현재 디렉토리 와 그 아래의 모든 하위 디렉토리를 검색하려면 아래와 같이 줄임표만 사용합니다.

```
$ DIRECTORY [...]
```

줄임표를 사용하여 디렉토리 지정을 시작하면 현재 디렉토리부터 검색이 시작됩니다. 그러나 마침표를 사용하여 디렉토리 지정을 시작하는 경우에는 현재 디렉토리보다 한 수준 아래의 하위 디렉토리만 검색됩니다.

모든 최상위 디렉토리와 그 하위 디렉토리를 현재 디렉토리부터 검색하려면 별표(*) 뒤에 줄임표(...)를 사용합니다.

다음 보기에서는 현재 디렉토리가 [JONES]라고 가정하고 그 디렉토리에 있는 FEES.DAT라는 모든 파일의 최신 버전과 그 하위 디렉토리가 모두 표시될 것입니다.

```
$ TYPE [JONES...]FEES.DAT
```

다음 보기에서는 현재 기본 디렉토리가 [JONES]라고 가정하고 .SALES로 끝나는 모든 하위 디렉토리가 검색되며 FEDERAL.LIS 파일의 최신 버전이 표시됩니다.

```
$ TYPE [...SALES]FEDERAL.LIS
```

다음 보기에서는 [JONES] 디렉토리에 있는 모든 최신 버전 DEPT.DAT 파일과 그 하위 디렉토리가 모두 표시됩니다.

```
$ TYPE [...]DEPT.DAT
```

다음 보기에서는 현재 디렉토리가 [JONES]라고 가정하고 [.LICENSES] 하위 디렉토리에서 MAILING.LIS 파일을 검색합니다. 그러나 [JONES.LICENSES.MARRIAGE]는 검색하지 않습니다.

```
$ TYPE [.LICENSES]MAILING.LIS
```

다음 보기에서는 현재 디렉토리가 [JONES]라고 가정하고 그 하위 디렉토리인 [LICENSES]에 있는 모든 최신 버전 DEPT.DAT 파일을 표시합니다.

```
$ TYPE [...LICENSES...]DEPT.DAT
```

다음 보기에서는 8개 수준(최상위 디렉토리와 7개의 하위 디렉토리)에서 존재하는 수준까지 디렉토리 이름을 검색합니다. 이 명령을 사용하려면 READALL 권한이 필요합니다.

```
$ DIRECTORY [*...]
```

하이픈(-) 하위 디렉토리 문자

하이픈 문자는 현재 프로세스 기본 디렉토리 상위의 [sub-]디렉토리를 지정하는 방법으로 사용됩니다. 각 하이픈은 한 수준을 나타냅니다. 하이픈 뒤에 하위 디렉토리 이름(마침표로 구분)을 사용하여 디렉토리 계층을 따라 다른 경로를 지정할 수 있습니다.

하이픈을 너무 여러 개 입력하여 최상위 디렉토리 위를 가리키는 경우에는 오류 메시지가 표시됩니다.

다음 보기에서 현재 프로세스 기본 디렉토리는 [JONES.LICENSES]입니다.

다음 명령은 [JONES] 디렉토리의 최신 STAFF.DIS를 표시합니다.

```
$ TYPE [-]STAFF.DIS
```

다음 보기에서는 [JONES.LICENSES]가 기본 디렉토리입니다.

다음 명령은 [JONES.TAXES]의 최신 BILLING.DAT를 표시합니다.

```
$ TYPE [-.TAXES]BILLING.DAT
```

다음 보기에서 프로세스 기본 디렉토리를 디렉토리 계층에서 현재 수준보다 두 수준 위인 디렉토리로 변경합니다.

```
$ SET DEFAULT [--]
```

ODS-5 디스크 사용 OpenVMS Alpha 버전 7.2 이상에서는 파일 이름과 하위 디렉토리 이름이 하이픈만으로 구성될 수 있습니다. 하이픈으로 구성된 이름의 (sub-)디렉토리와 관련 지정을 구별하려면 하이픈으로 구성된 이름을 지정할 때 최소한 하나의 RMS 이스케이프 문자(^)를 사용해야 합니다. 다음 지정 방법은 현재 프로세스 기본 디렉토리보다 세 수준 위의 디렉토리를 참조합니다.

```
[---]
```

다음 지정은 “---” 디렉토리(UFD)를 참조합니다.

```
[^---]
```

UIC 형식 디렉토리 사용

이 장에서는 이름 형식의 디렉토리 사용 방법을 중점적으로 설명하고 있으나 디렉토리 이름을 UIC 형식으로 지정할 수도 있습니다. UIC 형식에는 사용자 파일 디렉토리(UFD)를 참조하는 두 부분의 8진수로 이루어진 **사용자 식별 코드(UIC)**가 사용됩니다. 파일 지정을 받아들이는 거의 모든 DCL 명령은 UIC 형식의 디렉토리 이름을 인식할 수 있습니다. 일반적으로 실시간 RSX(Resource Sharing Executive) 운영 체제를 사용하지 않는 경우에는 이 형식을 사용할 필요가 없습니다.

UIC 디렉토리 지정 형식은 다음과 같습니다.

[group,member]

예를 들면, [122,1]은 그룹 122의 구성원 1을 나타내는 UIC 디렉토리 지정입니다. UIC 형식 디렉토리 이름은 일반적으로 디렉토리 소유자의 UIC와 일치하지만 반드시 그렇지 않습니다.

UIC 디렉토리를 참조할 때는 다음 규칙을 따르십시오.

- 1-37776 범위의 8진수를 사용하여 그룹을 지정하십시오.
- 0-177776 범위의 8진수를 사용하여 구성원을 지정하십시오.
- 하이픈(-)이나 줄임표(...) 와일드카드는 사용하지 마십시오.

UIC 디렉토리와 와일드카드 사용

별표(*) 와일드카드를 사용하여 UIC 디렉토리를 지정할 수도 있습니다. 예를 들면, [*],6]은 어느 그룹이든 구성원 6인 모든 디렉토리를 나타냅니다. UIC 형식의 디렉토리에 대해서만 검색이 이루어집니다. 디렉토리 지정 [*,*]는 UIC 형식의 모든 디렉토리를 찾습니다. UIC 형식의 모든 디렉토리와 이름이 지정된 모든 디렉토리를 찾으려면 [*]를 사용합니다.

UIC 형식을 이름 형식으로 변환

UIC 형식 디렉토리 이름을 이름 형식으로 변환할 수 있습니다. 필요한 경우에는 그룹과 구성원 번호의 왼쪽에 0을 추가하여 6자로 된 이름을 만듭니다.

UIC 형식과 이름 형식을 결합하여 사용할 수는 없습니다. UIC 형식 디렉토리의 하위 디렉토리 중 하나를 지정하려면 UIC 형식을 이름 형식으로 바꾸십시오.

UIC 디렉토리 지정 [122,1]에 해당하는 이름 형식은 다음과 같습니다.

[122001]

[122,1]SUB.DIR 하위 디렉토리를 참조하려면 이름 형식 디렉토리인 [122001.SUB]를 사용하십시오.

디렉토리를 사용한 파일 정리

UIC 형식 디렉토리 사용

5 확장 파일 지정 방법

OpenVMS Alpha 버전 7.2에서는 다음 두 가지 주요 구성 요소로 구성된 확장 파일 지정 방법이 구현되었습니다.

- 더 광범위한 유효 문자를 사용하여, 더 긴 파일 이름을 지원하는 선택적 볼륨 구조인 ODS-5(On-Disk Structure Level 5)
- 중첩된 디렉토리

종합적으로 이러한 구성 요소는 Windows 환경에 사용되는 파일과 유사한 이름을 가진 파일을 저장, 관리, 처리 및 액세스하는 더 뛰어난 융통성을 OpenVMS Alpha 시스템 (OpenVMS용 Advanced Server를 사용하는)에 제공합니다.

중첩 디렉토리 및 확장 파일 이름은 다음과 같은 이점을 제공합니다.

- OpenVMS 사용자는 긴 파일 이름, 새 문자 지원, 소문자 파일 이름, 대/소문자가 혼합된 파일 이름 등을 사용할 수 있습니다. 이러한 새 기능 덕분에 Windows 사용자는 OpenVMS 파일 서버에서의 파일 작업을 더 확실하게 알 수 있습니다.
- OpenVMS 시스템 관리자는 Windows 사용자가 지정한 이름을 가진 OpenVMS 시스템의 파일을 볼 수 있습니다.
- 중첩 디렉토리를 지원하는 다른 환경의 어플리케이션을 이식하는 어플리케이션 개발자는 OpenVMS에서 병렬 구조를 사용할 수 있습니다.

ODS-5 볼륨 구조

ODS(On-Disk Structure)는 디스크에 저장된 정보에 주어지는 논리적 구조를 말합니다. OpenVMS 운영 체제의 기본 디스크 구조는 ODS-2입니다. ODS-5는 특히 다중 플랫폼 환경에서 유용하게 사용되는 ODS-2의 상위 집합입니다. ODS-5 볼륨 구조는 다음을 제공합니다.

- 긴 파일 이름
- 파일 이름에 사용할 수 있는 더 많은 유효 문자
- 파일 이름의 대/소문자 유지

긴 파일 이름

기본 ODS-2 파일 지정 방법은 파일 이름과 파일 형식을 구분하는 마침표(.) 하나만 지원하는 39.39 형식을 따릅니다.

ODS-5 볼륨의 경우, 파일 이름과 파일 형식을 합친 최대 길이는 8비트 문자 236개 또는 16비트 문자 118개가 될 수 있습니다.¹ 예를 들면 다음과 같습니다.

```
$ CREATE This.File.Name.Has.A.Lot.Of.Periods.DAT
$ CREATE -
_ $ ThisIsAVeryLongFileName^&ItWillKeepGoingForLotsAndLotsOfCharacters.Exceed -
_ $ ingThe39^,39presentInPreviousVersionsOfOpenVMS
$ DIRECTORY
```

```
Directory TEST$ODS5:[TESTING]
```

```
ThisIsAVeryLongFileName^&ItWillKeepGoingForLotsAndLotsOfCharacters.Exceeding
The39^,39presentInPreviousVersionsOfOpenVMS;1
This^.File^.Name^.Has^.A^.Lot^.Of^.Periods.DAT;1
```

```
Total of 2 files.
```

1. 수정되지 않은 프로그램과 유틸리티는 전체 파일 지정 방법을 255바이트로 제한하거나 줄일 수 있습니다.

파일 이름에 사용할 수 있는 더 많은 유효 문자

기본 ODS-2 호환 파일 이름에는 영숫자(A-Z, a-z, 0-9), 달러 기호(\$), 밑줄(_) 및 하이픈(-)이 사용될 수 있습니다. ODS-5는 파일 이름에 사용할 수 있는 더 광범위한 문자 집합을 제공합니다.

ISO LATIN-1 및 유니코드(UCS-2) 문자 집합

ODS-5는 8비트 ISO Latin-1 문자 및 16비트 유니코드(UCS-2) 문자 집합을 사용하는 파일 이름을 지원합니다. ISO Latin-1 다국적 문자 집합은 기본 ASCII 문자 집합의 상위 집합입니다. 확장 파일 지정 방법에서는 별표(*)와 물음표(?)를 제외한 8비트 ISO Latin-1 다국적 문자 집합의 모든 문자를 사용할 수 있습니다.

특수 문자

일부 ISO Latin-1 문자는 제대로 해석되려면 파일 지정 방법에서 앞에 이스케이프 문자가 와야 합니다. 확장 파일 이름에서 RMS와 DCL은 악센트 기호(^)를 이스케이프 문자로 해석합니다. 다음은 이스케이프 문자 사용에 적용되는 규칙을 나열한 것입니다.

- 밑줄(_) 또는 공백 앞에 오는 이스케이프 문자(^)는 공백을 나타냅니다.
- 다음 문자 앞에 오는 이스케이프 문자(^)는 해당 문자가 특별한 의미를 갖는 것이 아니라 파일 이름의 일부로 사용된다는 것을 나타냅니다.
., ; [] % ^ &
- 이스케이프 문자와 함께 또는 이스케이프 문자 없이 리터럴 마침표(.)를 파일 이름에 사용할 수 있습니다. 시스템은 파일 형식 및 버전 번호에 대한 구분 기호 역할을 하지 않은 모든 마침표에 이스케이프 문자를 추가합니다. 디렉토리 이름에 리터럴 마침표(.)를 사용할 경우에는 반드시 이스케이프 문자가 앞에 와야 합니다.
- 이스케이프 문자 뒤에 16진수가 있을 경우에는 또 다른 16진수가 하나 더 와야 합니다. 바로 뒤에 오는 문자 두 개는 임의의 1바이트 문자에 대한 16진수 값으로 해석됩니다. 예를 들어, ^20은 공백을 나타냅니다.
- 파일 지정 방법에서 “U” 앞에 오는 이스케이프 문자는 그 뒤에 오는 4개의 16진수를 유니코드로 해석해야 한다는 것을 나타냅니다(예: ^U012F).

파일 지정 방법에서 이스케이프 문자(^)가 앞에 오지 않는 모든 문자는 ISO Latin-1로 간주됩니다.

주

VAX 시스템에서는 특수 문자가 포함된 파일 이름에 액세스할 수 없습니다. 혼합 아키텍처 환경에 대한 자세한 내용은 182페이지의 “혼합 환경에서 작업”을 참조하십시오.

마침표 해석(.)

확장 파일 이름에서 마침표(.)가 리터럴 문자로 사용되면 RMS는 파일 이름 문자인 마침표와 구분 기호인 마침표를 확인해야 합니다.

확장 파일 이름에서 마침표(.)가 하나만 사용되면 해당 마침표는 구분 기호로 해석됩니다. 이전 버전의 OpenVMS와 마찬가지로 단일 마침표 뒤에 숫자가 올 경우에도 같은 방식으로 처리됩니다. 예를 들어, 다음 명령은

```
$ CREATE Test.1
```

다음 파일을 만듭니다.

```
Test.1;1
```

버전 번호 결정

파일 이름에 마침표(.)가 여러 개 있으면 RMS는 마지막 마침표 뒤에 있는 모든 문자를 확인합니다.

표 5-1

조건	처리 방법
마지막 마침표 뒤에 있는 문자가 모두 숫자일 경우	숫자 문자열이 버전 번호가 됩니다.
마지막 마침표 뒤에 빼기 기호(-)가 있고 그 뒤에 있는 문자가 모두 숫자일 경우	숫자 문자열이 버전 번호가 됩니다.
마지막 마침표 뒤에 6자 이상의 숫자가 있는 경우	RMS는 파일 이름을 잘못된 것으로 거부합니다.
마지막 마침표 뒤에 숫자가 아닌 문자가 있는 경우	파일 형식 구분 기호로 해석됩니다.

예를 들어, 다음 명령은

```
$ CREATE Test4.3.2.1
```

다음 파일을 만듭니다.

```
Test4^.3.2;1
```

여기서 2는 파일 형식이고 1은 파일 버전입니다.

세미콜론(;)으로 명시적으로 구분되는 버전 번호는 5자 이하의 숫자여야 하고, 그 앞에 빼기 기호(-)가 올 수 있습니다.

대/소문자 유지

이전 버전의 OpenVMS에서 DCL과 RMS는 모든 파일 지정 방법을 대문자로 변환했습니다.

ODS-5 볼륨의 경우에는 파일 이름을 대문자, 소문자 또는 대/소문자 혼합으로 입력할 수 있습니다. 모든 파일 이름의 대/소문자는 처음 만들었을 때 그대로 유지됩니다. 예를 들면 다음과 같습니다.

```
$ CREATE KitContents.Txt
```

```
$ DIRECTORY
```

```
Directory  DISK1:[USER1]
```

```
KitContents.Txt;1
```

대/소문자가 다른 동일한 이름을 사용하여 여러 개의 파일을 만들면, DCL은 후속 파일을 원본 파일의 새 버전으로 간주하고 원본 파일과 동일한 대/소문자를 사용하도록 후속 파일을 변환합니다. 예를 들면 다음과 같습니다.

```
$ CREATE CaPri
```

```
$ CREATE CAPRI
```

```
$ CREATE capri
```

```
$ DIRECTORY
```

```
Directory  DISK1:[USER1]
```

```
CaPri.;1  CaPri.;2  CaPri.;3
```

와일드카드 사용

단일 및 다중 문자 와일드카드는 ODS-5 파일에서 제대로 작동합니다. 단일 문자 와일드카드는 파일 이름이나 파일 형식에서 정확히 문자 하나를 나타내지만, 파일 버전 문자열에 사용할 수 없습니다. 다중 문자 와일드카드는 파일 이름이나 파일 형식에서 모든 수의 문자(문자 수가 0개인 경우 포함)를 나타낼 수 있습니다. 다중 문자 와일드카드는 버전 문자열 대신 사용할 수 있습니다.

와일드카드 문자

다음 문자는 항상 유효한 와일드카드 문자 역할을 수행합니다.

- 별표(*)는 다중 문자 와일드카드입니다.
- 백분율 기호(%)는 단일 문자 와일드카드입니다.
- 물음표(?)는 단일 문자 와일드카드입니다.

백분율 기호(%)는 기존 어플리케이션과의 호환성을 유지하기 위해 계속 단일 문자 와일드카드로 사용됩니다. 백분율 기호(%)는 악센트 기호(^)가 앞에 오면 리터럴 문자로 사용되고, Windows 파일 이름에서도 리터럴 문자입니다. 백분율 기호 외에 RMS는 물음표(?)도 단일 문자 와일드카드로 인식합니다. 물음표는 OpenVMS 7.2 이상에서 백분율 기호와 동일한 와일드카드 문자 기능을 수행합니다. 백분율 기호와 물음표는 검색 패턴에서 정확히 문자 하나와 일치합니다.

주

이스케이프 문자(예: ^.) 또는 이스케이프 시퀀스(예: ^EF 또는 ^U0101)는 와일드카드 일치를 위해 단일 문자로 간주됩니다.

와일드카드 구문

DCL이 확장 파일 이름의 대/소문자를 유지하지만 와일드카드 일치는 대/소문자를 무시합니다.

와일드카드를 사용한 검색 작업은 대상 파일의 동일한 부분에 해당 문자가 있는 경우에만 일치합니다. 표 5-2는 몇 개의 와일드카드 검색 예를 보여 줍니다.

표 5-2**와일드카드 및 일치 패턴 예**

패턴	일치	불일치
A*B;*	AHAB;1	A.B;1
A.*B*	A^.DISK.BLOCK;1	A^.C^.B.DAT;1
A?B.TXT;*	A^.B.TXT;5	A^.^.B.TXT;1
*.DAT	Lots^.of^.Periods.dat;1	DAT;1
Mil?no.dat	Milano.dat;1	Millaano.dat;1
NAPOLI?.DAT	napoli.q.dat;1	napoli.abc77.dat;1

중첩 디렉토리 구조

ODS-2 및 ODS-5 볼륨 구조는 다음과 같이 OpenVMS Alpha에서 중첩 디렉토리를 지원합니다.

- 최대 255 수준의 디렉토리를 가질 수 있습니다.
- ODS-2에서 디렉토리 이름 형식은 39.39입니다.
- ODS-5에서 각 디렉토리 이름은 최대 236개의 8비트 문자 또는 118개의 16비트 문자를 포함할 수 있습니다.

예를 들어, 다음과 같은 중첩 디렉토리를 만들 수 있습니다.

```
$ CREATE/DIRECTORY [.a.b.c.d.e.f.g.h.i.j.k.l.m]
```

ODS-5 볼륨에서는 다음과 같이 긴 이름을 가진 디렉토리를 만들 수 있습니다.

```
$ CREATE/DIRECTORY  
[.AVeryLongDirectoryNameWhichHasNothingToDoWithAnythingInParticular]
```

수정되지 않은 어플리케이션에 전체 길이가 255바이트를 초과하는 파일 지정 방법이 제공 되면 RMS는 해당 파일 지정 방법을 간략화합니다.

디렉토리 이름 지정 구문

ISO Latin-1 문자 집합 사용 시에 파일 이름에 적용되던 규칙이 대부분 ODS-5 볼륨의 디렉토리 이름에 그대로 사용됩니다. 마침표와 특수 문자를 디렉토리 이름에 사용할 수 있지만, 경우에 따라서는 표 5-3에 나온 것처럼 앞에 악센트 기호(^)가 있어야 해당 문자가 리터럴 문자로 인식됩니다.

표 5-3

ODS-5 볼륨에서의 디렉토리 이름

CREATE/DIRECTORY...	결과
[Hi^]	Hi^1
[Lots^.Of^.Periods^.In^.This^.Name]	Lots^.Of^.Periods^.In^.This^.Name.DIR;1

디렉토리 ID 및 파일 ID 약어

경우에 따라 수정되지 않은 어플리케이션이 허용하는 255바이트보다 많은 문자가 전체 파일 지정 방법에 포함될 수 있습니다. 수정되지 않은 어플리케이션이 요구하는 파일 지정 방법의 길이가 255바이트를 초과하면 RMS는 디렉토리 이름(필요한 경우)을 각각 디렉토리 ID(DID)와 파일 ID(FID)로 간략화하여 더 짧은 파일 지정 방법을 생성합니다.

파일 지정 방법이 매우 긴 경우, RMS는 우선 디렉토리 ID로 디렉토리를 식별하여 더 짧은 디렉토리 지정 방법(이 지정 방법을 DID라고 부름)을 생성합니다.

```
TEST$ODS5:[5953,9,0]Alghero.TXT;1
```

이 디렉토리 이름 형태는 UIC 형식 디렉토리 이름과 혼동되지 않도록 숫자 세 개와 쉼표 두 개를 포함해야 합니다. DIRECTORY 명령을 사용하면 파일 지정 방법의 전체 버전과 더 짧은 DID 버전을 모두 볼 수 있습니다.

DCL의 확장 파일 지정 방법 분석 기능 사용

기본 DCL 분석 스타일은 ODS-2 스타일 파일 이름에 맞게 설정되어 있습니다.

DCL 명령행에서 확장 파일 이름을 사용할 때는 확장 파일 지정 방법이 허용 및 표시되도록 분석 스타일을 EXTENDED로 설정해야 합니다. 분석 스타일을 설정하려면 다음 명령을 입력합니다.

```
$ SET PROCESS/PARSE_STYLE=EXTENDED
```

이 명령은 OpenVMS VAX 시스템에 영향을 주지 않습니다.

이 명령을 입력하면 DCL은 다음과 같은 파일 이름을 허용합니다.

```
$ CREATE MY^[FILE
```

자세한 내용은 *OpenVMS DCL Dictionary: N-Z*에서 SET PROCESS/PARSE_STYLE 명령에 대한 설명을 참조하십시오.

DCL을 기본 분석 스타일로 다시 설정하려면 다음 명령을 입력합니다.

```
$ SET PROCESS/PARSE_STYLE=TRADITIONAL
```

이 명령을 입력하면 DCL은 ODS-2 파일 이름 형식만 허용합니다.

확장 파일 지정 방법을 사용할 수 있는 경우

일부 DCL 명령과 OpenVMS 유틸리티는 확장 파일 지정 방법을 완벽하게 지원합니다. 확장 파일 이름의 모든 기능을 사용하도록 수정되었기 때문에, 이러한 명령과 유틸리티는 대/소문자 수정이나 오류 없이 확장 파일 지정 방법을 허용 및 처리할 수 있습니다. 또한, 기존의 255바이트 제한을 초과하는 긴 파일 지정 방법을 허용 및 생성할 수 있으며, 이 때 해당 파일 지정 방법을 디렉토리 ID(DID)나 파일 ID(FID) 형식으로 간략화할 필요가 없습니다.¹

기본 지원이 포함된 DCL 명령과 OpenVMS 유틸리티는 확장 파일 이름을 사용하기 위해 따로 수정해야 할 필요가 거의 또는 전혀 없습니다. 이러한 명령과 유틸리티는 확장 파일 지정 방법의 거의 모든 속성(예: 새로운 문자, 중첩 디렉토리 구조 등)을 제대로 처리하지만, 경우에 따라서는 대/소문자가 잘못된 파일 이름을 만들거나 표시할 수 있습니다.

완벽한 지원이 포함된 유틸리티와 달리 기본 지원만 포함된 유틸리티는 긴 파일 지정 방법을 처리하기 위해 RMS가 제공하는 DID 및 FID 약어에 의존합니다. 결과적으로 이러한 유틸리티는 DID 및 FID 약어와 관련된 다음 제한을 받습니다.

- FID 약어가 사용되는 환경에서는 일치 작업이 제대로 수행되지 않을 수 있습니다. 예를 들어, 긴 파일 이름이 간략화된 숫자 FID 형태로 표시되는 경우가 있으므로 와일드카드 일치 작업은 모든 대상 파일 이름을 캡처하지 못할 수 있습니다. 이 제한은 특히 RMS 외부에서 수행되는 일치 작업에 적용됩니다.
- 와일드카드와 고정 기본값을 FID 약어와 함께 사용할 수 없습니다. 예를 들어, 다음 명령은 올바르지 않습니다.

```
$ DIRECTORY a[1,2,3]*.txt
$ COPY a[1,2,3].txt *.txt2
```

FID 약어는 특정 파일의 고유한 숫자 표시이기 때문에 다른 파일과 일치시키거나 다른 파일을 나타내도록 사용할 수 없습니다.

- FID 약어를 사용하여 파일을 만드는 것은 허용되지 않습니다.

DID 및 FID 약어에 대한 자세한 내용은 *Guide to OpenVMS File Applications*를 참조하십시오.

특정 명령 또는 유틸리티에 대한 자세한 내용은 OpenVMS 문서에서 해당 설명서를 참조하십시오.

1. DCL 명령행에서 긴 파일 지정 방법을 입력하는 경우에는 여전히 명령행 길이가 255바이트로 제한됩니다.

확장 파일 이름 지정 지원 없음

확장 파일 이름을 지원하지 않는 OpenVMS 유틸리티와 명령은 ODS-5 볼륨에서 사용할 수 있지만, 기본 파일 지정 방법에 대해서만 작동하도록 제한됩니다. 이러한 유틸리티와 명령은 확장 파일 지정 방법이 적용되었을 때 제대로 작동하는지 확실하지 않으므로 ODS-5 볼륨에서 신중하게 사용해야 합니다.

ODS-5 지원 없음

ODS-5 볼륨 구조를 지원하지 않는 OpenVMS 유틸리티와 명령은 확장 파일 이름을 처리할 수 없습니다. 이러한 유틸리티와 명령은 기본 파일 지정 방법만 적용되었을 때에도 제대로 작동하는지 확실하지 않으므로 ODS-5 볼륨에서 신중하게 사용해야 합니다.

표 5-4에는 확장 파일 이름 또는 ODS-5에 대한 제한 때문에 확장 파일 지정 방법을 지원하지 않는 OpenVMS 유틸리티와 명령이 나와 있습니다.¹

표 5-4

지원되지 않는 OpenVMS 구성 요소

구성 요소	참고
ODS-5 지원 없음	
디스크 조각 모음	ODS-5 볼륨을 지원하도록 특정 조각 모음 도구가 업데이트되었다는 문서가 없으면 지원되지 않습니다. ¹
확장 파일 이름 지정 지원 없음	
코드 컴파일러	객체 파일에 확장 파일 이름을 사용할 수 없습니다. 단, 확장 이름을 지원하는 어플리케이션을 만들 수는 있습니다.
INSTALL 이미지	흔히 사용하는 이미지 형태로 설치할 때 확장 파일 이름을 사용하지 마십시오.
LINK	확장 파일 이름을 사용하여 이미지를 출력할 수 없습니다.
MONITOR	확장 파일 이름을 안정적으로 처리하지 못합니다.
네트워크 파일(NET*.DAT)	확장 파일 이름으로 이름을 바꾸면 안 됩니다.
객체 모듈(.OBJ)	확장 파일 이름으로 이름을 바꾸면 안 됩니다.

1. DFO가 ODS-5 볼륨을 지원하도록 수정되었다는 점에 주의하십시오.

표 5-4

지원되지 않는 OpenVMS 구성 요소 (계속)

구성 요소	참고
페이지 및 스왑 파일	확장 파일 이름을 사용하면 안 됩니다.
SYSGEN	확장 파일 이름을 사용하여 매개 변수를 기록하면 안 됩니다.
시스템 시작 파일	확장 파일 이름으로 이름을 바꾸면 안 됩니다.

확장 파일 이름 표시

일부 DCL 명령은 다음과 같은 새 한정자를 사용하여 확장 파일 이름의 표시를 제어합니다.

```
/STYLE= [CONDENSED | EXPANDED]
```

이 한정자를 사용하면 수정된 DCL 명령이 확장 파일 이름 및 관련된 모든 프롬프트를 표시하는 방법을 제어할 수 있습니다.

CONDENSED 키워드는 많은 유틸리티에서 적용되는 255바이트 문자열 제한을 초과하지 않으면서 파일 지정 방법을 생성된 그대로 표시합니다. 필요할 경우, 이 파일 지정 방법에는 DID 약어 또는 FID 약어가 포함될 수 있습니다. EXPANDED 키워드는 디스크에 저장된 파일 지정 방법을 전체 길이 그대로 표시하고 DID 약어 또는 FID 약어를 포함하지 않습니다.

다음 절부터는 DIRECTORY, TYPE, PURGE 및 DELETE 명령과 함께 /STYLE 한정자를 사용하는 보기가 나와 있습니다.

DIRECTORY 명령

DIRECTORY 명령을 사용하면 디렉토리 내용을 볼 때 표시되는 파일 이름 형식을 선택할 수 있습니다.

```
DIRECTORY/STYLE=(keyword[, keyword])
```

DIRECTORY 명령은 기본적으로 다음 보기에 나온 것처럼 파일 이름을 표시합니다. 즉, 필요할 경우 DID를 사용하고 DID가 필요하지 않은 곳에서는 다시 전체 디렉토리 지정 방법으로 전환합니다.

```
$ DIRECTORY
```

```
Directory TEST$ODS5:[23,1,0]
```

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz  
tuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ  
NOPQRSTUVWXYZ.abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZNOPQRSTUVWXYZabcdef  
ghijklmnopqrst;2
```

```
Total of 1 file.
```

```
Directory TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]
```

```
AddressFiles.DIR;1 LOGIN.COM;3 test.1;1 test^.1.clue;1  
Travel.LIS;1 whee.;5 work.dat;8
```

Total of 8 files.

Grand total of 2 directories, 9 files.

/STYLE 한정자와 함께 키워드 두 개를 모두 사용하는 DIRECTORY 명령은 두 열로 이루어진 디렉토리 목록을 생성합니다. 각 열에는 모든 파일 이름이 나열되는데, CONDENSED 열은 필요한 모든 DID 또는 FID를 포함하고 EXPANDED 열은 전체 디렉토리 이름과 파일 이름을 포함합니다. 모든 파일 오류는 CONDENSED 열에 표시됩니다. 다음 보기에서는 키워드 두 개가 모두 포함된 /STYLE 한정자를 가진 DIRECTORY 명령의 결과를 보여 줍니다.

```
$ DIRECTORY /STYLE=(CONDENSED,EXPANDED)
```

Directory TEST\$ODS5:[23,1,0]	TEST\$ODS5:[TEST.RANDOMTESTING.RANDOM]

abcdefghijklmnpqrstuvwxyABCDEFGH	abcdefghijklmnpqrstuvwxyABCDEFGH
IJKLMNOPSTUVWXYabcdefghijklmnp	IJKLMNOPSTUVWXYabcdefghijklmnp
qrstuvwxyABCDEFGHIJKLMNPQRSTU	qrstuvwxyABCDEFGHIJKLMNPQRSTU
VWXYZabcdefghiJKLMNOPQRSTUVW	VWXYZabcdefghiJKLMNOPQRSTUVW
XYABCDEFGHIJKLMNPQRSTUVWXYZ	XYABCDEFGHIJKLMNPQRSTUVWXYZ
abcdefghijklmnpqrstuvwxy	abcdefghijklmnpqrstuvwxy
ABCDEFGHIJKLMNPQRSTUVWXYZ	ABCDEFGHIJKLMNPQRSTUVWXYZ
abcdefghijklmnpqrst;2	abcdefghijklmnpqrst;2
AddressFiles.DIR;1	AddressFiles.DIR;1
LOGIN.COM;3	LOGIN.COM;3
test.1;1	test.1;1
test^.1.clue;1	test^.1.clue;1
Travel.LIS;1	Travel.LIS;1
whee.;5	whee.;5
work.dat;8	work.dat;8

Total of 8 files.

DIRECTORY는 /STYLE 한정자와 함께 키워드를 하나 또는 둘 다 사용할 수 있습니다.

TYPE 명령

TYPE 명령은 **/STYLE** 한정자를 사용하여 파일과 프롬프트를 입력할 때 시스템 메시지에 표시되는 파일 이름 형식을 선택합니다.

```
$ TYPE/STYLE=(keyword)
```

다음 보기에서는 **TYPE=EXPANDED** 및 **CONFIRM** 한정자와 함께 **TYPE** 명령을 사용하는 것을 보여 줍니다.

```
$ TYPE/CONFIRM/STYLE=EXPANDED abc*.*rst;2
TYPE TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]abcdefghijklmnopqrstuvwxyzABCDEF
GHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyzGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
[N]: Y
```

[System outputs contents of file]

DELETE 명령

DELETE 명령은 **/STYLE** 한정자를 사용하여 명령 수행 시에 표시할 파일 이름 형식을 선택합니다.

```
$DELETE/STYLE=(keyword)
```

다음 보기에서 줄임표(...)는 파일 이름에 문자가 많이 들어 있다는 것을 나타내며, 시스템 메시지를 생성하기 위해 **CONFIRM** 한정자 사용됩니다.

다음은 기본값(**CONDENSED**)이 사용된 **DELETE** 명령입니다.

```
$ DELETE/CONFIRM abc*.*.*
DELETE TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]abcAlphabet.stuff;1 ? [N]: Y
DELETE TEST$ODS5:[23,1,0] abcdefg. . .QRSTUVWXYZ.abcdefg. . .tuvw
xy;1 ? [N]: Y
```

전체 파일 지정 방법이 필요한 경우에는 다음과 같이 **/STYLE** 한정자 및 **EXPANDED** 키워드와 함께 **DELETE** 명령을 사용합니다.

```
$ DELETE/CONFIRM/STYLE=EXPANDED abc*.*.*
DELETE TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]abcAlphabet.stuff;1 ? [N]: Y
DELETE TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]abcdefg. . .QRSTUVWXYZ
Y.abcdefg. . .tuvwxyz;1 ? [N]: Y
```

PURGE 명령

PURGE 명령은 **/STYLE** 한정자를 사용하여 명령 수행 시에 표시할 파일 이름 형식을 선택합니다.

```
$ PURGE/STYLE=(keyword)
```

다음 보기에서 줄임표(...)는 파일 이름에 문자가 많이 들어 있다는 것을 나타내며, 시스템 메시지를 생성하기 위해 **CONFIRM** 한정자 사용됩니다.

다음은 기본값(**CONDENSED**)이 사용된 **PURGE** 명령입니다.

```
$ PURGE/CONFIRM
DELETE TEST$ODS5:[23,1,0]abcdefg. . .QRSTUVWXYZ.abcdefg. . .tuvwxy;1
? [N]: Y
```

전체 파일 지정 방법이 필요한 경우에는 다음과 같이 **/STYLE** 한정자 및 **EXPANDED** 키워드와 함께 **PURGE** 명령을 사용합니다.

```
$ PURGE/CONFIRM/STYLE=EXPANDED
DELETE TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]abcdefg. . .QRSTUVWXYZ.ab
cdefg. . .tuvwxy;1 ? [N]: Y
```

터미널에 확장 파일 이름 표시

확장 파일 이름을 표시하려면 터미널이 **ISO Latin-1** 문자 집합을 표시하도록 설정되어야 합니다. 그렇지 않으면 터미널에 표시되는 문자가 PC에 표시되는 문자와 일치하지 않을 수 있습니다. 터미널에 표시되는 문자 집합을 보거나 변경하려면 **Terminal Setup** 대화 상자를 사용합니다. 표시할 문자 집합을 선택하는 옵션은 대개 **General** 탭에 있습니다.

부록 A에는 **DEC** 다국적 및 **ISO Latin-1** 문자 집합 간에 다르게 사용되는 문자가 나열되어 있습니다.

혼합 환경에서 작업

시스템이 OpenVMS Alpha 7.2 이상을 실행하는 중이면 ODS-5 볼륨에서 모든 확장 파일 지정 방법 기능을 사용할 수 있고, 7.2 이전 버전의 파일과 디렉토리에도 계속 액세스할 수 있습니다. 예를 들어, 다음과 같은 작업을 수행할 수 있습니다.

- ODS-2 볼륨에서 중첩 디렉토리 구조를 작성 및 액세스합니다.
- 이전 버전의 OpenVMS에서 작성된 BACKUP 저장 집합을 읽습니다.
- 이전 버전의 OpenVMS를 실행하는 시스템에서 ODS-5 이름을 가진 파일을 ODS-2 이름을 가진 파일에 복사합니다.

혼합 버전 또는 혼합 아키텍처 OpenVMS Cluster에서 작업하는 경우, 몇 가지 제한을 받게 됩니다. 이전 버전의 OpenVMS를 실행하는 시스템은 ODS-5 볼륨을 마운트하거나, 확장 파일 이름을 제대로 처리하거나, 확장 파일 이름을 표시하지 못합니다. OpenVMS 7.2 이전 버전 사용자는 ODS-5 볼륨의 파일에 액세스할 수 없습니다. 이는 ODS-5 볼륨이 물리적으로 CI 또는 SCSI 버스에서 연결되거나 MSCP 또는 QIO 서버에 의해 연결된 경우에도 마찬가지입니다. 또한, 이전 버전 사용자는 ODS-5 이미지 저장 집합을 만들거나 복원할 수 없습니다. 단, ODS-5 저장 집합트에서 ODS-2 호환 파일 이름을 복원할 수는 있습니다.

OpenVMS 7.2 VAX 시스템이 제공하는 확장 파일 지정 방법 기능은 다음과 같이 제한적입니다.

- ODS-5 볼륨을 마운트하는 기능
- ODS-5 볼륨에서 ODS-2 호환 파일을 관리하고 기록하는 기능
- ODS-5 파일 지정 방법에 액세스할 때 의사 이름(\pISO_LATIN\.... 또는 \pUNICODE\....)을 확인하는 기능

OpenVMS Alpha와 OpenVMS VAX 시스템이 모두 포함된 환경에서 작업하는 경우에는 다음 사항을 알고 있어야 합니다.

- 시스템 유형 및 운영 체제 버전
- 기본 디렉토리가 ODS-2 기반인지, 아니면 ODS-5 기반인지 여부
- 만들려는 파일의 대상이 ODS-2 볼륨인지, 아니면 ODS-5 볼륨인지 여부

OpenVMS 7.2에서 VAX 시스템이 ODS-5 볼륨을 마운트하는 것이 허용되지만, OpenVMS VAX 시스템 사용자는 ODS-2 호환 파일 이름을 가진 파일만 액세스할 수 있습니다.

OpenVMS Alpha 시스템에서 볼륨을 ODS-5로 변환할지 여부를 선택할 수 있습니다. ODS-2 볼륨과 ODS-5 볼륨이 혼합된 환경에서 작업하면서 ODS-5 볼륨에서 파일을 만들 때는 ODS-2 파일 이름의 제한 사항을 염두에 두어야 합니다. 또한 특수 문자가 이름에 포함된 파일을 ODS-5 볼륨에서 ODS-2 볼륨으로 복사하는 경우에는 해당 파일에 ODS-2 호환 이름을 부여해야 합니다.

확장 파일 지정 방법
혼합 환경에서 작업

6 디스크 및 테이프 드라이브 사용

이 장에서는 **OpenVMS** 시스템의 디스크 및 테이프 드라이브 사용에 대한 일반적인 개념을 설명합니다. 디스크 및 테이프 드라이브 등 **OpenVMS** 시스템에 연결된 주변 기기를 장치라고 부릅니다. 로그인하면 기본 장치 및 디렉토리에 대한 액세스 권한이 자동으로 부여됩니다. 공용 장치 및 디렉토리에 액세스할 수 있습니다. 대부분의 경우 사용자 그룹이 공유하는 장치의 설치 및 유지 관리는 시스템 관리자가 담당합니다.

개인용 드라이브 사용 시에는 할당, 초기화 및 장착 방법을 알아야 합니다. 이 장에서는 개인용 디스크 및 테이프 드라이브 액세스 구현에 필요한 다음 개념에 대해 설명합니다.

- 물리적 장치 이름
- 장치 정보 표시
- 논리적 장치 이름
- 일반 장치 이름
- **OpenVMS Cluster** 장치 이름
- 볼륨 및 볼륨 집합
- 장치 관리

추가 정보는 다음을 참조하십시오.

- 이 장에서 설명한 명령에 대한 정보는 *OpenVMS DCL Dictionary* 또는 온라인 도움말을 참조하십시오.
- 장치 사용에 대한 정보는 *OpenVMS System Manager's Manual Volume 1: Essentials*를 참조하십시오.
- **MOUNT** 명령에 대한 정보는 *OpenVMS System Management Utilities Reference Manual*을 참조하십시오.
- **OpenVMS Cluster** 환경의 장치에 대한 정보는 *OpenVMS Cluster Systems*를 참조하십시오.

물리적 장치 이름

시스템 상의 각 물리적 장치는 고유한 **물리적 장치 이름**을 사용하여 식별합니다. 물리적 장치 이름은 디스크 드라이브 또는 터미널 등의 장치 유형을 식별합니다.

대부분의 물리적 장치 이름은 다음 요소들로 구성됩니다.

- 하드웨어 유형을 나타내는 장치 코드
- 장치가 연결된 하드웨어 컨트롤러를 식별하는 컨트롤러 지정자
- 특정 컨트롤러 상의 장치를 식별하는 장치 번호

VTA12, FX09, DAD44 등은 장치 이름입니다.

특정 장치 이름 지정 형식에 대한 정보는 *OpenVMS System Manager's Manual*을 참조하십시오.

장치 정보 표시

시스템 상의 장치에 대한 정보를 표시하려면 **SHOW DEVICES** 명령을 입력합니다. 추가 정보 또는 특정 장치에 대한 정보를 보려면 다음 중 한 가지 방법으로 **SHOW DEVICES** 명령을 입력합니다.

- 밀도, 볼륨, 레이블, UIC 및 마운트된 볼륨의 볼륨 번호를 확인하려면 **SHOW DEVICES/FULL** 명령을 입력합니다.
- 시스템에 구성된 특정 유형의 모든 장치에 대한 정보를 표시하려면 일반 장치 코드 (예: **SHOW DEVICES DK**)를 지정합니다.
- 특정 드라이브에 마운트된 볼륨에 대한 정보를 표시하려면 물리적 장치 이름 (예: **SHOW DEVICES DKA1**)을 지정합니다.

다음 보기에서 **SHOW DEVICES** 명령은 **DAD40**에 대한 정보를 표시합니다.

```
$ SHOW DEVICES DAD40
```

Device Name	Device Status	Error Count	Volume Label	Free Blocks	Trans Count	Mnt Cnt
DAD40:	Mounted wrtlck	0	CHICAGO	540088	1	1

논리적 장치 이름

시스템 관리자가 시스템의 장치를 나타내는 **논리적 장치 이름**을 설정할 수 있습니다. 논리적 장치 이름은 다소 암호 같은 장치 이름으로서 장치에 대해 설명하는 짧은 이름입니다. 장치를 지칭할 때 물리적 장치 이름 대신 이러한 논리적 장치 이름을 사용할 수 있습니다.

자세한 논리적 이름 사용 방법은 11장에서 설명합니다.

일반 장치 이름

일반 장치 이름은 장치 코드로 구성되며 특정 컨트롤러나 장치 번호는 생략합니다.

MOUNT 또는 ALLOCATE 명령과 함께 일반 장치 이름을 사용하면 시스템에서는 사용 가능한 첫 번째 컨트롤러나 물리적 장치 이름이 지정한 일반 장치 이름의 일부와 일치하는 장치를 찾습니다.

그 외의 명령에서 일반 장치 이름을 지정하면 다음과 같이 기본값이 적용됩니다.

- 컨트롤러 지정자를 생략하면 기본값 **A**를 사용합니다.
- 장치 번호를 생략하면 기본값 **0**을 사용합니다.

OpenVMS Cluster 장치 이름

OpenVMS Cluster 장치 이름은 장치가 연결된 노드의 이름과 물리적 장치 이름 사이에 달러 기호(\$)를 사용합니다. 예를 들어, ROXXY\$DUA1는 ROXXY 노드 상의 DUA1 디스크를 말합니다.

일반적으로 이중 경로 OpenVMS Cluster 디스크는 항상 노드 할당 클래스 장치 이름을 사용하여 식별합니다. 모든 OpenVMS Cluster 노드는 항상 이 이름만 인식합니다.

OpenVMS Cluster 환경에서의 장치 이름 형식 사용에 대한 추가 정보는 *OpenVMS Cluster Systems*를 참조하십시오.

두 개의 노드에 연결된 이중 경로 장치인 경우에는 OpenVMS Cluster 장치 이름을 다음 형식으로 지정하십시오.

`$node-allocation-class$ddcu`

요소는 다음과 같습니다.

표 6-1

node-allocation-class	이중 경로 장치를 연결하는 노드에 지정된 값입니다. 예를 들어, \$1\$DJA16은 노드 할당 클래스 값이 1인 두 노드 사이에 이중 경로로 연결된 디스크를 식별합니다.
dd	하드웨어 장치 유형의 장치 코드를 나타냅니다. 예를 들어, DK라는 장치 코드는 RZ23 디스크를 나타냅니다.
c	장치가 연결된 하드웨어 컨트롤러를 식별합니다. 장치 번호는 컨트롤러 지정자와 함께 시스템 하드웨어 구성 내의 장치의 위치를 나타냅니다. 컨트롤러에는 A부터 Z까지가 지정됩니다.
u	특정 컨트롤러 상에 있는 장치 번호를 고유하게 식별합니다. 장치 번호는 0부터 65535까지의 십진수입니다.

볼륨 및 볼륨 집합

OpenVMS 운영 체제에서는 디스크와 테이프를 실제 하드웨어 드라이브와 분리하여 볼륨으로 인식합니다. 볼륨은 정리된 데이터 모음입니다. 또한, 시스템에서는 집합을 인식합니다. 하나의 볼륨 집합은 두 개 이상의 관련된 볼륨으로 구성됩니다. 볼륨을 볼륨 집합으로 묶으면 여러 개의 새 볼륨을 정의하지 않고 같은 집합에 볼륨을 추가함으로 파일에서 사용할 수 있는 공간이 늘어납니다. 볼륨 집합 작성 프로시저는 *OpenVMS System Manager's Manual*을 참조하십시오. 단일 볼륨 작성 프로시저와는 다릅니다.

장치 관리

개인용 디스크 드라이브 사용 시에는 다음과 같은 설치 단계를 잘 알고 있어야 합니다.

표 6-2

단계	작업
1	DCL 명령인 ALLOCATE 를 사용하여 디스크 드라이브를 사용자 프로세스에 할당합니다.
2	DCL 명령인 INITIALIZE 를 사용하여 디스크 볼륨 형식을 지정하고 필요하면 볼륨에 식별 레이블을 기록합니다.
3	DCL 명령인 MOUNT 를 사용하여 사용자 프로세스에서 볼륨과 볼륨에 포함된 파일 또는 데이터에 액세스할 수 있도록 합니다.

장치 할당

장치를 할당할 때 사용자 프로세스에서 독점적으로 사용할 장치를 예약합니다. 해당 장치는 DCL 명령인 **DEALLOCATE**를 사용하여 할당을 취소하거나 로그아웃할 때까지 사용자 프로세스에 할당되어 있습니다.

디스크나 테이프 드라이브를 사용자 프로세스에 할당하려면 **ALLOCATE** 명령을 사용하십시오. **ALLOCATE** 명령 형식은 다음과 같습니다.

```
ALLOCATE device-name[:][,...] [logical-name[:]]
```

요소는 다음과 같습니다.

표 6-3

device-name	볼륨이 로드되는 드라이브를 지정합니다. 물리적 이름, 일반 이름 또는 논리적 이름일 수 있습니다.
logical-name	장치와 연관된 선택적인 논리적 이름을 지정합니다.

블록 초기화

디스크 또는 자기 테이프 블록 초기화는 블록 형식을 지정합니다. 블록을 처음 사용하기 전에 초기화하면 매번 사용할 때마다 초기화할 필요가 없으며 원할 때는 언제든지 완전히 지울 수 있습니다. 블록을 초기화하려면 DCL 명령인 INITIALIZE를 사용합니다. 이 명령은 다음과 같은 작업을 수행합니다.

- 블록 상에 새로운 파일 구조를 작성합니다. 초기화할 때 디스크에 저장되어 있던 데이터는 초기화 과정 중에 삭제됩니다.
- 블록에 식별을 위한 레이블을 기록합니다.
- 소유자 UIC와 블록에 대한 보호를 정의합니다.

주

INITIALIZE 명령을 사용할 때 다른 사용자의 블록을 초기화는 것을 방지하려면 블록을 초기화하기 전에 장치를 할당하십시오.

초기화 권한이 없거나 하는 이유 때문에 다른 사용자가 블록을 초기화하도록 하는 경우에는 블록 레이블, 소유자 UIC 및 **보호 코드**를 제공해야 합니다.

INITIALIZE 명령 형식은 다음과 같습니다.

INITIALIZE device-name[:] volume-label

필드는 다음과 같습니다.

표 6-4

device-name	블록이 물리적으로 마운트되는 장치의 이름을 지정합니다.
volume-label	블록을 식별합니다. 디스크 블록은 12개까지의 영숫자, 자기 테이프 블록은 6개까지의 영숫자를 지정할 수 있습니다.

디스크 블록 초기화

기본적으로 INITIALIZE 명령은 새 블록에 Files-11 구조를 구축합니다. OpenVMS 운영 체제 또는 OpenVMS 운영 체제로 초기화된 디스크 블록의 기본 형식을 Files--11 On-Disk 구조 수준 2라고 합니다. INITIALIZE 명령은 디스크 블록을 Files--11 On-Disk 구조 수준 1 형식으로 초기화할 수도 있습니다.

내용이 기록된 적이 없는 비어 있는 디스크 볼륨이나 사용자의 현재 UIC 또는 UIC [0,0]의 소유인 디스크 볼륨의 경우에는 논리적 보호를 무시하는 특별한 권한이 필요하지 않습니다. 그 외의 모든 경우에는 **VOLPRO** 사용자 권한이 있어야 디스크 볼륨을 초기화할 수 있습니다.

다음 보기에서는 **DKA300** 상의 볼륨을 초기화하고 볼륨에 **ACCOUNTS**라는 레이블을 지정합니다.

```
$ INITIALIZE DKA300: ACCOUNTS
```

볼륨 마운트

디스크 볼륨을 할당한 후 그 볼륨에 포함된 파일을 사용하려면 볼륨을 마운트해야 합니다. DCL 명령인 **MOUNT**를 사용하여 사용자 프로세스에서 볼륨과 그 볼륨에 포함된 파일에 액세스할 수 있도록 합니다.

MOUNT 명령을 입력하면 시스템에서 다음 조건에 부합되는지 확인합니다.

- 다른 사용자에 의해 할당된 장치가 아닌지 확인합니다.
- 장치를 할당할 수 있는 장치 보호인지 확인합니다.
- 볼륨이 지정된 장치에 물리적으로 로드되어 있는지 확인합니다.
- 볼륨 레이블이 지정한 레이블과 일치하는지 확인합니다.

단일 볼륨 또는 볼륨 집합을 마운트할 수 있습니다. 볼륨 집합 작성 및 마운트 프로시저에 대해서는 *OpenVMS System Manager's Manual*을 참조하십시오. 단일 볼륨을 작성하는 프로시저와는 다릅니다.

MOUNT 명령 형식은 다음과 같습니다.

```
MOUNT device-name[:][,...] [volume-label[,...]] [[logical-name[:]]
```

요소는 다음과 같습니다.

표 6-5

device-name	볼륨이 마운트된 장치의 물리적 또는 논리적 이름을 지정합니다.
volume-label	볼륨이 초기화된 레이블을 지정합니다. 다음 MOUNT 한정자 중 하나를 사용하는 경우에는 볼륨 레이블을 지정할 필요가 없습니다. /FOREIGN , /NOLABEL 또는 /OVERRIDE=IDENTIFICATION .
logical-name	장치와 연관된 이름을 정의합니다. 논리적 이름을 생략하면 기본 논리적 이름인 DISK\$volume-label 및 TAPE\$volume-label 이 디스크 및 테이프 드라이브에 각각 지정됩니다.

운영자 지원 요청

운영자가 시스템과 개인용 볼륨의 물리적 마운트(및 분리)를 수행할 수 있습니다. 사용하려는 드라이브에 볼륨이 이미 있는 경우에는 운영자 지원이 필요하지 않습니다.

MOUNT 메시지가 **TAPE** 및 **DISK** 메시지를 받을 수 있는 모든 운영자에게 보내집니다. 예를 들어, 디스크 장치를 마운트하는 데 운영자 지원이 필요한 경우 디스크 운영자에게 메시지가 전달됩니다. **MOUNT** 요청을 받고 응답할 수 있는 운영자가 없는 경우에는 그러한 상황을 알려주는 메시지가 표시됩니다. **/NOASSIST** 한정자를 지정하면 운영자 지원을 피할 수 있습니다.

다음 **MOUNT** 명령은 운영자에게 마운트 요청을 통보하고 터미널에 메시지를 표시합니다.

```
$ MOUNT DKA300: DISK VOL1
%MOUNT-I-OPRQST, PLEASE MOUNT DEVICE _MARS$DKA300:
```

장치가 성공적으로 마운트된 후 다음 메시지가 표시됩니다.

```
%MOUNT-I-MOUNTED, DISK mounted on _DKA300:
```


다음 보기에서는 디스크 볼륨의 할당, 초기화 및 마운트 방법을 보여 줍니다.

```
$ ALLOCATE DKA300: TEMP
%DCL-I-ALLOC, _MARS$DKA300: allocated
$ INITIALIZE TEMP: BACKUP_FILE
$ MOUNT TEMP: BACKUP_FILE
%MOUNT-I-MOUNTED, BACKUP_FILE mounted on _DKA300:
$ CREATE/DIRECTORY TEMP:[ARCHIE]
```

볼륨에 파일을 넣으려면 먼저 **CREATE/DIRECTORY** 명령을 사용하여 디렉토리를 작성해야 합니다.

외부 디스크 볼륨 마운트

Files-11 이외의 파일 구조를 가진 외부 디스크 볼륨을 마운트하려면 다음 보기와 같이 **/FOREIGN** 한정자를 사용하십시오.

```
$ MOUNT/FOREIGN DISK
%MOUNT-I-MOUNTED, BACKUP_FILE mounted on DISK$DMA2:
```

MOUNT/FOREIGN 명령은 시스템에서 볼륨의 내용을 사용할 수 있게 만들지만 파일 구조에 관해서는 아무런 가정을 하지 않습니다. 앞의 보기에서 **MOUNT** 명령은 디스크가 외부 장치로 마운트되었지만 **Files-11** 구조라는 것을 나타내는 볼륨 레이블을 보고합니다. 디스크가 인식되는 파일 구조를 가지고 있지 않은 경우에는 레이블이 표시되지 않습니다.

/FOREIGN 한정자를 사용하여 **Files-11** 구조를 가진 디스크를 마운트하려면 소유자 **UIC**가 일치하는 경우 외에는 **VOLPRO** 사용자 권한이 필요합니다.

개인용 장치의 파일 액세스

개인용 장치에 있는 파일에 액세스하려면 장치 이름을 지정하거나 **SET DEFAULT** 명령을 사용하여 장치와 디렉토리 이름을 기본값으로 설정해야 합니다.

장치의 물리적, 논리적 또는 일반 이름을 사용할 수 있습니다. 또한, **OpenVMS** 클러스 시스템의 일부인 시스템의 경우에는 **OpenVMS Cluster** 시스템의 모든 구성원에서 액세스할 수 있는 장치도 있습니다. 테이프 볼륨 집합의 파일에 액세스하려면 그 볼륨 집합에 할당된 장치를 지정하십시오.

개인 소유 볼륨에서 파일을 인쇄할 수 있기는 하지만 이 경우 파일 인쇄가 완료된 후까지 인쇄할 파일을 포함하는 볼륨이 마운트되어야 합니다.

일부 명령의 경우에는 출력 파일을 지정할 수 있습니다. 다음 보기처럼 프린터나 터미널 같은 레코드 지향 장치의 이름을 출력 파일로 지정할 수 있습니다.

```
$ COPY DFILE.DAT TTB4:
```

위의 보기에서 **COPY** 명령은 **DFILE.DAT** 파일을 **TTB4**라는 이름의 터미널로 보냅니다. 터미널에서는 한 번에 한 레코드씩 파일을 받아서 표시합니다. 장치 이름을 파일 지정 방법으로 사용할 때는 장치 이름 뒤에 콜론(:)을 사용합니다.

볼륨 분리

디스크 또는 테이프 볼륨의 파일 사용을 마치면 **DISMOUNT** 명령을 사용하여 볼륨을 분리할 수 있습니다. 볼륨이 분리되기 전에 분리를 완료하는데 방해가 될 수 있는 조건들을 확인합니다. 예를 들어, 볼륨에 설치된 스왑 및 페이지 파일, 설치된 이미지, 또는 열려 있는 사용자 파일이 포함되어 있는 경우에는 볼륨을 분리할 수 없다는 것을 나타내는 오류 메시지가 표시됩니다.

기본적으로 **DISMOUNT** 명령은 볼륨을 드라이버로터 자동으로 언로드합니다. 볼륨을 분리한 후 다시 마운트하거나 초기화할 계획이 있는 경우에는 다음 보기처럼 **/NOUNLOAD** 한정자를 사용하여 불필요한 처리를 줄이고 시간을 절약할 수 있습니다.

```
$ DISMOUNT/NOUNLOAD MTA1:
```

이 보기에서 자기 테이프 볼륨은 논리적으로 분리되고 테이프는 되감기되지만 물리적으로는 테이프가 **MTA1** 드라이브에 로드된 채로 있습니다.

볼륨을 물리적으로 언로드하기 전에 항상 **DISMOUNT** 명령을 사용하여 분리해야 합니다. 볼륨을 제거하기 전에 드라이브가 언로드되기를 기다리십시오. **SHOW DEVICES** 명령을 입력하여 분리가 완료되었는지 확인할 수 있습니다.

볼륨을 마운트했던 작업에서 로그아웃하면 볼륨이 자동으로 분리되고 언로드됩니다. 그러나 시스템 장애가 발생하면 볼륨이 자동으로 분리되지 않습니다. **ALLOCATE** 명령을 사용하여 할당된 장치를 분리하는 경우에는 **DISMOUNT** 명령을 사용하여 분리한 후에도 할당된 채로 남아 있습니다. **MOUNT** 명령을 사용하여 할당된 장치인 경우에는 **DISMOUNT** 명령을 사용하면 할당이 취소됩니다.

7

Mail을 사용하여 다른 사람과 통신

OpenVMS Mail 유틸리티(MAIL)를 사용하면 현재 사용 중인 시스템이나 OpenVMS 또는 DECnet 네트워크용 Compaq TCP/IP가 있는 시스템에 연결된 다른 컴퓨터의 다른 사용자들에게 메시지를 보낼 수 있습니다. 이 장에서 설명하는 내용은 다음과 같습니다.

- 우편 호출 및 종료
- 메시지 읽기
- 메시지 보내기
- 네트워크를 통해 우편 보내기
- 여러 사용자에게 메시지 보내기
- 우편의 파일 조작
- 메시지를 보내는 기타 방법
- 메시지 구성
- 메시지 삭제
- 우편 메시지 인쇄
- 우편 파일 보호
- Mail에서 텍스트 편집기 사용
- Mail 환경 사용자 정의
- Mail 명령 요약
- MIME 유틸리티 사용

추가 정보를 보려면 다음을 참조하십시오.

- Mail 명령과 한정자에 대한 자세한 내용을 보려면 DCL 프롬프트에서 **HELP MAIL** 명령을 입력하거나 **MAIL>** 프롬프트에서 **HELP** 명령을 입력합니다.
- 사용자 계정을 통한 Mail 사용 제어에 대한 자세한 내용은 *OpenVMS System Manager's Manual*을 참조하십시오.
- TCP/IP 우편 명령과 한정자에 대한 자세한 내용을 보려면 DCL 프롬프트에서 **HELP TCPIP_SERVICES**를 입력합니다.
- TCP/IP 서비스를 사용하여 우편을 보내고 받는 데 대한 자세한 내용은 *Digital TCP/IP Services for OpenVMS User's Guide*를 참조하십시오.

다음 그림에서는 우편 메시지 예와 구성 요소를 보여 줍니다.



ZK-0980A-GE

우편 호출 및 종료

다음 절에서는 Mail을 호출하고 종료하는 방법에 대해 설명합니다.

Mail 호출

Mail 유틸리티를 호출하려면 다음과 같이 DCL 명령 MAIL을 입력합니다.

```
$ MAIL  
MAIL>
```

일단 Mail 유틸리티 안에 있으면 MAIL> 프롬프트에서 적절한 명령을 입력한 다음 Enter 키를 눌러서 다음 작업을 수행합니다.

- 우편 메시지 읽기
- 우편 메시지 보내기
- 우편 메시지에 응답
- 우편 메시지 전달
- 우편 메시지를 파일과 폴더로 구성
- 우편 메시지 삭제
- 우편 메시지 인쇄

Mail 종료

Mail을 종료하려면 다음과 같이 MAIL> 프롬프트에서 EXIT 명령을 입력합니다.

```
MAIL> EXIT  
$
```

또한 Ctrl/Z를 누르거나 QUIT 명령을 사용하여 Mail을 종료할 수도 있습니다.

메시지 읽기

Mail은 사용자가 받는 메시지를 기본 파일 유형인 .MAI로 된 우편 파일에 저장합니다. 이 파일에서 Mail은 기본적으로 이전 메시지와 새 메시지를 저장하는 두 개의 **폴더**를 제공합니다. 새 메시지는 **NEWMAIL**이라고 하는 폴더에 자동으로 보관되며, 이전 메시지는 **MAIL**이라는 폴더에 보관됩니다. 새 메시지를 읽은 후에는 **FILE**, **MOVE** 또는 **DELETE** 명령을 입력하지 않은 경우 메시지가 자동으로 **NEWMAIL** 폴더에서 **MAIL** 폴더로 이동됩니다. 사용자가 새 우편 메시지를 모두 읽은 후 다른 폴더를 선택하거나 Mail을 종료하면 Mail에서 **NEWMAIL** 폴더를 삭제합니다.

새 우편 읽기

사용자가 자신의 계정에 로그인하여 우편 메시지를 받으면 Mail이 사용자에게 알려 줍니다. 예를 들어, **FELLINI** 사용자가 보낸 메시지에 대한 알림은 다음과 같이 표시됩니다.

```
New mail on node DOODAH from STONE::FELLINI      (10:02:23)
```

새 메시지를 읽으려면 다음과 같이 Mail을 호출하고 **MAIL>** 프롬프트에서 **Enter** 키를 누릅니다.

```
$ MAIL
You have 1 new message.
MAIL>
```

새 메시지가 두 개 이상일 때는 **MAIL>** 프롬프트에서 **Enter** 키를 눌러 다른 메시지를 읽습니다. 새 메시지를 모두 읽으면 Mail은 “%MAIL-E-NOMOREMSG, no more messages”라는 메시지를 표시하고 사용자가 Mail을 종료할 때까지 계속해서 명령 프롬프트가 나타납니다.

Mail 안에 있는 상태에서 우편 메시지를 받는 경우에는 **READ/NEW** 명령을 입력하여 새 메시지를 읽습니다.

이전 메시지 읽기

기본 Mail 폴더에서 이전 우편 메시지를 다시 읽으려면 다음 절차를 수행하십시오.

표 7-1

단계	작업
1	MAIL> 프롬프트에서 SELECT 명령을 입력합니다. 예를 들면 다음과 같습니다. <pre>MAIL> SELECT MAIL</pre> Mail은 MAIL이라는 폴더를 표시합니다.
2	기본 MAIL 폴더의 첫 번째 메시지를 읽으려면 MAIL> 프롬프트에서 Enter 키를 누르거나 READ 명령을 입력합니다. Mail은 기본 우편 파일의 첫 번째 메시지 (1)를 표시합니다.
3	다음 메시지를 표시하려면 Enter 키를 누릅니다. 메시지가 너무 길어서 한 화면에 표시할 수 없을 경우, Enter 키를 누르면 다음 부분의 메시지가 표시됩니다. 메시지의 나머지를 건너 뛰고 다음 메시지를 표시하려면 NEXT 명령을 입력합니다.

기본 MAIL 폴더의 특정 메시지를 읽으려면 다음 절차를 수행하십시오.

표 7-2

단계	작업
1	MAIL> 프롬프트에서 DIRECTORY 명령을 입력합니다. 메시지의 부속 집합을 목록에서 선택하려면 DIRECTORY 명령 한정자 /FROM 또는 /SUBJECT를 사용합니다.
2	MAIL> 프롬프트에서 읽을 메시지의 번호를 입력합니다. Mail은 선택한 메시지를 표시합니다.

다음 보기에서는 이전 메시지를 표시하는 데 **DIRECTORY** 명령을 사용하고 메시지 2를 선택하여 읽습니다.

MAIL> DIRECTORY

MAIL

#	From	Date	Subject
1	STONE::FELLINI	11-DEC-1999	Sales presentation on May 11
2	DOODAH::JONES	11-DEC-1999	Status

MAIL> 2

메시지 검색

메시지가 많은 경우에는 여러 개의 메시지에서 문자열을 찾는 데 **SEARCH** 명령을 사용하여 특정 메시지를 찾을 수 있습니다. 문자열을 검색하려면 해당 문자열을 **SEARCH** 명령에 대한 매개 변수로 지정합니다.

새 문자열을 지정할 때마다 **SEARCH** 명령은 메시지 번호 1부터 검색을 시작합니다. 지정된 문자열이 포함된 메시지를 폴더에서 계속 검색하려면 매개 변수를 지정하지 않은 상태에서 **SEARCH** 명령을 사용합니다. 다른 폴더에서 동일한 문자열을 검색하려면 **SELECT** 또는 **SET FOLDER folder-name** 명령을 입력하고 매개 변수를 지정하지 않은 상태에서 **SEARCH** 명령을 계속 사용합니다.

다음 보기에서는 현재 폴더의 메시지에서 *appointment*라는 문자열이 포함된 첫 번째 메시지를 검색합니다.

MAIL> SEARCH "appointment"

메시지 보내기

시스템에서 임의의 사용자에게 우편 메시지를 보내려면 다음을 수행하십시오.

표 7-3

단계	작업
1	MAIL> 프롬프트에서 SEND 를 입력합니다. Mail 은 메시지를 받을 사용자의 이름을 입력하라는 프롬프트를 표시합니다.
2	메시지를 받을 사용자의 이름을 입력하고 Enter 키를 누릅니다. Mail 은 메시지의 제목을 입력하라는 프롬프트를 표시합니다.
3	메시지의 제목을 입력하고 Enter 키를 누릅니다. 이 정보 입력은 선택 사항입니다. Mail 은 메시지의 텍스트를 입력하라는 프롬프트를 표시합니다.
4	메시지의 텍스트를 입력하거나 Enter 키를 누릅니다. 이 정보 입력은 선택 사항입니다.
5	Ctrl/Z 를 눌러 메시지를 보냅니다. 메시지를 보내지 않기로 결정하여 Ctrl/C 를 누르면 Mail 을 종료하지 않은 상태에서 보내기 작업이 취소됩니다.

다음 보기에서는 **THOMPSON**이라는 사용자에게 메시지를 보냅니다.

```
MAIL>SEND
To: THOMPSON
Subj: Meeting on April 20
Enter your message below. Press CTRL/Z when complete, or CTRL/C to quit:
I have some new ideas about the Hubbub Cola account.
Let me know when you are available to talk about them.

--Jeff
```

네트워크를 통해 우편 보내기

다음 절에서는 네트워크를 통해 우편을 보내는 방법에 대해 설명합니다.

네트워크 프로토콜 지정

사용자가 메시지를 받으면 Mail은 지정된 주소를 다음과 같이 해석합니다.

- 주소의 노드 구성 요소에 마침표 (.)가 있으면 이 주소를 인터넷 주소로 해석합니다.
MAIL\$INTERNET_TRANSPORT 논리 이름으로 대체 프로토콜을 정의하여 시스템에서 다른 인터넷 프로토콜을 사용하도록 이전에 설정하지 않았다면 Mail은 SMTP 프로토콜을 기본적으로 사용합니다.
- 주소의 노드 구성 요소에 마침표가 들어 있지 않으면 이 주소를 DECnet 주소로 해석합니다.

그러나 시스템에서 특정 프로토콜을 강제로 선택하도록 Mail 환경을 사용자 정의할 수 있습니다. 이 옵션은 우편 주소가 인터넷이나 DECnet 프로토콜에 대해 유효한 것으로 해석될 수 있는 경우에 유용합니다.

프로토콜을 지정하려면 MAIL\$INTERNET_MODE 논리 이름을 다음과 같이 정의할 수 있습니다.

- HYBRID(기본값)
주소의 노드 구성 요소에 마침표 (.)가 있으면 Mail에서 인터넷 프로토콜로 사용합니다. 마침표가 없으면 Mail은 DECnet 프로토콜을 사용합니다.
- DECNET
Mail은 주소의 노드 구성 요소를 항상 DECnet 노드 지정 방법으로 해석합니다.
- SMTP
Mail은 주소의 노드 구성 요소를 항상 인터넷 주소 지정 방법으로 해석합니다.
MAIL\$INTERNET_TRANSPORT 논리를 사용하여 대체 인터넷 전송을 정의하지 않는 경우 기본 전송은 SMTP입니다.

이러한 방법 중 하나로 Mail 환경을 수정하려면 MAIL\$INTERNET_MODE 및 MAIL\$INTERNET_TRANSPORT 논리 이름을 LOGIN.COM 파일에 정의하는 것이 좋습니다. 논리 이름 사용과 정의에 대한 자세한 정보는 11장을 참조하십시오.

예를 들어, 시스템이 기본값(HYBRID)을 사용하도록 설정된 경우 Mail 주소 smith@pluto는 주소에 마침표가 없으므로 DECnet 주소로 해석됩니다. 그러나 Mail에서 DECnet 대신 SMTP를 사용하도록 하려면 LOGIN.COM 파일에 다음 행을 추가하면 됩니다.

```
$ DEFINE MAIL$INTERNET_MODE SMTP
```

smith@pluto를 지정하면 Mail은 이 주소를 인터넷 주소로 해석하고 SMTP 프로토콜(예: SMTP%“smith@pluto.xyz.dec.com”)을 사용합니다.

노드 이름 지정

컴퓨터 시스템이 네트워크의 일부인 경우, 네트워크의 다른 모든 사용자에게 우편을 보낼 수 있습니다. 다른 노드에 있는 다른 사용자에게 우편을 보낼 경우에는 사용자의 노드 이름과 사용자 이름을 To: 프롬프트에 입력합니다. 사용자 이름에 특수 문자나 공백이 있으면 사용자 이름을 인용 부호(“ ”)로 묶어야 합니다. 다음 형식을 사용하십시오.

nodename::username

원격 노드에 대한 네트워크 연결을 사용할 수 없을 경우, Mail은 메시지를 표시합니다. 잠시 기다린 후 메시지 보내기를 다시 시도하십시오.

노드 이름 지정에 대한 추가 정보는 116페이지의 “네트워크 노드 이름”을 참조하십시오.

```
MAIL> SEND
```

```
To: CHEETA::HIGGINS
```

인터넷 우편 주소 사용

네트워크에서 전체 인터넷 우편 주소를 사용하여 사용자에게 우편을 보낼 수도 있습니다. 이러한 주소는 특히 조직 밖으로 우편을 보낼 경우에 일반적으로 사용됩니다.

username@company.com

To: 프롬프트에서 우편을 받을 사용자의 전체 인터넷 주소를 입력합니다. 이러한 주소를 대/소문자를 구분하는 경우는 거의 없습니다.

```
MAIL> SEND
```

```
To: J_SMITH@COMPANYNAME.COM, Kate.Muir@school.edu
```

논리 노드 이름 사용

논리 이름을 사용하여 사용자의 이름과 노드를 나타낼 수 있습니다. 그런 다음에 이 논리 이름을 사용하여 우편을 보낼 수 있습니다. **Mail**은 노드 이름이나 논리 이름의 액세스 제어 정보를 모두 무시합니다.

다음 보기에서는 **CHEETA::HIGGINS** 대신 **HENRY**를 사용합니다. 먼저 논리 이름 (**HENRY**)을 정의한 다음 이 이름을 사용자 이름과 노드 대신 사용합니다.

```
$ DEFINE HENRY CHEETA::HIGGINS
$ MAIL
MAIL> SEND
To: HENRY
```

여러 사용자에게 메시지 보내기

다음 절에서는 두 명 이상의 사용자에게 우편을 보내는 방법에 대해 설명합니다.

개별 이름 사용

To: 프롬프트에서 개별 사용자 이름을 사용하거나 우편 그룹을 사용하는 등 두 가지 방법 중 하나로 여러 사용자에게 동시에 우편을 보낼 수 있습니다. 사용자 이름을 사용하여 동일한 노드의 여러 사용자에게 동일한 메시지를 보내려면 **To:** 프롬프트에서 각 사용자 이름을 쉼표나 공백으로 구분하여 입력합니다.

다음 보기에서는 Thompson, Jones 및 Barney에게 메시지를 보냅니다.

```
MAIL> SEND
To:      THOMPSON, JONES, BARNEY
Subj:    Meeting on January 9
```

우편 그룹 만들기

우편 그룹은 사용자와 해당 노드 이름의 목록이 들어 있는 파일입니다. 우편 그룹을 만들려면 텍스트 편집기를 사용해야 합니다. **Mail** 유틸리티에서는 우편 그룹을 만들 수 없습니다.

열린 파일 할당량(사용자 계정에 대한 한계)은 우편을 한 번에 보낼 수 있는 서로 다른 노드 수와 우편 그룹을 중첩할 수 있는 깊이를 결정합니다. 이 할당량을 초과하면 **Mail**에서 오류 메시지를 표시합니다. 시스템 관리자에게 문의하여 할당량을 늘리거나 노드를 적게하여 한 번에 일괄 처리로 우편을 보내십시오.

기본적으로 시스템은 파일 유형이 **.DIS**인 우편 그룹 파일을 찾습니다. 우편 그룹이 포함된 파일의 유형이 다른 경우, **To:** 프롬프트에서 해당 파일 이름과 파일 유형을 지정합니다. 우편 그룹이 있는 디렉토리와 우편 그룹이 포함된 파일이 있는 디렉토리가 다른 경우 **Mail**을 호출하려면 **To:** 프롬프트에서 우편 그룹의 전체 디렉토리 이름을 입력합니다.

우편 그룹을 만들려면 다음 절차를 수행하십시오.

표 7-4

단계	작업
1	텍스트 편집기를 사용하여 파일 유형이 .DIS인 우편 그룹 파일을 만듭니다.
2	파일에서 각 행마다 하나의 사용자 이름을 입력합니다.
3	다른 우편 그룹의 이름을 파일에 포함하려면(즉, 목록을 “중첩”시키려면) 골뱅이 기호 (@) 다음에 우편 그룹의 이름을 지정합니다.
4	파일에 주석을 포함시키려면 주석 앞에 감탄 부호(!)를 입력합니다.

다음 보기는 우편 그룹 파일을 보여 줍니다.

```
! ALLBUDGET.DIS
!
! Budget Committee Members
@BUDGET          ! listed in BUDGET.DIS.
! Staff
  Thompson
  BRUTUS::JONES
  PORTIA::BARNEY
```

BUDGET.DIS 파일이 만들려는 새 우편 그룹 파일(ALLBUDGET.DIS)과 동일한 디렉토리에 있지 않은 경우, BUDGET.DIS에 대한 파일 지정 방법을 새 배포 파일에 포함시킵니다. ALLBUDGET.DIS를 만드는 위치에 따라 BUDGET.DIS가 있는 장치와 디렉토리를 지정해야 할 수도 있습니다. 파일 지정 방법에 대한 자세한 내용은 3장을 참조하십시오.

우편 그룹에 메시지 보내기

우편 그룹을 사용하여 여러 사용자에게 우편을 보내려면 다음 절차를 수행하십시오.

표 7-5

단계	작업
1	Mail을 호출합니다.
2	MAIL> 프롬프트에서 SEND를 입력하고 Enter 키를 누릅니다.
3	To: 프롬프트에서 골뱅이 기호(@)와 우편 그룹의 파일 이름을 입력하고 Enter 키를 누릅니다.
4	Subj: 프롬프트에서 메시지 제목을 입력하고 Enter 키를 누릅니다.
5	텍스트 프롬프트에서 메시지의 텍스트를 입력합니다.

다음 보기에서는 메시지를 우편 그룹 ALLBUDGET.DIS로 보냅니다.

```
MAIL> SEND
To: @ALLBUDGET
Subj: Tomorrow's Meeting
Enter your message below. Press CTRL/Z when complete, or CTRL/C to quit:
The meeting about the Hubbub Cola account is tomorrow at 2:00.
```

--Jeff

DCL 수준에서 우편 그룹으로 파일을 보낼 수도 있습니다. 파일 유형 .DIS를 생략할 경우, 파일을 우편 그룹으로 식별하기 위해 (@)와 파일 이름을 인용 부호(" ")로 묶습니다. 제목을 포함시키려면 MAIL 명령과 함께 /SUBJECT 한정자를 사용합니다.

다음 보기에서는 파일 MEETING.TXT를 사용자 THOMAS와 FRIENDS.DIS 우편 그룹으로 보냅니다.

```
$ MAIL/SUBJECT="update" MEETING THOMAS, "@FRIENDS.DIS"
```

다음 보기에서는 파일 NOTICE.TXT를 우편 그룹 WRITERS.DIS로 보냅니다. 여기에서 /SUBJECT 한정자는 포함되지 않으므로 제목이 표시되지 않은 채로 메시지가 보내집니다.

```
$ MAIL NOTICE "@WRITERS"
```

Mail에서의 파일 조작

Mail 내에서 또는 DCL 수준에서 다른 사용자에게 파일을 보낼 수 있습니다. 다음 절차에 따라 Mail 내에서 파일을 보냅니다.

표 7-6

단계	작업
1	MAIL> 프롬프트에서 SEND와 보낼 파일의 이름을 입력합니다.
2	To: 프롬프트에서 파일을 받을 사람의 사용자 이름을 입력합니다.
3	Subj: 프롬프트에서 파일의 제목을 입력합니다.
4	Enter 키를 눌러 파일을 보냅니다. 보내기 작업을 취소하려면 Ctrl/C 또는 Ctrl/Y를 누릅니다. Ctrl/C를 누르면 Mail 내에서 실행 중인 상태로 유지되지 만, Ctrl/Y를 누르면 DCL 수준으로 돌아옵니다.

다음 보기에서는 MEMO.TXT 파일을 사용자 EDGELL에게 보냅니다.

```
MAIL> SEND MEMO.TXT
To: EDGELL
Subj: Another memo
```

우편을 통해 파일을 보내는 경우 다음 제한 사항에 주의하십시오.

- COPY 명령을 사용하여 파일을 복사하면 운영 체제에서 데이터 무결성 확인을 수행합니다. 우편을 통해 파일을 보내는 경우 이러한 확인이 수행되지 않으며 실행 파일과 같은 외부 파일을 보낼 때 이 확인으로 인해 손상된 파일이 발생할 수 있습니다.
- 큰 파일을 보내는 경우 주의하십시오. 일부 시스템의 사용자들은 PostScript 파일과 같은 큰 파일을 받지 못할 수도 있습니다.

DDIF 파일 보내기

파일이 DIGITAL DDIF(디지털 문서 교환 형식) 지정 방법에 따라 구조화된 복합 문서인 경우, Mail은 OpenVMS AXP Version 1.0 또는 VAX/VMS Version 5.2-2 이상의 시스템에 대해서만 OpenVMS RMS 파일 태그와 DDIF 의미 체계를 유지합니다. DDIF 파일이 포함된 우편 메시지를 OpenVMS가 아닌 다른 운영 체제나 OpenVMS AXP Version 1.0 또는 VAX/VMS Version 5.2-2 이전의 OpenVMS 시스템에 보내려고 시도하는 경우, Mail에서 오류 메시지를 반환합니다.

DCL에서 파일 보내기

DCL 수준에서 파일을 보내는 경우, Mail이 호출되지만 사용자는 대화형 세션에 들어가지 않으며 MAIL> 프롬프트도 표시되지 않습니다. 파일을 보내면 자동으로 DCL 수준으로 돌아옵니다. 적절한 한정자와 함께 MAIL 명령을 입력한 후 Enter 키를 눌러 파일을 보내거나 Ctrl/C를 눌러 보내기 작업을 취소합니다.

다음 사항도 주의하십시오.

- 파일 지정 방법에서 와일드카드는 사용할 수 없습니다. 파일 유형을 생략하는 경우, 기본 파일 유형은 .TXT입니다.
- 파일 지정 방법으로 SYS\$INPUT을 지정하면 계속해서 DCL 수준으로 유지된 상태에서 메시지 텍스트에 대한 프롬프트가 표시됩니다. SYS\$INPUT 사용에 대한 자세한 내용은 11장을 참조하십시오.
- DCL 수준에서 파일을 보낼 경우 선택적인 /SUBJECT 한정자에 대한 인수에 공백이나 영문자 및 숫자 이외의 문자가 있으면 이 인수를 인용 부호로 묶어야 합니다.

다음 보기에서는 DCL 수준에서 노드 CHEETA의 사용자 EDGELL에게 MEMO.TXT 파일을 보냅니다.

```
$ MAIL/SUBJECT="Another memo" MEMO.TXT CHEETA::EDGELL
```

다음 보기에서는 지정된 파일 이름이 SYS\$INPUT이기 때문에 메시지 텍스트를 입력하라는 메시지가 표시됩니다.

```
$ MAIL SYS$INPUT:
To: ARMSTRONG
Enter your message below. Press CTRL/Z when complete, or CTRL/C to quit:
The text of the message is here.
Ctrl/Z
$
```

메시지에서 파일 만들기

메시지에서 텍스트 파일을 만들려면 해당 메시지를 읽고 있는 동안 **MAIL>** 프롬프트에서 **EXTRACT** 명령과 파일 이름을 입력합니다. Mail을 종료하면 다른 디렉토리를 지정하지 않은 경우 현재 디렉토리에 파일이 나열됩니다. 파일이 **DDIF** 파일이면 Mail에서 **OpenVMS RMS** 파일 태그와 **DDIF** 의미 체계가 유지됩니다(**VAX/VMS Version 5.2-2** 이상).

우편 헤더는 **From:**, **To:**, **Subj:** 행 등으로 이루어집니다. 헤더 정보가 포함되지 않은 파일을 만들려면 **EXTRACT** 명령에 **/NOHEADER** 한정자를 지정합니다. 메시지의 헤더가 두 개 이상(예: 전달된 메시지)이면 맨 위의 헤더만 삭제됩니다.

EXTRACT 명령에 **/APPEND** 한정자를 사용하여 메시지를 기존 파일의 끝에 복사합니다. 현재 폴더의 모든 파일을 기존 파일에 복사하려면 **/ALL** 한정자를 사용합니다.

다음 보기에서는 표시된 우편 메시지로부터 **DEC_MEETINGS.TXT**라는 파일이 만들어집니다.

```
#1                      01-DEC-1999  14:12:27          NEWMAIL
From:  STONE::FELLINI
To:    Thompson
Subj:  Dates for December sales meetings

Sales meetings in December will be held on the following dates:
      Wednesday Dec.  8, 1999
      Tuesday   Dec. 14, 1999
      Monday    Dec. 20, 1999
      Thursday  Dec. 30, 1999
MAIL> EXTRACT DEC_MEETINGS.TXT
%MAIL-I-CREATED, DISK:[THOMPSON]DEC_MEETINGS.TXT
```

다음 보기는 메시지 번호 3번의 텍스트가 포함된 **JANUARY_MEETINGS.TXT**라는 파일을 만드는 방법을 보여 줍니다.

```
MAIL> READ 3
.
.
.
MAIL> EXTRACT/NOHEADER JANUARY_MEETINGS.TXT
%MAIL-I-CREATED, DISK1:[JONES]JANUARY_MEETINGS.TXT;1 created
MAIL>
```

메시지에 파일 추가

우편 메시지의 끝에 작은 파일을 자동으로 추가하려면 **SET SIGNATURE_FILE** 명령을 사용합니다. 지정한 파일이 **ANSWER, FORWARD, MAIL, REPLY** 또는 **SEND** 명령을 사용하여 보내는 모든 우편 메시지에 자동으로(기본적으로) 추가됩니다. 서명 파일의 예로는 사용자의 회사 이름, 주소, 전화 번호 및 인터넷 주소가 포함된 명함 형식의 텍스트 파일을 들 수 있습니다.

파일을 메시지에 선택적으로 추가하거나 기본 서명 파일 설정을 무시하려면 **ANSWER, FORWARD, MAIL, REPLY** 또는 **SEND** 명령과 함께 **/SIGNATURE_FILE[=*file-name*]** 한정자를 사용합니다.

SHOW SIGNATURE_FILE 명령을 사용하여 기본 서명 파일을 지정했는지 여부를 표시합니다. **SHOW ALL** 명령도 서명 파일 정보를 표시합니다.

DCL 명령 **MAIL**과 함께 **/SIGNATURE_FILE[=*file-name*]** 한정자를 사용하여 **DCL** 수준에서 기본 서명 파일을 설정할 수도 있습니다.

서명 파일이 포함된 우편 메시지를 만드는 경우 작업 중에 임시 파일이 만들어지기 때문에 기본 메시지의 경우보다 임시 디스크 공간이 더 많이 필요합니다. 메시지를 보낸 후에는 이러한 임시 파일이 삭제됩니다.

또한 서명 파일 이름을 지정하는 경우 다음 사항을 주의하십시오.

- 파일 유형을 지정하지 않으면 기본적으로 **.SIG**가 사용됩니다.
- 디렉토리를 지정하지 않으면 **Mail** 유틸리티가 우편 디렉토리에서 서명 파일을 검색합니다.

다음 보기에서는 **FORWARD, MAIL, REPLY** 또는 **SEND** 명령을 사용하여 보내는 모든 우편 메시지에 자동으로 추가될 기본 파일로 **BUSINESS_CARD.SIG** 파일이 지명됩니다.

```
MAIL> SET SIGNATURE_FILE BUSINESS_CARD.SIG
```

다음 보기에서는 기본 서명 파일 대신 해당 특정 응답에 자동으로 추가될 파일로 **GREETINGS.SIG** 파일이 지명됩니다.

```
MAIL> REPLY/SIGNATURE_FILE=GREETINGS.SIG
```

메시지를 보내는 기타 방법

다음 절에서는 Mail 유틸리티를 사용하여 메시지를 보내는 기타 방법에 대해 설명합니다.

메시지에 응답

받은 메시지에 응답하려면 다음 절차를 수행하십시오.

표 7-7

단계	작업
1	MAIL> 프롬프트에서 REPLY를 입력하고 Enter 키를 누릅니다.
2	메시지를 입력하고 Ctrl/Z를 눌러 메시지를 보내거나 Ctrl/C를 눌러 종료합니다.

다음 보기에서는 응답을 STONE::THOMPSON에게 보냅니다. 응답 명령을 입력하면 Mail에서 To:와 Subj: 프롬프트에 프롬프트를 자동으로 표시합니다.

To: STONE::THOMPSON
Subj: RE: Budget Meeting
Enter your message below. Press CTRL/Z when complete. CTRL/C to quit:

중첩된 인용 부호가 있는 주소에 응답

대부분의 경우 Mail 명령 REPLY를 사용하여 중첩된 인용 부호가 있는 주소로부터 받은 우편에 응답할 수 있습니다. 시스템에 이 기능이 없는 경우에는 시스템 관리자에게 문의하십시오.

메시지 전달

우편 메시지를 다른 사용자에게 전달하려면 메시지를 읽은 후 MAIL> 프롬프트에서 FORWARD 명령을 입력합니다. Mail에서는 수신인 이름과 제목 행을 묻는 프롬프트를 표시합니다. 해당 정보를 입력한 후 Enter 키를 눌러 메시지를 보냅니다.

.DDIF 파일로 구성된 메시지를 전달하면 Mail은 .DDIF 의미 체계와 .DDIF 태그를 비롯한 전체 .DDIF 파일을 수신인에게 보냅니다.

다음 보기에서는 사용자 **STONE::JONES**에게 메시지가 전달됩니다.

```
MAIL> FORWARD
To: STONE::JONES
Subj: FYI - Status of proposed budget meeting
```

SET FORWARD 명령

SET FORWARD 명령을 사용하여 자신이 받은 모든 우편 메시지를 다른 **OpenVMS Cluster**나 다른 시스템에 있는 다른 계정으로 재지정할 수 있습니다. 이 명령을 사용하면 기본적으로 전달 주소가 만들어집니다. 정기적으로 확인하지 않을 계정에 대해서만 전달 주소를 설정합니다. 예를 들어, **OLD** 클러스터의 우편 계정에서 모든 우편을 **STAR** 클러스터의 우편 계정으로 전달하려는 경우, **OLD**로 로그인한 후 **Mail** 유틸리티를 입력하고 다음 명령을 입력합니다.

```
MAIL> SET FORWARD STAR::SMITH
```

OLD::SMITH로 보낸 모든 메시지가 **STAR** 노드의 우편 계정으로 자동으로 재지정됩니다. 전달 주소를 인터넷 우편 주소로 설정할 수도 있습니다.

```
MAIL> SET FORWARD SMITH@Company.com
```

이 경우, **OLD::SMITH**로 보낸 모든 우편이 **SMITH@Company.com**으로 보내집니다.

항상 테스트 메시지를 이전 계정으로 보내서 해당 계정이 정확히 전달되고 있는지 확인합니다. 우편 메시지가 무한정으로 전달되기만 하고 도착되지 않는 전달 루프가 만들어지는 것을 방지하려면 자신과 동일한 계정이나 다른 전달 계정으로 전달하도록 계정을 설정해서는 안 됩니다. **OLD::SMITH**를 **OLD::SMITH**로 전달하지 마십시오. 또한 **OLD::SMITH**를 **STAR::SMITH**로 전달한 다음 **STAR::SMITH**를 **OLD::SMITH**로 전달하지 마십시오.

계정이 전달되는 위치를 확인하려면 다음 명령을 입력합니다.

```
MAIL> SHOW FORWARD
Your mail is being forwarded to STAR::SMITH.
```

전달 주소를 제거하려면 다음 명령을 입력합니다.

```
MAIL> SET NOFORWARD
MAIL> SHOW FORWARD
You have not set a forwarding address.
```

전달 주소가 제거된 것을 확인하고 해당 계정에 테스트 메시지를 보냅니다.

주

이전 버전의 OpenVMS 운영 체제에서는 SET FORWARD 명령에 인용 부호를 포함해야 할 경우 한 쌍을 추가로 더 지정해야 했습니다. 이 명령에서 첫 번째 인용 부호 쌍을 자동으로 제거했기 때문입니다. OpenVMS Version 7.0부터는 SET FORWARD 명령에서 더 이상 인용 부호의 첫 쌍을 제거하지 않으므로 인용 부호를 추가로 지정할 필요가 없습니다.

메시지 구성

다음 절에서는 우편 메시지를 구성하는 방법에 대해 설명합니다.

폴더 만들기

우편 메시지를 구성하기 위해 사용자가 고유의 우편 파일과 폴더를 만들 수 있습니다. 우편 파일에는 폴더가 포함되며, 폴더에는 우편 메시지가 포함됩니다. 각 폴더와 파일에 포함될 수 있는 메시지의 수에는 제한이 없습니다.

일반적으로 우편 파일을 만들기보다는 폴더를 만들어서 메시지를 구성합니다. 기본 우편 폴더(NEWMAIL, MAIL, WASTEBASKET)와 마찬가지로 사용자가 만든 폴더는 일반적으로 MAIL.MAI 우편 파일에 저장됩니다. READ 또는 DIRECTORY 명령을 입력할 때마다 현재 폴더의 이름이 화면 오른쪽 위 모서리에 표시됩니다. 현재 폴더에 있는 메시지에 대해서만 작업을 할 수 있습니다.

우편 파일이 매우 클 경우(500블록 이상), 더 큰 폴더에 대해 별도의 우편 파일을 여러 개만 들어서 Mail의 성능을 향상시킬 수도 있습니다.

우편 하위 디렉토리 만들기

우편 메시지를 받으면 이 우편 메시지가 MAIL\$xxxxxxxxx.MAI라는 파일에 쓰여지고 최상위 디렉토리에 놓여 집니다. 여기서 x 문자는 임의의 긴 파일 지정 방법을 나타냅니다. 사용자가 우편 메시지를 처음 받으면 최상위 디렉토리에 기본 우편 파일인 MAIL.MAI가만 들어집니다.

.MAI 파일이 최상위 디렉토리에 표시되는 것을 방지하려면 Mail 명령 SET MAIL_DIRECTORY를 사용합니다. 이 명령은 우편 하위 디렉토리를 만들고 모든 .MAI 파일을 해당 하위 디렉토리로 이동합니다. .MAI 파일을 하위 디렉토리에서 최상위 디렉토리로 다시 이동하려면 SET NOMAIL_DIRECTORY 명령을 사용합니다.

모든 .MAI 파일이 포함된 하위 디렉토리의 이름을 표시하려면 MAIL> 프롬프트에서 SHOW MAIL_DIRECTORY를 입력합니다.

다음 보기에서는 사용자(FRED)가 .MAIL 디렉토리를 만듭니다.

```
MAIL> SET MAIL_DIRECTORY [.MAIL]
MAIL> SHOW MAIL_DIRECTORY
Your mail file directory is SY$LOGIN:[FRED.MAIL]
```

폴더로 메시지 이동

FILE 명령이나 MOVE 명령을 사용하여 현재 메시지를 다른 폴더에 둘 수 있습니다. 해당 폴더가 존재하지 않을 경우, 폴더를 만들 것인지 묻는 메시지가 표시됩니다. 지정한 폴더에서 메시지를 기입한 후 현재 폴더에서 메시지가 자동으로 삭제됩니다.

폴더 간의 메시지 복사

Mail 명령 COPY는 현재 메시지의 사본을 사용자가 지정한 폴더에 둡니다. 해당 폴더가 존재하지 않을 경우, 폴더를 만들 것인지 묻는 메시지가 표시됩니다.

다음 보기에서는 현재 폴더에서 MEETING이라는 단어가 포함된 모든 메시지가 SCHEDULE이라는 폴더로 복사됩니다. COPY 명령 실행이 완료되면 각 메시지의 사본이 두 개씩 만들어지는데, 하나는 현재 폴더에, 다른 하나는 SCHEDULE이라는 폴더에 있습니다.

```
MAIL> SEARCH MEETING
MAIL> COPY SCHEDULE
Folder SCHEDULE does not exist.
Do you want to create it (Y/N, default is N)?Y
%MAIL-I-NEWFOLDER, folder SCHEDULE created
```

다음 명령은 “meeting”이라는 단어가 포함된 다음 메시지를 선택하고 표시합니다.

```
MAIL> SEARCH

MAIL> COPY SCHEDULE
MAIL> SEARCH
%MAIL-E-NOTFOUND, no messages containing 'MEETING' found
```

폴더 선택

현재 우편 파일의 폴더 목록을 표시하려면 DIRECTORY/FOLDER 명령을 입력합니다. 새 폴더를 현재 폴더로 선택하려면 다음 명령 중 하나를 사용합니다.

- **SELECT** *foldername*

지정한 폴더를 현재 폴더로 선택합니다. READ와 DIRECTORY 같은 후속 Mail 명령에서는 선택한 폴더를 사용합니다. 각 명령에 대해 폴더 이름을 지정하지 않아도 됩니다.

- **SET FOLDER** *foldername*

이 명령은 SELECT 명령과 동일하게 작동합니다.

- **DIRECTORY** *foldername*

지정한 폴더를 현재 폴더로 선택하고 폴더의 메시지를 나열합니다.

- **READ** *foldername*

선택한 폴더를 현재 폴더로 선택하고 지정한 메시지(기본적으로 해당 폴더의 첫 번째 메시지)를 표시합니다.

다음 보기에서는 **MEMOS** 폴더를 선택합니다.

```
MAIL> DIRECTORY/FOLDER
Listing of folders in SYS$LOGIN:[FRED]MAIL.MAI;1
      Press CTRL/C to cancel listing
MAIL                                MEETING_MINUTES
MEMOS                              PROJECT_NOTES STAFF
MAIL> SELECT MEMOS
```

폴더 삭제

우편 폴더를 삭제하려면 해당 폴더에 있는 메시지를 모두 삭제하거나 다른 폴더로 이동합니다. 폴더에서 모든 메시지를 삭제하면 다른 폴더를 선택하는 즉시 이 빈 폴더도 자동으로 삭제됩니다.

다음 보기에서는 **MUSIC** 폴더의 메시지를 삭제합니다.

```
MAIL> SELECT MUSIC
%MIL-I-SELECTED, 2 messages selected
MAIL> DELETE/ALL
```

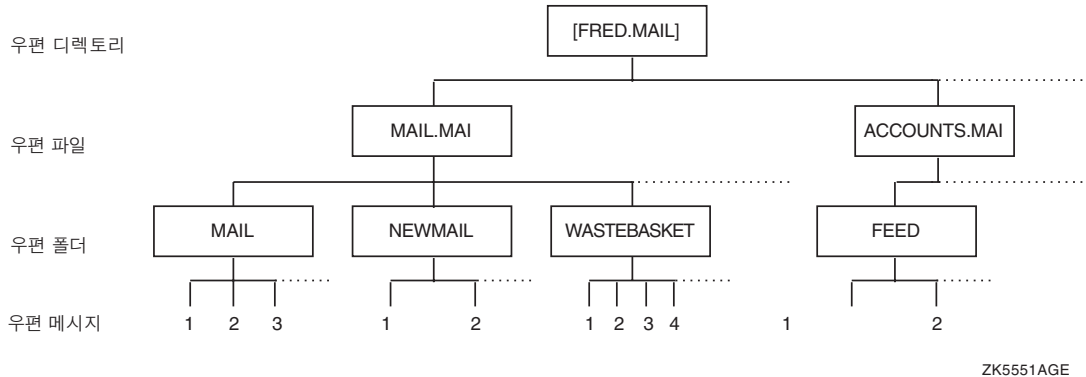
우편 파일 만들기 및 액세스

우편 메시지를 구성하기 위해 파일을 만들 수도 있습니다. 폴더를 만드는 데 사용하는 명령과 동일한 명령, 즉 **COPY**, **MOVE**, **FILE** 등을 사용하여 우편 파일을 만듭니다. 폴더 이름을 묻는 프롬프트가 표시된 후 파일 이름을 묻는 프롬프트도 표시됩니다. 새 파일 이름을 **File:** 프롬프트에서 입력하면 새 우편 파일이 만들어집니다.

기본 우편 파일이 아닌 다른 우편 파일 내에서 작업하려면 **Mail** 명령 **SET FILE**을 사용하여 대체 파일을 지정합니다. **Mail** 명령 **SHOW FILE**은 현재 우편 파일의 이름을 표시합니다. 우편 파일을 변경하면 현재 우편 파일의 **WASTEBASKET** 폴더가 비워지고 삭제되며 (**AUTO_PURGE**가 설정된 경우) 우편 파일이 닫힙니다.

그림 7-1은 일반 사용자가 우편을 구성하는 방법을 보여 줍니다.

그림 7-1 우편 구성



다음 보기에서는 현재 메시지가 ACCOUNTS 파일의 FEED라는 폴더로 이동됩니다.

MOVE 명령은 ACCOUNTS.MAI 우편 파일을 만들고, 현재 메시지를 FEED 폴더로 이동하고, 현재 폴더와 파일에서 그 메시지를 삭제합니다.

```
MAIL> MOVE
_Folder: FEED
_File: ACCOUNTS
```

다음 보기에서는 FEED 폴더(ACCOUNTS 파일이 있음)를 선택합니다.

```
MAIL> SET FILE ACCOUNTS
MAIL> SET FOLDER FEED
MAIL> SHOW FILE
Your current mail file is SYS$LOGIN:[FRED.MAI]ACCOUNTS.MAI;1.
```

우편 메시지 개수 수정

화면에 표시된 새(읽지 않은) 우편 메시지 개수가 실제 새 메시지 개수와 일치하지 않을 경우, 새 우편이 없을 때 READ/NEW 명령을 입력합니다. READ/NEW 명령을 입력하고 다음과 같은 시스템 메시지 중 하나가 나타날 때는 새 우편이 없다는 것을 알게 됩니다.

```
"%MAIL-W-NONEWMAIL, no new messages"
```

```
"%MAIL-E-NOMOREMSG, no more messages"
```

메시지 삭제

현재 폴더에서 우편 메시지를 삭제하려면 메시지를 읽고 있는 중에 **DELETE** 명령을 입력하거나, **DELETE** 명령 다음에 삭제할 메시지의 번호(또는 번호의 범위)를 입력합니다. 하이픈 (-) 또는 콜론(:)을 사용하여 삭제할 메시지의 범위를 정의할 수 있습니다.

다음 보기에서는 4, 5, 6, 11, 12, 14, 15, 16, 17번 메시지를 삭제합니다.

```
MAIL> DELETE 4-6,11,12,14:17
```

삭제된 메시지 복구

메시지를 삭제하면 이 메시지는 **WASTEBASKET**이라는 폴더로 이동됩니다. **Mail**을 종료하거나 다른 우편 파일을 지정하여 현재 우편 파일을 종료할 때까지는 삭제된 메시지가 **WASTEBASKET** 폴더에 수집됩니다. **SET AUTO_PURGE** 명령을 실행한 경우, 현재 우편 파일을 종료하면 **WASTEBASKET**이 비워지고 폴더 자체가 삭제됩니다. 대화형 **Mail** 세션 중에는 휴지통 폴더에서 메시지를 이동시켜 삭제된 메시지를 복구할 수 있습니다. **PURGE** 명령을 입력하여 **WASTEBASKET** 폴더를 비울 수도 있습니다.

다음 보기에서는 12번으로 식별된 우편 메시지를 삭제한 다음 **WASTEBASKET** 폴더에서 복구합니다.

```
MAIL> DELETE 12
```

```
MAIL> SELECT WASTEBASKET
%MAIL-I-SELECTED, 1 message selected
```

```
MAIL> DIRECTORY
# top
From Date Subject
    1 FABLES::WEST 11-DEC-1999 Meeting this week
MAIL> MOVE MAIL
```

우편 메시지 인쇄

우편 메시지를 인쇄하려면 **MAIL>** 프롬프트에서 **PRINT** 명령을 입력합니다. 기본적으로 **Mail**에서는 메시지를 **SYS\$PRINT** 대기열로 보냅니다. **Ctrl/Z**를 누르거나, **EXIT** 명령을 입력하거나, **PRINT/PRINT** 명령을 입력해야만 우편 파일이 인쇄 대기열로 보내집니다.

다른 대기열을 지정하려면 **PRINT** 명령 한정자 **/QUEUE**를 사용합니다.

SET QUEUE queue-name 명령을 실행하여 다른 대기열을 선택할 수도 있습니다. 이 대기열은 사용자가 **Mail**을 종료하더라도 다른 **SET QUEUE** 명령을 입력할 때까지 기본 인쇄 대기열로 유지됩니다.

다음 보기에서는 우편 메시지를 **AK34\$PRINT** 인쇄 대기열로 제출합니다.

```
MAIL> PRINT/QUEUE=AK34$PRINT
```

다음 보기에서는 기본 인쇄 대기열을 **SYS\$PRINT**에서 **AK34\$PRINT**로 변경합니다.

```
MAIL> SET QUEUE AK34$PRINT
```

우편 파일 보호

다음 절에서는 우편 파일을 보호하는 방법에 대해 설명합니다.

기본 보호

우편 파일(예: **MAIL.MAI**)은 사용자가 실수로 삭제하지도 못하도록 보호되며 다른 사용자가 읽을 수 없도록 보호됩니다. Mail에서 **.MAI** 파일에 제공하는 보호 코드는 **(S:RW,O:RW,G:W:)**입니다. 시스템(Mail 자체 포함)과 소유자(현재 사용자)는 파일을 읽고 파일에 쓸 수 있으며, **group**과 **world**에게는 모든 사용이 거부됩니다.

또한 Mail 유틸리티에는 우편 변경을 억제하기 위한 기본 파일 보호 기능이 있습니다. 그러나 변경을 하지 못하도록 Mail이 완벽하게 보호된 것은 아닙니다. 충분한 특권을 가진 사람은 누구든지 보호를 변경하여 우편 파일을 액세스할 수 있습니다.

보안 단위

우편 파일이 사용자 자신의 디렉토리 안에 있으므로, 10장에 설명된 중요 파일에 대한 파일 보호 방법을 적용하는 옵션을 사용할 수 있습니다. 또한 다음 사항을 주의하십시오.

- 우편 요청에 대한 응답은 스스로 판단하십시오. 노드가 로컬 **OpenVMS Cluster** 환경 외부에 있으면 소스 노드가 실수로 또는 의도적으로 잘못 식별될 가능성이 있습니다.
- 우편 메시지의 내용과 대상 선택에 있어 신중을 기하는 것이 가장 좋습니다.
- 암호를 공개하거나 계정 사용 방법에 대한 자세한 내용을 보내서는 안 됩니다. 일단 보낸 우편 메시지의 정보는 제어할 수 없게 됩니다.

Mail에서 텍스트 편집기 사용

다음 절에서는 Mail 환경에서 텍스트 편집기를 사용하는 방법에 대해 설명합니다.

EVE 사용

메시지를 보내기 전에 텍스트 편집기를 사용하여 메시지를 작성할 수 있습니다. 메시지를 작성하려면 **SEND** 명령과 함께 **/EDIT** 한정자를 지정합니다. **To:**와 **Subj:** 프롬프트에 해당 내용을 입력하면 Mail에서 텍스트 편집기를 호출합니다. 다른 편집기를 선택한 경우가 아니라면 DECTPU 기반 EVE 편집기를 호출합니다.

사용자가 텍스트를 입력함에 따라 **[End of file]** 표시기가 아래로 이동합니다. EVE 편집기에 대한 자세한 내용은 8장을 참조하십시오. 메시지를 보내려면 **Do** 키를 누르고 **EXIT** 명령을 입력합니다. 보내기 작업을 취소하려면 **Do** 키를 누르고 **QUIT** 명령을 입력합니다.

다음 보기에서는 EVE를 사용하여 우편 메시지를 만듭니다.

```
MAIL> SEND/EDIT
```

```
[End of file]
```

```
Buffer: MAIN | Write | Insert | Forward
```

주

.DDIF 우편 파일은 더 이상 .DDIF 파일로 사용할 수 없게 되므로 이 파일을 편집하지 마십시오. .DDIF 우편 파일을 편집할 경우에는 이 파일의 텍스트만 액세스할 수 있습니다.

/EDIT 한정자 키워드 사용

Mail을 호출할 때 **/EDIT** 한정자를 지정하면 이어지는 우편 세션 중 보내기, 응답, 전달 작업 등에 편집기를 사용할 수 있습니다. **/EDIT** 한정자와 함께 키워드를 사용하여 Mail에 대한 기본값을 설정할 수도 있습니다.

메시지에 응답할 때만 편집기를 호출하려면 **MAIL/EDIT** 명령과 함께 **REPLY** 키워드를 사용합니다. 편집기를 호출하고 응답할 메시지를 표시하려면 **=EXTRACT** 옵션과 함께 **REPLY** 키워드를 사용합니다. **/EDIT**에 키워드를 지정하지 않을 경우 기본적으로 **/EDIT=(SEND,REPLY)**가 사용됩니다.

메시지를 보내거나 메시지에 응답하려면 편집기를 종료합니다. **SEND** 또는 **REPLY** 명령을 취소하려면 **QUIT** 명령을 입력하여 편집기를 종료합니다.

보기

다음 보기에서는 보내거나 전달되는 모든 우편 메시지에 대해 편집기가 호출됩니다.

```
$ MAIL/EDIT=(SEND, FORWARD)
```

다음 보기에서는 응답을 받는 모든 메시지에 대해 편집기가 호출됩니다.

```
$ MAIL/EDIT=(REPLY)
```

다음 보기에서는 편집기가 호출되고, 메시지에 응답할 때마다 응답할 메시지가 텍스트로 포함됩니다.

```
$ MAIL/EDIT=(REPLY=EXTRACT)
```

편집기 선택

기본적으로 사용자가 **Mail** 명령 **SEND/EDIT**을 지정하는 경우 **Mail**에서 **DECTPU** 기반 **EVE** 편집기를 호출합니다. **Mail** 명령 **SET EDITOR**를 입력하면 **EVE** 대신 다른 편집기가 호출되도록 지정할 수 있습니다. 예를 들어, **EDT** 편집기를 선택하려면 **Mail** 명령 **SET EDITOR EDT**를 실행합니다. 다른 **SET EDITOR** 명령을 입력할 때까지는 시스템에서 로그아웃했다가 다시 로그인하더라도 **EDT** 편집기가 기본 **Mail** 편집기로 유지됩니다.

선택한 **Mail** 편집기의 이름을 표시하려면 **Mail** 명령 **SHOW EDITOR**를 입력합니다.

명령 파일을 사용하여 우편 편집

Mail을 입력하기 전에 논리 이름 **MAIL\$EDIT**를 명령 파일로 정의할 수 있습니다. 그런 다음에 편집기를 호출하는 **Mail** 명령을 실행할 경우, 편집 수행을 위해 이 명령 파일이 호출됩니다. 명령 파일에서는 맞춤법 검사기 등 다른 유틸리티를 호출할 수 있으며 명령 파일에서 수행될 수 있는 함수도 지정할 수 있습니다. 주석을 단 **MAILEDIT.COM** 명령 프로시저에 대한 보기를 보려면 부록 **B**를 참조하십시오. 또한 명령 파일에 대한 자세한 내용은 **13**장과 **14**장을 참조하십시오.

선택한 편집기 무시

선택한 편집기를 일시적으로 무시하려는 경우에는 **MAIL\$EDIT**를 원하는 편집기 이름이 추가된 “**CALLABLE_**” 문자열로 정의할 수 있습니다. 예를 들어, 호출 가능한 **EVE**가 아닌 호출 가능한 **EDT**를 사용하기 위해 다음 명령을 입력할 수 있습니다.

```
$ DEFINE MAIL$EDIT CALLABLE_EDT
```

정의된 **MAIL\$EDIT**과 함께 호출된 세션 중에 **SET EDITOR** 명령을 실행하면 선택한 영구적인 편집기 및 현재 편집기 설정이 모두 무시됩니다. **MAIL\$EDIT**에 의해 정의된 명령과 일을 다시 사용하려면 **Mail**을 종료했다가 다시 시작해야 합니다.

Mail 키패드 사용

키보드의 숫자 키패드를 사용하여 **Mail**의 명령을 실행할 수 있습니다. 대부분의 키패드 키는 두 가지 명령을 실행할 수 있습니다.

그림 7-2에서는 **Mail** 키패드를 보여 줍니다. 표시된 각 키의 위쪽 명령을 입력하려면 해당 키를 누르고, 표시된 아래쪽 명령을 입력하려면 **PF1** 키를 먼저 누른 다음 원하는 기능 키를 누릅니다.

그림 7-2

Mail 유틸리티 키패드

PF1 GOLD	PF2 HELP DIR/FOLDER	PF3 EXT/MAIL EXTRACT	PF4 ERASE SEL MAIL
7 SEND SEND/EDIT	8 REPLY REP/ED/EXT	9 FORWARD FORWD/EDIT	— READ/NEW SHOW NEW
4 CURRENT CURRENT/EDIT	5 FIRST FIRST/EDIT	6 LAST LAST/EDIT	, DIR/NEW DIR MAIL
1 BACK BACK/EDIT	2 PRINT PRINT/PR/NOT	3 DIR DIR/ST=99999	ENTER SELECT
0 NEXT NEXT/EDIT	. FILE DELETE		

ZK-1744-GE

Mail 명령 **SEND**를 실행하려면 **KP7**을 누릅니다. **Mail** 명령 **SEND/EDIT**를 실행하려면 **PF1** 키를 먼저 누른 다음 **KP7**을 누릅니다.

키패드 키 재지정

Mail 안에 있을 때 Mail 명령을 실행하도록 키패드 키를 재정의할 수 있습니다. 키를 재정의 하면 그 키에 대한 이전 정의가 대체됩니다.

Mail에서 키패드 키를 정의하는 것은 DCL 명령을 실행하도록 키패드 키를 정의하는 것과 비슷합니다.

다음 보기에서는 KP2 키를 Mail 명령 PRINT/PARAM=PAGE_ORIENT=LANDSCAPE로 정의합니다. KP2가 정의되면 이 키를 눌러서 PRINT/PARAM=PAGE_ORIENT=LANDSCAPE 명령을 표시할 수 있습니다.

```
MAIL> DEFINE/KEY KP2 "PRINT/PARAM=PAGE_ORIENT=LANDSCAPE"
```

추가 키 정의 할당

터미널에서 사용할 수 있는 키 정의의 수를 늘리려면 /STATE 한정자를 사용합니다. 동일한 키에 여러 개의 정의를 할당할 수 있습니다. 단 각 정의는 다른 상태에 관한 것이라야 합니다. 상태 이름은 임의의 영숫자 문자열일 수 있습니다. 상태를 지정하면 키를 한 번 눌러서 명령을 입력하고 한 번 더 눌러서 한정자를 입력할 수 있습니다.

다음 보기에서는 PF1(두 번 누름)을 DIRECTORY/FOLDER로 정의합니다.

```
MAIL> DEFINE/KEY PF1 "DIRECTORY"/SET_STATE=FOLDER /NOTERMINATE
MAIL> DEFINE/KEY PF1 "/FOLDER" /IF_STATE=FOLDER /TERMINATE
```

PF1을 두 번 눌러서 DIRECTORY/FOLDER 명령을 입력합니다. /TERMINATE 한정자를 입력하면 명령행이 끝나므로 Enter 키를 누르지 않아도 됩니다.

영구 키 정의 만들기

Mail을 종료하면 Mail 세션 중 정의한 모든 키패드 키가 손실됩니다. 한 Mail 세션에서 다른 Mail 세션으로 키패드 키 정의를 유지하려면 최상위 디렉토리에 키 정의가 포함된 파일(예: MAIL\$KEYDEF.INI)을 만듭니다. 예를 들어, 다음 MAIL\$KEYDEF.INI 파일에는 여섯 개의 키 정의가 들어 있습니다.

```
DEFINE/KEY PF1 "DIRECTORY " /NOTERMINATE /SET_STATE=folder
DEFINE/KEY PF1 "/FOLDER" /TERMINATE /IF_STATE=folder
DEFINE/KEY PF2 "SELECT " /NOTERMINATE /SET_STATE=mail
DEFINE/KEY PF2 "MAIL" /TERMINATE /IF_STATE=mail
DEFINE/KEY PERIOD "READ " /NOTERMINATE /SET_STATE=new
DEFINE/KEY PERIOD "/NEW" /TERMINATE /IF_STATE=new
```

Mail을 호출할 때마다 이러한 명령을 실행하려면 로그인 명령 파일(LOGIN.COM)에 다음 명령행을 입력합니다.

```
$ DEFINE MAIL$INIT SYS$LOGIN:MAIL$KEYDEF.INI
```

Mail 명령 요약

이 절에는 Mail 유틸리티의 모든 명령에 대한 요약이 들어 있습니다. 이러한 명령과 함께 사용하는 한정자에 대한 자세한 정보를 보려면 온라인 도움말을 참조하십시오.

MIME 유틸리티를 사용하여 MIME 인코딩 메시지를 읽고 작성하는 방법에 대한 자세한 내용은 243페이지의 “MIME 유틸리티”를 참조하십시오.

메시지 읽기

다음 명령을 사용하여 메시지를 읽습니다.

- **BACK**

마지막으로 실행한 명령이 **READ**일 경우, 현재 메시지 또는 마지막으로 읽은 메시지 이전의 메시지를 표시합니다. 마지막으로 실행한 명령이 **DIRECTORY**일 경우, **BACK** 명령은 디렉토리 목록의 이전 화면을 표시합니다.

- **CURRENT**

현재 읽고 있는 메시지의 시작 부분을 표시합니다.

- **DIRECTORY** [folder-name]

메시지 번호, 발송자 이름, 날짜, 제목 등과 함께 메시지 목록을 현재 우편 파일에 표시합니다.

- **ERASE**

터미널 화면을 지웁니다.

- **EXTRACT**

현재 메시지의 사본을 지정한 출력 파일에 넣습니다. 우편 메시지를 Mail 파일의 폴더에 복사하려면 **COPY**, **FILE** 또는 **MOVE** 명령을 사용합니다.

- **FIRST**

현재 폴더의 첫 번째 메시지를 표시합니다.

- **LAST**

현재 폴더의 마지막 메시지를 표시합니다.

- **NEXT**

다음 메시지로 건너 뛰고 그 메시지를 표시합니다.

- **READ** [folder-name] [message-number]

메시지를 표시합니다. **Enter** 키를 누르는 것은 매개 변수 없이 **READ** 명령을 입력하는 것과 동일합니다.

- **SEARCH** search-string

현재 선택한 폴더에서 지정한 텍스트 문자열이 첫 번째로 나오는 메시지를 검색합니다.

- **SHOW NEW_MAIL_COUNT**

읽지 않은 우편 메시지의 개수를 표시합니다.

메시지 교환

다음 명령을 사용하여 메시지를 교환합니다.

- **ANSWER** [filespec]

REPLY [filespec]

현재 읽고 있는 메시지나 마지막으로 읽은 메시지의 발송자에게 메시지를 보냅니다.

- **FORWARD**

현재 읽고 있거나 방금 읽은 메시지의 사본을 한 명 이상의 사용자에게 보냅니다.

- **MAIL** [filespec]

SEND [filespec]

메시지를 한 명 이상의 사용자에게 보냅니다.

메시지 제거

다음 명령을 사용하여 메시지를 제거합니다.

- **DELETE** [message-number]

현재 읽고 있는 메시지나 메시지 범위 또는 방금 읽은 메시지를 삭제하여 **WASTEBASKET** 폴더로 이동합니다.

- **PURGE**

WASTEBASKET 폴더의 모든 메시지를 삭제합니다. Mail을 종료하거나 새 우편 파일을 선택하기 위해 SET FILE 명령을 입력할 경우, 이전에 SET NOAUTO_PURGE 명령을 입력하지 않았으면 WASTEBASKET 폴더를 비우기 위한 암시적인 소거가 이루어집니다.

- **SET [NO]AUTO_PURGE**

EXIT 또는 SET FILE 명령을 입력할 경우 Mail에서 WASTEBASKET 폴더를 비울지 여부를 결정합니다. SET NOAUTO_PURGE 명령을 사용할 때는 PURGE 명령을 정기적으로 입력하여 WASTEBASKET 폴더의 메시지를 삭제해야 합니다.

- **SHOW AUTO_PURGE**

EXIT 또는 SET FILE 명령을 입력할 경우, WASTEBASKET 폴더의 메시지가 삭제(자동 소거)되는지 여부를 표시합니다.

메시지 인쇄

다음 명령을 사용하여 메시지를 인쇄합니다.

- **PRINT**

현재 읽고 있는 메시지의 사본을 인쇄 대기열에 추가합니다. PRINT 명령으로 만들어진 파일은 Mail을 종료할 때 인쇄 대기열로 릴리즈됩니다. /NOW 또는 /PRINT 한정자를 사용하지 않으면 여러 메시지가 하나의 인쇄 작업으로 연결됩니다.

- **SET [NO]FORM form-name**
SHOW FORM

Mail 내에서 이루어진 인쇄에 대한 기본 인쇄 양식을 설정합니다. SET NOFORM 명령은 기본 인쇄 양식을 지웁니다. SHOW FORM 명령은 기본 인쇄 양식을 표시합니다.

- **SET [NO]QUEUE queue-name**
SHOW QUEUE

Mail 내에서 PRINT 명령을 입력할 경우 사용할 기본 인쇄 대기열을 설정합니다. SET NOQUEUE는 이전에 정의한 인쇄 대기열을 지우고 대기열을 기본 인쇄 대기열인 SYS\$PRINT로 설정합니다. SHOW QUEUE 명령은 기본 인쇄 대기열을 표시합니다.

메시지 구성

다음 명령을 사용하여 메시지를 구성합니다.

- **COPY** folder-name [filename]

메시지를 현재 폴더에서 삭제하지 않은 채로 다른 폴더로 복사합니다. 지정한 폴더가 존재하지 않을 경우에는 새로 만들어집니다.

- **FILE** folder-name [filename]
MOVE folder-name [filename]

현재 메시지를 지정한 폴더로 이동하고 원래 폴더에서 이 메시지를 삭제합니다.

- **SELECT** [folder-name]
SET FOLDER [folder-name]

사용자가 영향을 미칠 수 있는 메시지 집합을 그룹으로 구성합니다. 이 메시지 집합을 한 폴더에서 다른 폴더로 복사하거나 이동할 수 있습니다. 메시지 집합을 읽고 삭제하거나 검색하고 추출할 수도 있습니다. 또한 한 폴더에서 다른 폴더로 이동하는 데 **SELECT**와 **SET FOLDER** 명령을 사용할 수 있습니다.

- **SET FILE** filename

다른 파일을 현재 우편 파일로 설정하거나 엽니다. 기본 우편 파일은 **MAIL.MAI**입니다. **COPY** 명령, **FILE** 명령 또는 **MOVE** 명령을 사용하여 다른 우편 파일(예: **JOKES.MAI** 또는 **HISTORY.MAI**)을 만들 경우, **SET FILE** 명령을 사용하여 Mail 파일을 열 수 있습니다.

- **SHOW FILE**

현재 열려 있는 우편 파일의 이름을 표시합니다.

- **SHOW FOLDER** [folder-name]

현재 폴더 이름을 표시합니다.

- **SET WASTEBASKET_NAME** folder-name

삭제할 메시지가 포함된 **WASTEBASKET** 폴더의 이름을 변경합니다. **PURGE** 명령을 입력하여 **WASTEBASKET** 폴더의 모든 메시지를 삭제할 수 있습니다.

AUTO_PURGE가 설정된 경우, **EXIT** 명령을 입력하면 **WASTEBASKET** 폴더의 메시지가 삭제됩니다. **AUTO_PURGE**가 설정된 경우, **QUIT** 명령을 입력하면 **WASTEBASKET** 폴더의 메시지가 삭제되는 것을 막을 수 있습니다.

- **SHOW WASTEBASKET_NAME**

WASTEBASKET 폴더의 이름을 표시합니다.

- **SHOW DELETED**

현재 우편 파일에서 삭제된 메시지 공간을 표시합니다.

메시지 표시

다음 명령은 메시지를 표시하는 데 사용합니다.

- **MARK** [message-number]

현재 메시지 또는 지정한 메시지를 표시된 것으로 설정합니다. 표시된 메시지는 디렉토리 목록의 왼쪽 열에 별표(*)와 함께 표시됩니다. 표시된 메시지를 선택하거나 구성하려면 **SELECT** 명령을 /**MARKED** 한정자와 함께 사용합니다.

- **UNMARK** [message-number]

현재 메시지 또는 지정한 메시지를 표시되지 않은 것으로 설정합니다. 디렉토리 목록의 왼쪽 열에 있는 별표(*)가 삭제됩니다.

Mail 환경 사용자 정의

다음 명령은 우편 환경을 사용자 정의하는 데 사용합니다.

- **DEFINE/KEY** key-name string

Mail 명령을 실행하도록 키를 정의합니다. 명령 이름을 입력하는 대신 이 키를 대신 눌러서 명령을 입력할 수 있습니다.

- **SHOW KEY** [key-name]

DEFINE/KEY 명령으로 만들어진 키 정의를 표시합니다.

- **EDIT** [filename]

선택한 편집기를 호출하여 메시지를 보내기 전에 편집할 수 있도록 합니다.

- **HELP** [topic]

Mail에 대한 정보를 표시합니다. 개별 명령이나 항목에 대한 정보를 보려면 **HELP** 다음에 해당 명령 또는 항목 이름을 입력합니다.

- **SET [NO]CC_PROMPT**
메시지를 보낼 때 CC:(참조) 프롬프트를 표시할지 여부를 결정하기 위한 기본값을 설정합니다.
- **SET COPY_SELF command[,command]**
SEND, REPLY 또는 FORWARD 명령이 보낼 메시지의 사본을 발송자에게 반환하는지 여부를 결정하기 위한 기본값을 설정합니다.
- **SHOW COPY_SELF**
사용자에게 메시지의 사본을 자동으로 보내는 명령(SEND, REPLY 또는 FORWARD)을 표시합니다.
- **SET [NO]SIGNATURE_FILE**
ANSWER, FORWARD, MAIL, REPLY 또는 SEND 명령을 사용할 때마다 우편 메시지의 끝에 서명 텍스트 파일이 자동으로 추가되도록 Mail 유틸리티를 설정합니다.
- **SHOW SIGNATURE_FILE**
기본 서명 파일을 지정했는지 여부를 보여 주고, 지정한 경우 해당 파일 이름을 보여주는 정보를 표시합니다. SHOW ALL 명령도 서명 파일 정보를 표시합니다.
- **SET EDITOR editor-name**
메시지를 편집할 때 사용할 텍스트 편집기를 선택합니다(예: Mail 명령 SEND/EDIT 사용). 시스템에서 있는 모든 호출 가능 편집기를 사용할 수 있습니다. 이 명령은 명령 프로시저 MAIL\$EDIT에서 설정한 모든 정의를 무시합니다.
- **SHOW EDITOR**
선택한 텍스트 편집기의 이름을 표시합니다.
- **SET [NO]FORWARD address**
우편의 전달 주소를 설정합니다.
- **SHOW FORWARD**
현재 전달 주소의 이름을 표시합니다.
- **SET [NO]MAIL_DIRECTORY[.subdirectory-name]**
모든 우편 파일(.MAI 파일 유형)이 SYS\$LOGIN 디렉토리에서 지정한 하위 디렉토리로 이동되도록 지정합니다.

- **SHOW MAIL_DIRECTORY**

모든 .MAI 파일이 포함된 장치 및 디렉토리의 이름을 표시합니다.

- **SET [NO]PERSONAL_NAME**“text-string”

사용자가 보내는 우편 메시지의 **From:** 필드 끝에 텍스트 문자열을 추가합니다. 사용자의 전체 이름이나 다른 정보로 이 필드를 채울 수 있습니다. 사용자의 개인 이름은 문자로 시작되어야 하며 공백 두 개를 연속으로 포함할 수 없습니다.

- **SHOW PERSONAL_NAME**

SET PERSONAL_NAME 명령으로 설정된 텍스트 문자열을 표시합니다.

- **SHOW ALL**

현재 Mail 설정에 대한 자세한 정보를 표시합니다.

종료 또는 제어 전송

다음 명령은 Mail을 종료하거나 제어를 전송하는 데 사용합니다.

- **ATTACH** [process-name]

터미널의 제어를 현재 프로세스에서 해당 작업의 다른 프로세스로 전환할 수 있도록 합니다. 예를 들어, 파일을 편집하는 동안 SPAWN 명령을 사용하여 새 우편 메시지를 읽는 하위 프로세스(Mail)로 이동할 수 있습니다. 그런 다음에 ATTACH 명령을 입력하여 편집 세션으로 다시 이동할 수 있습니다.

- **EXIT**

Mail을 종료합니다. EXIT 명령을 입력하면 SET NOAUTO_PURGE 명령을 실행하지 않은 경우 WASTEBASKET 폴더의 메시지가 모두 삭제됩니다. Ctrl/Z를 눌러서 Mail을 종료할 수도 있습니다.

- **QUIT**

WASTEBASKET 폴더를 비우지 않은 채로 Mail을 종료합니다. EXIT 명령을 입력하거나 Ctrl/Z를 누르는 않으면 삭제된 메시지는 완전히 없어지지 않습니다. QUIT 명령은 Ctrl/Y와 동일한 기능을 수행합니다.

- **SPAWN** [command]

현재 프로세스의 하위 프로세스를 만듭니다. SPAWN 명령을 사용하여 Mail을 일시적으로 끝냈다가 다른 기능(디렉토리 목록 표시 또는 파일 인쇄 등)을 수행한 다음 Mail로 돌아올 수 있습니다.

우편 파일 압축

다음 명령은 우편 파일을 압축하는 데 사용합니다.

- **COMPRESS** [filespec]

인덱스 우편 파일을 더 작게 만듭니다. 파일 이름을 지정하지 않으면 Mail에서 현재 열려 있는 우편 파일을 압축합니다. 열려 있는 우편 파일이 없으면 Mail에서 기본 우편 파일(MAIL.MAI)을 압축합니다.

시스템 관리 명령

다음 명령은 시스템 관리에 사용합니다.

- **REMOVE** user name

시스템의 우편 프로파일인 SYS\$SYSTEM:VMSMAIL_PROFILE.DATA 데이터 파일에서 사용자 레코드를 제거합니다. SYSPRV 특권이 필요합니다.

- **SHOW FORWARD**

사용자의 현재 전달 주소 이름을 표시합니다.

- **SHOW PERSONAL_NAME**

사용자가 SET PERSONAL_NAME 명령으로 설정한 텍스트 문자열을 표시합니다.

MIME 유틸리티

MIME(Multipurpose Internet Mail Extension)은 비텍스트 파일을 우편 메시지에 첨부하는 데 사용하는 표준입니다. MIME 유틸리티를 사용하면 MIME 인코딩 우편 메시지를 작성하고 읽을 수 있습니다. MIME에서는 그래픽이나 사운드 파일 같은 비텍스트 파일의 텍스트를 읽을 수 없더라도 이 파일을 인코딩하여 일반 텍스트로 보냅니다. MIME 유틸리티는 MIME 파일을 원래 형식으로 디코딩하여 사용자가 MIME 인코딩 파일을 만들 수 있도록 합니다. OpenVMS Mail 유틸리티를 사용하여 이 파일을 우편 메시지로 보낼 수 있습니다.

MIME 유틸리티 호출

시스템 관리자가 MIME에 대한 외부 명령을 이미 설정했을 수도 있습니다. 그렇지 않은 경우, LOGIN.COM에 다음 행을 추가하여 직접 설정할 수 있습니다.

```
$ MIME ::= $SYS$SYSTEM:MIME.EXE
```

MIME은 MIME 인코딩 텍스트 파일만 엽니다. Mail을 먼저 사용하여 MIME 인코딩 메시지를 텍스트 파일로 추출해야 합니다. 지시 사항을 보려면 217페이지의 “메시지에서 파일 만들기”를 참조하십시오.

DCL 프롬프트에서 MIME 유틸리티를 호출하려면 다음을 입력합니다.

```
$ MIME file-name.TXT
```

파일 이름 한정자는 선택 사항입니다. 지정된 파일이 존재하는 경우 READ_ONLY로 열립니다.

- /READ_ONLY는 해당 파일이 디코딩을 시도하는 사용자가 받은 메시지의 내용임을 나타냅니다. 이 값이 기본값입니다.
- /DRAFT는 이전 세션에서 만들어지고 WRITE_ACCESS용으로 열린 메시지가 해당 파일에 포함되어 있음을 나타냅니다.

MIME 유틸리티는 To: 또는 From: 필드 같은 헤더 정보를 구성하지 않습니다. MIME 헤더와 메시지의 본문 텍스트만 만들어서 Mail에서 나중에 보낼 파일에 텍스트를 저장합니다. 열리도록 지정된 파일에 이러한 인식 가능 헤더나 RFC822 헤더가 들어 있으면 해당 파일이 열리고 기본값은 /READ_ONLY입니다.

지정한 파일에 인식 가능 헤더가 들어 있지 않거나 이 파일이 존재하지 않으면 OPEN FILE ERROR 메시지가 발생합니다.

두 개의 파일 `MIME$MAILCAP.DAT`와 `MIME$FILETYPES.DAT`를 만들어서 MIME 인코딩 메시지 표시에 대한 시스템 기본값을 설정할 수 있습니다.

`MIME$MAILCAP.DAT`에는 MIME 인코딩 파일에 대해 로컬로 인식되는 각 콘텐츠 형식을 정의하는 어플리케이션이 들어 있습니다. `MIME$FILETYPES.DAT`는 각 콘텐츠 형식을 파일 확장자에 연결합니다. 사용자는 `SYS$LOGIN`에서 이러한 파일을 만들어 기본값을 무시할 수 있습니다.

MIME 유틸리티 초기화

사용자가 MIME 유틸리티를 시작하면 초기화 프로세스에서 다음 단계를 수행하십시오.

1. 사용자의 VMSmail 프로파일에서 MIME 유틸리티는 MIME 유틸리티와 함께 사용할 기본 편집기와 사용자의 우편 디렉토리를 찾습니다.
2. MIME 유틸리티는 `MIME$MAILCAP.DAT`와 `MIME$FILETYPES.DAT` 파일을 읽습니다.
3. MIME 유틸리티는 다음과 같은 내부 기본값 목록을 참조합니다.

- 콘텐츠 형식

MIME 유틸리티는 수신 메시지를 표시하기 전에 콘텐츠 형식의 목록을 참조합니다. 이 목록에는 MIME 유틸리티가 인식하는 콘텐츠 형식과 각 콘텐츠 형식을 원래 형식으로 디코딩하는 데 필요한 정보가 들어 있습니다.

다음은 RFC 1524의 MAILCAP 항목에 대한 보기입니다.

```
image/*; xview %s
```

`MIME$MAILCAP.DAT` 파일을 만들어 MIME이 인식하는 목록에 콘텐츠 형식을 추가할 수 있습니다. 보기 7-1에는 `MIME$MAILCAP.DAT` 파일의 보기가 있습니다.

- 파일 확장자

발송 메시지를 작성하는 동안 MIME 유틸리티는 파일 확장자 목록을 참조합니다. 이 목록에는 OpenVMS 파일 확장자와 각 확장자와 연결된 콘텐츠 형식이 들어 있습니다. MIME 유틸리티에서는 작성하는 MIME 형식의 메시지 본문에 이러한 확장자가 포함되어야 합니다.

파일 확장자 목록의 각 행은 다음 항목들로 구성됩니다.

확장자, 콘텐츠 형식/하위 형식, (선택 사항) Content-Transfer-Encoding 문 작업

다음은 파일 확장자 목록의 행에 대한 보기입니다.

```
doc, application/ms-word, base64
```

표 7-8에 설명된 MIME\$FILETYPES.DAT 파일을 만들어서 MIME 유틸리티가 인식하는 목록에 일치하는 콘텐츠 형식과 파일 확장자를 추가할 수 있습니다.

선택적인 MIME 유틸리티 파일 만들기

표 7-8에서는 시스템의 MIME 유틸리티를 사용자 정의하기 위해 만들려고 하는 파일을 나열하고 설명합니다.

표 7-8

MIME 유틸리티 옵션 파일

파일	용도
MIME\$MAILCAP.DAT	수신 메시지 표시 및 구분 분석
MIME\$FILETYPES.DAT	첨부된 출력 파일에 콘텐츠 형식 할당

이러한 파일은 SYS\$LOGIN 디렉토리에 둡니다.

MIME\$MAILCAP.DAT 파일 처리

MIME\$MAILCAP 파일의 형식은 1993년 9월 N. Borenstein의 RFC 1524, *A User Agent Configuration Mechanism for Multimedia Mail Format Information*에서 시작되었습니다. MIME 유틸리티에서는 이 파일의 지시 사항에 따라 메시지와 첨부 파일을 해석하고 표시합니다. MIME 사용자 에이전트는 이러한 지시 사항에 따라서 외부 프로그램을 호출하여 MIME 메시지에서 찾은 콘텐츠 형식을 표시합니다.

특정 콘텐츠 형식에 대한 FDL(파일 설명자 언어)을 지정하여 시스템의 메시지 부분들을 추출하도록 MIME\$MAILCAP.DAT 파일을 사용자 정의할 수 있습니다.

보기 7-1에는 MIME\$MAILCAP.DAT 파일의 보기가 있습니다.

주

프로그램 이름에 대한 참조는 논리 이름이거나 유효한 파일 지정 방법이어야 합니다.

보기 7-1

MIME\$MAILCAP.DAT 파일

```
#
# MIME$MAILCAP.DAT
#
# Local customizations of content types and processing options
#
# Use xv.exe to display images
image/*; xv %s
#
# Use Netscape for HTML attachments
text/HTML; netscape %s
#
```

MIME\$FILETYPES.DAT 파일 처리

선택적인 MIME\$FILE_TYPES.DAT 파일에는 OpenVMS 파일 확장자와 각 확장자와 연결된 MIME 콘텐츠 형식의 목록이 들어 있습니다. ADD 명령 처리에서는 FILETYPE 구조를 사용하여 작성한 메시지에 첨부할 OpenVMS 파일의 콘텐츠 형식을 지명합니다.

파일 형식의 구문은 “#” 문자가 주석을 나타내는 MIME\$MAILCAP.DAT 파일의 구문과 비슷합니다. 파일의 각 행에는 단일 파일 확장자(앞에 '.'이 없음) 다음에 해당 확장자를 사용하는 파일과 연결할 콘텐츠 형식 및 하위 유형을 포함합니다.

선택적으로 이 행에는 메시지에서 전송할 파일의 내용을 인코딩하는 데 사용하는 Content-Transfer-Encoding 문자열(7비트, 8비트, Base64 또는 Quoted-printable)이 포함될 수 있습니다. 7비트, 8비트, Base64 또는 Quoted-printable은 유일하게 허용되는 표준 MIME 인코딩입니다. 인코딩이 지정되지 않은 경우 MIME 유틸리티에서는 7비트를 사용합니다.

MIME 유틸리티를 사용하여 MIME 인코딩 파일 추출

MIME 유틸리티를 사용하여 MIME 인코딩 파일을 추출하려면 디코딩할 파일을 먼저 엽니다. 두 가지 방법 중 하나로 파일을 열 수 있습니다. 즉, MIME 유틸리티를 호출하여 파일 이름을 지정하거나, MIME 유틸리티에서 파일을 열 수 있습니다. EXTRACT는 지정한 첨부 파일을 원시 파일 형식이나 /FDL 한정자로 지정된 다른 형식의 파일로 추출합니다.

다음은 메시지 파일을 열고, 메시지를 읽을 수 있는 텍스트로 표시하고, 메시지 속성을 나열하는 데 사용하는 일반적인 MIME 유틸리티 명령입니다.

```
MIME> OPEN file-name
MIME> READ
MIME> LIST
```

첨부 파일을 추출하려면 다음 명령을 입력합니다.

```
MIME> EXTRACT /ATTACHMENT=n destination-file-name
```

추출할 첨부 파일의 개수를 지정하는 **/ATTACHMENT=*n*** 한정자를 추가하여 단일 첨부 파일을 지정할 수 있습니다. 지정한 첨부 파일을 출력 파일로 변환하는 데 사용할 **FDL**(파일 설명자 언어) 정의 파일을 지정하는 **/FDL=*filename***을 사용할 수도 있습니다. 개별 첨부 파일의 번호는 **LIST** 명령과 함께 표시됩니다.

MIME 유틸리티에서 사용하는 명령의 전체 목록을 보려면 248페이지의 “**MIME** 유틸리티 명령”을 참조하십시오.

MIME 유틸리티를 사용하여 파일 인코딩

첨부 파일로 보낼 파일을 인코딩하려면 **MIME** 유틸리티를 호출하고 **NEW** 명령을 지정하여 새 파일을 먼저 만들어야 합니다. 파일 이름을 지정하지 않으면 **NEW**에서 파일 이름을 묻는 프롬프트를 표시합니다.

```
$ MIME NEW new-file-name
```

또는 **MIME** 유틸리티에서 **OPEN** 명령을 사용하여 임시 메시지 파일을 열 수 있습니다.

```
MIME> OPEN/DRAFT file-name
```

이전 세션에서 만든 파일을 열려면 명령에 **/DRAFT** 한정자를 지정합니다.

파일에 첨부 파일을 추가하려면 다음 명령을 입력합니다.

```
MIME> ADD file-name
```

이 명령에 대한 선택적인 한정자의 전체 목록을 보려면 248페이지의 “**MIME** 유틸리티 명령”을 참조하십시오.

현재 정보를 파일에 쓰려면 **SAVE** 명령을 사용합니다. 한 번 저장되면 OpenVMS Mail 유틸리티에서 **MIME** 인코딩 파일을 파일로 보낼 수 있습니다.

MIME 유틸리티를 종료하려면 **QUIT** 또는 **EXIT** 명령을 입력합니다.

MIME 유틸리티에서 사용하는 명령의 전체 목록을 보려면 248페이지의 “**MIME** 유틸리티 명령”을 참조하십시오.

MIME 유틸리티 명령

다음 목록에는 MIME 유틸리티에서 사용할 수 있는 명령, 매개 변수 및 한정자에 대한 설명이 있습니다. 각각의 설명 다음에 보기가 나옵니다.

ADD — 편집할 메시지에 새 본문 부분 또는 첨부 파일을 추가합니다. ADD 명령에는 첨부할 파일의 이름이 매개 변수로 필요합니다. 선택적 한정자는 다음과 같습니다.

- **/BINARY** — 콘텐츠 형식을 “application/octet-stream”으로 설정하고 content-transfer-encoding을 “base64”로 설정합니다. 임의의 이진 데이터 스트림을 나타내는 데 이 형식을 사용할 수 있습니다.
- **/CONTENT_TYPE=type** — 기본 콘텐츠 형식을 지정한 문자열(예: “IMAGE/JPEG”)로 대체합니다.
- **/ENCODING_TYPE={7Bit|8Bit|Base64|Quoted-Printable}** — 기본 인코딩을 지정한 인코딩 유형으로 대체합니다.
- **/MESSAGE** — 첨부 파일은 메시지 파일(표준 RFC822)입니다.
- **/TEXT** — 첨부 파일은 콘텐츠 형식 텍스트입니다.

MIME> ADD file-name/TEXT

CLOSE — 현재 메시지 파일을 닫습니다. 최신 변경 내용을 저장하지 않은 경우, 닫기 전에 MIME 유틸리티에서 저장할 것인지 묻는 프롬프트를 표시합니다. 파일이 **/READ_ONLY** 일 경우 그 파일은 변경되지 않은 상태로 유지됩니다.

MIME> CLOSE

EDIT — 지정한 첨부 파일에 대한 사용자의 기본 텍스트 편집기를 호출합니다.

MIME> EDIT attachment-number

EXIT — 진행 중인 모든 작업을 저장하고 MIME 편집기를 종료합니다.

MIME> EXIT

EXTRACT — 지정한 첨부 파일을 해당 원시 파일 형식의 파일로 추출합니다.

- **/ATTACHMENT=n** — 출력할 첨부 파일의 개수를 지정합니다.
- **/FDL=filename** — 지정한 첨부 파일을 출력 파일로 변환하는 데 사용할 FDL(파일 설명자 언어) 정의 파일을 지정합니다.

MIME> EXTRACT file-name/ATTACHMENT=n

HELP — MIME 유틸리티에 대한 도움말 파일을 표시합니다.

MIME> HELP

LIST — 첨부 파일 번호 등 본문 부분과 속성의 목록을 비롯한 현재 메시지에 대한 정보를 표시합니다.

MIME> LIST

NEW — 새 메시지를 만듭니다.

MIME> NEW file-name

OPEN — 지정한 파일 이름으로 메시지를 엽니다. 사용할 수 있는 한정자는 다음과 같습니다.

- **/DRAFT** — 메시지 파일은 이전 세션에서 만들어진 임시 파일입니다.
- **/READ** — 메시지는 읽기 전용이므로 업데이트할 수 없습니다.

MIME> OPEN file-name/NEW

QUIT — 현재 메시지를 저장하지 않은 채로 현재 MIME 편집 세션을 중단합니다.

MIME> QUIT

READ — 현재 메시지를 읽을 수 있는 텍스트로 표시합니다. 가능한 경우 첨부 파일을 표시합니다.

MIME> READ

REMOVE — 현재 메시지에서 지정한 첨부 파일을 삭제합니다.

MIME> REMOVE 1

SHOW — 지정한 옵션에 따라 MIME 환경에 대한 정보를 표시합니다. 가능한 옵션은 **CONTENT_TYPE**, **FILE_TYPES** 및 **VERSION**입니다.

MIME> SHOW option

SAVE — 현재 메시지를 파일에 씁니다. 파일 이름을 지정하면 해당 이름이 사용됩니다.

MIME> SAVE file-name

오류 처리

오류 조건은 OpenVMS 신호 하위 시스템, 특히 `lib$signal()`와 `lib$stop()`를 사용하여 보고 됩니다. 오류 조건에 대한 심각도는 **Fatal**, **Error**, **Warning** 등 세 가지 수준이 있습니다. 이러한 수준은 조건에서 예상할 수 있는 결과를 나타냅니다. 심각도와 해당 결과에 대해서는 다음 목록에서 설명합니다.

- **Fatal (-F-)**은 프로그램을 즉시 종료시킵니다.
- **Error (-E-)**는 기존 메시지 컨텍스트를 유지하면서 현재 활성화된 명령을 종료시킵니다.
- **Warning (-W-)**은 MIME 편집 세션을 중단시키지 않으면서 현재 명령을 완료시킵니다. 그러나 이 명령이 모든 해당 작업을 성공적으로 완료했다고는 볼 수 없습니다. 오류가 있는지 결과를 확인하십시오.

8 EVE를 사용하여 텍스트 파일 편집

텍스트 **편집기**를 통해 텍스트 파일을 작성 및 수정할 수 있습니다. 텍스트 편집기를 사용하면 키보드에서 텍스트를 입력하고 텍스트 편집 명령을 사용하여 텍스트를 수정할 수 있습니다. 예를 들어, 보고서에 사용할 데이터를 입력한 다음 섹션 재배열, 정보 복제, 구 대체, 텍스트 형식화 등을 수행할 수 있습니다. 또한, 텍스트 편집기를 사용하면 프로그래밍 언어의 소스 파일을 작성 및 수정할 수 있습니다. 이 운영 체제에서는 여러 텍스트 편집기가 지원됩니다.

EVE(Extensible Versatile Editor)는 DECTPU(DEC Text Processing Utility)에 기반을 두는 범용 텍스트 편집기입니다. 이 장에서는 다음과 같은 내용이 다루어집니다.

- EVE 기능
- 도움말 보기
- 편집 세션 시작
- 명령 입력
- 편집 내용 저장 및 EVE 종료
- 커서 이동
- 텍스트 입력
- 텍스트 지우기 및 복원
- 텍스트 이동
- 텍스트 복사
- 상자 편집
- 삭제 보류 사용
- 텍스트 찾기 및 바꾸기
- 명령행 한정자 사용
- EVE를 실행하는 다른 방법
- 저널링 및 복구
- EVE 형식화 명령
- 버퍼 사용
- 하위 프로세스 만들기

EVE에 대한 자세한 내용은 EVE의 온라인 도움말과 *Extensible Versatile Editor Reference Manual*을 참조하십시오.

EDT에 대한 자세한 내용은 *OpenVMS EDT Reference Manual*을 참조하십시오.

규칙

이 장에서 **SHOW KEY** 또는 **HELP KEYS** 명령과 함께 표시되는 EVE 키 이름은 Ctrl 키, Shift 키 및 Alt 키 조합의 경우에는 슬래시를 사용하고 GOLD 키 시퀀스의 경우에는 공백이나 대시를 사용합니다. 따라서, 다른 키를 누를 때 누르고 있어야 하는 키(예: Ctrl)의 조합을 나타내는 경우에는 슬래시가 사용되고, 특정 키를 누른 다음 다른 키를 눌러야 하는 키 조합(예: GOLD-Help)을 나타내는 경우에는 공백이나 대시가 사용됩니다.

EVE 기능

DECTPU는 프로그래밍 가능한 고성능 텍스트 프로세서입니다. EVE 소프트웨어를 사용하면 비즈니스 레터, 기술 문서 및 프로그램 소스 파일과 같은 텍스트 파일을 작성 및 편집할 수 있습니다.

EVE는 OpenVMS 운영 체제의 기본 편집기입니다. 다른 기본 편집기를 정의하지 않은 경우, EDIT 명령을 입력하면 EVE가 실행됩니다.

EVE를 사용하면 다음 작업을 수행할 수 있습니다.

- 레터, 보고서, 프로그램 소스, 기타 문서 등의 텍스트 파일을 작성 및 편집합니다.
- 지우기, 잘라내기, 붙여넣기, 채우기, 찾기, 바꾸기, 페이지 매기기 등의 텍스트 서식 지정 작업을 수행합니다.
- 여러 개의 버퍼와 창을 사용하여 동일한 편집 세션에 있는 다른 파일들을 확인 및 편집합니다.
- 여러 명령이나 키 입력을 단일 키로 묶기 위한 시퀀스를 배우거나 EDT 키패드 또는 WPS-PLUS 키패드를 설정하는 것을 비롯하여 편집 작업을 위한 키 정의를 수행합니다.
- 잘라내기/붙여넣기 또는 기타 편집 작업을 위해 상자나 선행 범위에서 텍스트를 선택합니다.
- 와일드카드를 사용하여 텍스트 패턴을 검색합니다.
- 편집기에서 DCL 명령(예: DIRECTORY)을 실행합니다.
- DECspell을 실행하여 선택 영역이나 전체 버퍼를 검사합니다.
- 하위 프로세스를 생성하거나 다른 프로세스에 연결합니다.
- DECTPU 프로시저를 컴파일 및 실행하여 EVE를 확장합니다.
- DECwindows 인터페이스에서 메뉴 항목을 추가 또는 삭제합니다.
- 컴파일된 프로시저, 메뉴 정의, 키 정의 및 이후 세션에 대한 기타 사용자 정의를 저장합니다.
- 시작할 때와 편집 세션 도중에 초기화 파일을 사용합니다.
- 시스템 오류로 인해 편집 세션이 중단되면 키 입력 또는 버퍼 변경 저널링을 사용하여 작업을 복구합니다.

- 내장된 DECTPU 프로시저를 비롯하여 EVE 명령, 키, 메뉴 항목, 기타 주제에 대한 포괄적인 온라인 도움말을 확인합니다.

EVE를 실행하고 명령을 입력하는 방법을 알게 되면 EVE 명령을 사용하여 파일을 작성 및 편집할 수 있습니다. 또한, 편집 키와 명령을 사용하여 커서를 이동하고, 버퍼 모드를 설정하며, 텍스트 입력, 지우기, 복원 및 이동과 같은 편집 작업을 수행할 수 있습니다.

도움말 보기

편집 세션 도중에 언제든지 온라인 도움말을 볼 수 있습니다. 다음 두 종류의 온라인 도움말을 EVE 편집기에서 사용할 수 있습니다.

- 터미널의 **Help** 키를 사용하여 액세스하는 키패드 도움말
- EVE 명령 프롬프트에서 **HELP** 명령을 사용하여 액세스하는 EVE 도움말

키패드 도움말 사용

키패드 도움말에 액세스하려면 다음 단계를 따릅니다.

1. **Help** 키를 누릅니다.

도움말 유틸리티가 키패드의 다이어그램을 표시합니다.

2. 화면의 지침에 따라 다음 사항에 대한 정보를 확인합니다.

- **EVE 명령**

EVE 명령에 대한 도움말을 보려면 명령 이름이나 물음표(?)를 입력하고 **Enter** 키를 누릅니다.

- **정의된 키**

정의된 키에 대한 도움말을 보려면 해당 키를 누르거나 **SHOW KEY** 명령을 사용합니다.

- **키 정의 나열**

모든 키 정의를 나열하려면 **keys**를 입력하고 **Enter** 키를 누르거나 **GOLD HELP**를 누릅니다. **GOLD** 키는 숫자 키패드에 있는 **NumLock** 키 또는 **PF1** 키입니다.

3. **Enter** 키를 눌러 도움말을 종료합니다.

EVE 도움말 사용

HELP 명령을 사용하여 EVE 도움말에 액세스하려면 다음 단계를 따릅니다.

1. Do 키를 누릅니다.

2. HELP 명령을 입력합니다.

Prev Screen 및 Next Screen 키를 사용하여 사용 가능한 도움말 항목의 목록을 스크롤합니다.

3. Enter 키를 눌러 도움말을 종료합니다.

특정 명령에 대한 정보를 보려면 **HELP**와 명령 이름을 차례로 입력하고 **Enter** 키를 누릅니다. 도움말 텍스트가 화면에 나타납니다. **HELP TPU** 명령을 입력하여 내장된 **DECTPU** 프로시저에 대한 도움말을 볼 수도 있습니다.

다음 보기는 **MOVE BY LINE** 명령에 대한 도움말 텍스트를 보여 줍니다.

MOVE BY LINE

Moves the cursor a line at a time in the current direction.

Keys:	EVE Default	VT100 Keypad

	F12	MINUS on keypad

Steps:

1. If necessary, set the direction to move in --- forward or reverse.
2. Use MOVE BY LINE (see key list above).

Usage notes:

- o In forward direction, moves to the end of the current line, or to the end of the next line, if any.
- o In reverse direction, moves to the start of the current line, or to the start of the next line, if any.

Related topics:

CHANGE DIRECTION	END OF LINE	LINE	START OF LINE

편집 세션 시작

EVE는 OpenVMS 운영 체제의 기본 편집기입니다. 사용자 자신이나 시스템 관리자가 기본 편집기를 다시 정의하지 않은 경우, 다음과 같이 **EDIT** 명령을 입력하면 **EVE** 편집기가 자동으로 시작됩니다.

```
$ EDIT
```

EVE가 기본 편집기가 아닌 시스템에서는 **EDIT/TPU** 명령을 사용하여 **EVE**를 시작합니다. 편집 세션이 시작되면 기존 파일의 이름이나 만들려는 새 파일의 이름을 지정할 수 있습니다. 파일 이름을 지정하지 않으면 **Main**이라는 기본 버퍼에 텍스트를 추가한 후 편집 세션을 끝내려고 할 때 파일 이름을 묻는 메시지가 표시됩니다. 버퍼 사용에 대한 자세한 내용은 314페이지의 “버퍼 사용”을 참조하십시오.

다음은 **EVE**를 실행하여 **NEWFILE.DAT**라는 새 파일을 만드는 보기입니다.

```
$ EDIT NEWFILE.DAT
```

```
[End of file] [1]
```

[2]

```
Buffer: NEWFILE.DAT | Write | Insert | Forward [3]
```

```
Command: [4]
```

```
Editing new file. Could not find: FABLES.TXT [5]
```

EVE 화면 표시를 검토할 경우 다음 사항에 주의합니다.

- [1] **End-of-file** 표시는 **EVE** 버퍼의 끝을 표시합니다. 이 표시는 화면에서만 볼 수 있고 파일에는 포함되지 않습니다. 버퍼에 텍스트를 추가하면 **End-of-file** 표시가 아래로 이동합니다. 터미널 화면의 길이에 따라, 많은 행의 텍스트가 포함된 버퍼의 시작 부분을 볼 때 이 표시가 보이지 않을 수도 있습니다.
- [2] 창은 버퍼를 표시하는 화면 영역입니다. **EVE** 버퍼는 편집 세션 동안에만 존재합니다. 편집 세션이 끝날 때 편집 내용을 저장하거나 취소할 수 있습니다.
- [3] 강조된 상태 표시줄은 **EVE** 창 아래쪽에 나타나고, 창에 표시된 버퍼에 대한 정보를 제공합니다. 상태 표시줄은 버퍼 이름, 편집 상태(쓰기 또는 읽기 전용), 현재 모드(삽입 또는 겹쳐쓰기) 및 현재 방향(정방향 또는 역방향)을 보여 줍니다.
- [4] 명령행을 사용하여 행 모드 명령을 입력합니다(260페이지의 “명령 입력” 참조). 명령행을 표시하려면 **Do** 키를 누릅니다.

- [5] EVE를 실행하고 명령행에서 파일 이름을 지정하면, 메시지 창은 강조된 상태 표시줄 밑에 정보 메시지를 표시합니다. 이 메시지는 해당 파일이 새 파일이거나 기존 파일에서 특정 개수의 행을 읽었다는 사실을 알려 줍니다. 편집 세션 도중에 다른 여러 메시지가 메시지 창에 표시됩니다.

명령 입력

다음과 같은 두 가지 방법으로 **EVE** 명령을 입력할 수 있습니다.

- 명령행 인터페이스에서 명령을 입력합니다.
- EDT 또는 WPS 키패드에서 정의된 키를 사용합니다.

명령 입력

명령을 입력하려면 다음 단계를 따릅니다.

1. Do 키를 누릅니다.

커서가 명령 창으로 이동하고 명령 입력 프롬프트가 표시됩니다.

2. 명령을 입력합니다. 명령의 처음 몇 개 문자를 사용하여 명령을 간략화할 수 있습니다. **EVE**는 대/소문자를 구분하지 않습니다. **FIND** 및 **REPLACE** 명령에 대한 문자열을 지정하는 경우를 제외하고는 명령행에서 대/소문자를 임의로 조합하여 사용할 수 있습니다.

3. Do 키나 Enter 키를 누릅니다.

EVE는 명령을 실행하거나 추가 정보를 나타내는 프롬프트를 실행합니다.

정의된 키 사용

정의된 키를 사용하여 **EVE** 명령을 입력할 수 있습니다. 정의된 키는 각각 하나의 편집 명령을 수행합니다. 고유한 키를 정의하여 **EVE** 기능을 수행할 수도 있습니다.

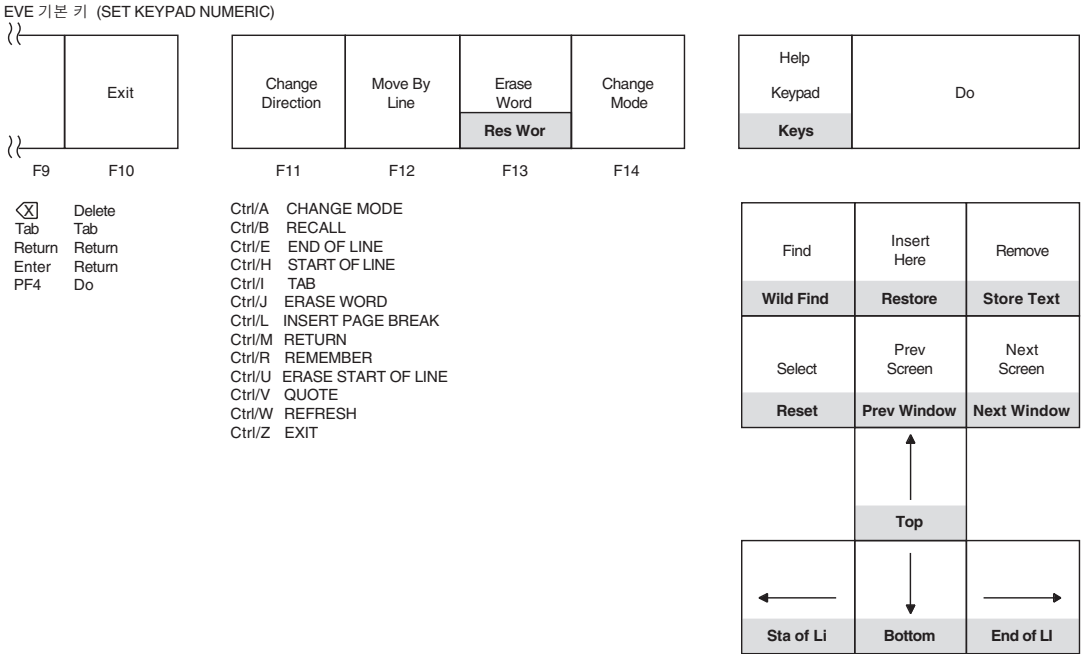
EVE는 몇 가지 키를 기본적으로 정의합니다. VT200, VT300 및 VT400 계열 터미널에서 미리 정의된 키는 다음과 같습니다.

- 미니 키패드(기본 키보드 키와 숫자 키패드 사이의 화살표 위에 있음)
- 특정 기능 키
- 특정 Ctrl 키 시퀀스

Ctrl 키 및 화살표 키와 Tab, Return 및 Delete 키는 세 가지 터미널에서 동일하게 정의됩니다.

그림 8-1은 VT200, VT300 및 VT400 계열 터미널에서 미리 정의된 키를 보여 줍니다.

그림 8-1 EVE 키 — VT200, VT300 및 VT400 계열 터미널



GOLD 키 기능은 회색 음영으로 보여 줍니다.

ZK-6300-GE

VT100 계열 터미널의 경우, EVE는 대부분의 숫자 키패드 키, 네 개의 화살표 키, 특정 Ctrl 키를 자동으로 정의합니다. 그림 8-2는 VT100 계열 터미널의 미리 정의된 키를 보여 줍니다.

그림 8-2 EVE 키 — VT100 계열 터미널

EVE VT100 키패드							
				Find	Help Keypad	Change Direction	Do
Top	Bottom	Start of Line	End of Line	Select	Remove	Insert Here	Move By Line
							Erase Word
							Change Mode
				Next Screen		Prev Screen	

Delete

Tab

Return

Backspace

Linefeed

Delete

Tab

Return

Start of Line

Erase Word

Ctrl/A

Ctrl/B

Ctrl/E

Ctrl/H

Ctrl/I

Ctrl/J

Ctrl/L

Ctrl/M

Ctrl/R

Ctrl/U

Ctrl/V

Ctrl/W

Ctrl/Z

CHANGE MODE

RECALL

END OF LINE

START OF LINE

TAB

ERASE WORD

INSERT PAGE BREAK

RETURN

REMEMBER

ERASE START OF LINE

QUOTE

REFRESH

EXIT

GOLD 키 기능은 회색 음영으로 보여 줍니다.
ZK-6301-GE

편집 내용 저장 및 EVE 종료

다음 방법 중 하나를 사용하여 편집 내용을 저장할 수 있습니다.

- **WRITE FILE 명령**
편집 세션을 종료하지 않고 파일을 저장합니다.
- **EXIT 명령**
편집 세션을 종료하고 변경 내용을 파일에 저장합니다.
- **QUIT 명령**
변경 내용을 파일에 저장하지 않고 편집 세션을 종료합니다.

WRITE FILE 명령 사용

EVE를 종료하지 않고 파일에 기록하는 방법으로 버퍼의 텍스트를 저장하려면 **WRITE FILE** 명령을 사용합니다. 버퍼와 연결된 파일이 없으면 다음과 같이 파일 이름을 묻는 메시지가 표시됩니다.

Type filename for buffer Main (press RETURN to not write it):

파일 이름을 입력하고 **Enter** 키를 눌러 버퍼를 파일에 기록합니다.

EXIT 명령 사용

편집된 텍스트를 저장하려면 **EXIT** 명령을 사용합니다. **F10** 키나 **Ctrl/Z**를 눌러 **EXIT** 명령을 입력할 수 있습니다.

현재 버퍼를 수정한 경우, **EVE**는 원래 버전과 동일한 파일 이름 및 형식을 사용하여 새로운 버전의 파일을 만듭니다(버전 번호는 1씩 증가). 예를 들어, **FUN.DAT;1**이라는 파일을 수정한 후 **EXIT** 명령을 사용하면 출력 파일의 이름은 **FUN.DAT;2**가 됩니다.

QUIT 명령 사용

편집 내용을 저장하지 않고 세션을 끝내려면 **QUIT** 명령을 입력합니다. **YES(Y)**를 입력하고 **Enter** 키를 누르면 편집 내용이 저장되지 않고 세션이 종료됩니다. 편집 내용을 저장하려는 경우에는 **N**을 입력하고 **Enter** 키를 누른 다음, **EXIT** 명령을 사용하여 버퍼를 종료합니다.

수정된 버퍼가 현재 버퍼가 아닌 경우, 다른 버퍼의 내용을 저장할 것인지 묻는 메시지가 표시됩니다. **Y**를 입력하면 기존 파일의 새로운 버전이 만들어집니다(버전 번호는 1씩 증가). 파일이 존재하지 않을 경우에는 파일 이름을 묻는 메시지가 나타납니다.

수정된 버퍼가 없으면 **EXIT**와 **QUIT**는 동일합니다. 예를 들어, 편집 작업 없이 **EVE**를 사용하여 파일을 검사한 경우에는 **Ctrl/Z**를 눌러 종료할 수 있습니다.

다음 보기는 **FUN.DAT**라는 버퍼를 수정했다고 가정하고 **QUIT** 명령을 입력한 결과를 보여줍니다.

```
Command: QUIT
```

```
Buffer modifications will not be saved, continue quitting (Y or N)?
```

커서 이동

EVE를 사용하여 파일을 편집할 때, 편집을 수행하려는 위치로 커서를 이동합니다. 텍스트에서 커서를 더 신속하고 효율적으로 이동할수록 편집 세션의 작업 시간을 더 많이 줄일 수 있습니다. 키보드나 명령을 사용하여 커서를 이동할 수 있습니다.

표 8-1은 커서를 이동하는 EVE 편집 키를 보여 줍니다. 나열된 GOLD 키 조합에 대한 자세한 내용은 온라인 도움말에서 GOLD 항목을 참조하십시오.

표 8-1

커서를 이동하는 EVE 편집 키

키 또는 키 시퀀스	기능
위쪽 화살표 키	MOVE UP과 동일합니다. 커서를 한 행 위로 이동합니다. VT100 계열 터미널의 경우, KP5도 MOVE UP으로 정의됩니다.
아래쪽 화살표 키	MOVE DOWN과 동일합니다. 커서를 한 행 아래로 이동합니다. VT100 계열 터미널의 경우, KP2도 MOVE DOWN으로 정의됩니다.
왼쪽 화살표 키	MOVE LEFT와 동일합니다. 커서를 문자 하나 또는 열 하나만큼 왼쪽으로 이동합니다. VT100 계열 터미널의 경우, KP1도 MOVE LEFT로 정의됩니다.
오른쪽 화살표 키	MOVE RIGHT와 동일합니다. 커서를 문자 하나 또는 열 하나만큼 오른쪽으로 이동합니다. VT100 계열 터미널의 경우, KP3도 MOVE RIGHT로 정의됩니다.
Ctrl/E 또는 GOLD 오른쪽 화살표 키	END OF LINE과 동일합니다. 커서를 현재 행의 끝 부분으로 이동합니다.
Ctrl/H 또는 GOLD 왼쪽 화살표 키	START OF LINE과 동일합니다. 커서를 현재 행의 시작 부분으로 이동합니다.
GOLD 위쪽 화살표 키	TOP과 동일합니다. 커서를 현재 버퍼의 맨 위로 이동합니다.
GOLD 아래쪽 화살표 키	BOTTOM과 동일합니다. 커서를 현재 버퍼의 맨 아래로 이동합니다.

표 8-1 커서를 이동하는 EVE 편집 키(계속)

키 또는 키 시퀀스	기능
GOLD Next Screen	NEXT WINDOW와 동일합니다. 둘 이상의 창을 사용하는 경우 화면의 다음 창에서 커서가 차지하던 마지막 위치로 커서를 이동합니다.
GOLD Prev Screen	PREVIOUS WINDOW와 동일합니다. 둘 이상의 창을 사용하는 경우 화면의 이전 창에서 커서가 차지하던 마지막 위치로 커서를 이동합니다.

표 8-2는 커서를 이동하는 EVE 명령을 보여 줍니다.

표 8-2 커서를 이동하는 EVE 명령

명령	기능
BOTTOM	커서를 현재 버퍼의 끝 부분으로 이동합니다. 기본적으로 EVE는 GOLD 아래쪽 화살표 키를 BOTTOM으로 정의합니다.
CHANGE DIRECTION	현재 버퍼의 방향을 변경합니다. 버퍼 방향은 상태 표시줄에 나타납니다.
END OF LINE	커서를 현재 행의 끝 부분으로 이동합니다. 기본적으로 EVE는 Ctrl/E 및 GOLD 오른쪽 화살표 키를 END OF LINE으로 정의합니다.
FORWARD	기본 설정입니다. 현재 버퍼의 방향을 정방향으로, 즉 오른쪽 아래로 설정합니다. 버퍼 방향은 상태 표시줄에 나타납니다.
GO TO	커서를 지정된 위치(이전에 MARK 명령으로 표시한 위치)로 이동합니다.
LINE	커서를 번호가 지정된 행의 시작 부분으로 이동합니다.
MARK	현재 위치에 보이지 않는 표식을 달고 지정된 이름을 표식과 연결합니다. 나중에 GO TO 명령을 사용하여 표시된 위치로 되돌아갈 수 있습니다.

표 8-2

커서를 이동하는 EVE 명령(계속)

명령	기능
MOVE BY LINE	정방향인 경우: 커서를 현재 행의 끝 부분으로 이동하고, 커서가 이미 행의 끝 부분에 있는 경우에는 다음 행의 끝 부분으로 이동합니다. 역방향인 경우: 커서를 현재 행의 시작 부분으로 이동하고, 커서가 이미 행의 시작 부분에 있는 경우에는 이전 행의 시작 부분으로 이동합니다. VT200, VT300 및 VT400 계열 터미널에서는 F12 키가 MOVE BY LINE으로 정의되고, VT100 계열 터미널에서는 키패드의 빼기(-) 키가 MOVE BY LINE으로 정의됩니다.
MOVE BY PAGE	현재 방향에 따라 커서를 다음 또는 이전 페이지 구분(form feed)으로 이동합니다. 현재 방향에 페이지 구분이 없는 경우, 버퍼의 맨 아래나 위로 커서가 이동합니다.
MOVE BY WORD	정방향인 경우: 커서를 다음 단어의 시작 부분으로 이동하거나, 커서가 행의 끝 부분에 있는 경우에는 다음 행의 시작 부분으로 이동합니다. 역방향인 경우: 커서를 이전 단어의 시작 부분으로 이동하거나, 커서가 행의 시작 부분에 있는 경우에는 이전 행의 끝 부분으로 이동합니다.
NEXT SCREEN	현재 창의 행 수에서 하나를 뺀 행 수만큼 현재 버퍼에서 앞으로 스크롤합니다. 예를 들어, 현재 창의 행 수가 12개이면 NEXT SCREEN 명령은 11행만큼 앞으로 스크롤합니다. VT200, VT300 및 VT400 계열 터미널에서는 E6 키(Next Screen)가 NEXT SCREEN으로 정의되고, VT100 계열 터미널에서는 키패드의 KP0 키가 NEXT SCREEN으로 정의됩니다.
NEXT WINDOW 또는 OTHER WINDOW	커서를 화면의 다음 창(있는 경우)으로 이동합니다. 커서는 다음 창에서 마지막으로 차지하던 위치에 표시됩니다. EVE는 GOLD Next Screen을 NEXT WINDOW로 정의합니다.

표 8-2

커서를 이동하는 EVE 명령(계속)

명령	기능
PREVIOUS SCREEN	현재 창의 행 수에서 하나를 뺀 행 수만큼 현재 버퍼에서 뒤로 스크롤합니다. 예를 들어, 현재 창의 행 수가 12개이면 PREVIOUS SCREEN 명령은 11행만큼 뒤로 스크롤합니다. VT200, VT300 및 VT400 계열 터미널에서는 E5 키(Prev Screen)가 PREVIOUS SCREEN으로 정의되고, VT100 계열 터미널에서는 키패드의 마침표(.) 키가 PREVIOUS SCREEN으로 정의됩니다.
PREVIOUS WINDOW	커서를 화면의 이전 창(있는 경우)으로 이동합니다. 커서는 이전 창에서 마지막으로 차지하던 위치에 표시됩니다. EVE는 GOLD Prev Screen을 PREVIOUS WINDOW로 정의합니다.
REVERSE	현재 버퍼의 방향을 역방향으로, 즉 왼쪽 위로 설정합니다. 버퍼 방향은 상태 표시줄에 나타납니다.
SET CURSOR BOUND	텍스트 흐름을 따라 커서를 이동시킵니다. 버퍼의 사용되지 않은 부분으로는 커서를 이동할 수 없습니다. EDT, WPS 및 기타 편집기의 커서 동작과 유사합니다.
SET CURSOR FREE	기본 설정입니다. 버퍼의 임의 위치로 커서를 이동한 다음 텍스트를 입력할 수 있습니다.
SET SCROLL MARGINS	커서를 위/아래로 이동할 때 스크롤이 자동으로 시작되는 위쪽 및 아래쪽 간격을 설정합니다. 이 간격은 행 수로 지정하거나 창 크기의 백분율로 지정할 수 있습니다. 기본 설정은 0입니다. 즉, 창의 맨 위나 아래를 지나 커서를 이동할 때 스크롤이 시작됩니다.
SHIFT LEFT	현재 EVE 창을 지정된 열 수만큼 왼쪽으로 이동합니다. SHIFT RIGHT 및 SHIFT LEFT 명령을 사용하면 창 너비를 변경하거나 132열 모드를 사용할 필요 없이 긴 텍스트 행의 표시되지 않은 부분을 볼 수 있습니다. SHIFT LEFT 명령은 SHIFT RIGHT 명령이 사용된 경우에만 창을 이동합니다.

표 8-2

커서를 이동하는 EVE 명령(계속)

명령	기능
SHIFT RIGHT	현재 EVE 창을 지정된 열 수만큼 오른쪽으로 이동합니다. SHIFT RIGHT 및 SHIFT LEFT 명령을 사용하면 창 너비를 변경할 필요 없이 긴 텍스트 행의 표시되지 않은 부분을 볼 수 있습니다
START OF LINE	커서를 현재 행의 시작 부분으로 이동합니다. 기본적으로 EVE는 Ctrl/H 및 GOLD 왼쪽 화살표 키를 START OF LINE으로 정의합니다.
TOP	커서를 현재 버퍼의 시작 부분(왼쪽 위 모서리)으로 이동합니다. 기본적으로 EVE는 GOLD 위쪽 화살표 키를 TOP으로 정의합니다.

연습: EVE에서 커서 이동

버퍼에서 커서를 이동하려면 다음 단계를 따릅니다.

1. 다음 명령을 사용하여 EVE를 실행하고 SCHEDULE.DAT 버퍼를 만듭니다.

```
$ EDIT SCHEDULE.DAT
```

버퍼 맨 위에 커서가 놓이고 텍스트 입력 대기 상태가 됩니다.

2. 다음 텍스트를 입력합니다.

```
Schedule for 1 July
10:00 AM meeting with supervisor
Read and review memo from Sally
Work on Pascal program
```

텍스트를 입력하면 버퍼의 [End of file] 표시가 아래로 이동하고 입력한 텍스트의 맨 끝에 커서가 놓입니다.

3. TOP 명령을 입력하여 커서를 파일의 시작 부분으로 이동합니다.
4. Ctrl/E를 눌러 커서를 첫 번째 텍스트 행의 끝 부분으로 이동합니다.
Ctrl/E는 EVE와 DCL에서 동일한 기능을 수행합니다.
5. BOTTOM 명령을 입력하여 커서를 버퍼의 끝 부분으로 이동합니다.
6. 위쪽 화살표 키를 눌러 커서를 한 행 위인 네 번째 텍스트 행으로 이동합니다.

7. **Change Direction** 키를 눌러 현재 버퍼 방향을 역방향으로 변경합니다.
8. **Move by Line** 키를 눌러 커서를 세 번째 텍스트 행의 시작 부분으로 이동합니다.
9. **LINE 1** 명령을 입력하여 커서를 버퍼의 첫 번째 행 시작 부분으로 이동합니다.
10. EVE를 종료하려면 **Ctrl/Z**를 누릅니다.

텍스트 입력

키보드 문자, 전체 파일 및 인쇄되지 않는 특수 문자(예: 제어 문자)를 현재 편집 중인 버퍼에 입력할 수 있습니다. 텍스트를 입력하려면 키패드나 명령을 사용합니다. 또한 텍스트, 파일 및 특수 문자를 버퍼에 추가할 수도 있습니다.

텍스트 추가

키보드에서 문자를 입력하여 버퍼의 현재 커서 위치에 추가할 수 있습니다. 버퍼의 현재 모드(삽입 또는 겹쳐쓰기)에 따라 입력한 문자는 기존 문자를 보충하거나 교체합니다.

파일 포함

Do 키를 누르고 **INCLUDE FILE** 명령을 입력하여 전체 파일을 추가할 수 있습니다.

File to include: 프롬프트에서 파일 지정 방법을 입력하고 **Enter** 키를 누릅니다. 버퍼의 현재 모드(삽입 또는 겹쳐쓰기)에 상관 없이 커서가 위치한 행의 바로 앞에 지정된 파일의 전체 내용이 삽입됩니다.

파일 지정 방법에서 와일드카드를 사용할 수 있습니다. 와일드카드가 포함된 파일 지정 방법과 일치하는 파일이 여러 개이면, 선택 목록이 표시되고 더 완전한 파일 지정 방법을 제공하라는 메시지가 나타납니다. 지정된 파일이 존재하지 않으면 **EVE**는 파일을 포함할 수 없다는 메시지를 표시합니다.

인쇄되지 않는 특수 문자

QUOTE 명령을 사용하면 **Ctrl/V** 다음에 해당 문자를 눌러 인쇄되지 않는 특수 문자를 추가할 수 있습니다. 예를 들어, 이스케이프 문자를 버퍼에 삽입하려면 **Ctrl/V**와 **Ctrl/[**를 차례로 누릅니다. 버퍼의 현재 모드(삽입 또는 겹쳐쓰기)에 따라 특수 문자는 기존 문자를 보충하거나 교체합니다.

텍스트 입력을 위한 EVE 편집 키

다음 표는 텍스트 입력에 사용할 수 있는 EVE 편집 키를 보여 줍니다.

표 8-3

키 또는 키 시퀀스	기능
Ctrl/A	CHANGE MODE 명령과 동일합니다. 강조된 상태 표시줄에 나타난 현재 버퍼의 편집 모드를 변경합니다. 삽입 모드는 기존 텍스트를 밀어내면서 현재 위치에 텍스트를 삽입하고, 겹쳐쓰기 모드는 현재 위치에서 기존 텍스트를 덮어씁니다. VT200, VT300 및 VT400 계열 터미널에서는 F14 키가 CHANGE MODE로 정의되고, VT100 계열 터미널에서는 키패드의 Enter 키가 CHANGE MODE로 정의됩니다.
Ctrl/V	QUOTE 명령과 동일합니다. 인쇄되지 않는 문자나 제어 코드를 삽입할 수 있습니다. 특수 문자를 검색하려면, Find 키를 누르고 Ctrl/V와 찾으려는 특수 문자를 누른 다음, Enter 키를 눌러 검색을 시작합니다.

텍스트 입력을 위한 EVE 명령

다음 표는 텍스트 입력에 사용할 수 있는 명령을 보여 줍니다.

표 8-4

명령	기능
CHANGE MODE	Ctrl/A와 동일합니다. 강조된 상태 표시줄에 나타난 현재 편집 모드를 변경합니다. 삽입 모드는 기존 텍스트를 밀어내면서 현재 위치에 텍스트를 삽입하고, 겹쳐쓰기 모드는 현재 위치에서 기존 텍스트를 덮어씁니다. VT200, VT300 및 VT400 계열 터미널에서는 F14 키가 CHANGE MODE로 정의되고, VT100 계열 터미널에서는 키패드의 Enter 키가 CHANGE MODE로 정의됩니다.
INCLUDE FILE	현재 버퍼의 커서 위치 바로 위에 있는 행에 지정된 파일의 내용을 삽입합니다. 파일을 결합하는 경우에 유용합니다.

표 8-4

(계속)

명령	기능
INSERT MODE	현재 버퍼의 모드를 겹쳐쓰기와 반대인 삽입 모드로 설정합니다. 삽입 모드는 기존 텍스트를 밀어내면서 현재 위치에 텍스트를 삽입합니다.
OVERSTRIKE MODE	현재 버퍼의 모드를 삽입과 반대인 겹쳐쓰기 모드로 설정합니다. 겹쳐쓰기 모드는 현재 위치에서 기존 텍스트를 덮어씁니다.
QUOTE	Ctrl/V와 동일합니다. 인쇄되지 않는 문자나 제어 코드(키를 눌러 지정)를 입력할 수 있습니다. FIND 또는 REPLACE 명령에 사용할 문자열을 입력할 때 제어 코드나 기타 문자를 인용 부호로 묶을 수 있습니다. 예를 들어, Tab 키를 인용 부호로 묶어 탭 문자를 검색할 수 있습니다.

버퍼 모드 설정

텍스트 입력을 시작하기 전에 버퍼의 현재 모드(삽입 또는 겹쳐쓰기)를 확인합니다.

버퍼의 현재 모드를 확인하려면 강조된 상태 표시줄을 살펴봅니다. 버퍼가 삽입 모드이면 입력한 텍스트가 커서 위치에 삽입되고 기존 텍스트는 뒤로 밀립니다. 반면, 버퍼가 겹쳐쓰기 모드이면 입력한 텍스트가 기존 텍스트를 덮어쓰면서 커서 위치에 삽입됩니다.

현재 버퍼 모드를 변경하려면 Ctrl/A를 누릅니다.

연습: 삽입 또는 겹쳐쓰기 모드에서 텍스트 추가

삽입 모드 및 겹쳐쓰기 모드에서 텍스트를 파일에 추가하려면 다음 단계를 따릅니다.

1. EVE를 실행하여 기존의 SCHEDULE.DAT 파일을 편집합니다.
2. 강조된 상태 표시줄에서 EVE가 삽입 모드인지 확인합니다.
3. EVE가 겹쳐쓰기 모드이면 Ctrl/A를 눌러 삽입 모드로 변경합니다.
4. 커서를 *supervisor*의 첫 글자인 *s*로 이동하고 *Engineering*을 입력한 다음, 스페이스 바를 누릅니다.

*Engineering*이 텍스트 버퍼에 삽입되고 같은 줄에 있는 텍스트의 나머지 부분은 오른쪽으로 이동합니다.

```
Schedule for 1 July
10:00 AM meeting with Engineering supervisor
Read and review memo from Sally
Work on Pascal program
[End of file]
```

Buffer: SCHEDULE.DAT | Write | Insert | Forward

5. **Ctrl/A**를 눌러 겹쳐쓰기 모드로 변경합니다.
6. 커서를 *Sally*의 첫 글자인 **S**로 이동하고 *Peggy*를 입력합니다.

*Peggy*가 *Sally*를 덮어쓰면서 텍스트 버퍼에 삽입됩니다.

```
Schedule for 1 July
10:00 AM meeting with Engineering supervisor
Read and review memo from Peggy
Work on Pascal program
[End of file]
```

Buffer: SCHEDULE.DAT | Write | Overstrike | Forward

7. EVE를 종료하려면 **Ctrl/Z**를 누릅니다.

텍스트 지우기 및 복원

EVE를 사용하면 간단하게 편집 세션 도중에 만들어진 실수를 고치거나 텍스트를 지울 수 있습니다. 텍스트를 실수로 지운 경우에는 가장 최근에 지운 텍스트를 원래 위치로 복원하거나 커서를 이동하여 다른 위치로 복원할 수 있습니다.

버퍼에서 텍스트를 지우려면 지우려는 텍스트로 커서를 이동한 다음, 해당 편집 키를 누르거나 해당 **EVE** 명령을 입력합니다.

표 8-5는 텍스트를 지우고 복원하는 **EVE** 편집 키를 보여 줍니다.

표 8-5 텍스트 지우기 및 복원을 위한 **EVE** 편집 키

키 또는 키 시퀀스	기능
Delete 키 또는 Delete	커서 왼쪽에 있는 문자를 지웁니다. DELETE 명령과 동일합니다. 삭제 보류가 활성화되어 있으면 DELETE 는 선택 범위의 텍스트를 지운 다음, Restore Selection 버퍼에 놓습니다. 삭제 보류 사용에 대한 자세한 내용은 279페이지의 “텍스트 이동”을 참조하십시오.
Ctrl/J	ERASE WORD 와 동일합니다. 현재 단어를 지우거나, 커서가 단어 사이에 있으면 다음 단어를 지웁니다. VT200 , VT300 및 VT400 계열 터미널에서는 F13 키가 ERASE WORD 로 정의되고, VT100 계열 터미널에서는 키패드의 쉼표(,) 키가 ERASE WORD 로 정의됩니다.
Ctrl/U	ERASE START OF LINE 과 동일합니다. 행의 시작 부분에서 커서 왼쪽에 있는 문자를 지웁니다.
GOLD Insert Here	RESTORE 와 동일합니다. EVE 명령이나 편집 키를 사용하여 마지막으로 지운 단어, 행 또는 문장을 현재 위치에 다시 삽입합니다.
GOLD F13	RESTORE WORD 와 동일합니다(WPS 키패드 제외). 마지막으로 지운 단어를 현재 위치에 다시 삽입합니다.

표 8-6은 텍스트를 지우고 복원하는 EVE 명령을 보여 줍니다.

표 8-6

텍스트 지우기 및 복원을 위한 EVE 명령

명령	기능
DELETE	커서 왼쪽에 있는 문자를 지웁니다. 삽입 모드는 기존 텍스트를 이동하여 삭제된 문자를 메우고, 겹쳐쓰기 모드는 삭제된 문자가 공백으로 바뀝니다. 행의 시작 부분에서 DELETE 를 사용하면 모드에 상관 없이 이전 행의 캐리지 리턴이 지워지고 현재 행이 위로 올라갑니다. 삭제 보류가 활성화되어 있으면 DELETE 는 선택 범위의 텍스트를 지운 다음, Restore Selection 버퍼에 놓습니다. 삭제 보류 사용에 대한 자세한 내용은 279페이지의 “텍스트 이동”을 참조하십시오.
ERASE CHARACTER	커서가 위치한 곳의 문자를 지웁니다. 삽입 모드는 기존 텍스트를 이동하여 삭제된 문자를 메우고, 겹쳐쓰기 모드는 삭제된 문자가 공백으로 바뀝니다. 행의 끝 부분에 커서가 있으면 모드에 상관 없이 캐리지 리턴이 지워지고 다음 행이 위로 올라옵니다.
ERASE LINE	현재 문자에서 행 끝 부분까지 지우고 다음 행을 현재 행의 끝 부분에 연결합니다. 행의 끝 부분에 커서가 있으면 캐리지 리턴만 지워지고 다음 행이 위로 올라옵니다.
ERASE PREVIOUS WORD	이전 단어나 커서가 위치한 곳의 단어를 지웁니다. 커서가 단어 사이에 있거나 단어의 첫 문자에 있으면 이전 단어가 지워집니다. 커서가 단어 중간에 있으면 해당 단어가 모두 지워집니다(ERASE WORD 와 동일). 커서가 행의 시작 부분에 있으면 이전 행 끝에 있는 캐리지 리턴이 지워지고 현재 행이 위로 올라갑니다.
ERASE START OF LINE	커서 왼쪽에 있는 문자부터 행의 시작 부분까지 현재 행을 지웁니다. 커서가 행의 시작 부분에 있으면 아무 것도 지워지지 않습니다.

표 8-6

텍스트 지우기 및 복원을 위한 EVE 명령(계속)

명령	기능
ERASE WORD	현재 단어를 지우거나 커서가 단어 사이에 있으면 다음 단어를 지웁니다. Ctrl/J와 동일합니다. VT200, VT300 및 VT400 계열 터미널에서는 F13키가 ERASE WORD로 정의되고, VT100 계열 터미널에서는 키패드의 쉼표(,) 키가 ERASE WORD로 정의됩니다. 커서가 행의 끝 부분에 있으면 캐리지 리턴만 지워지고 다음 행이 위로 올라옵니다.
RESTORE	EVE 명령이나 편집 키를 사용하여 마지막으로 지운 단어, 행 또는 문장을 현재 위치에 다시 삽입합니다. 단일 문자는 RESTORE에 의해 복원되지 않습니다. EVE는 GOLD Insert Here를 RESTORE로 정의합니다.
RESTORE CHARACTER	EVE 명령이나 편집 키를 사용하여 마지막으로 지운 문자를 현재 위치에 다시 삽입합니다. 겹쳐쓰기 모드에서는 커서가 있는 곳의 문자가 복원된 문자로 바뀌고, 삽입 모드에서는 기존 텍스트가 뒤로 밀리면서 현재 위치에 복원된 문자가 삽입됩니다.
RESTORE LINE	EVE 명령이나 편집 키를 사용하여 마지막으로 지운 행을 현재 위치에 다시 삽입합니다.
RESTORE SELECTION	삭제 보류 작업을 사용하여 마지막으로 지운 텍스트를 현재 위치에 다시 삽입합니다. 삭제 보류 사용에 대한 자세한 내용은 289페이지의 “삭제 보류 사용”을 참조하십시오.
RESTORE WORD	EVE 명령이나 편집 키를 사용하여 마지막으로 지운 단어를 현재 위치에 다시 삽입합니다. EVE는 GOLD F13을 RESTORE WORD로 정의합니다(WPS 키패드 제외).

연습: 텍스트 지우기 및 복원

텍스트를 지우고 복원하려면 다음 단계를 따릅니다.

1. EVE를 실행하여 RHYMES.DAT 버퍼를 만들고 다음 텍스트를 입력합니다.

```
She rhymes with tree,  
also with bee,  
and this one makes three.
```

2. 커서를 *also*의 문자 *l*로 이동합니다. ERASE LINE 명령을 입력합니다.

*also*의 문자 *l*부터 행의 끝 부분까지 모든 문자가 지워지고 다음 행이 현재 행에 연결됩니다.

```
She rhymes with tree,  
aand this one makes three.
```

3. 커서를 *rhymes*의 문자 *y*로 이동합니다. ERASE WORD 명령을 입력합니다.

*rhymes*가 지워지고 나머지 텍스트가 왼쪽으로 이동합니다.

```
She with tree,  
aand this one makes three.
```

4. 커서를 두 번째 행의 두 번째 문자인 *a*로 이동합니다. RESTORE LINE 명령을 입력합니다.

방금 삭제했던 행(*lso with bee*)이 복원됩니다.

```
She with tree,  
also with bee,  
and this one makes three.
```

5. 커서를 첫 번째 행에 있는 *with*의 문자 *w*로 이동합니다. RESTORE WORD 명령을 입력합니다.

방금 삭제했던 단어(*rhymes*)가 복원됩니다.

```
She rhymes with tree,  
also with bee,  
and this one makes three.
```

6. EVE를 종료하려면 Ctrl/Z를 누릅니다.

279페이지의 “텍스트 이동”에서는 버퍼에서 텍스트를 지우는 데 함께 사용할 수 있는 SELECT 및 REMOVE 명령의 기능이 설명됩니다.

텍스트 이동

EVE 명령을 사용하면 텍스트 섹션을 선택한 다음 복사, 이동, 삭제 또는 기타 편집 작업을 수행할 수 있습니다. 이 절에서는 텍스트를 이동하는 방법에 대해 설명합니다.

특정 버퍼에서 다른 버퍼로 텍스트를 이동하는 방법에 대한 자세한 내용은 314페이지의 “버퍼 사용”을 참조하십시오.

텍스트의 선형 범위가 아니라 사각형 영역(상자)을 선택하여 텍스트를 이동, 삭제 또는 복제할 수도 있습니다. 상자 편집 명령 사용에 대한 자세한 내용은 285페이지의 “상자 편집”을 참조하십시오.

텍스트를 이동하려면 다음 단계를 따릅니다.

표 8-7

단계	작업
1	EVE에서 파일을 연 다음, 이동하려는 첫 번째 문자로 커서를 이동합니다.
2	Select 키를 누릅니다.
3	이동할 마지막 문자 바로 뒤에 있는 문자로 커서를 이동합니다. 역방향의 경우에는 커서를 마지막 문자로 이동합니다. 이동할 텍스트가 역방향 비디오 로 강조됩니다. 버퍼에서 텍스트를 제거하지 않으려면 Select 키를 다시 눌러 선택을 취소합니다.
4	Remove 키를 누릅니다. EVE는 강조된 텍스트를 화면에서 삭제한 다음, Insert Here 버퍼에 놓습니다.
5	Insert Here 키를 눌러 텍스트를 삽입합니다. 텍스트가 커서 위치에 삽입됩니다. 새로운 텍스트를 선택하여 Insert Here 버퍼에 놓을 때까지 Insert Here 버퍼에 포함된 텍스트를 횡수에 상관 없이 원하는 커서 위치에 삽입할 수 있습니다. Insert Here 버퍼는 마지막으로 복사 또는 제거된 텍스트를 포함합니다.

표 8-8에는 텍스트를 이동하는 데 사용되는 EVE 편집 키가 설명되어 있습니다.

표 8-8 텍스트를 이동하는 EVE 편집 키

키 또는 키 시퀀스	기능
Insert Here	INSERT HERE 또는 PASTE 명령과 동일합니다. 제거 또는 복사된 텍스트를 현재 위치에 삽입합니다.
Remove	REMOVE 또는 CUT 명령과 동일합니다. SELECT로 표시되거나 FIND에 의해 강조된 텍스트를 제거한 다음, Insert Here 버퍼에 놓습니다.
Select	처음 커서 위치에서 커서를 이동한 위치까지 텍스트를 표시합니다(역방향 비디오로 강조). 강조된 텍스트를 선택 영역이라고 부릅니다. 선택을 취소하려면 Select 키를 다시 누르거나 RESET을 사용합니다.
GOLD Select	RESET과 동일합니다. 다음 작업을 취소하고 버퍼 방향을 정방향으로 초기화합니다. • 선택 또는 검색 범위 강조 • GOLD 키를 누르거나 반복 횟수를 위한 GOLD n 조합 • 불완전하거나 다시 호출된 명령행 또는 Choices 버퍼 표시 • 작업 중이던 버퍼를 반환하는 SHOW, SHOW DEFAULTS BUFFER, SHOW SUMMARY 또는 SHOW WILDCARDS 출력
GOLD Remove	STORE TEXT 또는 COPY 명령과 동일합니다. SELECT 또는 FIND에 의해 표시된 텍스트를 복사하여 Insert Here 버퍼에 놓습니다. 복사된 텍스트는 원래 위치에 제거되지 않습니다.

표 8-9에는 텍스트를 이동하는 데 사용되는 EVE 명령이 설명되어 있습니다.

표 8-9

텍스트를 이동하는 EVE 명령

명령	기능
INSERT HERE 또는 PASTE	복사 또는 제거된 텍스트를 삽입합니다. 기본적으로 EVE는 E2 키(VT200, VT300 및 VT400 계열 터미널의 경우 미니 키패드의 Insert Here)와 KP9 키(VT100 계열 터미널의 경우)를 INSERT HERE로 정의합니다.
REMOVE 또는 CUT	SELECT로 표시되거나 FIND에 의해 강조된 텍스트를 제거한 다음, Insert Here 버퍼에 놓습니다. 기본적으로 EVE는 E3 키(VT200, VT300 및 VT400 계열 터미널의 경우 미니 키패드의 Remove)와 KP8 키(VT100 계열 터미널의 경우)를 REMOVE로 정의합니다.
RESET	다음 작업을 취소하고 버퍼 방향을 정방향으로 초기화합니다. • 선택 또는 검색 범위 강조 • GOLD 키를 누르거나 반복 횟수를 위한 GOLD n 조합 • 완료되지 않았거나 다시 호출된 명령행 또는 Choices 버퍼 표시 • 작업 중이던 버퍼를 반환하는 SHOW, SHOW DEFAULTS BUFFER, SHOW SUMMARY 또는 SHOW WILDCARDS 출력
RESTORE SELECTION	삭제 보류 작업에 의해 삭제된 텍스트를 다시 삽입합니다. 삭제 보류 사용에 대한 자세한 내용은 289페이지의 “삭제 보류 사용”을 참조하십시오.
SELECT	처음 커서 위치에서 커서를 이동한 위치까지 텍스트를 역방향 비디오로 강조합니다. 강조된 텍스트를 선택 영역이라고 부릅니다. 선택을 취소하려면 SELECT 명령을 다시 입력하거나 RESET을 사용합니다. 기본적으로 EVE는 E4 키(VT200, VT300 및 VT400 계열 터미널의 경우 미니 키패드의 Select)와 KP7 키(VT100 계열 터미널의 경우)를 SELECT로 정의합니다.

표 8-9 텍스트를 이동하는 EVE 명령(계속)

명령	기능
SELECT ALL	커서 위치에 상관 없이 현재 버퍼의 모든 텍스트를 역방향 비디오로 강조합니다. 강조된 텍스트를 선택 영역이라고 부릅니다. 선택을 취소하려면 SELECT 명령을 입력하거나 RESET 을 사용합니다. SELECT ALL 명령은 실수로 버퍼의 모든 텍스트를 지우는 것을 방지하기 위해 삭제 보류를 일시적으로 비활성화합니다.
SET NOPENDING DELETE	기본 설정입니다. Delete 키를 사용하거나 새 텍스트를 입력할 때 선택된 텍스트가 삭제되지 않게 합니다. 버퍼의 텍스트를 선택하고 새 텍스트를 입력하면 문자가 선택 범위에 추가되고 Delete 키를 사용하면 커서 왼쪽에 있는 문자만 지워집니다.
SET PENDING DELETE	텍스트 블록을 신속하게 지울 수 있게 하는 삭제 보류를 활성화합니다. 우선 삭제 보류를 활성화한 다음, SELECT 명령을 사용하여 지울 텍스트를 선택합니다. Delete 키나 영숫자 키패드의 다른 키를 눌러 텍스트를 지웁니다. 삭제한 텍스트를 다시 삽입하려면 원하는 위치로 커서를 이동한 다음, RESTORE SELECTION 명령을 입력합니다. 기본 설정은 SET NOPENDING DELETE 입니다.
STORE TEXT 또는 COPY	SELECT 또는 FIND 에 의해 표시된 텍스트를 복사하여 Insert Here 버퍼에 놓습니다. 복사된 텍스트는 원래 위치에서 제거되지 않습니다.

연습: 텍스트 이동

특정 위치의 텍스트를 선택 및 제거하여 다른 위치에 삽입하려면 다음 절차를 수행합니다.

1. EVE를 실행하여 RHYMES.DAT 파일을 편집합니다.
2. 커서를 RHYMES.DAT의 두 번째 행의 시작 부분으로 이동한 다음, **Select** 키를 누릅니다.
3. 아래쪽 화살표 키를 한 번 누릅니다.

텍스트의 두 번째 행이 강조됩니다.

4. **Remove** 키를 누릅니다.

텍스트의 두 번째 행이 현재 버퍼에서 제거됩니다.

```
She rhymes with tree,
and this one makes three.
[End of file]
```

5. **Enter** 키를 두 번 누른 다음, **Insert Here** 키를 누릅니다.

Insert Here 버퍼의 텍스트가 현재 커서 위치에 삽입됩니다.

```
She rhymes with tree,

also with bee,
and this one makes three.
[End of file]
```

6. EVE를 종료하려면 **Ctrl/Z**를 누릅니다.

텍스트 복사

COPY 명령을 사용하면 어느 곳에서든 텍스트를 복사할 수 있습니다. STORE TEXT 명령은 COPY 명령과 동일합니다. 따라서, 다음 연습에 사용된 COPY 명령 대신 STORE TEXT 명령을 사용할 수 있습니다.

연습: 텍스트 복사

버퍼가 정방향으로 설정된 경우 텍스트를 복사하려면 다음 단계를 따릅니다.

1. EVE를 실행하여 RHYMES.DAT 파일을 편집합니다.
2. 커서를 텍스트의 첫 번째 행으로 이동합니다.
3. Select 키를 누릅니다.
4. Ctrl/E를 눌러 커서를 첫 번째 행의 끝 부분으로 이동합니다.
5. COPY 명령을 입력합니다. 선택된 텍스트가 복사되어 Insert Here 버퍼에 포함됩니다.
6. 커서를 *also with bee*, 위에 있는 행으로 이동합니다.
7. Insert Here 키를 누릅니다. 버퍼 모양이 다음과 같아야 합니다.

```
She rhymes with tree,  
  
She rhymes with tree,  
also with bee,  
and this one makes three.  
[End of file]
```

8. 커서를 첫 번째 행의 시작 부분으로 이동합니다. Select 키를 누른 다음, Remove 키를 눌러 첫 번째 행을 삭제합니다.
9. EVE를 종료하려면 Ctrl/Z를 누릅니다.

상자 편집

표준 선행 범위 뿐만 아니라 사각형 영역(상자)을 가진 텍스트를 편집할 수 있습니다. 예를 들어, 테이블 열이나 목록이 포함된 상자를 선택한 다음, 잘라내기/붙여넣기를 수행하거나 상자에서 다른 편집 작업을 수행할 수 있습니다.

텍스트 상자 선택

텍스트 상자를 선택하려면 다음 단계를 수행합니다.

1. 선택을 시작할 위치에 커서를 놓습니다. 일반적으로 상자의 왼쪽 위 모서리가 되는 위치에 놓습니다.
2. **BOX SELECT** 명령을 입력합니다.
3. 대각선으로 상자의 맞은편 모서리가 되는 위치에 커서를 이동합니다. 일반적으로 왼쪽 위에서 오른쪽 아래로 이동합니다.

커서를 이동할 때, 가로지르는 텍스트 부분이 굵은 비디오로 강조됩니다. 일반 선택 영역의 경우에는 역방향 비디오로 강조됩니다. 상자는 대각선으로 맞은편에 있는 모서리에 의해 정의됩니다. 왼쪽 위에서 오른쪽 아래로 이동할 경우, 커서가 있는 문자는 상자의 바깥쪽에 놓입니다. 즉, 상자의 오른쪽 아래 모서리가 커서 왼쪽에 위치합니다.

선행 또는 사각형 영역에서 일반적으로 사용되는 편집 명령을 통해 상자를 편집할 수 있으므로 키를 다시 정의할 필요가 없습니다. 자세한 내용은 *Extensible Versatile Editor Reference Manual*을 참조하십시오.

FIND SELECTED(선택 영역이 행을 가로지르지 않는 경우) 또는 **OPEN SELECTED**를 사용할 수 있습니다. 또한, 삭제 보류를 사용할 수도 있습니다.

열이 여러 개인 테이블이나 목록을 편집하는 경우처럼 여러 개의 상자를 편집할 경우에는 **SET BOX SELECT** 명령을 사용합니다. **SET BOX SELECT**는 여러 명령과 키를 **BOX** 명령과 동일한 것으로 재정의하므로 해당 편집 작업이 선행 범위 대신 상자에서 작동합니다.

상자 선택을 취소하려면 **SELECT** 또는 **BOX SELECT**를 반복하거나 **RESET**을 사용합니다.

텍스트 상자 잘라내기/붙여넣기

일반적으로 상자를 잘라내면 상자 오른쪽에 있는 텍스트의 열 맞춤을 유지하기 위해 잘라낸 영역이 공백으로 채워지고, 상자를 붙여넣으면 기존 텍스트를 덮어씁니다. 상자 안의 탭 문자나 상자와 겹쳐지는 탭 문자도 텍스트의 열 맞춤을 유지하기 위해 공백으로 변환됩니다.

표 8-10에는 상자 편집을 위한 EVE 명령이 나와 있습니다.

표 8-10

상자 편집을 위한 EVE 명령

명령	기능
BOX COPY	텍스트 상자를 제거하지 않고 복사하며, 복사된 텍스트 상자를 어느 곳에나 붙여넣을 수 있습니다.
BOX CUT	텍스트 상자를 잘라내며, 잘라진 텍스트 상자를 어느 곳에나 붙여넣을 수 있습니다. 일반적으로 상자 오른쪽에 있는 텍스트의 열 맞춤을 유지하기 위해 잘라낸 영역이 공백으로 채워집니다.
BOX CUT INSERT	상자를 잘라내며, 이 때 간격을 없애기 위해 상자 오른쪽에 있는 텍스트를 왼쪽으로 “당깁니다”.
BOX CUT OVERSTRIKE	상자를 잘라내며, 이 때 상자 오른쪽에 있는 텍스트의 열 맞춤을 유지하기 위해 잘라낸 영역을 공백으로 채웁니다.
BOX PASTE	복사하거나 잘라낸 텍스트 상자를 붙여넣습니다. 일반적으로 기존 텍스트를 덮어씁니다.
BOX PASTE INSERT	기존 텍스트를 오른쪽으로 밀어내면서 상자를 붙여넣습니다.
BOX PASTE OVERSTRIKE	기존 텍스트를 덮어쓰면서 상자를 붙여넣습니다.
BOX SELECT	텍스트 상자를 선택합니다. 일반적으로 상자의 왼쪽 위 모서리에서 시작하여 오른쪽 아래 모서리가 되는 위치로 커서를 이동합니다.
RESTORE BOX SELECTION	삭제 보류를 사용하여 지운 상자를 되돌립니다(삭제 취소). 일반적으로 기존 텍스트를 덮어씁니다.
SET BOX NOPAD	버퍼가 겹쳐쓰기 모드가 아닌 경우, 상자 편집의 채우기와 겹쳐쓰기를 비활성화합니다.

표 8-10

상자 편집을 위한 EVE 명령(계속)

명령	기능
SET BOX NOSELECT	기본 설정입니다. 상자 선택, 잘라내기 및 붙여넣기를 비활성화합니다. SELECT , COPY 및 REMOVE 와 같은 명령에 표준 선택 범위가 사용됩니다. 상자를 편집하려면 BOX 명령을 사용합니다.
SET BOX PAD	기본 설정입니다. 버퍼 모드에 상관 없이 상자 편집의 자동 채우기와 겹쳐쓰기를 활성화합니다.
SET BOX SELECT	상자 선택을 활성화하여 SELECT , REMOVE 및 INSERT HERE 와 같은 명령을 해당 BOX 명령과 똑같이 만들기 때 문에 키를 다시 정의할 필요가 없습니다.

연습: 텍스트 잘라내기/붙여넣기

텍스트 상자를 선택한 다음, 잘라서 붙여넣으려면 다음 단계를 따릅니다.

1. EVE를 실행하여 **CITIES.DAT** 버퍼를 만들고 다음 텍스트를 입력합니다.

```
Rome    Paris    New York
London  Tunis    Boston
Tokyo   Bonn     Lisbon
```

2. 커서를 **Paris**의 문자 **P** 왼쪽으로 이동합니다. **BOX SELECT** 명령을 입력합니다.
3. 커서를 **Bonn**의 마지막 문자 **n**에서 오른쪽으로 두 칸 떨어진 위치(대각선으로 상자의 맞은편 모서리가 되는 위치)로 이동합니다. 가로지르는 텍스트 부분이 굵은 비디오로 강조됩니다. **BOX CUT** 명령을 입력합니다.

텍스트 상자가 제거됩니다.

4. 커서를 **New York**로 시작되는 열의 오른쪽으로 이동합니다.
5. **BOX PASTE** 명령을 입력합니다.

EVE는 다음과 같이 텍스트 상자를 새 열에 붙여넣습니다.

```
Rome      New York    Paris
London    Boston     Tunis
Tokyo     Lisbon     Bonn
[End of file]
```

SET BOX SELECT 명령

표 8-11에는 SET BOX SELECT 명령이 나와 있습니다.

표 8-11

SET BOX SELECT 명령

명령	해당 SET BOX SELECT 명령
INSERT HERE 또는 PASTE	BOX PASTE
REMOVE 또는 CUT	BOX CUT
RESTORE SELECTION	RESTORE BOX SELECTION
SELECT	BOX SELECT
STORE TEXT 또는 COPY	BOX COPY

위 표에 나온 것처럼 키를 다시 정의할 필요 없이 **Select**, **Remove** 및 **Insert Here** 키를 사용하여 상자를 선택, 잘라내기 및 붙여넣기할 수 있습니다.

삭제 보류 사용

삭제 보류를 사용하여 선택된 텍스트를 지울 수 있습니다. 삭제 보류는 새 텍스트를 입력하거나, 스페이스 바를 누르거나, 삭제 기능을 사용(일반적으로 **Delete** 키를 누름)하여 선택 영역을 삭제하는 것을 말합니다.

상자 선택 영역의 경우에는 삭제 보류가 **BOX CUT**과 동일한 기능을 수행하는데, 대개 상자 오른쪽에 있는 텍스트의 열 맞추를 유지하기 위해 잘라낸 영역을 공백으로 채웁니다.

삭제 보류는 **Insert Here** 버퍼를 사용하지 않기 때문에 텍스트 잘라내기/붙여넣기를 대신 하는 방법을 제공합니다. 삭제 보류에 대한 자세한 내용은 EVE 온라인 도움말에서 **Pending Delete** 항목을 참조하십시오.

삭제 보류로 선택 영역 지우기

삭제 보류를 사용하여 선택 영역을 지우려면 다음 단계를 따릅니다.

1. EVE에서 파일을 엽니다.
2. 삭제 보류를 활성화하려면 **SET PENDING DELETE** 명령을 사용합니다.
기본 설정은 **SET NOPENDING DELETE**입니다.
3. 삭제할 텍스트를 선택합니다. **SELECT** 또는 **BOX SELECT**를 사용할 수 있습니다.
SELECT ALL은 사용할 수 없습니다.
4. 새 텍스트를 입력하거나 **DELETE** 명령을 사용합니다.

삭제 보류로 지운 선택 영역 복원

삭제 보류로 지운 텍스트를 복원하려면 다음 단계를 따릅니다.

1. 텍스트를 복원할 위치로 커서를 이동합니다. 상자 선택 영역을 복원할 경우에는 상자의 왼쪽 위 모서리가 되는 위치에 커서를 놓습니다.
2. **RESTORE SELECTION**을 사용합니다. 삭제 보류를 사용하여 상자를 지웠다면 **RESTORE BOX SELECTION**을 사용합니다. **SET BOX SELECT**를 사용한 경우에는 키를 다시 정의할 필요 없이 **RESTORE SELECTION**을 사용할 수 있습니다.

상자 복원은 **BOX PASTE** 명령을 사용할 때와 동일하게 수행되는데, 대개 기존 텍스트를 덮어씁니다. **SET BOX NOPAD** 명령을 사용할 경우, 상태 표시줄에 나타나는 버퍼의 현재 모드(삽입 또는 겹쳐쓰기)에 따라 상자 편집의 효과가 달라집니다.

- 삽입 모드의 경우, 상자를 잘라내면 간격을 없애기 위해 상자 오른쪽에 있는 텍스트가 왼쪽으로 “당겨집니다”. 텍스트를 왼쪽으로 당길 때, 상자 오른쪽에 있는 탭 문자도 열 맞춤을 유지하기 위해 공백으로 변환됩니다. 이 방법은 4열 테이블을 2열 테이블로 바꾸는 경우처럼 테이블이나 목록에서 열을 제거하는 경우에 유용합니다. 또한, 상자를 붙여넣으면 기존 텍스트가 오른쪽으로 밀리므로 테이블 중간에 열을 추가하는 데 유용합니다.
- 겹쳐쓰기 모드의 경우, 상자를 잘라내면 상자 오른쪽에 있는 텍스트의 열 맞춤을 유지하기 위해 잘라낸 영역이 공백으로 채워집니다. 상자를 붙여넣으면 기존 텍스트를 덮어씁니다. 이러한 효과는 기본 설정인 **SET BOX PAD**와 동일합니다.

또한, 버퍼 모드는 삭제 보류를 사용한 상자 삭제 및 삭제된 상자 복원에도 영향을 줍니다.

텍스트 찾기 및 바꾸기

EVE 명령을 사용하면 버퍼의 특정 텍스트를 검색할 수 있습니다. 특정 텍스트의 모든 경우를 검색할 수 있을 뿐만 아니라 단일 행에 있거나 여러 행에 걸쳐 있는 텍스트를 검색할 수 있습니다. 또한, 와일드카드를 사용하여 텍스트를 검색할 수도 있습니다. 이 절에서는 텍스트를 검색 및 바꾸는 방법에 대해 설명합니다.

표 8-12에는 버퍼에서 텍스트를 검색하는 EVE 명령이 설명되어 있습니다.

표 8-12

버퍼에서 텍스트를 찾기 위한 EVE 명령

명령	기능
FIND	현재 버퍼에서 지정된 텍스트 문자열을 검색하여 찾은 텍스트를 강조합니다. 강조된 텍스트를 검색 범위라고 부릅니다.
FIND NEXT	FIND, REPLACE 또는 WILDCARD FIND 명령을 사용하여 마지막으로 지정한 텍스트 문자열을 검색합니다.
FIND SELECTED	입력된 문자열이 아니라 선택된 텍스트 문자열을 검색합니다. 문자열을 여러 행에 걸쳐 선택할 수는 없습니다.
SET FIND CASE EXACT	대/소문자 구분 검색을 활성화합니다. 소문자로 된 문자열만 검색하여 바꾸는 경우에 특히 유용합니다.
SET FIND CASE NOEXACT	기본 설정입니다. 대/소문자 구분 검색을 비활성화합니다. 검색 문자열을 모두 소문자로 입력하면 대/소문자 구분 없이 모든 경우가 검색됩니다.
SET FIND NOWHITESPACE	기본 설정입니다. 검색 문자열에 지정된 탭과 공백이 정확히 일치하고, 단일 행에 있는 문자열만 검색하도록 FIND 및 WILDCARD FIND 명령을 설정합니다.
SET FIND WHITESPACE	공백, 탭 및 최대 하나까지의 줄 바꿈을 “공백”으로 간주하도록 FIND 및 WILDCARD FIND 명령을 설정합니다. 이렇게 하면 단어가 분리된 방식에 상관 없이 둘 이상의 단어로 된 문자열을 검색할 수 있습니다.

표 8-12

버퍼에서 텍스트를 찾기 위한 EVE 명령(계속)

명령	기능
SET WILDCARD VMS	OpenVMS의 기본 설정입니다. WILDCARD FIND에 사용되도록 OpenVMS 패턴을 활성화합니다.
SHOW WILDCARDS	WILDCARD FIND에서 사용할 수 있는 와일드카드 패턴을 나열합니다.
WILDCARD FIND	와일드카드를 사용하여 텍스트 패턴을 검색합니다.

텍스트 찾기

FIND 명령을 사용하면 현재 버퍼에서 특정 텍스트를 찾을 수 있습니다. 기본적으로 EVE는 E1 키(VT200, VT300 및 VT400 계열 터미널의 Find 키 및 VT100 계열 터미널의 PF1 키)를 FIND 명령으로 정의합니다.

검색 문자열이 모두 소문자로 되어 있으면 EVE는 대/소문자를 무시하고 모든 경우를 검색합니다. 따라서 검색 문자열 *the*를 입력하면 *the*, *THE*, *ThE*, *thE* 등이 검색됩니다. 검색 문자열에 하나 이상의 대문자가 들어 있으면, EVE는 각 문자의 대/소문자가 정확하게 일치하는 경우만 검색합니다. 따라서, 검색 문자열 *tHis*를 입력하면 *tHis*만 검색됩니다. 예를 들면 다음과 같습니다.

1. FIND 명령을 입력합니다.
2. 찾으려는 텍스트, 즉 검색 문자열을 입력합니다.

버퍼의 현재 방향에 따라 EVE의 처음 검색 방향(정방향 또는 역방향)이 결정됩니다.

현재 방향에 문자열이 없지만 반대 방향에서 문자열을 찾은 경우, EVE는 방향을 변경하라는 메시지를 표시합니다.

반대 방향으로 검색하려면 YES(Y)를 입력하고 Enter 키를 누릅니다. EVE는 반대 방향에서 처음 발견된 문자열로 커서를 이동합니다. 단, 강조된 상태 표시줄의 현재 방향은 바뀌지 않습니다.

검색 문자열을 찾은 경우

검색 문자열을 찾으면 EVE는 해당 문자열을 강조하고 커서를 문자열의 첫 문자로 이동합니다. 강조된 검색 문자열에서 사용할 수 있는 편집 명령의 목록을 보려면 *Extensible Versatile Editor Reference Manual*을 참조하십시오.

강조를 취소하려면 커서를 검색 문자열 밖으로 이동하거나 **RESET** 명령을 사용합니다.

검색 문자열의 다음 경우를 찾으려면 **Find** 키를 두 번 누르거나 **FIND NEXT** 명령을 입력합니다.

대/소문자 구분 검색 설정

소문자로 된 문자열을 검색할 때 대/소문자를 구분하여 검색하려면 **SET FIND CASE EXACT** 명령을 입력합니다. 이 명령을 사용하면 EVE는 모두 소문자로 된 검색 문자열을 검색할 때 대문자가 들어 있는 경우를 무시하고 소문자로 된 문자열만 검색합니다.

이 설정은 **FIND**, **REPLACE** 및 **WILDCARD FIND** 명령에 적용됩니다. 이후의 편집 세션에 사용할 수 있도록 설정을 섹션 파일이나 명령 파일에 저장할 수 있습니다. 기본 설정은 **SET FIND CASE NOEXACT**입니다.

EVE는 분음(악센트) 기호를 구분하므로 분음 기호가 정확히 일치하는 문자열만 검색합니다. 예를 들어, **ë**를 검색 문자열로 입력하면 **e**, **é**, **è** 또는 **ê**는 검색되지 않습니다.

다음 보기의 명령은 대/소문자 구분 검색을 활성화한 다음, **digital**을 소문자로 된 경우만 검색합니다. 즉, **Digital**, **DIGITAL** 등은 검색하지 않습니다.

Command: SET FIND CASE EXACT

Command: FIND digital

연습: 텍스트 찾기

기존의 **RHYMES.DAT** 파일에서 **FIND** 명령을 사용하려면 다음 단계를 따릅니다.

1. EVE를 실행하여 **RHYMES.DAT**를 엽니다. 버퍼의 첫 번째 행의 첫 번째 문자에 커서가 나타나고, 현재 방향은 정방향입니다.
2. **Find** 키를 누르고 문자열 **ree**를 입력한 다음, **Enter** 키를 누릅니다. 커서가 **tree**의 문자 **r**로 이동하고 문자열 **ree**가 강조됩니다.
3. **Find** 키를 두 번 눌러 **ree** 문자열의 다음 경우를 찾습니다. 커서가 **three**의 문자 **r**로 이동하고 문자열 **ree**가 강조됩니다.

검색 문자열이 검색 및 강조되면 선택 또는 검색 범위에서 작동하는 모든 명령 (SPELL 제외)을 사용할 수 있습니다. 단, 검색 범위에서는 삭제 보류 작업을 사용할 수 없습니다.

4. UPPERCASE WORD 명령을 입력합니다.

UPPERCASE WORD 명령은 다음 보기에 나온 것처럼 강조된 문자를 소문자에서 대문자로 변경합니다.

```
She rhymes with tree,  
also with bee,  
and this one makes thREE.  
[End of file]
```

연습: FIND SELECTED 명령 사용

특히 복잡하거나, 철자를 틀리기 쉽거나, 잘못 입력하기 쉬운 문자열을 검색하기 위해 FIND SELECTED를 사용하려면 다음 단계를 따릅니다.

1. 이전 연습에서 텍스트가 버퍼에 두 번 표시되도록 텍스트를 복사합니다.
2. 커서를 첫 번째 행에 있는 *rhymes with tree*, 문자열의 시작 부분으로 이동합니다.
3. SELECT 명령을 입력합니다.
4. 커서를 이동하여 문자열을 강조하고 텍스트를 선택합니다. 문자열을 여러 행에 걸쳐 선택할 수는 없습니다.
5. FIND SELECTED 명령을 입력합니다.

커서가 *rhymes with tree*, 문자열의 다음 경우로 이동합니다. 선택이 취소되고 검색된 문자열이 굵은 비디오로 표시됩니다.

와일드카드 사용

와일드카드를 사용하여 텍스트를 검색할 수 있습니다. 현재 와일드카드 설정에 맞는 와일드카드 패턴을 표시하려면 **SHOW WILDCARDS** 명령을 입력합니다.

연습: 와일드카드 사용

와일드카드 사용 방법을 배우려면 다음 단계를 따릅니다.

1. 커서를 버퍼의 시작 부분으로 이동합니다.
2. **WILDCARD FIND *ee** 명령을 입력하여 **ee**로 끝나는 텍스트 문자열을 검색합니다.

```
She rhymes with tree,  
also with bee,  
and this one makes thREE.  
[End of file]
```

EVE는 *tree*의 *r*이 포함된 행의 시작 부분에 커서를 놓습니다.

검색에 공백 포함

SET FIND WHITESPACE 및 **SET FIND NOWHITESPACE** 명령을 사용하면 **WILDCARD FIND** 및 **FIND** 명령이 단어 사이의 공백, 탭, 줄 바꿈 등을 처리하는 방법을 지정할 수 있습니다.

기본 검색 동작인 **SET FIND NOWHITESPACE** 명령은 지정된 탭과 공백이 정확히 일치하는 경우에 한하여 단일 행에 있는 여러 단어로 된 문자열을 검색하도록 **WILDCARD FIND** 및 **FIND** 명령을 설정합니다.

SET FIND WHITESPACE 명령은 단어가 분리된 방식에 상관 없이 여러 단어로 된 문자열을 검색하도록 **WILDCARD FIND** 및 **FIND** 명령을 설정합니다. 이렇게 하면 단어 사이에 여러 개의 공백 또는 탭과 최대 하나까지의 줄 바꿈이 포함된 문자열을 검색할 수 있습니다.

텍스트에서 위치 표시

MARK 및 GO TO 명령은 큰 파일을 편집한 다음, 현재 편집 세션에서 나중에 특정 위치로 되돌아가려는 경우에 유용합니다. 다음 표에는 MARK 및 GO TO 명령이 설명되어 있습니다.

표 8-13

명령	기능
MARK	현재 커서 위치에 보이지 않는 표시를 놓습니다. 이 표시는 현재 편집 세션이 끝나거나 사용자가 표시를 변경하기 전까지 존재하며, 종료 시 저장되지 않습니다.
GO TO	MARK 명령에 의해 레이블이 지정된 위치로 커서를 되돌립니다. 레이블이 지정된 위치가 다른 버퍼에 있을 경우, EVE는 커서를 해당 버퍼로 이동하고 커서가 있는 버퍼를 현재 창에 표시합니다.

현재 위치를 표시하려면 MARK 명령과 원하는 레이블 이름을 입력합니다. 레이블 이름은 영숫자 및 문장 부호 문자, 공백, 탭 문자를 비롯하여 하나 이상의 인쇄 가능한 문자가 될 수 있습니다. 커서를 표시된 위치로 되돌리려면 GO TO 명령과 레이블 이름을 차례로 입력합니다.

텍스트 바꾸기

REPLACE 명령을 사용하면 현재 버퍼의 텍스트 문자열을 다른 텍스트 문자열로 바꿀 수 있습니다. 이 방법은 긴 파일에서 단어의 철자를 잘못 입력했거나 철자가 잘못된 특정 단어의 모든 경우를 수정하려는 경우에 유용합니다.

REPLACE 명령 및 대/소문자 구분

REPLACE 명령은 대/소문자를 구분합니다. EVE는 이전 문자열에 대문자가 들어 있으면 대/소문자를 정확히 구분하여 검색하고, 이전 문자열이 모두 소문자로 되어 있으면 대/소문자에 상관 없이 문자열의 모든 경우를 검색합니다. 새 문자열에 대문자가 들어 있으면 검색된 문자열의 대/소문자에 상관 없이 모두 새 문자열로 바뀌고, 이전 문자열과 새 문자열이 모두 소문자이면 다음 규칙이 적용됩니다.

- 첫 문자가 대문자로 된 이전 문자열(첫 문자를 제외한 나머지 문자는 소문자)은 첫 문자가 대문자인 새 문자열로 바뀝니다.

- 모두 대문자로 된 이전 문자열은 모두 대문자로 된 새 문자열로 바뀝니다. 마찬가지로 모두 소문자로 된 이전 문자열은 모두 소문자로 된 새 문자열로 바뀝니다.

다음 표는 EVE에서 문자열의 대/소문자가 사용되는 방법을 보여 줍니다.

표 8-14

이전 문자열	새 문자열	강조되는 문자열	바뀌는 문자열
butter	margarine	butter	margarine
		Butter	Margarine
		BUTTER	MARGARINE
		BUtteR	margarine
Butter	margarine	Butter	margarine
butter	Margarine	butter	Margarine
		Butter	Margarine
		BUTTER	Margarine
		BUtteR	Margarine
Butter	Margarine	Butter	Margarine

소문자로 된 문자열만 검색 또는 바꾸려면 SET FIND CASE EXACT 명령을 입력합니다. 이 경우, 모두 소문자로 된 검색 문자열을 입력하면 대문자가 들어 있는 경우는 건너뛰고 소문자로 된 경우만 검색됩니다. 이 설정은 FIND, REPLACE 및 WILDCASE FIND 명령에 적용됩니다.

다음 표는 SET FIND CASE EXACT 명령을 입력한 경우 EVE가 소문자 문자열만 검색하여 바꾸는 것을 보여 줍니다.

표 8-15

이전 문자열	새 문자열	강조되는 문자열	바꾸는 문자열
butter	margarine	butter	margarine

기본 대/소문자 설정은 SET FIND CASE NOEXACT입니다.

REPLACE 명령 응답

다음 표는 REPLACE 명령 쿼리에 대한 응답과 그 효과를 보여 줍니다.

표 8-16

응답	효과
Yes	기본 응답이며, 현재 문자열을 바꾸고 다음 문자열을 찾습니다. Enter 키를 누릅니다.
No	현재 문자열을 건너뛰고 다음 문자열을 찾습니다.
All	검색되는 모든 문자열을 바꿉니다. 반대 방향으로 문자열이 검색되지 않는 한 프롬프트가 표시되지 않습니다.
Last	현재 문자열을 바꾸고 작업을 중지합니다.
Quit	현재 문자열을 건너뛰고 작업을 중지합니다.

명령행 한정자 사용

EVE를 실행할 때, 명령행 한정자를 사용하여 고급 EVE 편집 기능을 지정할 수 있습니다. 문자-셀 화면 업데이터를 사용하는 경우, 기본 삽입 또는 겹쳐쓰기 모드는 터미널 설정에 의해 결정됩니다.

표 8-17에는 EDIT 명령과 함께 EVE를 실행하는 데 사용할 수 있는 한정자가 나와 있습니다.

표 8-17

EDIT 명령행 한정자

한정자	기본값
명령 파일	/COMMAND=TPU\$COMMAND.TPU
파일 작성	/CREATE
패키지 디버깅	/NODEBUG
디스플레이 모드 지정	/DISPLAY=CHARACTER_CELL
초기화 파일	/INITIALIZATION=EVE\$INITEVE
저널링	/JOURNAL
Main 버퍼 수정	/MODIFY
출력 지정	/OUTPUT=output-file
읽기 전용 액세스	/NOREAD_ONLY
복구	/NORECOVER
섹션 파일	/SECTION=TPU\$SECTION
시작 위치	/START_POSITION=(1,1)
작업 파일	/WORK=SYS\$SCRATCH:TPU\$WORK.TPU\$WORK

다른 위치에서 시작

시작 위치 한정자는 명령행에 지정된 버퍼에 커서가 처음 표시되는 행과 열을 결정합니다.

EVE의 경우, 기본 시작 위치는 버퍼의 왼쪽 위 모서리를 나타내는 1,1(행 1, 열 1)입니다. 시작 위치 한정자는 편집 세션 도중에 만든 다른 버퍼의 초기 커서 위치에는 영향을 주지 않고, 버퍼 크기를 제한하지 않습니다.

시작 위치 한정자의 형식은 다음과 같습니다.

`/START_POSITION=(row[,column])`

각 필드의 내용은 다음과 같습니다.

표 8-18

<code>/START_POSITION</code>	EDIT 명령에는 <code>/START_POSITION=</code> 한정자를 사용해야 합니다.
<code>row</code>	EVE를 실행할 때 커서를 표시할 행입니다.
<code>column</code>	EVE를 실행할 때 커서를 표시할 열입니다.

시작 위치 한정자를 사용하면 특정 행(또는 줄)이나 특정 문자 위치(또는 열)에서 편집을 시작할 수 있습니다. 예를 들어, 파일의 표준 머리글을 건너뛰려는 경우나 프로그램의 특정 행에 오류가 있다는 것을 일괄 처리 로그 파일 또는 오류 메시지를 통해 확인한 경우, 해당 행 번호를 시작 행으로 지정하여 프로그램 소스 파일을 편집할 때 커서를 해당 행으로 바로 이동할 수 있습니다. 다음 명령은 `test.com`이라는 파일을 열 때 커서를 행 10, 열 5로 이동합니다.

```
$ EDIT TEST.COM /START_POSITION=(10,5)
```

파일의 특정 행에서 시작하려면 두 번째 매개 변수(열)를 생략합니다.

작업 파일 사용

작업 파일 한정자는 매우 큰 파일을 편집하기 위해 메모리를 스왑하는 데 사용되는 작업 파일을 결정합니다. 편집 세션 당 하나씩 존재하는 작업 파일은 종료 시 자동으로 삭제되는 임시 파일입니다.

기본 작업 파일은 `TPU$WORK.TPU$WORK`입니다. 별도로 지정하지 않으면 EVE는 `SY$SCRATCH`에 작업 파일을 만듭니다.

다음과 같은 두 가지 방법으로 다른 작업 파일을 지정할 수 있습니다.

- **TPU\$WORK** 논리 이름을 정의합니다.

이 방법은 **SYS\$SCRATCH**가 아닌 다른 영역(예: 더 큰 디스크)에서 작업 파일을 만들려는 경우에 유용합니다. 정의된 이름을 **LOGIN.COM** 파일에 포함할 수 있습니다.

- **/WORK=** 한정자를 사용하여 작업 파일을 지정합니다.

이 방법은 모든 **TPU\$WORK** 논리 이름 정의를 무시합니다. 예를 들어, 다음 명령은 **EVE**를 실행하고 작업 파일이 **SYS\$SCRATCH:MYWORK.TPU\$WORK**가 되도록 지정합니다.

```
$ EDIT /WORK=MYWORK
```

SYS\$SCRATCH가 아닌 영역에 작업 파일을 만들려면 장치(디스크) 및 디렉토리가 포함된 완전한 파일 지정 방법을 사용합니다. 와일드카드를 사용하여 작업 파일을 지정할 수는 없습니다.

Main 버퍼 수정

수정 한정자는 명령행에 지정된 버퍼를 수정할 수 있는지 여부를 결정합니다. 편집 세션 도중에 만든 다른 버퍼는 이 한정자의 영향을 받지 않습니다.

기본적으로 버퍼의 텍스트를 편집하여 버퍼를 수정할 수 있습니다. 버퍼를 수정하고 **EVE**를 종료하면 해당 버퍼가 파일에 기록됩니다.

파일을 변경 없이 검사만 하려면 **/NOMODIFY**를 사용합니다. 이 경우, 커서 이동 명령을 사용할 수 있지만 텍스트를 변경할 수는 없습니다.

/MODIFY 또는 **/NOMODIFY**를 지정하지 않으면 버퍼를 수정할 수 있는지 여부가 어플리케이션에 의해 결정됩니다. **EVE**의 기본 동작은 버퍼를 수정하는 것입니다.

/READ_ONLY 또는 **/NOWRITE**의 효과를 무시하려면 **/MODIFY**를 사용합니다.

/READ_ONLY 또는 **/NOWRITE**와 함께 **/MODIFY**를 사용하면 파일을 기록하지 않으면서 편집 작업을 연습할 수 있습니다. 예를 들어, 다음 명령은 **EVE**를 실행하여 명령행에 지정된 버퍼를 읽기 전용(또는 쓰기 금지)로 만든 다음, 수정 가능하게 만듭니다.

```
$ EDIT /READ_ONLY /MODIFY
```

EVE에서는 **SET BUFFER** 명령을 사용하여 버퍼의 수정 속성을 설정 또는 변경할 수 있습니다.

EVE를 실행하는 다른 방법

검색 목록, 와일드카드, 와일드카드 디렉토리 이름 또는 여러 입력 파일의 네 가지 다른 방법을 사용하여 EVE를 실행할 수 있습니다.

검색 목록에서 EVE 실행

검색 목록을 사용하여 EVE를 실행하여 해당 검색 목록의 파일을 편집할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ DEFINE STAFFMEMOS HIRING.DAT,PROMOTION.LIS,SALARY.TXT
$ EDIT STAFFMEMOS
```

이 보기에서 검색 목록의 첫 번째 파일이 존재하면 EVE는 해당 파일(HIRING.DAT)을 버퍼에 복사하고 해당 파일의 이름과 형식을 버퍼 이름으로 사용합니다. 첫 번째 파일이 존재하지 않으면 두 번째 파일(PROMOTION.LIS)의 검색을 시도합니다. 검색 목록의 모든 파일이 존재하지 않으면 EVE는 검색 목록에 있는 첫 번째 파일의 이름을 사용하여 HIRING.DAT라는 빈 버퍼를 만듭니다.

와일드카드를 사용하여 EVE 실행

EVE를 실행하여 기존 파일을 편집할 때, 파일 이름과 파일 형식의 일부 또는 전체 문자를 대체하는 별표(*) 와일드카드 문자를 사용할 수 있습니다. EVE에서 와일드카드를 사용하려면 DCL에서 와일드카드를 사용할 때와 동일한 규칙을 따라야 합니다. 백분율 기호(%) 와일드카드 문자로 문자 하나를 대체할 수 있고, 줄임표(...) 와일드카드 문자로 디렉토리 지정 방법을 대체할 수 있습니다. 일치하는 파일이 한 개이면 해당 파일이 화면에 표시되고, 일치하는 파일이 여러 개이면 파일 목록이 표시된 다음 더 완전한 파일 지정 방법을 제공하라는 메시지가 표시됩니다. 일치하는 파일이 존재하지 않으면 Main이라는 버퍼가 만들어집니다.

와일드카드 요청과 일치하는 파일이 여러 개이면, 원하는 파일을 선택할 수 있도록 일치하는 파일이 모두 표시됩니다.

일치하는 파일이 검색되지 않으면 Main이라는 빈 버퍼가 만들어집니다. 검색 목록 또는 와일드카드 디렉토리를 사용하여 입력 파일을 지정하면 EVE는 \$CHOICES\$ 버퍼를 표시하지 않고 일치하는 첫 번째 파일을 사용합니다. \$CHOICES\$ 버퍼 사용에 대한 자세한 내용은 EVE 온라인 도움말에서 Choices Buffer 항목을 참조하십시오.

다음 보기는 파일 형식이 **.TXT**인 모든 파일의 목록을 표시합니다.

```
$ EDIT *.TXT
```

위 보기와 같이 ***.TXT**가 지정되면 **EVE**는 와일드카드 요청과 일치하는 파일을 다른 창에 있는 **\$CHOICES\$**라는 시스템 버퍼에 나열합니다.

와일드카드 디렉토리 이름을 사용하여 EVE 실행

와일드카드 디렉토리 이름(...)을 사용하여 **EVE**를 실행하고 현재 디렉토리나 현재 디렉토리의 하위 디렉토리에서 작업할 수 있습니다.

이러한 검색 목록 또는 와일드카드 디렉토리 처리 방법은 **EDIT** 명령 뿐만 아니라 파일 지정 방법을 매개 변수로 사용하는 **EVE** 명령에도 적용됩니다. 파일 지정 방법을 매개 변수로 사용하는 **EVE** 명령은 다음과 같습니다.

콜뱅이 기호(@)

GET FILE

INCLUDE FILE

OPEN

OPEN SELECTED

RECOVER BUFFER

다음 보기는 디렉토리 트리를 검색하여 첫 번째로 발견된 **PINK.TXT** 파일(있을 경우)을 가져옵니다.

```
$ EDIT [...]PINK.TXT
```

여러 입력 파일을 사용하여 EVE 실행

EVE를 실행하는 명령행에서 여러 개의 입력 파일을 지정할 수 있습니다. 파일 이름은 쉼표로 구분해야 하며, 선택적으로 공백이 포함될 수 있습니다. 파일 이름에 와일드카드 문자가 존재하면 일치하는 파일이 두 개 이상인 첫 번째 와일드카드 파일 이름에 대해서만 일치하는 파일이 표시됩니다. 다른 모호한 파일 이름에 대해서는 경고 메시지가 출력됩니다.

저널링 및 복구

저널 파일은 시스템 오류로 인해 편집 세션이 중단되었을 때 작업을 복구할 수 있도록 편집 내용을 기록합니다.

기본적으로 **EVE**에서 버퍼 변경 저널링은 작성된 각 텍스트 버퍼에 대해 별도의 저널 파일을 만듭니다. 버퍼 변경 저널링은 **DECwindows** 및 문자-셀 터미널에서 모두 작동합니다. 일반적으로 **EVE**에서 **RECOVER BUFFER** 명령을 사용하여 버퍼를 한 번에 하나씩 복구 하며, 필요한 경우 다른 편집 세션의 버퍼를 복구할 수 있습니다. 이러한 복구 작업은 텍스트 만 복원합니다. 즉, 시스템 오류 이전의 설정, 키 정의, 시스템 버퍼(예: **Insert Here** 버퍼) 내용 등은 복원되지 않습니다.

EVE를 실행할 때 명령행에서 **/NOJOURNAL** 한정자를 사용하여 저널링을 비활성화할 수 있습니다. 이는 **EVE**를 사용하여 파일을 편집 없이 검사만 하거나 **EVE**를 데모 세션에 사용하는 경우에 유용합니다.

EVE 파일 백업은 비활성화됩니다. **OpenVMS** 파일 시스템이 버전 번호를 제공하고, 결과적으로 **EVE** 메커니즘이 필요하지 않기 때문에 **EVE** 파일 백업은 활성화할 수 없습니다.

버퍼 변경 저널링 사용

버퍼 변경 저널링은 각 텍스트 버퍼에 대해 저널 파일을 만듭니다. **Insert Here** 버퍼, **DCL** 버퍼 또는 **\$RESTORE\$** 버퍼와 같은 시스템 버퍼에 대해서는 버퍼 변경 저널 파일이 만들어지지 않습니다. 버퍼를 편집할 때, 저널 파일은 텍스트 삭제, 삽입 또는 재형식화와 같은 모든 변경 사항을 기록합니다. **EVE**를 종료하거나 버퍼를 삭제하면 저널 파일은 삭제됩니다. 시스템 오류로 인해 편집 세션이 중단되면 저널 파일이 저장됩니다. 경우에 따라, 시스템 오류 이전에 마지막으로 수행한 일부 키 입력이 손실될 수 있습니다.

표 8-19에는 버퍼 변경 저널링 및 복구를 위한 **EVE** 명령이 요약되어 있습니다.

표 8-19

버퍼 변경 저널링 및 복구를 위한 **EVE** 명령

명령	기능 또는 효과
RECOVER BUFFER	버퍼의 저널 파일을 사용하여 지정된 버퍼를 복구합니다. 복구할 버퍼 또는 파일 이름을 지정하거나 버퍼의 저널 파일 이름을 지정할 수 있습니다.
RECOVER BUFFER ALL	버퍼의 저널 파일을 사용하여 모든 텍스트 버퍼를 한 번에 하나씩 복구합니다.
SET JOURNALING	지정된 버퍼에 대해 버퍼 변경 저널링을 활성화합니다.
SET JOURNALING ALL	모든 버퍼에 대해 버퍼 변경 저널링을 활성화합니다. 기본 설정입니다.
SET NOJOURNALING	지정된 버퍼에 대해 버퍼 변경 저널링을 비활성화합니다.
SET NOJOURNALING ALL	모든 버퍼에 대해 버퍼 변경 저널링을 비활성화합니다.

버퍼 변경 저널 파일은 **TPU\$JOURNAL** 논리 이름에 의해 정의된 디렉토리에 기록됩니다. 기본적으로 이 디렉토리는 대개 최상위(로그인) 디렉토리인 **SYS\$SCRATCH**입니다. **TPU\$JOURNAL** 논리 이름을 다시 정의하여 저널 파일을 다른 디렉토리에 기록할 수 있습니다. 예를 들어, 다음 명령은 **[USER.JOURNAL]**이라는 하위 디렉토리를 만든 다음, **TPU\$JOURNAL**을 이 하위 디렉토리로 정의합니다.

```
$ CREATE/DIRECTORY [USER.JOURNAL]
$ DEFINE TPU$JOURNAL [USER.JOURNAL]
```

이러한 정의를 **LOGIN.COM** 파일에 포함할 수 있습니다.

버퍼 변경 저널 파일은 용량이 커질 수 있는데, 경우에 따라서는 편집 중인 텍스트 파일보다 커질 수 있습니다. 버퍼 변경 저널 파일이 이러한 크기 문제를 갖고 있고 각 텍스트 버퍼에 대해 저널 파일이 존재하기 때문에, **SYS\$SCRATCH**가 아니라 큰 디스크에서 **TPU\$JOURNAL**을 디렉토리 또는 하위 디렉토리로 정의할 수 있습니다.

버퍼 변경 저널 이름 파생

버퍼 변경 저널 파일 이름은 편집 중인 파일 또는 버퍼의 이름과 운영 체제의 기본 파일 형식에서 파생됩니다. 현재 버퍼의 저널 파일 이름을 확인하려면 **EVE** 프롬프트에서 **SHOW** 명령을 입력합니다. **SHOW** 명령은 입력 파일, 출력 파일 및 저널 파일의 이름과 현재 버퍼에 대한 기타 정보를 표시합니다.

표 8-20은 버퍼 변경 저널 파일 이름을 보여 줍니다.

표 8-20

버퍼 변경 저널 파일 이름

텍스트 버퍼 이름	버퍼 변경 저널 파일
JABBER.TXT	JABBER.TXT.TPU\$JOURNAL
GUMBO_RECIPE.RNO	GUMBO_RECIPE_RNO.TPU\$JOURNAL
MAIN	MAIN.TPU\$JOURNAL
LATEST NEWS	LATEST_NEWS.TPU\$JOURNAL

버퍼 변경 저널링을 사용하여 편집 내용 복구

다음 두 가지 방법으로 버퍼 변경 저널 파일을 사용하여 편집 내용을 복구할 수 있습니다.

- EVE를 실행할 때 **EDIT** 명령행에서 **/RECOVER** 한정자를 사용합니다.
- EVE에서 **RECOVER BUFFER** 명령을 사용합니다.

다음 보기는 **JABBER.TXT**라는 파일을 편집하는 도중에 시스템 오류로 인해 편집 세션이 중단되었는데, 이 때 시스템 복구를 사용하여 편집 내용을 복구하는 방법을 보여 줍니다.

```
$ EDIT JABBER.TXT
.
.
.
*** system failure ***
.
.
.
$ EDIT JABBER.TXT/RECOVER
```

RECOVER BUFFER 명령 사용

RECOVER BUFFER 명령을 사용하려면 다음 절차를 따릅니다.

표 8-21

단계	작업
1	EVE를 실행하고 다음 명령을 입력하여 텍스트를 복구합니다. Command: RECOVER BUFFER <i>file-name.txt</i> 버퍼 변경 저널 파일을 사용할 수 있으면 EVE는 다음 정보를 표시하고 해당 버퍼를 복구할 것인지 묻습니다. • 버퍼 이름 • 버퍼의 원래 입력 파일(있을 경우) • 버퍼의 출력 파일(있을 경우) • 복구를 위한 소스 파일(있을 경우) • 편집 세션의 시작 날짜 및 시간 • 저널 파일 작성 날짜 및 시간
2	버퍼를 복구하려면 Enter 키를 누릅니다. 버퍼를 복구하지 않으려면 No를 입력하고 Enter 키를 누릅니다. 복구에 필요한 소스 파일을 삭제하거나 이름을 변경하면 복구는 오류가 발생합니다. 소스 파일은 버퍼가 있는 경우 초기에 읽는 파일이거나 시스템에 오류가 발생하기 전에 쓰여진 마지막 파일입니다. 기존의 버퍼를 복구하려면(일반적으로 Main 버퍼), EVE는 처음에 버퍼를 삭제한 뒤 복구를 수행합니다. 수정된 버퍼를 복구하려면 EVE는 복구하기 전에 버퍼를 삭제할 것인지의 여부를 확인합니다.

파일 이름을 모를 경우의 복구 방법

버퍼 이름이나 저널 파일 이름이 확실하지 않을 경우 다음과 같이 별표(*) 와일드카드를 지정합니다.

Command: RECOVER BUFFER *

이렇게 하면 사용 가능한 모든 저널 파일이 목록으로 표시되므로 원하는 저널 파일을 선택할 수 있습니다. 이 목록은 다른 창에 있는 \$CHOICES\$라는 EVE 시스템 버퍼에 표시됩니다. \$CHOICES\$ 버퍼 사용에 대한 자세한 내용은 EVE 온라인 도움말에서 Choices Buffer 항목을 참조하십시오.

모든 버퍼 복구 방법

모든 텍스트 버퍼를 한 번에 하나씩 모두 복구하려면 RECOVER BUFFER ALL 명령을 사용합니다. 이 명령을 사용하면 EVE는 버퍼 변경 저널을 사용할 수 있는 각 텍스트 버퍼의 복구를 시도합니다. 이는 버퍼 이름이나 저널 파일 이름을 지정할 필요 없이 RECOVER BUFFER 명령을 반복하는 것과 동일합니다. 각 텍스트 버퍼에 대해 버퍼 이름, 버퍼와 연결된 파일, 저널 파일 작성 날짜 및 시간 등의 정보가 표시됩니다. EVE는 다음 중 하나를 선택하라는 메시지를 표시합니다.

표 8-22

응답	효과
Yes	기본 응답이며, 버퍼를 복구하고 다음 버퍼(있을 경우)를 복구할 것인지 여부를 묻습니다. Enter 키를 누릅니다.
No	현재 복구를 건너뛵니다. 복구할 다른 버퍼가 있으면 이에 대한 복구 여부를 묻습니다.
Quit	복구를 취소합니다. 즉, 버퍼를 복구하지 않고 복구 작업을 중단합니다.

버퍼 변경 저널링 비활성화

SET NOJOURNALING 명령을 사용하여 특정 버퍼에 대한 버퍼 변경 저널링을 비활성화할 수 있습니다. 모든 버퍼에 대해 버퍼 변경 저널링을 비활성화하려면 SET NOJOURNALING ALL 명령을 사용합니다.

버퍼 변경 저널링 활성화

버퍼 변경 저널링을 비활성화한 경우, **SET JOURNALING** 명령을 사용하여 저널링을 활성화할 수 있습니다. 다음 명령은 **JABBER.TXT**라는 버퍼에 대해 저널링을 활성화합니다.

```
Command: SET JOURNALING JABBER.TXT
```

저널링 없이 **EVE**를 실행한 후에 편집 세션 도중 버퍼 변경 저널링을 활성화하려면, **EVE** 기본값인 **SET JOURNALING ALL** 명령을 사용합니다.

버퍼가 수정된 경우에는 버퍼 변경 저널링을 활성화할 수 없습니다. 이러한 경우, **EVE**는 다음 메시지를 표시합니다.

```
Command: SET JOURNALING MEMO.TXT
```

```
Buffer MEMO.TXT is not safe for journaling
```

따라서, **WRITE FILE** 또는 **SAVE FILE** 명령을 사용하여 버퍼를 기록(저장)한 다음 저널링을 활성화해야 합니다.

EVE 형식화 명령

EVE는 여백, 탭 및 자동 줄 바꿈을 설정하여 텍스트를 형식화할 수 있는 명령을 제공합니다. 행을 가운데로 맞추고, 텍스트에서 불필요한 공백을 제거하며, 페이지 구분을 삽입할 수 있습니다.

표 8-23에는 EVE 편집 키와 그 기능이 설명되어 있습니다.

표 8-23

EVE 편집 키 및 기능

키 또는 키 시퀀스	기능
Return 또는 Ctrl/M	새 행을 시작하거나 명령 입력을 종료하기 위해 현재 위치에 캐리지 리턴을 삽입합니다. VT200, VT300 및 VT400 계열 터미널에서는 Enter 키도 Return으로 정의됩니다.
Tab 또는 Ctrl/I	현재 위치(탭 모드에 따라)와 현재 설정된 탭 정지에 탭 문자를 삽입합니다.
Ctrl/L	새 페이지의 시작 부분을 표시하기 위해 현재 위치에 form-feed 문자를 삽입합니다. 항상 행 위에 단독으로 나타나는 작은 이중 F(F F)가 페이지 구분으로 표시됩니다. INSERT PAGE BREAK와 동일합니다.

표 8-24에는 EVE 텍스트 형식화 명령과 그 기능이 설명되어 있습니다.

표 8-24

EVE 텍스트 형식화 명령 및 기능

명령	기능
CAPITALIZE WORD	단어의 대/소문자를 변경하여 첫 문자를 대문자로 만들고 나머지 문자를 소문자로 만듭니다. 범위, 상자 또는 단어 하나에 적용됩니다.
CENTER LINE	현재 행을 왼쪽 및 오른쪽 여백 사이의 가운데에 놓습니다. 커서는 행과 함께 이동하여 이전 위치를 그대로 유지합니다.
CONVERT TABS	상자, 범위 또는 전체 버퍼에서 탭 문자를 해당되는 개수의 공백으로 변환합니다.

표 8-24

EVE 텍스트 형식화 명령 및 기능(계속)

명령	기능
FILL	최대한의 단어가 행에 들어가도록 버퍼의 여백에 따라 현재 단락, 범위 또는 상자를 다시 형식화합니다. 선택 범위나 검색 범위를 채울 경우, FILL 또는 FILL RANGE 명령은 범위에서 페이지 구분, DSR(DIGITAL Standard Runoff) 명령 또는 DOCUMENT 태그로 시작되는 행을 제외한 다른 행들을 다시 형식화합니다. 이러한 범위 채우기로 인해 빈 행이 삭제되지는 않습니다. 선택 범위에 대한 자세한 내용은 279페이지의 “텍스트 이동”을 참조하십시오.
FILL PARAGRAPH	버퍼에 설정된 여백에 따라 커서가 위치한 단락을 다시 형식화합니다. 단락을 채울 경우, FILL 명령은 단락에서 페이지 구분, DSR 명령 또는 DOCUMENT 태그로 시작되는 행을 제외한 다른 행들을 다시 형식화합니다.
FILL RANGE	현재 여백 설정에 따라 범위나 상자를 다시 형식화합니다. 선택 범위나 검색 범위를 채울 경우, FILL 또는 FILL RANGE 명령은 범위에서 페이지 구분, DSR 명령 또는 DOCUMENT 태그로 시작되는 행을 제외한 다른 행들을 다시 형식화합니다. 이러한 범위 채우기로 인해 빈 행이 삭제되지는 않습니다.
INSERT PAGE BREAK	새 페이지의 시작 부분을 표시하기 위해 현재 위치에 form-feed 문자를 삽입합니다. 항상 행 위에 단독으로 나타나는 작은 이중 F(F F)가 페이지 구분으로 표시됩니다. 기본적으로 Ctrl/L이 INSERT PAGE BREAK로 정의됩니다.
LOWERCASE WORD	현재 단어, 범위 또는 상자를 소문자로 변경합니다.

표 8-24

EVE 텍스트 형식화 명령 및 기능(계속)

명령	기능
PAGINATE	54행 페이지를 위한 “소프트” 페이지 구분을 삽입합니다. 소프트 페이지 구분은 form-feed 다음 null 문자가 오는 형태(F FN L)로 표시됩니다. PAGINATE 명령을 입력하면 EVE 는 이전 페이지 구분(있는 경우)으로 이동한 후 다음 54행 내에서 페이지 구분이 있는지 검사합니다. 54행 내에서 소프트 페이지 구분이 발견되면 EVE 는 해당 페이지 구분을 제거합니다. 그런 다음, 54행 아래로 내려와 소프트 페이지 구문을 삽입하고 커서를 다음 행에 놓습니다. 소프트 페이지 구분은 단일 행으로 삽입됩니다. 54행 내에서 하드 페이지 구분(form-feed 에만 해당)이 발견되면 EVE 는 하드 페이지 구분을 지울 수 있도록 페이지 구분 다음 행에 커서를 놓습니다.
SET LEFT MARGIN	현재 버퍼의 왼쪽 여백을 설정합니다. 왼쪽 여백은 0보다 커야 하지만 오른쪽 여백보다 작아야 합니다. 왼쪽 여백의 기본값은 1(맨 왼쪽 열)입니다.
SET RIGHT MARGIN	현재 버퍼의 오른쪽 여백을 설정합니다. 오른쪽 여백은 왼쪽 여백보다 커야 합니다. 기본적으로 오른쪽 여백은 너비에서 1을 뺀 값으로 설정됩니다. 너비가 일반적으로 80이므로 기본 여백은 대개 79입니다.
SET PARAGRAPH INDENT	작성하거나 다시 형식화하는 단락의 첫 번째 행에서 빼거나 더할 공백 수를 지정합니다. 기본값은 0(들여쓰기 없음)입니다.
SET TABS AT	지정된 열에서 탭 정지를 설정합니다. 열 번호는 오름차순이어야 하고 공백으로 구분해야 합니다. 기본적으로 탭 정지는 열 8개마다 설정됩니다. 이 명령은 터미널의 하드웨어 탭 설정에 영향을 주지 않습니다.
SET TABS EVERY	지정된 간격으로 탭 정지를 설정합니다. 기본적으로 탭 정지는 열 8개마다 설정됩니다. 이 명령은 터미널의 하드웨어 탭 설정에 영향을 주지 않습니다.

표 8-24

EVE 텍스트 형식화 명령 및 기능(계속)

명령	기능
SET TABS INSERT	기본 설정입니다. Tab 키를 눌렀을 때 현재 열에 탭 문자가 삽입되도록 탭 모드를 변경합니다. 커서와 텍스트는 다음 탭 정지로 이동합니다.
SET TABS MOVEMENT	Tab 키가 커서 이동 키가 되도록 탭 모드를 변경합니다. Tab 키를 누르면 커서가 다음 탭 정지로 이동하지만 탭 문자가 삽입되지 않습니다.
SET TABS SPACES	Tab 키를 눌렀을 때 탭 문자가 아니라 해당하는 개수의 공백이 삽입되도록 탭 모드를 변경합니다. 이전의 기존 탭 문자는 영향을 받지 않습니다.
SET TABS INVISIBLE	기본 설정입니다. 탭 문자를 공백으로 표시하여 화면에서 볼 수 없게 만듭니다.
SET TABS VISIBLE	탭 문자를 작은 H_T (가로 탭)로 표시하여 화면에서 볼 수 있게 만듭니다.
SET NOWRAP	버퍼의 오른쪽 여백에서 자동 줄 바꿈을 비활성화합니다. 새 행을 시작하려면 Enter 키를 누르거나 FILL 명령을 사용합니다.
SET WRAP	기본 설정입니다. 버퍼의 오른쪽 여백에서 자동 줄 바꿈을 활성화합니다. Enter 키를 누르거나 FILL 명령을 사용하지 않더라도 새 행이 시작됩니다.
UPPERCASE WORD	현재 단어, 범위 또는 상자를 대문자로 변경합니다.

버퍼 사용

버퍼는 편집 세션 도중에만 존재하는 저장 영역입니다. 기존 파일을 편집할 때 해당 파일의 내용이 버퍼로 읽힙니다. 강조된 상태 표시줄에는 버퍼 이름, 편집 상태(읽기 전용 또는 쓰기), 편집 모드(삽입 또는 겹쳐쓰기), 방향(정방향 또는 역방향) 등이 표시됩니다.

표 8-25에는 버퍼 작성, 조작 및 삭제에 사용되는 **EVE** 명령이 설명되어 있습니다.

표 8-25

버퍼 조작을 위한 **EVE** 명령

명령	기능
BUFFER	지정된 버퍼를 현재 창에 표시하고 커서를 해당 버퍼에서 마지막으로 차지하던 위치로 이동합니다. 지정된 버퍼가 존재하지 않으면 새 버퍼를 만듭니다.
DELETE BUFFER	이름으로 지정된 버퍼를 삭제합니다.
GET FILE 또는 OPEN	지정된 파일을 현재 EVE 에 창에 표시하고, 필요할 경우 새 버퍼를 만듭니다. 파일이 존재하면 현재 창의 새 버퍼에 해당 파일이 복사되고, 파일이 존재하지 않으면 파일 이름과 파일 형식이 버퍼 이름으로 사용되어 빈 버퍼가 새로 만들어집니다. 해당 이름을 가진 버퍼가 이미 존재하면 다른 이름을 제공하라는 메시지가 표시됩니다.
GO TO	MARK 명령에 의해 레이블이 지정된 위치로 커서를 되돌립니다. 레이블이 지정된 위치가 다른 버퍼에 있을 경우, EVE 는 커서를 해당 버퍼로 이동하고 커서가 있는 버퍼를 현재 창에 표시합니다. 319페이지의 “여러 버퍼 편집”에는 편집 세션에서 여러 버퍼를 사용하는 방법이 설명되어 있습니다.
INCLUDE FILE	지정된 파일의 내용을 현재 버퍼의 커서 위치 바로 위에 삽입합니다. 이 명령은 파일을 결합하는 경우에 유용합니다.
NEW	Main 이라는 새 버퍼를 만들어 현재 창에 표시합니다. Main 버퍼가 이미 존재하면 새 버퍼의 이름을 제공하라는 메시지가 표시됩니다.

표 8-25

버퍼 조작을 위한 EVE 명령(계속)

명령	기능
NEXT BUFFER	다음 버퍼(있을 경우)를 현재 창에 표시하고 커서를 해당 버퍼에서 마지막으로 차지하던 위치로 이동합니다. 이 명령을 사용하면 버퍼 이름을 지정하지 않고 특정 버퍼에서 다른 버퍼로 이동할 수 있습니다.
OPEN SELECTED	이름이 선택 또는 검색된 파일을 엽니다. 이 명령은 파일 이름을 입력할 필요 없이 GET FILE 또는 OPEN 명령을 사용하는 것과 같습니다.
REMOVE 또는 CUT	Buffer List 버퍼에서는 DELETE BUFFER와 동일합니다. 버퍼 이름을 입력하지 않고 REMOVE 명령을 사용하여 버퍼를 삭제하려면, 우선 SHOW BUFFERS 명령을 입력하여 Buffer List 버퍼를 표시합니다. 그런 다음, 커서를 삭제할 버퍼 이름으로 이동하고 REMOVE 명령을 입력합니다.
SAVE FILE	편집 세션을 끝내지 않고 현재 버퍼의 내용을 버퍼와 연결된 파일에 기록합니다. 이 명령에서 파일 이름을 지정하지 않으면 출력 파일 지정 방법을 묻는 메시지가 표시됩니다. WRITE FILE과 비슷합니다.
SAVE FILE AS	편집 세션을 끝내지 않고 현재 버퍼의 내용을 지정된 파일에 기록합니다. 예를 들어, FIRST.DAT라는 파일을 편집하는 경우 이 파일을 SECOND.TXT로 저장할 수 있습니다. 이 명령은 버퍼 이름을 변경하지 않지만, 버퍼를 지정된 파일과 연결하여 이후에 SAVE FILE, WRITE FILE 또는 EXIT 명령을 사용할 때 버퍼가 지정된 파일에 기록되도록 합니다. 이 명령에서는 파일 지정 방법을 제공해야 합니다.
SELECT 또는 RETURN	Buffer List 버퍼에서 지정된 버퍼가 선택됩니다. 버퍼 이름을 입력하지 않고 SELECT 명령을 사용하여 버퍼를 선택하려면, 우선 SHOW BUFFERS 명령을 입력합니다. 그런 다음 커서를 선택할 버퍼 이름으로 이동하고 SELECT 명령을 입력합니다.

표 8-25

버퍼 조작을 위한 EVE 명령(계속)

명령	기능
SET BUFFER	버퍼의 편집 상태를 지정할 수 있게 합니다. 즉, 버퍼를 수정할 수 있는지 여부나 EVE 종료 시 버퍼를 파일에 기록할 수 있는지 여부를 이 명령으로 지정할 수 있습니다.
SHOW	편집 세션 도중에 작성된 버퍼에 대한 정보를 표시합니다. 편집 세션에서 여러 개의 버퍼가 활성화되어 있으면 이 명령은 현재 편집 중인 버퍼에 대한 정보를 표시합니다. 다른 활성 버퍼에 대한 정보를 보려면 Do 키를 누릅니다. 편집을 다시 시작하려면 다른 키를 아무거나 누릅니다.
SHOW BUFFERS	편집 세션 도중에 작성된 버퍼를 나열합니다. 목록에서 커서를 이동한 다음, Select 키를 눌러 보고자 하는 특정 버퍼를 지정할 수 있습니다.
SHOW DEFAULTS BUFFER	\$DEFAULTS\$라는 EVE 시스템 버퍼에 대한 정보(예: 여백, 탭 정지, 방향, 모드, 최대 행 수 등)를 표시합니다. 새 버퍼를 만들면 기본적으로 이러한 설정이 사용됩니다.
SHOW SYSTEM BUFFERS	Message, Help, Insert Here 및 \$RESTORE\$ 버퍼와 같이 EVE에 의해 작성되는 시스템 버퍼를 나열합니다. 목록에서 커서를 이동한 다음, Select 키를 눌러 보고자 하는 버퍼를 지정할 수 있습니다.
WRITE FILE	편집 세션을 끝내지 않고 현재 버퍼의 내용을 버퍼와 연결된 파일이나 명령행에 지정된 파일에 기록합니다. 현재 버퍼에 연결된 파일 지정 방법이 없을 경우, 출력 파일 지정 방법을 제공하라는 메시지가 표시됩니다. SAVE FILE과 비슷합니다.

버퍼 정보 보기

현재 버퍼에 대한 자세한 정보를 표시하려면 **SHOW** 명령을 입력합니다. 버퍼의 수정 여부를 비롯하여 다음 정보가 표시됩니다.

- 버퍼 이름
- 입력, 출력 및 버퍼 변경 지널 파일의 이름
- 현재 모드 및 방향
- 행 수
- 여백 및 화면 너비 설정
- 단락 들여쓰기
- WPS 자동 줄 바꿈
- 줄 바꿈 들여쓰기
- 탭 정지

편집 세션 동안 활성화된 버퍼가 여러 개이면 **EVE**는 **Do** 키를 눌러 다른 버퍼에 대한 정보를 확인하라는 메시지를 표시합니다.

버퍼 삭제

버퍼를 삭제하려면 **DELETE BUFFER** 명령을 입력하고 삭제할 버퍼의 이름을 지정합니다. 버퍼가 비어 있거나 수정되지 않았으면 **EVE**는 해당 버퍼를 삭제하고, 버퍼가 수정되었으면 선택 프롬프트를 표시합니다. 버퍼 이름은 완전하게 입력해야 하며, 약어는 사용할 수 없습니다. 삭제할 버퍼가 현재 보고 있는 버퍼이면 **EVE**는 해당 버퍼를 편집 세션에서 가장 오래된 버퍼와 바꿉니다.

다음 표에 나온 선택 항목 중 하나를 입력할 수 있습니다.

표 8-26

키워드	효과
DELETE_ONLY	지정된 버퍼를 삭제합니다.
WRITE_FIRST	지정된 버퍼를 기록(저장)한 다음 삭제합니다.
QUIT	기본 선택 항목입니다. 버퍼를 삭제하지 않습니다.

다음 보기는 수정된 **MYFILE.TXT** 버퍼의 삭제를 요청합니다.

```
Command: DELETE BUFFER MYFILE.TXT
That's a modified buffer. Type delete_only, write_first, or quit:
```

버퍼 상태 변경

SET BUFFER 명령을 사용하여 버퍼의 편집 상태를 변경합니다. 즉, 버퍼를 수정할 수 있는지 여부와 **EVE** 종료 시 버퍼가 파일에 기록되는지 여부를 이 명령을 통해 변경할 수 있습니다.

각 명령에 대해 다음 **SET BUFFER** 명령 키워드 중 하나를 지정할 수 있습니다.

표 8-27

키워드	효과
MODIFIABLE	기본 설정입니다. 버퍼를 수정할 수 있습니다. 또한, 버퍼의 이전 모드(삽입 또는 겹쳐쓰기)가 복원됩니다.
READ_ONLY	버퍼가 수정된 경우에도 저장(기록)되지 않습니다(WRITE 와 반대). 또한, 버퍼가 수정할 수 없는 상태로 설정됩니다. 단, 사용자는 필요에 따라 버퍼를 수정 가능하게 설정할 수 있습니다.
UNMODIFIABLE	버퍼를 수정할 수 없습니다. 또한, 버퍼의 모드(삽입 또는 겹쳐쓰기)가 무시됩니다.
WRITE	기본 설정입니다. 종료시 수정된 버퍼가 저장(기록)됩니다(READ_ONLY 와 반대). 버퍼가 읽기 전용이거나 수정 불가능한 경우, SET BUFFER WRITE 는 해당 버퍼를 수정 가능하게 만들고 이전 모드(삽입 또는 겹쳐쓰기)를 복원합니다.

기본적으로 버퍼 상태가 **MODIFIABLE** 및 **WRITE**로 설정되므로 버퍼 내용을 변경하고 변경된 버퍼를 파일에 저장할 수 있습니다.

내용이 실수로 변경되지 않도록 버퍼 상태를 바꾸려면 다음 명령을 사용하여 버퍼를 **READ_ONLY**(수정 불가능하다는 것을 나타냄)로 설정합니다.

```
Command: SET BUFFER READ_ONLY
```

버퍼가 임시 저장 영역(즉, “스크래치 패드”)이 되도록 버퍼 상태를 변경하려면 다음 명령을 사용하여 버퍼를 **READ_ONLY** 및 **MODIFIABLE**로 설정합니다.

Command: SET BUFFER READ_ONLY

Command: SET BUFFER MODIFIABLE

이 경우, 버퍼를 편집할 수 있지만 **EVE**를 종료할 때 버퍼가 저장되지 않습니다.

Messages 버퍼 표시

EVE는 화면 맨 아래에 표시되는 메시지 창을 사용하여 편집 세션 도중의 오류 및 정보 메시지를 전달합니다. 메시지 창은 **Messages** 버퍼의 마지막 메시지를 표시합니다.

이러한 메시지를 **BUFFER** 명령을 사용하여 표시할 수 있습니다. **Messages** 버퍼의 내용을 표시하려면 **Do** 키를 누르고 **BUFFER MESSAGES** 명령을 입력합니다. 편집하던 버퍼로 되돌아가려면 **Do** 키를 누르고 **BUFFER** 명령과 해당 버퍼의 이름을 입력합니다.

또한, **SHOW BUFFERS** 명령을 입력하여 작성한 버퍼를 표시하고 **Select** 키를 눌러 버퍼를 선택할 수 있습니다.

여러 버퍼 편집

여러 파일을 편집하거나 텍스트 블록 조작을 위한 임시 저장 영역을 원할 경우, 버퍼를 여러 개 사용할 수 있습니다. **GET FILE** 또는 **OPEN**, **OPEN SELECTED**, **BUFFER** 중에서 하나를 사용하여 새 버퍼를 만들 수 있습니다.

GET FILE 명령 사용

이미 존재하는 파일을 사용하여 새 버퍼를 만들려면 **GET FILE**(또는 **OPEN**) 명령과 새 버퍼에 복사할 파일의 이름을 입력합니다. 파일 이름과 파일 형식의 일부 또는 전체 문자를 대체하는 별표(*) 와일드카드 문자를 사용할 수 있습니다. 또한, 백분율 기호(%) 와일드카드 문자로 파일 이름과 파일 형식의 문자 하나를 대체할 수 있고, 줄임표[...] 와일드카드 문자로 디렉토리 지정 방법을 대체할 수 있습니다.

OPEN SELECTED 명령 사용

다음과 같이 **OPEN SELECTED** 명령을 사용하여 새 버퍼를 만들 수 있습니다.

1. 커서를 열고자 하는 파일 이름으로 이동합니다.
2. **OPEN SELECTED** 명령을 입력합니다.

BUFFER 명령 사용

특정 버퍼를 현재 **EVE** 창에 표시하려면 **BUFFER** 명령과 함께 현재 창에 표시할 버퍼의 이름을 입력합니다. 버퍼 이름에는 와일드카드 문자를 사용할 수 없습니다. 별표(*) 및 백분율 기호(%)는 버퍼 이름에서 리터럴 문자로 간주됩니다. 지정한 버퍼가 존재하지 않으면 새 버퍼가 만들어집니다.

지정된 파일이 존재하면 파일 내용이 새 버퍼로 읽히고 현재 창에 버퍼가 표시됩니다. 와일드카드가 사용된 파일 지정 방법과 일치하는 파일이 여러 개이면 **EVE**는 **\$CHOICES\$** 버퍼를 통해 해당 파일들을 나열하고, 더 완전한 파일 지정 방법을 제공하라는 메시지를 표시합니다. 검색 목록이나 줄임표(...) 와일드카드가 사용되면 **EVE**는 일치하는 첫 번째 파일을 엽니다. 일치하는 파일이 없으면 **EVE**는 빈 버퍼를 만들어 현재 창에 표시합니다.

현재 창의 버퍼를 변경하려면 **Do** 키를 누르고 **BUFFER** 명령과 화면에 표시할 버퍼의 이름을 입력한 다음, **Enter** 키를 누릅니다. 버퍼 이름을 잊은 경우에는 **SHOW BUFFERS** 명령을 입력하여 편집 세션에서 사용 중인 활성화된 버퍼의 이름을 표시하고 **Select** 키로 버퍼를 지정합니다.

EVE로 파일 읽어오기

다음 네 가지 방법을 통해 파일을 **EVE** 버퍼로 읽어올 수 있습니다.

- 파일 지정 방법을 지정하여 **EVE**를 실행합니다.
- **INCLUDE FILE** 명령과 포함할 파일의 이름을 입력합니다. **EVE**는 커서 위치의 바로 앞 행에 해당 파일의 전체 내용을 추가합니다. **INCLUDE FILE** 명령을 사용한다고 해서 상태 표시줄의 버퍼 이름이 바뀌지는 않습니다.
- **GET FILE** 또는 **OPEN** 명령과 사용할 파일의 이름을 입력합니다. 두 명령은 새 버퍼를 만들고 기존 파일의 내용을 버퍼로 읽어옵니다. 상태 표시줄의 버퍼 이름은 **GET FILE** 또는 **OPEN** 명령에서 지정한 파일 이름과 동일합니다(319페이지의 “여러 버퍼 편집” 참조).
- 파일 이름을 선택 또는 검색한 다음, **OPEN SELECTED** 명령을 입력합니다.

EVE에서 파일 쓰기

현재 버퍼의 내용을 파일에 쓰려면 **WRITE FILE** 명령을 입력합니다. **WRITE FILE** 명령에 파일 지정 방법을 포함할 수 있습니다. 파일 지정 방법을 포함하지 않으면 **EVE**는 입력 파일 지정 방법을 사용하여 파일을 기록합니다. **BUFFER** 또는 **NEW** 명령으로 현재 버퍼를 만든 경우, 파일 기록을 위한 파일 지정 방법을 제공하라는 메시지가 표시됩니다.

다음 보기는 버퍼와 연결된 출력 파일을 사용하여 버퍼를 파일에 기록하는 방법을 보여 줍니다.

```
Command: GET FILE RHYMES.DAT
.
.
.
Command: WRITE FILE

3 lines written to WORKDISK:[USER]RHYMES.DAT;2
```

창 사용

EVE 편집 세션 도중, 현재 편집하고 있는 버퍼가 화면의 창에 표시됩니다. 창 아래쪽에 표시되는 강조된 상태 표시줄은 버퍼 이름, 현재 편집 모드, 버퍼의 현재 방향을 알려줍니다. EVE에서는 동시에 여러 개의 창을 터미널 화면에서 볼 수 있습니다. 예를 들어, 터미널 화면에서 두 개의 창을 사용하여 동일한 버퍼의 다른 섹션을 확인 및 편집할 수 있습니다. 표 8-28에는 창을 만들고 조작하는 데 사용되는 EVE 키가 설명되어 있습니다.

표 8-28 EVE 창에서 사용되는 키

키 또는 키 시퀀스	창 환경에서의 기능
GOLD Next Screen	커서를 다음 또는 다른 창으로 이동합니다. NEXT WINDOW 명령과 동일합니다.
GOLD Prev Screen	커서를 이전 또는 다른 창으로 이동합니다. PREVIOUS WINDOW 명령과 동일합니다.

표 8-29에는 창을 만들고 조작하는 데 사용되는 EVE 명령이 설명되어 있습니다.

표 8-29 EVE 창 명령

명령	창 환경에서의 기능
DELETE WINDOW	창을 여러 개 사용하는 경우 현재 창을 삭제합니다.
ENLARGE WINDOW	현재 창을 지정된 행 수만큼 확대합니다. 예를 들어, ENLARGE WINDOW 5는 창을 5행만큼 확대합니다. 인접한 창은 그만큼 축소됩니다.

표 8-29

EVE 창 명령(계속)

명령	창 환경에서의 기능
NEXT WINDOW 또는 OTHER WINDOW	커서를 다음 또는 다른 창으로 이동합니다.
ONE WINDOW	현재 창을 하나의 큰 창으로 복원합니다. 다른 모든 창은 화면에서 삭제되지만, 이러한 창과 연결된 버퍼가 삭제되는 것은 아닙니다.
PREVIOUS WINDOW	커서를 이전 또는 다른 창으로 이동합니다.
SET WIDTH	화면에 표시되는 행 너비를 설정합니다. 화면 너비는 양의 정수로 설정되며, 기본적으로 터미널 설정(대개 80열)과 같습니다. 너비가 80보다 큰 값으로 설정되면 EVE는 현재 편집 세션에 대해 터미널을 132열 모드로 설정합니다. EVE를 종료하면 터미널은 기본 설정으로 복원됩니다. 너비를 설정하면 모든 창에서 텍스트 표시가 변경됩니다.
SHIFT LEFT	현재 창을 지정된 열 수만큼 왼쪽으로 이동합니다. SHIFT RIGHT 명령의 효과를 취소하려는 경우에만 SHIFT LEFT 명령을 사용할 수 있습니다.
SHIFT RIGHT	현재 터미널 화면에 나타나지 않는 문자 열을 볼 수 있도록 현재 창을 지정된 열 수만큼 오른쪽으로 이동합니다.
SHRINK WINDOW	현재 창을 지정된 행 수만큼 축소합니다. 예를 들어, SHRINK WINDOW 5는 창을 5행만큼 축소합니다. 인접한 창은 그만큼 확대됩니다.
SPLIT WINDOW	현재 창을 분할하여 더 작은 두 개의 창을 만듭니다. 이 명령에서 숫자를 지정하여 창을 두 개 이상의 부분으로 나눌 수 있습니다. 예를 들어, SPLIT WINDOW 3은 현재 창을 세 개의 창으로 분할합니다.
TWO WINDOWS	SPLIT WINDOW 2 명령과 동일합니다.

특정 버퍼의 섹션 두 개 보기

특정 파일의 섹션 두 개를 동시에 보려면 **SPLIT WINDOW** 명령을 사용합니다. 이 명령은 화면을 분할하여 두 개의 동일한 창을 만듭니다. 커서는 버퍼에서 원래 위치를 유지하지만 아래쪽 창에만 표시됩니다. 버퍼 이름은 두 개의 상태 표시줄에서 동일하게 나타납니다.

긴 파일의 섹션 두 개를 표시하면 파일 내에서 텍스트를 효율적으로 이동할 수 있습니다. 파일의 특정 부분에서 텍스트를 선택 및 제거하여 다른 부분에 삽입할 수 있습니다. 커서를 특정 창에서 다른 창으로 이동하려면 **NEXT WINDOW** 명령을 입력합니다.

화면에서 두 번째 창을 제거하고 현재 창을 전체 편집 영역을 차지하도록 확대하려면 **Do** 키를 누르고 **ONE WINDOW** 명령을 입력한 다음, **Enter** 키를 누릅니다.

두 개의 버퍼 편집

다른 파일을 포함하는 두 개의 버퍼를 편집하려면 다음 절차를 따릅니다.

표 8-30

단계	작업
1	SPLIT WINDOW 명령을 입력하여 화면에 두 개의 창을 만듭니다. EVE는 화면을 분할하고 두 개의 창을 만듭니다. 커서는 버퍼에서 원래 위치를 유지하지만 아래쪽 창에만 표시됩니다. 버퍼 이름은 각각의 강조된 상태 표시줄에서 동일하게 나타납니다.
2	GET FILE, OPEN 또는 OPEN SELECTED 명령을 사용하여 두 번째 파일을 현재 창에 표시합니다. 이전에 편집 세션 도중 만든 버퍼를 현재 창에 표시하려면 BUFFER 명령과 표시할 버퍼의 이름을 입력합니다. 터미널 화면에 두 개의 다른 버퍼가 표시됩니다. 특정 버퍼에서 텍스트를 선택 및 제거하여 다른 버퍼에 삽입할 수 있습니다. 커서를 특정 창에서 다른 창으로 이동하려면 NEXT WINDOW 명령을 입력합니다.

하위 프로세스 만들기

편집 세션을 종료하지 않고 EVE 편집 세션과 DCL 명령 수준 간에 전환하기 위해 SPAWN 명령을 사용하여 하위 프로세스를 만들 수 있습니다. 이 명령이 사용되면 EVE는 현재 편집 세션을 일시 중단하고 터미널을 새 하위 프로세스에 연결합니다. 그리고 터미널 화면에 DCL 프롬프트(\$)가 표시됩니다.

생성

하위 프로세스에서 임의의 OpenVMS 유틸리티나 DCL 명령을 실행할 수 있지만, 하위 프로세스를 생성하는 가장 일반적인 이유는 Mail 유틸리티를 호출하고 화면 지향 프로그램을 실행하는 것입니다.

편집 세션으로 되돌아가려면 DCL 명령 LOGOUT을 입력하여 하위 프로세스에서 로그아웃합니다. 이렇게 하면 편집 세션이 다시 시작되고 커서가 하위 프로세스를 생성하기 전에 있던 위치에 표시됩니다. DCL 명령을 SPAWN 명령의 매개 변수로 제공하여 특정 하위 프로세스를 만들 수도 있습니다.

다음 보기는 EVE에서 Mail 유틸리티를 생성합니다.

```
[End of file]
```

```
Buffer: MAIN | Write | Insert | Forward  
Command: SPAWN MAIL
```

Mail 유틸리티의 프롬프트(MAIL>)가 화면에 나타납니다. Mail을 종료하면 하위 프로세스에서 자동으로 로그아웃되고 편집 세션이 다시 시작됩니다.

DCL에서 EVE 하위 프로세스 생성

프로세스를 생성하여 DCL을 사용하는 대신, EVE 편집 세션에 대한 프로세스를 생성한 다음 상위 DCL 프로세스에 연결하여 DCL 명령과 유틸리티를 사용할 수 있습니다.

DCL 명령 수준으로 되돌아가려면 EVE 명령 ATTACH를 사용하여 상위 프로세스로 이동합니다.

편집 세션을 다시 시작하려면 DCL 명령 ATTACH와 함께 하위 프로세스의 프로세스 이름을 사용하여 편집 하위 프로세스에 다시 연결합니다.

다음 보기에서는 DCL 명령 SPAWN을 사용하여 하위 프로세스를 만듭니다. SPAWN 명령이 SMITH_1이라는 하위 프로세스를 만들면, 하위 프로세스 수준에서 EVE가 실행되고 편집 세션이 수행됩니다. 편집 세션의 끝에서 DCL로 되돌아가기 위해 ATTACH 명령이 입력됩니다. 이어서 편집 세션을 다시 시작하기 위해 SMITH_1 하위 프로세스의 프로세스 이름과 함께 DCL 명령 ATTACH가 입력됩니다.

```
$ SPAWN
%DCL-S-SPAWNED, process SMITH_1 spawned
%DCL-S-ATTACHED, terminal now attached to process SMITH_1
```

[End of file]

```
Buffer: MAIN | Write | Insert | Forward
Command: ATTACH SMITH

$ ATTACH SMITH_1
```

EVE를 사용하여 텍스트 파일 편집
하위 프로세스 만들기

9 파일 정렬 및 병합

이 장에서는 OpenVMS Sort/Merge 유틸리티(SORT/MERGE)를 사용하는 방법을 설명합니다. Sort/Merge 유틸리티는 다음 두 가지 작업을 수행합니다.

- 사용자가 선택한 **필드**에 따라 하나 이상의 입력 파일의 레코드를 정렬하고 다시 정렬된 출력 파일을 하나 생성합니다.
- 이전에 같은 **키** 필드에 따라 정렬된 최대 10개(고성능 Sort/Merge 유틸리티의 경우 최대 12개)의 입력 파일을 병합하고 하나의 출력 파일을 생성합니다.

Alpha 시스템에서 고성능 Sort/Merge 유틸리티를 선택할 수도 있습니다. 이 유틸리티는 Alpha 아키텍처를 사용하여 대부분의 정렬 및 병합 작업을 위한 성능을 향상시킵니다. 자세한 내용은 329페이지의 “고성능 Sort/Merge”를 참조하십시오.

이 장에서는 다음을 설명합니다.

- 고성능 Sort/Merge
- 파일 정렬
- 조합 순서 지정
- 일괄 처리 작업으로 Sort 실행
- 파일 병합
- 터미널에서 레코드 입력
- Sort/Merge 지정 방법 파일 사용
- 정렬 또는 병합 작업 최적화
- Sort/Merge 한정자 요약

추가 정보는 다음을 참조하십시오.

- 이 장에서 사용된 명령에 대한 내용은 *OpenVMS DCI Dictionary*를 참조하십시오.
- 시스템 관리자가 Sort/Merge 유틸리티를 더욱 효과적으로 사용할 수 있는 방법에 대해서는 *OpenVMS System Manager's Manual*을 참조하십시오.

고성능 Sort/Merge

Alpha 시스템에서 고성능 Sort/Merge 유틸리티를 선택할 수도 있습니다. 이 유틸리티는 Alpha 아키텍처를 사용하여 대부분의 정렬 및 병합 작업에서의 성능을 향상시킵니다.

고성능 Sort/Merge 유틸리티는 SORT/MERGE와 동일한 명령행 인터페이스를 사용합니다. 고성능 Sort/Merge 유틸리티와 SORT/MERGE 간의 차이점에 대해서는 이 장 전체에서 설명합니다.

고성능 Sort/Merge 유틸리티를 선택하려면 SORTSHR 논리를 사용합니다. 다음과 같이 SYS\$LIBRARY에서 고성능 정렬 실행 파일을 가리키도록 SORTSHR을 정의합니다.

```
$ define sortshr sys$library:hypersort.exe
```

SORT/MERGE로 돌아가려면 SORTSHR 지정을 취소합니다. SORTSHR을 정의하지 않으면 SORT/MERGE 유틸리티가 기본값이 됩니다.

주

메모리 할당 차이로 인해, Sort/Merge 유틸리티가 같은 양의 가상 메모리에서 수행할 수 있는 것과 같은 수의 동시 정렬 작업을 수행하기 위한 고성능 Sort/Merge 유틸리티의 기능이 제한될 수 있습니다.

이러한 경우에는 프로세스에 사용할 수 있는 가상 메모리의 양을 늘리거나 작업 집합 범위를 줄일 수 있습니다. 시스템 매개 변수를 사용하여 가상 메모리의 양을 변경하거나 작업 집합 범위를 줄이는 데 대한 자세한 내용은 *OpenVMS System Management Utilities Reference Manual*을 참조하십시오.

고성능 Sort/Merge 유틸리티의 동작은 표 9-1에 설명된 것을 제외하곤 SORT/MERGE와 동일합니다.

지원되지 않는 한정자를 사용하거나 한정자에 지원되지 않는 값을 할당하려는 경우 고성능 Sort/Merge 유틸리티에서 오류를 생성합니다.

표 9-1 고성능 Sort/Merge: 동작의 차이점

기능	고성능 Sort/Merge 동작
키 데이터 유형	H-FLOATING 및 ZONED 십진 데이터 유형은 지원되지 않습니다. BINARY 데이터 유형 키의 크기는 1, 2, 4 또는 8바이트이어야 합니다. 16바이트 이진 키는 지원되지 않습니다. 이 기능의 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다.
조합 순서	NCS(국가별 문자 집합) 조합 순서는 지원되지 않습니다. 이 기능의 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다. /COLLATING_SEQUENCE 한정자에 NCS 조합 순서의 이름을 지정하지 마십시오. ASCII, EBCDIC 및 MULTINATIONAL 조합 순서는 지원됩니다. 기본값은 ASCII입니다. 지정 방법 방법을 사용하여 사용자 고유의 조합 순서를 정의하거나 수정할 수 없습니다. 이 기능의 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다.
지정 방법 파일	지정 방법 파일은 지원되지 않습니다. 이 기능의 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다. /SPECIFICATION 한정자를 사용하지 마십시오.
내부 정렬 프로세스	레코드 정렬 프로세스만 지원됩니다. 이 기능의 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다. /PROCESS=RECORD를 지정하거나 /PROCESS 한정자를 생략할 수 있습니다. /PROCESS 한정자의 TAG, ADDRESS 및 INDEX 값은 지원되지 않습니다.

표 9-1

고성능 Sort/Merge: 동작의 차이점 (계속)

기능	고성능 Sort/Merge 동작
통계 요약 정보	현재 다음 통계 정보가 지원됩니다. Records read(읽은 레코드) Records sorte(정렬된 레코드) Records output(출력된 레코드) Input record length(입력 레코드 길이) 다음 통계는 사용할 수 없습니다. Internal length(내부 길이) Output record length(출력 레코드 길이) Sort tree size(정렬 트리 크기) Number of initial runs(초기 실행 수) Maximum merge order(최대 병합 순서) Number of merge passes(병합 통과 수) Work file allocation(작업 파일 할당) 이 기능의 완전한 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다.

파일 정렬

파일을 정렬하려면 **DCL** 명령 **SORT**를 사용합니다. 정렬할 파일의 이름을 쉼표로 구분하여 지정한 다음에 만들려는 정렬된 출력 파일의 이름을 지정합니다.

옵션으로 정렬할 각 필드에 대해 키를 지정할 수도 있습니다. 각 키에는 다음 정보가 포함되어 있습니다.

- 레코드에서 키 필드의 시작 위치(필수)
- 키의 크기(필수)
- 키의 데이터 유형
- 레코드가 정렬된 순서
- 키의 우선 순위

키를 지정하지 않으면 **Sort**에서는 키가 하나뿐이고 이 키 필드가 다음과 같다고 가정합니다.

- 레코드의 처음 위치에서 시작합니다.
- 전체 레코드를 포함합니다.
- 문자 집합을 포함합니다.
- 오름차순으로 정렬됩니다.

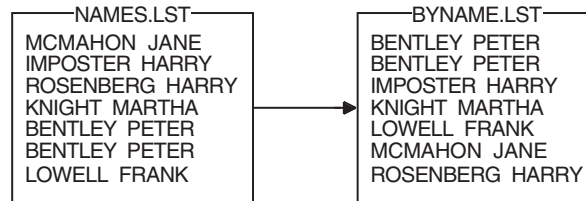
다음 두 보기에서는 기본 키를 사용합니다.

1. 이 보기에서는 **NAMES.LST** 파일을 오름차순으로 정렬합니다.

```
$SORT NAMES.LST BYNAME.LST
```

이 명령은 그림 9-1과 같이 정렬된 출력 파일 **BYNAME.LST**를 만듭니다.

그림 9-1 오름차순으로 정렬된 목록



ZK-5055A-GE

2. 이 보기에서는 `NAMES.LST`와 `NAMES2.LST` 파일이 정렬된 출력 파일 `BYNAME.LST`로 정렬됩니다. `Sort`에서는 이러한 파일이 하나의 큰 파일인 것처럼 처리합니다.

```
$ SORT NAMES.LST,NAMES2.LST BYNAME.LST
```

`SORT` 한정자의 전체 목록은 364페이지의 “정렬/병합 한정자 요약”을 참조하십시오.

키 정의

키를 정의하려면 `/KEY` 한정자를 사용합니다. 여러 키를 정의할 경우 각각의 키마다 별도의 `/KEY` 한정자를 사용합니다.

표 9-2에서는 키를 구성하는 다섯 가지 요소를 설명합니다.

표 9-2 /KEY 한정자 값

키 요소	값	설명
키 위치	POSITION:n	레코드 내에서 키 필드의 첫째 바이트 위치입니다. 레코드의 첫째 바이트는 위치 1입니다. POSITION:n은 필수적입니다.
키 크기	SIZE:n	키 필드의 길이입니다. SIZE:n은 부동 소수점 데이터인 경우를 제외하고는 필수적입니다. 키에 지정하는 데이터 유형에 따라 크기를 지정할 때 사용할 수 있는 값이 결정됩니다. 다음 표에서는 각 데이터 유형에 사용할 수 있는 값과 키 크기를 지정하는 데 사용되는 단위를 나열합니다.
	데이터	유효한 범위 단위
	문자	1에서 32,767 문자
	이진수	1, 2, 4, 8 또는 16(고성능 Sort/Merge 유틸리티의 경우 이진 데이터 유형 키의 크기는 1, 2, 4 또는 8바이트이어야 합니다. 16바이트 이진 키의 지원은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다.) 바이트
	십진수	1에서 31 자릿수
	부동 소수점	값이 필요하지 않습니다.
십진 데이터인 경우 소수점 기호가 별도의 바이트로 저장되면 이 바이트는 데이터 크기로 계산되지 않습니다.		
레코드 끝을 벗어나는 키를 지정하면 Sort에서는 누락된 문자를 null 문자로 처리합니다.		
데이터 유형	CHARACTER BINARY	문자 데이터입니다. CHARACTER는 기본 데이터 유형입니다. 이진 데이터입니다. SIGNED — 부호 있는 이진 또는 십진 데이터입니다. SIGNED는 이진 및 십진 데이터의 기본값입니다. UNSIGNED — 부호 없는 이진 또는 십진 데이터입니다.

키 요소	값	설명
	F_FLOATING	F_FLOATING 형식 데이터입니다.
	D_FLOATING	D_FLOATING 형식 데이터입니다.
	G_FLOATING	G_FLOATING 형식 데이터입니다.
	H_FLOATING	VAX 시스템에서 H_FLOATING 형식 데이터입니다. 고성능 Sort/Merge 유틸리티에서는 현재 지원되지 않습니다.
	S_FLOATING	Alpha 시스템에서 IEEE S_FLOATING 형식 데이터입니다.
	T_FLOATING	Alpha 시스템에서 IEEE T_FLOATING 형식 데이터입니다.
	DECIMAL	십진 데이터입니다. TRAILING_SIGN — 후행 기호 십진 데이터입니다. TRAILING_SIGN 은 십진 데이터의 기본값입니다. LEADING_SIGN — 선행 기호 십진 데이터입니다. 선행 기호는 필드의 처음 위치에 있어야 하고 필드는 왼쪽에 0으로 채워져 있어야 합니다. OVERPUNCHED_SIGN — 편치가 많이 있는 십진 데이터입니다. OVERPUNCHED_SIGN은 십진 데이터의 기본값입니다. SEPARATE_SIGN — 분리 기호 십진 데이터입니다.
	ZONED	구역화된 십진 데이터입니다. 고성능 Sort/Merge 유틸리티에서는 현재 지원되지 않습니다.
	PACKED_DECIMAL	팩트형 십진 데이터입니다.
정렬 순서	ASCENDING	정렬 작업을 오름차순 사전순이나 숫자순으로 정렬합니다. ASCENDING이 기본 순서입니다.
	DESCENDING	정렬 작업을 내림차순 사전순이나 숫자순으로 정렬합니다.
키 우선 순위	NUMBER:n	키의 순위에서 여러 키를 나열하지 않는 경우 각 키의 우선 순위를 지정합니다. 1에서 255까지의 값을 지정할 수 있습니다.

키 필드의 데이터가 문자 데이터가 아닌 경우에는 데이터 유형을 지정해야 합니다.
Sort/Merge 유틸리티에서 인식하는 데이터 유형은 다음과 같습니다.

BINARY, [SIGNED]

BINARY, UNSIGNED

CHARACTER

DECIMAL, LEADING_SIGN, SEPARATE_SIGN [SIGNED]

DECIMAL, LEADING_SIGN, [OVERPUNCHED_SIGN, SIGNED]

DECIMAL [,SIGNED, TRAILING_SIGN, OVERPUNCHED_SIGN]

DECIMAL, [TRAILING SIGN], SEPARATE_SIGN, [SIGNED]

DECIMAL, UNSIGNED

D_FLOATING

F_FLOATING

G_FLOATING

H_FLOATING

S_FLOATING, IEEE (Alpha 시스템에만 해당)

T_FLOATING, IEEE (Alpha 시스템에만 해당)

PACKED_DECIMAL

ZONED

괄호 안의 항목은 기본값이므로 지정할 필요가 없습니다.

주

십진 문자열 데이터의 경우 Sort/Merge 유틸리티는 입력 문자열의 유효하지 않은 자릿수를 VAX 및 Alpha 시스템에 따라 다르게 보고합니다. VAX 시스템에서는 비교하기 위해 유효하지 않은 자릿수 또는 예약된 피연산자가 유효한 십진 문자열로 변환된다는 메시지를 표시합니다. Alpha 시스템인 경우 Sort/Merge 유틸리티가 같은 변환을 수행하지만 메시지를 표시하지는 않습니다. 두 경우 모두 입력 파일의 데이터가 변경되지 않고 출력 파일에 쓰여집니다.

그림 9-2에서 **EMPLOYEE.LST** 파일의 각 레코드는 (1) 부서 이름, (2) 계정 번호 및 (3) 직원 이름과 같은 세 개의 필드로 구성됩니다.

그림 9-2

목록의 레코드 필드

EMPLOYEE.LST		
①	②	③
BST	7828	MCMAHON JANE
ADM	7933	IMPOSTER HARRY
ADM	7933	ROSENBERG HARRY
COM	8102	KNIGHT MARTHA
ANS	8042	BENTLEY PETER
ANS	5243	BENTLEY PETER
BIO	7951	LOWELL FRANK

ZK-5056A-GE

다음 보기에서는 키 필드를 사용하는 경우와 사용하지 않는 경우에 **EMPLOYEE.LST**의 레코드를 정렬하는 방법을 보여 줍니다.

- 이 보기에서는 계정 번호 필드를 설명하는 **/KEY** 한정자를 사용하여 계정 번호에 따라 **EMPLOYEE.LST**를 정렬합니다.

```
$ SORT/KEY=(POSITION:5,SIZE:4,DECIMAL) EMPLOYEE.LST BILLING1.LST
```

이 명령은 키 필드(계정 번호)가 위치 5에서 시작하고 길이가 4자이며, 십진 데이터를 포함하고 오름차순(기본값)으로 정렬되어야 함을 지정합니다. 그림 9-3에서는 이 정렬 작업의 결과를 보여 줍니다.

그림 9-3

키 필드에 따라 정렬

EMPLOYEE.LST			BILLING1.LST		
BST	7828	MCMAHON JANE	ANS	5243	BENTLEY PETER
ADM	7933	IMPOSTER HARRY	BST	7828	MCMAHON JANE
ADM	7933	ROSENBERG HARRY	ADM	7933	ROSENBERG HARRY
COM	8102	KNIGHT MARTHA	ADM	7933	IMPOSTER HARRY
ANS	8042	BENTLEY PETER	BIO	7951	LOWELL FRANK
ANS	5243	BENTLEY PETER	ANS	8042	BENTLEY PETER
BIO	7951	LOWELL FRANK	COM	8102	KNIGHT MARTHA

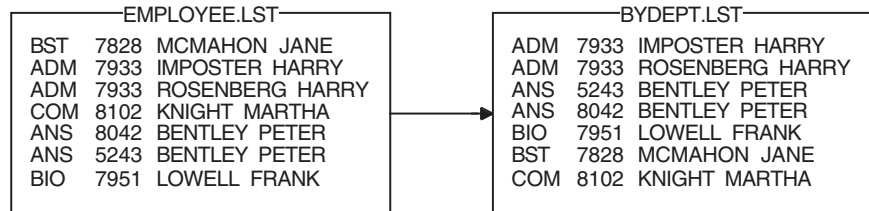
ZK-5058A-GE

- 이 보기에서는 키 필드를 지정하지 않고 **EMPLOYEE.LST** 파일을 정렬하는 방법을 보여 줍니다.

```
$ SORT EMPLOYEE.LST BYDEPT.LST
```

키가 지정되지 않았으므로 **Sort**에서는 기본 특성을 사용합니다. 그림 9-4에서는 이 정렬 작업의 결과를 보여 줍니다.

그림 9-4 기본 키 레코드로 정렬



ZK-5057A-GE

Sort에서는 **EMPLOYEE.LST**의 각 레코드를 하나의 문자 데이터 키로 처리합니다. 이 보기에서 각 레코드는 부서 이름, 계정 번호 및 직원 이름을 포함합니다. **Sort**는 중복된 부서 이름이 있으면 계정 번호에 따라 정렬합니다. 그런 다음 중복된 계정 번호가 있으면 직원 이름에 따라 정렬합니다. 계정 번호는 레코드의 일부입니다. 다르게 지정하지 않는 한 계정 번호는 문자 데이터로 처리됩니다.

다중 키 필드

두 개 이상의 키(최대 255개의 키로 제한됨)를 사용하여 정렬할 수 있습니다. 먼저 기본 키를 사용하고 다음엔 보조 키를 사용하는 등 키의 우선 순위에 따라 다중 키를 지정할 수 있습니다. 또는 **NUMBER:n**을 사용하여 키의 우선 순위를 지정할 수 있습니다. 각 키는 오름차순이나 내림차순일 수 있습니다.

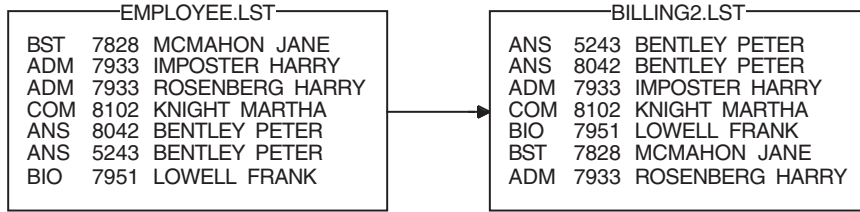
다음 보기에서는 **EMPLOYEE.LST** 파일을 먼저 직원 이름에 따라 정렬한 다음 같은 이름이 있으면 계정 번호에 따라 정렬합니다.

```
$ SORT /KEY=(POSITION:10,SIZE:15,CHARACTER) -
_$/KEY=(POSITION:5,SIZE:4,DECIMAL) EMPLOYEE.LST BILLING2.LST
```

그림 9-5에서는 이 정렬 작업의 결과를 보여 줍니다.

그림 9-5

다중 키 필드로 정렬



ZK-5059A-GE

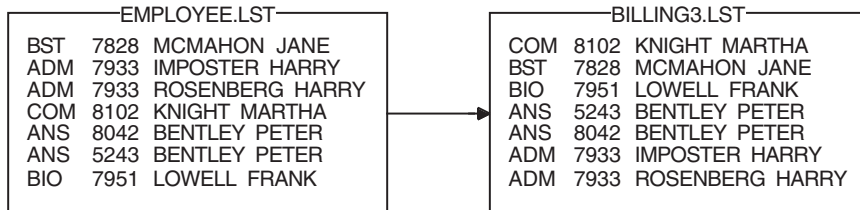
다음 보기에서는 레코드를 먼저 부서 이름을 기준으로 내림차순으로 정렬한 다음 직원 이름을 기준으로 오름차순으로 정렬합니다.

```
$ SORT/KEY=(POSITION:1,SIZE:3,DESCENDING) -
_$ /KEY=(POSITION:10,SIZE:15) -
_$ EMPLOYEE.LST BILLING3.LST
```

그림 9-6에서는 이 정렬 작업의 결과를 보여 줍니다.

그림 9-6

다중 키 필드로 정렬(오름차순 및 내림차순)



ZK-5060A-GE

동일한 키 필드

Sort/Merge 유틸리티(SORT/MERGE)정렬 동일한 키 필드 사용 기본적으로 Sort/Merge에 서는 키 필드가 같은 레코드를 보관하지만 입력 파일에 표시된 것과 같은 순서를 항상 유지 하는 것은 아닙니다. 동일한 키의 레코드가 정렬되는 방식을 제어하려면 다음 한정자 중 하나를 지정합니다.

/STABLE

키가 같은 레코드의 입력 순서를 유지합니다. 여러 입력 파일을 정렬하는 데 이 한정자를 사용하면, 출력에서 첫 번째 파일의 등가 키가 있는 레코드가 두 번째 파일의 등가 키가 있는 레코드 앞에 옵니다.

/NODUPPLICATES

동일한 키가 있는 레코드 사본 중 하나만 유지합니다. 보관할 중복된 레코드를 지정하려면 프로그램 수준에서 **Sort**를 호출하고 등가 키 루틴을 지정합니다.

/STABLE과 /NODUPPLICATES 한정자는 호환되지 않습니다. 같은 명령행에 두 한정자 모두를 지정할 수 없습니다.

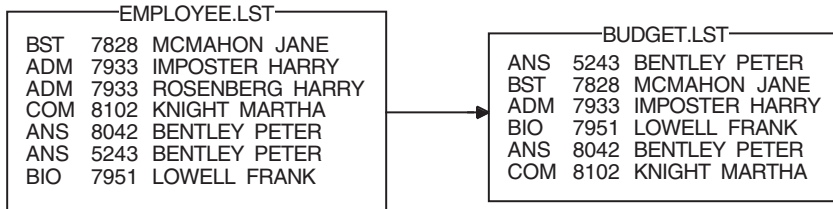
다음 보기에서는 중복된 계정 번호가 있는 레코드를 **EMPLOYEE.LST** 파일에서 삭제합니다.

```
$ SORT /KEY=(POSITION:5,SIZE:4) /NODUPPLICATES EMPLOYEE.LST BUDGET.LST
```

그림 9-7에서는 이 정렬 작업의 결과를 보여 줍니다.

그림 9-7

동일한 키 필드로 정렬



ZK-5061A-GE

비문자 데이터

문자 데이터가 아닌 항목을 포함하는 레코드를 정렬하는 경우 각 키의 데이터 유형을 지정합니다. 또한 비교하는 항목이 1바이트 이상을 사용할 수 있으므로 시작 위치와 크기를 주의해서 계산해야 합니다.

20자 다음에 3개의 **F_floating** 형식 부동 소수점 수가 오는 파일을 정렬할 경우 위치는 다음과 같습니다.

문자 데이터는 위치 1에서 20(20자)을 사용합니다.

첫 번째 **F_floating-point** 수는 위치 21에서 24를 사용합니다.

두 번째 **F_floating-point** 수는 위치 25에서 28을 사용합니다.

세 번째 **F_floating-point** 수는 위치 29에서 32를 사용합니다.

세 번째 부동 소수점 수를 기준으로 정렬하려면 다음과 같이 키 필드를 지정합니다.

```
$ SORT/KEY=(POSITION:29,F_FLOATING) STATS.RAW STATS.SOR
```

부동 소수점 수의 크기는 4바이트로 고정되어 있으므로 크기를 지정할 필요는 없습니다.

출력 파일 조직

기본적으로 **Sort**에서는 첫 번째 입력 파일과 같은 파일 조직으로 출력 파일을 만듭니다. 다른 출력 파일 조직을 지정하려면 **Sort** 명령행의 출력 파일 지정 방법 다음에 다음 한정자 중 하나를 포함시킵니다.

/FORMAT(레코드 형식)

이 출력 한정자를 사용할 경우 파일 레코드 형식, 길이 및 블록 크기를 정의할 수 있습니다.

/INDEXED_SEQUENTIAL

이 한정자를 사용하여 **인덱스 순차 파일** 조직이 있도록 출력 파일을 정의할 수 있습니다. 출력 파일 조직으로 인덱스 순차 파일을 지정하려면 다음 작업도 수행해야 합니다.

- 정렬 작업 수행 전에 출력 파일로 사용할 빈 파일을 만듭니다. **Sort**는 이미 존재하며 비어 있는 출력 파일을 필요로 합니다.
- SORT** 명령행에서 출력 파일 이름 다음에 **/OVERLAY** 한정자를 포함시킵니다.
/OVERLAY 한정자는 기존 파일이 입력 파일의 정렬된 레코드와 오버레이됨을 나타냅니다.

/RELATIVE

이 한정자를 사용하여 **상대적 파일 조직**이 있도록 출력 파일을 정의할 수 있습니다.

/SEQUENTIAL

이 한정자를 사용하여 **순차적 파일 조직**이 있도록 출력 파일을 정의할 수 있습니다.

다음 보기에서는 인덱스 순차 파일 **EMPLOYEE.LST**가 정렬된 후 순차 파일이 만들어집니다.

```
$ SORT/KEY=(POSITION:10,SIZE:15) -
_$ EMPLOYEE.LST BYNAME.LST/SEQUENTIAL
```

정렬 프로세스

Sort는 레코드, 태그, 주소 또는 인덱스와 같은 내부 프로세스 중 하나를 사용하여 파일을 정렬합니다. 고성능 Sort/Merge 유틸리티에서는 레코드 프로세스만 지원합니다. 태그, 주소 및 인덱스 프로세스 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다. 지정하는 프로세스에 따라 정렬 작업의 효율성에 영향을 줄 수 있습니다. 정렬 또는 병합 작업을 최적화하는 데 대한 자세한 내용은 360페이지의 “정렬 또는 병합 작업 최적화”를 참조하십시오.

다음 표에서는 이와 같은 네 가지 프로세스 유형을 설명합니다. 정렬 프로세스를 지정하려면 /PROCESS=type 한정자를 사용합니다.

표 9-3

정렬 프로세스	type	설명
레코드	RECORD	정렬하는 동안 레코드를 그대로 보관하고 전체 레코드로 구성되는 출력 파일을 만듭니다. 레코드는 기본 정렬 프로세스입니다.
태그	TAG	키 필드만 정렬한 다음 입력 파일을 다시 읽어 전체 레코드의 출력 파일을 만듭니다. 최종 결과는 전체 레코드 정렬과 같습니다. 태그 정렬은 정렬하는 동안 일반적으로 더 적은 작업 파일 공간을 사용하므로 디스크 공간이 적은 경우에 유용합니다. 대부분의 경우 태그 정렬은 입력 파일을 다시 읽는 데 시간이 더 걸리므로 레코드 정렬보다 느립니다.
주소	ADDRESS	키 필드만 정렬하고 이진 형식의 RFA(레코드 파일 주소) 의 인덱스인 출력 파일을 만듭니다. 주소 정렬은 레코드 정렬보다 빠르지만 레코드 주소와 입력 파일의 레코드를 연결하는 프로그램을 작성해야 합니다.
인덱스	INDEX	키 필드만 정렬하고 키와 RFA의 출력 파일(이진 형식)을 만듭니다. 주소 정렬과 마찬가지로 인덱스 정렬은 레코드 정렬보다 빠르지만 레코드 주소와 입력 파일의 레코드를 연결하는 프로그램을 작성해야 합니다.

조합 순서 지정

문자는 **조합 순서**에 따라 정렬됩니다. 문자 데이터를 포함하는 파일인 경우 `/COLLATING_SEQUENCE=sequence` 한정자를 사용하여 조합 순서를 지정할 수 있습니다. 다음 표에서는 조합 순서 옵션을 설명합니다.

표 9-4

조합 순서	순서	설명
ASCII	ASCII	문자 데이터의 기본 조합 순서입니다. ASCII 순서는 숫자(0-9), 대문자(A-Z), 소문자(a-z)순으로 정렬합니다.
EBCDIC	EBCDIC	EBCDIC 순서로 정렬된 출력 파일을 생성합니다. 데이터는 ASCII 표시로 남아 있습니다 EBCDIC 순서는 소문자(a-z), 대문자(A-Z), 숫자(0-9)순으로 정렬합니다.
DEC 다국적 문자 집합	MULTINATIONAL	다국적 조합 순서는 DEC 다국자 문자 집합(경로 확인 보류 참조)에 따라 문자를 조합합니다. MULTINATIONAL 문자 순서에서 문자는 다음 규칙에 따라 정렬됩니다. 문자의 분음 양식은 해당 문자의 조합 값이 지정됩니다(A', A", A`는 A로 조합됨). 소문자는 해당 대문자의 동일한 조합 값이 지정됩니다(a는 A로, a"는 A"로 조합됨). 두 문자열이 동등하게 비교되면 연결 구분이 수행됩니다. 이 문자열을 비교하여 분음 기호, 무시된 문자 또는 실제로 다르지만 동등하게 조합된 문자에 따른 차이점을 찾습니다. 문자열이 계속 동등하게 비교되면 문자의 숫자 코드에 따라 다른 비교가 수행됩니다. 마지막 비교에서 소문자가 대문자 앞에 정렬됩니다.
국가별 문자 집합(NCS)	조합 순서 이름	명명한 조합 순서가 NCS 라이브러리에 정의되어야 합니다. 자세한 내용은 <i>OpenVMS National Character Set Utility Manual</i>을 참조하십시오. 고성능 Sort/Merge 유틸리티에서는 NCS 조합 순서를 지원하지 않습니다. NCS에 대한 지원은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다.
사용자 정의 순서	(순서 문자열)	사용자 정의 조합 순서를 지정합니다. 사용자 정의 조합 순서는 명령 행 인터페이스가 아닌 지정 방법 파일을 통해서만 지원됩니다. 고성능 Sort/Merge 유틸리티에서는 사용자 정의 조합 순서를 지원하지 않습니다. 사용자 정의 조합 순서에 대한 지원은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다.

조합 순서	순서	설명
		단일 또는 이중 문자로 된 문자열이나 단일 문자의 범위를 지정하여 조합 순서를 정의합니다. 이중 문자는 한 문자인 것처럼 조합된 두 개의 단일 문자 집합입니다. 예를 들어 “CH”는 “C”로 조합되도록 정의할 수 있습니다. 이 문자열은 괄호로 묶어야 합니다. 또한 기수 연산자 %O, %D, %X를 사용하여 해당하는 8진수, 10진수 또는 16진수 값으로 문자를 나타낼 수 있습니다. 조합 순서를 지정하는 경우 다음 규칙을 따라야 합니다. - 문자를 인용 부호(“”)로 묶습니다. - 각 문자와 문자 범위를 쉼표(,)로 구분하고 전체 목록을 괄호로 묶습니다. - 정렬 또는 병합 작업의 문자 키에 표시되는 모든 문자에 조합 값을 지정합니다. 조합 값이 지정되지 않는 문자는 FOLD 또는 MODIFICATION 옵션이 지정되는 경우를 제외하고는 무시됩니다. - 문자를 한 번만 정의합니다. - 인용 부호(“”)를 사용하여 null 문자를 지정하면 안 됩니다. 대신 %X0와 같은 기수 연산자를 사용합니다. - 다른 인용 부호 집합(“”)으로 인용 부호를 묶거나 기수 연산자를 사용하여 지정합니다. 다음 문자열은 이중 문자 LL이 L과 M 사이의 단일 문자로 조합되는 조합 순서를 정의합니다. ("A" - "L", "LL", "M" - "Z")

주

추가적인 처리를 위해 파일을 정렬하거나 병합하는 데 다국적 조합 순서를 사용하는 경우 주의를 기울여야 합니다. 대부분의 프로그래밍 언어에서 순서 검사 프로시저는 숫자형 문자를 비교합니다. 다국적 순서는 문자를 나타내는 코드가 아닌 실제 그래픽 문자를 기준으로 하므로 일반 순서 검사가 작동하지 않습니다.

다음 보기에서는 지정 방법 파일에서 사용할 사용자 정의 조합 순서를 작성하는 것을 보여줍니다. 지정 방법 파일에 대한 자세한 내용은 353페이지의 “Sort/Merge 지정 방법 파일 사용”을 참조하십시오.

1.

```
/COLLATING_SEQUENCE=(SEQUENCE=ASCII, IGNORE=( "-" , " " ) )
```

IGNORE 옵션이 지정된 이 /COLLATING_SEQUENCE 한정자는 연결 구분을 수행하기 전에 다음 필드를 동등하게 비교합니다.

```
252-3412
252 3412
2523412
```

2.

```
/COLLATING_SEQUENCE=(SEQUENCE=( "A"- "L" , "LL" , "M"- "R" , "RR" , "S"- "Z" ) )
```

이 /COLLATING_SEQUENCE 한정자는 이중 문자 LL을 L과 M 사이의 단일 문자로 조합하고 이중 문자 RR을 R과 S 사이의 단일 문자로 조합합니다. 그렇지 않으면 이 이중 문자는 일반적인 사전순으로 표시됩니다. 기본적으로 이 사용자 정의 순서에서는 소문자 a-z와 같은 다른 문자를 정의하지 않습니다.

일괄 처리 작업으로 Sort 실행

일괄 처리 작업은 현재 세션과는 별도로 실행되는 프로그램이나 DCL 명령 프로시저입니다. 큰 파일을 정렬할 경우에는 정렬하는 데 시간이 걸리므로 일괄 처리 작업으로 정렬 작업을 실행하는 것이 좋습니다. 일괄 처리 작업과 명령 프로시저에 대한 자세한 내용은 16장, 13장 및 14장을 참조하십시오.

명령 프로시저

화면에 쓰는 것과 같이 명령 프로시저에 SORT 명령을 지정합니다. 기본 디렉토리에 정렬할 파일이 포함되어 있지 않으면 명령 프로시저에 기본 디렉토리를 명시적으로 설정하거나 명령 파일 지정 방법에 이 디렉토리를 포함시킵니다.

다음 보기에서는 DCL 명령 프로시저 SORTJOB.COM을 일괄 처리 작업으로 제출합니다. 명령 프로시저의 텍스트는 명령행 다음에 표시됩니다.

```
SUBMIT SORTJOB

! SORTJOB.COM
!
$ SET DEFAULT [USER.PER]    ! Set default to location of input files
$ SORT/KEY=(POSITION:10,SIZE:15) EMPLOYEE.LST BYNAME.LST
$ TYPE BYNAME.LST
$ EXIT
```

입력 레코드 포함

한 행 당 하나의 레코드를 SORT 명령 다음에 지정하여 일괄 처리 작업에 입력 레코드를 포함시킬 수 있습니다. 개별 정렬 레코드는 한 행보다 길 수 있습니다.

레코드의 터미널 입력에서와 같이 입력 파일 매개 변수를 SYS\$INPUT으로 지정합니다. /FORMAT 한정자를 사용하여 레코드 크기를 바이트로 지정하고 근사 파일 크기를 블록으로 지정합니다. 대략 80자로 된 여섯 행이 한 블록과 같습니다.

다음 보기에서는 명령 프로시저에 입력 레코드를 포함시키는 것을 보여 줍니다.

```
$ SUBMIT SORTJOB

! SORTJOB.COM
!
$ SET DEFAULT [USER.PER]
$ SORT/KEY=(POSITION:10,SIZE:15) -
SYS$INPUT-
/FORMAT=(RECORD_SIZE:24,FILE_SIZE:10) -
BYNAME.LST
$ DECK
BST 7828 MCMAHON JANE
ADM 7933 ROSENBERG HARRY
COM 8102 KNIGHT MARTHA
ANS 8042 BENTLEY PETER
BIO 7951 LOWELL FRANK
$ EOD
```


파일 병합

MERGE 명령은 최대 10개(고성능 **Sort/Merge** 유틸리티는 12개)의 정렬된 파일을 하나의 정렬된 출력 파일로 결합합니다. 형식이 같고 동일한 키 필드로 정렬된 입력 파일을 병합할 수 있습니다.

기본적으로 **Merge**에서는 입력 파일의 레코드 순서를 확인하여 제대로 정렬되어 있는지 확인합니다. **Merge**에서 순서를 확인하지 않도록 하려면 **/NOCHECK_SEQUENCE** 한정자를 지정합니다. **/CHECK_SEQUENCE** 한정자를 지정했는데 레코드 순서가 잘못된 경우(예: 입력 파일 중 하나를 정렬하지 않은 경우) **Merge**에서는 다음 오류를 보고합니다.

```
%SORT-W-BAD_ORDER, merge input is out of order
```

SORT 명령에서와 같은 한정자를 **MERGE** 명령에서 사용할 수 있지만 다음 두 가지 예외가 있습니다.

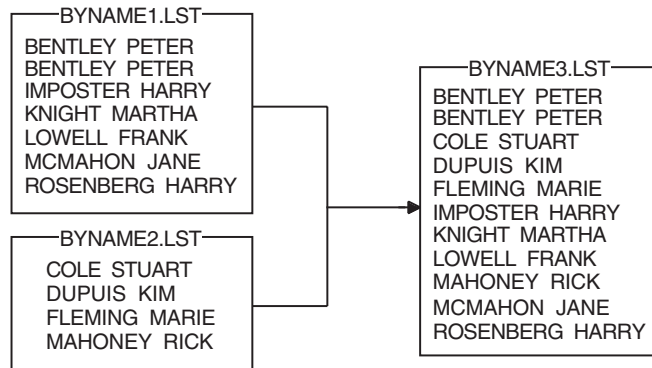
- 병합 작업에는 프로세스(**PROCESS**)를 지정할 수 없습니다.
- **/CHECK_SEQUENCE** 한정자는 병합 작업에만 사용됩니다.

다음 보기에서 **BYNAME1.LST**와 **BYNAME2.LST** 파일은 이미 직원 이름에 따라 오름차순으로 정렬되어 있습니다. 아래의 명령은 이 두 파일을 병합합니다.

```
$ MERGE BYNAME1.LST,BYNAME2.LST BYNAME3.LST
```

출력 파일 **BYNAME3.LST**는 다음 그림에서와 같이 두 파일 **BYNAME1.LST**와 **BYNAME2.LST**의 모든 레코드를 포함합니다.

그림 9-8



ZK-5062A-GE

정렬된 파일

특정 키를 사용하여 정렬된 파일을 병합하려면 **MERGE** 명령행의 **/KEY** 한정자와 동일한 키를 지정해야 합니다.

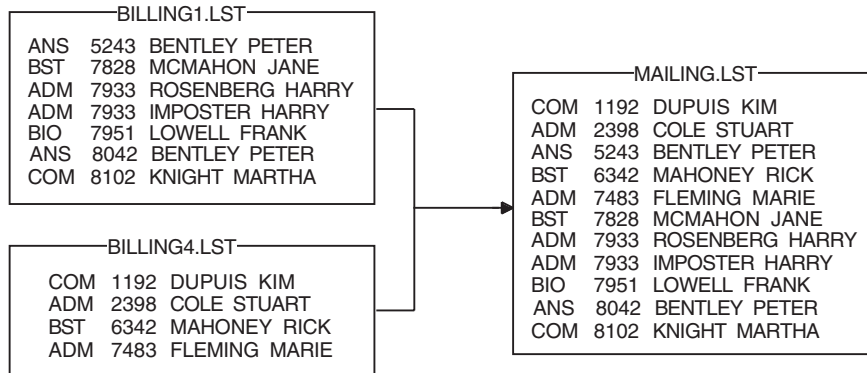
키를 지정하지 않으면 **Merge**에서는 350페이지의 “정렬된 파일”에 설명된 기본 키를 사용합니다.

다음 보기에서 **BILLING1.LST**와 **BILLING4.LST** 파일은 계정 번호 (**/KEY=POSITION:5,SIZE:4,DECIMAL**)에 따라 정렬됩니다. 이 파일을 출력 파일 **MAILING.LST**로 병합하려면 다음 명령행을 입력합니다.

```
$ MERGE/KEY=(POSITION:5,SIZE:4,DECIMAL) -
_$ BILLING1.LST,BILLING4.LST MAILING.LST
```

병합의 결과는 다음과 같습니다.

그림 9-9



ZK-5063A-GE

병합할 파일이 올바르게 정렬되어 있는 경우 `/NOCHECK_SEQUENCE` 한정자를 지정하여 순서 검사를 생략할 수 있습니다.

동일한 키 필드

정렬 작업에서와 마찬가지로 **Merge**는 입력 파일에 키 필드가 같은 레코드가 포함되어 있는 경우 입력 파일에 표시된 레코드의 순서와 항상 같게 유지하지는 않습니다. 키가 같은 레코드의 입력 순서를 유지하려면 **MERGE** 명령행에 `/STABLE` 한정자를 지정합니다. 키가 같은 레코드의 복사본 하나만을 유지하려면 `/NODUPPLICATES` 한정자를 지정합니다.

터미널에서 레코드 입력

정렬하거나 병합할 레코드가 파일에 있어야만 하는 것은 아닙니다. **SORT** 또는 **MERGE** 명령을 입력할 때 터미널에서 직접 레코드를 입력할 수 있습니다. 다음 표에서 그 절차를 설명합니다.

표 9-5

단계	작업
1	SORT 또는 MERGE 명령행에 입력 파일로 SYS\$INPUT 을 지정합니다. 입력 파일 한정자 /FORMAT 을 사용하여 가장 긴 레코드의 크기를 바이트로 지정하고 입력 파일의 근사 크기를 블록으로 지정합니다.
2	연속된 행에 입력 레코드를 입력합니다. Return 을 눌러 각 레코드를 종료합니다.
3	파일을 종료하려면 Ctrl/Z 를 누릅니다.

다음 보기에서는 정렬할 입력 레코드를 터미널에서 직접 입력하는 정렬 작업을 보여 줍니다.

```
$ SORT/KEY=(POSITION:8,SIZE:15) -  
_ $ SYS$INPUT/FORMAT=(RECORD_SIZE:24,FILE_SIZE:10) BYNAME.LST  
BST 7828 MCMAHON JANE  
ADM 7933 ROSENBERG HARRY  
COM 8102 KNIGHT MARTHA  
ANS 8042 BENTLEY PETER  
BIO 7951 LOWELL FRANK
```

이 명령 순서는 정렬된 레코드를 포함하는 출력 파일 **BYNAME.LST**를 만듭니다.

Sort/Merge 지정 방법 파일 사용

Sort/Merge에서는 **지정 방법 파일**로 정렬 정의를 유지하고 좀 더 복잡한 정렬 기준을 지정할 수 있습니다. 고성능 Sort/Merge 유틸리티에서는 지정 방법 파일을 지원하지 않습니다. 이 기능의 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다. 표준 편집기나 DCL CREATE 명령을 사용하여 지정 방법 파일을 만들 수 있습니다.

Sort/Merge 지정 방법 파일을 사용하여 다음을 수행할 수 있습니다.

- 정렬/병합 작업에 포함시킬 레코드 선택
- 출력 파일의 레코드 다시 형식화
- 조건부 키 또는 데이터 사용
- 여러 레코드 형식 지정
- 조합 순서 만들기 또는 수정
- 작업 파일 재지정
- 자주 사용하는 정렬/병합 작업 저장

지정 방법 파일을 작성한 후 /SPECIFICATION 한정자를 사용하여 파일 이름을 지정합니다. 지정 방법 파일의 기본 파일 유형은 .SRT입니다.

지정 방법 파일의 각 명령은 슬래시(/)로 시작해야 합니다. 명령이 두 행 이상 계속되는 경우 연결 문자를 사용할 필요가 없습니다.

주

지정 방법 파일에서 사용되는 여러 한정자는 Sort/Merge 명령행에서 사용되는 DCL 한정자와 유사합니다. 그러나 이러한 한정자의 구문은 다를 수 있습니다. 예를 들어, DCL 수준의 /KEY 한정자는 지정 방법 파일의 /KEY 한정자와 구문이 다릅니다. 지정 방법 파일 한정자에 대한 요약은 374페이지의 “지정 방법 파일 한정자”를 참조하십시오.

명령행에 지정하는 DCL 명령 한정자가 지정 방법 파일의 해당 항목보다 우선합니다. 예를 들어, DCL 명령행에 /KEY 한정자를 지정하면 Sort/Merge에서는 지정 방법 파일의 /KEY 절을 무시합니다.

일반적으로 지정 방법 파일에서 한정자를 지정할 때 따라야 하는 순서는 없습니다. 그러나 다음과 같은 경우에는 순서가 중요합니다.

- **NUMBER:n** 키 요소를 지정하지 않고 두 개 이상의 키 필드로 정렬하는 경우
- 출력 형식을 설명하는 경우
- 여러 레코드 유형을 정의하는 경우

/COLLATING_SEQUENCE 한정자에 **FOLD**, **MODIFICATION** 및 **IGNORE** 키워드를 지정하는 경우 **FOLD** 절 앞에 모든 **MODIFICATION** 및 **IGNORE** 절을 지정해야 합니다. /COLLATING_SEQUENCE 한정자에 대한 자세한 내용은 374페이지의 “지정 방법 파일 한정자”를 참조하십시오.

각 주석 행을 감탄 부호(!)로 시작하여 지정 방법 파일에 주석을 포함시킬 수 있습니다. DCL 명령행과는 달리 지정 방법 파일에서는 행을 연결하는 데 하이픈(-)을 사용할 필요가 없습니다.

보기

1. 다음은 음의 데이터와 양의 데이터를 오름차순으로 정렬하는 데 사용할 수 있는 지정 방법 파일의 보기입니다.

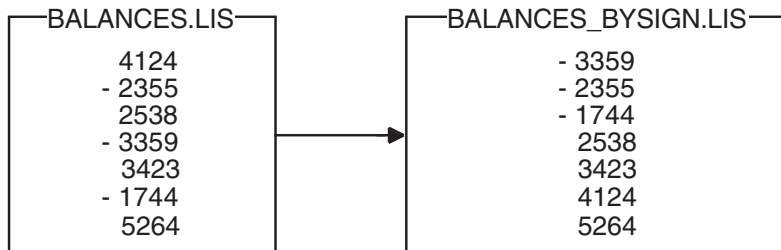
```
! Specification file for sorting negative and positive data
! in ascending order
!
/FIELD= (NAME=SIGN, POS:1, SIZ:1) [1]
/FIELD= (NAME=AMT, POS:2, SIZ:4) [2]
/CONDITION= (NAME=CHECK1, [3]
    TEST= (SIGN EQ "-") )
/CONDITION= (NAME=CHECK2, [4]
    TEST= (SIGN EQ " ") )
/INCLUDE= (CONDITION=CHECK1, [5]
    KEY= (AMT, DESCENDING) ,
    DATA=SIGN,
    DATA=AMT)
/INCLUDE= (CONDITION=CHECK2, [6]
    KEY= (AMT, ASCENDING) ,
    DATA=SIGN,
    DATA=AMT)
```

지정 방법 파일을 검토하면서 다음을 참조하십시오.

- [1] 이 명령행은 레코드의 바이트 1에서 시작하고 길이가 1바이트인 필드를 정의하고 이 필드에 **SIGN**이라는 이름을 할당합니다.
- [2] 이 명령행은 레코드의 바이트 2에서 시작하고 길이가 4바이트인 필드를 정의하고 이 필드에 **AMT**라는 이름을 할당합니다.
- [3] 조건문입니다. **SIGN** 바이트에 음수 기호(-)가 있으면 **CHECK1** 조건에 맞습니다.
- [4] 조건문입니다. **SIGN** 바이트가 비어 있으면 **CHECK2** 조건에 맞습니다.
- [5] 조건 **CHECK1**에 맞으면 레코드를 내림차순으로 정렬합니다.
- [6] 조건 **CHECK2**에 맞으면 레코드를 오름차순으로 정렬합니다.

그림 9-10에서는 **BALANCES.LIS**라는 입력 파일에서 이 지정 방법 파일을 사용한 결과를 보여 줍니다.

그림 9-10 지정 방법 파일을 사용한 출력



ZK-5448A-GE

2.

```

/FIELD= (NAME=RECORD_TYPE, POS:1, SIZ:1)      ! Record type, 1-byte
/FIELD= (NAME=PRICE, POS:2, SIZ:8)             ! Price, both files
/FIELD= (NAME=TAXES, POS:10, SIZ:5)            ! Taxes, both files
/FIELD= (NAME=STYLE_A, POS:15, SIZ:10)         ! Style, format A file
/FIELD= (NAME=STYLE_B, POS:20, SIZ:10)         ! Style, format B file
/FIELD= (NAME=ZIP_A, POS:25, SIZ:5)            ! Zip code, format A file
/FIELD= (NAME=ZIP_B, POS:15, SIZ:5)            ! Zip code, format B file
/CONDITION= (NAME=FORMAT_A,                   ! Condition test, format A
    TEST= (RECORD_TYPE EQ 'A') )
/CONDITION= (NAME=FORMAT_B,                   ! Condition test, format B
    TEST= (RECORD_TYPE EQ 'B') )
/INCLUDE= (CONDITION=FORMAT_A,                ! Output format, type A
    KEY=ZIP_A,

```

```

DATA=PRICE,
DATA=TAXES,
DATA=STYLE_A,
DATA=ZIP_A)
/INCLUDE=(CONDITION=FORMAT_B,           ! Output format, type B
KEY=ZIP_B,
DATA=PRICE,
DATA=TAXES,
DATA=STYLE_B,
DATA=ZIP_B)

```

이 보기에서는 한 부동산 중개소의 다른 두 지부의 입력 파일 두 개가 지정 방법 파일에 지정된 지시 사항에 따라 정렬됩니다. 처음 위치가 **A**로 시작하는 첫 번째 파일의 레코드 형식은 다음과 같습니다.

```

|A|PRICE|TAXES|STYLE|ZIP|
1  2      10    15    25

```

처음 위치가 **B**로 시작하고 예약된 스타일 및 우편 번호 필드가 있는 두 번째 파일의 레코드 형식은 다음과 같습니다.

```

|B|PRICE|TAXES|ZIP|STYLE|
1  2      10    15    20

```

레코드 형식 **A**의 우편 번호 필드에서 이 두 파일을 정렬하려면 우선 **/FIELD** 한정자로 두 레코드의 필드를 정의합니다. 그런 다음 **/CONDITION** 한정자에 두 유형의 레코드를 구분하기 위한 테스트를 지정합니다. 마지막으로 **/INCLUDE** 한정자는 출력에서 레코드 형식 유형 **B**를 레코드 형식 **A**로 변경합니다.

/INCLUDE 한정자에 키나 데이터 필드를 지정하는 경우 **/INCLUDE** 한정자에 정렬 작업을 위한 모든 키와 데이터 필드를 명시적으로 지정해야 합니다.

또한, 유형 **A** 또는 유형 **B**가 아닌 레코드는 정렬에서 제외됩니다.

3.

```

/COLLATING_SEQUENCE=(SEQUENCE=
("AN", "EB", "AR", "PR", "AY", "UN", "UL",
"UG", "EP", "CT", "OV", "EC", "0"- "9"),
MODIFICATION= (" " = "19"),
FOLD)

```

이 **/COLLATING_SEQUENCE** 한정자는 각 월에 연대순으로 고유한 값을 지정하는 사용자 정의 순서를 지정합니다. 예를 들어, 날짜에 따라 **SEMINAR.DAT**라는 파일을 정렬할 경우 파일 **SEMINAR.DAT**는 다음과 같이 설정됩니다.


```
16 NOV 1983   Communication Skills
05 APR 1984   Coping with Alcoholism
11 Jan '84    How to Be Assertive
12 OCT 1983   Improving Productivity
15 MAR 1984   Living with Your Teenager
08 FEB 1984   Single Parenting
07 Dec '83    Stress --- Causes and Cures
14 SEP 1983   Time Management
```

기본 키는 년도 필드이고 보조 키는 월 필드입니다. 월 필드는 숫자가 아니므로 월을 연대순으로 정렬하려면 사용자 고유의 조합 순서를 정의해야 합니다. 이렇게 하려면 각 월의 두 번째 글자 두 개에 따라(연대순으로) 정렬하여 각 월에 고유한 키 값을 지정합니다.

MODIFICATION 옵션은 어포스트로피(')가 19와 일치하도록 지정하므로 '83과 1984의 비교가 가능합니다. **FOLD** 옵션은 대문자와 소문자가 동일하게 처리되도록 지정합니다.

이 정렬 작업의 출력은 다음과 같습니다.

```
14 SEP 1983   Time Management
12 OCT 1983   Improving Productivity
16 NOV 1983   Communication Skills
07 Dec '83    Stress --- Causes and Cures
11 Jan '84    How to Be Assertive
08 FEB 1984   Single Parenting
15 MAR 1984   Living with Your Teenager
05 APR 1984   Coping with Alcoholism
```

사용자 정의 조합 순서를 만드는 다른 보기는 343페이지의 “조합 순서 지정”을 참조하십시오.

4.

```
/FIELD= (NAME=AGENT, POSITION:20, SIZE:15)
/CONDITION= (NAME=AGENCY,
             TEST= (AGENT EQ "Real-T Trust"
                   OR
                   AGENT EQ "Realty Trust") )
/DATA= (IF AGENCY THEN "Realty Trust" ELSE AGENT)
```

이 보기에서는 두 개의 부동산 파일이 정렬됩니다. 한 파일은 중개소를 **Real-T Trust**로 참조하고, 다른 파일은 같은 중개소를 **Realty Trust**로 참조합니다. **/CONDITION** 및 **/DATA** 한정자는 **Sort**가 정렬된 출력 파일의 **AGENT** 필드를 **Realty Trust**로 나열하도록 지시합니다.

5.

```

/FIELD= (NAME=ZIP, POSITION:60, SIZE:6)
/CONDITION= (NAME=LOCATION,
             TEST=(ZIP EQ "01863" ) )
/KEY=(IF LOCATION THEN 1
      ELSE 2)

```

이 보기에서는 우편 번호가 **01863**인 모든 레코드가 정렬된 출력 파일의 시작 부분에 표시됩니다. 조건 테스트는 **ZIP** 필드에 있으며 **/FIELD** 한정자에서 정의됩니다. 조건의 이름은 **LOCATION**입니다. 이 **/KEY** 한정자의 값 **1**과 **2**는 조건에 맞는 레코드와 맞지 않는 레코드에 대한 상대적인 순서를 나타냅니다.

6.

```

/FIELD= (NAME=ZIP, POSITION:60, SIZE:6)
/CONDITION= (NAME=LOCATION,
             TEST=(ZIP EQ "01863" ) )
/DATA=(IF LOCATION THEN "NORTH CHELMSFORD"
       ELSE "Outside district")

```

이 보기에서는 **/CONDITION** 한정자가 **01863** 우편 번호를 테스트합니다. **/DATA** 한정자는 테스트 결과에 따라 출력 레코드에 지역 이름 필드를 추가하도록 지정합니다.

7.

```

/FIELD= (NAME=FFLOAT, POS:1, SIZ:0, F_FLOATING)
/CONDITION= (NAME=CFFLOAT, TEST=(FFLOAT GE 100) )
/OMIT=(CONDITION=CFFLOAT)

```

이 보기에서는 필드 **FFLOAT**가 **/FIELD** 한정자에서 **F_FLOATING**으로 정의되므로 번호 **100**이 **F_FLOATING** 데이터 유형인 것으로 간주됩니다.

8.

```

/FIELD= (NAME=AGENT, POSITION:1, SIZE:5)
/FIELD= (NAME=ZIP, POSITION:6, SIZE:3)
/FIELD= (NAME=STYLE, POSITION:10, SIZE:5)
/FIELD= (NAME=CONDITION, POSITION:16, SIZE:9)
/FIELD= (NAME=PRICE, POSITION:26, SIZE:5)
/FIELD= (NAME=TAXES, POSITION:32, SIZE:5)
/DATA=PRICE
/DATA=" "
/DATA=TAXES
/DATA=" "
/DATA=STYLE
/DATA=" "
/DATA=ZIP
/DATA=" "
/DATA=AGENT

```

/FIELD 한정자는 형식이 다음과 같은 입력 파일의 레코드에 필드를 정의합니다.

AGENT ZIP STYLE CONDITION PRICE TAXES

/FIELD 한정자에 정의된 필드 이름을 사용하는 **/DATA** 한정자는 레코드를 다시 형식화하여 형식이 다음과 같은 출력 레코드를 만듭니다.

PRICE TAXES STYLE ZIP AGENT

정렬 또는 병합 작업 최적화

정렬 환경에 따라 정렬 또는 병합 작업의 효율성을 향상시킬 수 있는 여러 방법이 있습니다. 정렬 환경의 변수에 대한 정보를 보려면 **SORT** 또는 **MERGE** 명령에 **/STATISTICS** 한정자를 사용합니다.

통계 표시를 검토한 후 다음 절에서 제공된 최적화 옵션을 고려해 보십시오.

/STATISTICS 한정자와 함께 **SORT** 또는 **MERGE** 명령을 입력하면 다음과 유사한 출력이 나타납니다.

```
$ SORT/STATISTICS PAGEANT.LIS DOCUMENT.LIS
                                OpenVMS Sort/Merge Statistics

Records read:                   3 [1]      Input record length:       26
Records sorted:                 3          Internal length:           28
Records output:                 3          Output record length:      26
Working set extent: 16384 [2]      Sort tree size:                   42
Virtual memory:                 392        Number of initial runs:      0
Direct I/O:                     10         Maximum merge order:         0
Buffered I/O:                    11         Number of merge passes:         0
Page faults:                     158 [3]    Work file allocation:         0 [4]
Elapsed time: 00:00:00.54          Elapsed CPU:                   00:00:00.03 [5]
```

다음 필드를 검토하면서 다음을 참조하십시오.

[1] Records read

정렬 작업 동안 읽은 레코드 수를 나열합니다. 정렬 작업에서 레코드를 선택하여 생략하는 데 대한 내용은 362페이지의 “레코드 및 필드 생략”을 참조하십시오.

[2] Working set extent

정렬 작업을 수행하기 위해 예약된 블록 수를 나타냅니다. 작업 집합을 늘리는 데 대한 자세한 내용은 363페이지의 “작업 집합 범위 수정”을 참조하십시오.

[3] Page faults

운영 체제에서 프로세스의 부분을 실제 메모리에서 페이징 장치로 전송한 횟수를 보여줍니다. 페이징 제한에 대한 자세한 내용은 363페이지의 “작업 집합 범위 수정”을 참조하십시오.

[4] Work file allocation

작업 파일을 위해 예약된 디스크 공간을 보여 줍니다. 작업 파일에 대한 자세한 내용은 362페이지의 “작업 파일 할당”을 참조하십시오.

[5] Elapsed CPU

운영 체제에서 정렬 작업을 처리하는 데 걸린 CPU 시간을 보여 줍니다. 다른 정렬 방법을 선택하여 시간을 절약하는 데 대한 자세한 내용은 361페이지의 “정렬 프로세스”를 참조하십시오.

정렬 프로세스

Sort에서는 데이터 정렬을 위해 내부적으로 레코드, 태그, 주소, 인덱스 등 네 개의 프로세스를 정의합니다. 고성능 Sort/Merge 유틸리티는 레코드 프로세스만 지원합니다. 태그, 주소 및 인덱스 프로세스 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다. RECORD가 기본 프로세스입니다. 선택하는 프로세스 유형에 따라 저장 요구 사항 및 정렬 작업의 성능에 영향을 미칩니다. 다른 정렬 프로세스에 대한 내용은 342페이지의 “정렬 프로세스”를 참조하십시오.

정렬 프로세스를 선택하기 전에 다음 사항을 고려하십시오.

- 출력 파일 사용 방법
 - 레코드 및 태그 정렬에서는 정렬된 전체 레코드를 포함하는 파일을 생성하므로 다시 정렬된 파일을 바로 사용할 수 있습니다.
 - 주소 및 인덱스로 정렬된 출력 파일 모두는 Pascal, Fortran, MACRO 또는 C와 같은 프로그래밍 언어로 작성한 프로그램에서 처리할 수 있습니다.
 - 주소 정렬에서는 입력 파일의 레코드에 대한 포인터의 출력 파일을 만듭니다. 이 목록은 이진 RFA와 파일 번호(여러 입력 파일을 정렬할 경우)로 구성됩니다. 프로그램은 이 포인터를 사용하여 레코드를 액세스합니다.
 - 인덱스 정렬에서는 RFA와 키 필드 모두와 파일 번호(여러 파일 정렬 시)를 포함하는 출력 파일을 만듭니다. 이 키 필드의 형식은 입력 파일과 같습니다. 프로그램에서 미래의 처리 중에 결정할 키 필드 내용을 필요로 하는 경우, 주소 정렬이 아니라 인덱스 정렬을 선택합니다.

한 파일의 레코드를 다른 용도를 위해 여러 방법으로 다시 정렬해야 할 경우 주소 또는 인덱스 정렬의 여러 출력 파일을 저장합니다. 이러한 출력 파일을 사용하여 원하는 정렬 순서로 주 파일의 레코드를 액세스합니다.

- 정렬에 사용할 수 있는 임시 저장 공간

태그 정렬은 레코드 정렬보다 임시 공간을 적게 사용합니다. **레코드 정렬**에서는 정렬하는 동안 레코드를 그대로 유지하므로 파일이 클 경우 훨씬 많은 작업 공간을 사용합니다. 주소 및 인덱스 정렬은 임시 저장 공간을 거의 사용하지 않습니다.

- 사용된 입력 및 출력 장치 유형

레코드 정렬은 카드, 자기 테이프 및 디스크에서 입력을 받을 수 있는 유일한 프로세스입니다. 태그 및 레코드 정렬의 출력은 모든 출력 장치로 이동할 수 있습니다. 주소 및 인덱스 정렬의 출력은 이진 데이터를 수락하는 장치로 이동해야 합니다.

- 속도 차이

작업의 일정 지점에서 정렬된 레코드를 검색할 계획이라면 일반적으로 레코드 정렬이 가장 빠른 프로세스입니다. 그렇지 않은 경우 주소 및 인덱스 정렬이 가장 빠른 프로세스입니다.

레코드 및 필드 생략

지정 방법 파일에서 /CONDITION, /INCLUDE 및 /OMIT 한정자를 사용하여 출력 파일에 필요한 레코드만 처리하면 정렬 효율성을 향상시킬 수 있습니다. 고성능 Sort/Merge 유틸리티는 지정 방법 파일을 지원하지 않습니다. 이 기능의 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다. 또한 지정 방법 파일 한정자를 사용하여 출력 파일에서 필요 없는 필드를 생략하고 레코드를 다시 형식화할 수 있습니다. 이러한 한정자는 명령행 한정자로 사용할 수 없습니다.

작업 파일 할당

정렬 작업 동안 입력 파일의 레코드를 메모리로 읽어 들입니다. 할당된 메모리에서 모든 레코드를 보관할 수 없는 경우 Sort에서는 정렬된 데이터를 하나 이상의 임시 작업 파일로 전송합니다. Merge에서는 작업 파일을 사용하지 않습니다.

작업 파일 수를 변경하거나 작업 파일을 특정 장치에 할당하여 정렬 효율성을 향상시킬 수 있습니다.

- Sort 명령행 한정자 /WORK_FILES=*n*은 할당된 작업 파일 수보다 우선합니다.
- 일반적으로 Sort에서는 작업 파일을 장치 SYS\$SCRATCH에 저장하며 임의의 순서로 작업 파일을 액세스합니다. 다음 두 가지 방법으로 작업 파일을 특정 장치에 할당할 수 있습니다.

- 지정 방법 파일에서 `/WORK_FILES=(device,...)` 한정자는 작업 파일을 지정한 장치에 저장합니다. 지정 방법 파일에서 `/WORK_FILES` 한정자를 사용하는 데 대한 자세한 내용은 374페이지의 “지정 방법 파일 한정자”를 참조하십시오.
- 지정 방법 파일을 사용하지 않는 경우 DCL 명령 `ASSIGN`을 사용하여 작업 파일을 특정 장치에 할당할 수 있습니다.

`Sort`에서는 `SORTWORKn` 논리 이름을 사용하여 작업 파일에 대한 사용자 지정 장치 이름을 식별합니다. 여기서 `n`은 0에서 9까지의 값입니다. 고성능 `Sort/Merge` 유틸리티인 경우 `n`은 0에서 254까지의 값입니다. `SORTWORKn` 논리를 다음과 같이 정의합니다.

`ASSIGN device: SORTWORKn`

예를 들면 다음과 같습니다.

```
$ ASSIGN WORK$2: SORTWORK1
$ ASSIGN WORK$3: SORTWORK2
```

이 보기에서는 `SORTWORK1`을 장치 `WORK$2:`로 정의하고 `SORTWORK2`를 장치 `WORK$3:`으로 정의합니다. 논리 이름에 대한 자세한 내용은 11장을 참조하십시오.

작업 파일을 장치에 할당할 때 다음을 고려하십시오.

- 사용할 수 있는 가장 빠른 장치(예: 디스크와 같은 임의 액세스, 대용량 저장 장치)에 작업 파일을 할당합니다.
- 활동이 가장 적고 사용 가능한 공간이 가장 많은 장치를 선택합니다.
- 입력 및 출력의 중복을 최소화하려면 각각의 작업 파일을 다른 물리적 장치에 할당합니다.

작업 집합 범위 수정

`Sort`에서 작업 파일을 필요로 하는 경우(예: 큰 파일을 정렬할 경우) 더 큰 작업 집합이 정렬 효율성을 향상시킬 수 있습니다. 그러나 시스템이 과도하게 사용되는 경우 작업 집합 범위의 모든 페이지를 프로세스에 할당하지 못할 수 있습니다. 이렇게 되면 운영 체제에서 실제 메모리와 페이징 장치의 메모리 간의 프로세스 일부를 전송할 때 발생하는 페이징이 일어날 수 있습니다. 프로세스의 활성화된 부분만 실제 메모리에 남게 됩니다. 과도한 페이징을 방지하기 위해 프로세스의 작업 집합 범위를 줄일 수 있습니다. 작업 집합 범위를 줄이면 `SET WORKING_SET` 명령 사용합니다.

정렬/병합 한정자 요약

다음 목록에서는 **SORT** 및 **MERGE** 명령과 함께 사용되는 명령 한정자를 설명합니다. 명령 한정자를 사용하려면 **SORT** 또는 **MERGE** 명령 바로 다음에 한정자를 포함시킵니다.

/[NO]CHECK_SEQUENCE

MERGE 명령에만 적용됩니다. **MERGE** 입력 파일의 레코드 순서를 확인합니다. **Merge**에서는 기본적으로 레코드 순서를 확인합니다.

/CHECK_SEQUENCE 한정자는 하나 이상의 파일의 레코드(최대 10개, 고성능 **Sort/Merge** 유틸리티의 경우 최대 12개 지원)이 정렬되었는지 여부를 확인합니다. 레코드는 사용자가 지정해야 하는 출력 파일로 계속 보내집니다. 레코드가 전체 레코드가 아닌 키 필드에 따라 정렬되는지 여부를 확인하려면 순서 요청과 함께 키 정보를 지정해야 합니다.

Merge에서 레코드 순서를 확인하지 않도록 하려면 **/NOCHECK_SEQUENCE** 한정자를 사용합니다.

보기

```
$ MERGE/KEY=(SIZE:4, POSITION:3) /NOCHECK_SEQUENCE -  
_$_ PRICE1.DAT, PRICE2.DAT PRICE.LIS
```

이 보기에서 **/NOCHECK_SEQUENCE** 한정자는 입력 파일 **PRICE1.DAT**와 **PRICE2.DAT**의 순서를 확인하지 않도록 지정합니다.

/COLLATING_SEQUENCE=sequence

문자 키 필드에 대한 미리 정의된 세 가지 조합 순서 중 하나를 선택하거나 문자 키를 비교하는 데 사용할 **NCS** 조합 순서의 이름을 지정합니다. 고성능 **Sort/Merge** 유틸리티는 **NCS** 조합 순서를 지원하지 않습니다. **NCS** 조합 순서에 대한 지원은 다음 **OpenVMS Alpha** 릴리즈로 연기되었습니다. **Sort**는 **ASCII**(기본값), **EBCDIC** 또는 다국적 순서로 문자를 정렬합니다.

보기

```
$ SORT/COLLATING_SEQUENCE=MULTINATIONAL -  
_$_ NAMES.DAT, NOM.DAT LIST.LIS
```

이 **SORT** 명령은 입력 파일 **NAMES.DAT**와 **NOM.DAT**를 다국적 조합 순서에 따라 정렬하여 출력 파일 **LIST.LIS**를 만듭니다.

/[NO]DUPLICATES

기본적으로 **Sort**에서는 중복된 키가 있는 여러 레코드를 모두 유지합니다.

/NODUPLICATES 한정자는 중복된 키가 있는 여러 레코드 중 하나만 제외하고 모두 제거합니다. 유지되는 레코드는 입력 파일에 표시된 순서와 동일하게 표시되지 않을 수 있습니다. 유지할 중복된 레코드를 지정하려면 프로그램 수준에서 **Sort**를 호출하고 등가 키 루틴을 지정합니다.

/STABLE과 **/NODUPLICATES** 한정자는 서로 배타적입니다.

보기

```
$ SORT/KEY=(POSITION:3,SIZE:5,DECIMAL)/NODUPLICATES -
_$ ACCT1,ACCT2 ACCT.LIS
```

이 **SORT** 명령은 두 개의 입력 파일을 제공된 키에 따라 정렬하고 등가 키의 여러 레코드 중 하나를 제외하고 모두 제거합니다.

/KEY=(POSITION:n,SIZE:n[,field,...])

위치, 크기, 정렬 순서(**ASCENDING** 또는 **DESCENDING**), 우선 순위(**NUMBER:n**) 및 데이터 유형(**character**, **binary**, **h_floating** 등)을 비롯한 키 필드를 설명합니다. 기본적으로 **Sort**는 문자 데이터를 사용하여 전체 레코드를 오름차순으로 정렬합니다.

/KEY 한정자에 대한 자세한 내용은 333페이지의 “키 정의”를 참조하십시오.

/PROCESS=type

SORT 명령에만 적용됩니다. 내부 정렬 프로세스를 정의합니다. **/PROCESS** 한정자에서 레코드, 태그, 주소, 인덱스 등 네 가지 프로세스 중 하나를 선택할 수 있습니다. 고성능 **Sort/Merge** 유틸리티는 레코드 프로세스만 지원합니다. 태그, 주소 및 인덱스 프로세스의 구현은 다음 **OpenVMS Alpha** 릴리즈로 연기되었습니다.

/PROCESS 한정자에 대한 자세한 내용은 342페이지의 “정렬 프로세스”를 참조하십시오.

보기

```
$ SORT/KEY=(POS:40,SIZ:2,DESC)/PROCESS=TAG YRENDAVG.DAT -
_$ DESCYRAVG.LIS
```

이 **Sort** 작업은 태그 정렬 프로세스를 사용하여 출력 파일 **DESCYRAVG.LIS**를 만듭니다.

/SPECIFICATION=filespec

고성능 Sort/Merge 유틸리티는 이 한정자를 지원하지 않습니다. 이 기능의 구현은 다음 OpenVMS Alpha 릴리즈로 연기되었습니다.

정렬 또는 병합 작업에서 사용될 **Sort** 또는 **Merge** 지정 방법 파일을 식별합니다. 기본 지정 방법 파일 유형은 **.SRT**입니다.

지정 방법 파일 사용에 대한 자세한 내용은 353페이지의 “Sort/Merge 지정 방법 파일 사용” 및 374페이지의 “지정 방법 파일 한정자”를 참조하십시오.

/[NO]STABLE

기본적으로 등가 키가 있는 레코드는 입력 파일에 표시되는 순서대로 출력 파일에 위치한다고 보장할 수 없습니다. **/STABLE** 한정자는 해당 순서대로 레코드를 유지합니다.

/STABLE과 **/NODUPPLICATES** 한정자는 서로 배타적입니다.

보기

```
$ SORT/KEY=(POS:1,SIZ:5,DECIMAL)/STABLE PRICESA.DAT, -  
_ $ PRICESB.DAT,PRICESC.DAT SUMMARY.LIS
```

이 Sort 작업에서는 **PRICESA.DAT**의 등가 키가 있는 레코드, **PRICESB.DAT**의 해당 레코드와 **PRICESC.DAT**의 해당 레코드 순으로 나열됩니다.

/[NO]STATISTICS

최적화에 사용할 수 있는 **SYSS\$OUTPUT**에 통계 요약을 표시합니다. 이러한 통계를 파일에 저장하려면 다음 명령을 사용합니다.

```
$ DEFINE/USER SYSS$ERROR output-file
```

통계 요약에는 다음 정보가 포함됩니다.

표 9-6

통계	설명
Records read	Sort 또는 Merge에서 읽은 레코드 수입니다.
Records sorted	Sort를 사용하여 처리된 레코드 수입니다. 지정 방법 파일을 사용하여 정렬 또는 병합 작업에 대한 특정 레코드만 선택한 경우 읽은 레코드 수보다 이 수가 작을 수 있습니다.
Records output	출력 파일에 쓰여진 레코드 수입니다. /NODUPPLICATES를 선택했거나 출력 레코드가 쓰여질 때 입출력 오류가 발생한 경우 정렬된 레코드 수보다 이 수가 작을 수 있습니다.
Working set extent	프로세스 작업 집합 범위의 페이지 수입니다. 이 값은 정렬 데이터 구조의 크기에 대한 상한값으로 사용됩니다. 이 값을 조정하는 것은 Sort 작업의 효율성을 향상시키는 한 가지 방법입니다.
Virtual memory	데이터를 보관할 Sort 이미지에 추가된 가상 메모리의 페이지 수입니다.
Direct I/O + buffered I/O	이 합계는 데이터를 읽고 쓰는 데 필요한 입출력 이동 수입니다. 이 합계가 작을수록 정렬 작업의 효율성은 향상됩니다.
Page faults	데이터가 메모리에 얼마나 적합한지를 나타냅니다. 페이지 오류 수가 클수록 정렬 작업 효율성은 낮습니다.
Elapsed time	정렬 또는 병합 작업에 사용된 시간, 분, 초 및 1/100초 단위의 총 벽시계 시간입니다.
Input record length	이 값은 사용자가 제공하지 않는 경우 OpenVMS RMS(레코드 관리 서비스)로부터 얻습니다.
Internal length	내부 형식 노드의 바이트 단위 크기입니다. 여기에는 키, 데이터, 길이를 저장할 단어, 레코드 파일 주소(RFA) 및 변환된 키가 포함됩니다.
Output record length	출력 레코드의 길이입니다. 이 길이는 입력 레코드 길이, 정렬 프로세스 및 요청한 레코드 다시 형식화에 따라 계산됩니다.
Sort tree size	Sort 내부 데이터 구조에 맞는 레코드 수입니다.
Number of initial runs	데이터가 메모리에 맞는 정도를 나타내는 하나의 표시입니다.
Maximum merge order	한 번에 병합되는 정렬된 문자열의 최대 수입니다.

표 9-6 (계속)

통계	설명
Number of merge passes	하나의 정렬된 출력 문자열이 만들어질 때까지 Sort 유틸리티에서 문자열을 병합하는 횟수입니다. 초기 실행 수와 병합 통과 수는 데이터가 메모리에 적합한 정도를 나타냅니다. 이 수가 높을수록 작업 집합 크기가 데이터를 포함하는 위치에서 멀고 정렬하는 데 시간이 오래 걸립니다.
Work file allocation	작업 파일에 사용된 블록 수입니다. 두 개 이상의 병합 통과가 필요한 경우 이 크기는 입력 파일 할당 크기의 2배 정도입니다.
Elapsed CPU	정렬 작업에서 사용한 CPU 시간입니다. 여기에는 입출력 작업이 완료되는 동안 대기하는 시간이나 다른 프로세스가 실행하는 동안 대기하는 시간은 포함되지 않습니다.

보기

```
$ SORT/STATISTICS PRICE1.DAT,PRICE2.DAT PRICE.LIS
```

SORT /STATISTICS 명령은 다음 통계를 표시합니다.

```
OpenVMS Sort/Merge Statistics
```

```
Records read: 793           Input record length: 80
Records sorted: 793        Internal length: 80
Records output: 793        Output record length: 80
Working set extent: 100     Sort tree size: 412
Virtual memory: 433        Number of initial runs: 2
Direct I/O: 22             Maximum merge order: 2
Buffered I/O: 9            Number of merge passes: 1
Page faults: 3418          Work file allocation: 114
Elapsed time: 00:00:05.98   Elapsed CPU: 00:00:03.63
```

/WORK_FILES[=n]

SORT 명령에만 적용됩니다. 각 작업 파일 크기를 줄이려면 1에서 10 사이(1과 10 포함, 고성능 **Sort/Merge** 유틸리티의 경우 최대 255까지 지원)의 수로 **Sort** 작업 파일의 수를 늘립니다. 사용할 수 있는 디스크가 너무 적거나 작업 파일로 꽉 찬 경우 파일 수를 늘리면 **Sort** 작업의 효율성을 향상시킬 수 있습니다.

Sort는 필요할 때까지 작업 파일을 만들지 않습니다. **Sort**에서 작업 파일이 필요한 경우 기본적으로 **SYS\$SCRATCH** 디렉토리에 있는 두 개의 작업 파일(**SORTWORK0**, **SORTWORK1**)을 만듭니다.

보기

```
$ ASSIGN DRA5: SORTWORK0
$ ASSIGN DB0: SORTWORK1
$ ASSIGN DB1: SORTWORK2
$ SORT/KEY=(POS:1,SIZ:80)/WORK_FILES=3 -
_$ STATS1,STATS2,STATS3,STATS4 SUMMARY.LIS
```

이 **Sort** 작업의 입력 파일은 큰 파일이므로 세 개의 작업 파일을 지정하면 정렬 작업의 효율성을 향상시킵니다.

또한 디렉토리 이름을 포함시켜 장치의 특정 디렉토리에 작업 파일을 할당할 수도 있습니다. 예를 들어, **SORTWORK0**을 **DRA5**의 **[WORKSPACE]** 디렉토리에 할당하려면 다음 명령을 입력합니다.

```
$ ASSIGN DRA5:[WORKSPACE] SORTWORK0
```

입력 파일 한정자

다음 입력 한정자는 **SORT** 또는 **MERGE** 명령행의 입력 파일 지정 방법 바로 다음에 포함시켜야 합니다.

/FORMAT=(RECORD_SIZE:n,FILE_SIZE:n)

입력 파일 특성을 정의합니다. 이 한정자를 사용하여 레코드 또는 파일 크기를 지정하거나 무시할 수 있습니다. 이 한정자는 **Sort** 또는 **Merge** 명령행에서 입력 파일 지정 방법 바로 다음에 지정해야 합니다.

Sort는 입력 파일 크기 정보를 사용하여 필요한 메모리 양과 정렬 작업의 작업 파일 크기를 결정합니다. 파일 크기를 모를 경우(예: 디스크나 표준 **ANSI** 자기 테이프에 상주하지 않는 파일을 정렬할 경우), **Sort**에서는 상당히 큰 파일 크기를 가정합니다.

다음 한정자 값을 지정합니다.

표 9-7

RECORD_SIZE:n	입력 파일의 가장 긴 레코드 길이(LRL)를 바이트 단위로 지정합니다. 지정할 수 있는 가장 긴 레코드 길이는 파일 조직에 따라 다릅니다.
순차적	32,767
상대적	16,383
인덱스 순차적	16,362
	이 값은 고정 길이 제어(VFC) 형식인 가변 레코드의 제어 바이트를 포함합니다.
FILE_SIZE:n	입력 파일 크기를 블록으로 지정합니다. 허용되는 최대 파일 크기는 4,294,967,295블록입니다.

FORMAT은 출력 파일 한정자로도 사용할 수 있습니다.
자세한 내용은 370페이지의 “출력 파일 한정자”를 참조하십시오.

보기

```
$ SORT/KEY=(POS:40,SIZ:2,DESC) -
  _$CRA0:YRENDAVG.DAT/FORMAT=(RECORD_SIZE:41,FILE_SIZE:3) - _$DESCYRAVG.LIS
```

입력 파일 YRENDAVG.DAT는 디스크 장치나 ANSI 자기 테이프에 상주하지 않으므로 /FORMAT 한정자로 파일 조직을 설명해야 합니다.

출력 파일 한정자

다음 출력 한정자를 SORT 및 MERGE 명령과 함께 사용할 수 있습니다. 출력 파일 한정자를 사용하려면 SORT 또는 MERGE 명령행의 출력 파일 지정 방법 바로 다음에 한정자를 포함시킵니다.

/ALLOCATION=n

최적화를 위해 출력 파일에 미리 할당할 블록 수를 1과 4,294,967,295 사이에서 지정합니다. 출력 파일 할당이 전체 입력 파일 할당과 크게 다를 경우(예: 데이터를 다시 형식화하거나 레코드를 생략한 경우) 이 한정자를 사용합니다.

/ALLOCATION 한정자는 /CONTIGUOUS 한정자가 사용된 경우에 필요합니다.

보기

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT -
_$ SUMMARY.LIS/ALLOCATION=1000/CONTIGUOUS
```

이 SORT 명령은 출력 파일 SUMMARY.LIS에 1000개의 연속 블록을 할당합니다.

/BUCKET_SIZE=n

최적화를 위해 상대적 및 인덱스 순차 출력 디스크 파일에서 사용할 OpenVMS RMS 버킷 크기(버킷 당 512바이트 블록 수)를 지정합니다. 1에서 32까지의 값이 허용됩니다.

출력 파일 조직이 입력 파일과 같을 경우 기본값은 첫 번째 입력 파일의 버킷 크기와 같습니다. 출력 파일 조직이 다른 경우 기본값은 1입니다.

보기

```
$ SORT/KEY=(POS:1,SIZ:80) STATS1.DAT,STATS2.DAT -
_$ SUMMARY.LIS/BUCKET_SIZE=16/RELATIVE
```

이 SORT 명령은 버킷 크기가 16이고 상대적 조직의 출력 파일 SUMMARY.LIS를 만듭니다.

/CONTIGUOUS

액세스 시간을 줄이기 위해 출력 파일이 연속 디스크 블록에 저장되도록 요청합니다. /ALLOCATION 한정자와 함께 사용해야 합니다. 기본적으로 Sort/Merge는 출력 파일에 연속 디스크 블록을 할당하지 않습니다.

보기

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT - _$
SUMMARY.LIS/ALLOCATION=1000/CONTIGUOUS
```

이 SORT 명령은 출력 파일 SUMMARY.LIS에 1,000개의 연속 블록을 할당합니다.

/FORMAT=(type:n[,...])

출력 파일 레코드 형식이 입력 파일 형식과 다른 경우 출력 파일 레코드 형식 (FIXED:n, VARIABLE:n, or CONTROLLED:n)을 지정합니다. 파일 레코드의 크기 (SIZE:n)나 블록 크기(BLOCK_SIZE:n)도 지정할 수 있습니다.

Sort 작업이 레코드 또는 태그 정렬인 경우 기본 출력 레코드 형식은 첫 번째 입력 파일 레코드 형식과 같습니다. Sort 작업이 주소 또는 인덱스 정렬인 경우 기본 출력 레코드 형식은 고정 레코드 형식입니다. 입력 파일에 다른 레코드 서식이 있는 경우 Sort는 입력 파일의 가장 긴 레코드를 포함하기에 충분한 출력 레코드 크기를 제공합니다.

다음과 같은 한정자 값을 지정할 수 있습니다.

표 9-8

BLOCK_SIZE: <i>n</i>	출력 파일을 자기 테이프에 보낸 경우 출력 파일의 블록 크기를 바이트 단위로 지정합니다. 입력 파일이 테이프 파일인 경우 출력 파일의 블록 크기의 기본값은 입력 파일의 블록 크기로 설정됩니다. 그렇지 않은 경우 출력 파일 블록 크기의 기본값은 테이프가 마운트될 때 사용된 크기로 설정됩니다. <i>n</i> 에 사용 가능한 값은 20에서 65,532 사이지만 다른 Compaq 시스템과 올바른 데이터를 교환하려면 블록 사이즈를 512바이트 이하로 지정해야 합니다. Compaq에서 만들지 않은 시스템과의 호환성을 위해 블록 크기는 2,048바이트를 초과하지 않는 것이 좋습니다.
CONTROLLED: <i>n</i>	출력 파일의 고정 길이 제어(VFC) 레코드와 함께 변수를 지정합니다.
FIXED: <i>n</i>	출력 파일의 고정 길이 레코드를 지정합니다.
SIZE: <i>n</i>	VFC(CONTROLLED) 레코드의 고정 부분의 크기(바이트)를 최대 255바이트까지 지정합니다. SIZE 를 지정하지 않으면 기본값은 첫 번째 입력 파일의 고정 부분의 크기가 됩니다. 이 크기를 0으로 지정하면, OpenVMS RMS에서 이 값을 2바이트로 설정합니다.
VARIABLE: <i>n</i>	출력 파일의 가변 길이 레코드를 지정합니다.

모든 한정자 값에 대해 출력 레코드의 최대 레코드 크기(바이트)로 *n*을 지정할 수도 있습니다. 사용할 수 있는 최대 레코드 크기는 다음과 같이 파일 조직에 따라 다릅니다.

순차 파일	32,767
상대적 파일	16,383
인덱스 순차 파일	16,362

이 최대 레코드 크기는 고정 길이 제어(VFC) 형식으로 가변 레코드의 제어 바이트를 포함합니다.

보기

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT SUMMARY.LIS/FORMAT=FIXED:80
```

입력 파일 **STATS.DAT**는 길이가 80바이트인 가변 길이 레코드로 구성됩니다.
/FORMAT 한정자는 출력 파일 **SUMMARY.LIS**가 고정 길이 레코드로 구성되도록 지정합니다.

/INDEXED_SEQUENTIAL

출력 파일의 파일 조직을 인덱스 순차적으로 정의합니다. 출력 파일은 이미 존재하며 비어 있어야 합니다. 또한 **/OVERLAY** 한정자를 사용하여 이 빈 파일이 정렬된 레코드와 오버레이되도록 지정해야 합니다.

보기

```
$ CREATE/FDL=NEW.FDL AVERAGE.DAT
$ SORT/KEY=(POS:1,SIZ:80) DATA.DAT,STATS.DAT -
_$ AVERAGE.DAT/INDEXED_SEQUENTIAL/OVERLAY
```

CREATE/FDL 명령은 빈 파일 **AVERAGE.DAT**를 만듭니다. **SORT** 명령은 출력 파일에 인덱스 순차 조직이 있으며 이 출력 파일이 빈 파일 **AVERAGE.DAT**에 쓰여지도록 지정합니다.

/OVERLAY

출력 파일이 오버레이되거나 쓰여지는 기존의 빈 파일을 지정합니다.
/INDEXED_SEQUENTIAL 한정자를 사용하는 경우 **/OVERLAY** 한정자가 필요합니다.

입력 파일 조직이 인덱스 순차인 경우 출력 파일은 이미 존재하며 비어 있어야 합니다. 출력 파일이 비어 있지 않는 경우 **/OVERLAY**에서는 해당 파일에 덮어쓰지 않습니다. 대신, 기존 출력 파일에 정렬 결과를 추가합니다.

CREATE/FDL 유틸리티를 사용하여 빈 데이터 파일을 만들 수 있습니다. 빈 파일을 만들 때 지정하는 모든 속성은 **Sort** 출력 파일의 속성이 됩니다.

보기

```
$ CREATE/FDL=NEW.FDL AVERAGE.DAT
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT AVERAGE.DAT/OVERLAY
```

FDL 파일 **NEW.FDL**은 **AVERAGE.DAT** 파일의 특수 속성을 지정합니다. **Sort**에서 해당 파일에 출력을 쓸 때 만들어지는 **Sort** 출력 파일은 **FDL** 파일에는 지정한 속성이 있습니다.

/RELATIVE

출력 파일의 파일 조직을 상대적 파일로 정의합니다.

보기

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT SUMMARY.LIS/RELATIVE
```

입력 파일 **STATS.DAT**가 상대적 파일이 아니고 출력 파일 **SUMMARY.LIS**가 상대적 파일이 되므로 **/RELATIVE**는 출력 파일 지정 방법을 한정합니다.

/SEQUENTIAL

출력 파일의 파일 조직을 순차적으로 정의합니다. 이 구성은 주소 및 인덱스 정렬 작업의 기본값입니다. 레코드 및 태그 정렬 작업의 기본값은 첫 번째 입력 파일의 조직입니다.

보기

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT SUMMARY.LIS/SEQUENTIAL
```

입력 파일 **STATS.DAT**는 순차 파일이 아니고 출력 파일 **SUMMARY.LIS**이 순차 파일이 되므로 **/SEQUENTIAL**에서 출력 파일 지정 방법을 한정합니다.

지정 방법 파일 한정자

다음 한정자는 지정 방법 파일에서 사용할 수 있습니다. 고성능 **Sort/Merge** 유틸리티는 지정 방법 파일을 지원하지 않습니다. 이 기능의 구현은 다음 **OpenVMS Alpha** 릴리즈로 연기되었습니다. 이러한 한정자는 **Sort/Merge** 지정 방법 파일 내에서만 유효합니다.

/CDD_PATH_NAME="cdd-path-name"

CDD/Repository 명령을 사용하여 **Common Data Dictionary(CDD/Plus)**와 함께 사용하도록 정의된 필드와 속성을 식별합니다. 식별된 필드는 나중에 **/KEY**, **/CONDITION**, **/INCLUDE** 또는 **/OMIT**와 같은 다른 지정 방법 파일 한정자에서 사용할 수 있습니다.

/CDD_PATH_NAME은 **/FIELD** 문 대신 또는 **/FIELD** 문과 함께 사용할 수 있습니다.

"cdd-path-name" 값은 **CDD/Plus** 내의 **CDD/Plus** 레코드 정의입니다. 시스템에서 **CDD/Plus**를 설치한 경우에만 **/CDD_PATH_NAME** 한정자를 사용할 수 있습니다.

보기

```
/CDD_PATH_NAME="employee"
```

/CDD_PATH_NAME 한정자는 CDD/Plus에서 이전에 정의된 직원 레코드를 식별합니다.

/[NO]CHECK_SEQUENCE

MERGE 명령에만 적용됩니다. 입력 파일의 레코드 순서를 확인할지 여부를 지정합니다. 기본적으로 Merge는 레코드 순서를 확인합니다.

보기

```
/NOCHECK_SEQUENCE
```

/NOCHECK_SEQUENCE 한정자는 Merge 유틸리티의 기본 동작보다 우선합니다.

```
/COLLATING_SEQUENCE=(SEQUENCE=sequence-type  
[,MODIFICATION=("char1" operator "char2")]  
[,IGNORE=character or character range,...]  
[,FOLD]  
[,[,NO]TIE_BREAK])
```

문자 키 필드에 대해 세 가지 미리 정의된 조합 순서(ASCII, EBCDIC 또는 다국적) 중 하나 또는 사용자 정의 순서를 지정합니다. 미리 정의된 조합 순서나 이전에 정의된 사용자 정의 순서를 수정할 수 있게 합니다.

SCII, EBCDIC 및 다국적 조합 순서 사용에 관한 내용은 343페이지의 “조합 순서 지정”을 참조하십시오.

다음 한정자 값을 정의할 수 있습니다.

표 9-9

SEQUENCE	지정 방법 파일은 ASCII , EBCDIC , 다국적 및 사용자 정의 조합 순서를 지원합니다. 이러한 조합 순서에 대한 자세한 내용은 343페이지의 “조합 순서 지정”을 참조하십시오.
MODIFICATION	SEQUENCE 옵션에 지정된 조합 순서에 대한 변경 사항을 지정합니다. ASCII, EBCDIC, 다국적 또는 사용자 정의 순서를 수정할 수 있습니다. 순서가 기본값 (ASCII)인 경우에도 수정할 순서는 SEQUENCE 한정자에서 지정해야 합니다. character 조합 순서의 문자를 지정합니다. operator 문자를 비교하는 데 사용되는 연산자를 지정합니다. 큼(>), 작음(<) 또는 같음(=)을 지정할 수 있습니다. MODIFICATION 옵션에서 다음과 같은 사항을 변경할 수 있습니다. — 단일 문자 또는 이중 문자는 이미 조합 값이 할당된 단일 문자와 일치할 수 있습니다(“a”=“A”). — 단일 문자 또는 이중 문자는 이미 조합 값을 할당한 단일 문자 다음에 조합할 수 있습니다(“CH”>“C”). — 단일 문자 또는 이중 문자는 이미 조합 값이 할당된 단일 문자 앞에 조합할 수 있습니다(“D”<“A”). — 이중 문자는 이전에 정의된 이중 문자와 일치할 수 있습니다(“CH” = “SH”). — 단일 문자는 이중 문자 순서와 일치할 수 있습니다(“C” = “CH”).
IGNORE	Sort/Merge 에서 처음 비교를 할 때 조합 순서의 문자 또는 문자 범위를 무시하도록 지정합니다. 연결 구분이 발생하면 Sort/Merge 에서는 IGNORE 값에 지정된 문자를 고려합니다.
FOLD	모든 소문자에 해당 대문자의 조합 값이 제공되도록 지정합니다. ASCII, EBCDIC 및 사용자 정의 순서의 경우 소문자는 a에서 z까지입니다. 다국적 순서에서 소문자에는 이미 해당하는 대문자의 조합 값이 있으므로 FOLD를 사용할 필요가 없습니다.

[NO]TIE_BREAK

Sort/Merge에서 동일한 값이 있는 문자 간의 연결을 구분하기 위해 숫자 값을 사용해야 하는지 여부를 지정합니다. 기본적으로 연결 구분은 다국적 순서에서 발생합니다. NOTIE_BREAK를 지정하면 이 기본값이 무시되며 처음 비교 후에 추가 비교를 하지 않습니다.

SCII, EBCDIC 및 사용자 정의 순서에서 연결 구분이 발생하도록 하려면 TIE_BREAK 옵션을 지정해야 합니다. TIE_BREAK는 이러한 순서에 FOLD 또는 MODIFICATION 값을 지정할 때 사용해야 합니다.

보기

지정 방법 파일에서 조합 순서 사용에 관한 보기는 343페이지의 “조합 순서 지정” 및 353페이지의 “Sort/Merge 지정 방법 파일 사용”을 참조하십시오.

**/CONDITION=(NAME=condition-name,
TEST=(field-name operator test-condition
[logical-operator...]))**

지정 방법 파일을 사용하여 레코드의 상대적 순서를 변경하거나 레코드의 특정 필드 내용을 변경할 수 있습니다. 우선 /CONDITION 한정자를 사용하여 조건 테스트를 정의해야 합니다. /CONDITION 한정자를 사용하여 테스트를 정의한 후 같은 테스트를 /KEY 또는 /DATA 한정자에서 사용하여 레코드 순서를 변경할 수 있습니다. 또한 /OMIT 또는 /INCLUDE 한정자에서 이 테스트를 사용하여 레코드 내용을 변경할 수도 있습니다.

출력 파일의 레코드 순서를 변경하려면 우선 /CONDITION 한정자에 조건 이름을 지정하고 해당 조건에 맞는 대상에 대한 테스트를 설정한 후 다음과 같이 양식의 /KEY 한정자에 상대적 순서를 지정합니다.

```
/KEY=(IF condition-name THEN value ELSE value)
```

레코드의 상대적 순서를 지정하는 데 모든 값을 사용할 수 있습니다.

또한 /CONDITION 한정자를 사용하여 출력 레코드의 필드 내용도 변경할 수 있습니다. 먼저 조건 이름을 지정한 다음 조건에 맞는 대상에 대한 테스트를 설정합니다. 양식의 /DATA 한정자에 있는 필드에서 필요한 내용을 지정합니다.

```
/DATA=(IF condition-name THEN "new-contents" ELSE "new-contents")
```

다음과 같은 한정자 값을 지정할 수 있습니다.

표 9-10

NAME	테스트할 조건의 이름을 지정합니다. 이 조건 이름은 /CONDITION 한정자를 사용하여 정의한 후 /KEY, /DATA, /OMIT 및 /INCLUDE 한정자에서 사용할 수 있습니다.
TEST	조건 테스트를 지정합니다.
field-name	테스트할 필드의 이름을 지정합니다. field-name은 /FIELD 한정자에서 미리 정의해야 합니다.
operator	조건 테스트에 사용되는 논리 또는 관계 연산자를 지정합니다. 사용할 수 있는 논리 연산자는 AND와 OR입니다. 지정할 수 있는 관계 연산자는 다음과 같습니다. EQ = 같음 NE = 같지 않음 GT = 큼 GE = 크거나 같음 LT = 작음 LE = 작거나 같음
test-condition	테스트할 상수 또는 field-name을 지정합니다. 상수는 다음 형식으로 지정됩니다. Decimal_digits(기본값) %Ddecimal_digits %Ooctal_digits %Xhexadecimal_digits "character" 일반적으로 기수 연산자(%D)를 지정할 필요는 없지만 test-condition에서는 같은 데이터 유형을 field-name으로 간주합니다.

보기

지정 방법 파일에서 /CONDITION 한정자의 사용에 관한 보기는 353페이지의 “Sort/Merge 지정 방법 파일 사용”을 참조하십시오.

/DATA=field-name
/DATA=(IF condition THEN "new contents"
ELSE "new contents")

출력 레코드의 필드를 삭제하거나 다시 정렬하려면 /DATA 한정자를 사용합니다. 출력 레코드에 표시할 순서로 데이터 필드를 지정합니다. /DATA 한정자에서는 출력 파일로 보낼 레코드의 모든 필드를 식별해야 합니다. /DATA 한정자에서 식별한 필드만 출력 파일로 보내집니다.

먼저 조건 이름을 지정한 다음 /CONDITION 한정자에서 조건에 맞는 대상에 대한 테스트를 설정하여 출력 레코드의 필드 내용을 조건부로 변경할 수 있습니다. 그런 후 다음과 같은 양식인 /DATA 한정자의 필드에서 필요한 내용을 지정합니다.

/DATA=(IF condition-name THEN "new-contents" ELSE "new-contents")

다음과 같은 한정자 값을 지정할 수 있습니다.

표 9-11

field-name	레코드의 필드 이름을 지정합니다. field-name은 미리 /FIELD 한정자에서 정의해야 합니다.
condition-name	/CONDITION 한정자에서 미리 정의된 condition-name을 지정합니다.
new-contents	레코드를 변경할 방법을 지정합니다. new-contents는 /FIELD 한정자에서 정의된 상수나 field-name일 수 있습니다.

보기

지정 방법 파일에서 /DATA 한정자 사용에 관한 보기는 353페이지의 “Sort/Merge 지정 방법 파일 사용”을 참조하십시오.

/FIELD=(NAME=field-name,POSITION:n,SIZE:N, [DIGITS:n,]data-type)
/FIELD=(NAME=field-name,VALUE:n,SIZE:N,[DIGITS:n,] data-type)

출력 레코드의 순서나 형식을 변경할 때 입력 파일의 필드를 정의합니다. 이러한 필드에는 키 필드, 비교할 필드 및 출력 파일로 보낼 필드가 포함됩니다. 이름, 레코드에서의 위치와 크기 및 데이터 유형을 지정하여 각 필드를 식별합니다.

필드 이름을 고유해야 합니다. 중복된 이름은 허용되지 않습니다. 또한, 255개 이상의 필드 정의를 사용할 수 없습니다.

또한, **/FIELD**를 사용하여 상수를 정의하고 이 상수에 **/CONDITION**, **/DATA** 및 **/KEY** 문에서 사용할 유효한 **Sort/Merge** 데이터 유형의 값을 할당할 수 있습니다.

다음과 같은 한정자 값을 지정할 수 있습니다.

표 9-12

NAME	필드의 이름을 지정합니다. field-name 은 공백을 포함할 수 없고 영문자로 시작해야 하며 31자를 넘으면 안 됩니다.
POSITION:n	레코드에서 필드의 위치를 지정합니다.
VALUE:n	/CONDITION , /DATA 또는 /KEY 문에서 사용할 상수 필드에 값을 할당합니다. 이 필드는 상수이지 입력 레코드의 일부가 아니므로 VALUE:n 을 지정할 경우 /POSITION:n 을 지정하지 마십시오.
SIZE:n	문자나 이진 데이터를 포함하는 필드의 크기를 지정합니다. 지정 방법 파일에서 SIZE 는 바이트 길이를 의미합니다. 다음과 같이 데이터 유형에 따라 사용할 수 있는 값이 결정됩니다. — 문자 데이터인 경우 크기는 32,767자를 초과하면 안 됩니다. — 이진 데이터인 경우 지정된 크기는 1, 2, 4, 8 또는 16바이트이어야 합니다. — 고정 소숫점 데이터인 경우 지정된 크기가 없습니다.
DIGITS:n	십진 데이터를 포함하는 필드의 크기를 지정합니다. 십진 데이터를 포함하는 필드의 크기는 31자리를 초과하면 안 됩니다. DIGITS:n 은 십진 데이터를 포함하는 필드를 설명하는 경우에만 사용됩니다.
data-type	필드의 데이터 유형을 지정합니다. 문자인 경우 데이터 유형을 지정할 필요가 없습니다. Sort 에서는 기본적으로 문자 데이터 유형을 지정합니다. Sort/Merge 에서 인식하는 데이터 유형 목록은 333페이지의 “키 정의”를 참조하십시오.

보기

```
/FIELD= (NAME=SALARY, POSITION:10, DIGITS:8, DECIMAL)
```

이 **/FIELD** 한정자는 이름 **SALARY**로 레코드의 필드를 식별하고 이 필드가 레코드의 위치 10에서 시작하고 8자리이며 십진 데이터로 구성되도록 지정합니다.

/INCLUDE=(CONDITION=condition[,KEY=...] [,DATA=...])

레코드가 출력 파일에 조건부로 포함되도록 지정할 수 있습니다. /CONDITION 한정자에 조건을 정의한 후 조건에 맞는 레코드가 출력 파일에 포함되도록 레코드 선택을 /INCLUDE 한정자에 지정합니다.

지정 방법 파일에 여러 /INCLUDE 및 /OMIT 한정자를 지정할 수 있습니다. 이러한 한정자를 지정하는 순서에 따라 입력 레코드의 포함 여부를 테스트하는 순서가 결정됩니다. 마지막 /INCLUDE 한정자 뒤에 아직 포함되지 않았거나 명시적으로 누락된 모든 레코드는 누락됩니다.

이전에 누락되지 않았거나 조건 없이 /INCLUDE 한정자를 지정하여 포함된 레코드를 무조건 포함시킬 수 있습니다.

여러 레코드 형식을 정렬할 경우, 정렬할 레코드 중에 각각의 다른 레코드 형식에 하나의 /INCLUDE 한정자를 지정해야 합니다 INCLUDE 한정자로 KEY 옵션을 지정하지 않으면 Sort에서는 기본 키 정의를 가정합니다. /INCLUDE 한정자에 KEY를 지정하면 기본 키 정의는 사용되지 않습니다. /INCLUDE 한정자에서 KEY 필드의 순서에 따라 정렬을 위한 내부 키 구성 방법이 결정됩니다. /INCLUDE 한정자에서 DATA 필드의 순서는 출력 레코드가 형식화되는 방식을 결정합니다. /INCLUDE 한정자에서 키 또는 데이터 필드를 지정하면 레코드의 다른 모든 키 또는 데이터 필드를 정의해야 합니다.

다음과 같은 한정자 값을 지정할 수 있습니다.

표 9-13

CONDITION	이전의 /CONDITION 한정자에 지정된 condition-name을 참조합니다.
KEY	/KEY 한정자에 정의된 기본 레코드 유형이 사용되지 않으므로 키 필드를 정의합니다.
DATA	/DATA 한정자에 정의된 기본 레코드 유형이 사용되지 않으므로 데이터 필드를 정의합니다

보기

```
/FIELD= (NAME=ZIP, POSITION:20, SIZE:6)
/CONDITION= (NAME=LOCATION,
TEST=(ZIP EQ '01863' ) )
/INCLUDE= (CONDITION=LOCATION)
```

이러한 /CONDITION 및 /INCLUDE 한정자는 우편 번호가 01863인 레코드가 출력 파일에 포함되도록 지정합니다.

/KEY=field-name

/KEY=(field-name,order)

/KEY=((IF condition THEN value ELSE...) value [,order])

Sort 작업에서 사용할 키 필드를 지정합니다. 문자 데이터를 사용하여 전체 레코드를 정렬할 경우 키 필드를 지정할 필요가 없습니다. 그렇지 않은 경우 각각의 키에 대해 우선 순위에 따라 /KEY 한정자를 지정합니다. 255개 만큼의 키 필드에서 정렬할 수 있습니다.

/KEY 한정자를 사용하는 방법에는 다음 세 가지가 있습니다.

- 키 필드 이름을 식별합니다.
- 키 필드 이름을 식별하고 정렬 순서를 지정합니다. 이 경우 필드 이름과 순서 옵션을 괄호로 묶습니다.
- 조건부 한정자로 출력 파일의 레코드 순서를 변경합니다. 먼저 /CONDITION 한정자에 조건 이름을 지정하고 해당 조건에 맞는 대상에 대한 테스트를 설정합니다. 그런 후 다음과 같은 양식의 /KEY 한정자에 상대적 순서를 지정합니다.

/KEY=(IF condition-name THEN value ELSE value)

모든 값을 사용하여 레코드의 상대적 순서를 지정할 수 있습니다.

다음과 같은 한정자 값을 지정할 수 있습니다.

표 9-14

field-name	키 필드의 이름을 지정합니다. field-name은 미리 /FIELD 한정자에서 지정됩니다.
order	정렬 순서를 지정합니다. ASCENDING 옵션은 정렬 또는 병합 작업에 대해 오름차순을 지정하며 기본값입니다. DESCENDING 옵션은 정렬 또는 병합 작업에 대해 내림차순을 지정합니다.
value	키를 지정합니다. 값은 /FIELD 한정자에 정의된 상수 또는 field-name일 수 있습니다

보기

1.

```
/FIELD= (NAME=SALARY, POSITION:10, DIGITS:8, DECIMAL) /KEY= (SALARY, DESCENDING)
```

이 **/KEY** 한정자는 키 필드는 **SALARY**로, 정렬 순서는 내림차순으로 지정합니다.

2.

```
/FIELD= (NAME=ZIP, POSITION:20, SIZE:6)
/CONDITION= (NAME=LOCATION,
             TEST= (ZIP EQ "01863" ) )
/KEY= (IF LOCATION THEN 1
      ELSE 2)
```

이 보기에서는 우편 번호가 **01863**인 모든 레코드가 정렬된 출력 파일의 시작 부분에 표시됩니다. **/CONDITION** 한정자에 정의된 조건 테스트 **LOCATION**은 **/FIELD** 절에 명명된 **ZIP** 필드에 있습니다. 이 **/KEY** 절의 값 **1**과 **2**는 조건에 맞는 레코드와 맞지 않는 레코드의 상대적 순서를 의미합니다.

/OMIT=(CONDITION=condition-name)

/CONDITION 한정자에 정의된 조건에 따라 출력 파일에서 누락시킬 레코드를 지정합니다.

먼저 **/CONDITION** 한정자로 조건을 정의해야 합니다. 조건에 맞는 레코드가 **Sort** 작업에서 누락되도록 요청할 레코드를 **/OMIT** 한정자로 지정합니다. 기본적으로 **Sort/Merge**에서는 다른 모든 입력 레코드를 출력 파일에 포함시킵니다.

지정 방법 파일에 여러 **/OMIT** 및 **/INCLUDE** 한정자를 지정할 수 있습니다. 지정하는 순서에 따라 입력 레코드의 누락 여부를 테스트하는 순서가 결정됩니다. 마지막 **/OMIT** 한정자 다음에 아직 포함되지 않았거나 누락되지 않은 레코드가 모두 포함됩니다. **/OMIT** 한정자만을 지정하여 이전에 포함되거나 누락되지 않는 레코드를 무조건 누락시킬 수 있습니다.

보기

```
/FIELD= (NAME=ZIP, POSITION:20, SIZE:6)
/CONDITION= (NAME=LOCATION,
             TEST= (ZIP EQ '01863' ) )
/OMIT= (CONDITION=LOCATION)
```

이러한 **/CONDITION** 및 **/OMIT** 한정자는 우편 번호가 **01863**인 레코드가 출력 파일에서 누락되도록 지정합니다.

/PAD=single-character

Sort에서 레코드를 다시 형식화하거나 길이가 다른 문자열을 비교하는 경우 문자열을 확장하거나 "채우는" 데 사용할 문자를 지정합니다. 기본적으로 Sort는 이전 버전에 적합하도록 null 문자를 사용하여 채웁니다. 단일 문자로 정의할 수 있는 이중 문자("ch" > "c")는 채우기 문자로 사용할 수 없습니다. 문자, 10진수, 8진수 또는 16진수를 사용할 수 있습니다.

채우기 문자는 다음과 같이 지정해야 합니다.

- 문자의 경우 인용 부호를 사용합니다. 예를 들어, "#"는 번호 기호를 지정합니다.
- 십진수의 경우 십진 기수를 사용합니다. 예를 들어, %D35는 십진수 35를 지정합니다.
- 8진수의 경우 8진 기수를 사용합니다. 예를 들어, %O043은 8진수 043을 지정합니다.
- 16진수의 경우 16진 기수를 사용합니다. 예를 들어, %X23은 16진수 23을 지정합니다.

보기

```
/PAD=" . "
```

이 보기의 /PAD 한정자는 레코드가 마침표로 채워지도록 지정합니다.

/PROCESS=type

SORT 명령에만 적용됩니다. 정렬 작업의 처리 방법(레코드, 태그, 주소 또는 인덱스)을 정의합니다. 출력 레코드를 다시 형식화하려면 주소 또는 인덱스 정렬을 사용할 수 없습니다. 프로세스 유형을 RECORD, TAG, ADDRESS 또는 INDEX로 지정합니다.

프로세스의 네 가지 유형 비교는 361페이지의 “정렬 프로세스”를 참조하십시오.

보기

```
/PROCESS=tag
```

이 보기의 /PROCESS 한정자는 Sort에서 태그 정렬 프로세스를 사용하도록 지정합니다.

/[NO]STABLE

등가 키가 있는 레코드를 해당 입력 파일 순서로 출력 파일로 보내지 않도록 지정합니다. 기본 조건은 **/NOSTABLE**입니다.

기본적으로 레코드가 동일한 키로 정렬되는 경우 출력 파일에서 해당 레코드 순서는 입력 파일에서의 순서와 다를 수 있습니다. 지정 방법 파일에 **/STABLE** 한정자를 지정하면 명령행에 지정된 것처럼 입력 파일의 순서대로 출력 파일에서 등가 키가 있는 레코드를 정렬합니다. 여러 입력 파일을 정렬하는 데 이 한정자를 사용하면 출력 파일에서 첫 번째 파일의 등가 키가 있는 레코드가 두 번째 파일의 등가 키가 있는 레코드 앞에 옵니다.

보기

/STABLE

이 보기의 **/STABLE** 한정자는 등가 키가 있는 레코드가 입력 및 출력 파일에서 순서가 같게 합니다.

/WORK_FILES=(device[,...])

SORT 명령에만 적용됩니다. 성능을 향상시키기 위해 작업 파일을 다른 디스크 구조의 장치에 다시 할당합니다. 지정 방법 파일에 **/WORK_FILES** 한정자를 사용하면 명령행이나 프로그램 수준에서 **Sort**를 호출하기 전에 논리를 할당할 필요가 없습니다.

DCL 한정자 **/WORK_FILES=n**과 달리 지정 방법 파일 한정자

/WORK_FILES=(device[,...])는 작업 파일 수가 아닌 작업 파일 할당을 지정합니다.

작업 파일 사용에 대한 자세한 내용은 362페이지의 “작업 파일 할당”을 참조하십시오.

보기

/WORK_FILES=("WRKD\$:")

이 보기의 **/WORK_FILES** 한정자는 **Sort**의 작업 파일 중 하나를 **WRKD\$:** 장치에 할당합니다. 이 장치에 사용 가능한 공간이 가장 많기 때문입니다.

10 리소스에 대한 액세스 제어

시스템 사이트마다 고유한 보안 요구 사항이 있습니다. 따라서 모든 사이트에는 시스템 관리자와 사용자를 위한 물리적 보안 및 소프트웨어 보안 요구 사항이 요약된 시스템 보안 정책이 있어야 합니다. 시스템 보안을 확고히 하기 위해 OpenVMS 운영 체제에서는 시스템에 대한 액세스와 공유 정보가 포함된 객체에 대한 액세스를 모두 제어합니다. 장치, 볼륨, 논리 이름 표, 파일, 대기열과 같은 이러한 객체는 **보호된 객체**로 알려져 있습니다. 보호된 객체는 모두 주어진 방식으로 객체에 대한 액세스 권한을 부여할 사용자를 지정하는 일련의 액세스 요구 사항을 나열합니다.

OpenVMS Guide to System Security에서는 운영 체제에서 사용할 수 있는 보안 기능과 시스템 관리자가 계정 및 시스템 보안 유지를 위해 수행할 수 있는 작업에 대해 설명합니다. 이 장에서는 OpenVMS가 시스템 리소스를 보호하고 감사하는 방법 중 일부에 대해 설명합니다. 즉, 다음에 대한 정보가 포함됩니다.

- 프로세스의 권한 증명서 표시
- 객체의 보안 프로파일
- 보호 코드 해석
- 기본 파일 보호
- 네트워크에서 파일 액세스
- 계정 및 파일에 대한 액세스 감사

추가적인 보안 정보를 보려면 다음을 참조하십시오.

- 객체 보호 및 일반적인 시스템 보안에 대한 자세한 내용은 *OpenVMS Guide to System Security*를 참조하십시오.
- 이 장에 나오는 명령에 대한 자세한 내용은 *OpenVMS DCL Dictionary* 또는 온라인 도움말을 참조하십시오.

보안 기능

다음과 같은 방법으로 OpenVMS 보안 기능에 익숙해질 수 있습니다.

- 사용 중인 프로세스에 관련된 권한 **증명서** — 권한 증명서는 액세스할 수 있는 리소스를 결정합니다. 프로세스에 해당 증명서가 없으면 보호된 특정 객체를 액세스하지 못할 수도 있습니다.

권한 증명서 표시에 대한 자세한 내용은 390페이지의 “프로세스의 권한 증명서 표시”를 참조하십시오.

- 보호된 객체의 보안 프로파일 표시 — 보안 프로파일에는 보호된 객체에 대한 정보가 들어 있습니다. 자신이 소유한 객체의 보안 프로파일을 변경하여 다른 사용자들이 액세스할 수 있도록 하거나 액세스할 수 없도록 할 수 있습니다.

보안 프로파일에 대한 자세한 내용은 392페이지의 “객체의 보안 프로파일”을 참조하십시오.

- 네트워크에서 파일을 액세스하는 방법 — 액세스 제어 문자열이나 프록시 로그인 계정을 사용하여 수행할 수 있습니다.

원격 파일 액세스에 대한 자세한 내용은 398페이지의 “네트워크에서 파일 액세스”를 참조하십시오.

- 계정 및 파일에 대한 액세스 감사 — 모든 로그인 메시지를 자세히 관찰하고 시스템 관리자와 함께 해당 파일을 감사하여 수행할 수 있습니다.

계정 및 파일에 대한 액세스 감사에 대한 자세한 내용은 402페이지의 “계정 및 파일에 대한 액세스 감사”를 참조하십시오.

프로세스의 권한 증명서 표시

보호된 객체를 액세스하려고 하는 모든 프로세스는 권한 증명서로 알려진 자격 증명을 갖습니다. 보호된 모든 객체는 주어진 방식으로 객체에 대한 액세스 권한을 부여할 사용자를 지정하는 일련의 액세스 요구 사항을 나열합니다. 액세스 프로세스의 권한 증명서가 해당 객체의 권한 증명서와 일치하지 않을 경우 액세스가 거부됩니다.

다음 보기에서는 **SHOW PROCESS** 명령을 사용하여 현재 프로세스의 증명서를 표시하는 방법을 보여 줍니다.

```
$ SHOW PROCESS/ALL
25-NOV-2002 15:23:18.08   User: GREG               Process ID:   34200094
Node: ACCOUNTS           Process name: "GREG"
Terminal:                 VTA2195:  TNA2170:  (Host: 16.32.123.45 Port: 6789)
User Identifier:         [DOC,GREG]   [1]
Base priority:           4
Default file spec:       WORK1:[GREG.FISCAL_96]
Number of Kthreads:      1
Devices allocated:       ACCOUNTS$TWA2:

Process Quotas:
.
.
.
Process rights:
INTERACTIVE               [2]
LOCAL                     [3]
SALES                     [4]
MINDCRIME                 resource [5]
System rights:
SYS$NODE_ACCOUNTS        [6]
```

권한 증명서는 UIC, 환경, 일반 등 세 가지 유형이 있습니다. SHOW PROCESS 명령의 출력에는 이 세 가지가 모두 표시됩니다.

- [1] UIC 증명서, 사용자 Greg는 DOC 그룹의 구성원임을 나타냅니다.
- [2] 환경 증명서, 사용자 Greg는 대화형 사용자임을 나타냅니다.
- [3] 환경 증명서, 사용자 Greg는 로컬로 로그인되었음을 나타냅니다.
- [4] 일반 증명서, 사용자 Greg는 SALES 그룹의 구성원이기도 함을 나타냅니다.
- [5] 일반 증명서, 사용자 Greg는 리소스 속성과 함께 MINDCRIME 증명서를 보유하므로 디스크 공간을 해당 증명서에 지정할 수 있음을 나타냅니다.
- [6] 환경 증명서, 사용자 Greg는 ACCOUNTS 노드에서 작업 중임을 나타냅니다.

객체의 보안 프로파일

운영 체제에서는 많은 사용자를 동시에 지원하기 때문에 한 사용자의 작업이 다른 사용자의 작업을 방해하지 않도록 하는 보안 메커니즘이 내장되어 있습니다. 보호 코드, 액세스 제어, 하드웨어 디자인 등이 메모리, 공유 장치 및 데이터의 사용을 함께 보호하므로 많은 사용자들이 시스템을 공유할 수 있습니다. 객체의 보안 프로파일은 **UIC**(사용자 식별 코드), **ACL**, 해당 객체에 할당된 보호 코드로 구성됩니다. 사용자는 소유한 객체의 보안 프로파일을 표시하거나 수정할 수 있습니다.

보호된 객체의 보안 프로파일을 보려면 **DCL** 명령 **SHOW SECURITY**를 사용합니다. 예를 들어, 다음 명령은 **95_FORECAST.TXT** 파일에 대한 보안 정보를 요청합니다.

```
$ SHOW SECURITY 95_FORECAST.TXT
WORK_DISK$: [GREG] 95_FORECAST.TXT;1 object of class FILE
  Owner: [ACCOUNTING,GREG]
  Protection: (System: RWED, Owner: RWED, Group: RE, World)
  Access Control List: <empty>
```

표시된 내용에 따르면 **95_FORECAST.TXT** 파일은 사용자 **Greg**가 소유하고 있습니다. 또한, 시스템 사용자와 소유자에게 읽기, 쓰기, 실행 및 삭제 액세스 권한을 부여하는 파일의 보호 코드가 나열됩니다. 이 코드는 그룹 사용자에게 읽기 및 실행 액세스 권한을 부여하고 **world** 사용자에게는 액세스 권한을 제공하지 않습니다. 추가 설명을 보려면 394페이지의 “보호 코드 해석”을 참조하십시오. 파일에 대한 **ACL**은 없습니다.

보안 프로파일 수정

보호된 객체의 소유자, 보호 코드 또는 **ACL**에 새 값을 제공하거나, **SET SECURITY** 명령을 사용하여 한 객체의 프로파일을 다른 객체로 복사할 수 있습니다.

예를 들어, 392페이지의 “객체의 보안 프로파일”의 **SHOW SECURITY** 표시에서는 사용자 **Greg**가 **95_FORECAST.TXT** 파일을 소유하고 있음을 보여 줍니다. **Greg**는 소유자로서 해당 파일에 대한 보호 코드를 변경할 수 있습니다. 원래 코드는 **world** 범주의 사용자에게 액세스 권한을 부여하지 않았는데, 지금 **Greg**가 **world** 사용자에게 읽기 및 쓰기 액세스 권한을 허용하도록 변경했습니다.

```
$SET SECURITY/PROTECTION=(W:RW) 95_FORECAST.TXT
```

SHOW SECURITY 명령은 해당 파일에 대한 새 보호 코드를 확인합니다.

```
$ SHOW SECURITY 95_FORECAST.TXT

95_FORECAST.TXT object of class FILE
Owner: [GREG]
Protection: (System: RWED, Owner: RWED, Group: RE, World: RW) Access Control
List: <empty>
```

보호 코드 해석

보호 코드는 특정 사용자 또는 사용자 그룹에 허용되거나 거부되는 액세스 유형을 제어합니다. 보호 코드의 형식은 다음과 같습니다.

[범주 : 허용된 액세스 목록 (, 범주 : 허용된 액세스 목록 ,...)]

범주에는 System(S), Owner(O), Group(G) 및 World(W)가 있습니다. 각 범주에 대해 해당 첫 문자를 약어로 사용할 수 있습니다. 범주에 대한 정의는 다음과 같습니다.

표 10-1

System	UIC가 1~10(8진수) 범위 내에 있고, SYSPRV 권한을 갖거나, 소유자와 동일한 그룹에 있으면서 GRPPRV를 보유하는 모든 사용자 프로세스 또는 어플리케이션입니다.
Owner	UIC가 해당 객체의 UIC와 동일한 모든 사용자 프로세스 또는 어플리케이션입니다.
Group	그룹 UIC가 해당 객체의 그룹 UIC와 동일한 모든 사용자 프로세스 또는 어플리케이션입니다.
World	시스템의 모든 사용자 프로세스 또는 어플리케이션입니다.

사용자 범주를 두 개 이상 지정할 때는 각 범주를 쉼표로 구분하고 전체 코드를 괄호로 묶습니다. 사용자 범주와 액세스 유형을 임의의 순서로 지정할 수 있습니다.

null 액세스 지정은 액세스 권한이 없음을 의미하므로, 사용자 범주에 대한 액세스 유형을 생략할 경우 이 사용자 범주에는 해당 유형의 액세스가 거부됩니다. 사용자 범주에 대해 모든 액세스 권한을 거부하려면 액세스 유형이 없는 상태로 사용자 범주를 지정합니다. 사용자 범주에 대해 액세스를 거부하는 경우 해당 사용자 범주 다음에 오는 콜론을 생략합니다.

파일의 사용 목록에는 Read(R), Write(W), Execute(E) 또는 Delete(D) 액세스 유형이 포함됩니다. 액세스 유형은 각 소유권 범주에 할당되며 콜론(:)으로 구분됩니다. 파일 액세스 유형에는 각각 다음과 같은 의미가 있습니다.

표 10-2

Read	디스크 파일을 읽거나, 인쇄하거나, 복사할 수 있는 권한을 부여합니다. 디렉토리 파일에 대한 읽기 액세스는 파일을 읽거나 나열하고 와일드카드 문자와 함께 파일 이름을 사용하여 파일을 찾을 수 있는 권한을 부여합니다. 읽기 액세스에는 실행 액세스가 포함됩니다.
Write	파일의 내용을 삭제할 수는 없지만 변경하거나 기록할 수 있는 권한을 제공합니다. 쓰기 액세스는 파일의 내용을 설명하는 파일 특성을 수정할 수 있게 합니다. 디렉토리 파일에 대한 쓰기 액세스는 파일의 카탈로그에서 항목을 삽입하거나 삭제할 수 있는 권한을 부여합니다.
Execute	실행 프로그램 이미지나 DCL 명령 프로시저가 들어 있는 파일을 실행할 수 있는 권한을 부여합니다. 디렉토리 파일에 대한 실행 액세스는 이름을 알고 있는 파일을 찾을 수 있는 권한을 부여합니다.
Delete	파일을 삭제할 수 있는 권한을 부여합니다. 파일을 삭제하려면 파일에 대한 삭제 액세스 권한과 이 파일이 들어 있는 디렉토리에 대한 쓰기 액세스 권한이 있어야 합니다.

기본 파일 보호

새 파일은 상위 디렉토리의 기본 **UIC** 기반 보호와 기본 **ACL**(사용 제어 목록)을 받습니다. **ACL**은 파일, 디렉토리, 장치 등 보호된 특정 객체에 대해 사용자나 사용자 그룹이 갖는 액세스 권한을 정의하는 항목들의 모음입니다.

기본 **UIC** 보호 또는 기본 **ACL** 보호를 사용하여 새 파일에 부여된 기본 **UIC** 기반 보호를 무시할 수 있습니다.

기본 UIC 보호

운영 체제에서는 각 프로세스에 다음과 같은 **UIC** 기반 보호를 제공합니다.

```
(S:RWED, O:RWED, G:RE, W)
```

기본적으로 시스템 **UIC**가 있는 사용자와 객체의 소유자에게는 해당 객체에 대한 모든 액세스 권한이 있고, 객체 소유자와 동일한 **UIC** 그룹의 사용자에게는 해당 객체에 대한 읽기 및 쓰기 액세스 권한이 있으며, 그 밖의 모든 사용자에게는 해당 객체에 대한 액세스가 거부됩니다. 사용자가 만드는 파일에 대한 기본 보호를 변경하려면 **/DEFAULT** 한정자를 사용하여 **SET PROTECTION** 명령을 입력합니다. 예를 들어, 로그인 명령 프로시저에 다음 명령을 입력하면 사용자가 만드는 모든 파일에 대한 읽기 및 실행 액세스 권한을 모든 프로세스에 부여하는 것입니다. 이 명령을 실행하려면 로그인 명령 프로시저를 실행해야 합니다.

```
$ SET PROTECTION = (S:RWED,O:RWED,G:RE,W:RE)/DEFAULT
```

기본 ACL 보호

해당 디렉토리 파일의 **ACL**에 기본 보호 **ACE**(액세스 제어 항목)를 넣어서 지정된 디렉토리나 하위 디렉토리에 대해 기본 **UIC** 보호를 무시할 수 있습니다. **ACE**에 지정된 기본 보호는 지정된 디렉토리나 이 디렉토리의 하위 디렉토리에서 만들어진 모든 새 파일에 적용됩니다. 디렉토리 파일의 **ACL**에 있어야 하는 다음 **ACE**는 해당 디렉토리와 이 디렉토리의 하위 디렉토리에 대한 기본 보호가 시스템과 소유자 프로세스에 모든 권한을 허용하고, 그룹 프로세스에는 읽기 및 실행 액세스를 허용하며, **world** 사용자에게는 어떠한 액세스도 허용하지 않도록 지정합니다.

```
$ SET SECURITY/ACL = (DEFAULT_PROTECTION,S:RWED,O:RWED,G:RE,W:)  
[JONES] PERSONAL.DIR
```

해당 디렉토리에서 만들어지는 모든 후속 파일의 **ACL**에 기본 증명서 **ACE**가 복사되도록 지정하려면 디렉토리 파일의 증명서 **ACL**에 **DEFAULT** 옵션을 지정합니다.

다음 보기에 나오는 **ACE**는 디렉토리 파일에 적용되며, 네트워크 사용자가 해당 디렉토리에서 만들어진 모든 파일에 액세스하는 것을 거부합니다.

```
$ SET SECURITY/ACL = (IDENTIFIER=NETWORK,OPTIONS=DEFAULT,ACCESS=NONE) - _$
[JONES] PERSONAL.DIR
```

파일 이름 바꾸기

파일의 이름이 바뀌어도 보호는 변경되지 않습니다. 기존 파일의 새 버전은 이전 버전의 **UIC** 기반 보호와 **ACL**을 받습니다. **BACKUP, COPY, CREATE, SET FILE** 명령의 **/PROTECTION** 한정자를 사용하여 기본 **UIC** 기반 보호를 무시합니다.

명시적 파일 보호

/PROTECTION 한정자(**BACKUP, COPY, CREATE** 명령에 유효함)를 사용하여 새 파일에 대해 **UIC** 기반 보호를 명시적으로 지정할 수 있습니다.

SET SECURITY/PROTECTION 명령을 사용하여 기존 파일에 대한 **UIC** 기반 보호를 변경할 수 있습니다.

파일이 만들어지고 그 파일에 대한 **ACL**을 만든 후에는 해당 **ACL**을 수정하고 원하는 만큼 많은 항목을 추가할 수 있습니다. **ACL**로 지정된 보호는 파일의 사용자 식별 코드 보호를 무시합니다.

다음 보기에서는 **UIC** 기반 보호가 지정됩니다.

```
$ CREATE MAST12.TXT/PROTECTION=(S:RWED,O:RWED,G,W)
```

다음 보기에서는 **MAST12.TXT** 파일에 대한 **UIC** 기반 보호가 변경됩니다.

```
$ SET SECURITY/PROTECTION=(S:RWED,O:RWED,G:RE,W) MAST12.TXT
```

네트워크에서 파일 액세스

다음 절에서는 네트워크에서 파일을 액세스하는 방법에 대해 설명합니다.

액세스 제어 문자열

OpenVMS 네트워크용 DECnet에서 작업을 수행하는 DCL 명령의 파일 지정에 네트워크 액세스 제어 문자열을 포함할 수 있습니다. 액세스 제어 문자열은 로컬 노드의 사용자가 원격 노드의 파일을 액세스할 수 있도록 허용합니다.

액세스 제어 문자열은 다음과 같이 원격 계정에 대한 사용자 이름과 인용 부호로 묶은 사용자 암호로 구성됩니다.

```
NODE'username password'::disk:[directory]filename.filetype
```

주

액세스 제어 문자열에는 다른 사용자가 원격 계정으로 침입할 수 있는 충분한 정보가 있기 때문에 심각한 보안 노출이 생길 수 있습니다.

액세스 제어 문자열 보호

액세스 제어 문자열 정보를 보호하려면 다음을 수행하십시오.

- 하드카피 터미널 또는 비디오 터미널의 정보가 공개되는 것을 방지하십시오. 하드카피 터미널을 사용할 경우에는 출력을 적절히 처리합니다. 비디오 터미널을 사용할 경우, 네트워크 작업이 완료되면 DCL 명령 **RECALL/ERASE**를 사용하여 화면을 지우고 호출 버퍼를 비웁니다. 이렇게 하면 다른 사용자가 **Ctrl/B** 순서를 사용하거나 DCL 명령 **RECALL/ALL**을 사용하여 명령행을 표시하여 암호를 보는 것을 방지할 수 있습니다.
- 탐색의 대상이 되기 쉬운 명령 프로시저에서 액세스 제어 문자열을 포함하는 네트워크 명령을 넣지 마십시오.
- 명령 프로시저에 액세스 제어 문자열을 넣어야 할 경우에는 최적의 파일 보호와 함께 이러한 파일을 제공하십시오.

프록시 로그인 계정을 사용하여 암호 보호

액세스 제어 문자열에 대한 필요성을 없애기 위해 프록시 로그인 계정 사용을 선호할 수도 있습니다. 프록시 로그인을 사용하면 액세스 제어 문자열에 사용자 이름이나 암호를 지정하지 않고도 네트워크에서 파일을 액세스할 수 있습니다. 즉, 프록시 로그인에는 다음과 같은 보안 이점이 있습니다.

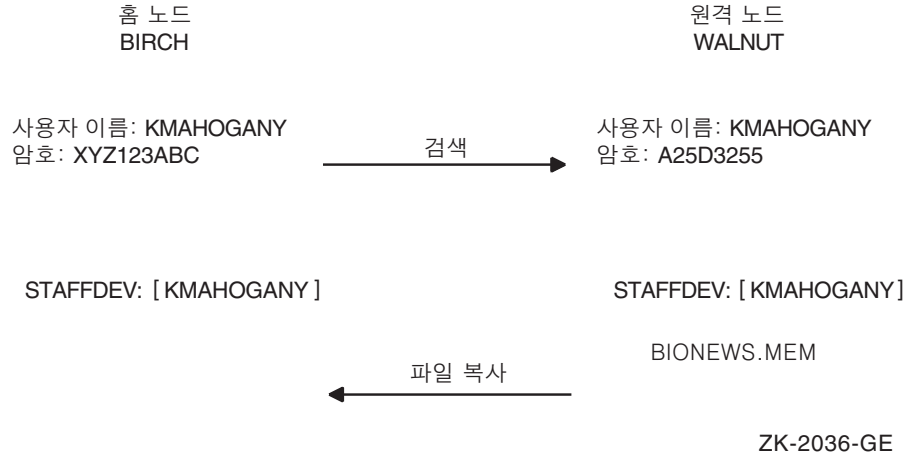
- 요청이 시작된 터미널에서는 암호가 에코되지 않습니다.
- 암호를 암호화되지 않은 양식으로 가로챌 수 있는 시스템 간에는 암호가 전달되지 않습니다.
- 원격 액세스 단계를 수행하기 위한 명령 파일에 암호가 필요하지 않습니다.

사용자가 프록시 로그인을 시작하려면 원격 노드의 시스템 또는 보안 관리자가 프록시 계정을 만들어 주어야 합니다. 프록시 계정은 일반 계정과 마찬가지로 **OpenVMS Authorize** 유틸리티(AUTHORIZE)를 사용하여 만들어지며, 대체로 권한이 없는 계정입니다. 보안 관리자는 한 개의 기본 프록시 계정과 15개까지의 다른 프록시 계정에 대한 액세스 권한을 사용자에게 허용할 수 있습니다. 프록시 로그인 은 시스템 관리자의 설치 작업을 더 많이 필요로 하지만, 더욱 안전한 네트워크 액세스를 제공하며 사용자가 액세스 제어 문자열을 입력해야 할 필요성을 없애 줍니다.

다음 보기는 일반적인 네트워크 로그인 요청과 프록시 로그인 요청 간의 차이점을 보여 줍니다. 각 보기에 대해 다음과 같은 조건이 있습니다.

- 사용자 **KMAHOGANY**에는 사용자 계정이 두 개 있습니다.
 - “XYZ123ABC” 암호를 가진 **BIRCH** 노드의 계정
 - “A25D3255” 암호를 가진 **WALNUT** 노드의 계정
- **KMAHOGANY**는 **BIRCH** 노드로 로그인했습니다.
- **KMAHOGANY**는 **WALNUT** 노드에 있는 계정의 기본 장치 및 디렉토리로부터 **BIONEWS.MEM** 파일을 복사하려고 합니다.

다음 그림에서는 이러한 조건을 보여 줍니다.



- 사용자 **KMAHOGANY**는 액세스 제어 문자열을 사용하여 **BIONEWS.MEM** 파일을 다음과 같이 복사할 수 있습니다.

```
$ COPY WALNUT"KMAHOGANY A25D3255"::BIONEWS.MEM BIONEWS.MEM
```

A25D3255 암호가 예코되는 것에 주의하십시오. 화면을 보고 있는 사람은 모두 이 암호를 볼 수 있습니다.

- **KMAHOGANY**가 **BIRCH** 노드로부터 **WALNUT** 노드의 계정에 이르는 프록시 액세스를 갖고 있을 경우, **BIONEWS.MEM** 파일을 복사하기 위한 명령은 다음과 같습니다.

```
$ COPY WALNUT::BIONEWS.MEM BIONEWS.MEM
```

KMAHOGANY는 액세스 제어 문자열에 암호를 지정하지 않아도 됩니다. 대신, 시스템에서 **BIRCH** 노드의 계정으로부터 **WALNUT** 노드의 해당 계정으로 프록시 로그인을 수행합니다. 암호 교환은 없습니다.

일반 액세스 프록시 계정

보안 관리자는 외부 노드의 사용자 그룹이 일반 액세스 프록시 계정을 사용하도록 공유를 인증할 수 있습니다. 예를 들어, WALNUT 노드의 보안 관리자는 다음과 같은 조건으로 일반 액세스 계정을 만들 수 있습니다.

- 사용자 이름 GENACCESS
- 네트워크 로그인에 대해 제한된 액세스
- 계정의 소유자만이 알고 있는 암호. 원격 사용자 중 어느 누구도 알아서는 안 됩니다. 이것은 계정 보호에 도움이 됩니다.
- 기본 장치 및 디렉토리 STAFFDEV:[BIOSTAFF]

보안 관리자가 GENACCESS 계정에 BIRCH::KMAHOGANY 프록시 액세스 권한을 부여하면 사용자 KMAHOGANY가 다음 명령을 입력하여 BIONEWS.MEM 파일을 복사할 수 있습니다.

```
$ COPY WALNUT::[KMAHOGANY]BIONEWS.MEM BIONEWS.MEM
```

BIONEWS.MEM 파일이 GENACCESS 계정에 대한 기본 장치 및 디렉토리 (STAFFDEV:[BIOSTAFF])에 있지 않으므로 KMAHOGANY는 [KMAHOGANY] 디렉토리를 지정해야 합니다. 또한 BIONEWS.MEM 파일에 대한 보호에서 GENACCESS 계정에 대한 액세스를 허용해야 합니다. 그렇지 않은 경우, 이 명령은 오류가 발생합니다. 주어진 노드에서 두 개 이상의 프록시 계정에 대한 액세스 권한을 갖고 있을 경우, 기본 프록시 계정을 사용하지 않으려면 해당 프록시 계정의 이름을 지정합니다. 예를 들어, GENACCESS 계정(기본 프록시 계정) 대신 PROXY2라는 프록시 계정을 사용하려면 KMAHOGANY는 다음 명령을 입력합니다.

```
$ COPY WALNUT"PROXY2"::[KMAHOGANY]BIONEWS.MEM BIONEWS.MEM
```

이 명령에서는 PROXY2 계정을 사용하여 WALNUT 노드의 [KMAHOGANY] 디렉토리에서 BIONEWS.MEM 파일을 복사합니다.

계정 및 파일에 대한 액세스 감사

시스템을 모니터링하여 가능한 침입 시도를 찾아내는 것이 보안 관리자의 작업이긴 하지만, 사용자가 자신의 계정과 파일에 대한 액세스 감사에서 보안 관리자를 도울 수 있습니다.

마지막 로그인 시간 보기

OpenVMS 시스템은 사용자가 자신의 계정에 마지막으로 로그인한 시간에 대한 UAF 레코드의 정보를 관리합니다. 보안 관리자는 로그인 시 시스템에서 이 정보를 표시해야 하는지 여부를 결정합니다. 보통 내지 높은 보안 요구 사항이 있는 사이트에서는 이 정보를 자주 표시하고 예외적이거나 설명할 수 없는 로그인 성공 및 설명할 수 없는 로그인 오류에 대해 확인하도록 사용자에게 요청합니다.

로그인되지 않은 시기에 대화형 또는 비대화형 로그인 보고서가 있는 경우 보안 관리자에게 즉시 보고하십시오. 그리고 암호를 변경하십시오. 보안 관리자는 계정 파일과 감사 로그를 사용하여 자세히 조사할 수 있습니다.

로그인 오류 메시지가 나타났는데 오류의 원인을 확인할 수 없는 경우에는 다른 사용자가 해당 계정을 액세스하려고 시도하고 있는 중일 가능성이 많습니다. 해당 암호가 67페이지의 “암호 보호에 대한 지침”에 나오는 암호 보안을 위한 권장 사항을 모두 준수하고 있는지 확인하십시오. 그렇지 않은 경우 암호를 즉시 변경하십시오.

로그인 오류 메시지가 나타날 것으로 예상했는데 나타나지 않거나 실패 횟수가 너무 적으면 암호를 변경하십시오. 이러한 로그인 오류 문제에 대한 표시를 보안 관리자에게 보고하십시오.

보안 관리자는 이벤트가 발생할 때 특별히 주의를 기울여야 할 이벤트의 유형을 한 개 이상 선택할 수 있습니다. 그러한 이벤트가 감지되면 보안 관리자는 시스템 보안 감사 로그 파일에 감사를 보내거나 보안 운영자 터미널로 사용되는 터미널에 경보를 보내도록 시스템에 지시합니다. 예를 들어, 보안 관리자는 쓰기 액세스가 금지된 파일을 한 개 이상 식별할 수 있습니다. 이러한 파일에 대한 액세스가 시도된 것을 나타내도록 감사를 활성화하거나 경보를 설정할 수 있습니다.

해당 계정에 대한 침입이 의심되는 경우 암호를 변경하십시오. 보안 관리자가 중요한 파일에 감사를 구현하도록 요청할 수도 있습니다.

보안 경보를 트리거할 수 있는 이벤트

감사 또는 경보를 트리거하는 이벤트에는 다음이 포함될 수 있습니다.

표 10-3

보안 감사 또는 경보를 시작하는 이벤트 예	
이미지 설치	시스템 및 사용자 암호 수정, 시스템 인증 파일,
특정 유형의 파일 액세스	네트워크 프로시 파일 또는 권한 데이터베이스
볼륨 마운트 및 분리	
ACL 파일 또는 글로벌 섹션이 요청한 이벤트 액세스	로그인, 로그아웃, 로그인 오류, 침입 시도

다음 보기에서는 CONFIDREVIEW.MEM 파일을 감사하려고 한다는 것을 전제로 합니다. 사용자 ABADGUY가 CONFIDREVIEW.MEM을 액세스할 경우 삭제 액세스 권한이 있으면, 다음 감사 레코드가 시스템 보안 감사 로그 파일에 기록됩니다.

```

##### OPCOM 11-DEC-1999 09:21:11.10 #####
Message from user AUDIT$SERVER on BOSTON
Security audit (SECURITY) on BOSTON, system id: 19424
Auditable event:      Attempted file access
Event time:          11-DEC-1999 09:21:10.84
PID:                 23E00231 Username:          ABADGUY
Image name:          BOSTON$DUA0:[SYS0.SYSCOMMON.] [SYSEXEC]DELETE.EXE
Object name:          _BOSTON$DUA1:[RWOODS]CONFIDREVIEW.MEM;1
Object type:          file
Access requested:     DELETE
Status:               %SYSTEM-S-NORMAL, normal successful completion
Privileges used:      SYSPRV
  
```

감사 메시지는 침입자의 이름, 액세스 방법([SYSEXEC]DELETE.EXE 프로그램을 사용하여 수행된 성공적인 삭제), 액세스 시간(9:21 A.M.), 파일에 액세스하기 위한 사용 권한 (SYSPRV) 등을 보여 줍니다. 이 정보를 사용하여 보안 관리자가 조치를 취할 수 있습니다.

보안 감사 로그 파일

보안 감사 메시지는 파일이 액세스되고 해당 파일에 대한 **ACL**의 감사 항목에 지정된 조건에 만족할 때마다 보안 감사 로그 파일에 기록됩니다(404페이지의 “중요한 파일에 **ACE** 추가” 참조). 보안 감사로 보호된 시스템의 파일에 대한 액세스와 마찬가지로, **CONFIDREVIEW.MEM** 파일에 액세스하면 보안 감사 로그 파일에 기록할 감사 레코드를 확인하는 메시지가 표시됩니다.

감사가 도입되었으면 보안 관리자에게 정기적으로 문의하여 추가적인 침입이 발생했는지 여부를 확인합니다.

중요한 파일에 **ACE** 추가

부적절하게 액세스된 키 파일이 있을 경우, 파일에 대한 액세스를 감사하기 위한 전략을 보안 관리자와 함께 세워야 할 수도 있습니다.

상황을 검토하고 표준 보호 코드와 일반 **ACL**(*OpenVMS Guide to System Security* 참조)로 파일을 보호하기 위해 가능한 모든 것을 수행한 경우, 보안 감사가 필요한 것으로 결론을 내릴 수 있습니다.

보안 감사를 지정하기 위해 소유한 파일이나 제어 액세스 권한을 가진 파일에 특수 **ACE**(액세스 제어 항목)를 추가할 수 있습니다. 그러나 감사 로그 파일은 시스템 메커니즘이므로 사이트 보안 관리자가 파일 감사의 사용을 제어하는 것이 좋다는 점을 염두에 두십시오. 제어 권한을 가지고 있는 파일에 감사 **ACE**를 추가할 수는 있지만, 시스템 관리자는 시스템 수준에서 파일의 감사를 활성화해야 합니다.

해당 계정에 대한 침입 시도가 의심되는 경우에는 보안 관리자가 모든 파일 액세스에 대한 감사를 일시적으로 활성화할 수 있습니다. 또한 보안 관리자는 감사를 활성화하여 파일 브라우저를 보기 위해 해당 파일에 대한 읽기 액세스를 모니터링할 수 있습니다.

한 파일의 액세스 위반은 다른 파일에 대한 액세스 문제를 나타내는 경우가 많습니다. 따라서 보안 관리자는 보안 감사 **ACE**가 있는 모든 키 파일에 대한 액세스를 모니터링해야 할 수도 있습니다. 원치 않는 액세스 권한이 키 파일에 부여되었을 때는 보안 관리자가 즉각적인 조치를 취해야 합니다.

다음 보기에서는 사용자 **RWOODS**와 해당 보안 관리자가 기밀 파일인 **CONFIDREVIEW.MEM**이 액세스되는 시기를 알아야 한다는 것에 동의하므로, **RWOODS**가 **CONFIDREVIEW.MEM** 파일에 대한 기존 **ACL**에 항목을 추가합니다.

```
$ SET SECURITY/ACL=(ALARM=SECURITY,ACCESS=READ+WRITE-  
_$_+DELETE+CONTROL+FAILURE+SUCCESS) CONFIDREVIEW.MEM
```

11 장치 및 파일의 논리 이름 정의

파일, 디렉토리, 장치 또는 대기열 같은 시스템 객체를 나타내는 다른 이름 대신에 논리 이름을 사용할 수 있습니다. 예를 들어, **기본 디스크**와 디렉토리에 논리 이름을 할당할 수 있습니다. 논리 이름은 가독성과 파일 독립성을 높이는 두 가지 주요한 기능을 합니다.

짧고 의미있는 논리 이름을 사용하여 일반적으로 사용되는 파일, 디렉토리 및 장치를 정의할 수 있습니다. 이러한 이름은 전체 파일 지정 방법보다 더 쉽게 기억하고 입력할 수 있습니다. 로그인 명령 프로시저에서 자주 사용하는 이름을 정의할 수 있습니다. 시스템 관리자는 시스템 시작 명령 프로시저에서 사람들이 자주 사용하는 이름을 정의할 수 있습니다.

프로그램 및 명령 프로시저를 물리적 파일 지정 방법에 관계 없이 보관하기 위해 논리 이름을 사용할 수 있습니다. 예를 들어, 명령 프로시저에서 논리 이름 **ACCOUNTS**를 참조하면 **ACCOUNTS**를 임의의 디스크의 모든 파일과 일치시킬 수 있습니다. 이 장에는 다음 정보가 포함되어 있습니다.

- 논리 이름 특성
- 시스템 정의 논리 이름 사용
- 논리 이름 만들기
- 논리 이름 삭제
- 논리 이름 변환
- 논리 이름 표시
- 검색 목록 만들기 및 사용
- 논리 이름 표 특성
- 기본 논리 이름 표
- 논리 이름 표 만들기
- 논리 이름 변환 순서 수정
- 논리 이름 표 삭제
- 영구 프로세스 논리 이름

이 장에서 설명하는 명령에 대한 자세한 내용은 *OpenVMS DCL Dictionary* 또는 온라인 도움말을 참조하십시오.

논리 이름 특성

논리 이름에는 다음과 같은 특성이 있습니다.

- 문자열(등가 문자열 또는 등가 이름이라고 함) 또는 등가 문자열 목록(검색 목록이라고 함)과 일치합니다. 논리 이름을 사용하면 등가 문자열로 논리 이름을 대체합니다.
- 사용자가 만든 기본 논리 이름 표나 논리 이름 표에 저장됩니다.
- 긴 파일 지정 방법을 줄일 수 있습니다.
- 사용자가 정의하거나 시스템에서 정의할 수 있습니다.
- 물리적 파일 지정 방법에 관계 없이 프로그램 및 명령 프로시저를 보관하는 데 사용할 수 있습니다. 예를 들어, 명령 프로시저에서 논리 이름 **ACCOUNTS**를 참조하면 명령 프로시저를 실행하기 전에 임의의 디스크의 모든 파일에 **ACCOUNTS**를 일치시킬 수 있습니다.

일반적으로 명령에서 시스템 객체를 받아들이면 이 명령은 사용자가 제공한 이름이 논리 이름인지 확인합니다. 이 이름이 논리 이름인 경우 시스템에서는 논리 이름을 실제 값으로 바꾼 다음 명령을 실행합니다.

다음 보기에서는 **DISK7:[WALSH.COMMAND_PROC]** 디렉토리를 나타내기 위해 논리 이름 **COMS**를 만듭니다.

```
$ DEFINE COMS DISK7:[WALSH.COMMAND_PROC]
```

그런 다음 보기에서와 같이 논리 이름 **COMS**를 **DCL** 명령에서 사용할 수 있습니다.

```
$ SET DEFAULT COMS
```

```
$ TYPE COMS:PAYROLL.COM
```

시스템 정의 논리 이름 사용

시스템에서는 사용자가 시스템을 시작하고 로그인하면 일련의 시스템 논리 이름을 만듭니다. 이러한 논리 이름을 사용하면 물리적 장치 이름을 사용하지 않고 일반적으로 사용되는 파일이나 장치를 참조할 수 있습니다. 이러한 이름의 목록에 대해서는 431페이지의 “시스템 논리 이름 디렉토리”를 참조하십시오.

로그인할 때마다 시스템에서는 프로세스에 대한 논리 이름 그룹을 만들어 이러한 이름을 사용자 프로세스 표에 놓습니다. 이러한 이름 목록은 429페이지의 “프로세스 논리 이름 디렉토리”를 참조하십시오.

운영 체제의 프로그램을 나열하기 위해서 이러한 프로그램이 저장되어 있는 디스크 및 디렉토리 이름을 알 필요는 없습니다. 대신 다음과 같이 논리 이름 **SYS\$SYSTEM**을 사용할 수 있습니다.

```
$ DIRECTORY SYS$SYSTEM
```

논리 이름 **SYS\$LOGIN**은 사용자가 로그인할 때 사용자 기본 장치 및 디렉토리를 참조합니다. **SET DEFAULT** 명령을 사용하여 현재 기본값을 변경한 경우 다음 명령을 사용하여 초기 기본 디렉토리의 파일을 표시할 수 있습니다.

```
$ TYPE SYS$LOGIN:DAILY_NOTES.DAT
```

논리 이름 만들기

DEFINE 명령이나 ASSIGN 명령 중 하나를 사용하여 논리 이름을 만들 수 있습니다. 이 장에서는 DEFINE 명령이 보기에 사용되었습니다.

일반적으로 프로세스 표에 논리 이름을 만들며 로그인할 때마다 논리 이름을 사용할 수 있도록 로그인 명령 프로시저(LOGIN.COM)에 논리 이름을 정의합니다. 또한, 대화형으로 논리 이름을 만들 수도 있습니다. 하지만 현재 프로세스가 활성화된 경우에만 이러한 논리 이름을 사용할 수 있습니다.

사용자가 프로세스 표에 만든 논리 이름은 다른 사용자 프로세스에서는 사용할 수 없습니다. 시스템 관리자나 다른 특권이 있는 사용자는 다른 사용자가 액세스할 수 있는 공유 표에 이름을 만들 수 있습니다. 그룹 및 시스템 표는 공유 표에 대한 보기입니다.

공유 표에 대한 자세한 내용은 433페이지의 “공유 논리 이름 표”를 참조하십시오.

DEFINE 명령 사용

DEFINE 명령을 사용하여 논리 이름을 정의하는 형식은 다음과 같습니다.

DEFINE logical-name equivalence-string[,...]

같은 형식을 사용하여 노드 이름, 파일 지정 방법, 장치 이름, 어플리케이션 특유의 정보 또는 기타 논리 이름에 대한 논리 이름을 만들 수 있습니다.

기본적으로 DEFINE 명령은 프로세스 논리 이름 표에 논리 이름을 놓습니다(426페이지의 “논리 이름 표 특성” 참조). 여기서 논리 이름은 사용자 프로세스 및 하위 프로세스에서만 사용할 수 있습니다. 논리 이름을 다른 논리 이름 표에 추가하려면 /JOB, /GROUP, /SYSTEM 또는 /TABLE=table_name 한정자 중 하나를 사용하여 다른 표를 지정할 수 있습니다. 처음 세 개의 한정자는 기본 작업, 그룹 및 시스템 논리 이름 표를 각각 지정합니다. /TABLE=table_name은 표 유형을 지정하는 데 사용할 수 있으며 클러스터 전체 표를 지정하는 데 사용할 수 있는 유일한 한정자입니다.

다음 보기에서 해당 명령은 논리 이름 WORKFILE을 만들어 이 이름을 등가 문자열 DISK2:[WALSH.REPORTS]WORK_SUMMARY.DAT와 일치시킵니다.

```
$ DEFINE WORKFILE DISK2:[WALSH.REPORTS]WORK_SUMMARY.DAT
```

WORKFILE을 논리 이름으로 정의한 다음 논리 이름을 등가 문자열과 교대로 사용할 수 있습니다.

다음 보기에서 해당 명령은 인쇄 대기열 BLDGC_LPS20_ANSI에 대해 논리 이름 MY_Q를 만듭니다.

```
$ DEFINE MY_Q BLDGC_LPS20_ANSI
```

그리고 다음 명령을 사용하여 FABLES.TXT 파일을 BLDGC_LPS20_ANSI 인쇄 대기열로 인쇄할 수 있습니다.

```
$ PRINT/QUEUE=MY_Q FABLES.TXT
```

다음 보기는 **/TABLE=table_name** 한정자를 사용하여 프로세스 논리 이름 표가 아닌 표에 논리 이름을 만드는 것을 보여 줍니다. LNM\$SYSCUSTER를 지정하면 논리 이름은 기본 클러스터 전체 표, LNM\$SYSCUSTER_TABLE에 놓이므로 클러스터의 모든 사용자가 액세스할 수 있습니다.

```
$ DEFINE/TABLE=LNM$SYSCUSTER CUSTOMERS
DISK1:[CUSTOMER_VISITS]CUSTOMERS.TXT
```

파일 I/O용 명령 프로시저에 논리 이름 만들기

명령 프로시저에서 논리 이름을 사용하여 파일 I/O(입력 및 출력)를 수행할 수 있습니다. OPEN 명령을 사용하여 파일을 열면 이 파일의 논리 이름도 만들 수 있습니다. 후속 READ, WRITE 및 CLOSE 명령에서는 실제 파일 지정 방법 대신 논리 이름을 사용하여 파일을 참조할 수 있습니다.

다음 보기에서 OPEN 명령은 논리 이름 INFILE을 만들고 CLOSE 명령은 이 이름을 삭제합니다.

```
$ OPEN INFILE DISK3:[WALSH]DATA.DAT
$ READ INFILE RECORD
$ CLOSE INFILE
```

논리 이름 만들기 규칙

DEFINE 명령을 사용하여 논리 이름을 만드는 경우 다음 규칙을 따르십시오.

- 등가 문자열과 논리 이름을 각각 255자 이내로 제한합니다. 논리 이름에는 영문자 및 숫자, 밑줄(_), 달러 기호(\$) 및 하이픈(-)이 포함될 수 있습니다.
- 등가 문자열을 지정하는 경우 파일 지정 방법에 필요한 구두점(콜론, 대괄호, 마침표)을 포함합니다. 예를 들어, 장치 이름은 콜론으로 끝나고 디렉토리 지정 방법은 대괄호로 묶고 파일 유형 앞에는 마침표가 와야 합니다.

- 논리 이름이 파일 지정 방법의 일부만 나타내는 경우 콜론을 사용하여 해당 이름을 파일 지정 방법의 다른 부분과 구분합니다. 완전한 파일 지정 방법을 나타내는 데 논리 이름을 사용하는 경우 종결 콜론이 필요하지 않습니다.

또한, 논리 이름은 파일 지정 방법의 가장 왼쪽 구성 요소여야 합니다.

- 옵션으로, 논리 이름은 콜론으로 끝납니다.

ASSIGN 명령은 논리 이름을 논리 이름 표에 놓기 전에 콜론을 제거하고 **DEFINE** 명령은 콜론을 논리 이름의 일부로 저장합니다.

- 논리 이름을 한 개의 등가 문자열에 일치시키는 경우 같은 논리 이름을 다른 등가 문자열에 일치시킵니다. 각 정의를 다른 **논리 이름 표**에 정의하거나 다른 액세스 모드로 정의하지 않는 한, 두 번째 정의는 첫 번째 정의에 우선합니다.

다음 명령은 **DISK1:[SALES_STAFF]PAYROLL.DAT** 파일을 표시합니다.

```
$ DEFINE PAY DISK1:[SALES_STAFF]PAYROLL.DAT
$ TYPE PAY

$ DEFINE PAY_FILE DISK1:[SALES_STAFF]PAYROLL
$ TYPE PAY_FILE:*.DAT

$ DEFINE PAY_DIR DISK1:[SALES_STAFF]
$ TYPE PAY_DIR:PAYROLL.DAT

$ DEFINE PAY_DISK DISK1:
$ TYPE PAY_DISK:[SALES_STAFF]PAYROLL.DAT
```

변환 속성

논리 이름을 만들면 시스템의 등가 문자열 해석 방법을 수정하는 변환 속성을 지정할 수 있습니다.

변환 속성을 등가 문자열에 적용하려면 **/TRANSLATION_ATTRIBUTES** 한정자를 **DEFINE** 명령에 사용합니다. 이 한정자는 위치 한정자입니다. 이 한정자를 명령행에 놓는 위치에 따라 변환 속성을 모든 등가 문자열에 적용하거나 특정 등가 문자열에만 적용할 수 있습니다.

다음 보기에서 장치 이름 **DJA3**는 논리 이름 **DISK**와 함께 숨겨집니다.

```
$ DEFINE/TRANSLATION_ATTRIBUTES=CONCEALED DISK DJA3:
$ SHOW DEFAULT
  DISK:[SAM.PUP]
$ SHOW LOGICAL DISK
  "DISK" = "DJA3" (LNM$PROCESS_TABLE)
```

논리 이름 **DISK**는 물리적 장치 **DJA3**를 나타냅니다. 따라서 **SHOW DEFAULT** 명령은 물리적 장치 이름 **DJA3**가 아닌 논리 이름 **DISK**를 표시합니다. **SHOW LOGICAL** 명령은 **DISK**의 변환을 표시합니다.

CONCEALED 속성은 시스템 메시지가 장치의 물리적 이름이 아닌 논리 이름을 표시하게 합니다. 일반적으로 **CONCEALED** 속성을 물리적 장치를 나타내는 논리 이름과 함께 사용합니다. 숨겨진 장치를 사용하면 디스크나 테이프를 보관하는 물리적 장치가 어떤 것인지에 대한 고려를 하지 않고 프로그램을 작성하고 명령 프로시저를 쓸 수 있으며 기타 작업을 수행할 수 있습니다. 또한, 물리적 장치 이름보다 더 의미있는 이름을 사용할 수 있습니다.

TERMINAL 속성은 논리 이름을 반복적으로 변환하는 것을 방지합니다. 즉, 등가 문자열이 논리 이름인지 확인하기 위해 검사하지 않습니다. 첫 번째 변환 이후의 변환은 “terminal” (마지막이거나 완료됨)입니다.

액세스 모드

OpenVMS 운영 체제에는 다음 네 가지 액세스 모드가 있습니다.

- 사용자 모드(가장 바깥쪽 및 최소 특권 모드)
- 감독자 모드
- 실행 모드
- 커널 모드(가장 안쪽 및 최대 특권 모드)

DCL 명령 **DEFINE**이나 **ASSIGN**을 사용하여 처음 세 개의 모드(사용자, 감독자 및 실행)에서 논리 이름을 만들 수 있습니다. 각 논리 이름 정의에 대한 다른 액세스 모드를 지정하면 같은 논리 이름을 같은 논리 이름 표의 다른 등가 문자열과 일치시킬 수 있습니다. 모든 논리 이름 표에서 실행 모드로 논리 이름을 만들려면 **SYSNAM**이나 **SYSPRV** 특권이 있어야 합니다.

사용자 모드

사용자 모드에서 만든 논리 이름은 임시적입니다. 다음 명령이나 이미지를 실행하는 동안만 논리 이름을 사용하려는 경우 사용자 모드에서 논리 이름을 정의하십시오.

다음 보기에서 논리 이름 **ADDRESSES**는 프로그램 **PAYABLE**을 실행한 후 자동으로 삭제됩니다.

```
$ DEFINE/USER_MODE ADDRESSES DISK1:[SAM.ACCOUNTS]OVERDUE.LIS
$ RUN PAYABLE
```


감독자 모드

모드를 지정하지 않고 **DEFINE** 명령을 사용하는 경우 **DCL**에서는 감독자 모드에서 논리 이름을 만듭니다.

다음 보기에서 명령은 논리 이름 **ACCOUNTS**를 프로세스 논리 이름 표에 있는 두 개의 다른 등가 문자열(하나는 감독자 모드, 다른 하나는 실행 모드)에 일치시킵니다.

```
$ DEFINE ACCOUNTS DISK1:[ACCOUNTS]CURRENT.DAT  
$ DEFINE/EXECUTIVE_MODE ACCOUNTS DISK1:[JANE.ACCOUNTS]OBSOLETE.DAT
```

실행 모드

논리 이름을 조회할 때 **LOGINOUT**과 같은 모든 특권있는 이미지와 유틸리티는 사용자 모드와 감독자 모드 이름 및 표를 통과합니다. 논리 이름을 유틸리티를 비롯한 특권있는 이미지에서 사용하려는 경우 실행 또는 커널 모드 표의 실행 또는 커널 모드에서 정의되어야 합니다. 실행 모드에 정의되어 있는 논리 이름의 다른 후보는 인쇄 대기열 및 시스템 디스크 같은 사용자 작업 그룹과 시스템 리소스에서 사용하는 공용 디렉토리의 이름입니다.

커널 모드

운영 체제와 특권있는 프로그램만 커널 모드에서 논리 이름을 만들 수 있습니다.

논리 노드 이름 만들기

네트워크 노드 이름 대신 또는 노드 이름 및 액세스 제어 문자열 대신 논리 노드 이름을 사용할 수 있습니다. 한 번 논리 노드 이름을 정의하면 화면에서 사용자 이름 및 암호를 입력하고 표시하는 것을 방지하는 데 이 이름을 사용할 수 있습니다.

논리 노드 이름을 정의하려면 다음 규칙을 따르십시오.

- 밑줄(_)로 논리 이름을 시작하지 않습니다.
- 등가 문자열을 이중 콜론(::)으로 끝내고 인용 부호(“”)로 묶습니다.
- 액세스 제어 문자열에 인용 부호가 나타나게 하려면 두 개의 인용 부호(“”) 집합을 사용합니다.

액세스 제어 문자열에 대한 자세한 내용은 이 매뉴얼의 116페이지의 “네트워크 노드 이름”, 121페이지의 “액세스 제어 문자열 형식” 및 398페이지의 “네트워크에서 파일 액세스”를 참조하십시오.

- 1-255자를 포함하는 논리 이름을 지정합니다.

주

암호를 포함하는 **DEFINE** 명령을 파일(예: 사용자 로그인 명령 프로시저)에 넣지 마십시오. 다른 사용자가 파일을 읽으면 암호를 보게 됩니다.

다음 보기에서 해당 명령은 논리 이름 **BOS**를 노드 이름 **BOSTON**과 액세스 제어 문자열에 일치시킵니다. 여기서 **ADAMS**는 사용자 이름이고 **OLMEKIKI**는 암호입니다.

```
$ DEFINE BOS "BOSTON" "ADAMS OLMEKIKI" ":::"
```

파일 지정 방법에서 논리 노드 이름 사용

파일 지정 방법은 논리 노드 이름(시스템이 로컬 노드에서 변환하는 대상)과 논리 장치 이름(시스템이 원격 노드에서 변환하는 대상)을 모두 포함할 수 있습니다. 노드 이름만 나타내기 위해 논리 이름을 사용하려면 파일 지정 방법의 노드 위치에서 논리 이름을 사용할 때 이중 콜론(::)을 포함해야 합니다.

시스템에서 논리 노드 이름을 로컬 노드에서 변환한 다음 형식이 올바른지 결정하기 위해 파일 지정 방법의 나머지 부분을 구문 분석합니다.

다음 보기에서 시스템은 로컬 노드에서 논리 노드 이름 **NYC**를 변환하고 원격 노드 (**NEWYRK**)에서 논리 장치 이름(**DOC**)을 변환합니다.

```
$ DEFINE NYC NEWYRK::  
$ TYPE NYC::DOC:[PERKINS]TERM_PAPER.DAT
```

액세스 제어 문자열 무시

논리 노드 이름에서 액세스 제어 문자열을 무시하려면 명령행에서 논리 이름과 액세스 제어 문자열을 모두 지정하십시오.

다음 보기에서 액세스 제어 문자열 **"REVERE HTEBAZILE"**은 **BOS**의 등가 문자열에 제공된 액세스 제어 문자열을 무시합니다.

```
$ DEFINE BOS "BOSTON" "ADAMS OLMEKIKI" ":::"  
$ TYPE BOS"REVERE HTEBAZILE":::RIDE.DAT
```

시스템에서 논리 노드 이름을 반복적으로 변환하는 경우 처음으로 변환되는 논리 노드 이름의 액세스 제어 정보는 그 다음에 오는 액세스 제어 정보를 무시합니다. 예를 들어, 논리 이름 **TEST1**은 **TORONTO"TEST NAMWENLUAP"::DBA1:**으로 변환됩니다.

```
$ DEFINE TORONTO "TRNTO" "TEST EIZNEKCAM" ":::"  
$ DEFINE TEST1 "TORONTO" "TEST NAMWENLUAP" "::DBA1:"  
$ TYPE TEST1:PROC.DAT
```

TORONTO는 논리 노드 이름이므로 **반복적 변환**이 발생합니다. 즉, 정의에 있는 논리 이름의 모든 수준을 찾을 때까지 운영 체제는 논리 이름 표를 검색합니다. 하지만 **DEFINE TEST1** 논리 이름 할당의 액세스 제어 문자열은 **DEFINE TORONTO** 논리 노드 이름 할당의 액세스 제어 문자열을 무시합니다. 그러므로 **TYPE** 명령은 다음 파일을 표시합니다.

```
TRNTO"TEST NAMWENLUAP": :DBA1:PROC.DAT
```

같은 객체에 대해 여러 논리 이름 만들기

여러 **DEFINE** 명령을 사용하면 같은 객체를 참조하는 여러 논리 이름을 만들 수 있습니다. 예를 들어, 다음 명령은 논리 이름 **\$TERMINAL**과 **CONSOLE**을 터미널의 물리적 이름에 일치시키므로 두 개의 논리 이름 모두가 같은 장치(**LTA69**)로 변환됩니다.

```
$ DEFINE $TERMINAL LTA69
$ DEFINE CONSOLE LTA69
```

논리 이름 삭제

논리 이름을 삭제하려면 **DEASSIGN** 명령을 사용합니다. 사용자 프로세스와 작업 논리 이름 표에 논리 이름을 정의하면 사용자 프로세스가 종료되거나 사용자의 작업에 의해 논리 이름이 명시적으로 삭제될 때까지 삭제되지 않습니다. 하지만 **DEFINE** 명령에 **/USER_MODE** 한정자를 지정하면 논리 이름은 프로세스 논리 이름 표에 정의되며 그 다음 명령 이미지가 실행된 후에 자동으로 삭제됩니다.

콜론으로 끝나는 논리 이름을 삭제하려면 두 개의 콜론을 지정합니다. **ASSIGN** 명령처럼 **DEASSIGN** 명령은 일치하는 항목을 논리 이름 표에서 검색하기 전에 한 개의 콜론을 제거합니다.

논리 이름 변환

시스템에서 **DCL** 명령행에 있는 파일 지정 방법이나 장치 이름을 읽는 경우 가장 왼쪽 구성 요소가 논리 이름인지 확인하기 위해 파일 지정 방법이나 장치 이름을 확인합니다. 가장 왼쪽 구성 요소가 콜론, 공백, 쉼표 또는 행 종결자(예: **Enter**)로 끝나는 경우 시스템에서는 이 구성 요소를 논리 이름으로 변환하려고 시도합니다. 가장 왼쪽 구성 요소가 다른 문자로 끝나는 경우 시스템에서는 이 구성 요소를 논리 이름으로 변환하려고 시도하지 않습니다.

다음 보기에 표시된 명령을 입력하면 **PUP**가 파일 지정 방법의 가장 왼쪽 구성 요소이므로 시스템에서는 **PUP**가 논리 이름인지 확인합니다. 가장 왼쪽 구성 요소는 **Enter**로 종료되기 때문에 시스템에서는 **PUP**를 변환하려고 시도합니다.

```
$ TYPE PUP Enter
```

다음 보기에 표시된 명령을 입력하면 시스템에서 **DISK**가 논리 이름인지 확인합니다. **DISK**가 가장 왼쪽 구성 요소이며 콜론으로 끝나기 때문에 시스템에서 **DISK**를 변환하려고 시도합니다. 시스템에서 **PUP**는 확인하지 않습니다.

```
$ TYPE DISK:PUP Enter
```

세 번째 보기에 표시된 가장 왼쪽의 구성 요소는 오른쪽 대괄호()로 끝나므로 **[DRYSDALE]PUP**를 변환하려고 시도하지 않습니다.

```
$ TYPE [DRYSDALE]PUP Enter
```

반복적 변환

논리 이름 변환은 반복(iterative)될 수 있습니다. 시스템에서 논리 이름을 변환한 다음 첫 번째 논리 이름에 포함되어 있는 것을 찾은 논리 이름에 대해 변환 과정을 반복합니다.

시스템에서는 논리 이름 변환을 수행하는 수준 수를 제한합니다. 수준 수는 시스템 기능에 따라 다르지만 적어도 9개는 됩니다. 시스템 결정 수준 수보다 많이 정의하거나 순환 정의를 만드는 경우 논리 이름을 사용할 때 오류가 발생합니다.

다음 보기에 표시된 첫 번째 **DEFINE** 명령은 논리 이름 **DISK**를 장치 이름 **DUA1**에 일치시킵니다. 두 번째 **DEFINE** 명령은 논리 이름 **MEMO**를 파일 지정 방법 **DISK:[JEFF.MEMOS]COMPLAINT.TXT**에 일치시킵니다.

```
$ DEFINE DISK DUA1:
$ DEFINE MEMO DISK:[JEFF.MEMOS]COMPLAINT.TXT
```

시스템에서 논리 이름 **MEMO**를 변환할 때 등가 문자열

DISK:[JEFF.MEMOS]COMPLAINT.TXT를 찾습니다. 그런 다음 이 파일 지정 방법의 가장 왼쪽 구성 요소가 콜론, 공백, 쉼표 또는 행 끝 종결자로 끝나는지 확인합니다. **DISK** 다음에 있는 콜론을 찾습니다. 그리고 시스템에서 해당 논리 이름도 변환합니다. 파일 지정 방법의 최종 변환은 다음과 같습니다.

```
DUA1:[JEFF.MEMOS]COMPLAINT.TXT
```

시스템 기본값으로 채워진 누락 필드

시스템에서 논리 이름을 변환할 때 현재 기본 장치, 디렉토리 및 버전 번호를 사용하여 파일 지정 방법의 누락 필드를 채웁니다. 명령의 입력 파일을 지정하는 데 논리 이름을 사용하면 이 명령은 논리 이름을 사용하여 파일 지정 방법을 출력 파일에 할당합니다.

등가 문자열이 파일 이름과 파일 유형을 포함하면 출력 파일에도 같은 파일 이름과 파일 유형이 지정됩니다. 등가 문자열에 파일 유형이 포함되어 있지 않으면 기본 파일 유형이 제공됩니다. 제공되는 파일 유형은 사용하고 있는 명령에 따라 다릅니다.

입력 파일 목록에서 논리 이름을 사용하면 각 논리 이름의 등가 문자열에서 임시 기본값을 제공합니다.

다음 보기에서는 장치 이름이 논리 이름 **HIG**에 대해 지정되어 있지 않기 때문에 **MAL**의 장치 이름이 임시 기본 장치로 **DBA1**을 정의합니다.

```
$ SET DEFAULT DBA2:[CASEY]
$ DEFINE MAL DBA1:[MALCOLM]
$ DEFINE HIG [HIGGINS]
$ PRINT ALPHA,MAL:BETA,HIG:GAMMA
```

PRINT 명령은 다음 파일을 찾습니다.

DBA2:[CASEY]ALPHA.LIS

DBA1:[MALCOLM]BETA.LIS

DBA1:[HIGGINS]GAMMA.LIS

논리 이름 변환의 기본 검색 순서

동일한 논리 이름은 두 개 이상의 논리 이름 표에 존재할 수 있습니다. 시스템에서 파일 지정 방법의 논리 이름을 변환하면 일치하는 항목을 찾을 때까지 논리 이름 표 목록을 검색합니다. 시스템은 찾은 첫 번째 일치 항목을 사용합니다.

검색된 논리 이름 표 목록은 논리 이름 `LNМ$FILE_DEV`의 정의에서 지정됩니다. 기본값 목록은 프로세스, 작업, 그룹, 시스템 및 클러스터 전체 시스템 논리 이름 표로 구성됩니다. 검색 순서는 같습니다(프로세스, 작업, 그룹, 시스템 및 클러스터 전체 시스템).

444페이지의 “논리 이름 변환 순서 수정”에 설명된 대로 검색 순서를 수정할 수 있습니다.

논리 이름 표시

논리 이름과 해당 등가 문자열을 표시하려면 **SHOW LOGICAL** 명령을 사용합니다.

때때로 논리 이름 정의에 다른 논리 이름이 포함됩니다. **SHOW LOGICAL** 명령은 반복적인 변환을 수행합니다. 그런 다음 등가 문자열과 변환 수준 모두를 표시합니다. 수준 번호는 0을 기준으로 하므로 0은 첫 번째 수준이며 1은 두 번째 수준이 되는 형태입니다. 지정한 논리 이름에 대해 찾은 첫 번째 변환만 표시하려면 **SHOW TRANSLATION** 명령을 사용합니다. 자세한 내용은 *OpenVMS DCL Dictionary*를 참조하십시오.

영구 프로세스 파일의 등가 문자열을 결정하는 데 **SHOW LOGICAL** 명령을 사용하는 경우 (447페이지의 “영구 프로세스 논리 이름” 참조), 이 명령은 문자열의 장치 부분만 표시합니다. 예를 들면 다음과 같습니다.

```
$ SHOW LOGICAL SYS$INPUT
"SYS$INPUT" = "_TTB4:" (LNM$PROCESS_TABLE)
```

다음 보기에서는 논리 이름 **MYDISK**가 표시됩니다. 두 개의 변환이 수행되면 숫자 1은 변환의 두 번째 수준을 표시합니다.

```
$ SHOW LOGICAL MYDISK
"MYDISK" = "WORK4" (LNM$PROCESS_TABLE)
1 "WORK4" = "$255$DUA17:" (LNM$SYSTEM_TABLE)
```

다음 보기에서 논리 이름 **WORKFILE**의 등가 문자열이 표시됩니다.

```
$ SHOW LOGICAL WORKFILE
"WORKFILE" = "DISK2:[WALSH.REPORTS]WORK_SUMMARY.DAT" (LNM$PROCESS_TABLE)
```

시스템에서는 논리 이름, 논리 이름의 변환 및 논리 이름이 있는 표 이름을 표시합니다.

검색할 논리 이름 표 지정

기본적으로 **SHOW LOGICAL** 명령은 프로세스, 작업, 그룹, 시스템 및 클러스터 전체 표를 검색하고 일치하는 모든 항목을 표시합니다. 하지만 **/TABLE** 한정자를 사용하여 검색할 특정 논리 이름 표를 지정할 수 있습니다. 또한, **/GROUP**, **/SYSTEM**, **/JOB** 및 **/PROCESS** 한정자를 사용하여 그룹, 시스템, 작업 및 프로세스 논리 이름 표의 논리 이름을 각각 표시할 수도 있습니다.

다음 보기에서 **SHOW LOGICAL** 명령은 **/TABLE** 한정자를 사용하여 프로세스 논리 이름 표(**LNM\$PROCESS**)의 논리 이름을 표시합니다.

```
$ SHOW LOGICAL/TABLE=LNM$PROCESS
(LNM$PROCESS_TABLE)
  "DECW$DISPLAY" = "_WSA30:"
  "SYS$COMMAND" = "_FIFI$VTA65:"
  "SYS$DISK" [super] = "WORK1:"
  "SYS$DISK" [exec] = "WORK1:"
  "SYS$ERROR" = "_FIFI$VTA65:"
  "SYS$INPUT" = "_FIFI$VTA65:"
  "SYS$OUTPUT" [super] = "_FIFI$VTA65:"
  "SYS$OUTPUT" [exec] = "_FIFI$VTA65:"
  "TT" = "_VTA65:"
```

변환 속성 및 액세스 모드 표시

논리 이름의 변환 속성과 액세스 모드를 표시하려면 다음과 같이 **SHOW LOGICAL/FULL** 명령을 사용합니다.

```
$ SHOW LOGICAL/FULL SYS$ERROR
  "SYS$ERROR" [exec] = "_PADRAIC$TDA824:" [terminal] (LNM$PROCESS_TABLE)
```

이 보기에서는 실행 모드에서 논리 이름 **SYS\$ERROR**를 표시하고 변환 속성 **terminal**을 표시합니다.

검색 목록 만들기 및 사용

논리 이름이 단일 **DEFINE**(또는 **ASSIGN**) 명령에서 여러 개의 등가 문자열과 일치하면 검색 목록이 만들어집니다.

파일 지정 방법에서 검색 목록을 사용하면 검색 목록은 다음과 같이 변환됩니다.

- 검색 목록에 한 장치만 포함되어 있으면 원본 기본 디렉토리가 검색됩니다.
- 검색 목록에 장치와 디렉토리가 있는 경우 완전한 파일 지정 방법을 구성하는 데 두 개 모두가 사용됩니다.

시스템은 일치하는 항목을 찾을 때까지 사용자가 등가 문자열을 지정한 순서로 논리 이름을 변환합니다.

해당 명령은 찾은 첫 번째 파일에만 영향을 미치고 그 지점에서 검색이 종료됩니다. 일치하는 항목을 찾지 못하면 시스템이 찾으려고 시도한 마지막 파일에 대해서만 오류를 보고합니다.

검색 목록은 와일드카드가 아니고 검색할 위치의 목록입니다.

다음 보기에서 논리 이름 **GETTYSBURG**는 검색 목록입니다.

```
$ DEFINE GETTYSBURG [JONES.HISTORY],[JONES.WORKFILES]
$ SHOW LOGICAL GETTYSBURG
```

```
"GETTYSBURG" = "[JONES.HISTORY]" (LNM$PROCESS_TABLE)
              = "[JONES.WORKFILES]"
```

다음 보기에서 **TYPE** 명령은 **GETTYSBURG**의 선행 논리 이름 정의에 지정되어 있는 순서대로 **[JONES.WORKFILES]**를 검색하기 전에 등가 문자열 **[JONES.HISTORY]**를 검색합니다.

```
$ TYPE GETTYSBURG:SPEECH.TXT
```

```
DISK1:[JONES.HISTORY]SPEECH.TXT;2
```

```
Fourscore and seven years ago, our fathers brought
forth on this continent a new nation, conceived
in liberty, and dedicated to the proposition that
all men are created equal.
```

```
.
.
.
```

TYPE 명령에서 **SPEECH.TXT**라는 파일을 찾으면 검색을 끝내고 해당 파일을 표시합니다.

와일드카드를 허용하는 명령과 함께 검색 목록 사용

와일드카드를 허용하는 명령과 함께 검색 목록을 사용할 수 있습니다. 와일드카드를 사용하면 시스템은 검색 목록에 있는 각각의 등가 문자열을 사용하여 파일 지정 방법을 구성합니다. 명령은 기존 파일을 식별하는 각 파일 지정 방법에서 실행됩니다.

다음 보기에서 **DIRECTORY** 명령은 버전 필드의 와일드카드 문자로 지정됩니다. 따라서 이 명령은 **GETTYSBURG**에 의해 정의된 검색 목록에서 모든 버전의 **SPEECH.TXT**를 찾습니다.

```
$ DIRECTORY GETTYSBURG:SPEECH.TXT;*
```

```
Directory DISK1:[JONES.HISTORY]
```

```
SPEECH.TXT;2      SPEECH.TXT;1
```

```
Total of 2 files.
```

```
Directory DISK1:[JONES.WORKFILES]
```

```
SPEECH.TXT;1
```

```
Total of 1 file.
```

```
Grand total of 2 directories, 3 files.
```

검색 목록을 입력하면(예: **DIRECTORY** 명령 사용), 운영 체제에서는 목록의 다른 부분에서 누락된 파일 지정 방법의 일부를 제공하기 위해 목록의 한 부분에 있는 요소를 사용합니다. 다음 보기의 **SYS\$LOGIN**에서 표시하는 것처럼 파일 지정 방법이 완전하지 않으면 명령행은 여러 파일 및 파일 없음 상태를 만듭니다.

```
$ DIRECTORY SYS$MANAGER:LOGIN.COM,SYS$LOGIN
```

다음과 같이 파일 지정 방법 다음에 세미콜론을 붙이면 여러 파일 및 파일 없음 상태를 만드는 것을 방지할 수 있습니다.

```
$ DIRECTORY SYS$MANAGER:LOGIN.COM; ,SYS$LOGIN
```

SET DEFAULT 명령과 함께 검색 목록 사용

검색 목록을 **SET DEFAULT** 명령 매개 변수의 첫 번째 부분으로 지정하면 시스템에서는 변환되지 않은 검색 목록 이름을 **SYS\$DISK**에 할당합니다. **SYS\$DISK**는 기본 디스크로 변환되는 논리 이름입니다. 검색 목록을 **SET DEFAULT** 명령 매개 변수의 첫 번째 부분으로 지정할 때 검색 목록의 각 등가 문자열은 장치 이름을 포함해야 합니다.

다음 보기에서는 장치와 디렉토리가 모두 지정되어 있으므로 두 개 모두가 파일 지정 방법을 구성하는 데 사용됩니다.

```
$ DEFINE FIFI DISK1:[FRED],DISK2:[GLADYS],DISK3:[MEATBALL.SUB]
$ DIRECTORY FIFI:MEMO.LIS
```

이 명령은 다음 파일 목록을 표시합니다.

```
DISK1:[FRED]MEMO.LIS
DISK2:[GLADYS]MEMO.LIS
DISK3:[MEATBALL.SUB]MEMO.LIS
```

다음 보기에서 **SHOW DEFAULT** 명령은 기본 디스크 및 디렉토리를 **DISK2:[MEATBALL.SUB]**로 표시합니다. 다음으로 검색 목록 **FIFI**를 정의합니다. **SET DEFAULT** 명령은 검색 목록을 매개 변수로 사용합니다. **SHOW DEFAULT** 명령의 두 번째 사용은 기본 디렉토리가 변경되지 않았음을 표시합니다. 하지만 검색 목록 **FIFI**는 등가 문자열과 함께 기본 장치로 표시됩니다. **SHOW DEFAULT** 명령은 시스템에서 검색 목록을 평가하는 순서로 검색 목록을 표시합니다.

```
$ SHOW DEFAULT
    DISK2:[MEATBALL.SUB]
$ DEFINE FIFI DISK1:[FRED], DISK2:[GLADYS], DISK3:
$ SET DEFAULT FIFI
$ SHOW DEFAULT
    FIFI:[MEATBALL.SUB]
=    DISK1:[FRED]
=    DISK2:[GLADYS]
=    DISK3:[MEATBALL.SUB]
```

RUN 명령을 사용하여 검색 목록 사용

RUN 명령 다음에 검색 목록이 오면 시스템은 이전에 설명한 대로 파일 지정 방법을 구성하지만, 그런 다음 이 목록의 파일이 이미지로 설치되었는지 확인합니다. 검색 목록에서 설치된 이미지인 첫 번째 파일을 실행합니다. 그런 다음 **RUN** 명령이 종료됩니다.

파일 지정 방법 중 설치된 이미지가 없으면 시스템에서는 파일 지정 방법을 구성하는 프로세스를 반복합니다. 이번에는 디스크에서 각각의 파일 지정 방법을 찾고 여기서 찾은 첫 번째 파일을 실행합니다. 지정한 파일을 알려진 파일 목록이나 디스크에서 찾지 못하면 오류 메시지가 표시됩니다.

여러 검색 목록의 검색 순서

파일 지정 방법은 두 개 이상의 검색 목록을 포함할 수 있습니다. 이 경우 첫 번째 장치 이름은 상수로 유지되고 파일 이름 검색 목록의 각 항목이 사용됩니다. 파일 이름 검색 목록의 모든 항목이 첫 번째 장치 이름과 연결된 다음 두 번째 장치 이름과 연결됩니다. 각 장치를 검색할 때까지 이 프로세스가 계속됩니다.

검색 목록의 한 이름이 다른 검색 목록으로 변환되면 반복적(중첩된) 검색 목록이 있을 수도 있습니다. 이러한 경우 시스템에서는 다음 상위 수준 이름을 계속 처리하기 전에 하위 목록에 있는 각각의 이름을 사용합니다.

다음 보기는 파일 이름 및 장치 이름에 검색 목록이 있는 파일 지정 방법을 보여 줍니다.

```
$ DEFINE FILE CHAP1.RNO, CHAP2.RNO
$ DEFINE DISK WORK1:[ROSE], WORK2:[THORN]
$ SET DEFAULT DISK
$ DIRECTORY FILE
```

Directory WORK1:[ROSE]

CHAP1.RNO;2 CHAP2.RNO;1

Total of 2 files.

Directory WORK2:[THORN]

CHAP1.RNO;1 CHAP2.RNO;1

Total of 2 files.

Grand total of 2 directories, 4 files.

각 파일 이름에 대해 디렉토리 목록이 주어집니다. **WORK1:[ROSE]**에 대해서는 첫 번째 디렉토리 목록이 **WORK2:[THORN]**에 대해서는 두 번째 디렉토리 목록이 주어집니다.

다음 보기는 반복적 검색 목록을 보여 줍니다.

```
$ DEFINE NESTED FRED.DAT, NEW_LIST, RICKY.DAT
$ DEFINE NEW_LIST ETHEL.DAT, LUCY.DAT
```

검색 목록 **NESTED**의 검색 순서는 다음과 같습니다.

```
FRED.DAT
ETHEL.DAT
LUCY.DAT
RICKY.DAT
```

논리 이름 표 특성

논리 이름 표에는 다음 특성이 있습니다.

- 범위(공유인지 프로세스-개인인지 여부)
- 액세스 모드
- 이름
- 상위 논리 이름 표
- 액세스 제어(공유 논리 이름 표만 해당)
- 할당량(해당 논리 이름에서 사용하는 풀의 양을 제한하기 위해)

시스템 초기화 중에 몇 개의 공유 논리 이름 표가 만들어집니다. 새 프로세스를 만들 때 시스템에서 해당 프로세스에 대한 공유 및 프로세스-개인인 여러 가지 다른 표를 만듭니다. 이러한 모든 표는 428페이지의 “기본 논리 이름 표”에 나와 있습니다.

논리 이름 표를 만들 때 이 표의 액세스 모드를 지정할 수 있습니다. 지정하지 않으면 표 만들기 요청한 액세스 모드(일반적으로 감독자 모드나 사용자 모드)가 기본 모드가 됩니다. 논리 이름 표는 고유한 액세스 모드 및 하위 특권 액세스 모드의 논리 이름을 포함할 수 있습니다. 논리 이름 표는 동일하거나 하위 특권 액세스 모드의 다른 표에 대해 상위 표가 될 수 있습니다.

논리 이름 표는 이름(논리 이름)으로 식별할 수 있습니다. 각 이름 표 이름은 논리 이름 표 내에 논리 이름으로 포함되어야 합니다.

논리 이름 표 디렉토리

디렉토리라는 다음 두 개의 특수 논리 이름 표가 논리 이름 표 이름의 컨테이너로 존재합니다.

- 프로세스 디렉토리, LNM\$PROCESS_DIRECTORY

프로세스 디렉토리는 해당 프로세스의 모든 프로세스-개인 표의 이름과 고유의 표 이름을 포함합니다. 각 프로세스에는 고유의 프로세스-개인 디렉토리가 있습니다.

- 시스템 디렉토리, LNM\$SYSTEM_DIRECTORY

시스템 디렉토리는 모든 공유 표의 이름과 고유의 표 이름을 포함합니다. 시스템마다 하나의 LNM\$SYSTEM_DIRECTORY가 있습니다.

이러한 디렉토리는 반복적으로 표 이름으로 변환되는 이름을 포함합니다. 표로 변환되는 모든 논리 이름 표 이름과 논리 이름은 이러한 디렉토리에 보관됩니다.

논리 이름 표의 상위 표가 반드시 디렉토리 표일 필요는 없습니다. 즉, 이 계층 구조는 논리 이름 표 이름의 위치와 구별됩니다.

디렉토리 표 구조 표시

논리 이름 표에 대한 논리 이름 디렉토리 표의 관계를 표시하려면 다음 보기에서와 같이 **SHOW LOGICAL/STRUCTURE** 명령을 입력합니다.

```
$ SHOW LOGICAL/STRUCTURE
(LNM$PROCESS_DIRECTORY)
    (LNM$PROCESS_TABLE)
(LNM$SYSTEM_DIRECTORY)
    (LNM$SYSTEM_TABLE)
    (LMF$LICENSE_TABLE)
    (LNM$CLUSTER_TABLE)
        (LNM$SYSCLUSTER_TABLE)
    (LNM$GROUP_000123)
    (LNM$JOB_824E98E0)
    .
    .
    .
```

이 보기는 각 논리 이름 표 디렉토리에 있는 논리 이름 표 이름을 보여 줍니다. 또한, **LNM\$CLUSTER_TABLE**과 **LNM\$SYSCLUSTER_TABLE**의 관계도 보여 줍니다.

기본 논리 이름 표

428페이지의 “기본 논리 이름 표”에는 시스템 디렉토리 및 프로세스 디렉토리 표를 비롯한 실행 모드에서 만든 기본 표가 표시됩니다.

표 11-1 기본 논리 이름 표

표 이름	전체 표 이름	논리 이름	설명
프로세스 논리 이름 표			
프로세스 디렉토리	LNM\$PROCESS_DIRECTORY	(다른 논리 이름 없음)	반복적으로 표 이름으로 변환되는 이름 및 프로세스-개인 논리 이름 표 이름의 정의가 들어 있습니다.
프로세스 표	LNM\$PROCESS_TABLE	LNM\$PROCESS	SYS\$DISK 및 SYS\$INPUT과 같은 프로세스-개인 논리 이름이 들어 있습니다.
공유 논리 이름 표			
시스템 디렉토리	LNM\$SYSTEM_DIRECTORY	(다른 논리 이름 없음)	반복적으로 표 이름으로 변환되는 이름 및 공유 논리 이름 표 이름의 정의가 들어 있습니다.
시스템 표	LNM\$SYSTEM_TABLE	LNM\$SYSTEM	SYS\$LIBRARY 및 SYS\$SYSTEM과 같은 시스템의 모든 프로세스에서 공유하는 이름이 들어 있습니다.
클러스터 전체 시스템 표	LNM\$SYSCLUSTER_TABLE	LNM\$SYSCLUSTER	OpenVMS Cluster 시스템의 모든 프로세스에서 공유하는 이름이 들어 있습니다.
클러스터 전체 상위 표	LNM\$CLUSTER_TABLE	LNM\$CLUSTER	LNM\$SYSCLUSTER_TABLE을 포함하는 모든 클러스터 전체 논리 이름 표의 상위 표입니다.
그룹 표	LNM\$GROUP_gggggg ^a	LNM\$GROUP	해당 UIC 그룹의 모든 프로세스에서 공유하는 이름이 들어 있습니다.
작업 표	LNM\$JOB_xxxxxxx ^b	LNM\$JOB	SYS\$LOGIN 및 SYS\$SCRATCH와 같은 작업 트리의 모든 프로세스에서 공유하는 이름이 들어 있습니다.

- a. 문자열 *gggggg*는 프로세스의 UIC 그룹 번호를 포함하는 6자리 8진수를 나타냅니다.
- b. 문자열 *xxxxxxxx*는 작업 정보 블록의 주소인 8자리 16진수를 나타냅니다.

프로세스 논리 이름 디렉토리

표 11-2에는 로그인할 때 프로세스 논리 이름 디렉토리 표,
LNM\$PROCESS_DIRECTORY에서 만든 프로세스-개인 논리 이름이 표시됩니다.

표 11-2 프로세스 논리 이름 디렉토리

이름	설명
LNM\$GROUP	LNM\$GROUP_ <i>gggggg</i> 로 정의된 그룹 논리 이름입니다. 여기서 <i>gggggg</i> 는 해당 그룹 번호를 나타냅니다. LNM\$GROUP_ <i>gggggg</i> ^a 는 UIC 그룹에서 사용하는 논리 이름 표입니다. 표 LNM\$GROUP_ <i>gggggg</i> 는 시스템 디렉토리 표에서 카탈로그로 만들어집니다. 그러므로 LNM\$GROUP은 반복적으로 그룹 논리 이름 표의 이름으로 변환되는 논리 이름입니다.
LNM\$JOB	LNM\$JOB_ <i>xxxxxxxx</i> 로 정의된 작업 논리 이름입니다. 여기서 <i>xxxxxxxx</i> ^b 는 작업 트리에 대해 고유한 번호를 나타냅니다. LNM\$JOB_ <i>xxxxxxxx</i> 는 사용자 작업에서 사용하는 논리 이름 표입니다. 표 LNM\$JOB_ <i>xxxxxxxx</i> 는 시스템 디렉토리 표에서 카탈로그로 만들어집니다. 그러므로 LNM\$JOB은 반복적으로 사용자 작업 논리 이름 표의 이름으로 변환되는 논리 이름입니다.
LNM\$PROCESS	프로세스 논리 이름 표의 이름인, LNM\$PROCESS_TABLE로 반복적으로 변환되는 프로세스 논리 이름입니다.
LNM\$PROCESS_DIRECTORY	프로세스 디렉토리 논리 이름 표의 이름입니다.

- a. 문자열 *gggggg*는 프로세스의 UIC 그룹 번호를 포함하는 6자리 8진수를 나타냅니다.
- b. 문자열 *xxxxxxxx*는 작업 정보 블록의 주소인 8자리 16진수를 나타냅니다.

프로세스 논리 이름 표

시스템의 각 프로세스에는 LNM\$PROCESS_TABLE이라는 프로세스 논리 이름 표가 있습니다. 프로세스 표의 이름은 해당 프로세스 및 후속 하위 프로세스에서만 사용할 수 있습니다. 로그인하면 시스템에서는 프로세스의 논리 이름을 만들고 프로세스 표에 놓습니다.

이름 LNM\$PROCESS를 통해서 간접적으로 LNM\$PROCESS_TABLE을 참조할 수 있습니다. 다음 보기에서와 같이 이러한 간접 참조를 통해 LNM\$PROCESS를 여러 등가 이름으로 재정의하여 그 안에 있는 한 개 이상의 사용자 고유 표를 포함시킬 수 있습니다.

```
$CREATE/NAME_TABLE APPLICATION_NAMES
$DEFINE/TAB=LNM$PROCESS_DIRECTORY LNM$PROCESS APPLICATION_NAMES,
LNM$PROCESS_TABLE
```

기본적으로 프로세스 표에는 430페이지의 “프로세스 논리 이름 표의 기본 논리 이름”에 표시된 논리 이름이 들어 있습니다. 논리 이름 SYS\$INPUT, SYS\$OUTPUT, SYS\$ERROR 및 SYS\$COMMAND는 영구 프로세스 파일(프로세스 수명 동안에 열린 상태로 있는 파일)을 참조합니다. 영구 프로세스 파일에 대한 자세한 내용은 447페이지의 “영구 프로세스 논리 이름”을 참조하십시오.

표 11-3 프로세스 논리 이름 표의 기본 논리 이름

이름	설명
SYS\$COMMAND	DCL이 입력을 읽어들이는 초기 파일(일반적으로 사용자 터미널)입니다. DCL이 입력을 읽어들이는 파일을 입력 스트림이라고 합니다. 명령 인터프리터는 SYS\$COMMAND를 사용하여 원본 입력 스트림을 “기억”합니다.
SYS\$DISK	로그인 시 설정되거나 SET DEFAULT 명령으로 변경되는 기본 장치입니다.
SYS\$ERROR	DCL이 경고, 오류, 심각한 오류로 인해 발생하는 시스템 오류 메시지를 쓰는 기본 장치나 파일입니다.
SYS\$INPUT	DCL이 입력을 읽어들이는 기본 파일입니다.
SYS\$NET	OpenVMS용 DECnet 작업 대 작업 통신의 대상 프로세스를 호출하는 원본 프로세스입니다. 대상 프로세스에서 열면 SYS\$NET은 프로세스가 파트너와 데이터를 교환할 수 있는 논리 링크를 표시합니다. SYS\$NET은 작업 대 작업 통신 중에만 정의됩니다.
SYS\$OUTPUT	DCL이 출력을 기록하는 기본 파일(일반적으로 사용자 터미널)입니다. DCL이 출력을 쓰는 파일을 출력 스트림이라고 합니다.
TT	터미널의 기본 장치 이름입니다.

시스템 논리 이름 디렉토리

표 11-4에는 시스템 디렉토리 표 LNM\$SYSTEM_DIRECTORY에 포함된 기본 시스템 논리 이름이 표시됩니다.

표 11-4 시스템 논리 이름 디렉토리의 기본 논리 이름

이름	설명
LNM\$CLUSTER	반복적으로 LNM\$CLUSTER_TABLE로 변환되는 클러스터 전체 상위 표의 논리 이름입니다.
LNM\$DCL_LOGICAL	LNM\$FILE_DEV로 정의되는 DCL 논리 이름입니다. LNM\$DCL_LOGICAL은 SHOW LOGICAL 명령, SHOW TRANSLATION 명령 및 F\$TRNLNM 렉시칼 함수에 의해 검색되고 표시되는 논리 이름 표의 목록으로 반복적으로 변환됩니다. 기본적으로 이러한 명령은 프로세스, 작업, 그룹, 시스템 및 클러스터 전체 시스템 논리 이름 표 순서로 검색하고 표시합니다.
LNM\$DIRECTORIES	LNM\$PROCESS_DIRECTORY 및 LNM\$SYSTEM_DIRECTORY로 정의되는 디렉토리 논리 이름입니다.
LNM\$FILE_DEV	파일 지정 방법을 처리할 때 시스템에서 검색하는 논리 이름 표 목록으로 정의되는 검색 목록의 논리 이름입니다. LNM\$PROCESS, LNM\$JOB, LNM\$GROUP 및 LNM\$SYSTEM으로 정의되므로 시스템은 프로세스, 작업, 그룹, 시스템 및 클러스터 전체 시스템 논리 이름 표 순서로 검색합니다.
LNM\$GROUP	사용자 그룹 표, LNM\$GROUP_gggggg에 정의되는 그룹 논리 이름입니다. ^a
LNM\$JOB	LNM\$JOB_xxxxxxxx로 정의되는 작업 논리 이름입니다. ^b
LNM\$PERMANENT_MAILBOX	LNM\$SYSTEM으로 정의되는 영구 우편함 논리 이름입니다. 영구 우편함과 연결된 논리 이름은 논리 이름 LNM\$PERMANENT_MAILBOX가 반복적으로 변환되는 논리 이름 표에 입력됩니다.
LNM\$SYSCLUSTER	반복적으로 LNM\$SYSCLUSTER_TABLE로 변환되는 클러스터 전체 시스템 논리 이름 표의 논리 이름입니다.
LNM\$SYSTEM	반복적으로 LNM\$SYSTEM_TABLE, LNM\$SYSCLUSTER로 변환되는 시스템 논리 이름 표 이름입니다.
LNM\$TEMPORARY_MAILBOX	LNM\$JOB으로 정의되는 임시 우편함 논리 이름입니다. 임시 우편함과 연결된 논리 이름은 논리 이름 LNM\$TEMPORARY_MAILBOX가 반복적으로 변환되는 논리 이름 표에 입력됩니다.

a. 문자열 gggggg는 프로세스의 UIC 그룹 번호를 포함하는 6자리 8진수를 나타냅니다.

b. 문자열 xxxxxxxx는 작업 정보 블록의 주소인 8자리 16진수를 나타냅니다.

공유 논리 이름 표

이 절에서는 기본 공유 논리 이름 표를 설명합니다.

- 클러스터 전체 시스템 표
- 클러스터 전체 상위 표
- 그룹 표
- 작업 표
- 시스템 표

클러스터 전체 시스템 표, LNM\$SYSCUSTER_TABLE

LNM\$SYSCUSTER_TABLE은 클러스터 전체 시스템 논리 이름 표의 이름입니다. 이 표에는 클러스터의 모든 사용자가 사용할 수 있는 논리 이름이 들어 있습니다.

이름 LNM\$SYSCUSTER를 통해 간접적으로 LNM\$SYSCUSTER_TABLE을 참조할 수 있습니다. 간접 참조를 통해 LNM\$SYSCUSTER를 여러 등가 이름으로 재정의하고 그 안에 사용자 고유 표를 포함시킬 수 있습니다.

클러스터 전체 상위 표, LNM\$CLUSTER_TABLE

LNM\$CLUSTER_TABLE은 LNM\$SYSCUSTER_TABLE을 비롯한 모든 클러스터 전체 논리 이름 표의 상위 표입니다. 이 표를 참조하려면 논리 이름 LNM\$CLUSTER를 사용합니다.

그룹 표, LNM\$GROUP_gggggg

사용자 그룹 표의 이름은 LNM\$GROUP_gggggg(gggggg)는 사용자 식별 코드[UIC] 그룹 번호를 나타냄)입니다. 이 표의 이름은 같은 UIC 그룹 번호를 갖는 모든 사용자가 사용할 수 있습니다. 시스템의 각 그룹에는 해당되는 그룹 논리 이름 표가 있습니다.

이름 LNM\$GROUP을 통해 간접적으로 LNM\$GROUP_gggggg를 참조할 수 있습니다. 간접 참조를 통해 LNM\$GROUP_gggggg를 여러 등가 이름으로 재정의하고 그 안에 사용자 고유 표를 포함시킬 수 있습니다. 또한, 이렇게 하면 UIC 그룹 번호를 기억할 필요가 없으며 가장 최근에 정의된 표를 사용하고 있는지 확인합니다.

작업 표, LNM\$JOB_xxxxxxx

사용자 작업 표의 이름은 LNM\$JOB_xxxxxxx(xxxxxxx)는 작업 트리에 대해 시스템에서 정의한 작업 정보 블록[JIB]을 나타냅니다.

작업 표에는 작업 트리의 모든 프로세스(해당 프로세스 및 모든 하위 프로세스)에서 사용할 수 있는 논리 이름이 들어 있습니다. 시스템의 각 작업 트리에 대해 한 개의 작업 표가 있습니다. 작업 표는 공유 가능하므로 작업 트리의 모든 프로세스에서 액세스할 수 있습니다.

이름 LNM\$JOB을 통해 간접적으로 LNM\$JOB_xxxxxxx를 참조할 수 있습니다. 이 간접 참조를 통해 LNM\$JOB을 여러 등가 이름으로 재정의하고 그 안에 사용자 고유 표를 포함시킬 수 있습니다. 또한, LNM\$JOB을 사용하면 JIB 주소를 찾을 필요가 없으며 가장 최근에 정의된 작업 표를 사용하고 있는지 확인할 수 있습니다.

시스템은 마운트된 디스크, 마운트된 테이프 및 임시 우편함에 대해 만든 논리 이름을 작업 논리 이름 표에 놓습니다. 또한, 시스템에서는 다음 논리 이름을 만듭니다.

- SYS\$LOGIN

사용자가 로그인할 때의 기본 장치 및 디렉토리입니다.

- SYS\$LOGIN_DEVICE

사용자가 로그인할 때의 기본 장치입니다.

- SYS\$REM_ID

OpenVMS용 DECnet 네트워크 연결을 통해 초기화된 작업에 대한 것으로, 작업이 시작된 원격 노드의 프로세스 ID입니다. OpenVMS 운영 체제에서 이 프록시 로그인이 활성화되어 있는 경우 이 ID는 프로세스의 사용자 이름이고 프록시 로그인이 활성화되어 있지 않은 경우에는 프로세스 ID(PID) 번호입니다. 프록시 계정에 대한 프록시 로그인을 통해 사용자가 액세스 제어 문자열을 지정하지 않고 네트워크를 통해 파일에 액세스할 수 있습니다.

- SYS\$REM_NODE

OpenVMS용 DECnet 네트워크 연결을 통해 초기화된 작업에 대한 것으로, 작업이 시작된 원격 노드의 이름입니다.

- SYS\$SCRATCH

임시 파일이 쓰여진 기본 장치 및 디렉토리입니다.

시스템 표, LNM\$SYSTEM_TABLE

시스템 표의 이름은 LNM\$SYSTEM_TABLE입니다. 시스템 표에는 시스템 수준에서 시스템의 모든 사용자가 사용할 수 있는 논리 이름이 있습니다.

시스템 표는 일반적으로 검색 목록 LNM\$SYSTEM_TABLE, LNM\$SYSCLUSTER로 정의된 LNM\$SYSTEM을 통해 간접적으로 참조할 수 있습니다. 이 노드에 로컬인 시스템 이름과 클러스터의 모든 노드에 일반적인 시스템 이름을 포함하려면 이름 LNM\$SYSTEM을 사용합니다.

표 11-5에는 시스템이 시작할 때 시스템 표에 자동으로 정의되는 논리 이름이 표시됩니다.

표 11-5 시스템 논리 이름 표의 기본 논리 이름

이름	설명	기본 주소
DBG\$INPUT	OpenVMS 디버거의 기본 입력 스트림으로 프로세스 수준의 SYS\$INPUT과 일치합니다.	적용 안 됨
DBG\$OUTPUT	OpenVMS 디버거의 기본 출력 스트림으로 프로세스 수준의 SYS\$OUTPUT과 일치합니다.	적용 안 됨
SYS\$COMMON	SYS\$SYSROOT의 일반적 부분에 대한 장치 및 디렉토리 이름입니다.	SYS\$SYSDEVICE:[SYSn.SYSCOMMON.], 여기서 <i>n</i> 은 프로세서의 루트 디렉토리 번호입니다.
SYS\$ERRORLOG	오류 로그 데이터 파일의 장치 및 디렉토리 이름입니다.	SYS\$SYSROOT:[SYSERR]
SYS\$EXAMPLES	시스템 예제의 장치 및 디렉토리 이름입니다.	SYS\$SYSROOT:[SYSHLP.EXAMPLES]
SYS\$HELP	시스템 도움말 파일의 장치 및 디렉토리 이름입니다.	SYS\$SYSROOT:[SYSHLP]
SYS\$INSTRUCTION	시스템 지시 사항 데이터 파일의 장치 및 디렉토리 이름입니다.	SYS\$SYSROOT:[SYSCBI]
SYS\$LIBRARY	시스템 라이브러리의 장치 및 디렉토리 이름입니다.	SYS\$SYSROOT:[SYSLIB]

표 11-5 시스템 논리 이름 표의 기본 논리 이름 (계속)

이름	설명	기본 주소
<code>SYS\$LOADABLE_IMAGES</code>	운영 체제 실행 로드 가능한 이미지, 장치 드라이버 및 기타 실행 로드된 코드의 장치 및 디렉토리입니다.	<code>SYS\$SYSROOT:[SYS\$LDR]</code>
<code>SYS\$MAINTENANCE</code>	시스템 유지보수 파일의 장치 및 디렉토리 이름입니다.	<code>SYS\$SYSROOT:[SYSMAINT]</code>
<code>SYS\$MANAGER</code>	시스템 관리자 파일의 장치 및 디렉토리 이름입니다.	<code>SYS\$SYSROOT:[SYSMGR]</code>
<code>SYS\$MESSAGE</code>	시스템 메시지 파일의 장치 및 디렉토리 이름입니다.	<code>SYS\$SYSROOT:[SYSMSG]</code>
<code>SYS\$NODE</code>	시스템에서 OpenVMS용 DECnet이 활성화되고 사용자가 네트워크에 연결되어 있는 경우 로컬 시스템에 대한 네트워크 노드 이름입니다.	적용 안 됨
<code>SYS\$PROCUMP</code>	이미지 덤프가 쓰여지는 디렉토리이며 사용자가 설정합니다.	기본 설정 없음
<code>SYS\$SHARE</code>	시스템 공유 이미지의 장치 및 디렉토리 이름입니다.	<code>SYS\$SYSROOT:[SYSLIB]</code>
<code>SYS\$SPECIFIC</code>	<code>SYS\$SYSDEVICE</code> 의 노드 특유의 부분에 대한 장치 및 디렉토리 이름입니다.	<code>SYS\$SYSDEVICE:[SYSn.]</code> , 여기서 <i>n</i> 은 사용자 프로세서의 루트 디렉토리 번호입니다.
<code>SYS\$STARTUP</code>	시스템 시작 파일의 장치 및 디렉토리 이름입니다.	먼저 <code>SYS\$SYSROOT:[SYS\$STARTUP]</code> 을 가리킨 다음 <code>SYS\$MANAGER</code> 를 가리키는 검색 목록입니다.
<code>SYS\$SYSDEVICE</code>	시스템 디렉토리를 포함하는 시스템 디스크입니다.	일반적으로 <code>SYS\$DISK</code> 입니다.

표 11-5 시스템 논리 이름 표의 기본 논리 이름 (계속)

이름	설명	기본 주소
SYS\$SYSROOT	시스템 디렉토리의 장치 및 루트 디렉토리입니다.	먼저 SYS\$SYSDEVICE:[SYSn.]을 가리킨 다음 SYS\$COMMON을 가리키는 검색 목록입니다. 여기서 <i>n</i> 은 프로세서의 루트 디렉토리 번호입니다.
SYS\$SYSTEM	운영 체제 프로그램 및 프로시저의 장치 및 디렉토리입니다.	SYS\$SYSROOT:[SYSEXE]
SYS\$TEST	UETP(사용자 환경 테스트 패키지) 파일의 장치 및 디렉토리 이름입니다.	SYS\$SYSROOT:[SYSTEST]
SYS\$UPDATE	시스템 업데이트 파일의 장치 및 디렉토리 이름입니다.	SYS\$SYSROOT:[SYSUPD]

공유 논리 이름 표의 기본 보호

운영 체제에서 제공하는 공유 논리 이름 표는 기본 보호로 만들어집니다. 표 11-6에는 공유 논리 이름 표의 각 유형에 대한 기본 보호가 표시됩니다.

표 11-6 공유 논리 이름 표의 기본 보호

표 유형	표 이름	기본 보호
작업 표	LNM\$JOB_xxxxxxxx ^a	SYSTEM=RWCD, OWNER=RWCD, GROUP=NO ACCESS, WORLD=NO ACCESS
그룹 표	LNM\$GROUP_gggggg ^b	SYSTEM=RWCD, OWNER=R, GROUP=R, WORLD=NO ACCESS
시스템 표	LNM\$SYSTEM_TABLE	SYSTEM=RWC, OWNER=RWC, GROUP=R, WORLD=R
클러스터 전체 시스템 표	LNM\$SYSCLUSTER_TABLE	SYSTEM=RWC, OWNER=RWC, GROUP=R, WORLD=R

표 11-6 공유 논리 이름 표의 기본 보호 (계속)

표 유형	표 이름	기본 보호
클러스터 전체 상위 표	LNМ\$CLUSTER_TABLE	SYSTEM=RWC, OWNER=RWC, GROUP=R, WORLD=R
사용자 작성 표	사용자 지정	SYSTEM=RWCD, OWNER=RWCD, GROUP=NO ACCESS, WORLD=NO ACCESS

- 문자열 `xxxxxxxx`는 작업 정보 블록의 주소인 8자리 16진수를 나타냅니다.
- 문자열 `gggggg`는 프로세스의 UIC 그룹 번호를 포함하는 6자리 8진수를 나타냅니다.

공유 논리 이름 관리를 위한 특권 및 액세스 요구 사항

표 11-7에는 공유 논리 이름 표의 각 유형에 있는 논리 이름 만들기, 삭제 및 읽기(변환)에 필요한 특권 및 액세스 권한이 표시됩니다. 특권, 액세스 유형 및 액세스 제어에 대한 자세한 내용은 이 매뉴얼의 10장을 참조하십시오.

표 11-7 공유 논리 이름 작업에 필요한 특권 또는 액세스 유형

이름이 있는 표	표 이름	작업	필요한 특권 또는 액세스 유형
작업 표	LNМ\$JOB_XXXXXXXX ^a	논리 이름 만들기 또는 삭제	이름이 만들어지거나 이름이 삭제되는 표에 대한 WRITE(W) 액세스
		논리 이름 읽기(변환)	이름이 있는 표에 대한 READ(R) 액세스
그룹 표	LNМ\$GROUP_gggggg ^b	논리 이름 만들기 또는 삭제	이름이 만들어지거나 이름이 삭제되는 표에 대한 WRITE(W) 액세스 또는 GRPNAM 특권
		논리 이름 읽기(변환)	이름이 있는 표에 대한 READ(R) 액세스
시스템 표	LNМ\$SYSTEM_TABLE	논리 이름 만들기 또는 삭제	시스템 UIC 그룹 번호(0과 시스템 매개 변수 MAXSYSGROUP의 값 사이) 또는 SYSNAM 특권

표 11-7 공유 논리 이름 작업에 필요한 특권 또는 액세스 유형 (계속)

이름이 있는 표	표 이름	작업	필요한 특권 또는 액세스 유형
		논리 이름 읽기(변환)	이름이 있는 표에 대한 READ(R) 액세스
클러스터 전체 시스템 표	LNMSYSCLUSTER_ TABLE	논리 이름 만들 기 또는 삭제	시스템 UIC 그룹 번호(0과 시스템 매개 변수 MAXSYSGROUP 의 값 사이) 또는 SYSNAM 특권
		논리 이름 읽기(변환)	이름이 있는 표에 대한 READ(R) 액세스
클러스터 전체 상위 표	LNMCCLUSTER_ TABLE	논리 이름 만들 기 또는 삭제	시스템 UIC 그룹 번호(0과 시스템 매개 변수 MAXSYSGROUP 의 값 사이)
		논리 이름 읽기(변환)	이름이 있는 표에 대한 READ(R) 액세스
사용자 작성 표	사용자 지정	논리 이름 만들 기 또는 삭제	이름이 만들어지거나 이름이 삭제되는 표에 대 한 WRITE(W) 액세스
		논리 이름 읽기(변환)	이름이 있는 표에 대한 READ(R) 액세스

- 문자열 **xxxxxxxx**는 작업 정보 블록의 주소인 8자리 16진수를 나타냅니다.
- 문자열 **gggggg**는 프로세스의 **UIC** 그룹 번호를 포함하는 6자리 8진수를 나타냅니다.

논리 이름 표 만들기

CREATE/NAME_TABLE 명령은 논리 이름 표를 만들고 디렉토리 논리 이름 표 중 하나에서 이 표를 카탈로그로 만듭니다. 논리 이름 표를 식별하거나 반복적으로 논리 이름 표로 변환되는 논리 이름은 항상 디렉토리 논리 이름 표 중 하나에 입력되어야 합니다.

프로세스-개인 논리 이름 표 만들기

사용자 프로세스에 대한 개인 논리 이름 표를 만들려면 LNM\$PROCESS_DIRECTORY(기본값)에 표를 만듭니다.

디렉토리 표의 이름은 1-31자를 포함할 수 있습니다. 대문자 영숫자 문자, 달러 기호(\$) 및 밑줄(_)만 유효합니다. 소문자 표 이름을 지정하면 자동으로 대문자로 변환됩니다.

다음 보기는 TAX라는 프로세스-개인 논리 이름 표를 만들어 논리 이름 CREDIT의 정의를 표에 놓은 다음 표 작성을 확인합니다. SHOW LOGICAL/TABLE 명령을 사용하면 표시할 논리 이름 표를 지정할 수 있습니다.

```
$ CREATE/NAME_TABLE TAX
$ DEFINE/TABLE=TAX CREDIT [ACCOUNTS.CURRENT]CREDIT.DAT
$ SHOW LOGICAL/TABLE=TAX CREDIT
```

```
"CREDIT" = "[ACCOUNTS.CURRENT]CREDIT.DAT" (TAX)
```

파일 조회 중에 시스템에서 새 표를 자동으로 검색하게 하려면 다음 보기에서와 같이 LNM\$PROCESS를 재정의할 수 있습니다.

```
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$PROCESS LNM$PROCESS_TABLE, TAX
```

공유 논리 이름 표 만들기

공유 논리 이름 표를 만들려면 /PARENT_TABLE 한정자를 사용하여 공유 표 이름을 지정합니다. 예를 들면 다음과 같습니다.

```
$ CREATE/NAME_TABLE/PARENT_TABLE=LNM$SYSTEM_DIRECTORY NEWTAB
```

클러스터 전체 논리 이름 표 만들기

다른 공유 논리 이름 표를 만드는 것과 같은 방식으로 클러스터 전체 논리 이름 표를 만들 수 있습니다. 클러스터 전체 논리 이름 표는 공유 논리 이름 표의 특수 유형이며 모든 공유 논리 이름 표에 적용되는 특권 및 액세스 요구 사항을 따릅니다(441페이지의 “특권 및 액세스 요구 사항” 참조).

다음 보기는 클러스터 전체 논리 이름 표를 만드는 방법을 보여 줍니다.

```
$ CREATE/NAME_TABLE/PARENT_TABLE=LNMS$CLUSTER_TABLE -  
_$_ new_clusterwide_logical_name_table
```

새 클러스터 전체 논리 이름 표에 있게 되는 클러스터 전체 논리 이름을 만들려면 다음 보기에서와 같이 **/TABLE** 한정자를 사용하여 새 표 이름을 지정하여 **DEFINE** 명령으로 새 클러스터 전체 논리 이름을 정의합니다.

```
$ DEFINE/TABLE=new_clusterwide_logical_name_table  
logical_name -  
_$_ equivalence_string
```

특권 및 액세스 요구 사항

특권이 있는 사용자는 특수 목적으로 공유 논리 이름 표를 만들 수 있습니다. 예를 들어, 어플리케이션에서 한 개 이상의 공유 논리 이름 표를 만들어 파일 위치와 같은 정보를 어플리케이션 사용자에게 전달하도록 할 수 있습니다.

```
$ CREATE/NAME_TABLE APPX_FILE_LOCATOR /PARENT=LNMS$SYSTEM_DIRECTORY -  
_$_ /PROTECTION = (S:RWD,O:RWD,G:R,W:R)
```

공유 논리 이름 표를 만들려면 다음이 있어야 합니다.

- 상위 표에 대한 **CREATE(C)** 액세스
- **SYSPRV** 특권 또는 **LNMS\$SYSTEM_DIRECTORY**에 대한 **WRITE(W)** 액세스

공유 논리 이름 표를 삭제하려면 다음이 있어야 합니다.

- 표에 대한 **DELETE(D)** 액세스
- **SYSPRV** 특권 또는 **LNMS\$SYSTEM_DIRECTORY**에 대한 **WRITE(W)** 액세스

기본 보호 수정

운영 체제는 자신이 만들거나 사용자가 만든 공유 논리 이름 표에 대한 기본 보호를 제공합니다. 기본 보호는 시스템 관리자나 표 소유자가 수정할 수 있는 보안 프로파일에 저장됩니다. 자세한 내용은 *OpenVMS Guide to System Security*를 참조하십시오.

다음과 같은 방법으로 사용자가 만든 표의 기본 보호를 수정할 수 있습니다.

- **DCL CREATE/NAME_TABLE** 명령과 함께 **/PROTECTION** 한정자를 사용합니다. 이 명령을 사용하면 **UIC** 기반 보호를 설정할 수 있습니다.
- **SET SECURITY/ACL/OBJECT_TYPE=LOGICAL_NAME_TABLE** 명령이나 **ACL** 편집기로 이미 만든 표에 **ACL** 보호를 적용합니다.

공유 논리 이름 표의 **ACL**은 시스템 부팅 사이에서 저장되지 않습니다. 시스템이 부팅될 때마다 이러한 논리 이름 표에 대해 **ACL**을 다시 설정해야 합니다.

공유 논리 이름 표에 대한 **ACL** 보호 적용에 대한 자세한 내용은 *OpenVMS DCL Dictionary*의 **SET SECURITY/ACL** 명령을 참조하십시오.

논리 이름 표의 할당량 설정

할당량은 주어진 논리 이름 표에서 소비할 수 있는 시스템 리스소 양을 제한할 때 사용됩니다. 프로세스, 그룹 및 시스템 논리 이름 표에는 무제한 할당량이 있습니다. 기본적으로 논리 이름 표를 만들면 이 표에도 무제한 할당량이 있습니다.

할당량을 지정하여 사용자가 만든 논리 이름 표의 크기(바이트)를 제한할 수 있습니다. 논리 이름을 만들기 전에 표에 남아있는 할당량과 비교하여 해당 데이터 구조의 크기를 검사합니다. 새 항목에 사용할 수 있는 할당량이 충분하지 않으면 시스템에서 오류 메시지를 표시합니다.

한 번 표의 할당량을 설정하면 변경할 수 없습니다. 표에 여유 공간이 없으면 **DEASSIGN** 명령을 사용하여 이전 논리 이름을 삭제합니다. 이렇게 하면 새 논리 이름에 필요한 공간이 확보됩니다.

다음 보기에서는 논리 이름 표 **ABC**가 만들어지고 **500**바이트의 할당량이 제공됩니다.

```
$ CREATE/NAME_TABLE/QUOTA=500 ABC
```

작업 표 할당량 설정

작업 논리 이름 표는 공유 표입니다. 표가 만들어지면 작업 논리 이름 표의 할당량이 설정됩니다. 할당량은 한 개 이상의 다음 기준에 의해 결정됩니다.

- 시스템 사용자 인증 파일 **SYSUAF.DAT**의 사용자에게 대해 설정된 **JTQUOTA** 값
(프로세스에서 활성화한 첫 번째 이미지가 시스템 이미지 **LOGINOUT**인 경우)
- 프로세스 작성(**\$CREPRC**) 시스템 서비스에 대한 호출에 지정된 **PQL\$_JTQUOTA** 할당량 목록 값
- 분리된 프로세스를 만드는 데 사용된 **RUN** 명령의 **/JOB_TABLE_QUOTA** 한정자 값
- **SYSGEN** 매개 변수 **PQL_DJTQUOTA**(이전 조건이 적용되지 않은 경우). 이 매개 변수의 표준 기본 값은 1024바이트이지만 시스템 관리자가 값을 변경할 수 있습니다. 매개 변수 **PQL_DJTQUOTA**(기본 작업 논리 이름 표 할당량)와 **PQL_MJTQUOTA**(최소 작업 논리 이름 표 할당량)의 값을 표시하고 설정하는 데 시스템 생성 유틸리티 (**SYSGEN**)를 사용할 수 있습니다.

작업 논리 이름 표에 대한 할당량 값이 0이라는 것은 할당량이 없다는 의미입니다.

실용적인 모든 용도를 위해 할당량은 제한되지 않습니다.

논리 이름 변환 순서 수정

LNM\$FILE_DEV는 검색할 논리 이름 표와 모든 논리 이름 변환의 검색 순서 모두를 정의합니다. 일반적으로는 기본 검색 순서를 수정할 필요가 없습니다. 하지만 사용자가 LNM\$FILE_DEV로 지정한 표 앞에서 처음으로 검색한 새 프로세스-개인 논리 이름 표의 이름을 추가해야 할 경우가 있습니다. 마찬가지로 시스템 관리자가 LNM\$FILE_DEV로 지정한 표 앞에서 검색할 한 개 이상의 공유 논리 이름 표의 이름을 추가해야 할 경우가 있습니다. 시스템에서 처음으로 검색할 새 논리 이름 표를 사용하여 LNM\$FILE_DEV의 프로세스-개인 정의를 만들려면 다음을 수행하십시오.

1. 새 논리 이름을 포함하는 파일을 만듭니다.
2. 이 새 파일을 새 논리 이름 표로 변환합니다.
3. 프로세스 논리 이름 디렉토리 표를 상위 표로 지정하여 LNM\$FILE_DEV의 개인 정의를 만듭니다.
4. LNM\$FILE_DEV의 개인 정의에 있는 표 이름 목록 시작 부분에 새 논리 이름 표 이름을 추가합니다.

다음 보기에서는 새 논리 이름 표, NEWTAB을 만들고 검색할 첫 번째 표로 나열된 NEWTAB을 사용하여 LNM\$FILE_DEV의 프로세스-개인 정의를 만듭니다.

```
$ CREATE/NAME_TABLE NEWTAB
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$FILE_DEV -
_$ NEWTAB, LNM$PROCESS, LNM$JOB, LNM$GROUP, LNM$SYSTEM
```

위의 보기에서 시스템은 다음과 같은 이유로 NEWTAB 표를 먼저 검색합니다.

- 기본 시스템 버전 대신 LNM\$FILE_DEV의 프로세스-개인 정의가 사용됩니다.
- LNM\$FILE_DEV 내에서 NEWTAB은 다른 논리 이름 표 앞에 나열됩니다.

새 논리 이름 표를 LNM\$FILE_DEV의 시스템 정의에 추가하려면 SYSNAM 또는 SYSPRV 특권이 있어야 합니다.

다음 보기는 NEWTAB이 프로세스-개인 표가 아닌 공유 표로 작성되었다는 점만 제외하면 이전 보기와 비슷합니다.

```
$ CREATE/NAME_TABLE/PARENT=LNM$SYSTEM_DIRECTORY NEWTAB
$ DEFINE/TABLE=LNM$SYSTEM_DIRECTORY LNM$FILE_DEV -
_$ NEWTAB, LNM$PROCESS, LNM$JOB, LNM$GROUP, LNM$SYSTEM
```


또한, `LNМ$FILE_DEV`에서 정의한 검색 목록에서 논리 이름 표를 제거할 수 있습니다. 다음 보기에서는 프로세스 및 시스템 논리 이름 표만 포함하는 `LNМ$FILE_DEV`의 프로세스-개인 정의를 만듭니다. 프로세스-개인 정의에는 `LNМ$JOB`과 `LNМ$GROUP`이 포함되지 않으므로 논리 이름을 변환해야 할 후속 명령에서 작업이나 그룹 표를 검색하지 않습니다.

```
$ DEFINE/TABLE=LNМ$PROCESS_DIRECTORY -  
_$ LNМ$FILE_DEV LNМ$PROCESS, LNМ$SYSTEM
```

논리 이름 표 삭제

논리 이름 표를 삭제하려면 이 표(시스템 또는 프로세스 디렉토리 논리 이름 표)를 포함하는 표 및 표 이름을 지정합니다. 상위 논리 이름 표를 삭제하면 하위 표 및 하위 표 자체의 모든 논리 이름이 삭제됩니다.

공유 논리 이름 표를 삭제하려면 표에 대한 **DELETE** 액세스 또는 **SYSPRV** 특권이 있어야 합니다.

다음 보기의 명령은 논리 이름 **WORKFILE**을 삭제합니다.

```
$ DEASSIGN WORKFILE
```

다음 보기의 명령은 프로세스 디렉토리 표에서 논리 이름 표 **TAX**를 삭제합니다.

```
$ DEASSIGN/TABLE=LNMP$PROCESS_DIRECTORY TAX
```

영구 프로세스 논리 이름

DCL은 사용자가 로그인할 때 영구 프로세스 논리 이름을 만듭니다. 이러한 이름은 프로세스 수명 동안에 정의된 상태로 남아 있습니다. 이러한 논리 이름은 지정 취소할 수 없습니다. **DEFINE** 명령에서 다른 등가 문자열을 지정하여 논리 이름을 재정의할 수 있지만 재정의된 이름을 나중에 지정 취소하면 영구 프로세스 이름이 재설정됩니다.

다음 영구 프로세스 논리 이름을 사용할 수 있습니다.

- **SYS\$INPUT**
기본 입력 장치나 파일을 참조하는 논리 이름입니다.
- **SYS\$OUTPUT**
기본 출력 장치나 파일을 참조하는 논리 이름입니다.
- **SYS\$ERROR**
시스템에서 메시지를 기록하는 기본 장치나 파일을 참조하는 논리 이름입니다.
- **SYS\$COMMAND**
로그인할 때 **SYS\$INPUT** 값을 참조하는 논리 이름입니다.

대화형 프로세스와 일괄 처리 프로세스의 등가 이름 차이점

시스템을 대화형으로 사용하면 DCL은 **SYS\$INPUT**, **SYS\$OUTPUT**, **SYS\$ERROR** 및 **SYS\$COMMAND**를 사용자 터미널에 일치시킵니다. 하지만 명령 프로시저를 실행하고 일괄 처리 작업을 전송하면 DCL이 이러한 논리 이름에 대한 새 등가 문자열을 만듭니다.

명령 프로시저를 대화형으로 실행하면 다음이 발생합니다.

- **SYS\$INPUT**은 명령 프로시저와 일치합니다. 그러므로 DCL은 명령 프로시저에서 데이터를 가져옵니다. 이 할당은 임시적입니다. 명령 프로시저가 종료되면, **SYS\$INPUT**은 원래 값을 다시 얻습니다.
- **SYS\$OUTPUT**, **SYS\$COMMAND** 및 **SYS\$ERROR**는 터미널에 일치된 상태로 남아 있습니다.

일괄 처리 작업을 전송하면 다음이 발생합니다.

- **SYS\$INPUT** 및 **SYS\$COMMAND**는 일괄 처리 작업 명령 프로시저와 일치합니다.
- **SYS\$OUTPUT** 및 **SYS\$ERROR**는 일괄 처리 작업 로그 파일과 일치합니다.

명령 프로시저를 중첩하는 경우(즉, 다른 명령 프로시저를 실행하는 명령 프로시저를 작성하는 경우), **SYS\$INPUT**의 등가 문자열은 현재 실행 중인 명령 프로시저를 가리키도록 변경됩니다. 하지만 **SYS\$OUTPUT**, **SYS\$ERROR** 및 **SYS\$COMMAND**의 등가 문자열은 사용자가 명시적으로 변경하지 않는 한 같은 상태로 남아 있습니다.

또한, 파일을 여는 명령을 입력하면 **DCL**은 이 파일을 영구 프로세스 파일로 엽니다. 예를 들어, **OPEN** 명령을 사용하여 파일을 열면 이 파일이 영구 프로세스 파일로 열립니다. 파일을 명시적으로 닫거나 로그아웃할 때까지 파일은 열린 상태로 남아 있습니다.

영구 프로세스 파일은 메모리의 특수 영역에 저장됩니다. 동시에 여러 파일을 열린 상태로 두면 이 영역이 모두 사용하게 될 수 있습니다. 이러한 경우 일부 파일을 닫거나 로그아웃해야 합니다.

영구 프로세스 논리 이름을 사용하여 파일 I/O 재지정

영구 프로세스 논리 이름을 사용하여 파일 I/O를 재지정할 수 있습니다. 명령 프로시저에서는 이러한 이름을 사용하여 터미널의 데이터를 읽고 데이터를 표시할 수 있습니다(13장 및 14장 참조). **DCL**은 **SYS\$INPUT** 및 **SYS\$COMMAND**의 새 정의를 무시합니다.

OpenVMS Version 7.1에서는 **DCL PIPE** 명령이 사용됩니다. **PIPE** 명령은 파일 I/O를 재지정할 수 있는 대체 방법입니다. **PIPE** 명령에 대한 자세한 내용은 *OpenVMS DCL Dictionary: N-Z*를 참조하십시오.

SYS\$INPUT 재정의

SYS\$INPUT을 재정의하여 명령 프로시저에서 호출한 이미지가 터미널이나 다른 파일에서 입력을 읽어들이도록 할 수 있습니다. **DCL**은 항상 기본 입력 스트림에서 입력을 가져오므로 **DCL**은 **SYS\$INPUT** 재정의를 무시합니다.

다음 보기의 명령은 새 명령 프로시저 파일의 일부입니다. **DEFINE** 명령은 **SYS\$INPUT**을 **SYS\$COMMAND**로 재정의합니다. **SYS\$COMMAND**는 사용자가 로그인한 경우 초기 입력 스트림인, 터미널을 참조합니다. 이 새 정의를 사용하면 명령 프로시저에서 호출한 이미지는 특정 기간 동안만 명령 프로시저 파일(기본값)이 아닌 터미널에서 입력을 가져옵니다.

/USER_MODE 한정자는 SYS\$INPUT이 다음 이미지의 지속 기간 동안에만 재정의됨을 명령 프로시저에 알립니다. 이 보기에서 다음 이미지는 편집기입니다. 편집기를 종료하면 SYS\$INPUT은 기본값을 계속 사용합니다. 이 경우 기본값은 명령 프로시저 파일입니다.

```
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$ EDIT/TPU MYFILE.DAT
.
.
.
```

SYS\$OUTPUT 재정의

SYS\$OUTPUT을 재정의하여 기본 장치에서 다른 파일로 출력을 재지정할 수 있습니다.

SYS\$OUTPUT을 재정의하면 시스템에서 사용자가 논리 이름 할당에 지정한 이름을 사용하여 파일을 엽니다. SYS\$OUTPUT을 재정의하면 모든 후속출력이 새 파일로 지정됩니다.

SYS\$OUTPUT을 지정 취소해야 합니다. 그렇지 않으면 출력이 사용자가 지정한 파일에 계속 쓰여집니다. 사용자 모드에서 SYS\$OUTPUT을 재정의하여(DEFINE/USER_MODE 사용) 이미지로부터의 출력을 재지정할 수 있습니다. 이러한 정의는 다음 명령 이미지가 실행될 때까지만 유효합니다. 한 번 명령 이미지가 실행되면 즉, 출력이 파일에서 캡처되면 논리 이름 SYS\$OUTPUT은 해당 기본값을 계속 사용합니다.

로그인할 때 시스템에서 SYS\$OUTPUT이라는 두 개의 논리 이름을 만듭니다. 한 이름이 실행 모드에서 만들어지면 다른 이름은 감독자 모드에서 만들어집니다. SYS\$OUTPUT을 재정의하여 감독자 모드 논리 이름을 대체할 수 있습니다. 감독자 모드 이름을 지정 취소한 경우 시스템에서 실행 모드 등가 문자열을 사용하여 감독자 모드에서 SYS\$OUTPUT을 재정의합니다. 실행 모드 이름은 지정 취소할 수 없습니다.

SYS\$OUTPUT을 파일로 재정의하면 사용자가 지정한 파일로 출력이 지정되더라도 논리 이름에는 파일 지정 방법의 장치 부분만 포함됩니다.

SYS\$OUTPUT을 재정의할 때 사용자가 지정하는 파일을 시스템에서 열 수 없는 경우 오류 메시지를 표시합니다.

SYS\$OUTPUT을 재정의한 다음 대부분의 명령은 출력을 파일의 기존 버전으로 지정합니다. 하지만 특정 명령은 출력을 쓰기 전에 새 버전의 파일을 만듭니다.

다음 보기에서 **SYS\$OUTPUT**은 **SHOW DEVICES** 명령이 입력되기 전에 **MYFILE.LIS**로 정의됩니다. **SHOW DEVICES**로 만든 표시는 터미널이 아닌 현재 디렉토리의 **MYFILE.LIS**로 지정됩니다. 다른 텍스트 파일을 조작하는 방식으로 데이터를 조작할 수 있습니다.

```
$ DEFINE SYS$OUTPUT MYFILE.LIS
$ SHOW DEVICES
```

다음 보기에서 **SYS\$OUTPUT**은 **TEMP.DAT** 파일로 재정의됩니다. **SYS\$OUTPUT**이 재정의되면 **DCL** 및 이미지의 출력은 **TEMP.DAT** 파일로 지정됩니다.

SHOW LOGICAL 명령과 **SHOW TIME** 명령의 출력도 **TEMP.DAT**로 보내집니다.

SYS\$OUTPUT이 지정 취소되면 시스템에서 파일 **TEMP.DAT**를 닫고 **SYS\$OUTPUT**을 터미널로 재정의합니다. **TYPE** 명령이 입력되면 **TEMP.DAT**에 수집된 출력이 터미널에 표시됩니다.

```
$ DEFINE SYS$OUTPUT TEMP.DAT
$ SHOW LOGICAL SYS$OUTPUT
$ SHOW TIME
$ DEASSIGN SYS$OUTPUT
$ TYPE TEMP.DAT

"SYS$OUTPUT" = "DISK1:" (LNM$PROCESS_TABLE)
06-MAY-1996 13:26:53
```

SYS\$OUTPUT을 재정의하면 등가 문자열은 전체 파일 지정 방법이 아닌 장치 이름 **DISK1**을 포함합니다.

SYS\$ERROR 재정의

SYS\$ERROR를 재정의하여 지정한 파일로 오류 메시지를 지정할 수 있습니다. 하지만 **SYS\$ERROR**를 재정의하는 경우 **SYS\$OUTPUT**과 다르게 되고(또는 **SYS\$ERROR**는 재정의하지 않고 **SYS\$OUTPUT**만 재정의하는 경우), **DCL** 명령은 정보, 경고, 오류 및 심각한 오류 메시지를 **SYS\$ERROR**와 **SYS\$OUTPUT** 모두에 보냅니다. 그러므로 이러한 메시지를 두 번 받게 됩니다. 한 번은 **SYS\$ERROR** 정의에서 표시한 파일에서 그리고 또 한 번은 **SYS\$OUTPUT**에서 표시한 파일에서 받게 됩니다. 성공 메시지는 **SYS\$OUTPUT**에서 표시한 파일에만 보내집니다.

표준 오류 표시 메커니즘을 사용하는 **DCL** 명령 및 이미지는 **SYS\$ERROR**가 **SYS\$OUTPUT**과 다르더라도 **SYS\$ERROR**와 **SYS\$OUTPUT** 모두에 오류 메시지를 보냅니다. 하지만 **SYS\$ERROR**를 재정의한 다음 **SYS\$ERROR**를 참조하는 이미지를 실행하면 해당 이미지는 **SYS\$ERROR**에서 표시한 파일로만 오류 메시지를 보냅니다. **SYS\$ERROR**가 **SYS\$OUTPUT**과 다르더라도 **SYS\$ERROR**에서 표시한 파일로만 오류 메시지를 보냅니다.

SYS\$COMMAND 재정의

SYS\$COMMAND를 재정의할 수 있지만, DCL에서는 사용자 정의를 무시합니다. DCL에서는 항상 처음 입력 스트림의 기본 정의를 사용합니다. 하지만 SYS\$COMMAND를 참조하는 이미지를 실행하면 이미지에서 사용자의 새 정의를 사용할 수 있습니다.

장치 및 파일의 논리 이름 정의

영구 프로세스 논리 이름

12 기호, 명령 및 표현식 정의

기호는 숫자, 문자 또는 논리 값(true 또는 false 등)을 나타내는 이름입니다. DCL 명령행에 기호를 사용하면 DCL은 명령을 실행하기 전에 그 기호를 해당 값으로 대체합니다.

매개 변수, 여러 한정자 및 값을 포함하는 DCL 명령행을 입력하려면 많은 수고와 시간이 필요할 수 있습니다. DCL과의 상호 작용을 단순하게 하여 시간을 절약하려면 자주 입력하는 명령 대신 사용할 기호를 설정할 수 있습니다.

또한, 명령 프로시저에서 기호를 사용하여 특정 유형의 데이터를 수집, 저장 및 조작할 수도 있습니다. 명령 프로시저에 대한 자세한 내용은 13장 및 14장을 참조하십시오.

이 장에서는 다음을 설명합니다.

- 기호 사용
- 기호 표시
- 다른 기호로 기호 사용
- 기호를 사용한 데이터 저장 및 조작
- 문자열
- 숫자 값 및 식 사용
- 논리 값 및 식 사용
- 표현식의 값 유형 변환
- 기호표 이해
- 기호 값 마스크
- 기호 대체 이해
- 명령 처리의 3 단계
- 기호 사용에 대한 대안: 자동 외부 명령

추가 정보는 다음을 참조하십시오.

- 기호 및 그 사용에 관한 자세한 내용은 *OpenVMS DCL Dictionary*를 참조하십시오.
- 새 명령 정의에 대한 자세한 내용은 *OpenVMS Command Definition, and Message Utilities*를 참조하십시오.

기호

기호를 다음과 같이 사용할 수 있습니다.

- 명령, 매개 변수 또는 명령행의 동의어로 사용.
긴 명령행을 입력하지 않고 대신 사용할 기호를 만들 수 있습니다.
- 기호 이름만 입력하여 이미지를 실행할 수 있는 외부 명령을 정의.
이 명령은 **DCL**이 인식하지 않으므로 “외부”입니다.
- 명령 프로시저 파일에서 조건 실행과 변수 대체 등의 프로그램 작업을 실행.
표현식의 변수로 기호를 사용하거나 명령 프로시저로 매개 변수를 보내고 받기 위해 기호를 사용할 수 있습니다. 또한, **READ**, **WRITE** 및 **INQUIRE**와 같은 **DCL** 명령에서는 데이터 레코드를 참조하기 위해 기호를 사용합니다.

다음 보기에서는 자주 액세스하는 디렉토리를 기본값으로 설정하는 기호를 만듭니다. 이 명령은 기호 **WORK**를 정의 및 사용하여 기본값을 **WORK1:[JONES.WORK]** 디렉토리로 설정하는 방법을 보여 줍니다

```
$ WORK ::= SET DEFAULT DISK1:[JONES.WORK]
$ WORK
$ SHOW DEFAULT
  DISK1:[JONES.WORK]
```

논리 이름 및 기호 비교

논리 이름과 기호는 비슷한 것 같지만 다르게 사용됩니다. 다음 표에서 논리 이름과 기호의 기능, 용도 및 기타 특성을 비교합니다.

표 12-1

특성	논리 이름	기호
기능	장치, 파일, 대기열 및 다른 시스템 객체 사양을 나타냅니다.	명령 또는 명령 문자열의 일부를 나타냅니다.
용도	전체 장치, 디렉토리, 파일, 대기열 또는 다른 시스템 객체 사양 대신 사용됩니다. 논리 이름은 변환을 위해 파일 시스템에 전달될 명령 문자열 매개 변수의 일부로 사용해야 합니다.	명령 문자열 대신 사용됩니다. 기호는 명령 언어 인터프리터에서 변환할 명령 문자열의 첫 단어로 사용해야 합니다.
저장	사용자의 프로세스, 작업, 그룹 또는 시스템 논리 이름 표에 저장됩니다. 440페이지의 “논리 이름 표 만들기”를 참조하십시오.	글로벌 또는 로컬 기호 표에 저장됩니다. 485페이지의 “기호 표 이해”를 참조하십시오.
만들기	ASSIGN이나 DEFINE 명령을 사용하여 논리 이름을 만듭니다. 416페이지의 “논리 이름 삭제”를 참조하십시오.	할당문(= or ==)을 사용하여 기호를 만듭니다. 457페이지의 “기호 사용”을 참조하십시오.
표시	SHOW LOGICAL 또는 SHOW TRANSLATION 명령을 사용하여 논리 이름을 표시합니다. 420페이지의 “논리 이름 표시”를 참조하십시오.	SHOW SYMBOL 명령을 사용하여 기호를 표시합니다. 460페이지의 “기호 표시”를 참조하십시오.
삭제	DEASSIGN 명령을 사용하여 논리 이름을 삭제합니다. 416페이지의 “논리 이름 삭제”를 참조하십시오.	DELETE/SYMBOL 명령을 사용하여 기호를 삭제합니다. 459페이지의 “기호 삭제”를 참조하십시오.

기호 사용

로컬 및 글로벌의 두 가지 유형의 기호를 만들 수 있습니다. **로컬 기호**는 현재 명령 수준과 현재 명령 수준에서 실행된 명령 프로시저에서 액세스할 수 있습니다. **글로벌 기호**는 모든 명령 수준에서 액세스할 수 있습니다.

기호는 **문자열**, 숫자, 렉시칼 함수, 논리 값 또는 다른 기호로 정의할 수 있습니다. 기호 이름은 1~255자가 될 수 있고 문자, 밑줄(_) 또는 달러 기호(\$)로 시작해야 합니다. 기호 이름에서 대소문자는 모두 대문자로 처리됩니다.

기호를 만들려면 **할당문**(= or ==)이나 문자열 할당(:= or :=)을 사용합니다. 문자열 할당을 사용할 경우 모든 영문자는 대문자로 변환되고 여러 공백과 탭은 단일 공백으로 압축됩니다. 문자열 할당을 사용하여 **DCL** 명령을 나타내는 기호를 만들거나 외부 명령을 정의할 수 있습니다. 두 경우 모두 255자를 초과할 수 없습니다. 문자열 할당에서 두 줄을 넘어 문자열을 계속하려면 한 개의 하이픈을 사용합니다.

READ 및 **INQUIRE** 명령을 사용하여 기호를 만들 수도 있습니다(13장 및 14장 참조).

로컬 기호 만들기

다음 보기에서는 로컬 기호 **SS**를 **DCL** 명령 **SHOW SYMBOL**에 할당합니다.

```
$ SS = "SHOW SYMBOL"
```

다음 보기에서는 로컬 기호 **DB**를 **DCL** 명령 **DIRECTORY ACCOUNTS:[BOLIVAR]:**에 할당합니다.

```
$ DB := DIRECTORY ACCOUNTS:[BOLIVAR]
```

글로벌 기호 만들기

다음 보기에서는 글로벌 기호 **DC**를 사용하여 **DCL** 명령행을 나타냅니다. 이 기호 이름을 입력하면 지정된 한정자와 함께 **DCL** 명령 **DIRECTORY**가 실행됩니다.

```
$ DC == "DIRECTORY/SIZE=ALL DISK1:[JONES.TAX]MONEY.LIS"
```

다음 보기에서는 글로벌 기호 **READY**를 사용하여 **DCL** 명령행을 나타냅니다. 이 기호 이름을 입력하면 지정된 한정자와 함께 **DCL** 명령 **PRINT**가 실행됩니다.

```
$ READY := PRINT/CONFIRM/QUEUE=AKI$LN03/NOTIFY/RESTART
$ READY FILE.DAT
```

기호를 사용한 DCL 명령 표시

로그인 명령 파일(LOGIN.COM)에서나 DCL 수준에서 대화형으로 DCL 명령을 나타내는 기호를 정의할 수 있습니다. 로그인 명령 파일에 기호를 정의할 경우 로그인할 때마다 기호를 사용할 수 있지만 대화형으로 기호를 정의할 경우에는 현재 프로세스 동안에만 기호를 사용할 수 있습니다.

DCL 명령과 같은 이름으로 기호를 정의하면 정의한 기호가 DCL 명령 이름보다 우선합니다. 예를 들어, 기호 **HELP**를 명령 **TYPE HELP.LST**로 정의하면 더 이상 **HELP**를 입력하여 시스템의 도움말 유틸리티를 호출할 수 없습니다.

기호 약어

간략화할 수 있는 기호를 만들려면 별표(*)를 사용합니다. 일반적으로 기호를 사용할 수 있는 모든 상황에서 간략화한 기호 정의를 사용할 수 있습니다. 부분 문자열 대체를 포함하는 기호는 예외입니다. 자세한 내용은 468페이지의 “부분 문자열 대체”를 참조하십시오.

기존 기호는 대체될 수 있습니다. 기존 기호가 별표나 별표 다음의 새 기호와 정확히 일치하면 새 기호가 기존 기호를 대체합니다. 또한, 이름이 별표나 별표 다음의 기존 기호와 부분적으로 일치하는 다른 기호를 정의할 수 없습니다.

다음 보기에서는 **PR**, **PRI** 또는 **PRIN**으로 간략화할 수 있는 로컬 기호 **PRINT**를 만듭니다.

```
$ PR*INT = "PRINT/CONFIRM/QUEUE=AKI$LN03/NOTIFY/RESTART"
```

지정된 한정자와 함께 DCL 명령 **PRINT**를 실행하려면 기호나 그 약어를 입력할 수 있습니다.

외부 명령 정의

비 DCL 이미지의 파일 지정 방법을 기호와 일치시키면 기호 이름을 입력하여 그 이미지를 실행할 수 있습니다. 이미지를 실행하는 기호를 외부 명령이라고 합니다. 외부 명령은 명령 인터프리터가 DCL 명령으로 인식하지 않는 이미지입니다. DCL 명령의 각 요소와 마찬가지로 외부 명령도 255자를 초과할 수 없습니다.

기호를 외부 명령으로 정의하는 형식은 다음과 같습니다.

```
symbol-name :[=] $image-file-spec  
symbol-name =[ ] "$image-file-spec"
```

기호 정의 시작 부분에서 달러 기호(\$)가 파일 지정 방법 앞에 오고 그 사이에 공백이 없으면 이미지 실행 요청을 의미합니다.

이미지 파일 지정 방법의 경우 기본 장치 및 디렉토리 이름은 **SYS\$SYSTEM**이고 기본 파일 형식은 **.EXE**이며 기본 파일 버전 번호는 최상위 버전입니다.

외부 명령을 사용하는 대신 **Command Definition** 유틸리티를 사용하여 새 명령을 정의할 수도 있습니다. 자세한 내용은 *OpenVMS Command Definition, Librarian, and Message Utilities Manual*을 참조하십시오.

기호를 지정하지 않고 외부 명령을 자동으로 실행하는 방법도 있습니다. 자세한 내용은 499 페이지의 “기호 사용에 대한 대안: 자동 외부 명령”을 참조하십시오.

다음 보기에서는 이미지 **DISK1:[ACCOUNTS]PRINTALL.EXE**를 실행하도록 글로벌 기호 **PRINTALL**을 정의합니다.

```
$ PRINTALL ::= $[ACCOUNTS]PRINTALL
```

명령행에서 **PRINTALL** 다음에 매개 변수를 입력할 수 있습니다.

다음 보기에서 파일 지정 방법 **RAT.DAT**는 **PRINTALL**에서 정의한 이미지에 전달되는 매개 변수입니다.

```
$ PRINTALL RAT.DAT
```

기호 대체

명령 인터프리터는 어포스트로피(')로 묶인 기호를 찾아 변환합니다. 따라서 어포스트로피 다음에 기호나 렉시칼 함수를 사용하여 매개 변수를 지정하면 기호 대체가 일어납니다(489 페이지의 “기호 대체 이해” 참조). 또는 명령 인터프리터에서 해당 줄을 구문 분석하지 않습니다. 이미지에서는 매개 변수가 필요하고 **구문 분석**이나 명령행 평가를 수행해야 합니다.

기호 삭제

DELETE/SYMBOL 명령은 기호를 삭제합니다. 글로벌 기호를 삭제하려면 **/GLOBAL** 한정자를 포함시킵니다. 예를 들어, 글로벌 기호 **TEMP**를 삭제하려면 다음 명령을 입력합니다.

```
$ DELETE/SYMBOL/GLOBAL TEMP
```

기호 표시

SHOW SYMBOL 명령은 기호 값을 표시합니다. 특정 기호 값을 표시하려면 **SHOW SYMBOL** 명령 다음에 해당 기호의 이름을 입력합니다. 특정 글로벌 기호의 값을 표시하려면 **/GLOBAL** 한정자를 포함시킵니다. **SHOW SYMBOL/ALL** 명령은 모든 로컬 기호를 표시하고 명령 **SHOW SYMBOL/ALL/GLOBAL**은 모든 글로벌 기호를 표시합니다.

기호에 정수 값이 있는 경우 **SHOW SYMBOL** 명령은 값을 10진, 16진 및 8진 표기법으로 표시합니다.

다음 보기에서는 기호 **PR**을 표시합니다.

```
$ SHOW SYMBOL PR
PR*INT = "PRINT/CONFIRM/COPIES=2/QUEUE=DOC$LN03/NOTIFY/RESTART"
```

다음 보기에서는 기호 **TOTAL**의 정수 값이 표시됩니다.

```
$ SHOW SYMBOL TOTAL
TOTAL = 4    Hex = 00000004    Octal = 00000000004
```


다른 기호로 기호 사용

정의한 기호를 다른 기호 정의의 일부로 사용할 수 있습니다. DCL에서는 기호를 사용하는 상황에 따라 해당 기호를 문자열이나 숫자로 변환합니다.

다음 보기에서는 정수 값 3을 기호 COUNT에 할당합니다.

```
$ COUNT = 3
```

그런 다음 COUNT 값을 다른 할당문에서 사용할 수 있습니다. 예를 들어, 다음에서는 COUNT 값에 1을 더합니다.

```
$ TOTAL = COUNT + 1
```

결과(4)는 기호 TOTAL과 일치시킵니다.

기호 연결

더하기 기호(+)로 여러 기호를 연결하여 긴 문자열을 만들 수 있습니다. 또한, 어포스트로피 (')로 각 기호 이름을 묶어 둘 이상의 기호를 연결할 수도 있습니다.

기호 대체 요청에 대한 자세한 내용은 491페이지의 “기호 대체 연산자”를 참조하십시오.

다음 보기에서는 기호 “Saturday”와 “Sunday”를 사용하여 기호 “WEEKEND”를 만듭니다.

```
$ DAY1 = "Saturday, "
$ DAY2 = "Sunday"
$ WEEKEND = DAY1 + DAY2
$ SHOW SYMBOL WEEKEND
WEEKEND = "Saturday, Sunday"
```

다음 보기에서는 어포스트로피를 사용하여 기호 NAME과 TYPE를 연결합니다.

```
$ NAME = "MYFILE"
$ TYPE = ".DAT"
$ PRINT 'NAME' 'TYPE'
```

이 PRINT 명령은 MYFILE.DAT의 복사본을 인쇄합니다.

문자열 할당에 기호 포함

문자열 할당에 로컬 기호를 포함시키려면 콜론과 등호(:=)를 사용하고 문자열 할당에 글로벌 기호를 포함시키려면 콜론과 두 개의 등호(==)를 사용합니다. 두 유형(로컬 또는 글로벌)의 기호 모두에서 기호는 어포스트로피('')로 묶습니다. 그렇지 않으면 DCL에서 이것을 기호로 인식하지 않습니다.

기호에 **null** 문자열을 정의하면 그 기호의 값은 **0**이 됩니다.

다음 보기에서는 기호 **COUNT**가 문자열 할당문에 포함됩니다.

```
$ BARK := P'COUNT'
```

앞의 보기에서는 **COUNT**에 정수 값 3이 할당되었고 이 보기에서는 **COUNT**가 문자열로 변환되어 문자 **P**에 추가됩니다. 따라서 기호 **BARK**의 값은 **P3**가 됩니다.

다음 보기에서 기호 **A**는 **null**입니다.

```
$ A = ""
$ B = 2
$ C = A + B
$ SHOW SYMBOL C
C = 2  Hex = 00000002  Octal = 00000000002
```

기호를 사용한 데이터 저장 및 조작

명령 프로시저에서 기호를 변수로 사용할 수 있습니다. 변수는 계산하거나 문자가 아닌 값으로 할당하는 값을 포함할 수 있습니다. 예를 들어, 렉시칼 함수 값을 변수에 할당하거나 파일 레코드의 값을 변수로 읽어 올 수 있습니다.

표현식은 값들의 조합입니다. 명령 프로시저의 경우 표현식은 기호 할당문(등호의 오른쪽), IF 문, WRITE 명령에서 사용되고 렉시칼 함수의 인수로 사용됩니다.

기호를 정의할 때 할당문의 왼쪽에서는 기호 이름을 정의하고 할당문의 오른쪽에는 표현식이 포함됩니다. 표현식의 각 값(**피연산자**라고도 함)은 **연산자**로 다른 값과 연결할 수 있습니다. DCL은 표현식을 계산하고 그 결과를 기호에 할당합니다. 표현식이 문자열로 계산되면 기호는 문자열 값이 됩니다.

다음의 보기에서 로컬 기호 BARK는 세 수를 더하는 표현식이 됩니다.

```
$ BARK = 1 + 2 + 3
```

피연산자는 1과 2와 3이고 연산자는 더하기 기호(+)입니다. 계산된 식이 정수이므로 기호도 정수 값이 됩니다.

문자열

문자열은 인쇄할 수 있는 문자를 포함할 수 있습니다. 부록 A에는 ASCII 문자 집합과 DEC 다국적 문자 집합에 대한 표가 있습니다. 이 표에는 문자열에 포함시킬 수 있는 문자가 나열되어 있습니다.

문자는 다음과 같은 세 가지 기본 범주로 나뉩니다.

- 영숫자 문자
대문자 A-Z, 소문자 a-z, 숫자 1-9, 달러 기호(\$), 밑줄(_) 및 하이픈(-)
- 특수 문자
표시 또는 인쇄할 수 있는 다른 모든 문자: 감탄 부호(!), 인용 부호("), 번호 기호(#) 등
- 인쇄할 수 없는 문자
인쇄 또는 표시할 수 없는 모든 문자.
일반적으로 인쇄할 수 없는 문자는 표시 및 인쇄 용도에는 필요하지 않습니다.
그러나 일부 인쇄할 수 없는 문자는 다음과 같은 제어 기능으로 사용됩니다.

표 12-2

문자	기능
HT	다음 가로 탭에서 인쇄 또는 입력을 시작합니다.
LF	다음 줄에서 인쇄 또는 입력을 시작합니다.
FF	다음 페이지의 맨 위에서 인쇄 또는 입력을 시작합니다.
CR	같은 줄의 첫 공백에서 인쇄 또는 입력을 시작합니다.
ESC	터미널 이스케이프 시퀀스를 시작합니다.
SP	공백 하나를 삽입합니다.

문자열 정의

인용 부호(“)로 묶어서 문자열을 정의할 수 있습니다. 따라서 대소문자와 공백은 기호 할당에 이루어진 뒤에도 보존됩니다. 다음을 참조하십시오.

- 문자열 안에 인용 부호(“)를 포함시키려면 두 개의 연속된 인용 부호를 입력합니다.
- 문자열을 두 줄 이상 계속하려면 더하기 기호(문자열 연결) 및 하이픈(연결)을 사용합니다.

인용 문자열 안에서는 하이픈 연결 문자를 사용할 수 없습니다.

다음 보기에서 문자열 “YES”는 인용된 것이므로 인용 부호 내에 정의해야 합니다.

```
$ PROMPT = "Type "YES" or "NO""
$ SHOW SYMBOL PROMPT
PROMPT = "Type "YES" or "NO"
```

다음 보기에서는 문자열이 두 줄에 계속됩니다.

```
$ HEAD = "MONTHLY REPORT FOR" + -
_ $ " DECEMBER 1999"
$ SHOW SYMBOL HEAD
HEAD = "MONTHLY REPORT FOR DECEMBER 1999"
```

문자열 표현식

문자열 표현식은 문자열과 문자열로 평가되는 렉시칼 함수 또는 문자열 값을 가지는 기호를 포함할 수 있습니다. 표현식에 문자열을 사용하려면 문자열을 인용 부호(“)로 묶어야 합니다. 인용 부호를 사용하지 않으면 DCL에서는 문자열을 기호로 처리합니다.

문자열 표현식은 문자열 피연산자라는 다음과 같은 값을 결합합니다.

- 인용 부호에 표시해야 하는 문자열
- 문자열을 나타내는 기호
- 문자열로 평가되는 렉시칼 함수

문자열과 숫자 간의 연산이나 비교를 수행할 경우 DCL은 문자열을 숫자로 변환합니다.

다음의 하위 절에서 설명하는 바와 같이 문자열 피연산자는 추가(문자열 연결), 제거(문자열 빼기), 비교 또는 다른 문자열로 대체할 수 있습니다.

다음 보기에서 문자열 “CAT”은 인용 부호 안에 표시해야 합니다.

```
$ TEMP = "CAT"
```

다음 보기에서 기호 **TEMP**는 문자열 “CAT”을 나타냅니다. 기호 **TOPIC**은 문자열 “THE”와 기호 **TEMP**가 나타내는 문자열(“CAT”)의 연결이므로 “THE CAT”입니다.

```
$ TOPIC = "THE" + TEMP
```

다음 보기에서 기호 **COUNT**는 렉시칼 함수 **F\$STRING(65)**을 나타냅니다.

```
$ COUNT = F$STRING(65)
```

문자열 연산

다음과 같은 문자열 연산을 지정할 수 있습니다.

- 연결 — 더하기 기호는 두 문자열을 연결합니다.
- 빼기 — 빼기 기호는 첫 번째 문자열에서 두 번째 문자열을 제거합니다.

첫 번째 문자열에서 두 번째 문자열이 두 번 이상 나타나면 처음 나타나는 문자열만 제거됩니다.

다음 보기에서는 더하기 기호(+)를 사용하여 두 문자열을 연결합니다.

```
$ COLOR = "light brown"
$ WEIGHT = "30 lbs."
$ DOG2 = "No tag, " + COLOR + ", " + WEIGHT
$ SHOW SYMBOL DOG2
DOG2 = "No tag, light brown, 30 lbs."
```

다음 보기에서는 빼기 기호(-)를 사용하여 문자열을 제거합니다.

```
$ SHOW SYMBOL DOG2
DOG2 = "No tag, light brown, 30 lbs."
$ DOG2 = DOG2 - ", 30 lbs."
$ SHOW SYMBOL DOG2
DOG2 = "No tag, light brown"
```

문자열 비교

두 문자열을 비교할 경우 문자 대문자로 문자열을 비교합니다. 길이가 다른 문자열은 동일하지 않습니다. 예를 들어, “dogs”는 “dog”보다 큽니다.

비교 기준은 문자의 **ASCII** 값입니다. 이런 기준에 따라 숫자 **0-9**는 대문자 **A-Z**보다 작고 대문자 **A-Z**는 소문자 **a-z**보다 작습니다. 문자열 비교는 다음 조건 중 하나가 참이면 종료됩니다.

- 모든 문자열을 비교하며 이 경우 문자열은 동일합니다.
- 첫 번째 불일치가 발생합니다.

표 12-3에서는 여러 문자열 비교 유형을 나열합니다.

표 12-3

문자열 비교

비교	연산자	설명
같음	.EQS.	한 문자열과 다른 문자열이 같은지 비교합니다.
크거나 같음	.GES.	한 문자열과 다른 문자열을 비교하여 지정한 첫 번째 문자열에 크거나 같은 값이 있는지 확인합니다.
큼	.GTS.	한 문자열과 다른 문자열을 비교하여 지정한 첫 번째 문자열에 큰 값이 있는지 확인합니다.
작거나 같음	.LES.	한 문자열과 다른 문자열을 비교하여 지정한 첫 번째 문자열에 작거나 같은 값이 있는지 확인합니다.
작음	.LTS.	한 문자열과 다른 문자열을 비교하여 지정한 첫 번째 문자열에 작은 값이 있는지 확인합니다.
같지 않음	.NES.	한 문자열과 다른 문자열이 같지 않은지 비교합니다.

다음의 모든 보기에서 기호 **LAST_NAME**의 값은 “WHITEFIELD”입니다

- 다음 보기에서 기호 **TEST_NAME**은 0(거짓)으로 평가됩니다.
기호 **LAST_NAME** 값은 문자열 'HILL'과 같지 않습니다.

```
$ TEST_NAME = LAST_NAME .EQS. "Hill"
$ SHOW SYMBOL TEST_NAME
TEST_NAME = 0    ...
```
- 다음 보기에서는 기호 **TEST_NAME**이 1(참)로 평가됩니다.
기호 **LAST_NAME** 값은 문자열 “HILL”보다 크거나 같습니다.

```
$ TEST_NAME = LAST_NAME .GES. "HILL"
$ SHOW SYMBOL TEST_NAME
TEST_NAME = 1    ...
```
- 다음 보기에서는 기호 **TEST_NAME**이 1(참)로 평가됩니다.
기호 **LAST_NAME** 값은 문자열 “HILL”보다 큼니다.

```
$ TEST_NAME = LAST_NAME .GTS. "HILL"
$ SHOW SYMBOL TEST_NAME
TEST_NAME = 1    ...
```

- 다음 보기에서는 기호 **TEST_NAME**이 0(거짓)으로 평가됩니다.
기호 **LAST_NAME**은 문자열 “HILL”보다 작거나 같지 않습니다.

```
$ TEST_NAME = LAST_NAME .LES. "HILL"  
$ SHOW SYMBOL TEST_NAME  
TEST_NAME = 0    ...
```

- 다음 보기에서는 기호 **TEST_NAME**이 0(거짓)으로 평가됩니다.
기호 **LAST_NAME** 값은 문자열 “HILL”보다 작지 않습니다.

```
$ TEST_NAME = LAST_NAME .LTS. "HILL"  
$ SHOW SYMBOL TEST_NAME  
TEST_NAME = 0    ...
```

- 다음 보기에서는 기호 **TEST_NAME**이 1(참)로 평가됩니다.
기호 **LAST_NAME** 값은 문자열 “HILL”과 같지 않습니다.

```
$ TEST_NAME = LAST_NAME .NES. "HILL"  
$ SHOW SYMBOL TEST_NAME  
TEST_NAME = 1    ...
```

부분 문자열 대체

대체 문자열의 위치와 크기를 지정하여 문자열의 일부를 다른 문자열로 대체할 수 있습니다.
로컬 기호의 경우 형식은 다음과 같습니다.

symbol-name[offset,size] := replacement-string

글로벌 기호의 형식은 다음과 같습니다.

symbol-name[offset,size] ::= replacement-string

요소는 다음과 같습니다.

표 12-4

offset	원래 문자열의 첫 문자를 기준으로 대체 문자열의 위치를 나타내는 정수. offset 0 은 기호의 첫 번째 문자, offset 1 은 두 번째 문자를 의미합니다.
size	대체 문자열의 길이를 나타내는 정수.

부분 문자열을 교체하려면 다음 규칙을 따릅니다.

- 대괄호를 꼭 사용해야 합니다. 여는 대괄호와 기호 이름 사이에 공백이 있어서는 안 됩니다.
- **size** 및 **offset** 값의 범위는 **0-768**입니다.
- 대체 문자열은 문자열이어야 합니다.
- 지정하는 기호 이름이 처음에 정의되지 않은 이름일 수 있습니다. 할당문에서 기호 이름을 만들고 필요한 경우 기호 값의 앞이나 뒤에 공백을 넣습니다.
- **offset**과 **size**를 지정하여 빈 줄을 나타내는 기호를 만들 수 있습니다.

열의 레코드를 정렬하면 목록을 읽고 정렬하기가 쉬워집니다. 이 형식을 사용하여 데이터 정렬 방법을 지정할 수 있습니다.

다음 보기에서 첫 번째 할당문은 기호 **A**에 값 **PACKRAT**을 부여합니다. 두 번째 할당문에서는 **MUSK**로 **A** 값의 첫 네 문자를 교체합니다. 따라서 **A** 값은 **MUSKRAT**가 됩니다.

```
$ A := PACKRAT
$ A[0,4] := MUSK
$ SHOW SYMBOL A
A = "MUSKRAT"
```

다음 보기에서 기호 **B**는 이전 값이 아니므로 앞에 4개의 공백이 오고 그 뒤에 **RAT**가 오는 값이 됩니다.

```
$ B[4,3] := RAT
```

다음 보기에서는 80개의 빈 공백이 기호 **LINE**에 할당됩니다.

```
$ LINE[0,80] := " "
```

다음 보기에서 첫 번째 문은 **DATA**의 첫 15개 열을 값 **NAME**으로 채웁니다. 두 번째 문은 18열을 **GRADE**의 값으로 채웁니다. 16열과 17열은 공백을 포함합니다.

```
$ DATA[0,15] := "NAME"
$ DATA[17,1] := "GRADE"
```

숫자 값 및 식 사용

숫자는 다음 값이 될 수 있습니다.

- 10진수 — ASCII 문자 0-9
- 16진수 — ASCII 문자 0-9 및 A-F
- 8진수 — ASCII 문자 0-7

기호에 할당하는 수의 범위는 -2147483648 ~ 2147483647(십진수)이어야 합니다. 이 범위를 벗어나는 숫자를 지정하면 오류가 발생하거나 계산되더라도 잘못된 값을 냅니다.

숫자 지정

DCL 명령 수준이나 명령 프로시저에서 다음과 같이 숫자를 지정합니다.

- 양수
적절한 자릿수를 입력하여 양수를 지정합니다.
- 음수
음수 앞에는 빼기 기호(-)를 붙입니다.
- 기수
숫자 앞에(빼기 기호가 아님) 16진수인 경우 %X, 8진수인 경우 %0를 붙여 10진수가 아닌 기수로 숫자를 지정합니다.
- 분수
숫자에는 소숫점이 포함될 수 없습니다. 계산에서 분수는 버립니다.
예를 들어, 8/3은 2와 같습니다.

다음 보기에서는 숫자 13을 기호 DOG_COUNT에 할당합니다.

```
$ DOG_COUNT = 13
$ SHOW SYMBOL DOG_COUNT
DOG_COUNT = 13   Hex = 0000000D   Octal = 00000000015
```

다음 보기에서는 음수(-15237)를 빼기 기호(-)로 표시합니다.

```
$ BALANCE = -15237
$ SHOW SYMBOL BALANCE
BALANCE = -15237   Hex = FFFFC47B   Octal = 37777742173
```

다음 보기에서는 16진수 D를 접두사 %X로 표시합니다.

```
$ DOG_COUNT = %XD
$ SHOW SYMBOL
DOG_COUNT DOG_COUNT = 13    Hex = 0000000D  Octal = 00000000015
$ BALANCE = -%X3B85
$ SHOW SYMBOL BALANCE
BALANCE = -15237    Hex = FFFFC47B  Octal = 37777742173
```

숫자의 내부 저장

숫자는 내부적으로 긴 단어라는 부호형 4바이트 정수로 저장됩니다. 양수는 0~2147483647의 값을 가지고 음수는 4294967296에서 숫자의 절대 값을 뺀 값을 가집니다. 예를 들어, 숫자 -15237은 4294952059로 저장됩니다. 음수는 ASCII나 십진 표시인 경우 빼기 기호 형식으로 다시 변환되지만 16진수와 8진수 표시인 경우는 변환되지 않습니다. 예를 들어, 숫자 -15237은 16진수 -00003B85가 아닌 16진수 FFFFC47B(10진수 4294952059)로 표시됩니다.

숫자는 ASCII 규약을 사용하여 일련의 숫자로 텍스트 파일에 저장됩니다. 예를 들어, 숫자 1은 49의 저장 값을 가집니다.

숫자 식에서 포함되는 값은 리터럴 숫자(예: 3)이거나 숫자 값을 갖는 기호이어야 합니다. 또한, 숫자를 나타내는 문자열(예: “23” 또는 “-51”)을 사용할 수 있습니다. 숫자와 문자열 간의 연산 또는 비교를 수행할 경우 DCL은 문자열을 숫자로 변환합니다.

숫자 식은 다음 값(피연산자라고 함)을 합칩니다.

- 정수. 예를 들면 다음과 같습니다.

```
$ COUNT = 1
```

- 정수로 평가되는 렉시칼 함수. 예를 들면 다음과 같습니다.

```
$ B = F$INTEGER("-9" + 23)
```

- 정수 값을 갖는 기호. 예를 들면 다음과 같습니다.

```
$ A = B - 6
```

앞의 보기에서 기호 B는 F\$INTEGER 함수에서 반환하는 정수 값(-923)을 나타냅니다.

이 정수 피연산자를 다음 절에서 설명하는 대로 산술, 논리 및 비교 연산자로 연결할 수 있습니다.

산술 연산 수행

다음과 같은 산술 연산을 지정할 수 있습니다.

- 곱하기
별표(*)는 두 숫자를 곱합니다.
- 나누기
슬래시(/)는 지정한 첫 번째 숫자를 지정한 두 번째 숫자로 나눕니다. 숫자가 정확히 나뉘지 않으면 나머지는 버립니다. 반올림은 하지 않습니다.
- 더하기
더하기 기호(+)는 두 숫자를 더합니다.
- 빼기
빼기 기호(-)는 지정된 첫 번째 숫자에서 지정된 두 번째 숫자를 뺍니다.
- 단항 가산 및 감산
더하기와 빼기 기호는 첫 숫자의 부호를 변경합니다.

보기

- 다음 보기에서는 기호를 할당할 때 곱하기 사용을 보여 줍니다.

```
$ BALANCE = 142 * 14
$ SHOW SYMBOL BALANCE
BALANCE = 1988    Hex = 000007C4    Octal = 00000003704
```
- 다음 보기에서는 기호를 할당할 때 나누기 사용을 보여 줍니다.

```
$ BALANCE = BALANCE / 14
$ SHOW SYMBOL BALANCE
BALANCE = 142    Hex = 0000008E    Octal = 00000000216
```
- 다음 보기에서는 기호를 할당할 때 더하기 사용을 보여 줍니다.

```
$ BALANCE = BALANCE + 37
$ SHOW SYMBOL BALANCE
BALANCE = 179    Hex = 000000B3    Octal = 00000000263
```
- 다음 보기에서는 기호를 할당할 때 빼기 사용을 보여 줍니다.

```
$ BALANCE = BALANCE - 15416
$ SHOW SYMBOL BALANCE
BALANCE = -15237    Hex = FFFFC47B    Octal = 00000142173
```

- 다음 보기에서는 단항 감산 기호를 사용하여 숫자 -142의 부호를 변경하는 것을 보여줍니다.

```
$ BALANCE = -(- a142)
$ SHOW SYMBOL BALANCE
BALANCE = 142    Hex = 0000008E    Octal = 00000000216
```

숫자 비교

표 12-5에서는 숫자 비교의 여러 유형을 나열합니다.

표 12-5

숫자 비교

비교	연산자	설명
같음	.EQ.	두 숫자가 동일한지 비교합니다.
크거나 같음	.GE.	한 숫자와 다른 숫자를 비교하여 첫 번째 숫자에 크거나 같은 값이 있는지 확인합니다.
큼	.GT.	한 숫자와 다른 숫자를 비교하여 첫 번째 숫자에 큰 값이 있는지 확인합니다.
작거나 같음	.LE.	한 숫자와 다른 숫자를 비교하여 첫 번째 숫자에 작거나 같은 값이 있는지 확인합니다.
작음	.LT.	한 숫자와 다른 숫자를 비교하여 첫 번째 숫자에 작은 값이 있는지 확인합니다.
같지 않음	.NE.	두 숫자가 같지 않은지 비교합니다.

다음 보기에서는 기호 BALANCE의 값이 -15237이라고 가정합니다.

- 다음 보기에서 TEST_BALANCE는 1(참)로 평가됩니다. BALANCE는 -15237과 같습니다.

```
$ TEST_BALANCE = BALANCE .EQ. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 1    ...
```

- 다음 보기에서 **TEST_BALANCE**는 1(참)로 평가됩니다.
BALANCE는 -15237보다 크거나 같습니다.

```
$ TEST_BALANCE = BALANCE .GE. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 1    ...
```
- 다음 보기에서 **TEST_BALANCE**는 0(거짓)으로 평가됩니다.
BALANCE는 -15237보다 크지 않습니다.

```
$ TEST_BALANCE = BALANCE .GT. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 0    ...
```
- 다음 보기에서 **TEST_BALANCE**는 1(참)로 평가됩니다.
BALANCE는 -15237보다 작거나 같습니다.

```
$ TEST_BALANCE = BALANCE .LE. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 1    ...
```
- 다음 보기에서 **TEST_BALANCE**는 0(거짓)으로 평가됩니다.
BALANCE는 -15237보다 작지 않습니다.

```
$ TEST_BALANCE = BALANCE .LT. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 0    ...
```
- 다음 보기에서 **TEST_BALANCE**는 0(거짓)으로 평가됩니다.
BALANCE는 -15237과 같습니다.

```
$ TEST_BALANCE = BALANCE .NE. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 0    ...
```

숫자 오버레이 수행

특수한 형식의 할당문을 사용하여 현재 기호 값의 이진(비트 수준) 오버레이를 수행할 수 있습니다. 로컬 기호인 경우 형식은 다음과 같습니다.

symbol-name[**bit-position**,**size**] = **replacement-expression**

글로벌 기호인 경우 형식은 다음과 같습니다.

symbol-name[**bit-position**,**size**] == **replacement-expression**

요소는 다음과 같습니다.

표 12-6

bit-position	비트 0을 기준으로 오버레이가 일어날 위치를 나타내는 정수.
size	오버레이될 비트 수를 나타내는 정수.

숫자 오버레이를 사용하려면 다음 규칙을 따릅니다.

- 대괄호([])는 필수적으로 사용해야 합니다. 여는 대괄호와 기호 이름 사이에 공백이 있어서는 안 됩니다.
- 리터럴 값을 10진수로 간주합니다.
- 크기의 최대 길이는 32비트입니다.
- 대체 식은 숫자 식이어야 합니다.
- 기호 이름이 정의되어 있지 않거나 문자열로 정의되어 있는 경우 오버레이의 결과는 문자열입니다. 그렇지 않으면 결과는 정수입니다.

다음 보기에서는 기호 **BELL**을 값 7로 정의합니다. **BELL**의 최하위 바이트는 이진 값 00000111을 가집니다. 오프셋 5의 0을 1로 변경하여(0에서 시작하여 오른쪽에서 왼쪽으로 비트 수 계산) 이진 값 00100111(십진 값 39)을 만듭니다.

```
$ BELL = 7
$ BELL[5,1] = 1
$ SHOW SYMBOL BELL
  BELL = 39   Hex = 00000027   Octal = 00000000047
```

논리 값 및 식 사용

다음 절에서는 논리 값 및 식을 사용하는 방법을 설명합니다.

논리 연산

일부 연산에서는 숫자와 문자열을 다음 값의 논리 데이터로 변환합니다.

- 참(True)

숫자는 홀수(즉, 최하위 비트가 1)인 경우 참의 논리 값을 가집니다. 문자열은 첫 문자가 대문자나 소문자 **T** 또는 **Y**인 경우 참의 논리 값을 가집니다.

- 거짓(False)

숫자는 짝수(즉, 최하위 비트가 0)인 경우 거짓의 논리 값을 가집니다. 문자열은 첫 문자가 대문자나 소문자 **T** 또는 **Y**가 아닌 경우 거짓의 논리 값을 가집니다.

다음 보기에서는 **DOG_COUNT**에 값 13을 할당합니다. **IF STATUS**는 **STATUS**의 논리 값이 참인 경우를 의미합니다.

```
$ STATUS = 1
$ IF STATUS THEN DOG_COUNT = 13

$ STATUS = "TRUE"
$ IF STATUS THEN DOG_COUNT = 13
```

논리식

논리 연산자는 적용될 숫자의 모든 비트에 영향을 미칩니다. **논리식**의 값이 정수이면 식의 결과도 정수입니다. 논리식에 문자열 값을 지정할 경우 식이 평가되기 전에 문자열이 정수로 변환됩니다.

일반적으로 논리식을 사용하여 논리 값의 최하위 비트를 평가합니다. 즉, 값이 참인지 거짓인지 결정합니다. 다음과 같은 논리 연산을 지정할 수 있습니다.

- .NOT**

연산자 **.NOT**은 논리 값의 비트 구성을 반대로 만듭니다. 참 값은 거짓이 되고 거짓 값은 참이 됩니다.

- .AND.
연산자 .AND.는 두 논리 값을 다음과 같이 결합합니다.

표 12-7

비트 수준	엔터티 수준
1 .AND. 1 = 1	true .AND. true = true
1 .AND. 0 = 0	true .AND. false = false
0 .AND. 1 = 0	false .AND. true = false
0 .AND. 0 = 0	false .AND. false = false

- .OR.
연산자 .OR.은 두 논리 값을 다음과 같이 결합합니다.

표 12-8

비트 수준	엔터티 수준
1 .OR. 1 = 1	true .OR. true = true
1 .OR. 0 = 1	true .OR. false = true
0 .OR. 1 = 1	false .OR. true = true
0 .OR. 0 = 0	false .OR. false = false

다음 보기에서는 참 값을 거짓으로 되돌립니다. 이 표현식은 -2로 평가되는데 이 값은 짝수
이므로 거짓입니다.

```
$ SHOW SYMBOL STATUS
STATUS = 1   Hex = 00000001   Octal = 00000000001
$ STATUS = .NOT. STATUS
$ SHOW SYMBOL STATUS
STATUS = -2   Hex = FFFFFFFE   Octal = 37777777776
```

다음 보기에서는 참 값과 거짓 값을 결합해 거짓 값을 만듭니다.

```
$ STAT1 = "TRUE"
$ STAT2 = "FALSE"
$ STATUS = STAT1 .AND. STAT2
$ SHOW SYMBOL STATUS
STATUS = 0   Hex = 00000000   Octal = 00000000000
```

다음 보기에서는 참 값과 거짓 값을 결합해 참 값을 만듭니다.

```
$ STAT1 = "TRUE"  
$ STAT2 = "FALSE"  
$ STATUS = STAT1 .OR. STAT2  
$ SHOW SYMBOL STATUS  
STATUS = 1    Hex = 00000001    Octal = 00000000001
```

논리 연산 결과

다음 표에서는 비트별 및 숫자별 논리 연산 결과를 보여 줍니다. 논리 연산에서 대문자나 소문자 T 또는 Y로 시작하는 문자열은 숫자 1로 처리되고 다른 문자로 시작하는 문자열은 숫자 0으로 처리됩니다. 논리 연산에서 홀수는 참이고 짝수와 0은 거짓입니다.

표 12-9

조건:		결과:		
비트 A	비트 B	.NOT. A	A .AND. B	A .OR. B
1	1	0	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	0

표 12-10

조건:		결과:		
숫자 A	숫자 B	.NOT. A	A .AND. B	A .OR. B
홀수	홀수	짝수	홀수	홀수
홀수	짝수	짝수	짝수	홀수
짝수	홀수	홀수	짝수	홀수
짝수	짝수	홀수	짝수	짝수

렉시칼 함수에서 반환한 값 사용

일반적으로 명령 프로시저에서 사용되는 렉시칼 함수는 시스템에서 시스템 프로세스, 일괄 처리 및 인쇄 대기열, 사용자 프로세스 등에 대한 정보를 포함한 정보를 검색합니다. 렉시칼 함수를 사용하여 문자열을 조작하고 논리 이름을 변환할 수도 있습니다. 렉시칼 함수를 기호에 할당할 때 기호는 렉시칼 함수에서 반환한 정보(예: 숫자 또는 문자열)를 나타냅니다. 그런 다음 DCL 수준에서 DCL 명령 SHOW SYMBOL을 사용하여 정보를 표시할 수 있습니다. 명령 프로시저에서 기호에 저장된 정보를 나중에 사용할 수 있습니다. 렉시칼 함수에 대한 자세한 내용은 15장을 참조하십시오.

렉시칼 함수를 사용하려면 렉시칼 함수의 이름(항상 F\$로 시작)과 그 인수 목록을 지정해야 합니다. 다음 형식을 사용합니다.

F\$function-name(args[,...])

인수 목록은 함수 이름 다음 간격 사이마다 여러 공백과 탭을 넣습니다.

렉시칼 함수를 사용할 때 다음 규칙을 따릅니다.

- 인수 목록은 괄호로 묶습니다.
- 목록에서 인수를 정확한 순서로 지정하고 쉼표로 구분합니다. 선택적인 인수는 생략하더라도 쉼표는 생략하지 마십시오.
- 인수가 필요하지 않은 경우 비어 있는 괄호 집합을 입력합니다.
- 식 작성 규칙을 따릅니다. 문자열은 인용 부호로 묶고 정수, 기호 및 렉시칼 함수는 인용 부호로 묶지 않습니다.

문자열, 정수 및 기호를 사용하는 것과 마찬가지로 렉시칼 함수를 사용합니다. 표현식에 렉시칼 함수를 사용하면 DCL은 함수를 자동으로 평가하고 반환 값으로 함수를 대체합니다.

다음 보기에서 F\$LENGTH 함수는 인수인 BUMBLEBEE로 지정된 값의 길이를 반환합니다. DCL은 자동으로 반환 값(9)을 결정하고 이 값을 사용하여 식을 평가합니다. 따라서, 식 (9 + 1)의 결과는 10이 되고 이 값은 기호 SUM에 할당됩니다.

```
$ SUM = F$LENGTH("BUMBLEBEE") + 1
$ SHOW SYMBOL SUM
SUM = 10    Hex = 0000000A    Octal = 00000000012
```

각 렉시칼 함수는 정수 또는 문자열로 정보를 반환합니다. 또한, 렉시칼 함수의 인수를 정수 또는 문자열 식으로 지정해야 합니다.

예를 들어, **F\$LENGTH** 함수는 문자열 표현식인 인수를 필요로 하며 정수인 값을 반환합니다. 앞의 보기에서 인수 **"BUMBLEBEE"**은 문자열 표현식이고 반환 값 **(9)**는 정수입니다.

기호를 사용할 수 있는 위치에는 어디에나 렉시칼 함수를 사용할 수 있습니다. 기호를 어포스트로피로 묶어서 기호 대체를 강제로 수행해야 하는 위치(**489**페이지의 “기호 대체 이해” 참조)에서는 렉시칼 함수를 어포스트로피 내에 넣어 렉시칼 함수 평가를 강제로 수행해야 합니다. 렉시칼 함수도 다른 렉시칼 함수의 인수로 사용할 수 있습니다.

다음 보기들에서는 **F\$LENGTH** 함수의 인수를 지정할 수 있는 여러 방법을 보여 줍니다. 각 보기에서 인수는 문자열 표현식입니다.

- 다음 보기에서는 인수로 사용되는 기호를 보여 줍니다.

```
$ BUG = "BUMBLEBEE"
$ LEN = F$LENGTH(BUG)
$ SHOW SYMBOL LEN
LEN = 9   Hex = 00000009   Octal = 00000000011
```

기호 **BUG**를 인수로 사용할 때 인용 부호로 묶지 마십시오. 렉시칼 함수는 자동으로 **BUG**를 값 **"BUMBLEBEE"**로 대체하고 길이를 결정하여 값 **9**를 반환합니다.

- 다음 보기에서는 기호와 문자열을 모두 포함하는 인수를 보여 줍니다.

```
$ BUG = "BUMBLEBEE"
$ LEN = F$LENGTH(BUG)
$ SHOW SYMBOL LEN
LEN = 9   Hex = 00000009   Octal = 00000000011
$ LEN = F$LENGTH(BUG + "S")
$ SHOW SYMBOL LEN
LEN = 10  Hex = 0000000A   Octal = 00000000012
```

기호 **BUG**는 인용 부호로 묶지 않았지만 문자열 **"S"**는 그렇게 했습니다. **F\$LENGTH** 함수에서 길이를 결정할 수 있으려면 인수가 평가되어야 합니다.

기호 **BUG**(**"BUMBLEBEE"**)가 나타내는 값이 문자열 **"S"**와 연결되어 **"BUMBLEBEES"**라는 결과가 됩니다. **F\$LENGTH** 함수에서는 문자열 **"BUMBLEBEES"**의 길이를 결정하고 값 **10**을 반환합니다.

- 다음 보기에서는 **F\$LENGTH** 함수의 인수로 다른 렉시칼 함수를 사용합니다. **F\$DIRECTORY** 함수는 대괄호와 함께 현재의 기본 디렉토리 이름을 반환합니다. 다음 보기에서 현재 기본 디렉토리는 **[SALMON]**입니다.

```
$ LEN = F$LENGTH(F$DIRECTORY() )
$ SHOW SYMBOL LEN
LEN = 8   Hex = 00000008   Octal = 00000000010
```

F\$DIRECTORY 함수가 인수로 사용될 경우 이 함수를 인용 부호로 묶으면 안 됩니다. 함수는 자동으로 평가됩니다. F\$DIRECTORY 함수의 결과가 반환되어야 F\$LENGTH 함수에서 길이를 결정할 수 있습니다. 그런 다음 F\$LENGTH 함수에서는 대괄호를 포함하여 기본 디렉토리의 길이를 결정합니다.

작업 순서

표현식은 여러 연산과 비교를 포함할 수 있습니다. 다음 표에서는 식에 둘 이상의 연산자가 있을 경우 계산되는 순서에 따라 연산자를 나열합니다. 우선 순위가 높은 것에서 낮은 것으로 나열되어 있습니다. 즉, 표의 맨 위에 있는 연산자가 맨 아래에 있는 연산자보다 먼저 실행됩니다.

표 12-11

우선 순위	연산
7	단항 가산(+) 및 감산(-)
6	곱하기 (*) 및 나누기 (/)
5	더하기(결합) 및 제거(빼기)
4	모든 숫자와 문자 비교
3	논리 .NOT. 연산
2	논리 .AND. 연산
1	논리 .OR. 연산

식에 같은 우선 순위의 연산자가 포함되어 있는 경우 연산은 왼쪽에서 오른쪽으로 수행됩니다. 처음에 수행할 연산을 괄호로 묶어 일반적인 우선 순위(연산과 비교가 평가되는 순서)를 무시할 수 있습니다. 괄호를 중첩시킬 수도 있습니다.

다음 보기에서는 괄호를 사용하여 곱하기 전에 더하기를 수행합니다. 괄호가 없으면 곱하기가 먼저 수행되어 결과는 26이 됩니다.

```
$ RESULT = 4 * (6 + 2)
$ SHOW SYMBOL RESULT
RESULT = 32    Hex = 00000020    Octal = 00000000040
```

데이터 유형 평가

기호에 대한 DCL 평가의 결과는 문자열이나 정수 값 중 하나입니다. 기호가 문자인지 정수 인지는 현재 할당된 값의 데이터 유형에 따릅니다. 데이터 유형은 영구적이지 않습니다. 값의 데이터 유형이 변경되면 기호의 데이터 유형도 변경됩니다.

표현식에는 사용된 값의 유형과 연산자에 따라 정수나 문자열 값이 포함됩니다.

다음 보기에서는 로컬 기호 NUM이 처음에는 문자 값이 할당되고 이후 정수 식을 할당할 때 정수 값으로 변환됩니다.

```
$ NUM = "ABC"  
$ NUM = 2 + 5
```

다음 표에서는 DCL에서 표현식을 평가하는 방법을 요약합니다. 첫 열에서는 표현식에서 사용할 수 있는 여러 값과 연산자를 나열하고 두 번째 열에서는 각 경우에 전체 표현식과 일치하는 값이 무엇인지 알려 줍니다. 표에서 임의의 값은 문자열이나 정수를 나타냅니다.

표 12-12

표현식	결과 값 유형
정수 값	정수
문자열 값	문자열
정수 렉시칼 함수	정수
문자열 렉시칼 함수	문자열
정수 기호	정수
문자열 기호	문자열
+, -, .NOT. 임의의 값	정수
임의의 값 .AND. 또는 .OR. 임의의 값	정수
문자열 + 또는 - 문자열	문자열
정수 + 또는 - 임의의 값	정수
임의의 값 + 또는 - 정수	정수
임의의 값 * 또는 / 임의의 값	정수
임의의 값 (문자열 비교) 임의의 값	정수
임의의 값 (숫자 비교) 임의의 값	정수

표현식에서 값 유형 변환

DCL에서 식을 계산할 수 있으려면 표현식의 피연산자가 같은 값 데이터 유형이어야 합니다. 값은 문자열이나 정수 데이터 유형이 될 수 있습니다. 문자열 데이터에는 문자열, 문자열 값을 갖는 기호 및 문자열 값을 반환하는 렉시칼 함수가 포함됩니다. 정수 데이터는 정수, 정수 값을 갖는 기호 및 정수 값을 반환하는 렉시칼 함수를 포함합니다. 표현식에 숫자와 문자열 피연산자가 모두 있으면 DCL은 모든 문자열을 정수로 또는 모든 정수를 문자열로 변환합니다.

일반적으로 문자열과 정수 값을 모두 사용할 경우 문자열 값이 정수로 변환되지만 DCL에서 문자열 비교를 수행할 때에만 예외입니다. 이런 비교에서는 정수가 문자열로 변환됩니다.

또한, 다음 렉시칼 함수를 사용하여 표현식의 값을 결정 또는 변경할 수 있습니다.

- **F\$TYPE** — 기호의 현재 값 유형 결정
- **F\$INTEGER** — 문자열 표현식을 정수 값으로 변환
- **F\$STRING** — 정수 표현식을 문자열 값으로 변환

문자열을 정수로 변환

다음과 같이 문자열을 정수로 변환합니다.

- 숫자를 포함하는 문자열은 해당 정수 값으로 변환됩니다. 예를 들어, 문자열 “45”는 정수 45로 변환됩니다.
- 문자열이 T, t, Y 또는 y로 시작하면 정수 1로 변환됩니다.
- 문자열이 다른 문자로 시작하면 정수 0으로 변환됩니다.

다음 표에서는 문자열이 정수 값으로 변환되는 예를 보여 줍니다.

표 12-13

문자열	결과 정수
“123”	123
“12XY”	0 (거짓)
“Test”	1 (참)
“hello”	0 (거짓)

정수에서 문자열로 변환

정수가 문자열로 변환될 때 결과 문자열에는 정수 값에 해당하는 숫자가 포함됩니다. 다음 표에서는 정수가 문자열로 변환되는 방법을 보여 줍니다.

표 12-14

정수		결과 문자열
123		“123”
1		“1”
0		“0”

기호 표 이해

기호는 운영 체제에서 관리하는 로컬 또는 글로벌 기호 표에 저장됩니다.

로컬 기호 표

DCL에서는 사용자가 명령 프로시저를 실행하거나 CALL 명령을 사용하거나 일괄 처리 작업을 제출할 때 만드는 모든 명령 수준과 사용자의 기본 프로세스의 로컬 기호 표를 유지 관리합니다. 사용자가 로컬 기호를 만들면 DCL은 해당 기호를 현재 명령 수준의 로컬 기호 표에 저장합니다. 명령 수준이 활성화되면 DCL은 해당 명령 수준의 로컬 기호 표를 유지합니다. 명령 수준이 더 이상 활성화 아니면 로컬 기호 표와 여기에 포함된 모든 기호가 삭제됩니다. 프로세스, 명령 프로시저 및 일괄 처리 작업에 대한 자세한 내용은 16장을 참조하십시오.

로컬 기호 표에는 사용자가 만든 로컬 기호와 더불어 DCL에서 관리하는 8개의 기호가 들어 있습니다. 이 기호(P1, P2 등)는 명령 프로시저에 매개 변수를 전달하기 위해 사용됩니다. 명령 프로시저에 전달되는 매개 변수는 문자열로 간주됩니다. 또는 P1~P8이 null 문자열(“ ”)로 정의됩니다. 이 기호는 로컬 기호 표에 저장됩니다.

글로벌 기호 표

DCL에서는 프로세스 지속 기간 동안 하나의 글로벌 기호 표만을 유지하며 모든 글로벌 기호를 이 표에 저장합니다. 글로벌 기호 표는 사용자가 만든 글로벌 기호와 함께 예약된 글로벌 기호를 포함합니다. 이 글로벌 기호는 시스템 명령 및 유틸리티와 프로그램 및 명령 프로시저에 대한 상태 정보를 제공합니다.

\$STATUS 예약된 글로벌 기호

\$STATUS는 가장 최근에 실행된 명령에서 반환하는 조건 코드입니다.

기호 \$STATUS는 OpenVMS 운영 체제 메시지 코드를 확인합니다. 어플리케이션에서는 EXIT 명령에 매개 변수 값을 포함시켜 글로벌 기호 \$STATUS 값을 설정할 수 있습니다. 시스템은 \$STATUS 값을 사용하여 표시할 메시지(있는 경우)와 다음 상위 명령 수준에서 실행을 계속할지 여부를 결정합니다. \$STATUS의 가장 낮은 세 개 비트의 값은 글로벌 기호 \$SEVERITY에 저장됩니다.

\$SEVERITY 예약된 글로벌 기호

\$SEVERITY는 가장 최근에 실행된 명령에서 반환하는 조건 코드의 심각도 수준입니다. \$STATUS의 최하위 비트와 일치하는 기호 \$SEVERITY는 다음 값을 가질 수 있습니다.

표 12-15

0	경고
1	성공
2	오류
3	정보
4	심각한(치명적) 오류

\$RESTART 예약된 글로벌 기호

\$RESTART의 값은 일괄 처리 작업이 시스템 실패로 중단된 후 다시 시작된 경우 TRUE이고 그렇지 않은 경우 FALSE입니다.

기호 표 검색 순서

명령 인터프리터에서 기호 값을 결정할 때 다음과 같은 순서로 기호 표를 검색합니다.

1. 현재 명령 수준의 로컬 기호 표
2. 현재 수준으로부터 거꾸로 이전 명령 수준의 로컬 기호 표 검색
3. 글로벌 기호 표

기호 값 마스크

다음 절에서는 값 기호를 마스킹하는 방법을 설명합니다.

SET SYMBOL 명령

기본적으로 외부 명령 프로시저 수준에서 정의된 모든 기호(글로벌 및 로컬 모두)는 내부 프로시저 수준에 액세스할 수 있습니다. 그러나 **SET SYMBOL** 명령을 사용하여 명령 프로시저의 로컬 또는 글로벌 기호를 다른 명령 프로시저에서 정의된 기호와 분리할 수 있습니다. **SET SYMBOL** 명령은 로컬 및 글로벌 기호의 값을 삭제하지 않고 마스킹합니다. 따라서 명령 프로시저에서 다른 명령 프로시저를 실행할 경우 두 번째 프로시저에서 **SET SYMBOL** 명령을 지정하면 두 프로시저 모두에서 같은 기호 이름을 사용할 수 있습니다.

SET SYMBOL 명령은 또한, DCL에서 명령행을 처리하기 전에 동사 문자열(명령행의 첫 단어)을 기호로 변환하려 하는지 여부를 제어합니다. 기본 동작은 변환을 시도하는 것입니다. 이 동작을 변경하면 명령 프로시저에서 명령을 호출할 때 외부 프로시저 수준 환경에 영향을 받지 않는다는 장점이 있습니다.

기호 범위 상태

로컬 및 글로벌 기호의 **기호 범위**는 다릅니다. 이전 프로시저로 되돌아가기 위해 프로시저 수준을 종료하면 이전 수준의 기호 범위 컨텍스트가 로컬과 글로벌 기호 모두에 대해 복원됩니다.

현재의 일반적인 기호 범위 상태를 표시하려면 렉시칼 함수

F\$ENVIRONMENT("SYMBOL_SCOPE")를 사용합니다. 현재의 동사 범위 상태를 표시하려면 렉시칼 함수 **F\$ENVIRONMENT("VERB_SCOPE")**를 사용합니다.

로컬 기호 범위

로컬 기호는 프로시저 수준에 종속됩니다. 외부 프로시저 수준에서 로컬 기호를 정의하면 이 기호는 모든 내부 프로시저 수준에서 읽을 수 있습니다(쓰기는 안 됨). 외부 프로시저 수준의 로컬인 기호에 값을 할당하면 새 기호가 현재 프로시저 수준에서 만들어집니다. 그러나 외부 프로시저 수준의 기호는 수정되지 않습니다.

`SET SYMBOL/SCOPE=NOLOCAL` 명령을 사용하면 외부 프로시저 수준에서 정의된 모든 로컬 기호를 현재 프로시저 수준과 모든 내부 수준에서 액세스할 수 없게 됩니다. 예를 들어, 프로시저 수준 2와 4에서 `SET SYMBOL/SCOPE=NOLOCAL`을 지정하는 경우 다음과 같습니다.

- 프로시저 수준 2에서는 수준 2 로컬 기호만 읽고 쓸 수 있습니다.
- 프로시저 수준 3에서는 수준 2 로컬 기호를 읽기만 할 수 있습니다(쓰기는 안 됨). 수준 3은 수준 3 로컬 기호도 읽고 쓸 수 있습니다.
- 프로시저 수준 4에서는 수준 4 로컬 기호만 읽고 쓸 수 있습니다.

글로벌 기호 범위

글로벌 기호는 프로시저 수준에 관계 없습니다. 현재 글로벌 기호 범위 컨텍스트가 후속적으로 모든 프로시저 수준에 적용됩니다.

`/SCOPE=NOGLOBAL` 한정자를 지정하면 `/SCOPE=GLOBA` 한정자를 지정하거나 글로벌 기호를 액세스할 수 있는 이전 수준으로 빠져 나가지 않는 한 모든 글로벌 기호를 후속하는 모든 명령에서 액세스할 수 없게 됩니다. 또한, `/SCOPE=NOGLOBAL` 한정자를 지정하면 `/SCOPE=GLOBAL` 한정자를 지정하기 전까지 새 글로벌 기호를 만들 수 없습니다.

기호 대체 이해

특정 상황에서 **DCL**은 기호 이름이나 렉시칼 함수와 같은 문자로 시작하는 문자열을 사용합니다. 이런 상황에서 **DCL**은 기호 또는 렉시칼 함수를 해당 값으로 대체하려고 합니다. 기호를 현재 값으로 대체하는 것을 기호 대체라고 합니다. 다른 상황에서 기호나 렉시칼 함수를 사용할 경우 대체 연산자를 사용하여 기호 대체를 요청해야 합니다.

DCL은 기호와 렉시칼 함수를 다음과 같이 사용할 경우 자동으로 평가합니다.

- 할당(=) 문의 오른쪽에서
- 렉시칼 함수의 인수에서
- **IDEPOSIT**, **EXAMINE**, **IF** 또는 **WRITE** 명령에서
- 문자열 다음에 등호 또는 콜론이 올 때 명령행의 시작 부분에서
- 부분 문자열 대체나 숫자 오버레이(468페이지의 “부분 문자열 대체” 참조)를 수행하고 있을 때 할당문의 왼쪽에 있는 대괄호에서

다음 보기에서는 명령 인터프리터가 영문자로 시작하는 문자열을 기호 이름으로 사용하고 숫자나 기수 연산자(%)로 시작하는 문자열을 리터럴 숫자 값으로 사용합니다.

- 다음 보기에서 **COUNT**는 자동으로 기호로 인식되고 계산됩니다.

```
$ TOTAL = COUNT + 1
```

- 이 보기의 두 번째 줄에서 기호 **QUERY**는 **F\$LENGTH** 함수에서 사용될 때 자동으로 계산됩니다. 또한, **F\$LENGTH** 함수는 할당문의 오른쪽에 있으므로 자동으로 계산됩니다.

```
$ QUERY = "Have we met before?"
$ LEN = F$LENGTH(QUERY) + 5
$ SHOW SYMBOL LEN
      LEN = 27      Hex = 0000001B      Octal = 000033
```

- 다음 보기에서 **IF** 명령은 **A**와 **B**를 기호 이름으로 사용하여 그 현재 값을 사용합니다.

```
$ IF A .EQ. B THEN WRITE SYS$OUTPUT "DONE"
```

- 이 보기의 두 번째 줄에서 명령 인터프리터는 자동으로 **PDEL**을 현재 값으로 대체하고 결과 명령을 실행합니다.

```
$ PDEL = "DELETE SYS$PRINT/ENTRY="
$ PDEL 181
```

- 다음 보기에서 DCL은 자동으로 기호 BELL을 7의 값으로 정의하고 할당문의 왼쪽에 있는 괄호 안의 값에 따라 새 값을 할당합니다.

```
$ BELL = 7
$ BELL[5,1] = 1
$ SHOW SYMBOL BELL
BELL = 39    Hex = 00000027    Octal = 00000000047
```

강제 기호 대체

나열된 위치 중 하나에 없는 기호를 강제로 대체하려면 다음과 같이 기호를 어포스트로피(')로 묶습니다.

```
$ TYPE 'B'
```

인용된 문자열 내의 기호를 강제로 대체하려면 다음과 같이 해당 기호 앞에 두 개의 어포스트로피(')를 붙이고 기호 뒤에 한 개의 어포스트로피(')를 붙입니다.

```
$ T = "TYPE ''B'"
```

명령행을 처리할 때 DCL은 다음과 같은 순서로 기호를 해당 값으로 대체합니다.

- 강제 대체

DCL은 왼쪽에서 오른쪽으로 어포스트로피(또는 인용 부호 내의 문자열인 경우 두 개의 어포스트로피)로 분리된 모든 문자열을 대체합니다. 한 개의 어포스트로피가 앞에 오는 기호는 반복적으로 변환되지만 두 개의 어포스트로피가 앞에 오는 기호는 그렇지 않습니다.

- 자동 대체

DCL은 왼쪽에서 오른쪽으로 명령행의 각 값을 평가하여 명령이면 실행하고 표현식이면 계산합니다. 표현식의 기호는 할당된 값으로 대체되지만 반복적으로 대체되지는 않습니다.

다음 보기에서는 DCL이 기호를 대체하는 순서에 따른 영향을 보여 줍니다. 우선 기호 PN, FILE1 및 NUM이 정의됩니다.

```
$ PN = "PRINT/NOTIFY"
$ FILE1 = "[BOLIVAR]TEST_CASE.TXT"
$ NUM = 1
```

기호가 정의되면 다음 명령은 [BOLIVAR]TEST_CASE.TXT라는 파일을 인쇄합니다.

```
$ FILE = "'FILE''NUM'"
$ PN 'FILE'
```

첫 번째 명령에서는 강제 대체로 NUM은 1이 되므로 'FILE' 'NUM'은 FILE1이 됩니다. 명령 SHOW SYMBOL FILE을 입력하면 FILE = “'FILE1'”이 표시됩니다.

두 번째 명령은 두 가지 대체를 수행합니다. 우선 'FILE'이 'FILE1'로 대체됩니다. 'FILE1'도 어포스트로피(')로 묶이므로 대체가 필요합니다. 자동 대체로 FILE1은 [BOLIVAR]TEST_CASE.TXT가 됩니다. 그런 다음 이 파일 이름이 PRINT/NOTIFY인 PN 값에 추가됩니다. 따라서 다음과 같은 문자열이 됩니다.

```
$ PRINT/NOTIFY [BOLIVAR]TEST_CASE.TXT
```

기호 대체 연산자

DCL에서 통상적으로 대체를 수행하지 않는 경우 대체 연산자를 사용하여 기호 대체를 요청할 수 있습니다. DCL에서는 두 가지 대체 연산자를 사용할 수 있습니다.

- 어포스트로피(')
- 앰퍼샌드(&)

이 두 연산자는 대체가 일어나는 시간이 다릅니다. 어포스트로피가 앞에 붙은 기호는 DCL 명령 처리의 첫 번째 단계에 대체되지만 앰퍼샌드가 앞에 오는 기호는 두 번째 단계에 대체됩니다. 명령 처리 단계에 대한 자세한 내용은 494페이지의 “명령 처리 3 단계”를 참조하십시오.

어포스트로피(')

어포스트로피(')는 가장 자주 사용되는 대체 연산자입니다. 명령 매개 변수나 한정자 대신 기호를 사용할 때 기호 대체를 요청하려면 어포스트로피를 사용합니다. 어포스트로피를 사용하여 문자열 할당(=) 문의 오른쪽에서 기호 대체를 요청합니다.

인용된 문자열에서 기호 대체를 요청하려면 기호 이름 앞에 두 개의 어포스트로피와 이름 뒤에 한 개의 어포스트로피를 붙입니다.

어포스트로피를 사용하여 기호 대체를 요청할 경우 대체 값의 중간에서 하이픈 연결 문자로 줄을 계속할 수 없습니다.

다음 보기에서 TYPE 명령은 파일 지정 방법을 필요로 합니다. 어포스트로피는 LIT가 평가되어야 하는 기호임을 나타냅니다. 어포스트로피를 생략하면 DCL은 LIT.LIS라는 파일을 찾습니다. LIS는 TYPE 명령의 기본 파일 유형입니다.

```
$ LIT = "LIGHT.BILLS"
$ TYPE 'LIT'
```

다음 보기에서는 **FILE**이 **REPORT.DAT**가 되도록 **NAME** 값을 대체합니다.

```
$ NAME := REPORT
$ FILE := 'NAME'.DAT
$ SHOW SYMBOL FILE
FILE = "REPORT.DAT"
```

다음 보기에서 기호 **NAME**의 현재 값은 **FRED**입니다.

```
$ MESSAGE = "Creating file '"NAME'.DAT"
```

따라서 **MESSAGE**는 다음과 같은 값을 가집니다.

```
Creating file FRED.DAT
```

앰퍼샌드(&)

앰퍼샌드(&)도 명령 인터프리터에서 인식하는 대체 연산자입니다. 대부분의 경우 어포스트로피와 앰퍼샌드는 같은 기능을 수행합니다. 앰퍼샌드가 가장 효율적인 대체 연산자로 사용되는 경우는 어포스트로피와 함께 사용되어 대체가 수행되는 순서에 영향을 주는 경우입니다.

기호가 정의되지 않은 경우에 명령 인터프리터에서 수행하는 작업은 명령의 컨텍스트에 따라 다릅니다. 자세한 내용은 497페이지의 “정의되지 않은 기호”를 참조하십시오.

다음의 첫 번째 명령에서는 명령 인터프리터가 명령 처리의 첫 단계(스캔) 동안 기호 **NAME**을 현재 값으로 대체합니다. 두 번째 명령에서는 명령 처리의 두 번째 단계(구문 분석) 동안 기호 **NAME**을 현재 값으로 대체합니다. 방법은 다르지만 결과는 같습니다.

```
$ TYPE 'NAME'
$ TYPE &NAME
```

다음 보기에서는 어포스트로피와 함께 앰퍼샌드(&)를 사용하여 대체 순서에 영향을 줍니다.

```
$ P1 = "FRED.DAT"
$ COUNT = 1
$ TYPE &P'COUNT'
```

먼저 명령 인터프리터는 어포스트로피로 묶인 기호('COUNT')를 평가합니다.

결과는 다음과 같습니다.

```
TYPE &P1
```

두 번째로 명령 인터프리터는 앰퍼샌드가 앞에 오는 기호(P1)를 평가합니다.

결과는 다음과 같습니다.

```
TYPE FRED.DAT
```


다음 보기에서는 **P**와 **COUNT** 모두에 어포스트로피를 사용합니다.

```
$ TYPE 'P' 'COUNT'
```

명령 인터프리터는 왼쪽에서 오른쪽으로 작업하며 **P**를 평가합니다. **P**는 정의된 기호가 아니므로 **DCL**은 **null** 값을 부여합니다. 다음으로 기호 **COUNT**를 평가합니다. 결과는 다음과 같습니다.

```
TYPE 1
```

다음 보기에서 **A**는 **B**의 현재 값과 일치합니다.

```
$ B = "MYFILE.DAT"
$ A = "&B"
$ TYPE 'A'
```

앰퍼샌드(&)를 인용 부호(" ") 안에서 사용하면 기호 대체가 일어나지 않습니다. 따라서 할 당할 때 **B** 값은 대체되지 않습니다. 그러나 **TYPE** 명령은 **MYFILE.DAT**를 표시합니다. 그 이유는 명령 인터프리터가 우선 **A**를 **&B** 값으로 대체한 다음 기호 **&B**를 **MYFILE.DAT**로 대체하기 때문입니다. **B**를 다시 정의하면 **TYPE** 명령의 결과도 따라 변합니다.

앰퍼샌드를 사용할 경우 다음 규칙을 따릅니다.

- 앰퍼샌드를 기호 이름 뒤가 아닌 앞에 놓습니다.
- 앰퍼샌드는 구분 기호(공백이나 특수 문자) 다음에 와야 합니다.
- 인용 부호(" ")로 묶인 문자열 내에서는 앰퍼샌드를 사용하여 대체를 요청할 수 없습니다.
- 두 개 이상의 기호 이름을 연결하기 위해 앰퍼샌드를 사용할 수 없습니다.
- 일반적으로 기호를 올바른 변환하기 위해 필요한 경우를 제외하고는 기호 대체를 위해 앰퍼샌드를 사용하지 않습니다.

명령 처리 3 단계

명령 인터프리터는 3 단계로 기호 대체를 수행합니다.

1 단계: 명령 입력 스캔

명령 입력 스캔(렉시칼 입력 단계라고도 함) 시 명령 인터프리터는 왼쪽에서 오른쪽으로 어포스트로피가 앞에 붙은 기호를 평가합니다. 한 개의 어포스트로피가 앞에 있는 기호는 1 단계 대체에서 설명한 대로 반복해서 변환하고 두 개의 어포스트로피가 앞에 오는 기호는 반복해서 변환하지 않습니다.

2 단계: 명령 구문 분석

명령 구문 분석 단계에서 다음을 수행합니다.

- 명령 인터프리터가 명령행을 분석합니다. 명령행의 첫 번째 항목이 기호인지 확인합니다. 기호이면 평가합니다.
 - 명령 인터프리터가 왼쪽에서 오른쪽으로 어포스트로피가 앞에 붙은 기호를 평가합니다.
- 이 단계 동안의 기호 대체는 반복되지 않습니다.

3 단계: 표현식 평가

표현식 평가 단계에서는 다음을 수행합니다.

- 명령 인터프리터가 **DEPOSIT**, **EXAMINE**, **IF** 및 **WRITE** 명령이 앞에 오는 기호를 평가합니다.
- 명령 인터프리터가 렉시칼 함수 내의 기호를 평가합니다.

이 단계 동안의 기호 대체는 반복되지 않습니다.

명령 인터프리터는 명령 프로시저 내에서 실행된 명령이나 프로그램이 입력 데이터로 읽은 행은 스캔하지 않습니다. 따라서 명령 인터프리터는 이런 데이터 행에서 기호 대체를 수행하지 않습니다.

다음 보기에서는 프로그램 **AVERAGE**가 **SYS\$INPUT**(명령 입력 스트림)에서 **55**, **57** 및 **9999**를 읽습니다. 명령 인터프리터에서는 이 데이터 행을 읽지 않습니다. 입력 데이터로 입력한 기호 이름은 평가되지 않습니다.

```
$ RUN AVERAGE
55
57
9999
```

반복적인 대체

기호 대체는 반복될 수 있습니다.

- 단일 명령행에서 둘 이상의 대체 유형이 발생하는 경우 반복적인(**Repetitive**) 대체가 일어납니다.
- 명령 인터프리터에서 대체된 값 자체가 기호인지 확인하는 경우 반복적인(**iterative**) 대체가 발생합니다. 반복적인(**iterative**) 대체는 명령 처리의 1 단계에 어포스트로피가 앞에 붙은 기호를 변환할 때만 발생합니다.

1 단계 대체

어포스트로피(')를 사용하여 기호 대체를 요청하면 명령 인터프리터는 명령 처리의 1 단계 동안에 반복적인(**iterative**) 대체를 수행합니다.

기호가 인용된 문자열 안에 포함될 경우 어포스트로피를 사용한 대체가 반복되지 않습니다.

다음 보기에서는 대체가 반복됩니다.

```
$ MAC = "5"
$ A = "'MAC'"
$ B = 'A'
$ SHOW SYMBOL B
B = 5 Hex = 00000005 Octal = 00000000005
```

B = 'A' 문 이후에 기호 **B**의 값은 다음과 같은 이유로 **5**가 됩니다.

- 기호 이름 **A**는 어포스트로피로 묶여 있으므로 현재 값(**"MAC"**)으로 대체됩니다.
- 이 값(**"MAC"**)도 어포스트로피로 묶여 있으므로 명령 인터프리터는 **MAC**를 현재 값(**5**)으로 대체합니다.
- 이 값(**5**)에는 어포스트로피가 없으므로 명령 처리의 1 단계가 완료됩니다. 2 단계와 3 단계에서는 더 이상의 대체가 필요하지 않습니다. 따라서 **5**가 기호 이름 **B**에 지정되는 마지막 값이 됩니다.

그러나 인용된 문자열에 **A**를 포함시키면 다음과 같습니다.

```
$ B = "'A'"
$SHOW SYMBOL B
B = "'MAC'"
```

이 경우 **B**의 값은 **'MAC'**이 됩니다. 인용된 문자열 내에서는 대체가 반복되지 않으므로 기호 이름 **A**는 한 번만 대체됩니다.

2 단계 대체

명령 인터프리터는 명령행에 어포스트로피가 있는 경우에만 자동으로 반복적인(iterative) 대체를 수행합니다. 일부 경우에는 명령 동의의 정의를 중첩시킬 수도 있습니다.

다음 보기에서는 **EXEC**가 처리될 때 명령 인터프리터가 한 번만 대체를 수행합니다.

```
$ MAC = "TYPE A.B"
$ EXEC = "'MAC'"
$ EXEC
```

결과는 문자열 **'MAC'**이 됩니다. 명령 인터프리터는 **MAC**을 명령으로 인식하지 않으므로 오류 메시지를 표시합니다. 이 오류는 명령 처리 1단계에서 아무 대체도 수행되지 않았기 때문에 일어납니다(문자열 **EXEC**는 어포스트로피로 구분되지 않음). 2 단계에서 **EXEC**는 명령행이 첫 번째 값이므로 문자열 **'MAC'**으로 대체됩니다. 이 대체는 반복되지 않습니다. 따라서 **'MAC'**이 어포스트로피로 구분되어 있더라도 추가 대체가 수행되지 않습니다.

명령 동의어 **EXEC**를 올바르게 사용하려면 다음과 같이 어포스트로피로 묶습니다.

```
$ 'EXEC'
```

이 경우 기호 **EXEC**는 명령 처리 1 단계에서 평가됩니다. 대체가 반복되므로 (**'MAC'**)도 평가되어 문자열 **TYPE A.B**가 대체됩니다.

3 단계 대체

명령 인터프리터가 명령의 표현식을 분석할 때 식에 지정된 기호는 한 번만 대체됩니다. 그러나 식에 어포스트로피나 앰퍼샌드를 사용하여 대체가 강제로 반복하게 할 수 있습니다. 이렇게 강제로 반복하게 할 경우 다음을 주의하십시오.

- 명령 인터프리터는 어포스트로피와 앰퍼샌드로 요청한 모든 대체를 명령행을 실행하기 전에 수행합니다.
- 기호 대체를 자동으로 수행하는 명령은 명령 처리 1 단계와 2 단계 이후에 실행됩니다.

그러나 유효한 기호 이름으로 대체되지 않을 경우 명령이 실패합니다.

다음 보기에서는 **IF** 명령에서의 반복적인 대체를 보여 줍니다.

```
$ P1 = "FRED.DAT"
$ COUNT = 1
$ IF P'COUNT' .EQS. "" THEN GOTO END
```

명령 인터프리터가 이 줄을 스캔할 때 기호 **COUNT**를 현재 값으로 대체합니다. 결과는 다음과 같습니다.

```
IF P1 .EQS. "" THEN GOTO END
```

이 문자열에는 어포스트로피가 없으므로 명령 인터프리터는 더 이상의 대체를 수행하지 않습니다. 그러나 **IF** 명령이 실행 될 때 기호 이름 **P1**를 자동으로 평가하여 현재 값으로 대체합니다.

다음 보기에서 기호 이름 **FILENAME**은 잘못되었습니다.

```
$ FILENAME = "A.B"
$ IF 'FILENAME' .NES. "" THEN TYPE 'FILENAME'
```

명령 인터프리터는 기호 **FILENAME**을 현재 값(**A.B**)으로 대체합니다. 결과는 다음과 같습니다.

```
IF A.B .NES. "" THEN TYPE A.B
```

IF 명령이 명령행을 실행할 때 **A.B**는 유효한 기호가 아니므로 오류가 일어납니다. 이 **IF** 명령을 올바르게 처리하려면 다음과 같이 어포스트로피를 생략합니다.

```
$ IF FILENAME .NES. "" THEN TYPE 'FILENAME'
```

정의되지 않은 기호

기호가 명령행에서 사용될 때 정의되어 있지 않으면 명령 인터프리터는 상황에 따라 오류 메시지를 표시하거나 기호를 **null** 문자열로 대체합니다. 규칙은 다음과 같습니다.

- 명령 처리 1 단계와 2 단계에 명령 인터프리터는 어포스트로피나 앰퍼샌드가 앞에 오는 정의되지 않은 모든 기호를 **null** 문자열로 대체합니다.
- 명령 처리 3 단계에 명령 인터프리터는 정의되지 않은 기호를 찾으면 경고 메시지를 표시하고 처리를 마치지 않습니다.

명령 처리 3 단계

다음 보기에서는 명령 인터프리터가 어포스트로피가 앞에 붙은 정의되지 않은 기호를 처리하는 방법을 보여 줍니다.

```
$ FILE := MYFILE'FILE_TYPE'
$ SHOW SYMBOL FILE
    FILE = "MYFILE"
$ PRINT 'FILE'
```

기호 **FILE**이 만들어질 때 기호 **FILE_TYPE**은 현재 값으로 대체됩니다. **FILE_TYPE**가 정의되지 않은 경우 명령 인터프리터는 **FILE_TYPE**을 null 문자열로 대체합니다. 파일 지정 방법에 파일 유형이 없으므로 **PRINT** 명령은 기본 파일 유형 **.LIS**를 사용합니다. 따라서 파일 사용이 **MYFILE.LIS**로 변환됩니다.

다음 보기에서는 표현식이 명령 처리 3 단계에서 평가됩니다.

```
$ A = 1
$ C = A + B
%DCL-W-UNDSYM, undefined symbol - check validity and spelling
```

기호 **B**가 정의되어 있지 않으므로 명령 인터프리터는 식을 평가할 수 없습니다.

기호 사용에 대한 대안: 자동 외부 명령

프로시저에 대한 기호를 정의하지 않고 DCL 수준에서 명령 프로시저(.COM 파일 유형)를 호출하거나 실행 가능한 이미지(.EXE 파일 유형)를 실행할 수도 있습니다. DCL은 자동 외부 명령을 사용하여 명령 프로시저의 특정 디렉토리 집합이나 실행 가능한 이미지를 찾아 자동으로 실행할 수 있습니다.

DCL 기호가 아니며 DCL 명령 표에 없는 명령 동사를 입력하면 시스템은 일반적으로 다음 메시지를 표시합니다.

```
DCL-W-IVVERB, unrecognized command verb - check validity and spelling
```

그러나 논리 이름 DCL\$PATH가 공백이 아니고 정의되어 있으면 DCL은 대신 파일 이름에 잘못된 동사를 포함하는 파일에 대해 RMS \$SEARCH를 수행하고 기본 파일 지정 방법인 DCL\$PATH:.*를 수행합니다.

DCL에서 .COM 또는 .EXE 파일을 찾으면 DCL은 나머지 명령행을 매개 변수로 사용하여 해당 파일을 자동으로 실행합니다. 이 동작은 DOS, UNIX 등의 다른 운영 체제에 있는 PATH 옵션과 비슷합니다.

다음 보기에서는 DCL 기호 SYSGEN이 더 이상 필요 없습니다. DCL은 SYS\$SYSTEM 디렉토리를 조회하여 SYSGEN.EXE를 찾습니다. DCL은 “\$SYS\$SYSTEM:SYSGEN”으로 정의된 기호 “SYSGEN”과 같이 작동하여 SYSGEN 이미지가 외부 명령으로 활성화됩니다.

```
$ SYSGEN
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
\SYSGEN\
$ DEFINE DCL$PATH SYS$SYSTEM,SYS$DISK:[ ]FOO
$ SYSGEN SHOW MAXPROCESSCNT
```

Parameter Name	Current	Default	Min.	Max.	Unit	Dynamic
MAXPROCESSCNT	157	32	12	8192	Processes	

다음 보기에서는 DCL이 자동으로 SYS\$SYSTEM 디렉토리에서 SS.COM 또는 SS.EXE를 찾으므로 SS를 “@SS.COM”으로 정의할 필요가 없습니다. DCL은 파일을 찾지 못하면 현재 디렉토리에서 SS.COM이나 SS.EXE를 찾습니다.

```
$ TYPE SS.COM
$ SHOW SYMBOL/LOCAL/ALL
$ EXIT
$ SS "This is a parameter"
    P1 = "This is a parameter"
    P2 = " "
    P3 = " "
    P4 = " "
    P5 = " "
    P6 = " "
    P7 = " "
    P8 = " "
$ SS.EXE "This is a parameter"
    P1 = ".EXE"
    P2 = "This is a parameter"
    P3 = " "
    P4 = " "
    P5 = " "
    P6 = " "
    P7 = " "
    P8 = " "
```

위의 보기에서 DCL은 SS.COM을 찾고 “SS”가 “@SS.COM”으로 정의된 기호인 것처럼 작동합니다. 구문 분석된 나머지 명령행 부분을 매개 변수로 사용하여 명령 프로시저가 활성화됩니다. “SS.EXE”에서는 이미지 SS.EXE를 호출하지 않고 대신 두 개의 매개 변수(첫 번째는 텍스트 문자열 “.EXE”임)와 함께 SS.COM을 호출합니다. 이는 OpenVMS 운영 체제에서 명령 구문 분석과 기호 대체가 수행되는 방법과 같습니다.

자동 외부 명령 사용

다음을 참고하십시오.

- 논리 이름 DCL\$PATH는 검색 목록 형식 논리가 될 수 있습니다.
- 논리 이름의 각 변환 중 노드, 장치 및 디렉토리 부분만 사용됩니다.
- 일반 논리 우선 순위가 우선합니다. 사용자의 직접 정의하여 DCL\$PATH에 대한 시스템 정의를 무시할 수 있습니다. 시스템 정의가 있는 데 사용자가 그 기능을 원하지 않을 경우 “”에 대한 정의로 논리를 무시하여 기능을 끌 수 있습니다.

- DCL 동사와 기호 이름의 유효한 문자 집합이 파일 이름의 유효한 문자 집합과 다릅니다. 예를 들어, DCL 기호는 하이픈(-)을 포함할 수 없거나 달러 기호(\$)로 시작할 수 없습니다. 실행할 이미지나 프로시저가 DCL 기호 이름으로 유효하지 않은 경우 이 새 기능에서 직접 호출할 수 없습니다.
- DCL은 명령을 구문 분석하지 않았습니다. 호출될 이미지에서 자체적으로 명령 구문 분석을 수행합니다. C 프로그램인 경우 `main()` 루틴에 `“argc”`와 `“argv”` 매개 변수를 사용합니다. 다른 언어로 작성된 프로그램인 경우 `LIB$GET_FOREIGN`을 호출하여 전체 명령행을 가져와서 프로그램에서 구문 분석해야 합니다.
- 디렉토리에는 명령 프로시저와 실행 가능한 이미지가 모두 들어 있는 경우 처음 찾은 파일이 무엇이든 호출됩니다. OpenVMS 시스템에서 디렉토리는 사전순으로 정렬되므로 `“.COM”` 파일은 `“.EXE”` 파일 앞에 있습니다. 다른 운영 체제를 실행하는 노드를 가리키는 `DCL$PATH` 논리의 네트워크 파일에서는 `“.EXE”` 파일이 `“.COM”` 파일 앞에 있을 수 있습니다.

DCL에서는 잘못된 동사를 파일 지정 방법으로 사용하고 `“DCL$PATH:.”`를 기본 파일 지정 방법으로 사용하여 검색을 수행하므로 특정 파일을 찾는 방식으로 논리를 정의할 수 있습니다. 예를 들어, 논리 `FOO`가 `“FOO.EXE”`가 되게 정의하고 DCL 프롬프트에 `“FOO”`를 입력하면 `FOO.COM`은 호출하지 않고 `FOO.EXE`만 호출합니다.

주

사용자가 권한이 있는 사용자이고 자신의 기본 장치와 디렉토리를 다른 사용자의 계정에 설정한 경우 `“SYS$DISK:”`를 `DCL$PATH` 논리 이름 정의에 넣을 수 없습니다. 이렇게 하면 DCL에서 현재 디렉토리를 검색합니다. 따라서 검색 목록에서의 글자 오류나 잘못된 변환 배치 때문에 현재 디렉토리의 사용자 이미지가 발견되어 권한을 가지고 잘못 호출될 수 있습니다.

자동 외부 명령 제한

다음 제한 사항을 참고하십시오.

- OpenVMS 운영 체제 6.2 이전 버전에서는 자동 외부 명령을 사용할 수 없습니다.
- 새 동사가 언제라도 **DCL** 명령 표에 추가될 수 있으므로 자동 외부 명령에서 작동되던 명령이 작동하지 않을 수도 있습니다.
- 자동 외부 명령 기능은 모든 경우에 작동하는 것은 아닙니다. 다음 보기에서는 동사의 처음 네 문자만 검색하는 **DCL**이 **SHOW** 동사와 일치하는 것을 찾으면 (**SHOWME**의 처음 네 문자) **SHOWME.COM** 프로시저 대신 **SHOW USERS** 명령을 실행합니다. **SHOWME**를 **DCL** 기호를 정의하면 **SHOWME** 명령은 **SHOWME.COM**을 호출합니다.

```
$ DEFINE DCL$PATH SYS$SYSTEM,SYS$DISK:[]FOO
$ TYPE SHOWME.COM
$ SHOW SYMBOL P1
$ EXIT
$ SHOWME USERS

OpenVMS User Processes at MARCH 2, 1999 01:40 PM
Total number of users = 1,  number of processes = 11

Username      Interactive  Subprocess   Batch
RSMITH                9             2
```

13 명령 프로시저 소개

명령 프로시저란 DCL 명령과 DCL 명령이 사용하는 데이터 행이 들어 있는 파일입니다. 일부 단순 명령 프로시저에는 한 개 또는 두 개의 DCL 명령만 포함될 수 있지만, 복합 명령 프로시저는 정교한 컴퓨터 프로그램과 같은 기능을 할 수 있습니다. 명령 프로시저가 실행되면 DCL 인터프리터가 해당 파일을 읽고 그 파일에 포함된 명령을 실행합니다.

시스템 관리자가 **시스템 로그인 명령 프로시저**를 설정한 경우에는 사용자가 로그인할 때마다 그 명령 프로시저가 실행됩니다. 시스템 로그인 명령 프로시저를 사용하면 시스템 상의 사용자들이 로그인할 때 특정 명령이 항상 실행되도록 할 수 있습니다.

시스템 로그인 명령 프로시저 실행 후 시스템에서는 사용자의 **개인 로그인 명령 프로시저**가 있을 경우 그 명령 프로시저를 실행합니다. 사용자의 개인 로그인 명령 프로시저를 사용하여 자신의 컴퓨팅 환경을 사용자 정의할 수 있습니다. 사용자가 로그인할 때마다 그 안에 포함된 명령이 실행됩니다. 사용자가 로그인하면 시스템에서는 로그인 명령 프로시저를 두 개(시스템 로그인 명령 프로시저와 사용자 로그인 명령 프로시저)까지 실행합니다.

계정을 설정해 주는 사람이 사용자의 최상위 디렉토리에 로그인 명령 프로시저를 두었을 수도 있습니다. 로그인 명령 프로시저가 자신의 최상위 디렉토리에 없을 경우에는 사용자가 직접 만들 수 있습니다. 이름을 LOGIN.COM이라고 지정하여 최상위 디렉토리에 둡니다. 시스템 관리자가 다른 언급을 하지 않는 한, 사용자가 만드는 LOGIN.COM 파일은 사용자가 로그인할 때마다 실행됩니다.

이 장의 내용은 다음과 같은 주요 절로 나누어집니다.

- 명령 프로시저 작성을 위한 기본 정보
- 명령 프로시저 작성을 위한 단계별 절차
- 명령 프로시저 실행
- 종료, 중단 및 오류 처리 명령 프로시저
- 로그인 명령 프로시저

DCL 명령 프로시저는 다음 두 가지 유형이 있습니다.

- 단순
 - 일련의 DCL 명령을 작성된 순서대로 실행합니다.
- 복합
 - 프로그램과 유사한 기능을 수행합니다.

명령 프로시저 작성을 위한 기본 정보

명령 프로시저를 만드는 방법은 두 가지가 있습니다.

- EVE 같은 텍스트 편집기를 사용하여 새 파일 만들기
- DCL 명령 CREATE를 사용하여 새 파일 만들기

사용자가 만드는 파일에는 명령행, 레이블, 주석, 조건문, 변수 등이 포함될 수 있습니다.

기본 파일 유형

명령 프로시저의 기본 파일 유형은 .COM입니다. 명령 프로시저의 이름을 지정할 때 .COM 파일 유형을 지정할 경우, 해당 파일 이름만 지정하여 프로시저를 실행할 수 있습니다. 별도로 지정하지 않는 한, SUBMIT과 실행 프로시저(@) 명령의 파일 유형은 .COM으로 간주됩니다.

명령 작성

다음은 명령 프로시저에 명령을 포함하는 작업에 대한 제안 사항입니다.

- 명령과 한정자에 대해 전체 이름을 사용합니다. 전체 이름을 사용하면 나중에 릴리즈 되는 OpenVMS, 즉 상위 버전과의 호환성을 유지하는 데 도움이 됩니다.
- 연속선을 사용하여 프로시저를 더욱 읽기 쉽도록 만듭니다. 연속선은 달러 기호로 시작되지 않는 것에 주의하십시오. 예를 들면 다음과 같습니다.

```
$ PRINT LAB.DAT      -  
    /AFTER=17:00      -  
    /COPIES=20         -  
    /NAME="COMGUIDE"
```

명령행 작성

명령행 작성 시 주의할 사항은 다음과 같습니다.

- 명령, 주석 또는 레이블이 포함된 각 행의 맨 앞에 달러 기호(\$)를 사용해야 합니다.
- 데이터가 들어 있는 행을 포함시키려면 해당 행에서 달러 기호(\$)를 생략합니다.

- 달러 기호(\$)로 시작되는 데이터 행을 포함해야 할 경우, DCL 명령 DECK 및 EOD를 사용합니다. 예를 들면 다음과 같습니다.

```
$ ! Everything between the commands DECK and EOD
$ ! is written to the file WEATHER.COM
$ !
$ CREATE WEATHER.COM
$ DECK
$ FORTRAN SUMMER
$ LINK SUMMER
$ RUN SUMMER
$ EOD
$ !
$ ! Now execute WEATHER.COM
$ @WEATHER
$ EXIT
```

달러 기호로 시작되지 않는 명령행은 DCL에 의해 중단될 수 있지만, 모든 DCL 명령행 앞에는 달러 기호를 반드시 사용하는 것이 좋습니다.

명령행에서 레이블 사용

레이블은 DCL 명령 프로시저에서 루프, 코드의 섹션, 서브 루틴 등을 표시하는 데 사용됩니다. 레이블을 사용할 때는 다음 규칙에 주의하십시오.

- 루프, 서브 루틴 및 조건 코드가 눈에 더욱 잘 띄도록 별도의 행에 레이블을 둡니다.
- 길이가 255자보다 짧고 공백이 없는 레이블 이름을 사용합니다.
- 달러 기호(\$) 바로 다음에 레이블을 넣고 명령 앞에 공백을 넣어서 명령과 레이블을 구분합니다.
- 각 레이블 끝에 콜론을 사용합니다.
- 레이블을 삭제할 수 없습니다.

로컬 기호 테이블의 레이블

명령 인터프리터가 레이블을 만나면 그 레이블을 로컬 기호 테이블의 특수 섹션에 넣습니다. 레이블용으로 사용할 수 있는 공간의 양은 제한되어 있습니다. 명령 프로시저에 많은 레이블이 포함되어 있고 많은 기호가 사용될 경우에는 명령 인터프리터의 기호 테이블 공간이 모자라서 오류 메시지가 나타날 수 있습니다. 오류 메시지가 나타날 경우, DELETE/SYMBOL 명령을 프로시저에 포함하여 더 이상 필요하지 않은 기호를 삭제합니다. 그러나 레이블은 삭제할 수 없습니다.

중복 레이블

명령 프로시저에서 동일한 레이블을 두 번 이상 사용할 경우, 로컬 기호 테이블의 기존 정의가 새 정의로 바뀝니다.

중복 레이블이 있을 경우, GOTO 명령은 DCL이 최근에 처리한 레이블로 제어를 넘깁니다. 또한 GOTO 명령은 중복 레이블을 처리할 때 다음과 같은 규칙을 사용합니다.

- 모든 중복 레이블이 GOTO 명령 앞에 올 경우, 해당 GOTO 명령에 가장 가까운 레이블로 제어가 넘어갑니다.
- 중복 레이블이 GOTO 명령의 앞뒤로 올 경우, 해당 GOTO 명령에 가장 가까운 앞쪽 레이블로 제어가 넘어갑니다.
- 모든 중복 레이블이 GOTO 명령 뒤에 올 경우, 해당 GOTO 명령에 가장 가까운 레이블로 제어가 넘어갑니다.

명령 프로시저에서 주석 사용

명령 프로시저에 주석을 포함하는 것은 좋은 프로그래밍 습관입니다. 주석은 명령 프로시저의 업데이트 또는 문제 해결 시에 도움이 될 수 있습니다. 주석을 사용하는 방법은 다음과 같습니다.

- 프로시저의 시작 부분에서 프로시저를 설명하고 그 프로시저에 전달된 매개 변수를 설명합니다.
- 각 명령 블록의 시작 부분에서 프로시저의 해당 섹션을 설명합니다.
- 일련의 명령을 달러 기호와 감탄 부호(!)가 포함된 행으로 구분합니다. 이렇게 하면 명령 프로시저의 개요를 더욱 쉽게 파악할 수 있습니다. 공백 행을 삽입하면 명령 인터프리터에서 그 행을 데이터 행으로 해석하므로, 해당 데이터 행이 무시되었다는 경고 메시지가 만들어집니다.

명령 프로시저에서 주석을 작성할 때는 다음 규칙이 적용됩니다.

- 감탄 부호(!)를 사용하여 주석이 시작됨을 나타냅니다. 명령 인터프리터는 명령 프로시저가 실행될 때 감탄 부호의 오른쪽에 있는 모든 텍스트를 무시합니다.
- 명령행에 감탄 부호 문자 자체를 포함하려면 감탄 부호를 인용 부호(" ")로 묶습니다.

명령 프로시저 작성 방법

명령 프로시저를 작성하려면 먼저 그 명령 프로시저가 실행할 작업을 대화형으로 수행합니다. 필요한 명령을 입력하면서 함께 사용되는 변수와 조건 및 모든 반복을 기록합니다.

다음 절에서는 단순 명령 프로시저를 작성하는 단계에 대해 설명합니다. 해당 절 전체에서 사용되는 보기는 **CLEANUP.COM**이라는 명령 프로시저입니다. 이 프로시저는 디렉토리를 정리할 때 사용할 수 있습니다.

정의

- **변수**

작업을 수행할 때마다 변경되는 데이터입니다.

- **조건**

바뀔 수 있기 때문에 작업을 수행할 때마다 테스트해야 하는 명령 또는 명령 집합입니다.

- **반복**

조건이 만족될 때까지 반복적으로 수행되는 명령 또는 명령 집합입니다.

명령 프로시저 작성 단계

다음 단계를 수행하여 명령 프로시저를 작성합니다.

표 13-1

단계	작업
1	명령 프로시저를 디자인합니다.
2	변수를 할당하고 조건을 테스트합니다.
3	루프를 추가합니다.
4	명령 프로시저를 끝냅니다.
5	프로그램 논리를 테스트하고 디버그합니다.
6	정리 작업을 추가합니다.
7	프로시저를 마칩니다.

1 단계: 명령 프로시저 디자인

다음 단계를 수행하여 명령 프로시저를 디자인합니다.

표 13-2

단계	작업
1	프로시저에서 수행할 작업을 결정합니다.
2	명령 프로시저에서 사용할 변수와 그 변수를 로드하는 방법을 결정합니다.
3	명령 프로시저에 필요한 조건과 그 조건을 테스트하는 방법을 결정합니다.
4	명령 프로시저를 종료하는 방법을 결정합니다.

대체로 정리 작업 중에 실행되는 특정 명령이 있습니다. 다음 표에서는 그러한 명령과 그러한 명령이 수행하는 작업을 나열합니다.

표 13-3

명령	수행하는 작업
DIRECTORY	현재 디렉토리의 내용을 표시합니다.
TYPE filespec	파일을 표시합니다.
PURGE filespec	파일을 소거합니다.
DELETE filespec	파일을 삭제합니다.
COPY filespec new-filespec	파일을 복사합니다.

변수

작업을 수행할 때 변경되는 데이터가 변수입니다. 디렉토리에서 파일을 만들거나 삭제할 경우, 디렉토리를 지울 때마다 파일 이름이 달라지므로 CLEANUP.COM의 파일 이름은 변수가 됩니다.

조건

명령 프로시저를 실행할 때마다 테스트되어야 하는 명령은 모두 조건으로 간주됩니다. 수행해야 할 작업에 따라 CLEANUP.COM의 명령이 모두 또는 일부가 실행될 수 있기 때문에 각 명령은 조건부입니다.

디자인 결정

CLEANUP.COM 명령 프로시저에서 사용할 변수와 조건을 결정했으면 변수를 로드하고, 조건을 테스트하고, 명령 프로시저를 종료하는 방법을 결정해야 합니다. CLEANUP.COM 명령 프로시저에 대해서는 다음과 같은 결정이 이루어졌습니다.

표 13-4

작업	수행 방법
변수 로드	명령 프로시저에서 터미널로부터 파일 이름을 가져옵니다.
조건 테스트	명령 프로시저에서 다음을 수행하십시오. - 터미널로부터 명령 이름을 가져와서 그 명령 이름을 기반으로 적절한 명령문을 실행합니다. - 각 명령의 처음 두 문자를 읽어 DELETE와 DIRECTORY 명령을 구분할 수 있도록 합니다.
루프 종료	루프를 종료하려면 EXIT 명령을 입력해야 합니다.

명령 프로시저를 더욱 쉽게 이해하고 유지 관리하기 위해서는 프로시저에서 첫 번째 명령부터 마지막 명령까지 실행하도록 명령문을 작성합니다.

2 단계: 변수 할당 및 조건 테스트

변수에 값을 할당하는 방법은 여러 가지가 있습니다. 이 절에서는 **INQUIRE** 명령 사용에 대해 설명합니다. 추가 방법을 보려면 14장을 참조하십시오.

다음 단계를 수행하여 변수에 값을 할당하고 조건을 테스트합니다.

표 13-5

단계	작업
1	INQUIRE 명령을 사용하여 변수에 값을 할당합니다.
2	수행할 작업을 결정합니다.
3	IF 와 THEN 문을 사용하여 조건을 테스트합니다.
4	프로그램 스텝을 작성하여 명령 프로시저에 명령에 대한 자리 표시자로 삽입합니다.
5	필요할 경우 오류 메시지를 작성합니다.

INQUIRE 명령 사용

INQUIRE 명령은 값을 확인하는 프롬프트를 표시하고, 터미널로부터 값을 읽고, 그 값을 기호에 할당합니다.

기본적으로 INQUIRE 명령은 다음을 수행합니다.

- 응답을 대문자로 변환합니다.
- 여러 개의 공백과 탭을 단일 공백으로 바꿉니다.
- 선행 공백과 후행 공백을 제거합니다.
- 응답에 기호나 렉시칼 함수가 들어 있으면 어포스트로피 대체를 수행합니다.

다음 명령행은 CLEANUP.COM에서 사용자에게 명령 이름을 입력하라는 프롬프트를 나타내기 위해 사용됩니다. INQUIRE 명령은 COMMAND 기호에 입력된 값과 일치합니다.

```
$ INQUIRE COMMAND-  
"Enter command (DELETE, DIRECTORY, PRINT, PURGE, TYPE) "
```

리터럴 문자 보존

INQUIRE 명령을 사용할 때 소문자, 여러 개의 공백 및 탭을 유지하려면 사용자의 응답을 인용 부호(" ")로 묶습니다. 응답에 인용 부호를 포함하려면 인용된 텍스트를 인용 부호로 묶습니다(" " 텍스트 " ").

IF와 THEN을 사용하여 조건 테스트

INQUIRE 명령에서 변수를 입력하라는 프롬프트가 표시된 후에는 명령 프로시저에 수행할 작업을 결정하는 명령문이 포함되어야 합니다. 예를 들어, 실행할 명령을 결정하려면 사용자가 입력한 명령을 가능한 각 명령에 대해 확인하는 명령문을 명령 프로시저에 포함해야 합니다.

조건이 참인지 확인하려면 IF와 THEN 명령을 사용합니다. 다음 표에서는 CLEANUP.COM에서 확인해야 할 사항들을 보여 줍니다.

표 13-6

조건	수행할 작업
일치가 있을 경우,	명령을 실행합니다.
일치가 없을 경우,	다음 명령을 계속합니다.

표 13-6 (계속)

조건	수행할 작업
유효한 명령을 모두 확인한 후 일치 가 없을 경우,	오류 메시지를 출력합니다.

프로그램 스텝 작성

프로그램 스텝은 디자인을 테스트하는 동안 프로시저에서 사용하는 코드의 임시 섹션입니다. 대체로 프로그램 스텝은 바꿀 함수를 나타내는 메시지를 출력합니다. 전체 디자인이 제대로 작동한 후 각 스텝을 정확한 코딩으로 바꿉니다.

보기: 변수 할당 및 조건 테스트

다음은 변수를 할당하고 조건을 테스트하는 방법을 보여 주는 보기입니다.

```
$ INQUIRE COMMAND-  
"Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"  
$ IF COMMAND .EQS. "EXIT" THEN EXIT  
$!  
$! Execute if user entered DELETE  
$ DELETE:  
$     IF COMMAND .NES "DELETE" THEN GOTO DIRECTORY      [1] [2]  
$     WRITE SYS$OUTPUT "This is the DELETE section."    [3]  
$! Execute if user entered DIRECTORY  
$ DIRECTORY:                                             [4]  
$     IF COMMAND .NES "DIRECTORY" THEN GOTO PRINT  
$     WRITE SYS$OUTPUT "This is the DIRECTORY section."  
$     .  
$     .  
$     .  
$! Execute if user entered TYPE  
$ TYPE:  
$     IF COMMAND .NES "TYPE" THEN GOTO ERROR            [5]  
$     WRITE SYS$OUTPUT "This is the TYPE section."  
$!  
$ ERROR:  
$     WRITE SYS$OUTPUT "You have entered an invalid command." [6]  
$!  
$ EXIT
```

보기를 검사할 때는 다음 사항에 주의하십시오.

- [1] 이 IF 문은 사용자가 입력한 명령(COMMAND)이 “DELETE”와 같은지 여부를 테스트합니다. COMMAND가 DELETE와 같으면 명령 프로시저에서 다음 명령을 실행합니다.
- [2] 이 명령문에는 GOTO 명령도 포함됩니다. GOTO 명령은 실행의 흐름을 프로시저의 레이블로 변경하기 위해 사용됩니다. 이 경우, COMMAND가 DELETE와 같지 않으면 프로시저는 DIRECTORY 레이블로 이동합니다.
- [3] 이 명령문은 프로그램 스텝입니다. 명령 프로시저의 논리가 테스트된 후, 이 행은 DELETE 작업에 필요한 실제 명령으로 바뀝니다.
- [4] 이것은 DIRECTORY 서브 루틴에 대한 레이블입니다. 각 명령 블록을 식별하는 레이블은 옵션 목록의 명령들과 동일합니다. 따라서 GOTO 명령문에서 COMMAND 기호(사용자의 요청과 일치함)를 사용할 수 있습니다.
- [5] 이 IF 명령문은 “TYPE” 명령을 입력했는지 여부를 테스트합니다. “TYPE”이 입력되었으면 프로시저에서 “This is the TYPE section”을 출력합니다. 그러나 이 명령이 마지막으로 테스트할 명령이므로, 입력한 명령이 “TYPE”이 아니면 프로그램에서 오류 메시지가 표시됩니다.
- [6] 모든 명령이 테스트되고 유효한 명령 이름이 없을 경우에는 프로그램에서 “You have entered an invalid command”를 출력합니다.

3 단계: 루프 추가

루프는 조건이 만족될 때까지 반복적으로 실행되는 명령문 그룹입니다. 반복은 다음과 같이 작동합니다.

1. 사용자 입력으로부터 값을 받습니다.
2. 명령을 처리합니다.
3. 사용자가 명령 프로시저를 종료할 때까지 프로세스를 반복합니다.

루프를 작성하려면 다음 절차를 수행하십시오.

표 13-7

단계	작업
1	레이블을 사용하여 루프를 시작합니다.
2	변수를 테스트하여 루프에서 명령을 실행해야 하는지 확인합니다.
3	루프를 실행할 필요가 없으면 루프의 끝으로 이동합니다.
4	루프를 실행해야 하면 루프의 본문에 있는 명령을 수행한 다음 루프의 시작 부분으로 돌아갑니다.
5	루프를 끝냅니다.

다음은 **CLEANUP.COM** 명령 프로시저에서 루프의 사용을 보여 주는 보기입니다.

```
$ GET_COM_LOOP:
$   INQUIRE COMMAND-
$   "Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE) "
$   IF COMMAND .EQS. "EXIT" THEN GOTO END_LOOP
$!
$! Execute if user entered DELETE
$ DELETE:
$   IF COMMAND .NES. "DELETE" THEN GOTO DIRECTORY
$   WRITE SYS$OUTPUT "This is the DELETE section."
$   GOTO GET_COM_LOOP
.
.
.
$ END_LOOP:
$   WRITE SYS$OUTPUT "Directory 'F$DIRECTORY()' has been cleaned"
$ EXIT
```

명령이 일단 실행되면 사용자가 **EXIT** 명령을 입력할 때까지 **GET_COM_LOOP** 레이블로 제어가 다시 넘어갑니다. **EXIT** 명령이 입력되면 프로시저에서 디렉토리가 지워졌다는 메시지를 출력합니다.

4 단계: 명령 프로시저 종료

명령 프로시저를 종료하려면 다음 절차를 수행합니다.

표 13-8

단계	작업
1	명령 프로시저를 종료하거나 끝내야 할 수 있는 경우를 결정합니다.
2	EXIT 또는 STOP 명령을 적절히 넣습니다.

EXIT 명령 사용

EXIT 명령을 명령 프로시저에 넣어 다음을 수행할 수 있습니다.

- 프로시저에서 특정 행이 실행되지 않도록 합니다.
- 두 개 이상의 실행 경로가 있는 프로시저를 종료합니다.
- 명령 프로시저를 종료합니다.

다음은 EXIT 명령을 사용하여 프로시저 끝에 있는 오류 처리 루틴이 실행되지 않도록 하는 보기입니다.

```
.
.
.
$ EXIT ! End of normal execution path
$ ERROR_ROUTINE
.
.
.
```

다음은 EXIT 명령을 사용하여 두 개 이상의 실행 경로가 있는 프로시저를 종료하는 보기입니다.

```
$ START:
$   IF P1 .EQS. "TAPE" .OR. P1 .EQS. "DISK" THEN GOTO 'P1'
$   INQUIRE P1 "Enter device (TAPE or DISK)"
$   GOTO START
$ TAPE: !Process tape files
.
.
.
$   EXIT $ DISK: ! Process disk files
.
.
.
$   EXIT
```

각 레이블(TAPE와 DISK) 다음에 오는 명령은 프로시저를 통해 서로 다른 경로를 제공합니다. DISK 레이블 앞의 EXIT 명령은 프로시저가 해당 레이블로 명시적으로 분기되지 않는 한, DISK 레이블 다음에 오는 명령이 실행되지 않도록 합니다.

프로시저의 파일 끝에서는 암시적 EXIT 명령이 야기되므로 프로시저의 끝에서는 EXIT 명령이 필요 없습니다. 그러나 EXIT 명령을 사용하는 것이 좋습니다.

STOP 명령 사용

명령 프로시저에서 STOP 명령을 사용하여 서버 오류가 발생할 경우 프로시저가 종료되도록 할 수 있습니다. 대화형으로 실행되는 명령 프로시저에 STOP 명령이 있으면 DCL 수준으로 제어가 반환됩니다. 명령 프로시저가 일괄 처리 모드로 실행될 경우에는 일괄 처리 작업이 종료됩니다.

다음 명령은 심각한 오류가 발생할 경우 프로시저가 중지되도록 합니다.

```
$ ON SEVERE_ERROR THEN STOP
```

5 단계: 프로그램 논리 테스트 및 디버그

프로그램 스텝을 사용하여 코드를 작성한 경우에는 명령 프로시저의 전체 논리를 테스트해야 합니다. 가능한 모든 실행 경로를 테스트해야 합니다.

다음 절차를 수행하여 명령 프로시저를 테스트하고 디버그합니다.

표 13-9

단계	작업
1	명령 프로시저에 유효한 각 명령을 입력하여 프로그램 논리를 테스트합니다.
2	유효하지 않은 명령을 입력하여 프로그램 논리를 계속해서 테스트합니다.
3	EXIT 명령을 사용하여 명령 프로시저를 종료함으로써 프로그램 논리 테스트를 마칩니다.
4	필요할 경우, SET VERIFY, SET PREFIX 또는 SHOW SYMBOL 명령을 사용하여 프로그램을 디버그합니다.

다음 보기에서는 가능한 모든 명령, 유효하지 않은 명령을 입력 및 실행한 다음 종료하여 명령 프로시저를 테스트하는 방법을 보여 줍니다.

```
$ @CLEANUP
Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): DELETE
This is the DELETE section.
Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): DIRECTORY
This is the DIRECTORY section.
.
.
.
Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): PRINF
You have entered an invalid command.
Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): EXIT
$
```

명령 프로시저 디버깅

다음 명령을 사용하여 명령 프로시저의 디버그를 도울 수 있습니다.

- **SET VERIFY**

각 행이 실행되기 전에 해당 행을 표시합니다. 확인 설정과 함께 오류가 발생할 때는 오류와 그 오류를 생성한 해당 행이 표시됩니다. **SET VERIFY** 명령과 함께 키워드를 사용하여 확인할 명령행이나 데이터 행만 나타낼 수 있습니다.

SET VERIFY 명령은 로그아웃하거나, **SET NOVERIFY** 명령을 입력하거나, **F\$VERIFY** 렉시칼 함수를 사용하여 확인 설정을 변경할 때까지 유지됩니다. 확인 설정 변경에 대한 자세한 내용은 15장에 있습니다.

- **SET PREFIX**

확인이 적용되어 있을 때는 **DCL** 명령 **SET PREFIX**를 사용하여 각 명령행이 실행된 시간을 해당 명령행 앞에 붙여서 프로시저 로그 파일에 **time-stamp**를 지정할 수도 있습니다.

- **SHOW SYMBOL**

SHOW SYMBOL 명령을 사용하여 프로시저에서 기호가 정의되는 방식을 확인할 수 있습니다.

보기: SET VERIFY 명령을 사용하여 디버깅

다음 보기에서는 **END_LOP** 레이블의 철자가 틀리게 되어 있습니다. 확인이 설정되어 있기 때문에 오류의 위치를 정확하게 볼 수 있습니다.

```
$ SET VERIFY
$ @CLEAN
$ GET_COM_LOOP:
$   INQUIRE COMMAND -
"Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"
  Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): EXIT
$   IF COMMAND .EQS. "EXIT" THEN GOTO END_LOP
  %DCL-W-USGOTO, target of GOTO not found -
check spelling and presence of label
```

오류를 수정하려면 해당 레이블을 **END_LOOP**로 변경합니다.

보기: SET PREFIX 명령을 사용하여 디버깅

다음 보기에서는 **time-stamp**의 사용을 보여 줍니다.

```
$ SET VERIFY
$ @TEST
$ SET DEFAULT SYS$LOGIN
$ SHOW DEFAULT
USER$: [SMYTHE]
$ SET PREFIX " (!5%T) "
$ @TEST
(17:52) $ SET DEFAULT SYS$LOGIN
(17:52) $ SHOW DEFAULT
USER$: [SMYTHE]
```

보기: SHOW SYMBOL 명령을 사용하여 디버깅

다음 보기에서는 **SHOW SYMBOL** 명령을 사용하여 **COMMAND** 기호가 어떻게 정의되었는지 확인하는 방법을 보여 줍니다.

```
$ SET VERIFY
$ @CLEAN
$ GET_COM_LOOP:
$   INQUIRE COMMAND -
"ENTER COMMAND (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"
  ENTER COMMAND (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE): EXIT
$ SHOW SYMBOL COMMAND
COMMAND = "EXIT"
$   IF COMMAND .EQS. "exit" THEN GOTO END_LOOP
.
.
.
```

SHOW SYMBOL 명령은 COMMAND 기호에 “EXIT” 값이 있음을 보여 줍니다. INQUIRE 명령은 입력 내용을 대문자로 자동 변환하고, 명령을 테스트하는 IF 문은 “exit” 문자열에서 소문자를 사용하므로, DCL은 문자열이 서로 다른 것으로 판정합니다. 오류를 수정하려면 IF 문의 인용된 문자열이 대문자로 쓰이도록 해야 합니다. 문자열의 나머지 부분에는 대문자와 소문자 중 어느 것이든 사용할 수 있습니다.

실행 중 확인 활성화

확인을 활성화기 위해 실행 중인 명령 프로시저를 중단할 수도 있습니다. 명령 프로시저에 SET VERIFY 명령 또는 Ctrl/Y 키 시퀀스가 포함되어 있지 않은 경우에 한해 다음 단계를 수행하여 확인을 활성화할 수 있습니다.

표 13-10

단계	작업
1	Ctrl/Y를 눌러 실행을 중단합니다.
2	SET VERIFY 명령을 입력합니다.
3	CONTINUE 명령을 입력하여 명령 프로시저의 실행을 계속합니다(확인이 설정된 상태).

6 단계: 정리 작업 추가

일반적으로 명령 프로시저를 실행함으로 인해 사용자의 프로세스 상태가 변경되어서는 안 됩니다. 따라서 명령 프로시저에는 프로세스를 원래 상태로 되돌리는 명령 집합이 포함되어야 합니다. 이 명령 집합은 대체로 “CLEAN_UP”이라는 레이블이 지정된 서브 루틴의 일부입니다. 일반적인 정리 작업에는 파일 닫기와 기본 장치 및 디렉토리의 재설정 포함됩니다. 다음 절차를 수행하여 명령 프로시저에 정리 작업을 추가합니다.

표 13-11

단계	작업
1	정리 서브 루틴 앞에 CLEAN_UP과 같은 레이블을 지정합니다.
2	F\$GETJPI 렉시칼 함수를 사용하여 열려 있는 파일을 모두 테스트합니다.
3	DELETE 또는 PURGE 명령을 사용하여 임시 파일이나 관계 없는 파일을 모두 삭제합니다.

표 13-11

(계속)

단계	작업
4	장치나 디렉토리 같은 기본값을 변경한 경우에는 SET DEFAULT 명령을 사용하여 원래 상태로 복원합니다.
5	ON CONTROL_Y 문을 포함하여 정리 작업이 수행되도록 합니다.

파일 닫기

열려 있는 파일이 있으면 프로시저가 종료되기 전에 닫아야 합니다. **F\$GETJPI** 렉시칼 함수를 사용하여 프로세스에 대한 나머지 열려 있는 파일 할당량(**FILCNT**)을 검사할 수 있습니다. **FILCNT**가 명령 프로시저의 시작 부분과 끝 부분에서 동일하면 열려 있는 파일이 없는 것입니다.

다음은 사용자에게 열려 있는 파일이 한 개 있다는 경고를 보낼 때 사용하는 명령입니다.

```
$ FIL_COUNT = F$GETJPI ("", "FILCNT")
.
.
.
$ IF FILCNT .NE. F$GETJPI ("", "FILCNT") THEN-
WRITE SYS$OUTPUT "WARNING -- file left open)
```

임시 파일 또는 관계없는 파일 삭제

임시 파일을 만든 경우에는 그 파일을 삭제합니다. 일반적으로 파일을 업데이트했을 때는 해당 파일을 소거해야 이전 복사본을 삭제할 수 있습니다. 사용자가 자신이 만든 파일을 삭제하기 전에 확실히 삭제할 것인지 확인합니다. 예를 들어, 중요한 데이터가 들어 있는 파일을 업데이트한 경우에는 소거 작업을 선택 사항으로 만들고 싶을 경우도 있습니다.

기본 장치와 디렉토리 중 하나 또는 둘 다를 변경할 경우, 명령 프로시저가 종료되기 전에 원래 기본값을 다시 설정합니다. 원래 기본 디렉토리의 이름을 저장하려면

F\$ENVIRONMENT 렉시칼 함수의 **DEFAULT** 키워드를 사용합니다. 명령 프로시저의 끝에서, 저장된 장치와 디렉토리를 복원하는 **SET DEFAULT** 명령을 포함합니다.

다음 보기에 나오는 명령행에서는 장치와 디렉토리 기본값을 저장 및 복원합니다.

```
$ SAV_DEFAULT = F$ENVIRONMENT ("DEFAULT")
.
.
.
$ SET DEFAULT 'SAV_DEFAULT'
```

일반적으로 변경되는 프로세스 특성

다음 표에서는 일반적으로 변경되는 그 밖의 프로세스 특성, 그러한 특성을 저장할 때 사용하는 렉시칼 함수 및 그러한 특성을 복원할 때 사용하는 렉시칼 함수 또는 명령을 나열합니다.

표 13-12

특성	저장에 사용하는 렉시칼 함수	복원에 사용하는 렉시칼 함수
DCL 프롬프트	F\$ENVIRONMENT	SET PROMPT
기본 보호	F\$ENVIRONMENT	SET PROTECTION/DEFAULT
권한	F\$SETPRV	F\$SETPRV 또는 SET PROCESS/PRIVILEGES
제어 문자	F\$ENVIRONMENT	SET CONTROL
확인	F\$VERIFY	F\$VERIFY
메시지 형식	F\$ENVIRONMENT	SET MESSAGE
키 상태	F\$ENVIRONMENT	SET KEY

이러한 렉시칼 함수에 대한 전체 설명을 보려면 *OpenVMS DCL Dictionary*를 참조하십시오.

정리 작업의 수행 보장

명령 프로시저가 중단되더라도 정리 작업이 수행되도록 하려면 명령 프로시저에서 각 명령 수준을 다음 명령문으로 시작합니다.

```
$ ON CONTROL_Y THEN GOTO CLEANUP
```

ON CONTROL_Y 명령 사용에 대한 추가 정보를 보려면 14장을 참조하십시오.

7 단계: 명령 프로시저 완료

일반 디자인이 제대로 작동할 때는 다음 단계를 수행하여 명령 프로시저를 완료합니다.

표 13-13

단계	작업
1	명령 프로시저의 첫 번째 프로그램 스텝을 명령으로 대체합니다.
2	명령 프로시저를 테스트하여 새 명령이 제대로 작동하는지 확인합니다.
3	필요할 경우 명령 프로시저를 디버그합니다.
4	첫 번째 프로그램 스텝이 작동하면 모든 프로그램 스텝이 바뀔 때까지 계속해서 다음 프로그램 스텝으로 이동합니다.

보기: 프로그램 스텝을 명령으로 바꾸기

다음 보기에서는 CLEANUP.COM의 TYPE 섹션에 대한 코드를 보여 줍니다.

```
$! Execute if user entered TYPE
$! TYPE:
$      IF COMMAND .NES. "TYPE" THEN GOTO ERROR
$      INQUIRE FILE "File to type"
$      TYPE 'FILE'
$      GOTO GET_COM_LOOP
```

다음은 기존 코드를 바꿉니다.

```
$ WRITE SYS$OUTPUT "This is the TYPE section."
```

보기: CLEANUP.COM 명령 프로시저

다음은 완료된 CLEANUP.COM 명령 프로시저의 보기입니다.

```
$ GET_COM_LOOP:
$      INQUIRE COMMAND -
"Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"
$      IF COMMAND .EQS. 'EXIT' THEN GOTO END_LOOP
$!
$!Execute if user entered DELETE
$ DELETE:
$      IF COMMAND .NES. "DELETE" THEN GOTO DIRECTORY
$      INQUIRE FILE "File to delete? "
$      DELETE 'FILE'
$      GOTO GET_COM_LOOP
$!
$!Execute if user entered DIRECTORY
$ DIRECTORY:
$      IF COMMAND .NES. "DIRECTORY" THEN GOTO PRINT
$      DIRECTORY
```



```
$          GOTO GET_COM_LOOP
$!
$!Execute if user entered PRINT
$ PRINT:
$          IF COMMAND .NES. "PRINT" THEN GOTO PURGE
$          INQUIRE FILE "File to print? "
$          PRINT SYS$OUTPUT 'FILE'
$          GOTO GET_COM_LOOP
$!
$!Execute if user entered PURGE
$ PURGE:
$          IF COMMAND .NES. "PURGE" THEN GOTO TYPE
$          PURGE
$          GOTO GET_COM_LOOP
$!
$!Execute if user entered TYPE
$ TYPE:
$          IF COMMAND .NES. "TYPE" THEN GOTO ERROR
$          INQUIRE FILE "File to type"
$          TYPE 'FILE'
$          GOTO GET_COM_LOOP
$!
$ ERROR:
$          WRITE SYS$OUTPUT "You entered an invalid command."
$          GOTO GET_COM_LOOP
$!
$ END_LOOP:
$ WRITE SYS$OUTPUT "Directory 'F$DIRECTORY()' has been cleaned."
$
$ EXIT
```

명령 프로시저 실행

명령 프로시저가 실행되기 위해서는 사용자가 해당 명령 프로시저를 **실행**해야 합니다. 다음과 같은 방법으로 명령 프로시저를 실행할 수 있습니다.

- 다른 명령 프로시저에서 명령 프로시저 실행
- 원격 노드에서 명령 프로시저 실행
- DCL 한정자 또는 매개 변수를 사용하여 명령 프로시저 실행
- 대화형으로 명령 프로시저 실행
- 일괄 처리 작업으로 명령 프로시저 실행
- 디스크 및 테이프 볼륨에서 명령 프로시저 실행

다음 절에서는 이러한 방법들을 각각 설명합니다.

다른 명령 프로시저에서 명령 프로시저 실행

실행 프로시저(@) 명령을 포함하여 명령 프로시저에서 다른 명령 프로시저를 실행할 수 있습니다.

다음 명령 프로시저 WRITEDATE.COM에서는 명령 프로시저 GETDATE.COM을 호출합니다.

```
$! WRITEDATE.COM
$!
$ INQUIRE TIME "What is the current time in hh:mm format?"
$ @GETDATE [JONES.COM]GETDATE.COM
```

원격 노드에서 명령 프로시저 실행

TYPE 명령을 사용하여 원격 노드에서 다른 계정의 최상위 디렉토리에 있는 명령 프로시저를 실행할 수 있습니다. 다음과 같은 명령 프로시저를 실행할 수 있습니다.

- 클러스터 전체에 제공되지 않는 로컬 OpenVMS Cluster 시스템에서 서비스 상태를 표시하는 명령 프로시저
- 원격 노드에 로그인한 사용자를 나열합니다.

TYPE 명령 다음에 액세스 제어 문자열을 입력합니다. 다음 형식을 사용합니다.

```
$ TYPE nodename"username password"::"TASK=command_procedure"
```

username과 **password** 변수는 원격 노드의 계정에 대한 사용자 이름과 암호입니다.

다음 명령 프로시저에서는 명령 프로시저가 위치한 원격 노드에 로그인한 사용자를 표시합니다.

```
$!SHOWUSERS.COM
$!
$ IF F$MODE() .EQS. "NETWORK" THEN DEFINE/USER SYS$OUTPUT SYS$NET
$ SHOW USERS
```

다음 보기에서는 **SHOWUSERS.COM**이 **ORIOLE** 노드의 **BIRD** 계정의 최상위 디렉토리에 있으며 암호는 **BOULDER**입니다. **SHOWUSERS.COM**은 원격 노드 **ORIOLE**에서 **DCL** 명령 **SHOW USERS**를 실행합니다. **TYPE** 명령은 로컬 노드, 즉 **TYPE** 명령을 입력하는 터미널에서 **SHOWUSERS.COM**의 출력을 표시합니다.

```
$ TYPE ORIOLE"BIRD BOULDER"::"TASK=SHOWUSERS"
```

```
OpenVMS User Processes at 11-DEC-1999 17:20:13.30
Total number of users = 4, number of processes = 4
```

Username	Node	Interactive	Subprocess	Batch
FLICKER	AUTOMA	2	1	
ROBIN	FABLES	1	2	1
DOVE	MURMUR	1		
DUCK	FABLES	1	1	

보안 참고

액세스 제어 문자열과 함께 **TYPE** 명령을 사용할 경우 터미널에 암호가 표시됩니다. 적절한 보안 주의 사항을 따릅니다.

DCL 한정자 또는 매개 변수를 사용하여 명령 프로시저 실행

DCL 명령 매개 변수나 한정자를 지정하는 명령 프로시저를 만들 수 있습니다. 이러한 유형의 명령 프로시저는 한 개 이상의 명령과 함께 자주 사용하는 매개 변수 또는 한정자 집합이 있을 경우에 유용합니다.

한정자나 매개 변수를 일반적으로 지정하는 명령행에서 실행 프로시저 명령(@)을 입력합니다.

다음 명령 프로시저는 **LINK** 명령에 대한 한정자 집합을 입력할 때 사용할 수 있습니다.

```
$! This command procedure contains command  
$! qualifiers for the LINK command.  
$!  
  
/DEBUG/SYMBOL_TABLE/MAP/FULL/CROSS_REFERENCE
```

다음 명령행은 **DEFLINK.COM**에서 지정된 한정자를 사용하여 **SYNAPSE.OBJ**라는 객체를 연결합니다.

```
$ LINK SYNAPSE@DEFLINK
```

다음 명령 프로시저는 **CHAP1.TXT**, **CHAP2.TXT**, **CHAP3.TXT** 매개 변수를 **DCL** 명령과 함께 입력할 때 사용할 수 있습니다.

```
$! PARAM.COM  
$! This command procedure contains a list of  
$! parameters that can be used with commands.  
$!  
CHAP1, CHAP2, CHAP3
```

다음 명령행은 매개 변수 목록 대신 **PARAM** 명령 프로시저를 지정합니다. 다음 보기에서는 **PARAM.COM**에 나열된 파일 이름이 매개 변수입니다.

```
$ DIRECTORY/SIZE @PARAM
```

주

실행 프로시저 명령(@)을 사용할 때는 지정된 전체 파일이 **DCL**에 의한 명령 입력으로 처리됩니다.

제한 사항

명령 프로시저 실행 시에는 다음과 같은 제한 사항이 적용됩니다.

- 명령 프로시저가 한정자 이름으로 시작될 경우, 실행 프로시저 명령(@) 앞에 공백을 포함할 수 없습니다.
- 명령 프로시저가 매개 변수로 시작될 경우, 실행 프로시저 명령(@) 앞에 공백이 와야 합니다.

대화형으로 명령 프로시저 실행

명령 프로시저를 대화형으로 실행하려면 실행 프로시저 명령(@) 다음에 명령 프로시저의 파일 지정을 입력합니다.

예를 들어, 다음 명령은 **WORKDISK** 디스크의 **[MAINT.PROCEDURES]** 디렉토리에 있는 **SETD.COM** 프로시저를 실행합니다.

```
$ @WORKDISK:[MAINT.PROCEDURES]SETD Return
```

기호 이름을 정의하여 긴 명령행을 나타낼 수 있습니다. 그런 다음 그 기호를 사용하여 명령 프로시저를 실행할 수 있습니다.

기호를 사용하여 이전 보기에 표시된 명령 프로시저를 실행하려면 로그인 명령 프로시저에 다음 행을 포함합니다.

```
$ SETD == "@WORKDISK:[MAINT.PROCEDURES]SETD"
```

그런 다음 **SETD.COM** 프로시저를 실행하려면 다른 명령과 마찬가지로 기호 이름을 입력합니다.

```
$ SETD Return
```

기본적으로 명령 프로시저를 대화형으로 실행할 때는 운영 체제에서 사용자의 터미널에 출력 내용을 표시합니다. 그러나 실행 명령에 대해 **/OUTPUT** 한정자를 사용하여 출력을 파일로 재지정할 수 있습니다.

명령 프로시저 출력을 파일로 재지정하면 해당 프로시저에서 모든 오류 메시지를 터미널로 보내고 출력을 받을 파일로도 보냅니다.

다음 명령은 **SETD.COM**의 출력을 터미널 대신 **RESULTS.TXT** 파일에 기록합니다.

```
$ @SETD/OUTPUT=RESULTS.TXT
```

명령 프로시저 이름 다음에는 항상 공백 없이 **/OUTPUT** 한정자를 지정합니다. 그렇지 않을 경우, **DCL**은 한정자를 해당 프로시저에 전달되는 매개 변수로 해석합니다.

명령 프로시저를 일괄 처리 작업으로 실행

처리 시간이 오래 걸리는 명령 프로시저(예: 큰 프로그램의 컴파일 또는 어셈블)를 사용할 경우, 이러한 프로시저를 일괄 처리 작업으로 제출하면 터미널을 대화형으로 계속해서 사용할 수 있게 됩니다.

명령 프로시저를 일괄 처리 모드로 실행하려면 **DCL** 명령 **SUBMIT**을 입력하여 명령 프로시저를 일괄 처리 대기열(실행 대기 중인 일괄 처리 작업 목록)로 제출합니다. 작업을 제출하면 기본 일괄 처리 대기열인 **SY\$BATCH**로 작업이 지정되며 이 대기열에서는 실행 대기 중인 작업의 대기열 끝에 작업이 추가됩니다. 앞에 있는 작업들이 완료되면 해당 작업이 실행됩니다. **OpenVMS** 시스템에서는 시스템 관리자가 일괄 처리 대기열을 만들 때 동시에 실행될 수 있는 일괄 처리 작업의 개수가 지정됩니다.

다음 보기에서는 **JOB1.COM**이라는 명령 프로시저를 실행하는 방법을 보여 줍니다. **SUBMIT** 명령은 기본 파일 유형인 **.COM**을 사용하므로, 명령 프로시저의 파일 유형이 **.COM**일 경우 파일 유형을 입력하지 않아도 됩니다.

```
$ SUBMIT
```

```
JOB1 Job JOB1 (queue SY$BATCH, entry 651, started on SY$BATCH) )
```

원격 일괄 처리 작업

시스템이 네트워크의 일부일 경우, 명령 프로시저를 원격 노드의 일괄 처리 작업으로 제출할 수 있습니다. 명령 프로시저 안에서 **DCL** 명령을 사용하여 원격 노드의 파일을 열고 닫을 수 있으며 로컬 파일에 대해 동일한 명령과 한정자를 사용하여 해당 파일의 레코드를 읽고 쓸 수 있습니다.

일괄 처리 작업 다시 시작

기본적으로 작업이 끝나기 전에 시스템에 오류가 발생하면 일괄 처리 작업이 첫 행부터 다시 실행됩니다. 그러나 명령 프로시저에서 다음 기호를 사용하여 다시 시작하는 지점을 다르게 지정할 수 있습니다.

- **\$RESTART**

이러한 실행 전에 일괄 처리 작업이 최소한 한 번 실행되었을 경우 값이 참인 글로벌 기호입니다. **\$RESTART**의 값은 시스템에서 적절히 할당해 주므로, 사용자가 값을 지정하지는 않습니다.

- **BATCH\$RESTART**

SET RESTART_VALUE 명령을 사용하여 값을 지정하는 글로벌 기호입니다.

\$RESTART 및 BATCH\$RESTART 사용

다음 절차에서는 \$RESTART와 BATCH\$RESTART 기호를 사용하는 방법을 설명합니다.

표 13-14

단계	작업
1	프로시저의 가능한 각 시작 지점을 레이블로 시작합니다.
2	각 섹션의 첫 번째 단계로, SET RESTART_VALUE 명령을 사용하여 BATCH\$RESTART의 값을 레이블과 일치시킵니다.
3	프로시저의 시작 부분에서 \$RESTART를 테스트합니다.
4	\$RESTART가 참일 경우, BATCH\$RESTART를 사용하여 GOTO 문을 전송 레이블로 실행합니다.

다음 명령 프로시저에서는 라이브러리에서 많은 모듈을 추출하여, 그 모듈을 연결한 다음, 결과 파일을 정렬합니다.

```
$! SORT_MODULES.COM
!  
$! Set default to the directory containing  
$! the library whose modules are to be sorted  
$ SET DEFAULT WORKDISK:[ACCOUNTS.DATA83]  
$!  
$! Check for restarting  
$ IF $RESTART THEN GOTO "BATCH$RESTART"  
$!  
$ EXTRACT_LIBRARIES:  
$ SET RESTART_VALUE=EXTRACT_LIBRARIES  
.  
.  
.  
$ CONCATENATE_LIBRARIES:  
$ SET RESTART_VALUE=CONCATENATE_LIBRARIES  
.  
.  
.  
$ SORT_FILE:  
$ SET RESTART_VALUE=SORT_FILE  
.  
.  
.  
$ EXIT
```

이 명령 프로시저가 중단되면 BATCH\$RESTART의 값에 따라 CONCATENATE_LIBRARIES 레이블이 지정된 명령문이나 SORT_FILE 레이블이 지정된 명령문에서 파일의 처음부터 명령 프로시저가 다시 실행됩니다. 별도의 많은 모듈을 추출할 경우, 각 추출을 별도의 섹션으로 만들 수 있습니다.

디스크 및 테이프 볼륨에서 명령 프로시저 실행

다음 절에서는 디스크 및 테이프 볼륨에서 명령 프로시저를 실행하는 방법을 설명합니다.

개인 디스크에서 실행

SUBMIT 명령을 사용하여 명령 프로시저를 제출할 때는 할당된 장치의 파일을 액세스할 수 없습니다. 그러나 /SHARE 한정자로 마운트된 개인 디스크에 있는 명령 프로시저는 실행할 수 있습니다.

테이프 볼륨에서 실행

다음과 같은 경우, 테이프 볼륨에 위치한 명령 프로시저를 실행할 수 있습니다.

- 프로시저에서 다른 프로시저를 호출하지 않는 경우
- 프로시저에서 GOTO 명령 전의 프로시저에 있는 레이블을 참조하는 GOTO 명령을 실행하지 않는 경우

이러한 두 가지 조건 중 어느 하나라도 발생하면 다음을 수행하여 명령 프로시저를 실행할 수 있습니다.

표 13-15

단계	작업
1	명령 프로시저를 공유 디스크 볼륨으로 복사합니다.
2	공유 디스크 볼륨의 명령 프로시저를 실행합니다.

명령 프로시저 종료 및 중단

이 절에서는 명령 프로시저 종료 방법을 사용할 때는 명령 수준에 대해 설명합니다.

명령 수준은 DCL 수준 인터프리터에 대한 입력 스트림입니다. 터미널에서 명령을 입력할 때는 명령 수준 0을 입력하는 것입니다. CLEANUP.COM 같은 단순 대화형 명령 프로시저는 명령 수준 1에서 실행됩니다. 프로시저가 종료되고 DCL 프롬프트가 화면에 다시 나타나면 명령 수준 0으로 돌아갑니다.

종료 방법

명령 프로시저가 실행 중일 동안 그 명령 프로시저를 종료하는 방법은 세 가지가 있습니다.

- 명령 프로시저에 **EXIT** 명령을 넣습니다.
- 명령 프로시저에 **STOP** 명령을 넣습니다.
- 프로그램 실행 중에 **Ctrl/Y**를 입력합니다.

EXIT 명령을 사용하여 종료

프로시저 또는 **EXIT** 명령의 끝에서 종료가 이루어지면 더 높은 다음 명령 수준으로 제어가 반환됩니다. **EXIT** 명령의 매개 변수로 값을 지정하여 더 높은 명령 수준으로 상태 값을 반환할 수 있습니다.

SUB라는 명령 프로시저를 DCL 수준에서 호출하고 **SUB**가 서브 루틴 **SUB1**을 호출할 경우, 다음과 같은 상황이 발생합니다.

1. **SUB1**을 종료하면 **SUB1** 호출 다음의 명령행에서 **SUB**로 돌아갑니다.
2. **SUB**를 종료하면 DCL 명령 수준으로 돌아갑니다.

STOP 명령을 사용하여 종료

STOP 명령으로 인한 종료가 이루어지면 **STOP** 명령이 실행되는 명령 수준에 관계 없이 제어가 항상 DCL 명령 수준으로 반환됩니다.

STOP 명령을 일괄 처리 작업으로 실행하면 해당 일괄 처리 작업이 종료됩니다.

Ctrl/Y를 사용하여 종료

Ctrl/Y를 누른 다음 EXIT 또는 STOP 명령을 사용하여 프로시저를 종료하는 방법으로 명령 프로시저를 중단할 수 있습니다. 이 경우, EXIT와 STOP 명령은 사용자를 DCL 수준으로 돌려보냅니다.

다음 보기에서는 Ctrl/Y를 눌러서 TESTALL 프로시저를 중단합니다. EXIT 명령은 프로시저의 처리를 종료하고 사용자를 DCL 수준으로 돌려보냅니다. 프로시저를 중단한 후 STOP 명령을 입력할 수도 있습니다.

```
$ @TESTALL Return
```

```
Ctrl/Y
```

```
$ EXIT Return
```

```
$
```

종료 처리 루틴

명령 프로시저를 중단할 때 중단할 명령(또는 이미지)에서 종료 처리 루틴을 선언하는 경우, EXIT 명령은 이러한 루틴에 제어를 넘깁니다. 그러나 STOP 명령은 이러한 루틴을 실행하지 않습니다.

오류 처리

기본적으로 명령 인터프리터는 어떤 명령으로 인해 오류 또는 심각한 오류가 발생할 경우, **EXIT** 명령을 실행합니다. 이 경우 프로시저는 이전 명령 수준으로 종료됩니다. 그 밖의 심각도 수준(성공, 경고, 정보 등)의 경우, 명령 프로시저는 계속됩니다.

명령 인터프리터에서 오류를 처리하는 방법에 한 가지 예외가 있습니다. 명령 프로시저의 레이블과 존재하지 않는 레이블을 참조할 경우(예: 프로시저에서 레이블로 사용되지 않는 **GOTO ERR1**과 **ERR1** 명령을 포함할 경우), **GOTO** 명령은 경고를 내보내고 명령 프로시저는 종료됩니다.

시스템에서 **EXIT** 명령을 오류 처리 루틴의 일부로 실행할 경우, **\$STATUS**의 값이 한 가지 변경 사항과 함께 다시 이전 명령 수준으로 전달됩니다. 명령 인터프리터에서는 **\$STATUS**의 상위 자릿수를 1로 설정하여 명령 인터프리터에서 해당 상태 값과 관련된 메시지를 다시 표시하지 않도록 합니다.

다음 보기의 **TEST.COM** 명령 프로시저에는 출력 파일 지정에 오류가 들어 있습니다.

```
$ CREATE DUMMY.DAT\
THIS IS A TEST FILE
$ SHOW TIME
```

이 프로시저를 실행하면 **CREATE** 명령이 **\$STATUS**에 오류를 반환하고 해당 메시지를 표시합니다. 그러면 명령 인터프리터에서 **\$STATUS**의 값을 검사하고, 발생한 오류를 확인하고, **EXIT** 명령을 실행하고, **\$STATUS**의 값을 반환합니다. **CREATE** 명령이 오류 메시지를 이미 한 번 표시했기 때문에 프로시저가 종료될 경우 오류 메시지가 다시 표시되지는 않습니다. **DCL** 명령 수준에서는 **\$STATUS**에 오류 메시지가 들어 있지만 상위 자릿수는 1로 설정된 것을 볼 수 있습니다. 예를 들면 다음과 같습니다.

```
$ @TEST
%CREATE-E-OPENOUT, error opening DUMMY.DAT\ as output
-RMS-F-SYN, file specification syntax error
%DCL-W-SKPDAT, image data (records not beginning with "$") ignored
$ SHOW SYMBOL
$STATUS $STATUS = "%X109110A2"
$ WRITE SYS$OUTPUT F$MESSAGE(%X109110A2)
%CREATE-E-OPENOUT, error opening !AS as output
```

기본 오류 동작

다음 표에서는 명령 프로시저가 실행 중인 동안 오류 조건이나 Ctrl/Y 중단이 발생할 경우에 취할 기본 동작을 설명합니다. ON, SET [NO]ON, SET [NO]CONTROL=Y 명령 등을 사용하여 이러한 기본 동작을 무시할 수 있습니다.

표 13-16

중단	기본 동작
오류 또는 심각한 오류	프로시저가 다음 명령 수준으로 종료됩니다.
DCL 명령 수준 또는 명령 수준 1에서의 Ctrl/Y	프로시저가 중단됩니다. 종료를 강요하는 다른 이미지가 없을 경우에는 프로시저가 계속될 수 있습니다.
명령 수준 1 아래 수준에서의 Ctrl/Y	프로시저가 더 높은 다음 명령 수준으로 종료됩니다.

기타 오류 처리 방법

다음 절에서는 그 밖의 오류 처리 방법을 설명합니다.

ON 명령

ON 명령은 특정 심각도 이상의 오류가 발생할 경우 수행할 동작을 지정합니다. 그러한 오류가 발생하면 시스템에서 다음과 같은 동작을 취합니다.

- ON 명령으로 지정된 동작을 수행합니다.
- 지정한 ON 동작의 결과를 나타내도록 \$STATUS와 \$SEVERITY를 설정합니다. 일반적으로 성공으로 설정되어 있습니다.
- 오류 또는 심각한 오류가 발생할 경우 종료하도록 기본 오류 동작을 다시 설정합니다.

ON 명령 동작은 한 번만 실행됩니다. 따라서 ON 명령에 지정된 동작을 명령 프로시저에서 수행하면 기본 오류 동작이 다시 설정됩니다.

ON 명령으로 지정된 동작은 명령이 실행되는 명령 수준 안에서만 적용됩니다. 따라서 다른 프로시저를 호출하는 프로시저에서 ON 명령을 실행할 경우, ON 명령 동작이 중첩된 프로시저에 적용되지 않습니다.

ON 명령의 형식은 다음과 같습니다.

ON condition THEN [\$] command

여기서 “condition”은 다음 키워드 중 하나가 됩니다.

표 13-17

ON 키워드	취해지는 동작
WARNING	경고, 오류 또는 심각한 오류가 발생할 경우 명령 프로시저는 지정된 동작을 수행합니다.
ERROR	오류 또는 심각한 오류가 발생할 경우 명령 프로시저는 지정된 동작을 수행합니다. 경고가 발생할 경우 명령 프로시저는 계속 됩니다.

표 13-17

(계속)

ON 키워드	취해지는 동작
SEVERE_ERROR	심각한(치명적) 오류가 발생할 경우 명령 프로시저는 지정된 동작을 수행합니다. 경고나 오류가 발생할 경우 프로시저는 계속됩니다.

특정 보안 수준에 대해 ON 명령 동작이 설정될 경우, 명령 인터프리터는 심각한도가 동일하거나 더 심각한 오류가 발생할 때 지정된 동작을 수행합니다. 덜 심각한 오류가 발생하면 명령 인터프리터는 파일 처리를 계속합니다.

보기: ON 명령 사용

다음 명령을 사용하여 경고, 오류 또는 심각한 오류가 발생할 경우 프로시저가 종료되도록 기본 오류 처리를 무시할 수 있습니다.

```
$ ON WARNING THEN EXIT
```

보기: 오류 후 계속

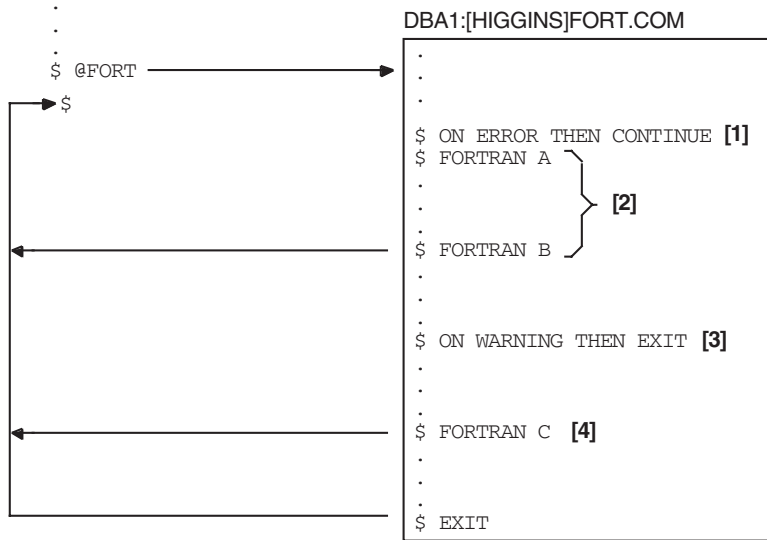
명령 프로시저에 다음 명령이 들어 있으면 오류 또는 심각한 오류가 발생할 때까지 명령 프로시저가 정상적으로 실행됩니다.

```
$ ON ERROR THEN GOTO ERR1
```

해당 오류가 발생하면 프로시저는 ERR1에서 실행을 계속합니다. \$STATUS와 \$SEVERITY는 성공으로 설정되고 기본 오류 동작이 다시 설정됩니다. 다른 ON 또는 SET NOON 명령이 실행되기 전에 두 번째 오류가 발생하면 프로시저가 이전 명령 수준으로 종료됩니다. ON 명령으로 지정된 동작은 명령이 실행되는 명령 수준 안에서만 적용됩니다. 따라서 다른 프로시저를 호출하는 프로시저에서 ON 명령을 실행하면 ON 명령 동작이 중첩된 프로시저에 적용되지 않습니다.

그림 13-1에서는 ON 명령 동작을 보여 줍니다.

그림 13-1 ON 명령 동작



ZK-0826-GE

- [1] 이 ON 명령은 기본 명령 동작(경고 시에는 계속, 오류 또는 심각한 오류 시에는 종료)을 무시합니다. **A.FOR**를 컴파일하는 동안 오류 또는 심각한 오류가 발생하면 명령 프로시저는 다음 명령을 계속합니다.
- [2] 이전의 ON 명령이 적용될 경우, 기본 명령 동작이 다시 설정됩니다. 따라서 **A.FOR**와 **B.FOR**를 모두 컴파일하는 동안 오류 또는 심각한 오류가 발생하면 명령 프로시저가 종료됩니다.
- [3] **C.FOR**를 컴파일하는 동안 경고, 오류 또는 심각한 오류가 발생하면 명령 프로시저가 종료됩니다.
- [4] 명령이 실행되기 전에 명령 프로시저가 종료되지 않으면 명령 동작이 적용됩니다.

부록 B의 보기 명령 프로시저인 **FORTUSER.COM**과 **CALC.COM**에서도 오류 처리 설정을 위한 ON 명령의 사용을 보여 줍니다.

SET NOON 명령 사용

명령 프로시저에서 SET NOON 명령을 사용하여 명령에서 반환된 상태를 명령 인터프리터가 검사하지 못하도록 할 수 있습니다. 이 명령은 ON 명령을 NO 상태로 설정합니다. SET NOON 명령을 사용하면 명령 인터프리터가 계속해서 값을 \$STATUS와 \$SEVERITY에 넣지만 오류 검사는 수행하지 않습니다. SET ON 명령 또는 ON 명령을 사용하여 오류 검사를 복원할 수 있습니다.

프로시저에서 오류 검사를 비활성화할 경우, 명령 또는 프로그램 실행 후의 \$STATUS 값을 명시적으로 확인할 수 있습니다.

다음 보기에서는 SET NOON 명령을 RUN 명령 앞에 사용하여 TESTA 또는 TESTB 프로그램에서 오류 조건을 반환하더라도 명령 프로시저가 계속되도록 합니다. SET ON 명령은 명령 인터프리터에 의한 기본 오류 검사를 복원합니다.

```
$ SET NOON
$ RUN TESTA
$ RUN TESTB
$ SET ON
```

다음 보기에서는 첫 번째 IF 명령이 \$STATUS에 참 값이 있는지(즉, 홀수 값인지) 여부를 확인합니다. 참 값이 있으면 FORTRAN 명령이 성공적이었으므로 LINK 명령이 실행됩니다. LINK 명령이 실행된 후에는 \$STATUS를 다시 테스트합니다. \$STATUS가 홀수이면 RUN 명령이 실행되고 그렇지 않으면 RUN 명령이 실행되지 않습니다. SET ON 명령은 현재 ON 조건 동작을 복원합니다. 즉, SET NOON 명령이 실행되기 전에 어떤 조건이 적용되었는지는 문제되지 않습니다.

```
$ SET NOON
$ FORTRAN MYFILE
$ IF $STATUS THEN LINK MYFILE
$ IF $STATUS THEN RUN MYFILE
$ SET ON
```

SET ON 또는 SET NOON 명령은 현재 명령 수준, 즉, 명령이 실행되는 해당 명령 수준에만 적용됩니다. 다른 명령 프로시저를 호출하는 명령 프로시저에서 SET NOON 명령을 사용할 경우, 중첩된 프로시저 안에서 기본 오류 검사 메커니즘이 적용됩니다. DCL 수준에서 대화형으로 SET NOON을 입력할 때는 아무런 의미가 없습니다.

Ctrl/Y 중단 처리

기본적으로 명령 프로시저를 실행하는 동안 Ctrl/Y를 누르면 명령 인터프리터는 Ctrl/Y 명령 수준을 호출한 특별한 명령 수준에서 명령 입력을 요구하는 프롬프트를 표시합니다.

Ctrl/Y 명령 수준에서 명령 인터프리터 안에서 실행되는 DCL 명령을 입력한 다음 CONTINUE 명령을 사용하여 명령 프로시저의 실행을 다시 시작할 수 있습니다. 또한 명령 프로시저의 실행을 중지시키는 DCL 명령을 입력하여 프로시저를 중지할 수 있습니다.

이 절에서는 명령 프로시저에서 ON 명령을 사용하여 Ctrl/Y 중단을 처리하는 방식을 무시하는 방법에 대해 설명합니다.

명령 프로시저 중지

Ctrl/Y를 눌러서 대화형으로 실행되고 있는 명령 프로시저를 중단할 수 있습니다. Ctrl/Y를 누르면 명령 인터프리터에서 Ctrl/Y 수준이라고 하는 새로운 명령 수준을 설정하고 명령 입력을 요구하는 프롬프트를 표시합니다. 중단이 이루어지는 시기는 실행 중인 명령 또는 프로그램에 따라 다릅니다.

- 명령 인터프리터에 의해 명령이 직접 실행될 경우(예: IF, GOTO 또는 할당문), 명령 인터프리터가 Ctrl/Y 수준에서 명령을 입력하라는 프롬프트를 표시하기 전에 명령 실행이 완료됩니다.
- 명령 또는 프로그램이 별개의 이미지(즉, 명령 인터프리터 외의 다른 이미지)일 경우, 명령이 중단되고 명령 인터프리터는 Ctrl/Y 수준에서 명령을 입력하라는 프롬프트를 표시합니다.

Ctrl/Y 수준에서는 명령 인터프리터가 이전에 설정된 모든 명령 수준에 대한 상태를 저장하여 Ctrl/Y 중단 후 정확한 상태를 복원할 수 있도록 합니다.

프로시저를 중단한 후에는 다음을 수행할 수 있습니다.

- 명령 인터프리터 안에서 실행되는 DCL 명령을 입력합니다.

그러한 명령으로는 SET VERIFY, SHOW TIME, SHOW TRANSLATION, ASSIGN, EXAMINE, DEPOSIT, SPAWN, ATTACH 명령 등이 있습니다. 이러한 명령을 한 개 이상 입력한 후에는 CONTINUE 명령을 사용하여 프로시저의 실행을 계속할 수 있습니다. 명령 인터프리터 안에서 실행되는 명령의 전체 목록을 보려면 575페이지의 “명령 및 이미지에서 출력 재지정”을 참조하십시오.

CONTINUE 명령을 입력하면 중단된 명령이나 프로그램 또는 최근에 완료된 명령의 다음 행에서 명령 프로시저가 실행을 계속합니다.

- 다른 이미지를 실행하는 DCL 명령을 입력합니다.

새 이미지를 호출하는 명령을 입력하면 명령 인터프리터는 명령 수준 0으로 돌아가서 명령을 실행합니다. 이렇게 하면 명령 프로시저의 실행이 종료됩니다. 중단된 이미지에 의해 선언된 모든 종료 처리기는 새 이미지가 시작되기 전에 실행될 수 있습니다.

- EXIT 또는 STOP 명령을 입력하여 명령 프로시저의 실행을 종료합니다.

EXIT 명령을 사용할 경우에는 중단된 이미지에 의해 선언된 종료 처리기가 실행될 수 있습니다. 그러나 STOP 명령은 이러한 루틴을 실행하지 않습니다.

주

Ctrl/Y 다음에 명령 프로시저를 명령 수준에서 명시적으로 또는 ON 루틴의 일부로 종료하지 않을 경우, 다음에 입력하는 명령이 명령 프로시저의 컨텍스트에서 중단됩니다. 예를 들어, 대화형 수준에서 다음 기호를 정의하는 경우를 가정합니다.

```
$ MAIL = "mail/edit=(send,reply,forward) "
```

Ctrl/Y를 입력하여 이 정의가 포함되지 않은 명령 프로시저를 중단한 다음 MAIL 명령을 입력하여 메시지를 보내면 편집기가 자동으로 호출되지 않습니다.

특권 있는 이미지 중지

특권 있는 이미지의 실행을 중단할 경우, 이미지의 컨텍스트를 저장하기 위해서는

CONTINUE, SPAWN 또는 ATTACH 명령만 입력할 수 있습니다. 사용자가 생성했거나 연결한 하위 프로세스 내의 경우를 제외한 그 밖의 명령을 입력할 경우, 특권 있는 이미지는 강제로 종료됩니다.

Ctrl/Y 동작 루틴 설정

다음 절에서는 Ctrl/Y 동작 루틴을 설정하는 방법을 설명합니다.

ON 명령 사용

오류 조건에 대해 취할 동작을 정의하는 ON 명령은 명령 프로시저의 실행 중에 발생하는 Ctrl/Y 중단에 대한 동작 루틴을 정의하는 방법도 제공합니다. 사용자가 지정하는 기본 동작은 기본 Ctrl/Y 동작(즉, Ctrl/Y 명령 수준에서 명령 입력을 요구하는 프롬프트)을 무시합니다. 예를 들면 다음과 같습니다.

```
$ ON CONTROL_Y THEN EXIT
```

프로시저에서 ON 명령을 실행하면 해당 프로시저의 실행 중 이어지는 Ctrl/Y 중단으로 인해 프로시저가 종료됩니다. 이전 명령 수준으로 제어가 넘어갑니다.

Ctrl/Y를 눌러서 ON CONTROL_Y를 사용하는 프로시저를 중단하면 다음 동작이 이루어 집니다.

- 현재 실행 중인 명령이 명령 인터프리터 안에서 실행된 명령일 경우, 해당 명령이 완료 되고 Ctrl/Y 동작이 수행됩니다.
- 현재 명령 또는 프로그램이 명령 인터프리터 외의 다른 이미지에 의해 실행될 경우, 이미지가 강제로 종료되고 Ctrl/Y 동작이 수행됩니다. 그러나 이미지가 종료 처리기를 선언한 경우에는 Ctrl/Y 동작이 수행되기 전에 종료 처리기가 실행됩니다. Ctrl/Y 동작 다음에는 이미지가 계속될 수 없습니다.

Ctrl/Y 입력의 효과

Ctrl/Y의 실행으로 기본 Ctrl/Y 동작(즉, Ctrl/Y 명령 수준에서 명령 입력을 요구하는 프롬프트 표시)이 자동으로 다시 설정되지는 않습니다. Ctrl/Y 동작은 다음 조건들 중 하나가 발생할 때까지 유지됩니다.

- Ctrl/Y를 누르거나, EXIT 또는 STOP 명령을 실행하거나, 기본 오류 조건 처리 동작의 결과로 프로시저가 종료됩니다.
- 다른 ON CONTROL_Y 명령이 실행됩니다.
- 프로시저가 SET NOCONTROL=Y 명령을 실행합니다(547페이지의 “Ctrl/Y 중단의 비활성화 및 활성화” 참조).

Ctrl/Y 동작은 활성화된 각 명령 수준에서 지정할 수 있으며 지정된 명령 수준에만 영향을 미칩니다.

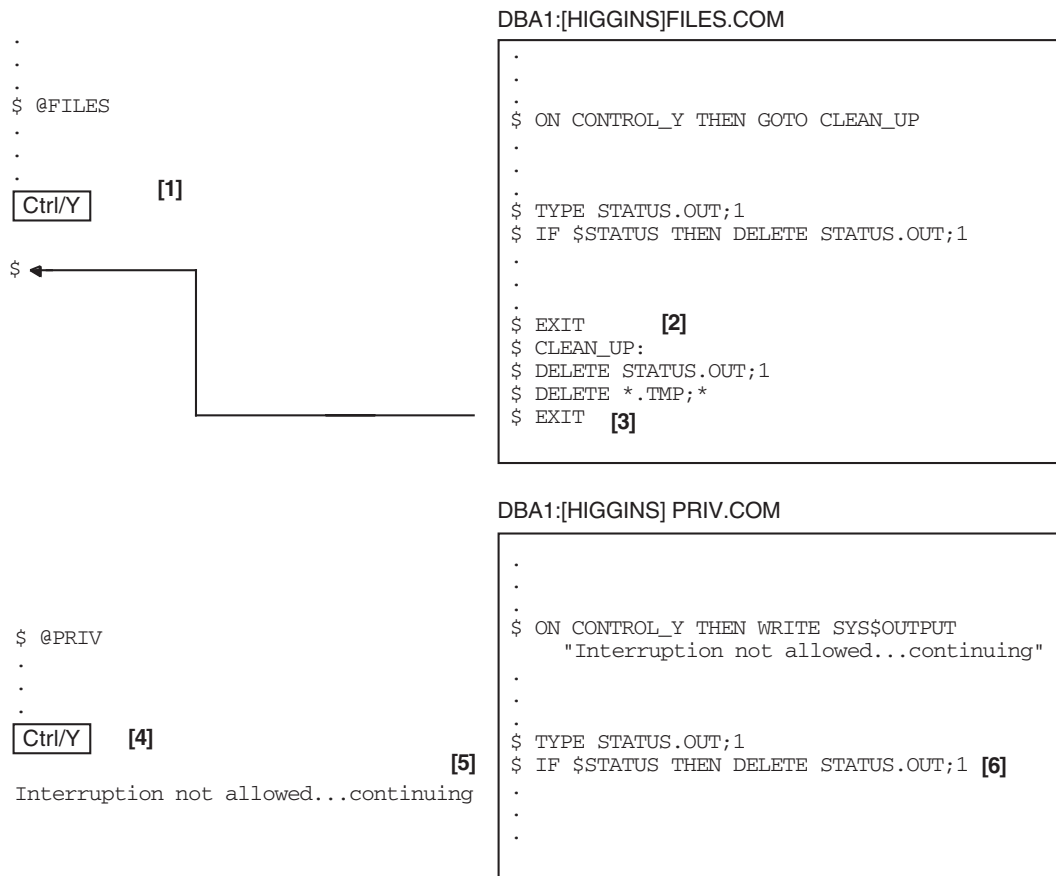
다음 보기에 나오는 명령 프로시저가 실행될 때는 각 Ctrl/Y 중단으로 인해 SHOW TIME 명령이 실행됩니다. 각 SHOW TIME 명령이 실행된 후 프로시저는 중단된 명령의 다음 명령에서 실행을 계속합니다.

```
$ ON CONTROL_Y THEN SHOW TIME
```

그림 13-2에서는 Ctrl/Y 중단 다음의 실행 흐름을 보여 줍니다.

그림 13-2

Ctrl/Y 동작 다음의 실행 흐름

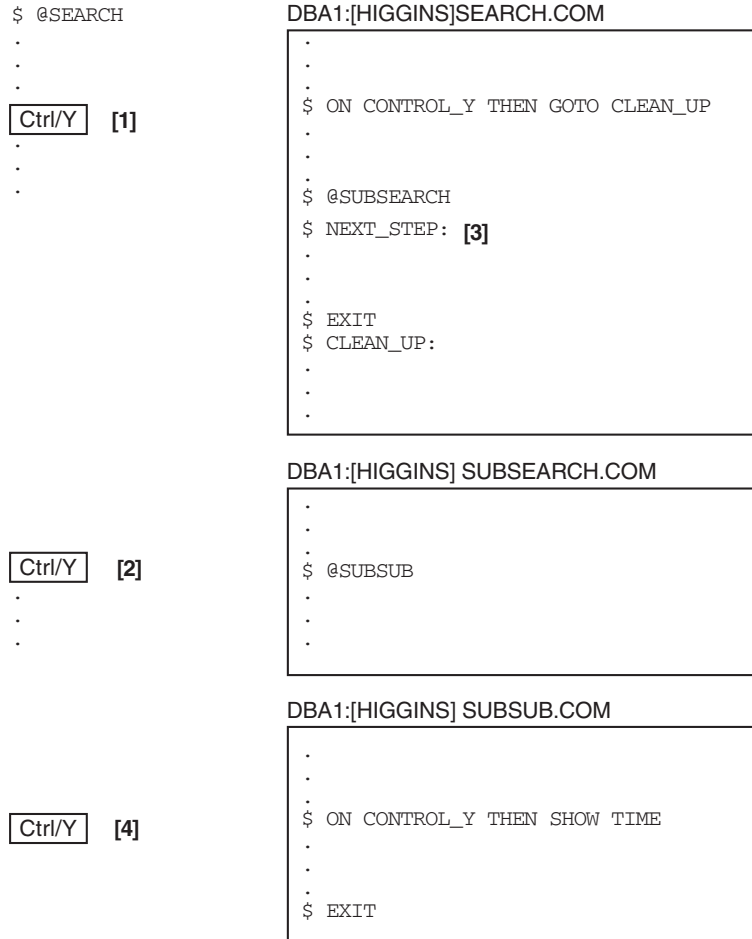


ZK-0827-GE

- [1] Ctrl/Y 중단은 TYPE 명령 실행 중에 발생합니다.
- [2] 그런 다음 CLEAN_UP 레이블로 제어가 넘어갑니다.
- [3] 루틴 실행 후 명령 프로시저가 종료되고 대화형 명령 수준으로 돌아갑니다.
- [4] Ctrl/Y 중단은 TYPE 명령 실행 중에 발생합니다.
- [5] ON 명령에 지정된 WRITE 명령이 실행됩니다.
- [6] 중단된 명령의 다음 명령에서 명령 프로시저의 실행이 계속됩니다.

그림 13-3에서는 중첩된 명령 프로시저를 실행하는 동안 Ctrl/Y를 누를 경우 어떤 상황이 발생하는지 보여 줍니다.

그림 13-3 중첩된 프로시저에서의 Ctrl/Y



ZK-0828-GE

- [1]** SEARCH.COM이 실행되는 동안 Ctrl/Y 중단이 발생하면 CLEAN_UP 레이블로 제어가 넘어갑니다.
- [2]** SUBSEARCH.COM이 실행되는 동안 Ctrl/Y 중단이 발생하면 SEARCH.COM의 NEXT_STEP 레이블로 제어가 넘어갑니다.
- [3]** SUBSEARCH.COM에 지정된 Ctrl/Y 동작이 없기 때문에 프로시저는 Ctrl/Y 중단이 발생한 이전 명령 수준으로 종료됩니다.
- [4]** SUBSUB.COM이 실행되는 동안 Ctrl/Y 중단이 발생하면 SHOW TIME이 실행됩니다.

Ctrl/Y 중단의 비활성화 및 활성화

다음 절에서는 Ctrl/Y 중단을 비활성화하고 활성화하는 방법을 설명합니다.

SET NOCONTROL=Y 사용

SET NOCONTROL=Y 명령은 Ctrl/Y 처리를 비활성화합니다. 즉, 명령 프로시저에서 SET NOCONTROL=Y 명령을 실행할 경우에는 Ctrl/Y를 눌러도 아무 효과가 없습니다.

또한 SET NOCONTROL=Y 명령은 ON CONTROL_Y 명령을 사용하여 설정된 현재의 Ctrl/Y 동작을 취소합니다. 기본 Ctrl/Y 동작을 다시 설정하려면 다음 두 명령을 사용합니다.

```
$ SET NOCONTROL=Y  
$ SET CONTROL=Y
```

SET NOCONTROL=Y 명령은 Ctrl/Y 처리를 비활성화하며 현재의 ON CONTROL_Y 동작을 취소합니다. SET CONTROL=Y 명령은 Ctrl/Y 처리를 활성화합니다. 이 시점에서 기본 동작이 복원됩니다. 즉, 프로시저 실행 중 Ctrl/Y를 누르면 명령 인터프리터는 Ctrl/Y 명령 수준에서 명령을 입력하라는 프롬프트를 표시합니다.

SET NOCONTROL=Y 명령은 모든 명령 수준에서 사용할 수 있습니다. 이 명령은 SET CONTROL=Y 명령이 Ctrl/Y 처리를 다시 활성화할 때까지 모든 명령 수준에 영향을 미칩니다.

SET CONTROL=Y 사용

ON CONTROL_Y 명령은 다른 ON CONTROL_Y 또는 SET NOCONTROL=Y 명령이 실행되거나 명령 프로시저가 종료될 때까지 유지됩니다.

Ctrl/Y가 비활성화될 경우 종료되지 않는 루프를 끝내려면 DCL 명령 STOP을 사용하여 다른 터미널에서 프로세스를 삭제해야 합니다. 기본 Ctrl/Y 동작을 비활성화할 경우에는 될 수 있는 대로 빨리 다시 설정하십시오. 기본 Ctrl/Y 동작을 다시 설정하려면 SET NOCONTROL=Y 명령 다음에 SET CONTROL=Y 명령을 실행합니다.

다음 명령 프로시저에서는 파일이 입력되는 동안 Ctrl/Y를 눌러서 END_TYPE 레이블로 제어를 넘깁니다.

```
.
.
.
$! Type a file
$ IF COMMAND .NES. "TY" THEN GOTO END_TYPE
$ ON CONTROL_Y THEN GOTO END_TYPE
$ TYPE 'FILESPEC'
$END_TYPE:
$!
$!
Reset default
$ SET NOCONTROL=Y
$ SET CONTROL=Y
.
.
.
```

주

ON CONTROL_Y와 SET NOCONTROL=Y 명령은 특수 어플리케이션을 위한 명령입니다. 일반적으로 Ctrl/Y 중단을 비활성화하도록 권장되지는 않습니다. Ctrl/Y가 비활성화될 경우 종료되지 않는 루프를 끝내려면 루프 반복 중인 프로시저가 실행되고 있는 프로세스를 다른 터미널에서 삭제해야 합니다.

조건 코드를 사용하여 명령 프로시저의 오류 감지

명령 프로시저의 각 DCL 명령 실행이 완료되면 명령 인터프리터는 명령이 종료된 이유를 설명하는 조건 코드를 저장합니다. 이 코드는 성공적인 완료를 나타낼 수도 있고, 정보 또는 오류 메시지를 식별할 수도 있습니다.

명령 인터프리터에서는 명령 프로시저의 각 명령을 수행한 후 조건 코드를 검사합니다. 특별한 동작이 요구되는 오류가 발생했으면 시스템에서 그 동작을 수행합니다. 그렇지 않을 경우 프로시저의 다음 명령이 실행됩니다.

조건 코드 표시(\$STATUS)

명령 인터프리터에서는 조건 코드를 32비트 긴 단어로 예약된 글로벌 기호 \$STATUS에 저장합니다. \$STATUS 기호는 다음과 같은 시스템 메시지 코드의 형식을 따릅니다.

- 비트 0-2에는 메시지의 심각도 수준이 포함됩니다.
- 비트 3-15에는 메시지 번호가 포함됩니다.
- 비트 16-27에는 메시지를 생성한 기능에 관련된 번호가 포함됩니다.
- 비트 28-31에는 내부 제어 플래그가 포함됩니다.

명령이 성공적으로 완료되면 \$STATUS는 홀수 값을 갖습니다. 비트 0-2에는 1 또는 3이 포함됩니다. 어떤 유형의 경고나 오류가 발생하든 간에 \$STATUS에는 짝수 값이 포함됩니다. 비트 0-2에는 0, 2 또는 4가 포함됩니다. 명령 인터프리터에서는 \$STATUS의 현재 16진수 값을 유지하고 표시합니다. SHOW SYMBOL \$STATUS 명령을 입력하여 \$STATUS의 ASCII 변환을 표시할 수 있습니다.

다음 보기에서는 파일 이름(%FRED.LIS)이 잘못 입력되었습니다.

```
$ CREATE %FILE.LIS
%CREATE-E-OPENOUT, error opening %FRED.LIS; as output
-RMS-F-WLD, invalid wildcard operation
$ SHOW SYMBOL $STATUS
$STATUS = " %X109110A2"
$ WRITE SYS$OUTPUT F$MESSAGE(%X109110A2)
%CREATE-E-OPENOUT, error opening !AS as output
```

조건 코드와 EXIT 명령

명령 프로시저가 종료되면 명령 인터프리터에서 `$STATUS`의 이전 명령에 대한 조건 코드를 반환합니다. 조건 코드는 최신 명령이 성공적으로 실행되었는지 여부에 대한 정보를 제공합니다.

명령 프로시저에서 **EXIT** 명령을 사용하면 DCL이 `$STATUS`에 할당한 값을 무시하는 값을 지정할 수 있습니다. 상태 코드라고 하는 이 값은 정수 식으로 지정되어야 합니다.

명령 프로시저에 여러 명령 수준을 만들기 위한 중첩된 프로시저가 들어 있을 때는 **EXIT** 명령을 사용하여 기본 조건 코드를 명시적으로 무시하는 값을 반환할 수 있습니다.

다음 두 명령 프로시저를 검사합니다.

```
$! This is file A.COM
$!
$ @B
.
.
.
$! This is file B.COM
$!
$ ON WARNING THEN GOTO ERROR
.
.
.
$ ERROR:
$ EXIT 1
```

B.COM의 **ON** 명령은 **B.COM**이 실행 중일 때 경고, 오류 또는 심각한 오류가 발생할 경우 프로시저가 **ERROR** 레이블로 지정된다는 것을 의미합니다. 여기서 조건 코드는 성공을 나타내는 1로 명시적으로 설정됩니다. 따라서 **B.COM**이 종료되면 오류의 발생 여부에 관계 없이 성공 코드를 **A.COM**으로 다시 전달합니다.

보안 수준 결정

\$STATUS의 최하위 세 비트는 명령을 종료시키는 조건의 심각도를 나타냅니다. 조건 코드의 이 부분은 예약된 글로벌 기호인 \$SEVERITY에 들어 있습니다. \$SEVERITY 기호에는 각각 다음 보안 수준 중 하나를 나타내는 0부터 4까지의 값이 있을 수 있습니다.

표 13-18

값	심각도
0	경고
1	성공
2	오류
3	정보
4	치명적(심각한) 오류

성공과 정보 코드는 홀수 값을 가지며, 경고와 오류 코드는 짝수 값을 갖습니다.

성공적인 완료 테스트

\$SEVERITY 또는 \$STATUS에 대한 논리 테스트를 수행하는 IF 명령을 사용하여 명령의 성공적인 완료를 다음과 같이 테스트할 수 있습니다.

```
$ IF $SEVERITY THEN GOTO OKAY
$ IF $STATUS THEN GOTO OKAY
```

이러한 IF 명령은 \$SEVERITY와 \$STATUS의 값이 참(홀수)일 경우 OKAY 레이블로 분기됩니다. \$SEVERITY와 \$STATUS의 현재 값이 홀수이면 명령 또는 프로그램이 성공적으로 완료된 것입니다. 명령 또는 프로그램이 성공적으로 완료되지 않았으면 \$SEVERITY와 \$STATUS는 짝수입니다. 따라서 IF 식은 거짓입니다.

조건이 참인지를 테스트하는 대신, 거짓인지 여부를 테스트할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ IF .NOT. $STATUS THEN ...
```

명령 인터프리터에서는 조건 코드의 심각도 수준을 사용하여 537페이지의 “기타 오류 처리 방법”에 설명된 대로 ON 명령에 의해 정의된 동작을 수행할지 여부를 결정할 수 있습니다.

\$STATUS를 설정하지 않는 명령 사용

대부분의 DCL 명령은 컴파일을 할 때 상태 값과 오류 메시지를 생성하는 시스템 유틸리티를 호출합니다. 그러나 성공적으로 완료되어도 \$STATUS와 \$SEVERITY 값을 변경하지 않는 명령이 몇 가지 있습니다. 이러한 명령은 다음과 같습니다.

표 13-19

CONTINUE	DECK	DEPOSIT
EOD	EXAMINE	GOTO
IF	RECALL	SET SYMBOL/SCOPE
SHOW STATUS	SHOW SYMBOL	STOP
WAIT		

이러한 명령들 중 하나로 인해 성공하지 못한 상태가 초래되면 조건 코드가 \$STATUS에 넣어지고 심각도 수준이 \$SEVERITY에 넣어집니다.

로그인 명령 프로시저

로그인 명령 프로시저는 사용자가 로그인할 때마다 운영 체제에서 자동으로 실행되는 명령 프로시저입니다. 시스템에서는 사용자가 제출하는 모든 일괄 처리 작업의 시작 부분에서도 이 프로시저를 실행합니다.

로그인 명령 프로시저에는 두 가지 유형이 있습니다.

- 시스템(또는 그룹 정의)
- 개인

시스템 로그인 명령 프로시저

시스템 로그인 명령 프로시저에는 다음과 같은 특성이 있습니다.

- 개인 로그인 명령 프로시저보다 먼저 실행됩니다.
- 시스템 로그인 명령 프로시저가 종료되면 로그인 명령 프로시저로 제어가 넘어갑니다.
- 사용자가 로그인할 때면 항상 특정 명령이 실행되도록 시스템 관리자가 지정할 수 있습니다.

시스템 로그인 명령 프로시저를 설정하려면 시스템 관리자가 해당 로그인 명령 프로시저에 논리 이름 **SYS\$SYLOGIN**을 일치시켜야 합니다. 시스템 관리자는 로그인 명령 프로시저가 모든 시스템 사용자 또는 특정 그룹의 사용자들에 대해 사용되도록 지정할 수 있습니다.

개인 로그인 명령 프로시저

개인 로그인 명령 프로시저를 만들어서 로그인할 때마다 동일한 명령을 실행할 수 있습니다.

로그인 명령 프로시저에 대한 파일 지정은 시스템 관리자가 할당합니다. 대부분의 설치에서 로그인 명령 프로시저는 **LOGIN.COM**이라고 합니다. 따라서 시스템 관리자가 다른 사항을 지시하지 않았다면 로그인 명령 프로시저 이름을 **LOGIN.COM**으로 지정하십시오.

다음은 **LOGIN.COM** 프로시저의 보기입니다.

```
$IF F$MODE() .NES. "INTERACTIVE" THEN EXIT
$SET TERMINAL/INSERT
$DIR :== DIR/DATE/SIZE
$EDIT :== EDIT/EDT
$EXIT
```

전용 계정의 로그인 명령 프로시저

시스템 관리자는 사용자 계정에 대한 **LGICMD** 필드에 특수 명령 프로시저의 이름을 지정하여 **전용 계정**을 설정할 수 있습니다. 전용 계정으로 로그인할 경우, 사용자는 자신의 계정에 대한 명령 프로시저에 지정된 기능만 수행할 수 있습니다. 전체 **DCL** 명령 집합은 사용할 수 없습니다. 전용 계정에 대한 자세한 내용은 *OpenVMS System Manager's Manual*을 참조하십시오.

확장 파일 지정 및 구문 분석 스타일

특정 파일 이름 구문 분석 스타일이 필요한 명령 프로시저에는 스타일을 전환하는 명령이 프로시저 안에 포함될 수 있습니다. 다음 명령 프로시저에서는 현재의 구문 분석 스타일을 저장하고, 구문 분석 스타일을 **TRADITIONAL**로 설정하고(지정되지 않은), 명령을 수행한 다음, 저장된 구문 분석 스타일을 복원합니다.

```
$ original_style= f$getjpi("", "parse_style_perm")
$ SET PROCESS/PARSE_STYLE=TRADITIONAL
.
.
.
$ SET PROCESS/PARSE_STYLE='original_style'
```

첫 번째 명령은 'original_style'을 현재 구문 분석 스타일과 일치시킵니다. 두 번째 명령은 구문 분석 스타일을 **TRADITIONAL**로 설정합니다. 마지막 명령은 구문 분석 스타일을 원래 스타일로 설정합니다.

DCL 명령 매개 변수에 확장 파일 이름 사용

파일 이름을 매개 변수로 사용하는 명령 프로시저는 ODS-5 환경에서 여러 가지 결과를 만들 수 있습니다.

구문 분석 스타일을 **TRADITIONAL**에서 **EXTENDED**로 전환할 수 있습니다. 이 절에서는 구문 분석 스타일을 이렇게 전환하도록 선택할 경우 영향을 받을 수 있는 다음 영역들에 대해 설명합니다.

- 명령 프로시저 파일 지정
- 대/소문자 보존 및 \$FILE
- 앰퍼샌드 및 어포스트로피 대체

구문 분석 스타일 간의 전환에 대한 자세한 내용은 173페이지의 “DCL의 확장 파일 지정 방법 분석 기능 사용”을 참조하십시오.

명령 프로시저 파일 지정

간접적인 명령 프로시저가 사용될 경우, 일부 프로시저 인수를 인용 부호로 묶어야 할 수도 있습니다.

다음 보기에서는 동일한 명령 파일인 **SS.COM** 사용 시 **TRADITIONAL**과 **EXTENDED** 구문 분석 스타일 간의 출력 차이점을 보여 줍니다.

```
$ create ss.com
$ if p1 .nes. "" then write sys$output "p1 = ",p1
$ if p2 .nes. "" then write sys$output "p2 = ",p2
$ if p3 .nes. "" then write sys$output "p3 = ",p3
```

- 구문 분석 스타일을 **TRADITIONAL**로 설정하고 **SS.COM**을 실행하면 다음과 같은 출력이 만들어집니다.

```
$ set process/parse_style=traditional
$ @ss ^ parg2 parg3
p1 = ^
p2 = PARG2
p3 = PARG3
```

악센트 기호(^)가 첫 번째 인수이고(이스케이프 문자 아님), **p2**와 **p3** 프로시저 인수에 대해서는 대/소문자가 보존되지 않는 것에 주의하십시오.

- 구문 분석 스타일을 **EXTENDED**로 설정하면 동일한 명령 프로시저를 실행할 때 다음과 같은 출력이 만들어집니다.

```
$ set process/parse_style=extended
$ @ss ^ parg2 parg3
p1 = ^ PARG2
p2 = PARG3
```

명령 프로시저에서는 공백을 인수 분리자가 아니라 리터럴 문자로 식별하는 이스케이프 문자로 악센트 기호(^)를 인식하며, “^ PARG2”가 첫 번째 인수라는 것에 주의하십시오. 대/소문자는 보존되지 않습니다.

- 악센트 기호(^)에 인용 부호를 추가하면 다음과 같은 결과가 만들어집니다.

```
$ @ss "^" parg2 parg3
p1 = ^
p2 = PARG2
p3 = PARG3
```

악센트 기호(^)는 인용된 문자열 안에 있으므로 이스케이프 문자로 처리되지 않습니다.

- p3 인수에 인용 부호를 추가하면 다음과 같은 결과가 만들어집니다.

```
$ @ss "^" parg2 "parg3"
p1 = ^
p2 = PARG2
p3 = parg3
```

p3 프로시저 인수에 대해서는 대/소문자가 보존됩니다.

- 구문 분석 스타일이 **TRADITIONAL**로 설정될 경우, 다음 명령에서는 악센트 기호(^)와 parg2, parg3 문자열을 프로시저 인수로 처리하며, 명령 프로시저에서는 다음과 같은 결과가 만들어집니다.

```
$ set process/parse_style=traditional
$ @ss^ parg2 parg3
p1 = ^
p2 = PARG2
p3 = PARG3
```

- 구문 분석 스타일이 **EXTENDED**로 설정될 경우, 악센트 기호(^)는 공백을 리터럴 문자로 식별하는 이스케이프 문자로 처리됩니다. DCL은 “SS^_PARG2.COM” 파일을 찾고 다음 보기에 표시된 오류를 만듭니다.

```
$ set process/parse_style=extended
$ @ss^ parg2 parg3
-RMS-E-FNF, file not found
```

대/소문자 보존 및 \$FILE

DCL은 파일 지정의 대/소문자 보존을 시도합니다. CDU(Command Definition Utility)로 정의된 명령에 대해서만 이러한 시도를 할 수 있습니다. DCL은 \$FILE 구문 분석 유형을 사용하여 명령 정의 파일(.CLD)에 정의된 모든 항목에 대한 대/소문자를 보존합니다.

자세한 내용은 *OpenVMS Command Definition, Librarian, and Message Utilities*를 참조하십시오.

앰퍼샌드 및 어포스트로피 대체

어포스트로피 대체와는 반대로, 앰퍼샌드(&) 대체를 사용하면 TRADITIONAL 구문 분석 중에 대/소문자를 보존할 수 있습니다.

다음 TRADITIONAL 구문 분석 보기에서는 문자열의 대/소문자를 변경하는 일련의 명령을 보여 줍니다.

```
$ set process/parse_style=traditional
$ x = "string"
$ define y 'x'
$ sho log y
"Y" = "STRING" (LNM$PROCESS_TABLE)
$ define y &x
%DCL-I-SUPERSEDE, previous value of Y has been superseded
$ sho log y
"Y" = "string" (LNM$PROCESS_TABLE)
```

앰퍼샌드(&) 사용으로 인해 x 변수에 할당된 문자열의 대/소문자가 보존된 것에 주의하십시오.

어포스트로피 대체는 명령행이 대문자로 설정되기 전에 이루어지며, 앰퍼샌드 대체는 명령행이 대문자로 설정된 후에 이루어집니다.

다음 확장 구문 분석 보기에서는 일련의 동일한 명령을 보여 줍니다.

```
$ set process/parse_style=extended
$ define y 'x'
%DCL-I-SUPERSEDE, previous value of Y has been superseded
$ sho log y
"Y" = "string" (LNM$PROCESS_TABLE)
$ define y &x
%DCL-I-SUPERSEDE, previous value of Y has been superseded
$ sho log y
"Y" = "string" (LNM$PROCESS_TABLE)
```

y 변수에 대한 두 문자열은 모두 소문자로 반환됩니다. 이것은 DEFINE 명령이 대/소문자를 보존하는 \$FILE을 사용하기 때문입니다.

따라서 구문 분석 스타일이 **TRADITIONAL**로 설정되어 있더라도 다음 보기에 나오는 것처럼 앰퍼샌드 대체를 사용하여 **EXTENDED** 파일 이름을 지정할 수 있습니다.

```
$ set process/parse=extended
$ cre file^ name.doc
Contents of an ODS5 file
Exit

$ set process/parse=traditional
$ a = "file^ name.doc"
$ type file^ name.doc
%DCL-W-PARMDEL, invalid parameter delimiter - check use of special
characters \^NAME\
$ type 'a'
%DCL-W-PARMDEL, invalid parameter delimiter - check use of special
characters \^NAME\
$ type &a
Contents of an ODS5 file
```

주

외부 명령에 대해서는 앰퍼샌드 대체가 작동하지 않습니다.

명령 프로시저 소개

DCL 명령 매개 변수에 확장 파일 이름 사용

고급 DCL 프로그래밍에는 복잡한 명령 프로시저와 PIPE 명령이 사용됩니다. 13장을 읽고 DCL 프로그래밍에 대한 기본 지식을 얻은 후 고급 프로그래밍 방법을 배우려면 이 장을 읽어야 합니다.

복잡한 명령 프로시저는 프로그램과 유사한 기능을 수행할 수 있습니다. 명령 프로시저에서 가변 입력 사용, 특정 조건이 참인 경우 프로시저의 섹션 실행, 서브 루틴 실행 또는 다른 명령 프로시저 호출 등이 가능합니다.

또한, DCL 명령인 PIPE 명령을 사용하여 프로그램과 같은 기능을 수행할 수 있습니다. 예를 들어, 동일한 DCL 명령행에서 PIPE 명령을 사용하여 다음 중 한 가지 이상의 작업을 실행할 수 있습니다.

- 파이프라인화(명령 순서)
- 입/출력 재지정
- 여러 명령 및 조건 명령 실행
- 백그라운드 처리

이 장에서는 다음 사항에 대하여 설명합니다.

- 명령 프로시저 입력
- 매개 변수를 사용하여 중첩된 명령 프로시저에 데이터 전달
- 명령 프로시저 출력
- 파일 읽기 및 쓰기(파일 I/O)
- 파일 I/O 오류 처리
- 실행 흐름 제어 방법
- 새로운 명령 수준 만들기
- Case 문 작성
- PIPE 명령 사용

명령 프로시저 입력

명령 프로시저에는 사용자가 제공하는 데이터가 필요한 경우가 많습니다. 이러한 데이터는 13장에서 설명한 것처럼 대화형으로 입력하거나 비대화형으로 제공할 수 있습니다. 이 장에서는 비대화형 입력 방법과 13장에서 설명하지 않은 다른 대화형 입력 방법에 대해 설명합니다.

특정 명령 프로시저가 실행될 때마다 동일한 데이터를 사용하려면 데이터가 필요한 명령 뒤에 오는 데이터 행의 명령 프로시저에 데이터를 포함시키십시오.

다음 명령 프로시저는 CENSUS.EXE라는 명령 프로시저를 실행합니다. CENSUS.EXE는 명령 프로시저가 실행될 때마다 1993, 1994, 1995의 데이터를 읽습니다.

```
$ ! CENSUS.COM
$ !
$ RUN CENSUS
1993
1994
1995
$ EXIT
```

명령 프로시저에 데이터 포함 시 제한 사항

데이터 행의 텍스트가 명령 프로시저에 직접 전달되므로 다음과 같이 번역되어야 할 데이터는 처리되지 않습니다.

- 기호
- 논리 이름
- 산술 표현식

그 밖의 데이터 입력 방법

명령 프로시저를 위한 그 밖의 데이터 입력 방법은 다음과 같습니다.

- 매개 변수를 사용하여 데이터 전달
- 매개 변수를 사용하여 일괄 처리 작업에 데이터 전달
- 매개 변수를 사용하여 중첩된 명령 프로시저에 데이터 전달
- INQUIRE 및 READ 명령을 사용하여 데이터 프롬프트 표시
- SYS\$INPUT 논리 이름을 사용하여 데이터 입력

매개 변수를 사용하여 데이터 전달

데이터를 매개 변수로서 명령 프로시저에 전달할 때는 다음 지침을 따르십시오.

- 명령 프로시저의 파일 지정 뒤에 매개 변수를 넣으십시오.
- 한 명령 프로시저에 최대 8개의 매개 변수까지 전달할 수 있습니다.
- 8개 이하의 매개 변수를 전달할 경우에는 **null** 값에 별도의 기호가 할당됩니다.
null 값은 문자가 없는 문자열이며 인용 부호(“ ”)로 표시됩니다.
- 각 매개 변수는 하나 이상의 공백이나 탭으로 구분하십시오.

명령 프로시저에 전달된 매개 변수는 로컬 기호 P1에서 P8에 배치됩니다. 첫 번째 매개 변수 값에 P1, 두 번째 매개 변수 값에 P2, 세 번째에 P3 하는 식으로 할당됩니다. 예를 들어, 다음 명령은 SUM.COM 명령 프로시저를 호출하고 그 프로시저에 8개의 매개 변수를 전달합니다.

```
$ @SUM 34 52 664 89 2 72 87 3
```

매개 변수를 정수로 지정

정수를 매개 변수로 지정하면 문자열로 변환됩니다. 다음 보기에서 P1은 문자열 값 24; P2는 문자열 값 25입니다.

```
$ @ADDER 24 25
```

필요한 변환이 자동으로 수행되므로 정수와 문자열 표현식 모두 P1에서 P8 기호를 사용할 수 있습니다.

매개 변수를 문자열로 지정

문자열의 공백, 탭 또는 소문자를 그대로 유지하려면 다음 보기와 같이 문자열 전후에 인용 부호(“ ”)를 넣습니다. 예를 들면 다음과 같습니다.

```
$ @DATA "Paul Cramer"
```

다음 보기에서 P1은 Paul Cramer이고 P2는 null입니다. 인용 부호를 생략하면 다음 보기와 같이 각 문자열이 별개의 매개 변수로 전달됩니다. 예를 들면 다음과 같습니다.

```
$ @DATA Paul Cramer
```

이 보기에서는 문자열 Paul과 Cramer가 대문자로 변환되어 P1은 PAUL이고 P2는 CRAMER입니다.

다음 명령을 사용하여 DATA.COM을 호출하는 경우는 다음과 같습니다.

```
$ @DATA "Paul Cramer" 24 "(555) 111-1111")
```

DATA.COM에서 P1에서 P8는 아래와 같이 정의됩니다.

P1 = Paul Cramer

P2 = 24

P3 = (555) 111-1111

P4-P8 = null

매개 변수를 기호로 지정

매개 변수 값을 기호로 전달하려면 기호 전/후에 어포스트로피를 넣습니다. 기호 값의 공백, 탭, 소문자를 그대로 유지하려면 기호 값을 세 쌍의 인용 부호로 묶어야 합니다. 인용 부호를 문자열의 일부로 포함시킬 경우에도 세 쌍의 인용 부호를 사용해야 합니다.

텍스트를 인용 부호로 묶고 어포스트로피를 기호 앞에 두 개, 뒤에 한 개씩 넣는 방법도 있습니다.

다음 보기에서는 기호를 명령 프로시저에 전달할 때 인용 부호가 제거되었기 때문에 P1은 Paul이고 P2는 Cramer입니다.

```
$ NAME = "Paul Cramer"  
$ @DATA 'NAME'
```

다음 보기에서 P1이 "Paul Cramer"이고 P2는 null입니다.

```
$ NEW_NAME = "" "Paul Cramer" ""  
$ @DATA 'NEW_NAME'
```

다음 보기에서 P1은 Paul Cramer로 변환됩니다.

```
$ ! DATA.COM  
$ @NAME "'P1'"
```

매개 변수를 null 값으로 지정

null 매개 변수를 전달하려면 명령 문자열에 자리 표시자로 한 쌍의 인용 부호를 사용합니다. 다음 보기에서 DATA.COM으로 전달된 첫 번째 매개 변수는 null 매개 변수입니다.

```
$ @DATA " " "Paul Cramer"
```

이 보기에서 P1은 null, P2는 Paul Cramer입니다.

매개 변수를 사용하여 일괄 처리 작업에 데이터 전달

매개 변수를 일괄 처리 모드로 명령 프로시저에 전달하려면 **SUBMIT** 명령을 **/PARAMETERS** 한정자와 함께 사용합니다.

하나의 **SUBMIT** 명령을 사용하여 두 개 이상의 명령 프로시저를 실행하는 경우에는 일괄 처리 작업의 각 명령 프로시저에 대해 지정된 매개 변수가 사용됩니다.

다음 보기에서는 세 개의 매개 변수가 일괄 처리 작업으로 실행되는 **ASK.COM**과 **GO.COM** 명령 프로시저에 전달됩니다.

```
$ SUBMIT/PARAMETERS=(TODAY,TOMORROW,YESTERDAY) ASK.COM, GO.COM)
```

다음 보기에서는 두 개의 매개 변수가 **LIBRARY.COM**과 **SORT.COM** 명령 프로시저에 전달됩니다.

```
$ SUBMIT-
_ $ /PARAMETERS=(DISK:[ACCOUNT.BILLS]DATA.DAT,DISK:[ACCOUNT]NAME.DAT) -
_ $ LIBRARY.COM, SORT.COM
```

일괄 처리 작업은 로그인하여 각 명령 프로시저를 실행한 것처럼 실행됩니다.

이 **SUBMIT** 명령은 사용자 계정으로 로그인된 일괄 처리 작업을 실행하고, 로그인 명령 프로시저를 실행한 후 다음 명령을 실행합니다.

```
$ @LIBRARY DISK:[ACCOUNT.BILLS]DATA.DAT DISK:[ACCOUNT]NAME.DAT)
$ @SORT DISK:[ACCOUNT.BILLS]DATA.DAT DISK:[ACCOUNT]NAME.DAT)
```

데이터를 명령 프로시저에 포함시키거나 **SYSS\$INPUT** 파일을 정의하여 일괄 처리 작업에 데이터를 전달하는 방법도 있습니다. 일괄 처리 작업의 각 명령 프로시저에 대해 지정된 매개 변수가 사용됩니다.

매개 변수를 사용하여 중첩된 명령 프로시저에 데이터 전달

최대 8개의 매개 변수를 중첩된 명령 프로시저에 전달할 수 있습니다. 중첩된 프로시저의 P1에서 P8의 로컬 기호는 호출하는 프로시저의 P1에서 P8 로컬 기호와 관계가 없습니다.

다음 보기에서 DATA.COM은 중첩된 명령 프로시저인 NAME.COM을 호출합니다.

```
$ ! DATA.COM  
$ @NAME 'P1' Joe Cooper
```

DATA.COM의 P1이 인용 부호가 없는 문자열 Paul Cramer이면 NAME.COM에 두 개의 매개 변수로 전달됩니다. NAME.COM에서 P1-P8은 다음과 같이 정의됩니다.

```
P1 = PAUL  
P2 = CRAMER  
P3 = JOE  
P4 = COOPER  
P5-P8 = null
```

DATA.COM에서 P1이 인용 부호가 있는 “Paul Cramer”이면 다음과 같이 세 쌍의 인용 부호로 묶어 하나의 매개 변수로 NAME.COM에 전달할 수 있습니다.

```
$ ! DATA.COM  
$ QUOTE = ""  
$ P1 = QUOTE + P1 + QUOTE  
$ @NAME 'P1' "Joe Cooper"
```

이 보기에서 NAME.COM 명령 프로시저의 P1은 Paul Cramer이고 P2는 Joe Cooper입니다.

데이터 프롬프트 표시

INQUIRE 명령(13장에서 설명)을 사용하거나 READ 명령을 사용하여 명령 프로시저에 필요한 데이터를 대화형으로 입력하도록 할 수 있습니다. 이 두 명령은 모두 입력을 위한 프롬프트를 표시하고 그 응답을 기호에 할당합니다.

READ 명령과 INQUIRE 명령은 다음과 같은 차이가 있습니다.

표 14-1

INQUIRE 명령	READ 명령
값을 위한 프롬프트를 표시합니다.	값을 위한 프롬프트를 표시합니다.
터미널로부터 값을 읽습니다.	첫 번째 매개 변수로 지정된 소스로부터 값을 읽습니다
값을 기호에 할당합니다.	값을 두 번째 매개 변수로서 이름이 지정된 기호에 할당합니다

READ 명령은 프롬프트에 대한 응답으로 터미널에 입력된 모든 문자를 대/소문자, 공백, 탭을 포함하여 그대로의 문자열로 받아들입니다. /PROMPT 한정자를 생략하면 기본 프롬프트인 Data:가 표시됩니다.

또한, 매개 변수를 받아들이거나 필수 매개 변수가 지정되지 않을 경우 사용자에게 입력하라는 프롬프트를 표시할 수 있는 명령 프로시저를 작성할 수 있습니다.

다음 명령은 터미널에 Filename:이라는 프롬프트를 표시하고 논리 이름인 SYS\$COMMAND에 의해 지정된 소스(기본적으로 터미널)로부터 응답을 읽어 그 응답을 FILE:이라는 기호에 할당합니다.

```
$ READ/PROMPT="Filename: " SYS$COMMAND FILE
```

다음 보기에서 프로시저가 호출될 때 파일 이름이 지정되어 있지 않으면 사용자에게 파일 이름을 입력하라는 프롬프트를 표시합니다.

```
$ ! Prompt for a file name if name  
$ ! is not passed as a parameter  
$ IF P1 .EQS. "" THEN INQUIRE P1 "Filename"  
$ COPY 'P1' DISK5:[RESERVED]*.*  
$ EXIT
```

주

일괄 처리 작업으로 실행하기 위해 명령 프로시저를 제출하는 경우에는 **INQUIRE** 명령 뒤에 나오는 데이터 행으로부터 **INQUIRE** 명령에 지정된 기호의 값을 읽습니다. 데이터 행을 포함시키지 않으면 기호에는 **null** 값이 할당됩니다.

SYS\$INPUT 논리 이름을 사용하여 데이터 입력

명령, 유틸리티 및 그 밖의 시스템 이미지는 기본 **입력 스트림**인 논리 이름 SYS\$INPUT에 의해 지정된 소스로부터 입력 데이터를 얻습니다. 명령 프로시저에서 SYS\$INPUT은 명령 프로시저 파일로 정의됩니다. 데이터를 필요로 하는 명령 또는 이미지인 경우에는 그 파일의 데이터 행에서 데이터를 검색합니다.

그러나 SYS\$INPUT을 재정의하여 터미널이나 별도의 입력 파일로부터 데이터를 제공할 수 있습니다.

SYS\$INPUT을 터미널로 재정의

SYS\$INPUT을 터미널로 재정의하여 명령 프로시저로부터 호출된 이미지에 대화형으로 데이터를 입력할 수 있습니다.

명령 프로시저에 대화형으로 데이터를 입력해야 하는 DCL 명령이나 유틸리티를 사용하려면 SYS\$INPUT을 터미널로 재정의해야 합니다.

다음 명령 프로시저는 CENSUS.EXE 이미지에 대화형으로 입력 사항을 제공할 수 있도록 합니다.

```
$ ! Execute CENSUS getting data from the terminal
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$ RUN CENSUS
$ EXIT
```

DEFINE/USER_MODE 명령은 CENSUS.EXE가 실행되는 동안 일시적으로 SYS\$INPUT을 재정의하여 CENSUS.EXE에 터미널로부터 입력 사항을 제공합니다. SYS\$INPUT은 CENSUS.EXE가 완료된 후 원래의 정의인 명령 프로시저 파일로 되돌아갑니다.

다음 명령 프로시저는 EVE를 텍스트 편집기로 사용합니다.

```
$ ! Obtain a list of your files
$ DIRECTORY
$ !
$ ! Get file name and invoke the EVE editor
$ EDIT_LOOP:
$     INQUIRE FILE "File to edit (Press Return to end)"
$     IF FILE .EQS. "" THEN EXIT
$     DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$     EDIT/TPU 'FILE'
$     GOTO EDIT_LOOP
```


명령 프로시저는 사용자가 **Return** 키를 눌러 루프를 종료할 때까지 파일 이름을 위한 프롬프트를 표시합니다. 이 프로시저는 사용자가 파일 이름을 입력하면 파일 편집을 위해 **EVE**를 자동으로 호출합니다. 편집기가 실행되고 있는 동안 **SYS\$INPUT**은 터미널로 정의되어 편집 사항을 대화형으로 입력할 수 있습니다.

SYS\$INPUT을 별개의 파일로 정의

SYS\$INPUT을 파일로 정의하여 파일을 통해 명령 프로시저에 입력 사항을 제공할 수 있습니다. 데이터 행은 처리되지 않는다는 점을 주의하십시오. 명령 프로시저는 데이터 행의 텍스트를 명령이나 이미지로 직접 전달하며 데이터 행에 **DCL** 기호나 표현식을 포함시킬 경우 기호를 값으로 대체하거나 표현식을 평가하지 않습니다. 데이터 행에 감탄 부호(!)를 사용하면 데이터를 전달받은 이미지에서 처리합니다.

또한, 데이터 파일 이름을 **SYS\$INPUT**으로 지정하여 명령 프로시저 파일에 프로그램을 포함시킬 수 있습니다. 이렇게 하면 컴파일러가 다른 파일이 아닌 명령 프로시저에서 프로그램을 읽게 됩니다.

다음 보기에서는 프로그램 문이 뒤에 오는 **FORTTRAN** 명령을 포함하는 명령 프로시저를 보여 줍니다.

```
$ FORTRAN/OBJECT=TESTER/LIST=TESTER SYS$INPUT
C  THIS IS A TEST PROGRAM
   A = 1
   B = 2
   STOP
   END
$ PRINT TESTER.LIS
$ EXIT
```

FORTTRAN 명령에서는 논리 이름인 **SYS\$INPUT**을 사용하여 컴파일될 파일을 식별합니다. **SYS\$INPUT**은 명령 프로시저와 같으므로 **FORTTRAN** 컴파일러는 **FORTTRAN** 명령 뒤의 명령문을 달러 기호로 시작하는 다음 행까지 컴파일합니다.

컴파일이 종료되면 **TESTER.OBJ**와 **TESTER.LIS** 두 개의 출력 파일이 만들어집니다. 그 다음에는 **PRINT** 명령이 파일을 인쇄합니다.

명령 프로시저 출력 수행

데이터, 오류 메시지, 명령행 확인과 같은 명령 프로시저로부터의 출력이 터미널이나 다른 파일로 이루어지도록 할 수 있습니다. 여기에서 설명할 출력 지정 방법은 다음과 같습니다.

- 데이터 표시
- 명령 및 이미지로부터 출력 다시 지정
- 명령 프로시저에서 데이터 반환
- 오류 메시지 다시 지정

데이터 표시

기호를 대체할 필요가 없는 몇 행으로 된 텍스트를 표시하려면 **TYPE** 명령을 사용합니다. **TYPE** 명령은 지정하는 파일의 데이터를 **SYS\$OUTPUT**에 기록합니다.

다음 보기에서 **SYS\$INPUT**은 데이터 파일로 지정됩니다. **TYPE** 명령은 그 뒤의 데이터 행에서 데이터를 읽어 터미널에 표시합니다.

```
$ ! Using TYPE to display lines
$ TYPE SYS$INPUT
REPORT BY MARY JONES
PREPARED APRIL 15, 2002
SUBJECT: Analysis of Tax Deductions for 2002
.
.
.
$ EXIT
```

기호 또는 렉시칼 함수를 포함하는 데이터를 쓰려면 **WRITE** 명령을 사용합니다. 데이터를 인용 부호(“”)로 묶지 않으면 **WRITE** 명령은 기호를 자동으로 대체합니다.

WRITE 명령을 사용하여 문자열을 리터럴 텍스트로 표시하려면 다음 보기처럼 문자열을 인용 부호(“”)로 묶습니다.

```
$ WRITE SYS$OUTPUT "Two files are written."
Two files are written.
```

문자열에 인용 부호를 포함시키려면 다음 보기처럼 두 쌍의 인용 부호(“”)를 사용합니다.

```
$ WRITE SYS$OUTPUT "Summary of ""Q & A"" Session"
Summary of "Q & A" Session
```

텍스트의 한 행을 두 행 이상 계속쓰려면 다음 보기처럼 더하기 기호(+)와 하이픈(-)을 사용하여 두 문자열을 연결합니다.

```
$ WRITE SYS$OUTPUT "Report by Mary Jones" + -
" Prepared April 15, 2002"
Report by Mary Jones Prepared April 15, 2002
```

WRITE 명령은 기호 대체를 자동으로 수행하고 기호의 값을 표시합니다. 문자열 내에서 기호 대체를 수행하려면 다음 보기처럼 기호를 어포스트로피로 묶습니다.

```
$ AFILE = "STAT1.DAT"$ BFILE = "STAT2.DAT"
$ WRITE SYS$OUTPUT "'AFILE' and 'BFILE' ready."
STAT1.DAT and STAT2.DAT ready.
```

이 보기에서 **STAT1.DAT**는 **AFILE** 기호로, **STAT2.DAT**는 **BFILE** 기호로 변환됩니다.

명령 및 이미지에서 출력 재지정

명령, 유틸리티, 그 외의 시스템 이미지는 논리 이름인 **SYS\$OUTPUT**에 의해 지정된 소스에 출력 사항을 씁니다. 기본적으로 **SYS\$OUTPUT**은 터미널에 해당합니다. 그러나 다음 중 한 가지 방법으로 출력을 다시 지정할 수 있습니다.

- 명령을 호출할 때 **/OUTPUT** 한정자를 사용합니다. **/OUTPUT** 한정자를 받아들이는 DCL 명령으로는 **ACCOUNTING**, **CALL**, **DIRECTORY**, **HELP**, **LIBRARY**, **RUN** (프로세스), **SPAWN**, 및 **TYPE**이 있습니다.
- **DEFINE/USER_MODE** 명령을 사용하여 **SYS\$OUTPUT**을 파일로 일시적으로 다시 지정합니다.
- **DEFINE/USER_MODE** 명령을 사용하여 **SYS\$OUTPUT**을 일시적으로 **null** 장치로 정의하여 명령에서 출력하지 못하도록 합니다.

다음 명령 프로시저는 **SHOW USERS** 명령의 출력을 파일로 재지정합니다. 이 새로운 **SYS\$OUTPUT** 정의는 **SHOW USERS** 명령 실행에만 유효합니다.

```
$ DEFINE/USER_MODE SYS$OUTPUT SHOW_USER.DAT
$ SHOW USERS
$ !
$ ! Process the information in SHOW_USER.DAT
$ OPEN/READ INFILE SHOW_USER.DAT
$ READ INFILE RECORD
.
.
.
$ CLOSE INFILE
$ EXIT
```

다음 보기에서 **SYS\$OUTPUT**은 **null** 장치(NL:)로 정의됩니다.

```
$ DEFINE/USER_MODE SYS$OUTPUT NL:
$ APPEND NEW_DATA.DAT STATS.DAT
.
.
.
```

/USER_MODE 한정자는 다음 이미지가 완료될 때까지만 유효한 임시 논리 이름 할당을 만드는데 사용됩니다. 명령이 실행된 후에는 **SYS\$OUTPUT**이 기본 정의(일반적으로 터미널)로 되돌아갑니다.

명령 인터프리터 내에서 실행되는 **DCL** 명령에서의 출력을 재정의할 때는 **DEFINE/USER_MODE** 명령을 사용할 수 없습니다. 그 대신 **DEFINE** 명령을 사용하여 **SYS\$OUTPUT**을 재정의하고 사용이 끝난 후 **DEASSIGN** 명령을 사용하여 그 정의를 삭제합니다.

다음 목록은 명령 인터프리터 내에서 수행되는 모든 **DCL** 명령 목록입니다.

표 14-2

=	ALLOCATE	ASSIGN
ATTACH	CALL	CANCEL
CLOSE	CONNECT	CONTINUE
CREATE/LOGICAL_NAME_TABLE	DEALLOCATE	DEASSIGN
DEBUG	DECK	DEFINE
DEFINE/KEY	DELETE/SYMBOL	DISCONNECT

표 14-2

(계속)

ELSE	ENDIF	ENDSUBROUTINE
EOD	EXAMINE	EXIT
GOSUB	GOTO	IF
INQUIRE	ON	OPEN
READ	RECALL	RETURN
SET CONTROL	SET DEFAULT	SET KEY
SET ON	SET OUTPUT_RATE	SET PROMPT
SET PROTECTION/DEFAULT	SET SYMBOL/SCOPE	SET UIC
SET VERIFY	SHOW DEFAULT	SHOW KEY
SHOW PROTECTION	SHOW QUOTA	SHOW STATUS
SHOW SYMBOL	SHOW TIME	SHOW TRANSLATION
SPAWN	STOP	SUBROUTINE
THEN	WAIT	WRITE

다음 보기는 **SHOW TIME** 명령에서 **TIME.DAT** 파일로 출력을 재지정하는 데 사용되는 명령을 보여 줍니다. **SYS\$OUTPUT**의 할당을 취소하면 기본 정의(터미널)로 되돌아갑니다.

```
$ DEFINE SYS$OUTPUT TIME.DAT
$ SHOW TIME
$ DEASSIGN SYS$OUTPUT
```

명령 프로시저에서 데이터 반환

글로벌 기호와 논리 이름은 명령 프로시저에서 호출하는 프로시저나 **DCL** 명령 수준에 데이터를 반환합니다. 글로벌 기호나 논리 이름은 어느 명령 수준에서나 읽을 수 있습니다. 논리 이름은 중첩된 명령 프로시저에서 호출하는 프로시저로 데이터를 반환할 수 있습니다. 다음 보기는 글로벌 할당문을 사용하여 만들어진 글로벌 기호로 값을 전달하는 방법을 보여 줍니다.

```
$ @DATA "Paul Cramer"$ ! DATA.COM
$ !
$ ! P1 is a full name.
$ ! NAME.COM returns the last name in the
$ ! global symbol LAST_NAME.
$ !
$ @NAME 'P1'
    $ ! NAME.COM
    $ ! P1 is a first name
    $ ! P2 is a last name
    $ ! return P2 in the global symbol LAST_NAME
    $ LAST_NAME == P2
    $ EXIT

$ ! write LAST_NAME to the terminal
$ WRITE SYS$OUTPUT "LAST_NAME = '"LAST_NAME'"
"LAST_NAME = CRAMER
```

DATA.COM은 **NAME.COM** 명령 프로시저를 호출하여 완전한 이름을 전달합니다.

NAME.COM은 성을 글로벌 기호인 **LAST_NAME**에 넣습니다. **NAME.COM**이 완료되면 **DATA.COM**이 계속 실행되어 **LAST_NAME**을 지정함으로써 성을 읽습니다.

NAME.COM 명령 프로시저는 별개의 파일에 있습니다. 이 보기에서는 명확성을 위해 그 부분을 들여썼습니다.

다음 명령 프로시저에서 **REPORT.COM**은 보고서 이름을 읽어 그 파일 이름을 논리 이름인 **REPORT_FILE**로 일치시켜 **REPORT_FILE**에 보고서를 쓰는 프로그램을 실행합니다.

```
$! Obtain the name of a file and then run
$! REPORT.EXE to write a report to the file
$!
$ INQUIRE FILE "Name of report file"$ DEFINE/NOLOG REPORT_FILE 'FILE'
$ RUN REPORT
$ EXIT
```

다음 보기에서 **REPORT.COM** 명령 프로시저는 다른 프로시저에서 호출됩니다. 호출하는 프로시저는 보고서 파일을 참조하는 논리 이름 **REPORT_FILE**을 사용합니다.

```
$! Command procedure that updates data files
$! and optionally prepares reports
$!
$ UPDATE:
.
.
.
$   INQUIRE REPORT "Prepare a report [Y or N]"
$   IF REPORT THEN GOTO REPORT_SEC
$   EXIT
$!
$ REPORT_SEC:
$   @REPORT
$   WRITE SYS$OUTPUT "Report written to ", F$TRNLNM("REPORT_FILE")
$   EXIT
```

오류 메시지 재지정

다음 절에서는 오류 메시지 재지정 방법에 대해 설명합니다.

SYS\$ERROR 재정의

기본적으로 명령 프로시저는 시스템 오류 메시지를 **SYS\$ERROR**에 의해 표시된 파일로 보냅니다. 시스템 오류 메시지를 지정된 파일로 보내도록 **SYS\$ERROR**를 재정의할 수 있습니다. 그러나 **SYS\$ERROR**를 **SYS\$OUTPUT**과 다르게 재정의하거나 **SYS\$ERROR**를 지정하지 않고 **SYS\$OUTPUT**를 재정의하면 표준 시스템 오류 표시 방법을 사용하는 DCL 명령과 이미지는 시스템 오류 메시지와 시스템 중대 오류 메시지를 **SYS\$ERROR**와 **SYS\$OUTPUT**에 모두 보냅니다. 따라서 **SYS\$ERROR** 정의에 표시된 파일에 한 번, **SYS\$OUTPUT**에 표시된 파일에 한 번, 오류 메시지가 두 번 표시됩니다. 성공 메시지, 정보 메시지 및 경고 메시지는 **SYS\$OUTPUT**에 표시된 파일에만 전달됩니다. DCL 명령에서 시스템 오류 메시지 표시를 막으려면 **SYS\$ERROR**나 **SYS\$OUTPUT** 중 어느 하나도 터미널로 지정하지 말아야 합니다.

명령 프로시저에서 사용자 정의 이미지 중 하나를 실행하고 그 이미지가 **SYS\$ERROR**를 참조하는 경우에는 **SYS\$ERROR**가 **SYS\$OUTPUT**과 다르더라도 **SYS\$ERROR**에 의해 표시된 파일에만 시스템 오류 메시지가 전달됩니다. 표준 시스템 오류 표시 방법을 사용하는 DCL 명령과 이미지만 **SYS\$ERROR**와 **SYS\$OUTPUT**이 다를 때 이 두 파일에 메시지를 보냅니다.

다음 명령 프로시저는 디렉토리 이름을 매개 변수로 받아 그 디렉토리를 기본값으로 설정하고 디렉토리에 있는 파일을 제거합니다. 시스템 오류 메시지가 표시되지 않도록 일시적으로 **SYS\$ERROR**와 **SYS\$OUTPUT**를 null 장치로 정의합니다.

```
$ ! Purge files in a directory and suppress messages
$ !
$ SET DEFAULT 'P1'
$ ! Suppress messages
$ !
$ DEFINE/USER_MODE SYS$ERROR NL:
$ DEFINE/USER_MODE SYS$OUTPUT NL:
$ PURGE
$ EXIT
```

시스템 오류 메시지 표시 방지

SET MESSAGE 명령을 사용하여 시스템 오류 메시지가 표시되지 않도록 하는 방법도 있습니다. **/NOFACILITY**, **/NOIDENTIFICATION**, **/NOSEVERITY** 또는 **/NOTEXT** 한정자를 사용하여 장치 이름, 메시지 ID, 오류 심각도 또는 메시지 텍스트가 표시되지 않도록 할 수 있습니다.

다음 보기에서는 두 번째 **SET MESSAGE** 명령에 이를 때까지 장치, ID, 심각도 및 텍스트 메시지가 일시적으로 표시되지 않습니다.

```
$ ! Purge files in a directory and suppress system messages
$ !
$ SET DEFAULT 'P1'
$ ! Suppress system messages
$ !
$ SET MESSAGE/NOFACILITY -
                        /NOIDENTIFICATION -
                        /NOSEVERITY -
                        /NOTEXT

$ PURGE
$ SET MESSAGE/FACILITY -
                        /IDENTIFICATION -
                        /SEVERITY
                        /TEXT

$ EXIT
```


읽기 및 쓰기 파일(파일 I/O)

명령 프로시저에서 파일을 읽고 쓰는 기본적인 단계는 다음과 같습니다.

표 14-3

단계	작업
1	OPEN 명령을 사용하여 파일을 엽니다. 파일을 열면 파일에 논리 이름이 할당되고 읽기 파일인지 쓰기 파일인지 또는 읽고 쓰는 파일인지 여부가 지정됩니다. 이후의 READ, WRITE, CLOSE 명령에서는 이 논리 이름을 사용하여 해당 파일을 참조합니다.
2	READ 또는 WRITE 명령을 사용하여 파일에 레코드를 읽거나 씁니다. 파일에 대한 입력과 출력은 일반적으로 레코드를 읽고 처리하여 수정한 레코드를 동일한 파일이나 다른 파일에 쓰는 루프를 통해 수행됩니다.
3	CLOSE 명령을 사용하여 파일을 닫습니다. CLOSE 명령을 포함시키지 않으면 로그아웃할 때까지 파일이 열려 있습니다.

주

SYS\$INPUT, **SYS\$OUTPUT**, **SYS\$COMMAND** 및 **SYS\$ERROR** 등의 영구 프로세스 파일은 별도로 읽거나 쓰지 않아도 로그인할 때 자동으로 열립니다.

다음 절에서 설명할 내용은 다음과 같습니다.

- **OPEN** 명령 사용
- 파일에 쓰기
- **WRITE** 명령 사용
- **READ** 명령 사용
- **CLOSE** 명령 사용
- 파일 수정
 - 레코드 업데이트
 - 새 출력 파일 만들기
 - 파일에 레코드 추가

OPEN 명령 사용

OPEN 명령은 순차적, 상대적 또는 인덱스 순차적 파일을 엽니다. 파일은 영구 프로세스로 열려 사용자가 CLOSE 명령을 사용하여 명시적으로 닫지 않는 한 프로세스가 지속되는 동안 열려 있습니다. 파일이 열려 있는 동안에는 영구 프로세스 파일에 대한 OpenVMS RMS 제한의 대상이 됩니다.

파일을 열 때 OPEN 명령은 첫 번째 매개 변수로 지정된 논리 이름을 두 번째 매개 변수로 지정된 파일에 할당하고 그 이름을 프로세스 논리 이름 테이블에 저장합니다. 이후의 READ, WRITE, CLOSE 명령에서는 이 논리 이름을 사용하여 파일을 참조합니다.

다음 보기에서 OPEN 명령은 INFILE이라는 논리 이름을 DISK4:[MURPHY]STATS.DAT에 할당합니다.

```
$ OPEN/READ INFILE DISK4:[MURPHY]STATS.DAT
```

주

OPEN 명령의 논리 이름은 고유한 이름이어야 합니다. OPEN 명령이 작동하지 않고 틀린 점이 없는 것으로 생각되는 경우에는 OPEN 명령의 논리 이름을 변경하십시오. 논리 이름 정의를 표시하려면 SHOW LOGICAL 명령을 사용합니다.

명령 프로시저가 올바른 파일에 액세스할 수 있도록 하려면 완전한 파일 지정 방법(예: DISK4:[MURPHY]STATS.DAT)을 사용하거나 파일을 열기 전에 SET DEFAULT 명령을 사용하여 올바른 장치와 디렉토리를 지정하십시오.

또한, 공유 파일을 지정할 수 있습니다. /SHARE 한정자를 사용하여 다른 열려 있는 파일을 활성화합니다. 그 외에도 DCL 명령인 TYPE과 SEARCH를 사용하여 사용자가 공유 파일에 액세스할 수 있습니다.

OPEN/READ 명령은 파일에 논리 이름을 할당하고 레코드 포인터를 파일 시작에 둡니다. 읽기를 위해 파일을 열면 레코드를 읽을 수만 있고 쓰지는 못합니다. 레코드를 읽을 때마다 포인터가 다음 레코드로 이동합니다.

다음 명령 프로시저에서 **OPEN/READ** 명령은 **STATS.DAT** 파일을 열고 논리 이름인 **INFILE**을 할당합니다.

```
$ OPEN/READ INFILE DISK4:[MURPHY]STATS.DAT
$ READ_FILE:
$ READ/END_OF_FILE=DONE INFILE DATA
$ GOTO READ_FILE
$ DONE:
$ CLOSE INFILE
$ EXIT
```

새 파일에 쓰려면 **OPEN/WRITE** 명령을 사용합니다. **OPEN/WRITE** 명령은 인쇄 파일 형식의 순차적 파일을 만듭니다. 파일을 위한 레코드 형식은 레코드 헤더가 2바이트인 고정 제어 변수(VFC)입니다. **/WRITE** 한정자는 **/APPEND** 한정자와 함께 사용할 수 없습니다.

이미 존재하는 파일을 지정하는 경우 **OPEN/WRITE** 명령은 기존 파일보다 버전 번호가 하나 큰 새 파일을 엽니다.

다음 보기에서 명령 프로시저는 쓰기를 위해 사용할 수 있는 새 파일 **NAMES.DAT**를 만듭니다.

```
$ OPEN/WRITE OUTFILE DISK4:[MURPHY]NAMES.DAT
$ UPDATE:
$ INQUIRE NEW_RECORD "Enter name"
$ WRITE OUTFILE NEW_RECORD
$ IF NEW_RECORD .EQS. "" THEN GOTO EXIT_CODE
$ GOTO UPDATE
$ EXIT_CODE:
$ CLOSE OUTFILE
$ EXIT
```

OPEN/APPEND 명령은 기존 파일 끝에 레코드를 추가합니다. 존재하지 않는 파일을 열려고 하면 오류가 발생하고 파일이 열리지 않습니다. **/APPEND** 한정자는 **/WRITE** 한정자와 함께 사용할 수 없습니다.

다음 보기에서는 기존 파일인 **NAMES.DAT**의 끝에 레코드가 추가됩니다.

```
$ OPEN/APPEND OUTFILE DISK4:[MURPHY]NAMES.DAT
$ INQUIRE NEW_RECORD "Enter name"$ WRITE OUTFILE NEW_RECORD
.
.
.
$ CLOSE OUTFILE
```

OPEN/READ/WRITE 명령은 파일의 처음에 레코드 포인트를 두어 첫 레코드를 읽을 수 있도록 합니다. 이 방법을 사용하여 파일을 열면 가장 최근에 읽었던 레코드만 대체할 수 있고 파일의 끝에 새 레코드를 쓸 수 없습니다. 그 외에도 수정된 레코드의 크기가 교체되는 레코드의 크기와 꼭 같아야 합니다.

다음 보기에서는 레코드 포인터가 **STATS.DAT** 파일의 시작에 있어서 첫 번째 레코드를 읽을 수 있습니다.

```
$ OPEN/READ/WRITE FILE DISK4:[MURPHY]STATS.DAT
```

파일에 쓰기

파일에 쓰려면 다음 절차를 사용하십시오.

표 14-4

단계	작업
1	쓰기 위해 파일을 엽니다.
2	레이블을 사용하여 쓰기 루프를 시작합니다. 파일 I/O는 단일 레코드를 쓰거나 읽는 경우가 아니면 항상 루프로 수행됩니다.
3	쓰여질 데이터를 읽습니다. INQUIRE 명령이나 READ 명령을 사용하여 데이터를 기호로 읽어 들입니다.
4	데이터를 테스트합니다. 데이터를 포함한 기호를 점검합니다. 기호가 null 이면(예: Return 키를 누르고 그 행에 데이터를 입력하지 않은 경우) 파일에 쓰여질 데이터의 끝에 이르렀으며 루프의 끝으로 가야 합니다. 그렇지 않으면 계속합니다.
5	데이터를 파일에 씁니다. WRITE 명령을 사용하여 기호의 값(한 개의 레코드)을 파일에 씁니다.
6	루프의 시작으로 돌아갑니다. 파일에 쓸 데이터가 없을 때까지 루프 내에 있어야 합니다.
7	루프를 끝내고 파일을 닫습니다.

다음 명령 프로시저는 새 파일 **STATS.DAT**에 데이터를 씁니다. 그 이름의 파일이 존재하지 않는 경우에는 새 버전이 만들어집니다.

```
$ ! Write a file
$ ON ERROR THEN EXIT                ! Exit if the command
$ !                                !   procedure cannot
$ !                                !   open the file
$ OPEN/WRITE IN_FILE DISK4:[MURPHY]STATS.DAT ! Open the file
$ ON CONTROL_Y THEN GOTO END_WRITE    ! Close the file if you
$ !                                !   quit execution with
$ !                                !   Ctrl/Y
$ ON ERROR THEN GOTO END_WRITE        ! Close the file if an
$ !                                !   error occurs
$WRITE:                              ! Begin the loop
$ INQUIRE STUFF "Input data"         ! Prompt for input
$ IF STUFF .EQS. " " THEN GOTO END_WRITE ! Test for the end of
$ !                                !   the file
$ WRITE IN_FILE STUFF                ! Write to the file
$ GOTO WRITE                         ! Go to the beginning
$END_WRITE:                          ! End the loop
$ !                                !
$ CLOSE IN_FILE                      ! Close the file
```

고유한 이름의 파일 만들기

고유한 이름의 파일을 만들려면 렉시칼 함수인 **F\$SEARCH**를 사용하여 디렉토리에 이미 존재하는 이름인지 확인해야 합니다. **F\$SEARCH**에 대한 자세한 내용은 *OpenVMS DCL Dictionary*의 렉시칼 함수에 대한 설명을 참조하십시오.

다음 명령 프로시저는 사용자에게 파일 이름을 입력하라는 프롬프트를 표시한 다음 **F\$SEARCH** 렉시칼 함수를 사용하여 기본 디렉토리에 있는 이름인지 검색합니다. 이미 존재하는 이름의 파일인 경우에는 **ERROR_1**으로 제어가 전달되어 “The file already exists”라는 메시지를 표시한 후 **GET_NAME** 레이블로 제어를 반환한 후 다음 보기에 나타난 것처럼 다른 파일 이름을 입력하도록 프롬프트가 표시됩니다.

```
$ ! FILES.COM
$ !
$GET_NAME:
$ INQUIRE FILE "File"           ! Prompt the user for a file name
$ IF F$SEARCH (FILE) .NES. ""    ! Make sure the file name is unique
$ THEN
$   WRITE SYS$OUTPUT "The file already exists"$      GOTO GET_NAME
$ ELSE
$   OPEN/WRITE IN_FILE 'FILE'    ! Open the file with WRITE access
$ ENDIF
.
.
.
$ EXIT
```

WRITE 명령 사용

다음 절에서는 **WRITE** 명령 사용 방법에 대해 설명합니다.

데이터 지정

WRITE 명령을 위해 데이터를 지정할 때는 12장에서 설명한 문자열 표현식 규칙을 따르십시오. 다음과 같은 방법으로 데이터를 지정할 수 있습니다.

- 문자열 표현식으로 쓰여질 데이터를 지정합니다. **WRITE** 명령은 기호와 렉시칼 함수를 자동으로 대체합니다.
- 문자열을 리터럴 문자열로 출력 파일에 씁니다. **WRITE** 명령은 인용 부호로 묶인 문자열에 대해서는 기호 대체를 수행하지 않습니다.
- 리터럴 문자열을 기호 이름과 결합합니다. 기호 대체를 강제 실행하려면 문자열 전체를 인용 부호 안에 넣고 어포스트로피를 기호 앞에 두 개, 뒤에 한 개 넣어 식별합니다.

리터럴 문자열을 기호 이름과 결합하는 다른 방법은 다음 보기처럼 기호 앞과 뒤에 쉼표를 삽입하고 구분된 기호 주위에 인용 부호를 넣고 문자열 전체를 인용 부호로 묶는 것입니다.

```
$ WRITE OUTFILE "Count is ",COUNT,"."
```

- **WRITE** 명령행에 어포스트로피를 사용하여 기호 대체를 강제 실행합니다.
- 어포스트로피를 사용하여 리터럴 문자열과 렉시칼 함수를 결합하여 문자열 내에서 기호 대체를 강제 실행합니다.

보기

```

$! Define symbols
$!
$ CREATED = "File created April 15, 2002"
$ COUNT = 4
$ P4 = "fourth parameter"
$!
$! Open the file DATA.OUT for writing
$!
$ OPEN/WRITE OUTFILE DISK4:[MURPHY]DATA.OUT
$!
$ WRITE OUTFILE CREATED [1]
$ WRITE OUTFILE "CREATED" [2]
$!
$ WRITE OUTFILE "Count is '"COUNT'.'" [3]
$ WRITE OUTFILE P'COUNT' [4]
$!
$ WRITE OUTFILE "Mode is '"f$mode()' '" [5]
$!
$ CLOSE OUTFILE

$ TYPE DISK4:[MURPHY]DATA.OUT Return [6]
File created April 15, 2002
CREATED
Count is 4.
fourth parameter
Mode is INTERACTIVE
$

```

위의 보기에서 주목할 사항은 다음과 같습니다.

- [1] 문자열 표현식으로 쓰여질 데이터를 지정합니다.
- [2] 문자열 *CREATED*를 리터럴 문자열로 출력 파일에 씁니다.
- [3] 리터럴 문자열을 기호 이름과 결합합니다.
- [4] **WRITE** 명령행에 어포스트로피를 사용하여 기호 대체를 강제 실행합니다. 이 보기에서 **WRITE** 명령은 **COUNT** 기호를 값으로 대체하고 그 결과인 문자열(**P4**)에 대해 기호 대체를 수행합니다.
- [5] 리터럴 문자열과 렉시칼 함수를 결합합니다.
- [6] 앞에 나오는 **WRITE** 명령에 의해 출력 파일 **DATA.OUT**에 쓰여진 데이터를 표시합니다.

/SYMBOL 한정자 사용

WRITE 명령은 레코드를 쓰고 레코드 포인터를 그 레코드 다음으로 이동합니다.

WRITE 명령을 사용하여 2,048바이트 길이의 레코드까지 쓸 수 있습니다.

다음 조건 중 하나가 존재하는 경우에는 레코드를 쓸 때 /SYMBOL 한정자를 사용하십시오.

- 레코드가 1,024바이트보다 긴 경우
- WRITE 명령에 있는 표현식이 255바이트보다 긴 경우

긴 레코드 쓰기에 대한 자세한 내용은 *OpenVMS DCL Dictionary*의 WRITE 명령에 대한 설명을 참조하십시오.

/UPDATE 한정자 사용

새 레코드를 삽입하는 것이 아니라 레코드를 변경하는 경우에는 WRITE 명령을 /UPDATE 한정자와 함께 사용할 수 있습니다. /UPDATE 한정자를 사용하려면 읽기와 쓰기를 모두 할 수 있도록 파일을 열어야 합니다.

READ 명령 사용

레코드를 읽고 그 내용을 기호에 할당하려면 **READ** 명령을 사용합니다. **READ** 명령을 사용하여 길이가 1,024자 이하인 레코드를 읽을 수 있습니다. 파일에서 데이터를 읽으려면 다음 절차를 사용하십시오.

표 14-5

단계	작업
1	입기를 위해 파일을 엽니다.
2	레이블을 사용하여 읽기 루프를 시작합니다. 파일 I/O는 단일 레코드를 쓰거나 읽는 경우가 아니면 항상 루프로 수행됩니다.
3	파일에서 데이터를 읽습니다. READ 명령을 /END_OF_FILE 한정자와 함께 사용하여 레코드를 읽고 그 내용을 기호에 할당합니다. /END_OF_FILE 한정자를 사용하면 파일 끝에 도달할 때 /END_OF_FILE 한정자에 의해 지정된 레이블로 제어가 전달됩니다. 일반적으로 사용자가 읽기 루프의 끝을 표시하는 레이블을 지정합니다.
4	데이터를 처리합니다. 파일을 순차적으로 읽을 때는 현재 레코드를 다음 레코드를 읽기 전에 처리합니다.
5	루프의 처음으로 돌아갑니다. 파일 끝에 이를 때까지 루프에 남아 있습니다.
6	루프를 종료하고 파일을 닫습니다.

다음 명령 프로시저는 STATS.DAT 파일의 각 레코드를 읽고 처리합니다. end-of-file 상태가 반환될 때까지 READ 명령을 반복하여 실행한 다음 END_READ라는 레이블이 있는 행으로 분기됩니다.

```
$ OPEN/READ INFILE DISK4:[MURPHY]STATS.DAT !Open the file
$ !
$READ_DATA:                                !Begin the loop
$ READ/END_OF_FILE=END_READ INFILE RECORD !Read a record; test for
$                                           ! end of file
$                                           ! Process the data
.
.
.
$ GOTO READ_DATA                            !Go to the beginning
$                                           ! of the loop
$END_READ:                                !End of loop
$ CLOSE INFILE                             !Close the file
$ EXIT
```

READ 명령을 위한 기호 이름을 지정하면 명령 인터프리터에서 그 이름을 현재 명령 수준의 로컬 기호 테이블에 저장합니다. 두 개 이상의 READ 명령을 위해 동일한 기호 이름을 사용하는 경우에는 각 READ 명령이 기호 이름 값을 재정의합니다. 예를 들어, 앞의 보기에서 READ 명령은 루프를 통과할 때마다 입력 파일인 STATS.DAT에서 새 레코드를 읽은 다음 이 레코드를 사용하여 RECORD 기호의 값을 재정의합니다.

/END_OF_FILE 한정자 사용

파일에서 읽을 때는 일반적으로 파일 끝에 도달할 때까지 각 레코드를 읽고 처리합니다. /END_OF_FILE 한정자를 READ 명령과 함께 사용하여 파일에서 레코드를 읽고 처리한 다음 모든 레코드를 다 읽으면 종료되는 루프를 구성할 수 있습니다.

/END_OF_FILE 한정자를 위해 지정하는 레이블에는 GOTO 명령을 위해 지정된 레이블과 동일한 규칙이 적용됩니다. GOTO 명령에 대한 자세한 내용은 13장을 참조하십시오.

READ 명령을 루프에서 사용할 때는 항상 /END_OF_FILE 한정자를 사용해야 합니다. 그렇지 않으면 OpenVMS RMS(Record Management Services)에서 end-of-file을 나타내는 오류 조건이 반환될 때 명령 인터프리터에서 현재 ON 명령에 의해 지정된 오류 조치를 수행합니다. 예를 들어, OpenVMS RMS에서 %RMS-E-EOF 오류 상태를 반환하면 명령 프로시저에서 자체적으로 오류를 처리하지 않는 한 그 프로시저는 종료됩니다.

/INDEX 및 /KEY 한정자 사용

인덱스된 순차적 파일에서 레코드를 임의로 읽으려면 READ 명령을 /INDEX 및 /KEY 한정자와 함께 사용합니다. 이 두 한정자는 인덱스에서 지정된 키를 찾아 그 키와 연관된 레코드를 반환함으로써 파일에서 레코드를 읽도록 지정합니다. 인덱스를 지정하지 않으면 기본 인덱스(0)가 사용됩니다.

레코드를 임의로 읽은 후 READ 명령을 /KEY 또는 /INDEX 한정자 없이 사용하여 파일의 나머지 부분을 순차적으로 읽을 수 있습니다.

/DELETE 한정자 사용

READ 명령과 함께 /DELETE 한정자를 사용하여 인덱스된 순차적 파일에서 레코드를 삭제할 수 있습니다. /DELETE 한정자를 사용하면 레코드를 읽은 후 그 레코드가 파일에서 삭제됩니다. 주어진 키에 의해 지정된 레코드를 삭제하려면 /DELETE 한정자를 /INDEX 및 /KEY 한정자와 함께 사용합니다.

/DELETE, /INDEX 및 /KEY 한정자에 대한 자세한 내용은 *OpenVMS DCL Dictionary*의 READ 명령에 대한 설명을 참조하십시오.

Close 명령 사용

CLOSE 명령을 사용하여 파일을 닫고 **OPEN** 명령으로 만들어진 논리 이름의 할당을 취소합니다. 명령 프로시저를 종료하기 전에 명령 프로시저에서 연 모든 파일을 닫아야 합니다. 열린 파일을 닫지 않으면 명령 프로시저를 종료할 때 파일이 열린 채로 남게 되고 그 열린 파일에 할당된 논리 이름이 프로세스 논리 이름 테이블에서 삭제되지 않습니다.

다음 보기에서 **CLOSE** 명령은 **STATS.DAT** 파일을 닫고 논리 이름인 **INFILE**의 할당을 취소합니다.

```
$ OPEN INFILE DISK4:[MURPHY]STATS.DAT
.
.
.
$ CLOSE INFILE
```

파일 수정

이 절에서는 다음 세 가지 파일 수정 방법에 대해 설명합니다.

- 레코드 업데이트
- 새 출력 파일 만들기
- 레코드 추가

레코드 업데이트

레코드를 업데이트하여 수정할 때는 파일에 있는 적은 수의 레코드에 대해 간단한 사항을 변경할 수 있습니다. 이 방법으로는 파일의 레코드 크기나 레코드 수를 변경할 수 없으므로 데이터 파일과 같이 형식화된 레코드가 있는 파일의 경우에만 이 방법을 사용하십시오.

파일에서 간단한 사항을 변경하려면 다음 절차를 사용하십시오.

표 14-6

단계	작업
1	읽기와 쓰기가 가능하도록 파일을 엽니다.
2	READ 명령을 사용하여 수정하려는 레코드에 도달할 때까지 파일을 읽습니다.
3	레코드를 수정합니다. 순차적 파일에서는 이 레코드 텍스트의 크기가 원래 레코드와 꼭 같아야 합니다. 수정한 레코드의 텍스트가 더 짧은 경우에는 원래 레코드와 같은 길이가 될 때까지 끝에 공백을 추가하십시오. 수정한 레코드 텍스트가 더 긴 경우에는 새 파일을 만들어야 합니다.
4	WRITE/UPDATE 명령을 사용하여 수정한 레코드를 파일에 다시 씁니다.
5	변경하려는 모든 레코드를 변경할 때까지 2 단계와 4 단계를 반복합니다.
6	CLOSE 명령을 사용하여 파일을 닫습니다. 레코드가 변경되었지만 파일의 버전 번호는 파일을 닫은 후에도 달라지지 않습니다.

다음 명령 프로시저는 레코드를 개별적으로 읽고 업데이트하여 순차적 파일을 수정하는 방법을 보여 줍니다.

```
$! Open STATS.DAT and assign it the logical name FILE
$!
$ OPEN/READ/WRITE FILE DISK4:[MURPHY]STATS.DAT
$ BEGIN_LOOP:
$! Read the next record from FILE into the symbol RECORD
$   READ/END_OF_FILE=END_LOOP FILE RECORD
$! Display the record and see if the user wants to change it
$! If yes, get the new record.  If no, repeat loop
$!
$   PROMPT:
$       WRITE SYS$OUTPUT RECORD
$       INQUIRE/NOPUNCTUATION OK "Change? Y or N [Y] "
$       IF OK .EQS. "N" THEN GOTO BEGIN_LOOP
$       INQUIRE NEW_RECORD "New record"
$! Compare the old and new records
$! If old record is shorter than new record, issue an
$! error message.  If old record and new record are the
$! same length, write the record.  Otherwise pad the new
$! record with spaces so it is correct length
$!
$       OLD_LEN = F$LENGTH(RECORD)
$       NEW_LEN = F$LENGTH(NEW_RECORD)
$       IF OLD_LEN .LT. NEW_LEN THEN GOTO ERROR
$       IF OLD_LEN .EQ. NEW_LEN THEN GOTO WRITE_RECORD
$       SPACES = "          "
$       PAD = F$EXTRACT(0,OLD_LEN-NEW_LEN,SPACES)
$       NEW_RECORD = NEW_RECORD + PAD
$!
$   WRITE_RECORD:
$       WRITE/UPDATE FILE NEW_RECORD
$       GOTO BEGIN_LOOP
$!
$   ERROR:
$       WRITE SYS$OUTPUT "Error -- New record is too long"
$       GOTO PROMPT
$!
$   END_LOOP:
$       CLOSE FILE
$       EXIT
```


시스템에서는 레코드를 터미널에 표시하고 사용자에게 수정할 것인지 묻습니다. 레코드를 수정하기로 선택하는 경우에는 새 레코드를 터미널에서 읽어 원래 레코드와 길이를 비교합니다. 원래 레코드의 길이가 더 길면 새 레코드에 공백을 추가하여 같은 크기로 만듭니다. 원래 레코드가 더 짧으면 오류 메시지가 표시되고 새 레코드를 위한 프롬프트가 다시 나타납니다.

새 출력 파일 만들기

파일의 많은 부분을 변경하려면 해당 파일을 읽기 액세스를 위해 열고 새 파일을 쓰기 액세스를 위해 엽니다. 새 출력 파일을 만드는 것이므로 레코드 크기를 변경하거나 레코드를 추가, 삭제 또는 삽입할 수 있습니다.

OPEN/WRITE 명령을 사용하여 새 파일을 쓰기 액세스를 위해 엽니다. 새 파일은 원래 파일 이름과 같은 이름을 사용할 수 있으며 버전 번호는 이전 파일보다 하나 위로 합니다.

주

올바른 파일을 열어서 읽도록 하려면 쓰기 액세스를 위한 새 버전을 열기 전에 읽기 액세스를 위해 기존 파일을 먼저 열어야 합니다.

수정할 수 있는 파일을 만들려면 다음 절차를 사용하십시오.

표 14-7

단계	작업
1	읽기 액세스를 위해 파일을 엽니다. 이 파일은 수정하려는 입력 파일입니다.
2	쓰기 액세스를 위해 새 파일을 엽니다. 이 파일은 새로 만드는 출력 파일입니다. 출력 파일에 입력 파일과 같은 이름을 사용하는 경우에는 출력 파일의 버전 번호가 입력 파일보다 하나 위가 됩니다.
3	READ 명령을 사용하여 수정하려는 파일의 각 레코드를 읽습니다. 원래 파일에서 각 레코드를 읽을 때 처리 방법을 결정합니다.
4	레코드를 끝까지 읽고 처리합니다.
5	CLOSE 명령을 사용하여 입력 파일과 출력 파일을 모두 닫습니다.

다음 표에서 RECORD 기호는 원래 파일에서의 레코드 읽기를 포함합니다.

표 14-8

레코드 상태	처리 방법
Not changed	같은 기호를 새 파일에 씁니다.
Changed	INQUIRE 명령을 사용하여 다른 레코드를 기호에 읽어들이고 다음 수정한 기호를 새 파일에 씁니다.
Deleted	기호를 새 파일에 쓰지 않습니다.
Inserted	루프를 사용하여 레코드를 기호로 읽어들이고 그 기호를 새 파일에 씁니다.

보기: 레코드 수정

- 다음 보기에서는 NEW_FILE 기호가 새 파일에 쓰여집니다.

```
$ ! No change
$ WRITE NEW_FILE RECORD
```

- 다음 보기에서는 INQUIRE 명령을 사용하여 수정한 기호를 새 파일에 씁니다.

```
$ ! Change
$ INQUIRE NEW_RECORD "New record"
$ WRITE NEW_FILE NEW_RECORD
```

- 다음 보기에서는 루프를 사용하여 기호를 새 파일에 씁니다.

```
$ ! Insertion
$LOOP:
$ !Get new records to insert
$ INQUIRE NEW_RECORD "New record"
$ IF RECORD .EQS. "" THEN GOTO END_LOOP
$ WRITE NEW_FILE NEW_RECORD
$ GOTO LOOP
$END_LOOP:
```

보기: 출력 파일 만들기

다음 보기는 입력 파일에서 레코드를 읽고 처리하여 출력 파일에 복사하는 명령 프로시저를 보여 줍니다.

```
$! Open STATS.DAT for reading and assign it
$! the logical name INFILE
$! Open a new version of STATS.DAT for writing
$! and assign it the logical name OUTFILE
$!
$ OPEN/READ INFILE DISK4:[MURPHY]STATS.DAT
$ OPEN/WRITE OUTFILE DISK4:[MURPHY]STATS.DAT
$!
$ BEGIN_LOOP:
$! Read the next record from INFILE into the symbol RECORD
$!
$      READ/END_OF_FILE=END_LOOP INFILE RECORD
$! Display the record and see if the user wants to change it
$! If yes, get the new record
$! If no, write record directly to OUTFILE
$!
$      PROMPT:
$          WRITE SYS$OUTPUT RECORD
$          INQUIRE/NOPUNCTUATION OK "Change? Y or N [Y] "
$          IF OK .EQS. "N" THEN GOTO WRITE_RECORD
$          INQUIRE RECORD "New record"$!
$      WRITE_RECORD:
$          WRITE OUTFILE RECORD
$          GOTO BEGIN_LOOP
$!
$! Close input and output files
$      END_LOOP:
$          CLOSE INFILE
$          CLOSE OUTFILE
$          EXIT
```

파일에 레코드 추가

기존 파일의 끝에 레코드를 추가하려면 다음 명령 프로시저(**OPEN/APPEND** 명령)를 사용하십시오.

표 14-9

단계	작업
1	OPEN 명령을 /APPEND 한정자와 함께 사용하여 레코드 포인터가 파일 끝에 위치하도록 합니다. /APPEND 한정자를 사용하면 파일의 새 버전이 만들어지지 않습니다.
2	WRITE 명령을 사용하여 새 데이터 레코드를 씁니다.
3	레코드를 계속 추가합니다.
4	CLOSE 명령을 사용하여 파일을 닫습니다.

다음 명령 프로시저는 **STATS.DAT** 파일 끝에 레코드를 추가합니다.

```
$! Open STATS.DAT to append files and assign
$! it the logical name FILE
$!
$ OPEN/APPEND FILE DISK4:[MURPHY]STATS.DAT
$!
$ BEGIN_LOOP:
$! Obtain record to be appended and place this
$! record in the symbol RECORD
$!
$   PROMPT:
$       INQUIRE RECORD -
$           "Enter new record (press RET to quit) "
$       IF RECORD .EQS. " " THEN GOTO END_LOOP
$! Write record to FILE
$!
$       WRITE FILE RECORD
$       GOTO BEGIN_LOOP
$!
$! Close FILE and exit
$!
$   END_LOOP:
$       CLOSE FILE
$       EXIT
```

파일 I/O 오류 처리

시스템 오류 메시지가 표시되지 않도록 하고 지정된 레이블로 제어를 전달하려면 **/ERROR** 한정자를 **OPEN**, **READ** 또는 **WRITE** 명령과 함께 사용합니다. 입력 또는 출력 작업 중에 오류가 발생하면 **/ERROR** 한정자는 **READ** 명령에 대한 **/END_OF_FILE** 한정자 외의 모든 다른 오류 제어 방법을 무시합니다.

다음 보기에서는 **/ERROR** 한정자를 **OPEN** 명령과 함께 사용합니다.

```
$ OPEN/READ/ERROR=CHECK FILE CONTINGEN.DOC
.
.
.
$ CHECK:
$ WRITE SYS$OUTPUT "Error opening file"
```

OPEN 명령은 **CONTINGEN.DOC** 파일을 읽기 위해 열도록 요청합니다. 파일을 열 수 없는 경우(예: 파일이 없는 경우)에는 오류 상태를 반환하고 **CHECK** 레이블로 제어를 전달합니다.

/ERROR 한정자에 의해 지정된 오류 경로는 명령 수준에 대한 현재의 **ON** 상태를 무시합니다. 오류가 발생하고 대상 레이블에 성공적으로 제어가 전달되면 예약된 글로벌 기호인 **\$STATUS**에 오류 코드가 보관됩니다. 오류 처리 루틴에 **F\$MESSAGE** 렉시칼 함수를 사용하여 **\$STATUS**의 메시지를 표시할 수 있습니다.

다음 보기에서는 렉시칼 함수인 **F\$MESSAGE**를 사용하여 **F\$STATUS** 렉시칼의 내용을 표시합니다.

```
$ OPEN/READ/ERROR=CHECK FILE 'P1'
.
.
.
$ CHECK:
$ ERR_MESSAGE = F$MESSAGE($STATUS)
$ WRITE SYS$OUTPUT "Error opening file: ",P1
$ WRITE SYS$OUTPUT ERR_MESSAGE
.
.
.
```

기본 오류 조치

사용자가 오류 조치를 지정하지 않고 **OPEN**, **READ**, **WRITE** 또는 **CLOSE** 명령을 사용하는 동안 오류가 발생할 경우에는 현재 **ON** 명령 조치가 취해집니다.

READ 명령 실행 시 **end-of-file** 메시지를 표시하는 오류 조치는 다음과 같은 경로로 결정된 것입니다.

- **/END_OF_FILE** 한정자를 사용한 경우에는 지정된 레이블로 제어가 전달됩니다.
- **/END_OF_FILE** 한정자를 사용하지 않은 경우에는 **/ERROR** 한정자를 사용하여 지정된 레이블로 제어가 전달됩니다.
- **/END_OF_FILE**이나 **/ERROR** 한정자를 지정하지 않은 경우에는 현재 **ON** 명령 조치가 취해집니다.

실행 흐름 제어 방법

명령 프로시저의 정상적인 실행 흐름은 순차적입니다. 즉, 프로시저의 명령들이 파일 끝에 도달할 때까지 순서대로 실행됩니다. 그러나, 특정 명령문 실행 여부나 프로시저를 계속 실행할 조건을 제어할 수 있습니다.

다음 절에서 설명하는 내용은 아래와 같습니다.

- 실행 흐름을 제어 또는 변경하는 데 사용하는 DCL 명령
 - IF, THEN, ELSE
 - GOTO
 - GOSUB
 - CALL
- 명령 블록 사용
- Case 문 작성
- 루프 작성

IF 명령 사용

IF 명령을 사용하여 표현식 값을 검사하고 표현식 결과가 참(true)일 때 명령이나 명령 블록을 실행합니다. 표현식 결과가 거짓(false)이면 다음 중 한 가지 결과가 발생합니다.

- THEN 명령 뒤에 하나의 명령이 나오면 그 명령이 실행되지 않고 그 다음에 나오는 명령이 실행됩니다.
- THEN 명령 뒤에 명령 블록이 나오고 ELSE 명령이 지정되어 있지 않으면 ENDIF 명령 바로 뒤에 나오는 명령이 실행됩니다.
- ELSE 명령이 지정되어 있으면 ELSE 명령 뒤에 나오는 명령이나 명령 블록이 실행됩니다.

DCL에서 제공하는 IF 명령은 두 가지 형식이 있습니다. 첫 번째 형식은 13장에서 설명한 것처럼 IF 명령에 지정된 표현식이 참일 때 단일 명령을 실행하는 것입니다.

두 번째 **IF** 형식은 블록 구조 형식입니다. 블록 구조의 **IF** 명령은 지정된 표현식이 참이고 **ELSE** 문이 있는 경우 두 개 이상의 명령을 실행합니다. **ELSE** 문은 표현식이 거짓인 경우 하나 이상의 명령을 실행합니다.

THEN 명령 사용

표현식이 참일 때 두 개 이상의 명령을 실행하려면 **THEN** 명령을 동사(앞에 달러 기호가 있는 **DCL** 명령)로 지정하고 **ENDIF** 문을 사용하여 결과 블록 구조 문을 종료합니다.

다음 보기에서는 **THEN** 문이 동사로 사용됩니다.

```
$ IF expression
$   THEN
$       command
$       command
$
$   .
$   .
$   .
$ ENDIF
```

ELSE 명령 사용

표현식이 거짓일 때 한 개 이상의 명령을 실행하려면 **ELSE** 문을 동사로 지정하고 **ENDIF** 문을 사용하여 결과 블록 구조 문을 종료합니다.

다음 보기에서는 **ELSE** 명령이 동사로 사용됩니다.

```
$ IF expression
$   THEN
$       command
$       command
$
$   .
$   .
$   .
$   ELSE
$       command
$       command
$
$   .
$   .
$   .
$ ENDIF
```


명령 블록 사용

명령 블록은 같은 명령 프로시저에 명령을 둘 것인지 아니면 다른 명령 프로시저에 두고 거기서 실행할 것인지에 따라 몇 가지 방법으로 실행할 수 있습니다. 다음 지침을 따르십시오.

- 명령을 그 명령 프로시저에 그대로 두는 경우에는 다음 보기처럼 **THEN** 문 다음에 넣습니다.

```
$ IF condition
$ THEN  command
        command
.
.
.
$ ENDIF
```

- 명령을 별개의 프로시저에 두려면 다음 보기처럼 그 명령 프로시저에 대한 호출을 **THEN** 문의 일부로 만듭니다.

```
$ IF condition
$ THEN @command_procedure
$ ELSE command
$      command
$ ENDIF
```

- 다음 보기처럼 블록 구조가 아닌 **IF** 형식을 지정하고 지정된 조건에 부합되면 레이블 영역으로 가도록 할 수 있습니다.

```
$ IF not condition THEN GOTO END_LABEL
.
.
.
$END_LABEL:
```

IF 표현식의 결과가 참이면 **THEN** 명령 다음에 오는 명령 블록을 실행할 수 있습니다. 명령 블록을 사용할 때는 **THEN** 명령을 **IF** 명령 뒤의 행에 첫 번째 명령으로 둡니다.

다음 보기에서는 두 개의 **SET TERMINAL** 명령이 실행되고 **F\$MODE**가 “**INTERACTIVE**” 일 때 **PROCEED** 레이블로 제어가 전달됩니다. **F\$MODE**가 “**INTERACTIVE**”가 아닐 때는 프로시저가 종료됩니다.

```
$ IF F$MODE () .EQS. "INTERACTIVE"$ THEN
$   SET TERMINAL/DEVICE=VT320
$   SET TERMINAL/WIDTH=132
$   GOTO PROCEED
$ ENDIF
$ EXIT
$PROCEED:
```

다음 보기는 **IF** 명령을 **ELSE** 명령과 함께 사용할 때 명령 블록을 사용하는 방법을 보여 줍니다.

```
$ INQUIRE DEV "Device to check"
$ IF F$GETDVI(DEV, "EXISTS")
$ THEN
$   WRITE SYS$OUTPUT "The device exists."
$   SHOW DEVICE 'DEV'
$   SET DEVICE/ERROR_LOGGING 'DEV'
$ ELSE
$   WRITE SYS$OUTPUT "The device does not exist."
$   WRITE SYS$OUTPUT "Error logging has not been enabled."
$ ENDIF
$ EXIT
```

조건이 참이면 **SYS\$OUTPUT**에 메시지가 쓰여지고 **SHOW DEVICE**와 **SET DEVICE** 명령이 실행됩니다. 조건이 참이 아니면 **SYS\$OUTPUT**에 두 개의 메시지가 쓰여집니다.

IF-THEN-ELSE 구조를 사용할 때는 다음 제한 사항을 지키십시오.

- 중첩된 **IF** 문의 수준은 15개를 넘으면 안됩니다.
- **THEN** 문으로 시작되는 명령 블록은 **ELSE** 문이나 **ENDIF** 문으로 끝납니다.
- **ELSE** 문으로 시작되는 명령 블록은 **ENDIF** 문으로 끝납니다.
- **THEN** 문을 **IF** 문 뒤에 첫번째 실행문으로 포함시킵니다.
- **THEN** 문이나 **ELSE** 문을 포함하는 행에 레이블을 지정하지 않도록 합니다. 그러나 **ENDIF** 문을 포함하는 행에는 레이블을 지정할 수 있습니다. 명령 블록 내에서 분기할 수 있지만 다른 명령 블록 중간으로 분기하는 것은 삼가하는 것이 좋습니다.

참(True) 표현식

IF 명령 뒤의 표현식은 논리, 산술 또는 문자열 연산자로 구분된 한 개 이상의 숫자 상수, 문자열 리터럴, 기호 이름, 또는 렉시칼 함수로 이루어질 수 있습니다. 다음 중 한 값을 가지는 표현식은 참입니다.

- 홀수 정수 값
- Y, y, T, 또는 t로 시작하는 문자열 값
- 홀수 값 정수인 숫자를 포함하는 문자열 값(예: 문자열 "27")

거짓(False) 표현식

다음 중 한 값을 가지는 표현식은 거짓입니다.

- 짝수 정수 값
- Y, y, T, 또는 t 이외의 글자로 시작하는 문자열 값
- 짝수 값 정수인 숫자를 포함하는 문자열(예: 문자열 "28")

표현식 작성

IF 명령을 위한 표현식을 작성할 때는 다음 규칙을 따르십시오.

- IF 문에 기호를 사용하면 그 값이 자동으로 대체됩니다. 반복 변환을 강제 실행하지 않는 한 어포스트로피(')를 대체 연산자로 사용하지 않습니다.
- 문자열 비교 연산자는 S자로 끝납니다. 예를 들면, .EQS., .LTS., 및 .GTS와 같은 연산자를 사용하여 문자열을 비교합니다. 이에 비해 .EQ., .LT., 및 .GT. 연산자는 정수를 비교하는 데 사용됩니다.
- 두 문자열이 같은 지 검사할 때는 대/소문자 사용이 같아야 일치 여부를 찾을 수 있습니다. 즉, 문자열 "COPY"는 문자열 "copy"나 "CoPy"와 다릅니다.

다음 보기는 IF 명령과 함께 사용할 수 있는 표현식을 보여 줍니다. 다른 보기는 *OpenVMS DCL Dictionary*의 IF 명령에 대한 설명을 참조하십시오.

첫 번째 보기에서는 논리 연산자를 사용하며 THEN 문 다음에 오는 하나의 명령만 실행합니다. CONT 기호가 참이 아니면 프로시저가 종료됩니다.

```
$ INQUIRE CONT "Do you want to continue [Y/N]"
$ IF .NOT. CONT THEN EXIT
.
.
.
```

다음 보기에서는 **IF** 표현식에 하나의 기호와 하나의 레이블을 사용합니다.

```
$ INQUIRE CHANGE "Do you want to change the record [Y/N]"
$ IF CHANGE THEN GOTO GET_CHANGE
.
.
.
$ GET_CHANGE:
.
.
.
```

CHANGE 기호가 참이면 **GET_CHANGE** 레이블로 제어가 전달됩니다.

그렇지 않으면 **IF** 문 뒤에 나오는 명령이 실행됩니다.

다음 보기에서는 두 개의 다른 **IF** 명령을 보여 줍니다.

```
$ COUNT = 0
$ LOOP:
$   COUNT = COUNT + 1
$   IF COUNT .EQ. 9 THEN EXIT
$   IF P'COUNT' .EQS. "" THEN EXIT
.
.
.
$ GOTO LOOP
```

첫 번째 **IF** 명령은 두 개의 정수를 비교하고 두 번째 **IF** 명령은 두 개의 문자열을 비교합니다. 정수 비교에는 **.EQ.** 연산자를, 문자열 비교에는 **.EQS.** 연산자를 사용합니다.

먼저 **COUNT** 값을 정수 **9**와 비교하여 값이 같으면 프로시저를 종료하고 값이 다르면 프로시저를 계속합니다. 8개(최대 허용 수)의 매개 변수를 처리한 후 루프를 종료합니다.

두 번째 **IF** 표현식에서는 **P'COUNT'** 기호의 문자열 값을 **null** 문자열과 비교하여 기호가 정의되어 있지 않은지 확인합니다. **COUNT** 기호의 반복 대체를 강제 실행하려면 어포스트로피를 사용해야 합니다. 예를 들어, **COUNT**가 2이면 첫 번째 변환 결과는 **P2**이며 **P2**의 값이 문자열 비교에 사용됩니다.

IF 문의 결과가 참일 때 별개의 명령 프로시저를 실행할 수도 있습니다. 다음 보기에서는 **IF** 표현식의 결과가 참일 때 **EXIT_ROUTINE.COM** 명령 프로시저를 실행합니다.

```
$ GET_COMMAND_LOOP:
$   INQUIRE COMMAND -
$     "Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"
$   IF COMMAND .EQS. "EXIT" THEN @EXIT_ROUTINE
```

GOTO 명령 사용

GOTO 명령은 명령 프로시저에서 레이블 있는 행으로 제어를 전달합니다. 레이블 사용에 대한 자세한 내용은 13장을 참조하십시오. GOTO 명령은 프로시저를 정방향 또는 역방향으로 분기시키는 THEN 문에서 특히 유용합니다. 예를 들어, 명령 프로시저에서 매개 변수를 사용할 때 프로시저 시작에서 매개 변수를 테스트하고 해당 레이블로 분기할 수 있습니다.

GOTO 또는 GOSUB 명령의 대상 레이블은 별개의 IF-THEN-ELSE 구조나 별개의 서브루틴 안에 있을 수 없습니다

다음 보기에서 IF 명령은 P1이 null 문자열이 아닌지 확인합니다.

```
$ IF P1 .NES. "" THEN GOTO OKAY
$ INQUIRE P1 "Enter file spec"$ OKAY:
$ PRINT/COPIES=10 'P1'
.
.
.
```

P1이 null 문자열이면 GOTO 명령은 실행되지 않고 INQUIRE 명령이 매개 변수 값을 위한 프롬프트를 표시합니다. 그렇지 않으면 INQUIRE 명령을 건너 뛰어 분기됩니다. 어느 경우든 OKAY 레이블이 있는 행 다음의 PRINT 명령이 실행됩니다.

다음 보기에서 GOTO 명령은 그 대상 (TEST_1)이 IF-THEN 구조 내에 있기 때문에 오류 메시지를 반환합니다.

```
$ GOTO TEST_1
$ EXIT
$ IF 1.EQ.1
$     THEN WRITE SYS$OUTPUT "What are we doing here?"
$ TEST_1:
$     WRITE SYS$OUTPUT "Got to the label"
$ ENDIF
$ EXIT
```

재실행 방지

GOTO 명령을 사용하여 성공적으로 완료된 작업의 일부를 재실행하는 것을 방지할 수 있습니다. 이를 수행하려면 다음 단계를 따르십시오.

표 14-10

단계	작업
1	프로시저의 각 가능한 시작 지점을 레이블로 시작하십시오.
2	레이블 다음에 SET RESTART_VALUE = label-name 명령을 사용하여 재시작 지점을 그 레이블로 설정합니다. SET RESTART_VALUE = label-name 명령이 실행된 후 일괄 처리 작업이 중단된 경우 일괄 처리 작업이 다시 시작될 때 해당 레이블 이름이 글로벌 기호인 BATCH\$RESTART에 할당됩니다.
3	프로시저 시작에서 \$RESTART 기호의 값을 테스트합니다. \$RESTART가 참이면 BATCH\$RESTART를 전달 레이블로 사용하여 GOTO 문을 실행합니다.

\$RESTART 글로벌 기호

\$RESTART는 시스템에서 유지 관리하는 예약된 글로벌 기호입니다. \$RESTART는 일괄 처리 작업이 중단되었다가 다시 시작된 경우에는 참이고 그렇지 않은 경우에는 거짓입니다. 예약된 글로벌 기호인 \$RESTART는 삭제할 수 없습니다.

명령 프로시저에 SET RESTART_VALUE 명령이 있지만 그 작업을 전부 재실행하지 않으려면 SET ENTRY/NOCHECKPOINT 명령을 입력하여 글로벌 기호인

BATCH\$RESTART를 삭제합니다. 중단되었던 작업을 다시 시작할 경우에는 중단되었던 섹션에서 작업이 실행되기 시작합니다.

다음 명령 프로시저는 일괄 처리 작업에서 재시작 값을 사용하는 방법을 보여 줍니다.

```
$ ! Set default to the directory containing
$ ! the file to be updated and sorted
$ SET DEFAULT DISK1:[ACCOUNTS.DATA84]
$
$ ! Check for restarting
$ IF $RESTART THEN GOTO 'BATCH$RESTART'
$
$ UPDATE_FILE:
$ SET RESTART_VALUE = UPDATE_FILE
.
.
.
$ SORT_FILE:
$ SET RESTART_VALUE = SORT_FILE
.
.
.
EXIT
```

이 명령 프로시저를 재시작할 수 있는 일괄 처리 작업으로 제출하려면 작업을 제출할 때 **SUBMIT** 명령에 **/RESTART** 한정자를 사용합니다. 중단되었던 작업은 중단되었던 섹션에서 실행되기 시작하므로 이 작업이 **SORT_FILE** 루틴 중에 중단된 경우에는 재시작할 때 **SORT_FILE** 레이블에서 실행을 시작합니다.

시스템 실패 시에는 대부분의 프로세스 환경이 유지되지 않습니다. 시스템 실패 시에도 유지 관리되는 기호는 **\$RESTART**와 **BATCH\$RESTART** 뿐입니다.

그러므로, 각 **SET RESTART_VALUE** 명령 다음이나 **\$RESTART**가 참인 경우 실행되는 **THEN** 블록에 사용된 기호나 프로세스 논리 이름은 재정의해야 합니다. 기호와 논리 이름을 **THEN** 블록에서 정의하는 경우에는 **GOTO 'BATCH\$RESTART'** 명령이 **THEN** 블록의 마지막 명령이어야 합니다.

GOSUB 및 RETURN 명령 사용

GOSUB 명령은 명령 프로시저에서 레이블 있는 서브 루틴으로 제어를 전달합니다. 명령 프로시저에 레이블이 없는 경우에는 프로시저 실행이 계속될 수 없으며 강제 종료됩니다. 레이블에 대한 자세한 정보는 13장을 참조하십시오. 프로시저 수준 당 16번까지 **GOSUB** 명령을 중첩할 수 있습니다.

GOSUB 명령은 로컬 서브 루틴 호출이므로 새로운 프로시저 수준을 만들지 않습니다. 결과적으로 현재 명령 수준에 정의된 모든 레이블과 로컬 기호를 **GOSUB**를 사용하여 호출한 서브 루틴에서 사용할 수 있습니다.

RETURN 명령은 서브 루틴을 종료하고 **GOSUB** 명령 뒤에 나오는 명령으로 제어를 반환합니다. **RETURN** 명령을 사용하여 DCL에서 **\$STATUS**에 할당된 값을 무시하는 **\$STATUS** 값을 서브 루틴 끝에 지정할 수 있습니다. 이 값은 0부터 4 사이의 정수이거나 이에 해당하는 표현식이어야 합니다. **\$STATUS** 값을 사용자가 지정하는 경우 DCL에서는 이 값을 조건 코드로 해석합니다. **\$STATUS** 값을 지정하지 않으면 **\$STATUS**의 현재 값이 저장됩니다.

다음 보기는 **GOSUB** 명령을 사용하여 서브 루틴에 제어를 전달하는 방법을 보여 줍니다.

```
$!
$! GOSUB.COM
$!
$ SHOW TIME
$ GOSUB TEST1 [1]
$ WRITE SYS$OUTPUT "GOSUB level 1 has completed successfully."
$ SHOW TIME
$ EXIT
$!
$! TEST1 GOSUB definition
$!
$ TEST1:
$     WRITE SYS$OUTPUT "This is GOSUB level 1."
$     GOSUB TEST2 [2]
$     RETURN %X1 [3]
$!
$! TEST2 GOSUB definition
$!
$ TEST2:
$     WRITE SYS$OUTPUT "This is GOSUB level 2."$     WAIT 00:00:02
$     RETURN [4]
```

위의 보기에서 주목할 사항은 다음과 같습니다.

- [1] 첫 번째 **GOSUB** 명령은 **TEST1** 레이블이 붙은 서브 루틴으로 제어를 전달합니다.
- [2] 프로시저는 서브 루틴 **TEST1**의 명령을 실행하여 **TEST2** 레이블이 있는 서브 루틴으로 분기합니다.
- [3] 서브 루틴 **TEST1**의 **RETURN** 명령은 주 명령 프로시저로 제어를 반환하고 성공적인 완료를 나타내는 1이라는 값을 **\$STATUS**에 전달합니다.
- [4] 서브 루틴 **TEST2**의 **RETURN** 명령은 서브 루틴 **TEST1**에 제어를 반환합니다. 이 명령이 3번 명령 전에 실행된다는 점에 주목하십시오.

새 명령 수준 만들기

새로운 명령 수준을 만드는 방법은 다음 두 가지입니다.

- 한 명령 프로시저 안에 다른 명령 프로시저를 호출하는 실행 프로시저(@) 명령을 사용하여 명령 프로시저를 중첩시키는 방법(13장 참조)
- CALL 명령을 사용하여 명령 프로시저 내에 있는 서브 루틴을 호출하는 방법

CALL 명령 사용

CALL 명령은 명령 프로시저에서 레이블 있는 서브 루틴으로 제어를 전달하고 새 프로시저 수준을 만듭니다. CALL 명령을 사용하여 단일 파일에서 두 개 이상의 관련된 명령 프로시저를 두어 관리하기 쉽도록 할 수 있습니다. 서브 루틴 레이블에는 고유한 이름을 사용해야 하며 명령 프로시저의 CALL 명령 앞이나 뒤에 둘 수 있습니다. 서브 루틴 레이블을 입력하는 방법은 13장을 참조하십시오.

레이블 외에 최대 8개의 선택 매개 변수를 서브 루틴에 전달할 수 있습니다. 매개 변수에 대한 자세한 내용은 565페이지의 “매개 변수를 사용하여 데이터 전달”을 참조하십시오.

CALL 명령을 사용하기 위한 규칙은 다음과 같습니다.

- SYS\$OUTPUT에 출력을 보냅니다.
- 출력을 서브 루틴에서 파일로 보낼 수 있는 /OUTPUT 한정자가 있습니다.
- 출력 파일에는 기본 파일 형식인 .LIS를 사용합니다.
- 출력 파일 지정에 와일드카드 문자를 사용할 수 없습니다.

CALL 명령 기본값

CALL 명령 사용과 관련된 추가 기본값은 다음과 같습니다.

- CALL 명령을 사용하여 호출한 서브 루틴과 실행 프로시저(@) 명령을 사용하여 호출한 프로시저를 최대 32개의 명령 수준까지 중첩시킬 수 있습니다.
- 바깥쪽 수준에 정의된 로컬 기호는 SET SYMBOL 명령을 사용하여 마스크한 경우가 아니면 안쪽 프로시저나 서브 루틴 수준에서 사용할 수 있습니다. 글로벌 기호는 어느 명령 수준에서나 사용할 수 있습니다.
- 레이블은 정의된 수준에 대해서만 유효합니다.

서브 루틴 시작 및 종료

CALL 서브 루틴의 시작과 끝은 SUBROUTINE과 ENDSUBROUTINE 명령으로 정의합니다. 서브 루틴에 대한 진입 지점을 정의하는 레이블은 SUBROUTINE 명령 바로 앞에 나옵니다. ENDSUBROUTINE 명령 바로 앞에 EXIT 명령을 넣을 수는 있지만 서브 루틴을 종료할 필요는 없습니다. ENDSUBROUTINE 명령은 서브 루틴을 종료하고 CALL 명령 바로 뒤에 나오는 명령행으로 제어를 전달합니다.

서브 루틴의 명령행은 그 서브 루틴이 CALL 명령을 사용하여 호출된 경우에만 실행됩니다. 명령 프로시저의 명령행을 차례로 실행하는 동안 명령어 인터프리터는 SUBROUTINE과 ENDSUBROUTINE 명령 사이의 모든 명령을 건너뛵니다.

서브 루틴 진입 지점의 범위와 레이블 참조 사용 범위를 정의할 때는 다음과 같은 제한 사항이 적용됩니다.

- 다른 서브 루틴 안에 정의된 서브 루틴 진입점은 그 서브 루틴에 대해 로컬입니다. 서브 루틴 진입점이 별개의 서브 루틴 블록 내에 있는 경우에는 서브 루틴을 호출할 수 없습니다.
- 서브 루틴 진입점이 IF-THEN-ELSE 블록 내에 있는 경우에는 IF-THEN-ELSE 블록 밖에서 이 서브 루틴을 호출할 수 없습니다.
- 모든 SUBROUTINE 명령에는 서브 루틴을 끝내는 ENDSUBROUTINE 명령이 있어야 합니다.

다음 보기에서는 CALL BAR 명령이 MAIN 서브 루틴 밖에 있기 때문에 호출이 유효하지 않습니다.

```
$ CALL BAR
$
$ MAIN: SUBROUTINE
$
$     BAR: SUBROUTINE
$     ENDSUBROUTINE
$
$ ENDSUBROUTINE
```

이 CALL 명령이 제대로 이루어지려면 SUBROUTINE과 ENDSUBROUTINE 지점 내에 있어야 합니다.

다음 보기의 호출은 **IF-THEN-ELSE** 블록 안에 있기 때문에 허용되지 않습니다.

```
$ IF 1
$ THEN
$   BOB:SUBROUTINE
$   ENDSUBROUTINE
$ ENDIF
$ CALL BOB
```

다음 보기에는 **SUB1**와 **SUB2**라는 이름의 두 서브 루틴이 포함되어 있습니다. 서브 루틴은 **CALL** 명령을 사용하여 호출하기까지는 실행되지 않습니다.

```
$
$! CALL.COM
$
$! Define subroutine SUB1.
$!
$ SUB1: SUBROUTINE
$   .
$   .
$   .
$   CALL SUB2!Invoke SUB2 from within SUB1.
$   .
$   .
$   .
$   @FILE!Invoke another command procedure file.
$   .
$   .
$   .
$   EXIT
$ ENDSUBROUTINE!End of SUB1 definition.
$!
$! Define subroutine SUB2.
$!
$ SUB2: SUBROUTINE
$   EXIT
$ ENDSUBROUTINE!End of SUB2 definition.
$!
$! Start of main routine. At this point, both SUB1 and SUB2
$! have been defined but none of the previous commands have
$! been executed.
$!
$ START:
$   CALL/OUTPUT=NAMES.LOG  SUB1 "THIS IS P1"
$   .
$   .
$   .
$   CALL SUB2 "THIS IS P1" "THIS IS P2"
$   .
$   .
$   .
$ EXIT !Exit this command procedure file.
```

CALL 명령은 SUB1 서브 루틴을 호출하고 출력을 NAMES.LOG 파일로 지정합니다. 서브 루틴 SUB1은 서브 루틴 SUB2를 호출합니다. 프로시저에서는 SUB2를 실행하고 실행 프로시저(@) 명령을 사용하여 명령 프로시저인 FILE.COM을 호출합니다. SUB1의 모든 명령이 실행되면 주 프로시저의 CALL 명령이 SUB2를 두 번째로 호출합니다. SUB2 실행이 끝나면 프로시저가 종료됩니다.

Case 문 작성

Case 문은 변수나 표현식의 값에 따라 일련의 명령 블록 중 한 명령 블록을 실행하는 특수한 형태의 조건 코드입니다. 일반적으로 Case 문의 유효값은 각 명령 블록의 처음에 있는 레이블입니다. Case 문은 GOTO 문에서 지정된 값을 대상 레이블로 사용하여 해당 코드 블록에 제어를 전달합니다.

Case 문을 작성하려면 다음과 같이 해야 합니다.

1. 레이블을 나열합니다.
2. Case 문을 작성합니다.
3. 명령 블록을 작성합니다.

레이블 나열

레이블을 나열하려면 기호를 슬래시 또는 구분 기호로 선택한 임의의 문자를 사용하여 구분한 레이블의 목록을 포함하는 문자열과 일치시킵니다. 이 기호 정의는 명령 블록 앞에 있어야 합니다.

다음 보기에서는 COMMAND_LIST 기호가 PURGE, DELETE 및 EXIT 레이블과 동일합니다.

```
$ COMMAND_LIST = "/PURGE/DELETE/EXIT/"
```

Case 문 작성

Case 문을 작성하려면 다음 절차를 따르십시오.

표 14-11

단계	작업
1	INQUIRE 명령을 사용하여 Case 변수의 값을 가져옵니다.
2	F\$LOCATE와 F\$LENGTH를 IF 명령과 함께 사용하여 Case 변수의 값이 유효한 지 여부를 확인합니다.
3	변수가 유효한 경우에는 GOTO 명령을 사용하여 Case 문을 실행하여 해당 코드 블록으로 제어를 전달합니다.
	그렇지 않으면 메시지를 표시하고 종료하거나 다른 Case 값을 요청합니다.

다음 보기에서는 레이블이 전체 명령 이름과 일치됩니다. 그러므로 F\$LOCATE는 구분 기호를 명령 이름 검색에 포함시켜 명령 이름이 축약되지 않도록 합니다.

```
$GET_COMMAND:
$ INQUIRE COMMAND -
  "Command (EXIT,PURGE,DELETE) "
$ IF F$LOCATE ( "/" +COMMAND+ "/" ,COMMAND_LIST) .EQ. -
  F$LENGTH (COMMAND_LIST) THEN GOTO ERROR_1
$ GOTO 'COMMAND'
.
.
.
$error_1:
$ WRITE SYS$OUTPUT "No such command as ''COMMAND'".
$ GOTO GET_COMMAND
```

명령 블록 작성

각 명령 블록에는 한 개 이상의 명령이 포함될 수 있습니다. 각 명령 블록은 고유한 레이블로 시작하고 명령 블록 목록 밖에 있는 레이블에 제어를 전달하여 끝냅니다.

다음 보기에서 각 명령 블록은 고유한 레이블인 **PURGE:**와 **DELETE:**로 시작하며 현재 명령 블록 밖의 **GOTO GET_COMMAND** 레이블에 제어를 전달함으로써 끝납니다.

```
$GET_COMMAND:
.
.
.
$PURGE:
$ INQUIRE FILE
$ PURGE 'FILE'
$ GOTO GET_COMMAND
$ !
$DELETE:
$ INQUIRE FILE
$ DELETE 'FILE'
$ GOTO GET_COMMAND
$ !
$EXIT:
```

루프 작성

명령 블록 처음에서 변수를 테스트하는 루프를 작성할 수 있습니다(13장 참조). 그러나 다음 절차를 따라 루프의 끝에서 종료 변수를 테스트하는 루프를 작성할 수도 있습니다.

표 14-12

단계	작업
1	루프를 시작합니다.
2	루프 본문의 명령을 수행합니다.
3	종료 변수를 변경합니다.
4	종료 변수를 테스트합니다. 조건에 부합되지 않는 경우에는 루프의 처음으로 갑니다.
5	루프를 끝냅니다.

루프 끝에서 종료 변수를 테스트할 때는 종료 변수의 값에 관계 없이 루프 본문의 명령이 최소한 한 번씩 실행된다는 점에 주의하십시오.

다음 보기의 명령 블록은 둘 다 **COMMAND**가 **"EX"** (**EXIT**)와 일치할 때 종료되는 루프를 실행합니다. **F\$EXTRACT**는 **COMMAND**를 처음 두 자만 남기고 자릅니다. 첫 번째 보기에서는 종료 변수인 **COMMAND**를 루프 처음에 테스트하고 두 번째 보기에서는 루프 끝에 테스트합니다.

```
$ ! EXAMPLE 1
$ !
$GET_COMMAND:
$ INQUIRE COMMAND-
  "Command (EXIT,DIRECTORY,TYPE,PURGE,DELETE,COPY) "
$ COMMAND = F$EXTRACT(0,2,COMMAND)
$ IF COMMAND .EQS. "EX" THEN GOTO END_LOOP
.
.
.
$ GOTO GET_COMMAND
$END_LOOP:
```



```

$ ! EXAMPLE 2
$ !
$ GET_COMMAND:
$ INQUIRE COMMAND-
    "Command (EXIT,DIRECTORY,TYPE,PURGE,DELETE,COPY) "$ COMMAND =
    F$EXTRACT(0,2,COMMAND)
.
.
.
$ IF COMMAND .NES. "EX" THEN GOTO GET_COMMAND
$ ! End of loop

```

지정된 횟수 만큼 루프를 수행하려면 계수기를 종료 변수로 사용하십시오. 다음 보기에서는 사용자가 입력한 10개의 파일 이름이 로컬 기호인 **FIL1**, **FIL2**, ..., **FIL10**에 저장됩니다.

```

$ NUM = 1                      ! Set counter
$ LOOP:                        ! Begin loop
$ INQUIRE FIL'NUM' "File"      ! Get file name
$ NUM = NUM + 1                ! Update counter
$ IF NUM .LT. 11 THEN GOTO LOOP ! Test for termination
$ END_LOOP:                    ! End loop
.
.
.

```

다음 보기에서는 계수기를 사용하여 루프를 실행하는 횟수를 제어합니다. 루프는 10회 실행되며 종료 변수는 루프 끝에서 테스트됩니다.

```

$! Obtain 10 file names and store them in the
$! symbols FILE_1 to FILE_10
$!
$ COUNT = 0
$ LOOP:
$     COUNT = COUNT + 1
$     INQUIRE FILE_'COUNT' "File"
$     IF COUNT .LT. 10 THEN GOTO LOOP
$!
$ PROCESS_FILES:
.
.
.

```

COUNT 기호는 루프가 실행되는 횟수를 기록하는 데 사용됩니다. 또한, COUNT를 사용하여 FILE_1, FILE_2 하는 식으로 FILE_10까지 기호 이름을 만듭니다. COUNT의 값은 루프 시작에서 증가되지만 테스트는 루프 끝에서 이루어진다는 점에 주의하십시오. 그러므로 COUNT가 9에서 10으로 증가될 때 루프는 FILE_10 값을 얻어 IF 문에서 거짓 결과를 발견하기 전에 마지막으로 실행됩니다.

알려진 값 순서를 위해 루프를 수행하려면 F\$ELEMENT 렉시칼 함수를 사용하십시오. F\$ELEMENT 렉시칼 함수는 구분 기호를 사용하여 구분한 항목 목록에서 항목을 가져옵니다. 항목 번호, 항목 구분 기호 및 목록을 F\$ELEMENT의 인수로 제공해야 합니다.

F\$ELEMENT 렉시칼 함수 사용 방법에 대한 자세한 내용은 *OpenVMS DCL Dictionary*를 참조하십시오.

다음 보기에서는 CHAP1, CHAP2, CHAP3, CHAP4, CHAPB, 및 CHAPC 파일이 순서대로 처리됩니다.

```
$ FILE_LIST = "1,2,3,A,B,C"$ INDEX = 0
$PROCESS:
$ NUM = F$ELEMENT(INDEX,"",FILE_LIST)
$ IF NUM .EQS. "" THEN GOTO END_LOOP
$ FILE = "CHAP'"NUM'"$ ! process file named by FILE
.
.
.
$ INDEX = INDEX + 1
$ GOTO PROCESS
$END_LOOP:
$ EXIT
```

다음 보기에서 명령 프로시저는 루프를 사용하여 **FILE_LIST** 기호에 나열된 파일을 다른 노드의 디렉토리에 복사합니다.

```
$ FILE_LIST = "CHAP1/CHAP2/CHAP3/CHAP4/CHAP5"
$ NUM = 0
$!
$! Process each file listed in FILE_LIST
$ PROCESS_LOOP:
$     FILE = F$ELEMENT(NUM, "/", FILE_LIST)
$     IF FILE .EQS. "/" THEN GOTO DONE
$     COPY 'FILE'.MEM MORRIS::DISK3:[DOCSET]*.*
$     NUM = NUM + 1
$     GOTO PROCESS_LOOP
$!
$ DONE:
$ WRITE SYS$OUTPUT "Finished copying files."
$ EXIT
```

F\$ELEMENT 렉시칼 함수는 첫 번째로 **CHAP1**, 두 번째로 **CHAP2** 하는 식으로 파일을 반환합니다. 루프를 한 번 거칠 때마다 **NUM** 값이 증가되어 다음 파일 이름을 가져옵니다.

F\$ELEMENT에서 슬래시가 반환되면 **FILE_LIST**의 모든 항목이 처리되었고 루프가 종료됩니다.

PIPE 명령 사용

PIPE 명령은 같은 명령행에서 두 개 이상의 DCL 명령 문자열을 실행합니다. PIPE 명령을 사용하여 명령 파이프라인화, 입/출력 재지정, 조건 및 백그라운드 실행과 같은 UNIX 스타일 명령 처리를 수행할 수 있습니다.

이러한 명령 처리 스타일은 호스트와 대상 시스템에 일정 형태의 파이프라인 명령 구문 분석이 예상되는 인터넷 소프트웨어 개발 및 사용을 지원합니다.

다음 절에서는 PIPE 명령을 사용하여 DCL 명령을 실행하는 방법, PIPE 명령 중단 방법 및 하위 프로세스 성능 향상 방법에 대해 보기를 들어 설명합니다.

PIPE 명령에 대한 자세한 내용은 *OpenVMS DCL Dictionary: N-Z*를 참조하십시오.

하나의 PIPE 명령에 여러 개의 DCL 명령을 지정할 수 있습니다. 지정한 DCL 명령들은 순차적으로 실행됩니다. 다음 형식을 사용하십시오.

```
PIPE command-sequence ; command-sequence [: command-sequences]...
```

조건 명령 실행을 위한 PIPE 명령 사용

명령 순서는 앞에 나온 명령 순서의 실행 결과에 따라 조건부로 실행됩니다. 다음 형식을 사용하십시오.

```
PIPE command-sequence1 && command-sequence2
```

*command-sequence2*는 *command-sequence1*이 성공적으로 실행된 경우에만 실행됩니다.

다음 형식을 사용하는 경우에는 *command-sequence1*이 실패하는 경우에만 *command-sequence2*가 실행됩니다.

```
PIPE command-sequence1 || command-sequence2
```

파이프라인 실행을 위한 PIPE 명령 사용

파이프라인은 파이프에 의해 연결된 일련의 파이프라인 세그먼트 명령입니다. 파이프는 세로 막대(|) 구분 기호로 표시합니다. 파이프라인 세그먼트 명령은 파이프라인에 나타나는 DCL 명령입니다. 파이프는 파이프라인 세그먼트의 SYS\$OUTPUT을 다음 명령의 SYS\$INPUT에 연결합니다. 파이프라인의 형식은 다음과 같습니다.

```
PIPE pipeline-segment-command | pipeline-segment-command [...]
```

각 파이프라인 세그먼트 명령은 다음 파이프라인 세그먼트 명령의 **SYS\$INPUT**에 연결된 **SYS\$OUTPUT**이 있는 별개의 하위 프로세스에서 실행됩니다. 이 하위 프로세스들은 병행하여 실행됩니다. 그러나 첫 번째를 제외한 각 파이프라인 세그먼트 명령이 선행 파이프라인 세그먼트 명령의 표준 출력을 표준 입력으로 읽을 정도로 동기화됩니다. 파이프라인은 마지막 파이프라인 세그먼트 명령이 끝날 때 실행을 완료합니다.

파이프라인에 “필터 어플리케이션”을 사용하는 것은 매우 일반적인 일입니다. 필터 어플리케이션은 **SYS\$INPUT**에서 데이터를 받아 특정한 방법으로 변환하여 **SYS\$OUTPUT**에 쓰는 프로그램입니다.

DCL의 어떤 면은 다음 보기처럼 파이프라인 컨텍스트에서 다른 기능을 합니다.

- **SYS\$COMMAND** 사용

하위 프로세스의 **SYS\$COMMAND**는 명령 프로시저가 관련되지 않는 경우 일반적으로 **SYS\$INPUT**과 동일합니다. 그러나 파이프라인에서 하위 프로세스의 **SYS\$COMMAND**는 파이프라인 세그먼트 명령의 **SYS\$INPUT**인 선행하는 파이프가 아니라 선행 프로세스의 **SYS\$COMMAND**로 설정됩니다.

- 파일 액세스 방법

파이프라인 세그먼트 명령은 **RMS** 순차적 파일 액세스 방법을 사용해서만 파이프에 읽고 쓸 수 있습니다. 특정 **OpenVMS** 유틸리티는 순차적 액세스가 아닌 방법을 사용하여 입력 및 출력 파일에 액세스할 수 있습니다. 파이프라인에서는 이러한 작업이 지원되지 않으므로 다음 보기처럼 실패하게 됩니다.

```
$ PIPE CC/NOOBJ/NOLIS TEST.C | SEARCH SYS$INPUT/WIND=(1,1) "%cc-w-"
```

```
%SEARCH-F-RFAERR, RMS error using RFA access
-RMS-F-RAC, invalid record access mode
```

이 보기에서 **SEARCH** 명령을 위한 **/WINDOW** 한정자는 관련 파일 액세스 방법을 필요로 합니다.

- 기호 대체

DCL에서 기호가 변환되는 순서에 주의하십시오. 기호 대체는 명령 처리의 1 단계 동안에 이루어집니다. 기호를 **PIPE** 명령의 일부로 정의하면 DCL에서는 그 기호가 실제로 정의된 명령을 수행하기 전에 기호를 대체하려고 시도합니다. 기호 대체를 지연하려면 앰퍼샌드(&)를 사용합니다. 자세한 내용은 458페이지의 “기호 약어”를 참조하십시오.

- **SYS\$PIPE** 사용

대부분의 경우 파이프의 입력 사항은 **SYS\$INPUT**에서 데이터를 읽어 가져올 수 있습니다. 그러나 명령 프로시저가 파이프라인 세그먼트 명령으로 호출되면 **SYS\$INPUT**이 그 명령 프로시저 파일로 재지정됩니다. 명령 프로시저 내의 파이프에서 데이터를 가져오려면 논리 **SYS\$PIPE**를 사용할 수 있습니다.

다음 보기는 파이프라인 **DCL** 어플리케이션인 **TEE.COM**의 예입니다.

```
$ ! TEE.COM - command procedure to display/log data flowing through
$ !           a pipeline
$ ! Usage: @TEE log-file
$
$ OPEN/WRITE tee_file 'Pl'
$ LOOP:
$ READ/END_OF_FILE=EXIT SYS$PIPE LINE
$ WRITE SYS$OUTPUT LINE ! Send it out to the next stage of the pipeline
$ WRITE tee_file LINE ! Log output to the log file
$ GOTO LOOP
$ EXIT:
$ CLOSE tee_file
$ EXIT
```

TEE.COM을 사용하려면 다음 **PIPE** 명령을 입력하십시오.

```
$ PIPE SHOW SYSTEM | @TEE showsys.log | SEARCH SYS$INPUT LEF
```

TEE.COM 명령 프로시저는 파이프라인을 통한 데이터 흐름을 기록하는 데 사용됩니다. 이 명령 프로시저는 **SYS\$INPUT** 대신 **SYS\$PIPE**에서 데이터를 읽어들이니다.

- 이미지 확인

파이프라인에서는 **PIPE** 명령이 입력되기 전에 **SET VERIFY=IMAGE** 명령이 실행될 때도 기본적으로 이미지 확인을 사용하지 않습니다. 따라서 파이프라인을 통과해 가는 데이터 레코드가 중복되지 않습니다.

파이프라인에서 이미지 확인을 사용하려면 **SET VERIFY=IMAGE** 명령을 파이프라인 세그먼트 명령 앞에 두어야 합니다. 다음과 같이 서브셸을 사용할 수 있습니다.

```
$ PIPE ... | (SET VERIFY=IMAGE ; ...) | ...
```

서브셸 실행을 위한 PIPE 명령 사용

서브셸은 구분 기호로 구분된 한 개 이상의 명령 순서를 괄호로 묶은 것입니다. 서브셸의 형식은 다음과 같습니다.

PIPE (command-sequence [separator command-sequence]...)

서브셸의 명령 순서는 하위 프로세스 환경에서 실행됩니다. DCL에서는 다음 명령 순서를 실행하기 전에 서브셸이 종료되기를 기다립니다. 구분 기호()는 **SPAWN/WAIT** 명령과 비슷합니다.

PIPE 명령을 이 형식으로 사용할 때는 기호 대체를 주의하여 처리하십시오. 기호를 정의한 후 그 기호에 대한 이후 참조 앞에는 앰퍼샌드(&)를 붙여 기호 대체를 지연합니다. 그렇지 않으면 명령 처리 1 단계 동안 기호 대체가 이루어져 기호 정의를 신뢰할 수 없게 됩니다.

백그라운드 실행을 위한 PIPE 명령 사용

다음 형식을 사용하여 명령 순서를 하위 프로세스 환경에서 실행할 수 있습니다.

PIPE command-sequence [separator command-sequence]... &

DCL에서 명령 순서가 끝날 때까지 기다리지 않습니다. 일단 백그라운드 하위 프로세스가 만들어지면 제어는 DCL로 다시 전달됩니다.

입/출력 재지정을 위한 PIPE 명령 사용

명령 순서를 사용하여 다음과 같이 명령을 실행하는 동안 **SY\$INPUT**, **SY\$OUTPUT** 또는 **SY\$ERROR**를 파일로 재지정할 수 있습니다.

- **SY\$INPUT** 재지정
PIPE command-sequence < redirected-input-file
- **SY\$OUTPUT** 재지정
PIPE command-sequence > redirected-output-file
- **SY\$ERROR** 재지정
PIPE command-sequence 2> redirected-error-file

파이프라인-세그먼트 명령을 사용하여 **SY\$INPUT**, **SY\$OUTPUT** 또는 **SY\$ERROR**를 지정할 수 있습니다. 그러나 **SY\$OUTPUT** 재지정은 마지막 파이프라인 세그먼트 명령에 대해서만 허용되며 **SY\$INPUT** 재지정은 첫 번째 파이프라인 세그먼트 명령에 대해서만 허용됩니다.

PIPE 명령 재지정은 **DEFINE** 또는 **ASSIGN** 명령을 사용하여 만든 것과 다릅니다.

차이점은 다음과 같습니다.

- 재지정은 감독자 모드에서 만들어집니다. 따라서 사용자 모드 어플리케이션과 **DCL** 명령이 모두 재지정의 영향을 받습니다.
- 재지정된 환경은 재지정 구문을 지정하는 명령 순서나 파이프라인 세그먼트 명령에만 적용됩니다. 명령 순서나 파이프라인 세그먼트 명령 실행 후 명령 실행을 계속하기 전에 원래 프로세스 입/출력 환경(예: **SYS\$INPUT**, **SYS\$OUTPUT**, **SYS\$ERROR**)이 복원됩니다.

SYS\$OUTPUT이 재지정되면 명령 순서가 실제로 **SYS\$OUTPUT**에 쓰거나 쓰지 않거나 항상 재지정된 출력 파일이 만들어집니다. 재지정된 출력 파일과 같은 이름의 파일 버전이 이미 존재하는 경우에는 그 파일의 새 버전이 만들어집니다. **DEFINE** 또는 **ASSIGN** 명령을 사용하여 감독자 모드에서 **SYS\$OUTPUT**을 재정의하는 경우에도 마찬가지입니다. 재지정된 파일은 명령 순서가 실행되기 전에 만들어진다는 점에 주의하십시오. 재지정된 파일이 명령 순서에서도 사용되는 경우에는 다음 보기처럼 작업이 실패할 수 있습니다.

```
$ PIPE SEARCH TRANS.LOG "alpha" > TRANS.LOG
```

```
%SEARCH-W-OPENIN, error opening TRANS.LOG;2 as input
-RMS-E-FLK, file currently locked by another user
```

이 보기에서는 **TRANS.LOG** 파일의 새 버전이 만들어지고 쓰기 액세스를 위해 열리고 **SEARCH** 명령은 예상되는 이전 버전 대신 가장 최근 **TRANS.LOG** 버전에 대해 읽기 액세스를 얻으려고 시도합니다.

SYS\$ERROR가 재지정될 때 재지정된 오류 파일은 명령 순서가 실행되는 동안 실제로 **SYS\$ERROR**에 쓰는 경우와 재지정된 오류 파일과 같은 이름의 기존 파일이 없는 경우에만 만들어집니다. 재지정된 오류 파일과 같은 이름의 파일이 이미 존재하는 경우에는 그 파일이 재지정된 오류 파일로 열립니다. 그 다음에는 이 명령 순서에 의해 생성된 오류 출력이 재지정된 오류 파일의 끝에 추가됩니다. **DEFINE** 또는 **ASSIGN** 명령을 사용하여 감독자 모드에서 **SYS\$ERROR**를 재정의하는 경우에도 마찬가지입니다.

PIPE 명령 중단

Ctrl/Y를 누르면 PIPE 명령을 중단할 수 있습니다. PIPE 명령이 파이프라인이나 서브셸 명령 순서에서 실행되는 경우에는 명령 순서와 PIPE 명령이 삭제됩니다. 이 경우 중단된 바로 뒤에 입력된 **CONTINUE** 명령은 PIPE 명령 실행을 재개하지 않을 것입니다.

PIPE 명령이 서브셸이나 파이프라인 명령 순서가 아닌 명령 순서를 실행하는 중이면 명령 순서가 PIPE 명령 동사 없이 DCL 명령으로 입력되었고 Ctrl/Y에 의해 중단된 것처럼 동작합니다. Ctrl/Y 중단에 대한 자세한 내용은 541페이지의 “Ctrl/Y 중단 처리”를 참조하십시오.

하위 프로세스 성능 향상

PIPE 명령은 실행되는 동안 몇 개의 하위 프로세스를 생성할 수 있습니다. 많은 경우 명령 순서에 의해 호출된 어플리케이션은 프로세스 논리 이름과 기호 이름에 의존하지 않습니다. 이러한 경우에는 프로세스 논리 이름과 기호가 PIPE 명령에 의해 생성된 하위 프로세스에 전달되지 않도록 하는 /NOLOGICAL_NAMES 및 /NOSYMBOLS 한정자를 사용하여 하위 프로세스 생성을 가속화할 수 있습니다.

다음은 PIPE 명령을 사용하는 보기들입니다.

- 다음 보기는 기호 정의가 포함된 여러 명령을 사용하여 명령 프로시저에서 유용한 도구를 만드는 두 가지 간단한 방법을 보여 줍니다.

```
$ CD_WORK ::= PIPE   SAVE_DIR=F$DIRECTORY() ; SET DEFAULT FOO:[WORK]
$ BACK    ::= SET DEF 'SAVE_DIR'
$
$ CD_WORK  ! Switch to working directory
$      :
$      :
$ BACK     ! Switch back to home directory
$ GET_RECORD ::= PIPE READ/END_OF_FILE=CLEANUP IN RECORD ; -
               F$EDIT(RECORD, "COMPRESS, TRIM")
$
$ OPEN IN EMPLOYEE.DAT
$ LOOP:
$ GET_RECORD
$      :
$      :
$ GOTO LOOP
$
$ CLEAN_UP:
$      :
```

- 다음 보기는 컴파일 및 링크 작업을 보여 줍니다. 컴파일에서 오류가 발생하지 않으면 객체 파일이 링크되어 실행 가능한 이미지가 생성됩니다. 프로그램 컴파일에서 오류가 발생하면 링크 단계를 건너뛵니다.

```
$ PIPE cc foo.c && link foo, sys$library:vaxcrtl.olb/lib
```

- 다음 보기는 조건 명령 실행을 사용하여 명령 프로시저에서 간단한 오류 처리 제어 흐름을 쉽게 설정하는 방법을 보여 줍니다. **COLLECT_DATA** 이미지가 실패하면 제어 가 **CLEAN_UP**으로 전달됩니다.

```
$ PIPE RUN COLLECT_DATA.EXE GOTO CLEAN_UP
$ :
$ :
$ EXIT
$
$ CLEAN_UP:
$:
$:
$ :
```

- 다음 보기에서 **PIPE** 명령은 백그라운드 프로세스를 만들어 크기가 큰 파일 복사를 처리합니다.

```
$ PIPE COPY LARGE_FILE.DAT REMOTE"user password"::[DESTINATION]*.* &
```

- 다음 보기는 하위 프로세스에서 수행될 서버셀 명령 순서를 설정하는 방법을 보여 줍니다. 결과적으로 기본 디렉토리와 같은 특정 프로세스의 특성을 변경해도 서버셀이 종료된 후 현재 프로세스는 영향을 받지 않습니다. 저장 집합은 하위 디렉토리에 복원되어 **FOO** 프로그램을 실행하는 데 필요한 데이터를 제공합니다.

```
$ PIPE (SET DEF [.DATA_DIR] ; BACKUP DATA.SAV/SAV [...]) ; RUN FOO
```

- 다음 보기에서는 파이프라인 기능을 사용하여 시스템 상의 모든 휴면 프로세스를 한 명령에서 식별합니다.

```
$ PIPE SHOW SYSTEM | SEARCH SYS$INPUT HIB
```

- 다음 보기에서는 파이프라인 기능을 사용하여 불필요한 중간 파일을 만들지 않고 한 명령에서 테스트를 실행하여 결과를 정렬하고 벤치마크 파일과 비교합니다.

```
$ PIPE RUN TEST | SORT/SPECIFICATION=TEST.SRT SYS$INPUT SYS$OUTPUT -
| DIFF SYS$INPUT TEST.BENCHMARK
```

- 다음 보기는 서버셀을 파이프라인의 파이프 세그먼트 명령으로 지정할 수 있는 한 가지 방법을 보여 줍니다.

```
$ PIPE ( SET DEF WRK$:[WORK] ; RUN REPORT ) | MAIL SYS$INPUT SMITH
```

- 다음 보기는 파이프라인 내의 /PAGE 한정자 사용을 보여 줍니다. 몇 가지 다른 DCL 명령에도 있는 /PAGE 기능은 PIPE 명령과 함께 사용하여 다른 유용한 도구를 만들 수 있습니다.

```
$ more :== TYPE/PAGE=SAVE SYSS$INPUT
$ PIPE      ANA/RMS PAGE.TXT | more
```

```
Check RMS File Integrity                26-JAN-2002 16:12:00.06   Page 1
SYSS$SYSDEVICE:[TEST]PAGE.TXT;2
```

FILE HEADER

```
File Spec: SYSS$SYSDEVICE:[TEST]PAGE.TXT;2
File ID: (4135,58220,0)
Owner UIC: [PIPE]
Protection: System: RWED, Owner: RWED, Group: RE, World:
Creation Date: 26-NOV-2002 16:08:50.05
Revision Date: 26-NOV-2002 16:09:09.06, Number: 1
Expiration Date: none specified
Backup Date: none posted
Contiguity Options: none
Performance Options: none
Reliability Options: none
Journaling Enabled: none
```

RMS FILE ATTRIBUTES

```
RETURN/SPACE=More, PREV/NEXT=Scroll, INS/REM=Pan, SELECT=80/132, Q=Quit
```


렉시칼 함수는 명령행이나 명령 프로시저로 정보를 반환합니다. 반환된 정보는 사용자 프로세스나 시스템, 파일 및 장치, 논리 이름, 문자열 또는 데이터 유형에 대한 것일 수 있습니다. 렉시칼 함수는 접두사 **F\$**에 의해 식별됩니다.

일반적으로 기호나 표현식을 사용하는 컨텍스트에서 렉시칼 함수를 사용할 수 있습니다. 명령 프로시저에서 렉시칼 함수를 사용하여 논리 이름을 변환하고 문자열 조작을 수행하며 프로시저의 현재 처리 모드를 확인할 수 있습니다. 많은 렉시칼 함수는 **DCL** 명령에서 가져올 수 있는 정보를 반환합니다.

이 장에는 다음 정보가 포함되어 있습니다.

- 렉시칼 함수 작동 방법
- 사용자 프로세스에 대한 정보 얻기
- 시스템에 대한 정보 얻기
- 파일 및 장치에 대한 정보 얻기
- 논리 이름 변환
- 문자열 조작
- 데이터 유형 조작

렉시칼 함수에 대한 자세한 내용은 온라인 도움말을 참조하십시오. 이 장에서 설명하는 명령에 대한 자세한 내용은 *OpenVMS DCL Dictionary*를 참조하십시오.

렉시칼 함수를 사용하는 이유

명령이 아닌 렉시칼 함수로부터 정보를 얻으면 명령 프로시저에서 이 정보를 좀 더 쉽게 조작할 수 있습니다. 예를 들어, `F$ENVIRONMENT` 함수나 `SHOW DEFAULT` 명령 중 하나를 사용하여 현재 기본 디렉토리의 이름을 얻을 수 있습니다. 그 차이는 다음과 같습니다.

- `F$ENVIRONMENT` 함수를 사용하면 결과를 기호에 할당한 다음 나중에 프로시저에서 이 기호를 사용할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ DIR_NAME = F$ENVIRONMENT("DEFAULT")
$ SET DEFAULT DISK4:[TEST]
.
.
.
$ SET DEFAULT 'DIR_NAME'
```

`F$ENVIRONMENT` 함수는 현재 기본 디스크와 디렉토리를 반환하며 이 값을 기호 `DIR_NAME`에 저장합니다. 프로시저 끝에서 `DIR_NAME` 기호를 사용하여 `SET DEFAULT` 명령으로 기본값을 복원할 수 있습니다.

- `F$ENVIRONMENT` 함수 대신 `SHOW DEFAULT` 명령을 사용하여 현재 기본 디렉토리의 값을 얻는 경우 이 출력을 기호에 직접 할당할 수 없습니다. 따라서 프로시저는 다음과 같이 집니다.

```
$! Redirect the output of the SHOW DEFAULT command to a file.
$ DEFINE/SUPERVISOR_MODE SYS$OUTPUT DISK4:[TEST]TEMPFILE.DAT
$ SHOW DEFAULT
$ DEASSIGN SYS$OUTPUT
$!
$ OPEN/READ DIR_FILE DISK4:[TEST]TEMPFILE.DAT ! Open the file.
$ READ DIR_FILE DIR_NAME,                      ! Read the file.
$ SET DEFAULT 'DIR_NAME'                        ! Reset the directory.
$ CLOSE DIR_FILE                                ! Close the file.
$ DELETE DISK4:[TEST]TEMPFILE.DAT;*             ! Delete the file.
```

사용자 프로세스에 대한 정보 얻기

명령 프로시저가 지속되는 동안 프로세스 특성을 변경한 다음 복원하는 경우가 종종 있습니다. 이런 경우 다음 렉시칼 함수를 사용하여 사용자 프로세스에 대한 정보를 얻을 수 있습니다.

표 15-1

F\$DIRECTORY	현재 기본 디렉토리 문자열을 반환합니다.
F\$ENVIRONMENT	사용자 프로세스의 명령 환경에 대한 정보를 반환합니다.
F\$GETJPI	시스템의 사용자 프로세스나 다른 프로세스에 대한 계정, 상태 및 식별 정보를 반환합니다.
F\$MODE	프로세스가 실행되고 있는 모드를 보여 줍니다.
F\$PRIVILEGE	사용자 프로세스가 지정된 권한을 가지고 있는지 표시합니다.
F\$PROCESS	사용자 프로세스 이름을 반환합니다.
F\$SETPRV	지정된 권한을 설정하고 F\$SETPRV 함수를 사용하기 전에 지정된 권한이 이미 활성화되어 있는지 표시합니다.
F\$USER	사용자 식별 코드(UIC)를 반환합니다.
F\$VERIFY	확인 기능이 켜져 있는지 꺼져 있는지 표시합니다.

다음 표는 명령 프로시저에서 일반적으로 변경되는 프로세스 특성을 보여 줍니다. 그리고 이러한 특성을 저장하는 렉시칼 함수와 원래 설정을 복원하는 DCL 명령을 제시합니다.

표 15-2

특성	작업	명령 또는 렉시칼 함수
제어 문자	저장	F\$ENVIRONMENT("CONTROL")
	복원	SET CONTROL
DCL 프롬프트	저장	F\$ENVIRONMENT("PROMPT")
	복원	SET PROMPT
기본 보호	저장	F\$ENVIRONMENT("PROTECTION")
	복원	SET PROTECTION/DEFAULT
키 상태	저장	F\$ENVIRONMENT("KEY_STATE")
	복원	SET KEY
메시지 형식	저장	F\$ENVIRONMENT("MESSAGE")
	복원	SET MESSAGE
권한	저장	F\$PRIVILEGE 또는 F\$SETPRV
	복원	F\$SETPRV 또는 SET PROCESS/PRIVILEGES
확인	저장	F\$VERIFY 또는 F\$ENVIRONMENT
	복원	F\$VERIFY 또는 SET VERIFY

프로세스 특성을 저장하는 경우 원래 특성을 복원하기 전에 오류나 **Ctrl/Y** 인터럽트로 인해 프로시저가 종료하지 않도록 해야 합니다. 오류 및 **Ctrl/Y** 인터럽트 처리에 대한 자세한 내용은 13장을 참조하십시오.

확인 설정 변경

F\$VERIFY 렉시칼 함수를 사용하여 명령 프로시저가 지속되는 동안 확인을 비활성화할 수 있습니다. 이렇게 하면 프로시저를 실행하는 동안 사용자가 프로시저 내용을 표시하지 못하게 할 수 있습니다.

두 가지 유형의 확인이 있습니다.

- 프로시저 확인
실행 중인 각 명령행을 표시합니다.
- 이미지 확인
처리 중인 각 데이터 행을 표시합니다.

기본적으로 **SET [NO]VERIFY** 명령과 **F\$VERIFY** 함수는 두 가지 확인 유형을 모두 활성화하거나 비활성화합니다. 일반적으로 한 프로시저에서 프로시저 및 이미지 확인 설정은 같습니다. 즉, 양쪽 모두 활성화되어 있거나 양쪽 모두 비활성화되어 있습니다. 하지만 설정을 변경하기로 결정하면 각 확인 설정을 따로 저장하십시오.

다음 보기에서 기호 **TEMP**는 확인을 활성화 및 비활성화하는 데 사용합니다.

```
$ ! Enable verification
$ !
$ TEMP = F$VERIFY(1)
$ LOOP:
$     INQUIRE FILE "File name"$      IF FILE .EQS."" THEN EXIT
$     PRINT 'FILE'
$     GOTO LOOP
$ ! Disable verification
$ !
$ TEMP = F$VERIFY(0)
$ EXIT
```

다음 보기에서는 확인 설정이 저장됩니다.

```
$ ! Save each verification state.
$ ! Turn both states off.
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
.
.
.
$ ! Restore original verification states.
$ SAVE_VERIFY_IMAGE = F$VERIFY(SAVE_VERIFY_PROCEDURE, -
    SAVE_VERIFY_IMAGE)
```

F\$ENVIRONMENT 함수가 현재 이미지 확인 설정을 반환하고 이 값을 기호 **SAVE_VERIFY_IMAGE**에 할당합니다. 그런 다음 **F\$VERIFY** 함수가 현재 프로시저 확인 설정을 반환하고 이 값을 기호 **SAVE_VERIFY_PROCEDURE**에 할당합니다. **F\$VERIFY** 함수는 이미지 및 프로시저 확인을 모두 비활성화합니다. **F\$VERIFY**를 사용하여 확인을 비활성화하기 전에 **F\$ENVIRONMENT** 함수를 사용하여 프로시저 확인 설정을 얻을 수 있습니다. 하지만 앞의 보기에서처럼 두 작업을 한 명령행에서 수행하려면 **F\$VERIFY**를 사용하는 것이 더 간단합니다.

이 프로시저의 끝에서 **F\$VERIFY** 함수는 기호 **SAVE_VERIFY_PROCEDURE**와 **SAVE_VERIFY_IMAGE**로 지정했던 원래 설정을 복원합니다.

주

time-stamp를 사용하고 있는 경우에는 확인이 활성화되어 있을 때만 적용된다는 사실을 기억해야 합니다. **time-stamp** 및 **SET PREFIX** 명령에 대한 자세한 내용은 *OpenVMS DCL Dictionary* 또는 DCL 도움말을 참조하십시오.

기본 파일 보호 변경

명령 프로시저 내에서 기본 파일 보호를 변경해야 하는 경우가 있습니다. 다음 명령 프로시저는 프로시저가 실행되는 동안 만든 파일과 연관된 기본 보호를 변경합니다. 프로시저에서는 종료하기 전에 원래의 기본 파일 보호를 복원합니다.

```
$ SAVE_PROT = F$ENVIRONMENT("PROTECTION")
$ SET PROTECTION = (SYSTEM:RWED, OWNER:RWED, GROUP, WORLD)/DEFAULT
.
.
.
$ SET PROTECTION=('SAVE_PROT')/DEFAULT
$ EXIT
```

F\$ENVIRONMENT 함수는 **SET PROTECTION** 명령에 필요한 구문을 사용하여 기본 보호 코드를 반환한다는 사실을 유의하십시오. 따라서 **SET PROTECTION** 명령과 함께 기호 **SAVE_PROT**을 사용하여 원래의 기본 파일 보호를 복원할 수 있습니다.

시스템에 대한 정보 얻기

다음 렉시칼 함수를 사용하여 시스템에 대한 정보를 얻을 수 있습니다.

표 15-3

F\$CONTEXT	F\$PID 함수에 사용할 선택 조건을 지정합니다. F\$CONTEXT 함수를 사용하면 F\$PID 함수가 OpenVMS Cluster 시스템의 모든 노드에서 프로세스에 대한 정보를 얻을 수 있습니다.
F\$CSID	OpenVMS Cluster 식별 번호를 반환하고 컨텍스트 기호를 업데이트하여 시스템의 OpenVMS Cluster 노드 목록에서 현재 위치를 가리키게 합니다.
F\$GETQUI	대기열, 현재 해당 대기열에 있는 일괄 처리 작업 및 인쇄 작업, 양식 정의 그리고 시스템 작업 대기열 파일에 보관된 특성 정의에 대한 정보를 반환합니다.
F\$GETSYI	로컬 시스템이나 OpenVMS Cluster 시스템의 노드(시스템이 OpenVMS Cluster의 일부인 경우)에 대한 정보를 반환합니다.
F\$IDENTIFIER	증명서를 이름이 지정된 형식에서 숫자 형식으로 또는 숫자 형식에서 이름이 지정된 형식으로 변환합니다.
F\$MESSAGE	상태 코드와 연결된 메시지 텍스트를 반환합니다.
F\$PID	사용자가 검사할 수 있는 프로세스의 프로세스 증명서(PID)를 반환합니다.
F\$TIME	현재 날짜와 시간을 반환합니다.

OpenVMS Cluster 노드 이름 확인

사용자의 시스템이 여러 개의 서로 다른 노드에 로그인할 수 있는 **OpenVMS Cluster** 환경의 일부인 경우에는 현재 사용하고 있는 노드를 나타내도록 **DCL** 프롬프트를 설정할 수 있습니다. 이 작업을 하려면 노드 이름을 확인할 수 있도록 로그인 명령 프로시저에 **F\$GETSYI** 함수를 포함해야 합니다. 그런 다음 **SET PROMPT** 명령을 사용하여 해당 노드에 대해 고유한 프롬프트를 설정해야 합니다.

노드 이름의 일부만 프롬프트 문자열에 사용하려면 **F\$EXTRACT** 함수를 사용하여 적절한 문자를 추출하십시오. 문자 추출에 대한 자세한 내용은 651페이지의 “문자열의 일부 추출”을 참조하십시오.

다음 보기에서 기호 **NODE**는 **FF\$GETSYI("NODENAME")**로 정의되어 있으며 노드 이름은 프롬프트로 사용됩니다.

```
$ NODE = F$GETSYI("NODENAME")
$ SET PROMPT = "'NODE'$ "
.
.
.
```

대기열 정보 얻기

F\$GETQUI 함수를 사용하여 일괄 처리 대기열 및 인쇄 대기열에 대한 여러 가지 유형의 정보를 얻을 수 있습니다. 대기열의 작업 및 파일에 대한 정보를 얻으려면 해당 작업에 대한 읽기 액세스 권한이나 **SYSPRV** 권한 또는 **OPER** 권한이 있어야 합니다.

다음 보기는 일괄 처리 대기열 **VAX1_BATCH**가 중단 상태에 있는지 확인하는 방법을 보여줍니다. 반환된 값은 **True**이거나 **False**입니다. 대기열이 중단되지 않았으면 명령 프로시저가 작업을 대기열로 전송합니다.

```
$ QSTOPPED = F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPED", "VAX1_BATCH")
$ IF QSTOPPED THEN GOTO NOBATCH
$ SUBMIT/QUEUE=VAX1_BATCH TEST.COM
$ NOBATCH:
.
.
.
```

프로세스 정보 얻기

F\$PID 함수를 사용하면 검토할 수 있는 모든 프로세스의 프로세스 증명서(PID)를 얻을 수 있습니다. 다음 프로세스의 PID 번호를 얻을 수 있습니다.

- **WORLD** 권한이 있는 경우 시스템의 모든 프로세스
- **GROUP** 권한이 있는 경우 그룹의 모든 프로세스
- **GROUP** 또는 **WORLD** 권한이 모두 없는 경우 사용자 프로세스

PID 번호를 얻었으면 **F\$GETJPI** 함수를 사용하여 프로세스에 대한 특정 정보를 얻을 수 있습니다.

다음 보기는 검토할 수 있는 프로세스의 PID 번호를 얻고 표시하는 방법을 보여 줍니다.

```
$ ! Display the time when this procedure
$ ! begins executing.
$ WRITE SYS$OUTPUT F$TIME()
$ !
$ CONTEXT = ""$ START:
$ ! Obtain and display PID numbers until
$ ! F$PID returns a null string.
$ !
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ WRITE SYS$OUTPUT "Pid --- '"PID'"$ GOTO START
```

시스템에서는 기호 **CONTEXT**를 사용하여 PID 번호의 시스템 목록 안에 포인터를 유지합니다. 루프를 통과할 때마다 시스템은 포인터를 변경하여 목록에서 다음 PID 번호를 찾습니다. PID 번호가 모두 표시되면 프로시저가 종료됩니다.

다음 보기에서 프로시저는 각 프로세스의 PID 번호와 UIC를 표시합니다.

```
$ CONTEXT = ""$ START:
$ ! Obtain and display PID numbers and UICs.
$ !
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ UIC = F$GETJPI(PID,"UIC")
$ WRITE SYS$OUTPUT "Pid --- '"PID'    Uic--- '"UIC' "$ GOTO START
```

다음과 같이 **WRITE** 명령 내에 **F\$GETJPI** 함수를 포함하여 이 명령 프로시저를 짧게 만들 수 있습니다.

```
$ CONTEXT = "$ START:
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ WRITE SYS$OUTPUT "Pid --- 'PID'   Uic --- 'F$GETJPI(PID,"UIC")'$ GOTO
START
```

F\$CONTEXT 렉시칼 함수

OpenVMS Cluster 시스템의 노드로부터 프로세스에 대한 정보를 얻으려면 **F\$CONTEXT** 함수를 사용하십시오.

다음 보기에서는 선택 조건을 설정하기 위해 **F\$CONTEXT**가 세 번 호출됩니다.

```
$! Establish an error and Ctrl/Y handler.
$!
$ ON ERROR THEN GOTO error
$ ON CONTROL_Y THEN GOTO error
$!
$ ctx = ""
$ temp = F$CONTEXT ("PROCESS", ctx, "NODENAME", "*", "EQL") [1]
$ temp = F$CONTEXT ("PROCESS", ctx, "USERNAME", "M*", "SYSTEM", "EQL") [2]
$ temp = F$CONTEXT ("PROCESS", ctx, "CURPRIV", "SYSPRV, OPER", "ALL") [3]
$!
$!Loop over all processes that meet the selection criteria.
$!Print the PID number and the name of the image for each process.
$!
$loop: [4]
$ pid = F$PID(ctx)
$ IF pid .EQS. "" THEN
$     GOTO endloop
$ ELSE
$     image = F$GETJPI(pid, "IMAGENAME") [5]
$     SHOW SYMBOL pid
$     WRITE SYS$OUTPUT image [6]
$     GOTO loop
$ ENDIF
$!The loop over the processes has ended.
$!
$endloop:
$!
```



```
$ EXIT
$!
$!Error handler. Clean up the context's memory with
$!the CANCEL selection item keyword.
$!
$error:
$ IF F$TYPE(ctx) .eqs. "PROCESS_CONTEXT" THEN - [7]
-$ temp = F$CONTEXT ("PROCESS", ctx, "CANCEL") [8]
$!
$ EXIT
```

보기를 검토할 때 다음 내용에 유의하십시오.

- [1] 첫 번째 호출에서는 **OpenVMS Cluster** 시스템의 모든 노드를 검색하도록 요청합니다.
- [2] 두 번째 호출에서는 사용자 이름이 **M**으로 시작하거나 **SYSTEM**인 프로세스만 처리할 것을 요청합니다.
- [3] 세 번째 호출은 현재 권한에 **SYSPRV**(시스템 권한)와 **OPER**(운영자)를 모두 포함하고 다른 권한도 가질 수 있는 프로세스만 선택하도록 제한합니다.
- [4] 레이블 “loop”과 “endloop” 사이의 명령행은 계속 **F\$PID**를 호출하여 **F\$CONTEXT** 호출에 설정된 조건에 맞는 프로세스를 얻습니다.
- [5] 각 **PID** 번호를 검색한 후 프로세스에서 실행 중인 이미지 이름을 반환하기 위해 **F\$GETJPI**가 호출됩니다.
- [6] 마지막으로 프로시저에서 이미지 이름을 표시합니다.
- [7] 오류나 **Ctrl/Y** 작업이 발생하는 경우 컨트롤은 *error*로 전달되고 필요한 경우 컨텍스트가 닫힙니다.
- [8] **PROCESS_CONTEXT** 기호 유형에 대한 검사에 유의하십시오. 기호가 이 유형이면 선택 조건은 **F\$CONTEXT**에 대한 호출에 의해 취소되어야 합니다. 기호가 **PROCESS_CONTEXT** 유형이 아닌 경우에는 선택 조건이 **F\$CONTEXT**에 아직 설정되지 않았거나, 또는 오류가 발생하거나 프로세스 목록 끝에 도달할 때까지 기호가 사용된 것입니다.

파일 및 장치에 대한 정보 얻기

다음 렉시칼 함수를 사용하여 파일 및 장치에 대한 정보를 얻을 수 있습니다.

표 15-4

F\$DEVICE	지정한 선택 조건을 만족하는 시스템의 모든 장치 이름을 반환합니다.
F\$FILE_ATTRIBUTES	파일 속성에 대한 정보를 반환합니다.
F\$GETDVI	지정된 장치에 대한 정보를 반환합니다.
F\$PARSE	파일 지정 방법을 구문 분석하고 요청한 필드를 반환합니다.
F\$SEARCH	디렉토리에서 파일을 검색합니다.

장치 검색

시스템 서비스 **\$GETDVI**를 사용하여 특정 장치에 대한 정보를 얻으려면 장치 이름을 서비스에 제공해야 합니다. 장치 이름을 모르는 경우 렉시칼 함수 **F\$DEVICE**를 사용하여 장치 이름을 찾을 수 있습니다.

F\$DEVICE 함수를 사용하면 장치 이름, 장치 클래스 또는 장치 유형으로 와일드카드 검색을 할 수 있습니다. 장치 유형으로 검색하는 경우 장치 클래스도 지정해야 합니다.

명령 프로시저의 루프에 **F\$DEVICE** 함수를 사용하여 지정한 선택 조건과 일치하는 장치 이름을 반환할 수 있습니다. **F\$DEVICE** 함수가 실행될 때마다 선택 조건에 일치하는 시스템에서 그 다음 장치를 반환합니다. 특정 순서로 장치가 반환되는 것은 아닙니다. 마지막 장치 이름이 반환된 **F\$DEVICE** 함수를 다시 호출하면 **null** 문자열이 반환됩니다.

다음 명령 프로시저는 번호가 0인 장치에서 모든 **RA60**의 장치 이름을 표시합니다.

```
$ START:
$   DEVICE_NAME = F$DEVICE("*0:", "DISK", "RA60")
$   IF DEVICE_NAME .EQS. "" THEN EXIT
$   SHOW SYMBOL DEVICE_NAME
$   GOTO START
```

디렉토리에서 파일 검색

파일을 처리하기 전에 명령 프로시저는 파일이 존재하는지 테스트하기 위해 **F\$SEARCH** 함수를 사용해야 합니다. 예를 들어, 다음 명령 프로시저는 **F\$PARSE**를 사용하여 장치 및 디렉토리 문자열을 **STATS.DAT** 파일에 적용합니다. 그런 다음 프로시저에서 **F\$SEARCH** 함수를 사용하여 **STATS.DAT**이 **DISK3:[JONES.WORK]**에 있는지 확인합니다. 파일이 있으면 명령 프로시저에서 파일을 처리합니다. 파일이 없으면 명령 프로시저는 다른 입력 파일을 입력하라는 메시지를 표시합니다.

```
$ FILE = F$PARSE("STATS.DAT","DISK3:[JONES.WORK]",,,"SYNTAX_ONLY")
$ IF F$SEARCH(FILE) .EQS. "" THEN GOTO GET_FILE
$ PROCESS_FILE:
.
.
.
$ GET_FILE:
$   INQUIRE FILE "File name"$   GOTO PROCESS_FILE
```

파일이 있는지 확인한 다음 프로시저는 **F\$PARSE** 또는 **F\$FILE_ATTRIBUTES** 함수를 사용하여 파일에 대한 추가 정보를 얻을 수 있습니다. 예를 들면 다음과 같습니다.

```
$ IF F$SEARCH("STATS.DAT") .EQS. "" THEN GOTO GET_FILE
$ PROCESS_FILE:
$   NAME = F$PARSE("STATS.DAT",,,"NAME")
.
.
.
$ GET_FILE:
$   INQUIRE FILE "File name"$   GOTO PROCESS_FILE
```

파일의 이전 버전 삭제

명령 프로시저가 종료된 후에 프로시저에 필요 없는 파일을 만드는 경우에는 해당 프로시저를 끝내기 전에 이 파일을 삭제하거나 지워야 합니다. 가장 최신 버전을 제외한 모든 버전을 삭제하려면 **PURGE** 명령을 사용하십시오. 파일의 특정 버전을 삭제하려면 **DELETE** 명령에 버전 번호를 사용하고, 파일의 모든 버전을 삭제하려면 **DELETE** 명령과 함께 버전 필드에 와일드카드 문자를 사용하십시오.

명령 프로시저 내에서 **DELETE** 명령을 사용할 때 오류 메시지를 피하려면 파일을 삭제하기 전에 **F\$SEARCH** 함수를 사용하여 파일이 있는지 확인하십시오. 예를 들어, 특정 모듈이 실행되는 경우에만 **TEMP.DAT**라는 파일을 만드는 명령 프로시저를 작성할 수 있습니다. 다음 명령행은 **TEMP.DAT**가 존재하는 경우에만 **DELETE** 명령을 실행합니다.

```
$ IF F$SEARCH("TEMP.DAT") .NES. "" THEN DELETE TEMP.DAT;*
```

논리 이름 변환

다음 렉시칼 함수를 사용하여 논리 이름을 변환할 수 있습니다.

표 15-5

F\$LOGICAL	논리 이름의 등가 문자열을 반환합니다.
F\$TRNLNM	논리 이름의 등가 문자열이나 요청된 속성 중 하나를 반환합니다.

주

F\$TRNLNM 함수는 **OpenVMS** 운영 체제의 이전 버전에서 사용되었던 **F\$LOGICAL** 함수보다 우선합니다. 명령 프로시저에서 현재 시스템 기술을 사용하여 논리 이름을 처리하게 하려면 **F\$LOGICAL** 대신 **F\$TRNLNM**을 사용해야 합니다.

일부 상황에서는 명령 프로시저에서 기호 대신 논리 이름을 변수로 사용해야 하는 경우가 있습니다. 보통 프로그램들은 **DCL** 기호에 액세스하는 것보다 더 쉽게 논리 이름에 액세스할 수 있습니다. 그러므로 명령 프로시저에서 실행하는 프로그램에 정보를 전달하려면 기호를 사용하여 정보를 얻은 다음 **DEFINE**이나 **ASSIGN** 명령을 사용하여 기호 값을 논리 이름과 일치시킬 수 있습니다.

F\$TRNLNM 함수를 사용하여 논리 이름의 값을 확인한 다음 이 값을 기호에 할당할 수도 있습니다.

다음 보기는 **NAMES**라는 논리 이름이 정의되어 있는지 테스트합니다. 논리 이름이 정의되어 있으면 프로시저에서 **PAYROLL.EXE**를 실행합니다. 논리 이름이 정의되어 있지 않으면 기호 **FILE**의 값을 얻은 다음 이 값을 사용하여 논리 이름 **NAMES**를 만듭니다. **PAYROLL.EXE**는 논리 이름 **NAMES**를 사용하여 직원 이름 파일을 참조합니다.

```
$ ! Make sure that NAMES is defined.
$ IF F$TRNLNM("NAMES") .NES. "" THEN GOTO ALL_SET
$ INQUIRE FILE "File with employee names"$ DEFINE NAMES 'FILE'
$ !
$ ! Run PAYROLL, using the file indicated by NAMES.
$ ALL_SET:
$ RUN PAYROLL
.
.
.
```

이 명령 프로시저는 프로그램 **PAYROLL**에 사용되는 논리 이름을 정의합니다.

```
$ DEFINE NAMES DISK4:[JONES]EMPLOYEE_NAMES.DAT
$ RUN PAYROLL
.
.
.
$ WRITE SYS$OUTPUT "Finished processing ",F$TRNLNM("NAMES")
```

프로시저 끝에서 **WRITE** 명령은 파일이 처리되었는지 나타내는 메시지를 표시합니다.

문자열 조작

다음 렉시칼 함수를 사용하여 문자열을 조작할 수 있습니다.

표 15-6

F\$CVTIME	시간 문자열에 대한 정보를 반환합니다.
F\$EDIT	문자열을 편집합니다.
F\$ELEMENT	요소가 구분 기호로 구분되어 있는 문자열에서 요소를 추출합니다.
F\$EXTRACT	문자열의 섹션을 추출합니다.
F\$FAO	출력 문자열 형식을 지정합니다.
F\$LENGTH	문자열의 길이를 결정합니다.
F\$LOCATE	문자열 내에서 문자나 부분 문자열을 찾아 오프셋을 반환합니다.

문자열이나 문자가 있는지 확인

문자열을 검토하는 일반적인 이유는 문자열 내에 문자 또는 부분 문자열이 있는지 확인하기 위한 것입니다. 확인하려면 **F\$LENGTH** 및 **F\$LOCATE** 함수를 사용하십시오.

F\$LOCATE 함수에서 반환하는 값이 **F\$LENGTH** 함수에서 반환하는 값과 같으면 찾는 문자가 없는 것입니다.

다음 프로시저에는 버전 번호가 포함된 파일 이름이 필요합니다. 버전 번호가 있는지 확인하기 위해 프로시저에서는 파일 이름의 버전 번호 앞에 있는 세미콜론(;)이 사용자가 입력한 파일 이름에 포함되어 있는지 테스트합니다.

```
$ INQUIRE FILE "Enter file (include version number)"
$ IF F$LOCATE(";", FILE) .EQ. F$LENGTH(FILE) THEN -
    GOTO NO_VERSION
.
.
.
```

F\$LOCATE 함수는 세미콜론의 오프셋을 반환합니다. 오프셋은 0에서 시작합니다. 따라서 세미콜론이 문자열의 첫 번째 문자이면 **F\$LOCATE** 함수는 정수 0을 반환합니다. 문자열 내에 세미콜론이 없으면 **F\$LOCATE** 함수는 문자열에서 마지막 문자의 오프셋보다 하나 큰 오프셋을 반환합니다. 이 값은 숫자 1로 시작하는 문자열 길이를 측정하는 **F\$LENGTH**에서 반환하는 길이와 같습니다.

문자열의 일부 추출

문자열의 일부를 추출하려면 **F\$EXTRACT** 함수나 **F\$ELEMENT** 함수 중 하나를 사용하십시오. 정의된 오프셋에서 시작하는 부분 문자열을 추출하려면 **F\$EXTRACT** 함수를 사용하십시오. 두 개의 구분 기호 사이에 있는 문자열의 일부를 추출하려면 **F\$ELEMENT** 함수를 사용하십시오. 이러한 함수 중 하나를 사용하려면 구문 분석할 문자열의 일반 형식을 알아야 합니다. 파일 지정 방법이나 시간 문자열을 구문 분석하는 데는 **F\$EXTRACT**나 **F\$ELEMENT**를 사용할 필요가 없습니다. 대신 **F\$PARSE**나 **F\$CVTIME**을 사용하여 파일 지정 방법이나 시간 문자열에서 필요한 부분을 추출하십시오.

그리고 그룹 이름을 추출할 때 그룹 이름의 길이도 같이 확인할 수 있습니다.

문자열의 다른 부분을 구분하는 구분 기호가 문자열에 있으면 **F\$ELEMENT** 함수를 사용하여 원하는 부분을 추출하십시오. **F\$ELEMENT**를 사용하면 쉼표 사이의 문자열 부분을 추출하므로 다양한 액세스 유형을 얻을 수 있습니다. 시스템 액세스를 확인하려면 첫 번째 요소를 가져오고, 소유자 액세스를 확인하려면 두 번째 요소를 가져오는 식으로 처리됩니다. **F\$ELEMENT** 함수를 사용할 때 요소 숫자는 0에서 시작합니다. 즉, 네 번째 요소를 지정하려면 정수 3을 사용해야 합니다.

다음 명령 프로시저는 **F\$EXTRACT** 함수를 사용하여 **UIC**의 그룹 부분을 추출합니다. 이렇게 하면 프로시저에서 사용자의 **UIC** 그룹에 따라 다른 명령 집합을 실행할 수 있습니다.

```
$ UIC = F$USER()
$ GROUP_LEN = F$LOCATE(", ", UIC) - 1
$ GROUP = F$EXTRACT(1, GROUP_LEN, UIC)
$ GOTO 'GROUP'_SECTION
.
.
.
$ WRITERS_SECTION:
.
.
.
$ MANAGERS_SECTION:
.
.
.
```

먼저 프로시저에서 **F\$USER** 함수를 사용하여 **UIC**를 확인합니다. 그런 다음 프로시저는 **F\$LOCATE** 함수로 쉼표의 오프셋을 찾아 그룹 이름 길이를 확인합니다. 쉼표는 **UIC**의 사용자 부분에서 그룹을 분리합니다. 왼쪽 대괄호와 쉼표 사이의 모든 내용이 그룹 이름 부분입니다. 예를 들어, **UIC [WRITERS,SMITH]**에서 그룹 이름은 **WRITERS**입니다.

길이를 확인한 다음 프로시저에서는 **F\$EXTRACT** 함수를 사용하여 그룹 이름을 추출합니다. 오프셋 **1**로 시작하는 이름은 쉼표 앞의 문자에서 끝납니다. 마지막으로 실행을 적절한 레이블로 지시합니다.

다음 보기는 그룹 이름을 추출하면서 동시에 그룹 이름 길이를 확인하는 방법을 보여 줍니다.

```
$ UIC = F$USER()
$ GROUP = F$EXTRACT(1, F$LOCATE(", ", UIC) - 1, UIC)
$ GOTO 'GROUP'_SECTION
```

다음 보기는 보호 코드의 각 액세스 유형을 쉼표로 구분하는 방법을 보여 줍니다.

```
$ PROT = F$ENVIRONMENT("PROTECTION")
$ SHOW SYMBOL PROT
PROT = "SYSTEM=RWED, OWNER=RWED, GROUP=RE, WORLD"
```

이 보기의 명령은 기본 보호 코드에서 **world** 액세스 부분(네 번째 요소)을 추출합니다.

```
$ PROT = F$ENVIRONMENT("PROTECTION")
$ WORLD_PROT = F$ELEMENT(3, ", ", PROT)
.
.
.
```

F\$ELEMENT 함수는 세 번째 쉼표와 문자열 끝 사이에 있는 모든 내용을 반환합니다. 따라서 기본 보호에서 **world** 사용자에게 읽기 액세스를 허용한 경우에는 문자열 **"WORLD=R"**이 반환됩니다.

world 액세스 문자열을 얻은 후에 문자열 내용을 검토해야 할 수 있습니다. 예를 들면 다음과 같습니다.

```
$ PROT = F$ENVIRONMENT("PROTECTION")
$ WORLD_PROT = F$ELEMENT(3, ", ", PROT)
$ IF F$LOCATE("=", WORLD_PROT) .EQ. F$LENGTH(WORLD_PROT) -
  THEN GOTO NO_WORLD_ACCESS
.
.
.
```


출력 문자열 서식 지정

WRITE 명령을 사용하여 문자열을 레코드에 기록할 수 있습니다. 레코드 열을 줄 맞추하려면 **F\$FAO** 함수를 사용하여 레코드 필드를 정의하고 이 필드에 프로세스 이름과 사용자 이름을 둘 수 있습니다. **F\$FAO** 함수를 사용할 때 제어 문자열을 사용하여 레코드에 필드를 정의한 다음 이 필드에 값이 놓이도록 지정하십시오.

레코드에서 필드의 서식을 지정하는 다른 방법은 문자열 오버레이를 사용하는 것입니다. 하지만 **F\$FAO** 함수의 기능이 문자열 오버레이보다 더 강력합니다. **F\$FAO** 함수를 사용하면 좀 더 광범위한 출력 작업을 수행할 수 있습니다.

다음 보기에 표시된 명령 프로시저는 **WRITE** 명령을 사용하여 시스템의 프로세스에 대한 프로세스 이름과 **PID** 번호를 표시합니다.

```
$ ! Initialize context symbol to get PID numbers
$ CONTEXT = "$ ! Write headings
$ WRITE SYS$OUTPUT "Process Name      PID"$ !
$ GET_PID:
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ WRITE SYS$OUTPUT F$GETJPI(PID,"PRCNAM"), "      ", F$GETJPI(PID,"PID")
$ GOTO GET_PID
```

WRITE 명령의 출력은 프로세스 이름과 사용자 이름 사이에 5개의 공백이 삽입되지만 열의 줄을 맞추지는 않습니다.

```
Process Name      PID
MARCHESAND        2CA0049C
TRACTMEN          2CA0043A
FALLON            2CA0043C
ODONNELL          2CA00453
PERRIN            2CA004DE
CHAMPIONS         2CA004E3
```

이 보기의 명령 프로시저는 **F\$FAO** 함수를 사용하여 16자 필드와 12자 필드를 정의합니다. **F\$FAO** 함수는 첫 번째 필드에 프로세스 이름을 두고 공백을 하나 둔 다음 두 번째 필드에 **PID** 번호를 둡니다.

```
$ ! Initialize context symbol to get PID numbers
$ CONTEXT = "$ ! Write headings
$ WRITE SYS$OUTPUT "Process Name      PID"$ !
$ GET_PID:
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
```

```
$ LINE = F$FAO("!16AS !12AS", F$GETJPI(PID,"PRCNAM"), F$GETJPI(PID,"PID"))
$ WRITE SYS$OUTPUT LINE
$ GOTO GET_PID
```

이제 프로시저를 실행하면 열이 정렬됩니다.

Process Name	PID
MARCHESAND	2CA0049C
TRACTMEN	2CA0043A
FALLON	2CA0043C
ODONNELL	2CA00453
PERRIN	2CA004DE
CHAMPIONS	2CA004E3

다음 보기는 오버레이를 사용하여 기호 **RECORD**의 첫 번째 **16**자(오프셋 **0**에서 시작)에 프로세스 이름을 둡니다. 그 다음 **12**자(오프셋 **17**에서 시작)에 **PID** 번호를 둡니다.

```
$ ! Initialize context symbol to get PID numbers
$ CONTEXT = "$ ! Write headings
$ WRITE SYS$OUTPUT "Process Name      PID"$ !
$ GET_PID:
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ RECORD[0,16]:= 'F$GETJPI(PID,"PRCNAM")'
$ RECORD[17,12]:= 'F$GETJPI(PID,"PID")'
$ WRITE SYS$OUTPUT RECORD
$ GOTO GET_PID
```

다음 프로시저는 **F\$FAO** 함수를 사용하여 만든 열에 같은 유형의 서식을 지정합니다.

Process Name	PID
MARCHESAND	2CA0049C
TRACTMEN	2CA0043A
FALLON	2CA0043C
ODONNELL	2CA00453
PERRIN	2CA004DE
CHAMPIONS	2CA004E3

데이터 유형 조작

다음 렉시칼 함수를 사용하여 데이터를 문자열에서 정수로 또는 정수에서 문자열로 변환할 수 있습니다.

표 15-7

F\$CVSI	문자열에서 비트 필드를 추출하여 결과를 정수(부호 있는 값)로 변환합니다.
F\$CVUI	문자열에서 비트 필드를 추출하여 결과를 정수(부호 없는 값)로 변환합니다.
F\$INTEGER	문자열 표현식을 정수로 변환합니다.
F\$STRING	정수 표현식을 문자열로 변환합니다.
F\$TYPE	기호의 데이터 유형을 결정합니다.

데이터 유형 변환

정수와 문자열 간을 변환하려면 **F\$INTEGER** 및 **F\$STRING** 함수를 사용하십시오. 예를 들어, 다음 명령 프로시저는 데이터 유형을 변환합니다. 문자열을 입력하면 명령 프로시저는 등가 정수를 보여 줍니다. 정수를 입력하면 명령 프로시저는 등가 문자열을 보여 줍니다. **GOTO** 문에서 레이블 이름의 서식을 지정하기 위해 **F\$TYPE** 함수를 사용하는 방법에 유의하십시오. **F\$TYPE**은 기호의 데이터 유형에 따라 **"STRING"**이나 **"INTEGER"**를 반환합니다.

```
$ IF P1 .EQS. "" THEN INQUIRE P1 "Value to be converted"$ GOTO  
CONVERT_'F$TYPE(P1)'  
$  
$ CONVERT_STRING:  
$ WRITE SYS$OUTPUT "The string 'P1' is converted to 'F$INTEGER(P1)'"$ EXIT  
$  
$ CONVERT_INTEGER:  
$ WRITE SYS$OUTPUT "The integer 'P1' is converted to 'F$STRING(P1)'"$ EXIT
```

표현식 평가

INQUIRE 및 READ 같은 일부 명령은 문자열 데이터만 받아들입니다. 정수 표현식으로 평가할 데이터를 얻는 데 이러한 명령을 사용하는 경우에는 F\$INTEGER 함수를 사용하여 데이터를 변환하고 값을 구해야 합니다.

기호 EXP를 F\$INTEGER 함수의 인수로 사용할 때 기호 EXP 주위에 어포스트로피('')를 삽입해야 합니다. 이렇게 하면 기호 대체의 첫 단계 동안 DCL이 EXP 값을 대체합니다.

다음 보기에서 F\$INTEGER 함수는 정수 표현식의 값을 구하는 데 사용됩니다.

```
$ INQUIRE EXP "Enter integer expression"
$ RES = F$INTEGER('EXP')
$ WRITE SYS$OUTPUT "Result is",RES
```

이 명령 프로시저의 출력은 다음과 같을 것입니다.

```
Enter integer expression: 9 + 7
Result is 16
```

“9 + 7” 값이 대체되었습니다. F\$INTEGER 함수가 인수 “9 + 7”을 처리하면 표현식을 계산하여 올바른 결과를 반환합니다.

기호의 존재 여부 확인

기호가 존재하는지 확인하려면 F\$TYPE 함수를 사용하십시오. F\$TYPE 함수는 기호가 정의되어 있지 않는 경우 null 문자열을 반환합니다. 예를 들면 다음과 같습니다.

```
.
.
.
$ IF F$TYPE(TEMP) .EQS. "" THEN TEMP = "YES"
$ IF TEMP .EQS. "YES" THEN GOTO TEMP_SEC
.
.
.
```

이 프로시저에서는 기호 TEMP가 이전에 정의되어 있었는지 테스트합니다. 정의되어 있었다면 TEMP의 현재 값이 유지됩니다. TEMP가 정의되어 있지 않으면 IF 문은 “YES” 값을 TEMP에 할당합니다.

프로세스란 사용자가 시스템과 상호 작용할 수 있는 OpenVMS 운영 체제에서 만드는 환경입니다. 프로세스는 **분리된 프로세스**(다른 프로세스와 독립된 프로세스)이거나 **하위 프로세스**(자신의 존재와 리소스가 다른 프로세스에 종속된 프로세스)일 수 있습니다. 상위 프로세스라고도 하는 주 프로세스는 분리된 프로세스입니다. 이 장에서 설명하는 내용은 다음과 같습니다.

- 프로세스 컨텍스트 해석
- 하위 프로세스 사용
- 가상 터미널의 연결이 끊긴 프로세스에 연결
- 일괄 처리 작업으로 작업

프로세스가 만들어지는 방법

사용자가 다음 작업 중 하나를 수행하면 시스템에서 프로세스를 만듭니다.

- 로그인
시스템에서 각 대화형 사용자에게 대한 프로세스를 만듭니다.
- 일괄 처리 작업 제출
시스템에서 각 일괄 처리 작업에 대한 프로세스를 만듭니다. 일괄 처리 작업이 완료되면 시스템에서 해당 프로세스를 삭제합니다.
- 하위 프로세스 생성
SPAWN 명령을 사용하면 시스템에서 프로세스를 만듭니다.
- 프로그램 실행
/DETACHED 한정자나 /UIC=uic 한정자를 사용하여 프로그램을 실행하면 시스템에서 프로세스를 만듭니다.

프로세스 컨텍스트 해석

특권, 기호, 논리 이름 등 프로세스가 사용하는 특성이 프로세스 컨텍스트를 구성합니다. 시스템은 프로세스에 고유한 특성을 **사용자 인증 파일(UAF)**로부터 취합니다. UAF는 시스템에 대한 액세스가 허용된 사용자를 나열하며 각 사용자의 프로세스에 대한 특성을 정의합니다. 일반적으로 시스템 관리자가 UAF를 유지 보수합니다. 프로세스 컨텍스트 해석은 한 번에 하나씩 시스템이 프로그램(이미지 또는 실행 이미지라고도 함)을 실행하는 프로세스 내에서 이루어집니다.

현재 프로세스에 대한 프로세스 컨텍스트를 표시하려면 **SHOW PROCESS/ALL** 명령을 입력합니다.

다음 보기에서는 프로세스 컨텍스트를 보여 줍니다.

```
11-DEC-2002 13:30:37.12 [1] User: CLEAVER [2] Process ID: 24E003DC [3]
Node: ZEUS           Process name: "CLEAVER" [4]
Terminal:            VTA2195: TNA2170: (Host: 16.32.123.45 Port: 6789)
User Identifier:     [DOC,CLEAVER] [5]
Base priority:       4 [6]
Default file spec:   DISK1:[CLEAVER] [7]
Number of Kthreads:  1
```

```
Devices allocated:  ALPHAI$VTA2195:
```

```
Process Quotas: [8]
```

```
Account name: DOC
CPU limit:                               Infinite Direct I/O limit:           1024
Buffered I/O byte count quota:           119616 Buffered I/O limit:           1024
Timer queue entry quota:                  400 Open file quota:                299
Paging file quota:                        100080 Subprocess quota:             30
Default page fault cluster:               64 AST quota:                       798
Enqueue quota:                            5000 Shared file limit:             0
Max detached processes:                   0 Max active jobs:                  0
```

```
Accounting information: [9]
```

```
Buffered I/O count:           16424 Peak working set size:           13920
Direct I/O count:             12014 Peak virtual size:             185392
Page faults:                  11113 Mounted volumes:                0
Images activated:              68
Elapsed CPU time:              0 00:04:18.55
Connect time:                  0 00:08:22.76
```

```
Authorized privileges:
```

```
NETMBX      TMPMBX
```

```
Process privileges: [10]
```

```
GROUP      may affect other processes in same group
TMPMBX     may create temporary mailbox
OPER       operator privilege
```

```

NETMBX                                may create network device

Process rights: [11]
CLEAVER                                resource
INTERACTIVE
LOCAL

System rights:
SYS$NODE_ZEUS

Auto-unshelve: on

Image Dump: off

Soft CPU Affinity: off

Parse Style: Traditional

Home RAD: 0

Scheduling class name: none

Process Dynamic Memory Area [12]
Current Size (Kb)          128.00    Current Size (Pagelets)      256
Free Space (Kb)            111.18    Space in Use (Kb)           16.81
Largest Var Block (Kb)     109.69    Smallest Var Block (bytes)   8
Number of Free Blocks      10        Free Blocks LEQU 64 Bytes    4

There is 1 process in this job: [13]
CLEAVER (*)
    
```

보기를 검토하면서 다음 사항에 주의하십시오.

[1] 현재 날짜 및 시간

SHOW PROCESS/ALL 명령이 실행되는 날짜와 시간입니다.

[2] 사용자 이름

프로세스에 관련된 계정에 할당된 사용자 이름입니다.

[3] PID(프로세스 ID) 번호

시스템에서 프로세스에 할당한 고유 번호입니다. SHOW PROCESS 명령은 PID 번호를 16진수로 표시합니다.

[4] 프로세스 이름

프로세스에 할당된 이름입니다. 프로세스 이름은 특정 UIC 그룹 내에서 고유하므로, 한 계정으로 로그인된 첫 번째 프로세스에 사용자 이름이 할당됩니다. 동일한 계정으로 로그인된 후속 프로세스에는 터미널 이름이 할당됩니다. DCL 명령 SET PROCESS/NAME을 사용하여 프로세스 이름을 변경할 수 있습니다.

[5] UIC(사용자 식별 코드)

프로세스에 관련된 계정에 할당된 그룹 및 구성원 번호(또는 문자)입니다(예:

[DOC,CLEAVER]). 해당 UIC 중 일부는 사용자가 속하는 그룹을 식별합니다. 그룹 내에서는 사용자는 그룹 간에서 보다 더 자유롭게 파일이나 시스템 리소스를 공유할 수 있도록 허용됩니다.

[6] 우선 순위

프로세스의 현재 우선 순위입니다.

[7] 기본 파일 지정

현재 장치 및 디렉토리입니다. DCL 명령 SET DEFAULT를 사용하여 현재 기본값을 변경합니다.

[8] 프로세스 할당량

프로세스에 관련된 할당량(한계)입니다. SHOW PROCESS 명령의 /QUOTAS 또는 /ALL 한정자를 사용하여 이러한 할당량을 점검합니다.

[9] 계정 정보

프로세스가 사용하는 메모리와 CPU 시간에 대해 계속적으로 업데이트되는 계정입니다. SHOW PROCESS 명령의 /ACCOUNTING 또는 /ALL 한정자를 사용하여 이 정보를 점검합니다.

[10] 프로세스 특권

프로세스에 부여된 특권입니다. 특권은 특정 시스템 작업의 수행을 특정 사용자에게로 제한합니다. SHOW PROCESS 명령의 /PRIVILEGES 또는 /ALL 한정자를 사용하여 특권을 점검합니다.

[11] 프로세스 권한

사용 제어 목록(ACL) 보호와 함께 사용되는 시스템 정의 증명서입니다. 증명서는 ACL에서 사용자를 지정하는 방법을 제공합니다. ACL은 파일, 장치, 우편함 등의 객체 사용자에게 부여하거나 거부할 액세스의 종류를 정의하는 보안 도구입니다.

[12] 프로세스 동적 메모리 영역

프로세스가 현재 사용하는 동적 메모리입니다. 이미지가 실행 중인 경우 시스템에서 동적 메모리를 해당 이미지에 할당합니다. 한 프로세스에서 해당 메모리를 더 이상 필요로 하지 않으면 시스템은 이 메모리를 다른 프로세스에 할당합니다. **SHOW PROCESS** 명령의 **/MEMORY** 또는 **/ALL** 한정자를 사용하여 이 정보를 점검합니다.

[13] 이 트리의 프로세스

상위 프로세스에 속하는 하위 프로세스의 목록입니다. 현재 프로세스 다음에 별표(*)가 나타납니다. **SHOW PROCESS/SUBPROCESSES** 또는 **/ALL** 명령을 사용하여 이 목록을 점검합니다.

분리된 프로세스 사용

분리된 프로세스는 상위 프로세스에 따라 대화형이거나 비대화형입니다. DCL 명령 `RUN` 또는 `Create Process` 시스템 서비스(`$CREPRC`)에 제공한 인수에 따라 사용자나 운영 체제가 로그인을 수행합니다. `RUN`과 `$CREPRC` 모두 `SYS$SYSTEM`에서 `LOGINOUT.EXE` 이미지를 실행합니다.

하위 프로세스 사용

SPAWN 명령을 사용하면 현재 프로세스의 하위 프로세스를 만들 수 있습니다. 이 하위 프로세스 내에서 시스템과 상호 작용하고 하위 프로세스에서 로그아웃하여 상위 프로세스로 돌아가거나 상위 프로세스와 하위 프로세스 간에 전환할 수 있습니다. 실행 중인 프로세스는 항상 하나뿐입니다.

시스템의 각 사용자는 **작업 트리**로 표시됩니다. 작업 트리는 주 프로세스가 맨 위에 있는 모든 프로세스 및 하위 프로세스의 계층 구조입니다. 하위 프로세스는 상위 프로세스에 종속되며 상위 프로세스가 종료되면 삭제됩니다. 기본적으로 하위 프로세스의 이름은 상위 프로세스의 이름과 밑줄 및 고유 번호로 이루어집니다. 예를 들어, 상위 프로세스 이름이 DOUGLASS이면 하위 프로세스 이름은 DOUGLASS_1, DOUGLASS_2 등으로 지정됩니다.

하위 프로세스를 사용하여 작업 생성

작업을 중단하고, 두 번째 작업을 수행한 다음 원래 작업으로 돌아가려면 **Ctrl/Y**를 사용하여 첫 번째 작업을 중단하고, 두 번째 작업을 수행할 하위 프로세스를 생성하고, 해당 하위 프로세스를 종료한 다음 **CONTINUE** 명령을 입력하여 첫 번째 작업으로 돌아갈 수 있습니다. 기본적으로 하위 프로세스를 생성하는 경우 상위 프로세스가 최대 절전 모드로 되며 사용자에게 해당 하위 프로세스 내에서 **DCL** 수준의 제어권이 부여됩니다. 사용자의 기본 디렉토리는 상위 프로세스의 현재 디렉토리입니다. 예를 들어, **Ctrl/Y**를 눌러 **EVE** 편집 세션을 중단할 경우 **CONTINUE** 명령을 입력하고 **Ctrl/W**를 눌러 화면을 새로 고칩니다.

하위 프로세스를 사용하여 여러 작업 수행

원래 작업 수행을 계속하면서 두 번째 작업을 수행하려면 **SPAWN/NOWAIT** 명령을 사용하여 하위 프로세스를 만들 수 있습니다. **SPAWN/NOWAIT**는 일괄 처리형의 비대화형 하위 프로세스를 생성하며 입력이 필요 없는 명령을 실행하는 데만 사용됩니다.

상위 및 하위 프로세스가 모두 동시에 실행되고 있기 때문에 두 프로세스가 모두 터미널을 제어하려고 시도합니다. 충돌을 막으려면 다음을 지정하십시오.

- **/OUTPUT** 한정자

하위 프로세스가 터미널이 아닌 지정된 파일에 출력을 기록해야 함을 나타냅니다.

- SPAWN 명령 매개 변수 또는 /INPUT 한정자

하위 프로세스가 터미널에서 입력을 읽는 것이 아니라 지정된 명령을 실행해야 함을 나타냅니다.

SPAWN 명령의 /INPUT 한정자를 지정하면 하위 프로세스는 파일 끝 표시기나 심각한 오류가 발생하면 종료되는 비대화형 프로세스로 만들어집니다. DCL 수준에서는 Ctrl/Z는 파일 끝 표시기로 처리됩니다.

하위 프로세스 만들기

사용자가 만드는 각각의 프로세스는 고유하기 때문에, 일반적으로 한 프로세스에서 실행된 명령이 다른 프로세스에 영향을 주지 않습니다. 그러나 터미널에 대한 제어권이 프로세스 간에 전달되므로, 터미널 특성에 영향을 주는 명령(예: SET TERMINAL)은 해당 터미널을 제어하는 프로세스에 영향을 미칩니다. 예를 들어, 한 프로세스에서 에코를 금지한 다음 다시 복원하지 않은 상태로 종료하면 터미널에 대한 제어권을 받는 다음 프로세스에 대해 에코가 금지된 상태로 남아 있게 됩니다. 변경된 모든 터미널 특성을 SET TERMINAL 명령을 사용하여 다시 설정합니다.

다음 보기에서는 사용자가 Ctrl/Y를 눌러서 명령 이미지(TYPE 명령)를 중단하고, 하위 프로세스를 생성하고, 이 하위 프로세스를 종료한 다음, 원래 프로세스로 돌아갑니다.

```
$ TYPE MICE.TXT
Once the weather turns cold, mice may find a crack in the foundation and
enter your house. They are looking for food and shelter from the harsh
weather ahead.

.
.
.

Ctrl/Y
$ SPAWN
%DCL-S-SPAWNED, process DOUGLASS_1 spawned
%DCL-S-ATTACHED, terminal now attached to process DOUGLASS_1
$ MAIL
MAIL>

.
.
.

MAIL> EXIT
$ LOGOUT
Process DOUGLASS_1 logged out at 31-DEC-1999 12:42:12.46
%DCL-S-RETURNED, control returned to process DOUGLASS
$ CONTINUE
Once inside, they may gnaw through electrical wires and raid your food.
```

Because mice reproduce so quickly, what started as one or two mice can quickly become an invasion. If you seal the cracks and holes on the exterior of your foundation, you can prevent these rodents from ever getting in.

하위 프로세스 종료

SPAWN 명령으로 만든 하위 프로세스를 종료하려면 다음 명령 중 하나를 사용합니다.

- LOGOUT

LOGOUT 명령을 사용하여 하위 프로세스를 종료하면 이 하위 프로세스는 자신이 만든 모든 하위 프로세스와 함께 삭제되고 사용자는 상위 프로세스로 돌아갑니다.

- ATTACH

ATTACH 명령을 사용하여 하위 프로세스를 종료하면 하위 프로세스가 최대 절전 모드로 되고 터미널에 대한 제어권은 지정된 프로세스로 전송됩니다. 프로세스 이름을 ATTACH 명령에 대한 매개 변수로 지정하거나 PID(프로세스 ID) 번호를 ATTACH 명령의 /IDENTIFIER 한정자 값으로 지정해야 합니다.

다음 보기에서는 DOUGLASS_1 하위 프로세스를 종료하고 DOUGLASS 프로세스에 연결하는 방법을 보여 줍니다.

```
$ ATTACH DOUGLASS
%DCL-S-RETURNED, control returned to process DOUGLASS
$ SHOW PROCESS
11-DEC-2002 10:34:58.50   User: DOUGLASS           Process ID:   2061C478
Node: ALPHAI             Process name: "DOUGLASS"
Terminal:                 VTA2195:  TNA2170:  (Host: 16.32.123.45 Port: 6789)

User Identifier:         [DOC,DOUGLASS]
Base priority:           4
Default file spec:       DISK1:[DOUGLASS]
Number of Kthreads:      1

Devices allocated:       ALPHAI$VTA2195:

Soft CPU Affinity:      off
```

하위 프로세스 컨텍스트

하위 프로세스 컨텍스트란 하위 프로세스가 상위 프로세스로부터 상속받은 환경입니다. 기본적으로 하위 프로세스가 상속 받는 항목은 기본값, 특권, 기호, 논리 이름, 제어 문자, 메시지 형식, 확인 상태, 키 정의 등입니다. 이러한 항목들은 집합적으로 하위 프로세스에 대한 환경을 만듭니다.

상위 프로세스로부터 상속되지 않는 항목은 다음과 같습니다.

- **PID(프로세스 ID) 번호**

만든 각각의 하위 프로세스에 고유한 **PID** 번호를 시스템에서 할당합니다.

- **프로세스 이름**

기본적으로 하위 프로세스 이름은 상위 프로세스의 이름과 밑줄 및 정수로 구성됩니다. **SPAWN** 명령의 **/PROCESS** 한정자를 사용하여 기본값이 아닌 다른 프로세스 이름을 지정합니다. 프로세스 이름은 고유해야 합니다.

- **만들어진 명령**

SET COMMAND 명령을 사용하여 상위 프로세스에서 정의한 명령은 하위 프로세스로 복사되지 않습니다. 만들어진 명령을 하위 프로세스에서 사용하려면 **SET COMMAND**를 사용하여 하위 프로세스에 대한 해당 명령을 만들어야 합니다.

- **특권 인증**

사용자가 하위 프로세스를 생성하는 경우, 이 프로세스 컨텍스트에는 상위 프로세스의 특권은 포함되지만, 사용자가 사용할 수 있도록 인증된 특권은 포함되지 않습니다. 예를 들어, **Mail**을 실행하는 동안 하위 프로세스를 생성하고 특권이 있는 작업을 수행하려는 경우, **Mail**을 호출하려면 먼저 상위 프로세스에서 적절한 특권을 설정해야 합니다.

다음 SPAWN 명령 한정자를 사용하여 하위 프로세스에 이러한 많은 항목들이 상속되는 것을 방지할 수 있습니다.

표 16-1

SPAWN 명령 한정자	상속 또는 변경된 항목
/CARRIAGE_CONTROL, /PROMPT	DCL 프롬프트
/NOCLI	CLI(명령 언어 인터프리터, 기본값: DCL)
/NOKEYPAD	키패드 정의
/NOLOGICAL_NAMES	논리 이름
/NOSYMBOL	기호

/SYMBOL 및 /LOGICAL_NAMES 한정자는 시스템 정의 기호(\$SEVERITY와 \$STATUS 등) 또는 시스템 정의 논리 이름(SYS\$COMMAND와 SYS\$OUTPUT 등)에 영향을 미치지 않습니다.

논리 이름과 기호를 하위 프로세스에 복사하는 데는 시간이 많이(몇 초 정도) 걸리므로, 하위 프로세스에서 논리 이름이나 기호를 사용하지 않으려는 경우에는 SPAWN 명령에 /NOLOGICAL_NAMES 및 /NOSYMBOL 한정자를 사용하려고 할 수도 있습니다. 하위 프로세스를 자주 사용할 경우에는 ATTACH 명령을 사용하는 것이 하위 프로세스를 시작하고 종료하기 위한 가장 효과적인 방법입니다. 이 방법을 사용하면 시스템에서 새로운 하위 프로세스를 만들 때까지 반복해서 기다리지 않고 상위 프로세스와 하위 프로세스 간에 제어권을 신속하게 전송할 수 있습니다.

가상 터미널의 연결이 끊긴 프로세스에 연결

가상 터미널은 사용할 수 있지만 모뎀 회선 연결이 끊긴 경우, 프로세스는 시스템에서 연결이 끊긴 가상 터미널 프로세스로 활성화된 상태로 유지됩니다. 사용자는 시스템 관리자가 지정한 시간(기본값은 900초 또는 15분) 안에 프로세스에 다시 연결해야 합니다. 이 시간이 만료될 때까지 프로세스에 다시 연결하지 못하면 시스템에서 이 프로세스를 삭제합니다.

주

UIC(사용자 식별 코드)에 관련된 가상 터미널 프로세스에만 연결할 수 있습니다.

터미널 연결 해제

다음과 같은 상황에서는 터미널의 연결이 끊길 수 있습니다.

- 호스트와 터미널 간의 모뎀 신호가 손실됩니다.
- `TT2$M_SECURE` 특성 집합을 사용하여 터미널의 **BREAK** 키를 누릅니다.
- `DCL` 명령 `DISCONNECT`를 입력합니다.
- `DCL` 명령 `CONNECT/CONTINUE`를 입력합니다.

프로세스의 연결이 끊긴 경우 이전 프로세스에 다시 연결하여 연결이 끊기기 전의 상태로 돌아가는 옵션을 사용할 수 있습니다. 로그인을 하면 시스템에서 다음과 같은 메시지를 표시합니다.

```
You have the following disconnected process:
Terminal   Process name   Image name
VTA52:     RWOODS          (none)
Connect to above listed process [YES]:
```

Return 키를 누르거나 **Yes**를 입력하면 마치 `DCL` 명령 `CONNECT/CONTINUE`의 자동 실행이 수행된 것처럼 현재 프로세스에서 로그아웃됩니다. **No**를 입력하는 경우 또는 응답을 너무 오래 연기하여 응답 시간 초과가 발생하는 경우에는 새 프로세스에 로그인된 상태로 유지됩니다. 이전 프로세스에는 연결할 수 없게 됩니다.

연결이 끊긴 세션이 여러 개 있는 경우 다시 연결할 가상 터미널의 이름을 입력하라는 메시지가 표시됩니다. 표시된 세션 중 어느 것에도 연결하지 않으려면 **No**를 입력합니다.

연결이 끊긴 프로세스 제거

일정 시간이 지나면 시스템에서 연결이 끊긴 프로세스를 자동으로 제거합니다. 그러나 다음과 같이 연결이 끊긴 프로세스에서 바로 로그아웃하면 시스템 리소스를 보존할 수 있습니다.

표 16-2

단계	작업
1	DCL 명령 SHOW USERS 를 입력하여 연결이 끊긴 다른 작업이 있는지 확인합니다.
2	DCL 명령 CONNECT/LOGOUT 을 입력하여 현재 프로세스에서 로그아웃합니다. 존재하는 마지막 프로세스에 도달할 때까지 각각의 관련 가상 터미널(터미널 접두사 VTA 로 표시됨)을 통해 다시 연결합니다.
3	DCL 명령 LOGOUT 을 입력합니다.

연결이 끊긴 프로세스 관리

가상 터미널을 사용하면 한 번에 연결이 끊긴 프로세스를 두 개 이상 유지 보수할 수 있습니다. 그러나 가상 터미널에 로그인된 동안에는 물리적 터미널의 연결이 끊겨 있음을 염두에 두어야 합니다. 현재 가상 터미널 프로세스에 연결된 물리적 터미널 외의 다른 장치로 보내지는 모든 **I/O** 요청은 대기 상태에 들어갑니다. 시간 초과 기간이 만료되면 지연 중인 프로세스가 종료됩니다. 그러나 **I/O** 요청을 받을 물리적 터미널에 다시 연결하면 대기 중 상태에 들어간 지점에서 프로세스가 계속됩니다. 해당 컨텍스트에 관련된 이름으로 각 프로세스를 명명하면 원하는 프로세스에 쉽게 다시 연결할 수 있습니다.

예를 들어, 파일 편집을 위해 프로세스를 실행하는 **SMITH**라는 사용자는 **SET PROCESS/NAME** 명령을 사용하여 프로세스를 **SMITH_EDIT**로 명명할 수 있습니다. **SMITH**는 나중에 편집을 계속하기 위한 적절한 프로세스를 쉽게 확인할 수 있습니다.

시스템 관리자는 시스템에서 또는 터미널 단위로 가상 터미널의 사용을 제한할 수 있습니다.

일괄 처리 작업으로 작업

일괄 처리 작업은 비대화형 프로세스입니다. 일괄 처리 작업은 자체 프로세스에서 실행되므로 서로 다른 작업을 수행하는 프로세스가 동시에 두 개 이상 있을 수 있습니다. 예를 들어, 다음과 같은 작업을 하는 데 일괄 처리 작업을 사용할 수 있습니다.

- 시스템에서 프로그램이나 명령 프로시저를 일괄 처리 모드로 실행하는 동안 작업을 대화형으로 수행합니다.
- 실행하는 데 시간이 오래 걸리는 명령 프로시저를 실행합니다.
- 명령 프로시저나 프로그램을 정규 업무 시간 이후에 실행합니다.
- 특정 프로그램의 우선 순위를 낮추어서 실행합니다(예: 프로그램에서 적절하지 않은 양의 시스템 리소스를 사용하는 경우).

일괄 처리 작업 제출

일괄 처리 작업을 제출하면 시스템에서 사용자의 계정과 프로세스 특성을 사용하여 분리된 프로세스를 만듭니다. 시스템은 해당 프로세스로부터 작업을 실행하고 작업이 완료되면 프로세스를 삭제합니다. 또한 시스템 로그인 명령 프로시저(SYLOGIN.COM)와 사용자의 로그인 명령 프로시저(LOGIN.COM)를 실행한 다음 일괄 처리 작업의 명령 프로시저도 실행합니다. 이러한 프로시저가 실행되면서 로그 파일에 출력이 기록됩니다. 일괄 처리 작업이 완료되면 로그 파일을 인쇄하거나 사용자의 디렉토리 중 하나에 저장할 수 있습니다.

작업을 일괄 처리 모드로 실행하려면 DCL 명령 **SUBMIT**을 입력하여 작업을 일괄 처리 대기열(실행을 기다리고 있는 일괄 처리 작업의 목록)에 제출합니다. 작업을 제출하면 이 작업은 기본 일괄 처리 대기열인 **SYS\$BATCH**로 보내집니다. 여기서 실행되기를 기다리고 있는 작업 대기열의 끝에 추가됩니다. 해당 작업은 이 작업의 앞에 있는 작업들이 완료되면 실행됩니다. **OpenVMS** 시스템에서는 동시에 실행될 수 있는 일괄 처리 작업의 개수가 시스템 관리자가 일괄 처리 대기열을 만들 때 지정됩니다. 기본적으로 **SUBMIT** 명령은 **.COM** 파일 유형을 사용합니다.

다음 보기에서는 **JOB1.COM**을 **SYS\$BATCH**에 넣습니다.

```
$ SUBMIT JOB1
Job JOB1 (queue SYS$BATCH, entry 651, started on SYS$BATCH)
```

시스템에서 작업의 이름, 작업이 포함된 대기열, 작업에 할당된 항목 번호 등을 표시합니다. 작업이 일괄 처리 대기열에 제출된 후 **DCL** 프롬프트가 나타납니다. **DCL** 명령(예: **DELETE/ENTRY**)에서 일괄 처리 작업을 참조해야 할 경우에는 작업 항목 번호를 사용합니다. **SHOW ENTRY** 명령을 사용하면 작업 항목 번호를 얻을 수 있습니다. 여러 프로시저가 하나의 일괄 처리 작업으로 제출되는 경우, 어느 한 프로시저가 오류 또는 치명적인(심각한) 시스템 메시지와 함께 끝나면 해당 일괄 처리 작업이 종료됩니다.

일괄 처리 작업은 일괄 처리 대기열로 제출될 때 반드시 실행이 시작되어야 하는 것은 아닙니다. 다른 시간을 지정하려면 **SUBMIT/AFTER** 명령을 입력합니다. 다음 보기에서는 작업이 **11:30 P.M.** 후에 제출됩니다.

```
$ SUBMIT/AFTER=23:30 JOB1.COM
```

일괄 처리 실행을 위해 명령 프로시저를 제출하면 시스템에서 버전 번호를 비롯한 해당 명령 프로시저에 대한 전체 파일 지정을 저장합니다. 명령 프로시저를 제출한 후 업데이트할 경우, 일괄 처리 작업은 새 버전이 아니라 제출한 명령 프로시저의 버전을 실행합니다.

사용자의 로그인 기본값은 일반적으로 명령 프로시저에서 나온 파일을 액세스하는 데 필요한 기본값이 아니므로, 다음 방법 중 하나를 사용하여 정확한 파일이 액세스되도록 해야 합니다.

- 전체 파일 지정 사용 — 명령 프로시저의 파일을 참조하거나 명령 프로시저에 파일을 전달하는 경우 장치 및 디렉토리 이름을 파일 지정의 일부로 포함합니다.
- **SET DEFAULT** 명령 사용 — 명령 프로시저의 파일을 참조하려면 명령 프로시저의 시작 부분에서 **SET DEFAULT** 명령을 사용하여 적절한 장치와 디렉토리를 지정합니다.

일괄 처리 작업이 실행되면서 로그 파일에 출력이 기록됩니다. 기본적으로 로그 파일 이름은 파일 유형 **.LOG**로 제출한 명령 프로시저와 동일합니다. 작업이 완료되면 시스템에서 로그 파일을 인쇄하고 사용자 디렉토리에서 이 파일을 삭제합니다. 로그 파일 저장에 대한 자세한 내용은 675페이지의 “일괄 처리 작업 출력 제어”를 참조하십시오.

로그인 명령 프로시저에서 일괄 처리 작업 검사

일괄 처리 작업을 제출할 때마다 시스템에서 사용자의 로그인 명령 프로시저를 실행합니다. 일괄 처리 작업을 테스트하는 데 **F\$MODE** 렉시컬 함수를 사용하여 일괄 처리 작업을 실행하는 경우 로그인 명령 프로시저의 섹션이 포함되거나 생략되도록 할 수 있습니다.

다음 보기의 로그인 명령 프로시저는 일괄 처리 작업에만 사용되는 명령, 논리 이름 및 기호를 포함합니다. 섹션에는 **BATCH_COMMANDS**라고 표시되고 로그인 명령 프로시저의 시작 부분에 다음 명령이 포함됩니다.

```
IF F$MODE() .EQS. "BATCH" THEN GOTO BATCH_COMMANDS
.
.
.
```

일괄 처리 작업을 제출할 때 시스템에서 로그인 명령 프로시저의 어떤 명령도 실행하지 못하도록 하려면 프로시저의 시작 부분에 다음 명령을 넣습니다.

```
IF F$MODE() .NES. "INTERACTIVE" THEN EXIT
```

로그인 명령 프로시저의 어느 위치에든 이 명령을 둘 수 있습니다. 일괄 처리 작업을 제출하면 시스템에서 이전 명령이 있는 지점까지만 로그인 명령 프로시저를 실행합니다.

여러 명령 프로시저 제출

SUBMIT 명령을 입력하는 경우 하나의 작업에서 실행할 명령 프로시저를 여러 개 지정할 수 있습니다. **/NAME** 한정자를 사용하여 이름을 지정하지 않을 경우, **SUBMIT** 명령은 첫 번째 명령 프로시저의 이름을 작업 이름으로 사용합니다. 오류로 인해 작업에 있는 어느 한 명령 프로시저라도 종료되면 전체 작업이 종료됩니다.

일괄 처리 작업이 실행되는 경우, 첫 번째 프로시저(**UPDATE.COM**)의 작동 컨텍스트가 두 번째 프로시저(**SORT.COM**)용으로 보존되지 않습니다. 시스템에서 **SORT.COM**이 실행되기 전에 **UPDATE.COM**이 만든 로컬 기호를 삭제하지만 전역 기호는 보존됩니다.

단일 작업 내에서 개별 명령 프로시저에 대해 서로 다른 매개 변수를 지정할 수 없습니다.

다음 보기에서는 **SUBMIT** 명령이 **UPDATE.COM**을 실행한 다음 **SORT.COM**을 실행하는 일괄 처리 작업을 만듭니다.

```
$SUBMIT UPDATE,SORT
Job UPDATE (queue SYS$BATCH, entry 207) started on SYS$BATCH
```

다음 보기에서는 **UPDATE.COM**과 **SORT.COM**에 동일한 두 개의 매개 변수를 전달합니다.

```
$ SUBMIT UPDATE, SORT/PARAMETERS = -
_$(DISK1:[ACCOUNT.BILLS]DATA.DAT, DISK2:[ACCOUNT]NAME.DAT)
$ Job UPDATE (queue SYS$BATCH, ENTRY 208) started on SYS$BATCH
```

일괄 처리 작업에 데이터 전달

일괄 처리 작업에 대한 기본 입력 스트림(**SYS\$INPUT**)은 실행되고 있는 명령 프로시저입니다. 분리된 프로세스가 일괄 처리 작업을 실행 중인 경우, 대화형으로 실행하는 명령 프로시저로 재정의할 수 있는 것과 같이 터미널에 대해 **SYS\$INPUT**을 재정의할 수 없습니다. 일괄 처리 작업에 입력을 전달하려면 다음 방법 중 하나를 사용합니다.

- 데이터를 명령 프로시저 자체에 포함합니다.
데이터를 명령 프로시저에 포함하려면 명령 또는 이미지 다음 행에 데이터를 넣습니다.
- 일시적으로 **SYS\$INPUT**을 파일로 정의합니다.
일시적으로 **SYS\$INPUT**을 파일로 정의하려면 **DEFINE/USER_MODE** 명령을 사용합니다.
- 명령 프로시저를 실행하기 위해 제출할 때 명령 프로시저에 매개 변수를 전달합니다.
명령 프로시저에 매개 변수를 전달하려면 일괄 처리 작업을 제출할 때 **/PARAMETERS** 한정자를 사용합니다.

단일 작업 내에서 개별 명령 프로시저에 대해 서로 다른 매개 변수를 지정할 수 없습니다. 다른 매개 변수 그룹을 전달해야 할 경우에는 별도의 **SUBMIT** 명령을 사용합니다.

다음 보기에서는 **AVERAGE.EXE** 이미지에 데이터 줄을 전달합니다.

```
$! Execute AVERAGE.EXE
$ RUN AVERAGE
647
899
532
401
$ EXIT
```

다음 보기에서는 **SYS\$INPUT**이 일시적으로 파일로 정의됩니다.

```
$ DEFINE/USER_MODE SYS$INPUT STATS.DAT
$ RUN AVERAGE
$ EXIT
```

다음 보기에서는 **EMPLOYEES.DAT** 파일의 매개 변수가 **CHECKS.DAT** 명령 프로시저로 전달됩니다.

```
$ SUBMIT/PARAMETERS=(DISK1:[PAYROLL]EMPLOYEES.DAT) CHECKS
Job CHECKS (queue SYS$BATCH, entry 209) started on SYS$BATCH
```

주

SHOW QUEUE/FULL 명령은 일괄 처리 대기열의 작업에 대한 전체 정보를 표시합니다. 표시되는 정보에는 프로시저에 전달하는 매개 변수도 포함됩니다. 따라서 일괄 처리 작업에 암호와 같은 기밀 정보를 전달해서는 안 됩니다.

일괄 처리 작업 출력 제어

기본적으로 로그 파일은 일괄 처리 작업의 첫 번째 명령 프로시저와 이름이 동일하며 .LOG 파일 유형을 갖습니다. 시스템에서는 일괄 처리 작업의 출력을 매 분마다 한 번씩 로그 파일에 기록합니다. 시간 간격을 다르게 지정하려면 명령 프로시저에 SET OUTPUT_RATE 명령을 포함시킵니다.

시스템이 로그 파일에 기록하는 동안 EDT 편집기를 사용하여 로그 파일을 읽으려고 하면 다른 사용자에게 의해 파일이 잠겨져 있다는 메시지가 나타납니다. 몇 초 정도 기다린 후 다시 시도하십시오. 그러나 EVE 편집기를 사용하면 일괄 처리 작업의 로그 파일을 읽을 수 있습니다. EDIT/TPU/READ_ONLY와 로그 파일 이름을 지정하면 EVE 명령을 사용하여 로그 파일을 이동하고 파일에 대해 변경한 내용이 저장되지 않도록 할 수 있습니다. /READ_ONLY 한정자를 생략하고 로그 파일을 어떤 식으로든 수정하면 일괄 처리 작업이 종료됩니다.

일괄 처리 작업은 사용자 이름으로 로그인하고 로그인 명령 프로시저를 실행하는 프로세스이므로, 일괄 처리 작업의 출력에는 로그인 명령 프로시저의 내용이 포함됩니다. 이 출력에는 일괄 처리 작업 로그 파일에 기록된 모든 것(명령 프로시저 출력, 오류 메시지 등)과 전체 로그아웃 메시지도 포함됩니다. 로그인 명령 프로시저가 일괄 처리 로그 파일에 기록되지 않게 하려면 로그인 명령 프로시저의 시작 부분에 다음 명령을 추가합니다.

```
$ IF F$MODE() .EQS. "BATCH" THEN SET NOVERIFY
```

기본적으로 로그 파일 이름은 작업을 제출하는 데 사용한 이름입니다. 또한 기본적으로 로그 파일에는 파일 유형 .LOG와 로그인 기본값으로 지정된 장치와 디렉토리가 있습니다. 작업을 제출할 때 다른 로그 파일 이름을 지정하려면 SUBMIT 명령에 /LOG_NAME 한정자를 사용합니다.

일괄 처리 작업 로그 파일에는 SYS\$OUTPUT과 SYS\$ERROR에 대한 모든 출력이 포함됩니다. 또한 명령 프로시저에서 실행된 모든 명령행도 기본적으로 포함됩니다. 명령행이 인쇄되지 않도록 하려면 명령 프로시저에서 SET NOVERIFY 명령 또는 F\$VERIFY 렉시칼 함수를 사용합니다. 작업이 완료되면 시스템에서 긴 양식의 시스템 로그아웃 메시지를 사용하여 작업 종료 정보를 로그 파일에 기록합니다.

SET VERIFY 명령이 실행되고 있는 경우 SET PREFIX 명령을 사용하여 각 명령행에 **time-stamp**를 지정하여 각 명령이 실행되는 시기를 정확하게 알 수도 있습니다.

일괄 처리 작업이 성공적으로 완료되지 못하면 로그 파일을 점검하여 명령 프로시저가 실패한 지점과 이 실패를 초래한 오류 상태를 확인할 수 있습니다.

로그 파일 저장

로그 파일을 저장하려면 /KEEP 또는 /NOPRINTER 한정자를 사용합니다. /KEEP 한정자는 로그 파일을 인쇄한 후 이 파일을 저장합니다. /NOPRINTER 한정자는 로그를 인쇄하지 않은 상태로 저장합니다. 이 두 한정자를 어느 것도 지정하지 않으면 기본 작업이 수행됩니다. 로그 파일이 기본 인쇄 대기열 SYS\$PRINT에 대해 대기되고 인쇄된 후 삭제됩니다. /KEEP과 /NOPRINTER 한정자는 기본 로그인 디렉토리에 로그 파일을 저장합니다. 로그 파일은 일괄 처리 작업의 첫 번째 명령 프로시저와 동일한 이름과 파일 유형 .LOG를 갖습니다. 대체 파일 이름과 디렉토리 이름 중 하나 또는 둘 다를 지정하려면 /LOG_FILE 한정자를 사용합니다. 로그 파일 이름을 바꾸어 저장하려면 /LOG_FILE과 함께 /KEEP 또는 /NOPRINTER 중 하나를 사용해야 합니다.

다음 보기에서는 로그 파일이 DISK2:[JONES.RESULTS]UPDATE.LOG라는 파일에 저장됩니다.

```
$ SUBMIT/LOG_FILE=DISK2:[JONES.RESULTS]/NOPRINTER -  
_ $ DISK2:[JONES.RESULTS]UPDATE
```

로그 파일 읽기

TYPE 명령을 사용하여 로그 파일을 읽으면 일괄 처리 작업이 얼마나 완료되었는지 확인할 수 있습니다. 그러나 시스템에서 로그 파일에 기록 중인 동안에 로그 파일을 표시하려고 시도하면 다른 사용자에게 의해 파일이 잠겨져 있다는 메시지가 나타납니다. 이 메시지가 나타나면 몇 초 정도 기다린 후 다시 시도하십시오.

일괄 처리 작업 로그에 명령 출력 포함

일반적으로 프로그램을 컴파일하고, 링크로 연결하고, 실행하는 일괄 처리 작업 명령 프로시저는 컴파일러 목록이나 링커 맵 같은 추가적인 인쇄 출력을 만듭니다. 이러한 파일에 대한 인쇄 복사본을 만들기 위해, 일괄 처리 작업 명령 프로시저가 해당 파일을 인쇄하는 데 필요한 PRINT 명령을 포함할 수 있습니다.

컴파일러의 인쇄 목록 또는 링커 출력 파일 등 명령 프로시저의 모든 출력을 일괄 처리 작업 로그에 포함시키려는 경우, 다음 중 하나를 수행할 수 있습니다.

- 명령 프로시저에서 **PRINT** 명령 대신 **TYPE** 명령을 사용합니다. **TYPE** 명령은 **SYS\$OUTPUT**에 기록합니다. 일괄 처리 작업에서 **SYS\$OUTPUT**은 일괄 처리 작업 로그 파일에 해당합니다.
- 적절한 명령에 대해 한정자를 사용하여 출력을 **SYS\$OUTPUT**으로 지정합니다.
이 방법을 사용하는 경우 로그 파일을 저장하지 않으면 출력 파일이 디스크에 저장되지 않습니다.

다음 보기에 나오는 명령 프로시저의 처리가 완료되면 일괄 처리 작업 로그, 컴파일러 목록, 링커 맵 등 별도의 출력 세 개가 만들어집니다.

```
$ FORTRAN/LIST BIGCOMP
$ PRINT BIGCOMP.LIS
$ LINK/MAP/FULL BIGCOMP
$ PRINT BIGCOMP.MAP
```

다음 보기에서는 한정자를 사용하여 **SYS\$OUTPUT**으로 출력을 지정하는 방법을 보여 줍니다.

```
$ FORTRAN/LIST=SYS$OUTPUT BIGCOMP
$ LINK/MAP=SYS$OUTPUT/FULL BIGCOMP
```

이러한 명령이 일괄 처리 작업으로 실행되면 컴파일러와 링커의 출력 파일은 로그 파일에 직접 기록됩니다.

일괄 처리 작업 특성 변경

작업을 대기열에 제출했지만 이 작업 실행이 시작되기 전에는 **SET ENTRY** 또는 **SET QUEUE/ENTRY** 명령을 적절한 한정자와 함께 사용하여 해당 작업에 관련된 특성을 변경할 수 있습니다.

다음 보기에서는 일괄 처리 대기열에서 보류 중인 일괄 처리 작업 이름을 변경할 수 있는 두 가지 방법을 보여 줍니다.

```
$ SET QUEUE/ENTRY=209/NAME=NEW_NAME SYS$BATCH
$ SET ENTRY 209 /NAME=NEW_NAME
```

이 두 명령은 모두 작업 번호 209의 이름을 **NEW_NAME**으로 변경합니다.

다음 목록에는 **SET ENTRY** 또는 **SET QUEUE/ENTRY** 명령으로 변경할 수 있는 일부 사항들이 있습니다. 전체 한정자 목록을 보려면 *OpenVMS DCL Dictionary*를 참조하십시오. 또한, **SUBMIT** 명령에 사용할 수 있는 대부분의 한정자는 **SET ENTRY**와 **SET QUEUE/ENTRY** 명령과 사용할 수 있습니다.

다음과 같은 변경 작업을 수행할 수 있습니다.

- 작업 처리를 지연시킵니다.
`/AFTER` 한정자를 사용하여 작업이 실행될 때까지 대기하는 시간을 지정합니다.
`/HOLD` 한정자를 사용하여 명시적으로 해제할 때까지 작업을 중지합니다.
- 작업을 해제합니다.
`/NOHOLD` 또는 `/RELEASE` 한정자를 사용하여 `/HOLD` 또는 `/AFTER` 한정자와 함께 제출된 작업을 릴리즈합니다.
- 작업을 다른 대기열로 보냅니다.
`/REQUEUE` 한정자를 사용하여 작업이 실행될 대기열을 변경합니다.
- 실행 특성을 변경합니다.
작업 집합 기본값, 작업 집합 범위, 작업 집합 크기, 작업 일정 우선 순위, CPU 시간 제한과 같은 실행 특성을 변경합니다.
- 작업에 전달할 매개 변수를 변경합니다.
`/PARAMETERS` 한정자를 사용하여 매개 변수를 변경합니다.

SUBMIT 명령 한정자

다음은 일괄 처리 작업 특성을 제어하기 위해 **SUBMIT** 명령으로 지정할 수 있는 한정자입니다. 작업 집합 기본값, 작업 집합 범위, 작업 집합 크기, 작업 일정 우선 순위, CPU 시간 제한과 같은 실행 특성도 지정할 수 있습니다.

- **/AFTER**
일괄 처리 작업이 실행될 때까지 대기하는 시간을 지정합니다. 지정된 시간까지는 작업이 일괄 처리 대기열에 남아 있게 됩니다. 명시적으로 해제할 때까지 대기열의 작업을 중지하려면 `/HOLD` 한정자를 사용합니다. 중지되고 있는 작업을 해제하려면 **SET ENTRY/RELEASE** 명령을 사용합니다.
- **/NAME**
일괄 처리 작업의 이름을 지정합니다. 이 이름을 지정하지 않으면 작업의 첫 번째 또는 유일한 명령 프로시저의 파일 이름이 작업 이름으로 사용됩니다.

- **/NOTE**

SHOW QUEUE/FULL 명령에 대한 표시의 일부로 나타날 메시지 문자열을 지정합니다. 작업에 대한 정보를 운영자나 시스템 관리자에게 전달할 수 있습니다.

- **/NOTIFY**

작업 완료에 대한 알림을 요청합니다. 일괄 처리 작업 실행이 완료되면 시스템이 사용자 터미널에 메시지를 보냅니다.

- **/PARAMETERS**

일괄 처리 작업에 매개 변수를 전달합니다.

- **/NOPRINTER** 또는 **/KEEP**

일괄 처리 작업 로그 파일을 저장합니다.

- **/QUEUE**

일괄 처리 작업을 **SYS\$BATCH**가 아닌 다른 대기열로 보냅니다. 원격 노드에 있는 명령 프로시저를 실행하려면 **/REMOTE** 한정자를 사용합니다. 이 한정자는 작업을 원격 노드의 **SYS\$BATCH**로 보냅니다.

- **/RESTART**

작업이 실행되는 동안 시스템에 오류가 발생하는 경우 작업을 다시 시작할 수 있게 합니다.

- **/RETAIN**

일괄 처리 작업이 완료된 후에도 그 작업을 대기열에 계속 둡니다. **SHOW QUEUE** 또는 **SHOW ENTRY** 명령을 사용하여 작업의 완료 상태를 볼 수 있습니다.

일괄 처리 대기열의 작업 표시

작업이 일괄 처리 작업 대기열에 들어오면 **SHOW ENTRY** 명령 또는 **SHOW QUEUE** 명령을 사용하여 작업의 상태를 모니터링할 수 있습니다. 대기열에 작업이 없으면 시스템에서 다음 메시지를 표시합니다.

```
$ SHOW QUEUE BOSTON_BATCH
Batch queue BOSTON_BATCH, on BOSTON::
```

작업에 대한 전체 정보를 보려면 **SHOW ENTRY** 또는 **SHOW QUEUE** 명령과 함께 **/FULL** 한정자를 사용합니다. 대기열의 다른 작업 상태를 보려면 **SHOW QUEUE/ALL** 명령을 사용합니다.

다음 보기에서는 항목 번호 999가 표시됩니다.

```
$ SUBMIT EXCHAN.DAT
Job EXCHAN (queue SYS$BATCH entry 999) started on SYS$BATCH
$ SHOW ENTRY 999
```

Entry	Jobname	Username	Blocks	Status
999	EXCHAN	BLASS	3	Executing

On batch queue SYS\$BATCH

```
$ SUBMIT/NOPRINTER/PARAMETER=STATS.DAT UPDATE
Job UPDATE (queue SYS$BATCH entry 1080) started on BOSTON_BATCH
$ SHOW QUEUE BOSTON_BATCH
Batch queue BOSTON_BATCH on BOSTON::
```

Entry	Jobname	Username	Blocks	Status
1080	UPDATE	ODONNELL	36	Executing

다음 보기에서 **/FULL** 한정자는 작업 번호 999에 관련된 특성과 **BOSTON_BATCH**에 대한 통계를 표시합니다.

```
$ SHOW ENTRY/FULL 999
```

Entry	Jobname	Username	Blocks	Status
999	EXCHAN	BLASS	3	Executing

On batch queue BOSTON_BATCH
Submitted 11-DEC-1999 13:12 /PRIORITY=100
WRKD: [BLASS]EXCHAN.DAT;3

```
$ SHOW QUEUE/FULL BOSTON_BATCH
Batch queue BOSTON_BATCH, on BOSTON::
/BASE_PRIORITY=3 /JOB_LIMIT=5 /OWNER=[EXEC] /PROTECTION=(S:E,O:D,G:R,W:W)
```

Entry	Jobname	Username	Blocks	Status
1080	UPDATE	ODONNELL	36	Executing

Submitted 11-DEC-1999 10:46 /KEEP /PARAM=('STATS.DAT') /NOPRINTER /PRIO=4
_BOSTON\$DQA2: [ODONNELL]TEMP.COM;1 (executing)

다음 보기에서는 **SHOW QUEUE/ALL** 명령을 사용하여 **BOSTON_BATCH** 대기열의 모든 작업을 표시합니다.

```
$ SHOW QUEUE/ALL BOSTON_BATCH
Batch queue BOSTON_BATCH on BOSTON::
```

Entry	Jobname	Username	Status
923	no privilege		Executing
939	no privilege		Holding until 11-DEC-1999 19:00
1080	UPDATE	ODONNELL	Executing

특권 있는 사용자가 아닌 경우 이 정보는 자신의 계정으로 제출한 작업에 대해서 제한됩니다.

일괄 처리 작업 삭제 및 중지

실행 전이나 실행 중에 일괄 처리 작업을 삭제할 수 있습니다. 일괄 처리 대기열에서 보류 중이거나 이미 실행 중인 항목을 삭제하려면 **DELETE/ENTRY** 명령을 사용합니다. 사용자가 직접 제출하지 않은 작업을 삭제하기 위해서는 특별한 특권이 필요합니다.

DELETE/ENTRY 명령으로 인해 작업이 종료되면 로그 파일이 인쇄되지도 않고 디렉토리에서 삭제되지도 않습니다.

DELETE/ENTRY 명령을 사용하여 작업을 종료하는 경우 운영 체제의 정상적인 작업 종료 활동이 우선하기 때문에 비정상적인 종료로 처리됩니다. 따라서 일괄 처리 작업 로그에는 작업 시간과 계정 정보가 요약된 표준 로그아웃 메시지가 포함되지 않습니다. 그러나 명시적인 **EXIT** 명령이나 **STOP** 명령 또는 이 두 명령 중 하나에 대한 암시적인 실행으로 인한 종료(현재의 **ON** 조건의 결과)는 정상적인 종료로 간주됩니다. 운영 체제는 정상적인 종료 후 적절한 축소 및 계정 프로시저를 수행합니다.

다음 명령은 **SYS\$BATCH**에서 작업 항목 210을 삭제합니다.

```
$ DELETE/ENTRY=210 SYS$BATCH
```

일괄 처리 작업 다시 시작

일괄 처리 작업이 실행되는 동안 시스템에 오류가 발생하면 작업이 완료되지 않습니다. 시스템이 복구되고 대기열이 다시 시작되면 해당 작업은 중단되고 대기열의 다음 작업이 실행됩니다. 그러나 일괄 처리 작업을 제출할 때 **/RESTART** 한정자를 지정하면 작업이 완료되기 전에 시스템에 오류가 발생하는 경우 시스템에서 이 작업이 다시 실행되도록 할 수 있습니다.

기본적으로 일괄 처리 작업은 첫 행부터 다시 실행됩니다. 다시 시작할 지점을 다르게 지정하기 위해 명령 프로시저에 추가할 수 있는 기호에 대한 자세한 내용은 13장과 14장을 참조하십시오.

시스템 오류로 인해 작업을 다시 시작하는 것 외에도 사용자가 작업을 명시적으로 중지한 후에도 작업을 다시 시작할 수 있습니다. 작업을 중지했다가 동일한 대기열이나 다른 대기열에서 이 작업을 다시 시작하려면 **STOP/QUEUE/REQUEUE/ENTRY** 명령을 사용합니다.

다음 보기의 명령은 **SYS\$BATCH**의 212 작업을 중지했다가 **SYS\$BATCH**로 다시 보냅니다.

```
$ STOP/QUEUE/REQUEUE/ENTRY=212 SYS$BATCH
```

이 명령을 입력하려면 **SUBMIT** 명령에 **/RESTART** 한정자를 사용하는 212 작업이 제출되어 있어야 합니다. 일괄 처리 작업이 두 번째로 실행될 때는 시스템에서 전역 기호 **BATCH\$RESTART**를 사용하여 작업 실행을 시작할 위치를 확인합니다.

일괄 처리 작업 실행 동기화

명령 프로시저 내에서 **SYNCHRONIZE**와 **WAIT** 명령을 사용하여 프로시저를 대기 상태로 둘 수 있습니다. **SYNCHRONIZE** 명령은 지정한 작업이 완료될 때까지 프로시저가 기다리도록 합니다. **WAIT** 명령은 지정한 시간이 경과될 때까지 프로시저가 기다리도록 합니다.

SYNCHRONIZE 명령을 사용하여 작업 이름을 지정할 경우, 동기화할 작업은 해당 사용자 이름과 관련되어야 합니다. 작업은 그 작업을 제출하는 프로세스의 사용자 이름과 관련됩니다. 여러 사용자에 대한 작업을 동기화하려면 **SYNCHRONIZE** 명령과 함께 **/ENTRY** 한정자를 사용하여 작업 항목 번호를 지정해야 합니다.

다음 보기에서는 두 개의 작업이 협동적인 기능을 수행하기 위해 동시에 제출될 경우 한 작업에 다음과 같은 명령이 포함될 수 있습니다.

```
$ SYNCHRONIZE BATCH25
```

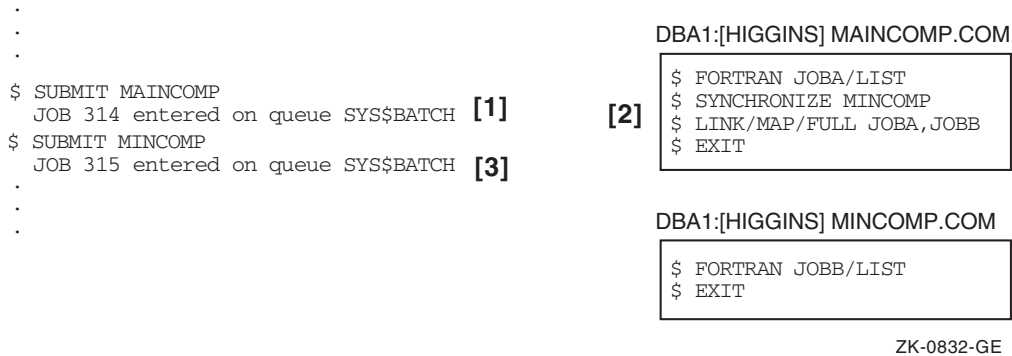
이 명령이 실행된 후 명령 프로시저는 작업 이름 **BATCH25**로 식별된 작업의 실행이 완료될 때까지는 실행을 계속할 수 없습니다.

이 **SYNCHRONIZE** 명령은 454 작업이 완료될 때까지 현재 명령 프로시저를 대기 상태로 둡니다.

```
$ SYNCHRONIZE/ENTRY=454
```

683페이지의 “일괄 처리 작업 실행 동기화”는 동시 실행을 위해 제출되었지만, 제대로 실행되기 위해 동기화되어야 하는 명령 프로시저 보기입니다. 각 프로시저는 큰 소스 프로그램을 컴파일합니다.

그림 16-1 일괄 처리 작업 실행 동기화



보기를 점검할 때는 다음 사항에 주의하십시오.

- [1] 개별 SUBMIT 명령이 두 개의 별도 작업을 제출하는 데 필요합니다. 첫 번째 프로세스가 만들어집니다.
- [2] FORTRAN 명령이 실행된 후 SYNCHRONIZE 명령이 실행됩니다. 315 작업이 수행 중이거나 보류 중인 경우, 314 작업은 다음 명령을 실행하지 않습니다.
- [3] 315 작업의 실행이 완료되면 다음 명령과 함께 314 작업이 계속됩니다.

WAIT 명령 사용

WAIT 명령은 디스크나 테이프 드라이브 같은 공유 시스템 리소스에 대한 액세스 권한이 있어야 하는 명령 프로시저에 유용합니다.

다음 보기에서는 테이프 드라이브의 할당을 요청하는 프로시저를 보여 줍니다.

```

$ TRY:
$   ALLOCATE DM: RK:
$   IF $STATUS THEN GOTO OKAY
$   WAIT 00:05
$   GOTO TRY
$ OKAY:
$ REQUEST/REPLY/TO=DISKS -
  "Please mount BACK_UP_GMB on 'F$TRNLNM("RK")'"
.
.
.

```

WAIT 명령이 성공적으로 완료되지 않으면 프로시저 자체가 대기 상태로 됩니다. 5분 후 요청을 다시 시도합니다.

ALLOCATE 요청 다음에 오는 **IF** 명령은 **\$STATUS**의 값을 확인합니다. **\$STATUS**의 값이 성공적인 완료를 나타낼 경우 명령 프로시저는 계속됩니다. 그렇지 않은 경우에는 프로시저가 **WAIT** 명령을 실행합니다. **WAIT** 명령은 시간 간격을 5분으로 지정합니다. 5분을 기다린 후 다음 명령인 **GOTO**가 실행되고 요청이 반복됩니다. 이 프로시저는 성공할 때까지 또는 일괄 처리 작업이 삭제되거나 중지될 때까지 장치 할당을 위한 반복과 시도를 계속합니다.

A 문자 집합

DEC MCS(다국적 문자 집합)는 00~FF의 16진수 값으로 식별되는 문자의 정의로 구성되며 DEC(Digital Equipment Corporation)에서 만들고 사용했습니다. DEC MCS는 ASCII 7 비트 문자 집합(00~7F의 16진수 값으로 식별)과 80~FF의 16진수 값으로 식별되는 8비트 문자열 집합의 두 부분으로 구성됩니다. DIGITAL이 만들고 판매한 소프트웨어를 사용해 본 사용자에겐 잘 알려져 있습니다.

UCS-2(Unicode Standard Character Set)는 0000~FFFF의 16진수 값으로 식별될 수 있는 16비트 문자의 집합에 대한 정의로 Unicode Consortium에서 만들었습니다.

ISO Latin-1 문자 집합은 00~FF의 16진수 값으로 식별되는 8비트 문자에 대한 UCS-2 정의입니다. ISO Latin-1 문자 집합 정의는 80~FF의 16진수 값에 대한 DEC MCS 정의와 약간 다릅니다.

표 A-1에는 DEC MCS가 포함되어 있습니다. 표 A-1에서는 두 문자 집합 간에 차이가 있는 문자를 표시하고 그림 A-1에서는 다른 문자를 보여 줍니다.

표 A-2에서는 DCL 문자 집합의 문자를 나열합니다.

Unicode(UCS-2) 문자 집합에 대한 자세한 내용은 Unicode Consortium에서 발간한 *The Unicode Standard*를 참조하십시오.

표 A-1

DEC 다국적 문자 집합

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
ASCII 제어 문자 ^a		
00	NUL	null 문자
01	SOH	제목의 시작(Ctrl/A)
02	STX	텍스트의 시작(Ctrl/B)
03	ETX	텍스트의 끝(Ctrl/C)
04	EOT	전송의 끝(Ctrl/D)
05	ENQ	조사(Ctrl/E)
06	ACK	수신 확인(Ctrl/F)
07	BEL	벨(Ctrl/G)
08	BS	백스페이스(Ctrl/H)
09	HT	수평 탭(Ctrl/I)
0A	LF	줄 바꿈(Ctrl/J)

표 A-1

DEC 다국적 문자 집합(계속)

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
0B	VT	수직 탭(Ctrl/K)
0C	FF	용지 공급(Ctrl/L)
0D	CR	캐리지 리턴(Ctrl/M)
0E	SO	shift out(Ctrl/N)
0F	SI	shift in(Ctrl/O)
10	DLE	데이터 링크 이스케이프(Ctrl/P)
11	DC1	장치 제어 1(Ctrl/Q)
12	DC2	장치 제어 2(Ctrl/R)
13	DC3	장치 제어 3(Ctrl/S)
14	DC4	장치 제어 4(Ctrl/T)
15	NAK	수신 거부(Ctrl/U)
16	SYN	동기화 대기(Ctrl/V)
17	ETB	전송 블록의 끝(Ctrl/W)
18	CAN	취소(Ctrl/X)
19	EM	미디어의 끝(Ctrl/Y)
1A	SUB	대체(Ctrl/Z)
1B	ESC	이스케이프
1C	FS	파일 구분
1D	GS	그룹 구분
1E	RS	레코드 구분
1F	US	단위 구분
ASCII 특수 및 숫자형 문자		
20	SP	공백
21	!	감탄 부호
22	"	인용 부호(큰 따옴표)
23	#	숫자 기호

표 A-1

DEC 다국적 문자 집합(계속)

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
24	\$	달러 기호
25	%	백분율 기호
26	&	앰퍼샌드
27	'	어포스트로피(작은 따옴표)
28	(	여는 괄호
29	)	닫는 괄호
2A	*	별표
2B	+	더하기
2C	,	쉼표
2D	-	하이픈 또는 빼기
2E	.	마침표 또는 소수점
2F	/	슬래시
30	0	0
31	1	1
32	2	2
33	3	3
34	4	4
35	5	5
36	6	6
37	7	7
38	8	8
39	9	9
3A	:	콜론
3B	;	세미콜론
3C	<	보다 작음
3D	=	같음
3E	>	보다 큼

표 A-1

DEC 다국적 문자 집합(계속)

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
3F	?	물음표
ASCII 영문자		
40	@	상업용 애틀 기호
41	A	대문자 A
42	B	대문자 B
43	C	대문자 C
44	D	대문자 D
45	E	대문자 E
46	F	대문자 F
47	G	대문자 G
48	H	대문자 H
49	I	대문자 I
4A	J	대문자 J
4B	K	대문자 K
4C	L	대문자 L
4D	M	대문자 M
4E	N	대문자 N
4F	O	대문자 O
50	P	대문자 P
51	Q	대문자 Q
52	R	대문자 R
53	S	대문자 S
54	T	대문자 T
55	U	대문자 U
56	V	대문자 V

표 A-1

DEC 다국적 문자 집합(계속)

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
57	W	대문자 W
58	X	대문자 X
59	Y	대문자 Y
5A	Z	대문자 Z
5B	[	왼쪽 대괄호
5C	\	역슬래시
5D	]	오른쪽 대괄호
5E	^	악센트 기호
5F	_	밑줄
60	`	억음 악센트 기호
61	a	소문자 a
62	b	소문자 b
63	c	소문자 c
64	d	소문자 d
65	e	소문자 e
66	f	소문자 f
67	g	소문자 g
68	h	소문자 h
69	i	소문자 i
6A	j	소문자 j
6B	k	소문자 k
6C	l	소문자 l
6D	m	소문자 m
6E	n	소문자 n
6F	o	소문자 o
70	p	소문자 p

표 A-1

DEC 다국적 문자 집합(계속)

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
71	q	소문자 q
72	r	소문자 r
73	s	소문자 s
74	t	소문자 t
75	u	소문자 u
76	v	소문자 v
77	w	소문자 w
78	x	소문자 x
79	y	소문자 y
7A	z	소문자 z
7B	{	왼쪽 중괄호
7C		수직 행
7D	}	오른쪽 중괄호(ALTMODE)
7E	~	물결표(ALTMODE)
7F	DEL	삭제(DELETE)
제어 문자		
80		[예약어]
81		[예약어]
82		[예약어]
83		[예약어]
84	IND	인덱스
85	NEL	다음 행
86	SSA	선택 영역의 시작 부분
87	ESA	선택 영역의 끝 부분
88	HTS	가로 맞춤 탭 설정

표 A-1

DEC 다국적 문자 집합(계속)

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
89	HTJ	양쪽 맞춤으로 가로 맞춤 탭 설정
8A	VTs	세로 맞춤 탭 설정
8B	PLD	일부 행 아래로
8C	PLU	일부 행 위로
8D	RI	역방향 인텍스
8E	SS2	단일 이동 2
8F	SS3	단일 이동 3
90	DCS	장치 제어 문자열
91	PU1	개인 사용 1
92	PU2	개인 사용 2
93	STS	전송 상태 설정
94	CCH	취소 문자
95	MW	메시지 대기
96	SPA	보호 영역의 시작 부분
97	EPA	보호 영역의 끝 부분
98		[예약어]
99		[예약어]
9A		[예약어]
9B	CSI	Control Sequence Introducer
9C	ST	문자열 종결자
9D	OSC	운영 체제 명령
9E	PM	개인 메시지
9F	APC	어플리케이션
기타 문자열		
A0		[예약어] ^b

표 A-1

DEC 다국적 문자 집합(계속)

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
A1	i	뒤집어진 감탄 부호
A2	¢	센트 기호
A3	£	파운드 기호
A4		[예약어] ^b
A5	¥	엔 기호
A6		[예약어] ^b
A7	§	섹션 기호
A8	¤	일반 통화 기호 ^b
A9	©	저작권 기호
AA	^a	여성형 서수 표시자
AB	«	왼쪽 꺾쇠 인용 부호 표시
AC		[예약어] ^b
AD		[예약어] ^b
AE		[예약어] ^b
AF		[예약어] ^b
B0	°	각도 기호
B1	±	+/- 기호
B2	²	위 첨자 2
B3	³	위 첨자 3
B4		[예약어] ^b
B5	μ	마이크로 기호
B6	¶	단락 기호
B7	·	중간 점 기호
B8		[예약어] ^b
B9	¹	위 첨자 1

표 A-1

DEC 다국적 문자 집합(계속)

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
BA	°	남성형 서수 표시자
BB	»	오른쪽 꺾쇠 인용 부호 표시
BC	¼	분수 4분의 1
BD	½	분수 2분의 1
BE		[예약어] ^b
BF	ı	뒤집어진 물음표
C0	À	억음 악센트가 있는 대문자 A
C1	Á	예음 악센트가 있는 대문자 A
C2	Â	악센트가 있는 대문자 A
C3	Ã	물결표가 있는 대문자 A
C4	Ä	움라우트가 있는 대문자 A(분음)
C5	Å	원 모양이 있는 대문자 A
C6	Æ	이중 모음 대문자 AE
C7	Ç	갈고리형 부호가 있는 대문자 C
C8	È	억음 악센트가 있는 대문자 E
C9	É	예음 악센트가 있는 대문자 E
CA	Ê	악센트가 있는 대문자 E
CB	Ë	움라우트가 있는 대문자 E(분음)
CC	Ì	억음 악센트가 있는 대문자 I
CD	Í	예음 악센트가 있는 대문자 I
CE	Î	악센트가 있는 대문자 I
CF	Ï	움라우트가 있는 대문자 I(분음)
D0		[예약어] ^b
D1	Ñ	틸데가 있는 대문자 N
D2	Ò	억음 악센트가 있는 대문자 O

표 A-1

DEC 다국적 문자 집합(계속)

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
D3	Ó	예음 악센트가 있는 대문자 O
D4	Ô	악센트가 있는 대문자 O
D5	Õ	틸데가 있는 대문자 O
D6	Ö	움라우트가 있는 대문자 O(분음)
D7	Œ	대문자 OE 연결선 2
D8	Ø	슬래시가 있는 대문자 O
D9	Ù	억음 악센트가 있는 대문자 U
DA	Ú	예음 악센트가 있는 대문자 U
DB	Û	악센트가 있는 대문자 U
DC	Ü	움라우트가 있는 대문자 U(분음)
DD	Ÿ	움라우트가 있는 대문자 Y(분음)
DE		[예약어] ^b
DF	ß	독일어 소문자 샵 s
E0	à	억음 악센트가 있는 소문자 a
E1	á	예음 악센트가 있는 소문자 a
E2	â	악센트가 있는 소문자 a
E3	ã	틸데가 있는 소문자 a
E4	ä	움라우트가 있는 소문자 a(분음)
E5	å	원 모양이 있는 소문자 a
E6	æ	이중 모음 소문자 ae
E7	ç	갈고리형 부호가 있는 소문자 c
E8	è	억음 악센트가 있는 소문자 e
E9	é	예음 악센트가 있는 소문자 e
EA	ê	악센트가 있는 소문자 e
EB	ë	움라우트가 있는 소문자 e(분음)

표 A-1

DEC 다국적 문자 집합(계속)

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름
EC	ì	억음 악센트가 있는 소문자 i
ED	í	예음 악센트가 있는 소문자 i
EE	î	악센트가 있는 소문자 i
EF	ï	움라우트가 있는 소문자 i(분음)
F0		[예약어] ^b
F1	ñ	틸데가 있는 소문자 n
F2	ò	억음 악센트가 있는 소문자 o
F3	ó	예음이 있는 소문자 o
F4	ô	악센트가 있는 소문자 o
F5	õ	틸데가 있는 소문자 o
F6	ö	움라우트가 있는 소문자 o(분음)
F7	œ	소문자 œ 연결선 2
F8	ø	슬래시가 있는 소문자 o
F9	ù	억음 액센트가 있는 소문자 u
FA	ú	예음 액센트가 있는 소문자 u
FB	û	악센트가 있는 소문자 u
FC	ü	움라우트가 있는 소문자 u(분음)
FD	ÿ	움라우트가 있는 소문자 y(분음) 2
FE		[예약어] ^b
FF		[예약어] ^b

a. ALTMODE 및 DELETE 문자(십진 125, 126 및 127)도 제어 문자입니다.

b. ISO Latin-1의 다른 문자입니다. 그림 A-1을 참조하십시오.

그림 A-1

16진 코드	MCS 문자 또는 약어	DEC 다국적 문자 이름	Isolatin-1 문자	Isolatin-1 문자 이름
A0		[예약어]		단어 잘림 방지 공백
A4		[예약어]	α	통화 기호
A6		[예약어]		끊어진 수직 막대
A8	α	통화 기호	..	분음 문자 배치
AC		[예약어]	¬	기호가 아님
AD		[예약어]	—	소프트 하이픈
AE		[예약어]	®	등록 상표 기호
AF		[예약어]	—	장음 문자 배치
B4		[예약어]	´	예음 문자 배치
B8		[예약어]	˘	갈고리형 문자 배치
BE		[예약어]	¾	분수 기호
D0		[예약어]	Ð	라틴어 대문자(eth)
D7	Œ	대문자 OE 연결선	×	곱하기 기호
DE		[예약어]	Þ	라틴어 대문자(thorn)
F0		[예약어]	Ø	라틴어 소문자(eth)
F7	œ	소문자 oe 연결선	÷	나누기 기호
FD	ÿ	움라우트가 있는 소문자 y (분음)	ý	라틴어 소문자 y 예음
FE		[예약어]	þ	라틴어 소문자(thorn)
FF		[예약어]	ÿ	라틴어 소문자 y 분음

VM-0128A-AI

표 A-2

DCL 문자 집합

기호	이름	의미
@	콜뱅이 기호	명령 프로시저 파일의 내용을 명령 입력 스트림에 넣습니다.
:	콜론	파일 지정 방법의 장치 이름 구분 기호입니다. 이중 콜론(:)은 노드 이름 구분 기호입니다. 콜론은 한정자 구분 기호로도 사용됩니다. 한정자 이름과 그 값을 구분합니다.
/	슬래시	한정자 접두사입니다.
+	더하기 기호	매개 변수 구분 기호입니다. 일부 명령에서는 매개 변수 연결 기호로 사용됩니다. 더하기 기호는 숫자 식에서 문자열 연결 연산자, 단항 가산 기호 및 더하기 연산자로도 인식됩니다.
,	쉼표	매개 변수나 인수 목록의 나열 요소 구분 기호입니다.
-	하이픈	연결 문자입니다. 하이픈은 숫자 식에서 문자열 제거 연산자, 단항 감산 기호, 빼기 연산자로도 인식되며 디렉토리 검색 와일드카드 문자로도 사용됩니다.
()	괄호	인수 목록의 나열 구분 기호입니다. 또한, 숫자 식에서 연산 순서를 지정하는 데 사용됩니다.
[]	대괄호	파일 지정 방법의 디렉토리 이름 구분 기호입니다. 꺾쇠 괄호와 동일합니다.
<>	꺾쇠 괄호	파일 지정 방법의 디렉토리 구분 기호입니다. 대괄호와 같습니다.
?	물음표	도움말 문자입니다.
&	앰퍼샌드	실행 시간 대체 연산자 또는 예약된 특수 문자입니다.
\	역슬래시	예약된 특수 문자입니다.

표 A-2

DCL 문자 집합(계속)

기호	이름	의미
=	등호	한정자 값 구분 기호입니다. 한정자 이름과 그 인수를 구분합니다. 등호(=)는 기호를 정의할 때 할당문으로 사용될 수도 있습니다.
^	악센트 기호	예약된 특수 문자입니다.
#	번호 기호	예약된 특수 문자입니다.
*	별표	파일 지정 방법의 와일드카드 문자입니다. 별표는 숫자 식에서 곱하기 연산자로도 사용되고 기호 정의에서 약어 구분 기호로도 사용됩니다.
'	어포스트로피	대체 연산자입니다.
.	마침표	파일 지정 방법의 파일 유형 및 버전 번호를 구분하는 기호입니다. 또한 하위 디렉토리 구분 기호로도 사용됩니다.
;	세미콜론	파일 지정 방법의 버전 번호를 구분하는 기호입니다.
%	백분율 기호	파일 지정 방법의 와일드카드 문자입니다. 기수 연산자로도 사용됩니다.
!	감탄 부호	주석을 나타냅니다.
"	인용 부호	리터럴 문자 구분 기호입니다.

B 주식 명령 프로시저

이 부록에는 13장, 14장 및 15장에 설명된 개념과 기술에 대한 전체 명령 프로시저가 포함되어 있습니다. 이 부록의 각 절에서는 명령 프로시저를 설명하고 다음 정보를 제공합니다.

- 프로시저 이름
- 프로시저 목록
- 프로시저에 사용되는 개념 또는 기술을 설명하는 참조 내용
- 프로시저의 보기 실행 결과

CONVERT.COM 명령 프로시저

이 명령 프로시저는 절대 시간(미래 시간)을 변경 시간으로 변환하고 현재 시간과 사용자가 지정한 시간 간의 시간 차이를 결정합니다. 이 프로시저에서는 **F\$TIME** 및 **F\$CVTIME** 렉시칼 함수와 할당문을 사용하여 산술 계산을 수행하고 기호 값을 연결하는 방법을 보여 줍니다.

보기: CONVERT.COM

```
$ ! Procedure to convert an absolute time to a delta time.
$ ! The delta time is returned as the global symbol WAIT_TIME.
$ ! P1 is the time to be converted.
$ ! P2 is an optional parameter - SHOW - that causes the
$ ! procedure to display WAIT_TIME before exiting
$ !
$ ! Check for inquiry
$ !
$ IF P1 .EQS. "?" .OR. P1 .EQS. "" THEN GOTO TELL [1]
$ !
$ ! Verify the parameter:  hours must be less than 24
$ !                        minutes must be less than 60
$ !                        time string must contain only hours
$ !                        and minutes
$ !
$ ! Change error and message handling to
$ ! use message at BADTIME
$ !
$ ON WARNING THEN GOTO BADTIME [2]
$ SAVE_MESSAGE = F$ENVIRONMENT("MESSAGE")
$ SET MESSAGE/NOFACILITY/NOIDENTIFICATION/NOSEVERITY/NOTEXT
$ TEMP = F$CVTIME(P1)
$ !
$ ! Restore default error handling and message format
$ ON ERROR THEN EXIT
$ SET MESSAGE'SAVE_MESSAGE'
$ !
$ IF F$LENGTH(P1) .NE. 5 .OR. - [3]
    F$LOCATE(":",P1) .NE. 2 -
    THEN GOTO BADTIME
$ !
$ ! Get the current time
$ !
$ TIME = F$TIME() [4]
$ !
$ ! Extract the hour and minute fields from both the current time
$ ! value (TIME) and the future time (P1)
$ !
$ MINUTES = F$CVTIME(TIME,"ABSOLUTE","MINUTE")      ! Current minutes [5]
$ HOURS = F$CVTIME(TIME,"ABSOLUTE","HOUR")           ! Current hours
$ FUTURE_MINUTES = F$CVTIME(P1,"ABSOLUTE","MINUTE") ! Minutes in future time
$ FUTURE_HOURS = F$CVTIME(P1,"ABSOLUTE","HOUR")      ! Hours in future time
```

```

$ !
$ !
$ ! Convert both time values to minutes
$ ! Note the implicit string to integer conversion being performed
$ ! $ CURRENT_TIME = HOURS*60 + MINUTES [6]
$ FUTURE_TIME = FUTURE_HOURS*60 + FUTURE_MINUTES
$ !
$ ! Compute difference between the future time and the current time
$ ! (in minutes)
$ !
$ !
$ MINUTES_TO_WAIT = FUTURE_TIME - CURRENT_TIME [7]
$ !
$ ! If the result is less than 0 the specified time is assumed to be
$ ! for the next day; more calculation is required.
$ !
$ IF MINUTES_TO_WAIT .LT. 0 THEN - [8]
$     MINUTES_TO_WAIT = 24*60 + FUTURE_TIME - CURRENT_TIME
$ !
$ ! Start looping to determine the value in hours and minutes from
$ ! the value expressed all in minutes
$ !
$     HOURS_TO_WAIT = 0
$ HOURS_TO_WAIT_LOOP: [9]
$     IF MINUTES_TO_WAIT .LT. 60 THEN GOTO FINISH_COMPUTE
$     MINUTES_TO_WAIT = MINUTES_TO_WAIT - 60
$     HOURS_TO_WAIT = HOURS_TO_WAIT + 1
$     GOTO HOURS_TO_WAIT_LOOP
$ FINISH_COMPUTE:
$ !
$ ! Construct the delta time string in the proper format
$ !
$ WAIT_TIME == F$STRING(HOURS_TO_WAIT)+ ":" + F$STRING(MINUTES_TO_WAIT)- [10]
$ + ":00.00"
$ !
$ ! Examine the second parameter
$ !
$ IF P2 .EQS. "SHOW" THEN SHOW SYMBOL WAIT_TIME [11]
$ !
$ ! Normal exit
$ !
$ EXIT
$ !
$ BADTIME: [12]
$ ! Exit taken if first parameter is not formatted correctly
$ ! EXIT command returns but does not display error status
$ !
$ SET MESSAGE'SAVE_MESSAGE'
$ WRITE SYS$OUTPUT "Invalid time value: ",P1," , format must be hh:mm"
$ WRITE SYS$OUTPUT "Hours must be less than 24; minutes must be less than
60"
$ EXIT %X10000000
$ !
$ !
$ TELL: [13]
$ ! Display message and exit if user enters inquiry or enters

```

```
$ ! an illegal parameter
$ !
$ TYPE SYS$INPUT
  This procedure converts an absolute time
  value to a delta time value. The absolute time must be in
  the form hh:mm and must indicate a time in the future.
  On return, the global symbol WAIT_TIME contains the
  converted time value. If you enter the keyword SHOW
  as the second parameter, the procedure displays the
  resulting value in the output stream. To invoke this
  procedure, use the following syntax:

                                @CONVERT hh:mm [SHOW]

$ EXIT
```

CONVERT.COM 명령 프로시저 설명

- [1] 이 프로시저에서는 매개 변수가 생략되었는지 또는 매개 변수에 입력된 값이 물음표(?) 문자인지를 검사합니다. 두 경우 모두 프로시저가 레이블 **TELL**에서 분기됩니다.
- [2] 이 프로시저는 **F\$CVTIME** 함수를 사용하여 시간 값이 유효한 24시간 단위 표기법으로 입력되었는지 확인합니다. **F\$CVTIME** 함수는 입력 시간이 유효하지 않은 경우 경고 메시지를 반환합니다. **F\$CVTIME** 함수가 오류를 반환할 경우 프로시저는 기본값 **ON** 동작을 레이블 **BADTIME**에 대한 직접 제어로 변경합니다.

프로시저는 **F\$ENVIRONMENT** 함수를 사용하여 현재 메시지 설정을 저장합니다. 그런 다음 경고 또는 오류 메시지가 표시되지 않도록 메시지 형식을 설정합니다. 시간 값을 검사한 다음 프로시저는 기본값 **On** 조건과 메시지 형식을 복원합니다.

- [3] 프로시저는 매개 변수의 형식을 검사합니다. 매개 변수는 다음과 같은 형식의 시간 값이어야 합니다.

hh:mm

IF 명령어는 (1) 입력된 값의 길이가 5자이고 (2) 세 번째 문자(2의 오프셋)가 콜론(:)인지 확인합니다. **IF** 명령어는 논리 **OR** 연산자를 포함합니다. 두 표현식 중 하나가 참(길이가 5가 아니거나 세 번째 문자 위치가 콜론이 아닐 경우)일 경우 프로시저가 레이블 **BADTIME**에서 분기됩니다.

- [4] **F\$TIME** 렉시칼 함수는 현재 시간 값을 기호 **TIME**에 입력합니다.
- [5] **F\$CVTIME** 함수는 기호 **TIME**에 저장된 현재 시간으로부터 “분” 및 “시” 필드를 추출합니다. 그런 다음 **F\$CVTIME** 함수는 변환할 시간으로부터 “분” 및 “시” 필드를 추출합니다.

- [6] 이 할당문은 현재 시간과 미래 시간을 분으로 변환합니다. 할당문에 기호 MINUTES, HOURS, FUTURE_HOURS 및 FUTURE_MINUTES를 사용하면 이 값이 정수로 자동 변환됩니다.
- [7] 그런 다음 프로시저는 미래 시간(분)에서 현재 시간(분)을 뺍니다.
- [8] 결과가 0보다 작으면 미래 시간이 그 다음 날에 해당되는 것으로 해석됩니다. 이 경우 프로시저는 미래 시간에 24시간을 더한 다음 현재 시간을 뺍니다.
- [9] 프로시저는 MINUTES_TO_WAIT 값으로부터 시간을 계산하는 루프를 입력합니다. 루프를 통과할 때마다 MINUTES_TO_WAIT가 60보다 큰지를 검사하여, 60보다 클 경우 MINUTES_TO_WAIT에서 60을 뺀 다음 시간 누적자(HOURS_TO_WAIT)에 1을 더합니다.
- [10] 루프가 종료되면 프로시저는 시간 값과 분 값을 시간 문자열로 연결합니다. 기호 HOURS_TO_WAIT 및 MINUTES_TO_WAIT가 해당 문자열로 바뀌고 간접 콜론으로 분리됩니다. 결과 문자열은 미래의 변경 시간 값을 보관하는 기호 WAIT_TIME에 할당됩니다. WAIT_TIME은 프로시저 CONVERT.COM이 종료될 때 삭제되지 않도록 글로벌 기호로 정의됩니다.
- [11] 두 번째 매개 변수인 SHOW를 입력하면 프로시저는 결과 시간 값을 표시합니다. 그렇지 않으면 프로시저가 종료됩니다.
- [12] 레이블 BADTIME에서 프로시저는 잘못 입력된 값과 필요한 형식을 보여 주는 오류 메시지를 표시합니다. 오류 메시지가 발행된 후 CONVERT.COM이 종료됩니다. EXIT 명령은 우선 순위 숫자가 1로 설정된 오류 상태를 반환합니다. 이 경우에는 오류 메시지가 표시되지 않습니다.

이 프로시저는 EXIT 명령을 사용하여 오류 상태를 명시적으로 지정하기 때문에 다른 프로시저에서 CONVERT.COM을 실행할 수 있습니다. CONVERT.COM이 완료되면 호출 프로시저에서 시간이 성공적으로 해석되었는지 확인할 수 있습니다.
- [13] 레이블 TELL에서 이 프로시저는 프로시저가 수행하는 작업에 대한 정보를 표시합니다. TYPE 명령은 입력 데이터 스트림인 SYS\$INPUT에 나열된 행을 표시합니다.

CONVERT.COM 명령 프로시저 보기 실행

```
$ SHOW TIME  
10-JUN-1999 10:38:26  
$$@CONVERT 12:00 SHOW  
WAIT_TIME = "1:22:00.00"
```

SHOW TIME 명령은 현재 날짜와 시간을 표시합니다. CONVERT.COM은 매개 변수 12:00 및 SHOW와 함께 실행됩니다. 이 프로시저에서는 절대 시간 12:00을 변경 시간 값으로 변환하여 터미널에 표시합니다.

REMINDER.COM 명령 프로시저

이 명령 프로시저는 지정된 시간에 터미널에 알람 메시지를 표시합니다. 이 프로시저는 메시지를 표시할 시간과 메시지의 텍스트를 묻는 프롬프트를 표시하고, **CONVERT.COM**을 사용하여 시간을 변경 시간으로 변환합니다. 그런 다음 지정된 시간이 되면 알람 메시지를 표시하는 하위 프로세스를 생성합니다. 이 프로시저에서는 **F\$ENVIRONMENT**, **F\$VERIFY** 및 **F\$GETDVI** 함수의 사용을 보여 줍니다.

보기: REMINDER.COM

```
$ ! Procedure to obtain a reminder message and display this
$ ! message on your terminal at the time you specify.

$ !
$ ! Save current states for procedure and image verification
$ ! Turn verification off for duration of procedure
$
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE") [1]
$ SAVE_VERIFY_PROC = F$VERIFY(0)
$ !
$ ! Places the current process in a wait state until a specified
$ ! absolute time. Then, it rings the bell on the terminal and
$ ! displays a message.
$ ! $ ! Prompt for absolute time
$ !
$
$ GET_TIME:
$ INQUIRE REMINDER_TIME "Enter time to send reminder (hh:mm)" [2]
$ INQUIRE MESSAGE_TEXT "Enter message"
$ !
$ ! Call the CONVERT.COM procedure to convert the absolute time
$ ! to a delta time
$ ! $ @DISK2:[JONES.TOOLS]CONVERT 'REMINDER_TIME' [3]
$ IF .NOT. $STATUS THEN GOTO BADTIME
$ !
$ !
$ ! Create a command file that will be executed
$ ! in a subprocess. The subprocess will wait until
$ ! the specified time and then display your message
$ ! at the terminal. If you are working at a DEC_CRT
$ ! terminal, the message has double size blinking
$ ! characters. Otherwise, the message has normal letters.
$ ! In either case, the terminal bell rings when the
$ ! message is displayed.
$
$ CREATE WAKEUP.COM [4]
$ DECK ! Lines starting with $ are data lines
$ WAIT 'WAIT_TIME' [5]
$ BELL[0,7] = %X07 ! Create symbol to ring the bell
$ IF F$GETDVI("SYS$OUTPUT", "TT_DECRT") .NES. "TRUE" THEN GOTO OTHER_TERM
$ !
```



```
$ DEC_CRT_ONLY:
$ ! Create symbols to set special graphics (for DEC_CRT terminals only)
$ !
$     SET_FLASH = "<ESC>[1;5m"           ! Turn on blinking characters
$     SET_NOFLASH = "<ESC>[0m"           ! Turn off blinking characters
$     TOP = "<ESC>#3"                     ! Double size characters (top
portion)
$     BOT = "<ESC>#4"                     ! Double size characters (bottom
portion)
$ !
$ ! Write double size, blinking message to the terminal and ring the bell
$ !
$     WRITE SYS$OUTPUT BELL, SET_FLASH, TOP, MESSAGE_TEXT
$     WRITE SYS$OUTPUT BELL, BOT, MESSAGE_TEXT
$     WRITE SYS$OUTPUT F$TIME(), SET_NOFLASH
$     GOTO CLEAN_UP
$ !
$ OTHER_TERM:
$     WRITE SYS$OUTPUT BELL,MESSAGE_TEXT
$     WRITE SYS$OUTPUT F$TIME()
$ !
$ CLEAN_UP:
$     DELETE WAKEUP.COM;*
$ EOD
$ !
$ ! Now continue executing commands.
$ !
$ SPAWN/NOWAIT/INPUT=WAKEUP.COM
$ END:
$ ! Restore verification
$     SAVE_VERIFY_PROC = F$VERIFY(SAVE_VERIFY_PROC, SAVE_VERIFY_IMAGE)
$     EXIT
$ !
$ BADTIME:
$     WRITE SYS$OUTPUT "Time must be entered as hh:mm"
$     GOTO GET_TIME
```

[6]
[7]

REMINDER.COM 명령 프로시저 설명

- [1] 이 프로시저에서는 **F\$ENVIRONMENT** 함수를 사용하여 기호 **SAVE_VERIFY_IMAGE**에 이미지 확인 설정을 저장합니다. 그런 다음 **F\$VERIFY** 함수를 사용하여 기호 **SAVE_VERIFY_PROC**에 프로시저 확인 설정을 저장합니다. 또한, **F\$VERIFY** 함수는 두 확인 유형을 모두 끕니다.
- [2] 이 프로시저는 **INQUIRE** 명령을 사용하여 메시지가 발송되는 시간을 묻는 프롬프트를 표시합니다. 이 값은 프로시저 **CONVERT.COM**에 대한 입력으로 사용됩니다. 또한, 메시지의 텍스트를 묻는 프롬프트를 표시합니다.
- [3] 이 프로시저는 중첩 프로시저인 **CONVERT.COM**을 실행합니다. 디스크와 디렉토리를 파일 지정 방법의 일부로 지정해야 합니다. 그러면 **REMINDER.COM**을 실행하는 위치에 관계 없이 시스템이 **CONVERT.COM**을 찾습니다.

CONVERT.COM은 알람을 변경 시간으로 변환하고 이 시간을 글로벌 기호 WAIT_TIME으로 반환합니다. 이 변경 시간은 현재 시간에서 메시지가 발송될 시간까지의 시간 간격을 나타냅니다. CONVERT.COM이 오류를 반환하면 프로시저가 레이블 BADTIME에서 분기됩니다.

- [4] 이 프로시저는 CREATE 명령을 사용하여 새 프로시저인 WAKEUP.COM을 작성합니다. 이 프로시저는 하위 프로시저 내에서 실행됩니다. CREATE 명령이 달러 기호로 시작되는 행을 읽을 수 있게 하려면 CREATE 명령에 대한 입력을 DECK 명령과 EOD 명령 사이에 넣습니다. 따라서 DECK 명령과 EOD 명령 사이에 있는 모든 행은 WAKEUP.COM에 기록됩니다.

- [5] WAKEUP.COM이 수행하는 작업은 다음과 같습니다.

- 기호 WAIT_TIME이 나타내는 시간이 될 때까지 기다립니다.
- 터미널 벨을 울리는 기호 BELL을 작성합니다.
- 터미널이 DEC_CRT 터미널이고 이스케이프 시퀀스를 적용하여 두 배 크기의 짝박이는 문자를 표시할 수 있는지 여부를 결정합니다. DEC_CRT 터미널인지 확인하려면 SHOW TERMINAL 명령을 입력하여 이 특성이 나열되는지 확인합니다.
- 터미널이 DEC_CRT 터미널일 경우 프로시저는 기호 SET_FLASH, TOP 및 BOT를 정의합니다. 이들 기호는 터미널이 반짝이는 두 배 크기 문자를 사용하게 합니다. 또한, 이 프로시저는 기호 SET_NOFLASH를 정의하여 터미널을 기본 상태로 복귀시킵니다. EDT 편집기를 사용하여 이 정의를 작성할 때 이스케이프 문자(<ESC>)를 입력하려면 ESC 키를 두 번 누릅니다.

이들 기호를 정의한 다음 프로시저는 터미널에 세 개의 행을 기록합니다. 첫 번째 행은 벨을 울리고, 반짝이는 문자를 켜고, 두 배 크기 문자를 사용하여 메시지의 상반부를 표시합니다. 두 번째 행은 벨을 다시 울리고 메시지의 하반부를 표시합니다. 세 번째 행은 현재 시간을 기록한 다음 반짝이는 문자를 해제하여 터미널을 기본 상태로 복귀시킵니다.

DEC_CRT 터미널이 없을 경우 프로시저는 터미널 벨을 울리고 메시지와 시간을 표시합니다.

- DELETE 명령은 프로시저 WAKEUP.COM이 실행된 다음 삭제합니다.
- [6] WAKEUP.COM이 작성되면 프로시저는 하위 프로세스를 생성하고 해당 하위 프로세스에 WAKEUP.COM을 입력 명령 파일로 사용하도록 지시합니다. /NOWAIT 한정자를 사용하면 하위 프로세스가 WAKEUP.COM으로부터 명령을 실행하는 동안에도 터미널에서 계속 작업할 수 있습니다. WAKEUP.COM은 터미널에 메시지를 표시합니다.

기본적으로 SPAWN 명령은 하위 프로세스에 글로벌 및 로컬 기호를 전달합니다. 따라서, REMINDER.COM에 기호 WAIT_TIME 및 MESSAGE_TEXT에 대한 값을 제공하더라도 WAKEUP.COM은 이들 기호에 액세스할 수 있습니다.

[7] 프로시저는 종료되기 전에 원본 확인 설정을 복원합니다.

REMINDER.COM 명령 프로시저 보기 실행

```
$ @REMINDER
Enter time to send reminder (hh:mm):12:00
Enter message:TIME FOR LUNCH
%DCL-S-SPAWNED, process BLUTO_1 spawned
$
.
.
.
TIME FOR LUNCH
11-DEC-1999 12:00:56.99
```

이 프로시저는 시간 값과 메시지를 묻는 프롬프트를 표시한 다음, 메시지를 표시하는 하위 프로세스를 생성합니다. 터미널에서 작업을 계속할 수 있습니다. 지정된 시간에 하위 프로세스는 터미널 벨을 울리고 메시지와 시간을 표시합니다.

DIR.COM 명령 프로시저

이 명령 프로시저는 DCL 명령 DIRECTORY/SIZE=ALL/DATE를 모방하여 지정된 파일의 (사용 및 할당된) 블록 크기와 작성 날짜를 표시합니다. 또한, F\$PARSE, F\$SEARCH, F\$FILE_ATTRIBUTES 및 F\$FAO 렉시칼 함수 사용을 보여 줍니다.

보기: DIR.COM

```
$ !
$ ! Command procedure implementation of DIRECTORY/SIZE=ALL/DATE
$ ! command
$ !
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ !
$ ! Replace any blank field of the P1 file specification with
$ ! a wildcard character
$ !
$ P1 = F$PARSE(P1,"*.*;") [1]
$ !
$ ! Define initial values for symbols
$ !
$ FIRST_TIME = "TRUE"
$ FILE_COUNT = 0
$ TOTAL_ALLOC = 0
$ TOTAL_USED = 0
$
$ LOOP: [2]
$     FILESPEC = F$SEARCH(P1)
$     ! Find next file in directory
$     IF FILESPEC .EQS. "" THEN GOTO DONE
$     ! If no more files, then done
$     IF .NOT. FIRST_TIME THEN GOTO SHOW_FILE
$     ! Print header only once
$     !
$     ! Construct and output the header line
$     !
$     FIRST_TIME = "FALSE" [3]
$     DIRSPEC = F$PARSE(FILESPEC,,, "DEVICE") -
+F$PARSE(FILESPEC,,, "DIRECTORY")
$     WRITE SYS$OUTPUT ""
$     WRITE SYS$OUTPUT "Directory ",DIRSPEC
$     WRITE SYS$OUTPUT ""
$     LASTDIR = DIRSPEC $
$     !
$     ! Put the file name together, get some of the file attributes, and
$     ! type the information out
$     !
$ SHOW_FILE:
$     FILE_COUNT = FILE_COUNT + 1
$     FILENAME = F$PARSE(FILESPEC,,, "NAME") - [4]
+ F$PARSE(FILESPEC,,, "TYPE") -
```

```
+ F$PARSE(FILESPEC,,, "VERSION")
$   ALLOC = F$FILE_ATTRIBUTES(FILESPEC, "ALQ")
$   USED = F$FILE_ATTRIBUTES(FILESPEC, "EOF")
$   TOTAL_ALLOC = TOTAL_ALLOC + ALLOC
$   TOTAL_USED = TOTAL_USED + USED
$   REVISED = F$FILE_ATTRIBUTES(FILESPEC, "RDT")
$   LINE = F$FAO("!19AS !5UL/!5!> !17AS",FILENAME,-
USED, ALLOC, REVISED)
$   WRITE SYS$OUTPUT LINE
$   GOTO LOOP
$
$ !
$ ! Output summary information, reset verification, and exit
$ !
$ DONE:
$   WRITE SYS$OUTPUT " " [5]
$   WRITE SYS$OUTPUT "Total of 'FILE_COUNT' files, " + -
"'TOTAL_USED'/'TOTAL_ALLOC' blocks."
$   SAVE_VERIFY_PROCEDURE =
F$VERIFY(SAVE_VERIFY_PROCEDURE,SAVE_VERIFY_IMAGE)
$   EXIT
```

DIR.COM 명령 프로시저 설명

- [1] 이 프로시저는 **F\$PARSE** 함수를 사용하여 사용자 제공 파일 지정 방법인 **P1**의 빈 필드에 별표를 넣습니다. **DIR.COM**을 실행할 때 매개 변수를 지정하지 않으면 **F\$PARSE** 함수가 **P1**에 값 **"*.*;"**를 할당합니다. 이 경우 **DIR.COM**을 실행하면 현재 기본 디렉토리의 모든 파일이 표시됩니다.
- [2] **F\$SEARCH** 렉시칼 함수는 디렉토리에서 **P1**이 나타내는 파일을 검색합니다. **P1**에 와일드카드(별표)가 포함되어 있을 경우 **F\$SEARCH** 함수는 일치하는 모든 파일 지정 방법을 반환합니다. 마지막 파일 지정 방법이 반환된 다음 프로시저는 레이블 **DONE**에서 분기됩니다.
- [3] 이 프로시저는 제일 처음 루프를 통과할 때 디렉토리 표시를 위한 머리글을 기록합니다. 이 머리글에는 장치 이름과 디렉토리 이름이 포함됩니다. 프로시저는 **F\$PARSE** 함수를 사용하여 이 이름을 얻습니다.
- [4] 이 프로시저는 **F\$PARSE** 렉시칼 함수를 사용하여 디렉토리의 각 파일 지정 방법으로 부터 파일 이름을 추출합니다. 그런 다음 **F\$FILE_ATTRIBUTES** 렉시칼 함수는 각 파일에 대한 사용된 블록, 할당된 블록 및 작성 날짜 정보를 얻습니다. 마지막으로 **F\$FAO** 함수는 디렉토리의 각 파일에 대한 단일 표시행을 서식화합니다. **F\$FAO** 함수는 파일 이름과 파일 속성 정보를 인수로 사용합니다.
- [5] **F\$SEARCH**가 null 문자열을 반환하면 프로시저가 레이블 **DONE**에서 분기되고 총 파일 수, 사용된 총 블록 수, 디렉토리에 할당된 총 블록 수 등을 표시합니다.

DIR.COM 명령 프로시저 보기 실행

```
$ @DIR [VERN]*.COM
```

```
Directory DISK4:[VERN]
```

BATCH.COM;1	1/3	11-DEC-1999 11:43
CALC.COM;3	1/3	11-DEC-1999 11:30
CONVERT.COM;1	5/6	11-DEC-1999 15:23
.		
.		
.		
LOGIN.COM;34	2/3	11-DEC-1999 13:17
PID.COM;7	1/3	11-DEC-1999 09:49
SCRATCH.COM;6	1/3	11-DEC-1999 11:29)

```
Total of 15 files, 22/48 blocks.
```

이 프로시저는 디렉토리 [VERN]에 있는 모든 .COM 파일에 대한 정보를 반환합니다.

SYS.COM 명령 프로시저

이 명령 프로시저는 현재 프로세스, 그룹 내의 모든 프로세스(현재 프로세스에 **group** 권한이 있을 경우) 및 시스템 내의 모든 프로세스(현재 프로세스에 **world** 권한이 있는 경우)에 대한 통계를 반환합니다. 이 프로시저에서는 **F\$PID**, **F\$EXTRACT** 및 **F\$GETJPI** 렉시칼 함수의 사용 방법을 보여 줍니다.

보기: SYS.COM

```
$ !
$ ! Displays information about owner, group, or system processes.
$ !
$ ! Turn off verification and save current settings
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ CONTEXT = "" ! Initialize PID search context [1]
$ !
$ ! Output header line.
$ !
$ WRITE SYS$OUTPUT " PID      Username      Term      Process " + - [2]
$ "name      State      Pri Image"
$ !
$ ! Output process information.
$ !
$ LOOP:
$ !
$ ! Get next PID. If null, then done.
$ !
$ PID = F$PID(CONTEXT) [3]
$ IF PID .EQS. "" THEN GOTO DONE
$ !
$ ! Get image file specification and extract the file name.
$ !
$ IMAGENAME = F$GETJPI(PID,"IMAGENAME") [4]
$ IMAGENAME = F$PARSE(IMAGENAME,,,"NAME","SYNTAX_ONLY")

$ !
$ ! Get terminal name. If none, then describe type of process.
$ !
$ TERMINAL = F$GETJPI(PID,"TERMINAL") [5]
$ IF TERMINAL .EQS. "" THEN -
$ TERMINAL = "-" + F$EXTRACT(0,3,F$GETJPI(PID,"MODE")) + "-"
$ IF TERMINAL .EQS. "-INT-" THEN TERMINAL = "-DET-"
$ IF F$GETJPI(PID,"OWNER") .NE. 0 THEN TERMINAL = "-SUB-"
$ !
$ ! Get more information, put process line together,
$ ! and output it.
$ !
$ LINE = F$FAO("!AS !12AS !7AS !15AS !5AS !2UL/!UL !10AS", -[6]
$ PID,F$GETJPI(PID,"USERNAME"),TERMINAL,-
$ F$GETJPI(PID,"PRCNAM"),-
```

```
                F$GETJPI (PID, "STATE"), F$GETJPI (PID, "PRI"), -  
                F$GETJPI (PID, "PRIB"), IMAGNAME)  
$      WRITE SYS$OUTPUT LINE  
$      GOTO LOOP  
  
$ !  
$ ! Restore verification and exit.  
$ ! $DONE:  
$   SAVE_VERIFY_PROCEDURE =  
F$VERIFY (SAVE_VERIFY_PROCEDURE, SAVE_VERIFY_IMAGE)  
$   EXIT
```

SYS.COM 명령 프로시저 설명

- [1] 기호 CONTEXT는 null 값으로 초기화됩니다. 이 기호는 F\$PID 함수와 함께 사용되어 프로세스 ID 번호 목록을 구합니다.
- [2] 이 프로시저는 표시할 머리글을 작성합니다.
- [3] 이 프로시저는 첫 번째 프로세스 ID(PID) 번호를 가져옵니다. 현재 프로세스에 group 또는 world 권한이 부족할 경우 현재 프로세스의 PID 번호가 반환됩니다. 현재 프로세스에 group 권한이 있을 경우 그룹 목록의 첫 번째 PID 번호가 반환됩니다. 현재 프로세스에 world 권한이 있을 경우 시스템 목록의 첫 번째 PID 번호가 반환됩니다. 이 함수는 마지막 PID 번호가 반환될 때까지 순차적으로 다음 PID 번호를 반환합니다. 이 지점에서 null 문자열이 반환되고 프로시저가 끝으로 분기됩니다.
- [4] 이 프로시저는 F\$GETJPI 렉시칼 함수를 사용하여 각 PID 번호에 대한 이미지 파일 지정 방법을 가져옵니다. F\$PARSE 함수는 F\$GETJPI 함수에 의해 반환되는 지정 방법으로부터 파일 이름을 추출합니다.
- [5] 이 프로시저는 F\$GETJPI 함수를 사용하여 각 PID 번호에 대한 터미널 이름을 가져옵니다. F\$EXTRACT 함수는 F\$GETJPI(PID, "MODE")에 의해 반환되는 MODE 지정 방법의 처음 세 문자를 추출하여 프로세스 유형을 결정합니다. F\$GETJPI 함수를 다시 사용하여 프로세스가 하위 프로세스인지 여부를 결정합니다.
- [6] 이 프로시저는 F\$GETJPI 렉시칼 함수를 사용하여 반환된 각 PID 번호에 대한 사용자 이름, 프로세스 이름, 프로세스 상태, 프로세스 우선 순위, 프로세스 기본 우선 순위 등을 가져옵니다. F\$FAO 렉시칼 함수는 이 정보를 화면 표시로 서식화합니다.

SYS.COM 명령 프로시저 보기 실행

\$ @SYS

PID	Username	Term	Process name	State	Pri	Image
00050011	NETNONPRIV	-NET-	MAIL_14411	LEF	9/4	MAIL
00040013	STOVE	RTA6:	STOVE	LEF	9/4	
00140015	MAROT	-DET-	DMFB0ACP	HIB	9/8	F11BAC
00080016	THOMPSON	-DET-	MTA0ACP	HIB	12/8	MTAAACP
00070017	JUHLES	TTF1:	JUHLES	LEF	9/4	
.						
.						
.						
00040018	MARCO	TTA2:	MARCO	HIB	9/4	RTPAD
0018001A	VERN	RTA3:	VERN	LEF	9/4	
0033001B	YISHA	RTA7:	YISHA	CUR	4/4	
0002004A	SYSTEM	-DET-	ERRFMT	HIB	12/7	ERRFMT

이 프로시저는 시스템에 있는 모든 프로세스에 대한 정보를 반환합니다. 현재 프로세스에 world 권한이 있습니다.

GETPARMS.COM 명령 프로시저

이 명령 프로시저는 프로시저에 전달된 매개 변수 개수를 반환합니다. 다른 프로시저에서 GETPARMS.COM을 호출하여 호출 프로시저에 전달된 매개 변수 수를 확인할 수 있습니다.

보기: GETPARMS.COM

```
$ ! Procedure to count the number of parameters passed to a command
$ ! procedure. This number is returned as the global symbol PARMCOUNT.

$ !
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE") [1]
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ ! $ IF P1 .EQS. "?" THEN GOTO TELL [2]
$ !
$ ! Loop to count the number of parameters passed. Null parameters are
$ ! counted until the last non-null parameter is passed.
$ !
$ COUNT = 0 [3]
$ LASTNONNULL = 0
$ LOOP:
$ IF COUNT .EQ. 8 THEN GOTO END_COUNT
$ COUNT = COUNT + 1
$ IF P'COUNT' .NES. "" THEN LASTNONNULL = COUNT
$ GOTO LOOP
$ !
$ END_COUNT: [4]
$ !
$ ! Place the number of non-null parameters passed into PARMCOUNT.
$ !
$ PARMCOUNT == LASTNONNULL
$ !
$ ! Restore verification setting, if it was on, before exiting
$ ! [5]
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(SAVE_VERIFY_PROCEDURE,SAVE_VERIFY_IMAGE)
$ EXIT
$ !
$ TELL: [6]
$ TYPE SYS$INPUT
    This procedure counts the number of parameters passed to
    another procedure. This procedure can be called by entering
    the following string in any procedure:

        @GETPARMS 'P1 'P2 'P3 'P4 'P5 'P6 'P7 'P8

    On return, the global symbol PARMCOUNT
    contains the number of parameters passed to the procedure.
$ !
$ EXIT
```

GETPARMS.COM 명령 프로시저 설명

- [1] 이 프로시저는 확인 기능을 끄기 전에 현재 이미지 및 프로시저 확인 설정을 저장합니다.
- [2] 물음표 문자가 프로시저에 매개 변수로 전달된 경우 프로시저는 레이블 TELL에서 분기됩니다(설명 6 참조).
- [3] 프로시저에 전달된 매개 변수의 수를 계산하도록 루프가 설정되어 있습니다. 카운터 COUNT 및 LASTNONNULL은 루프로 들어가기 전에 0으로 초기화됩니다. 루프 내에서 COUNT는 값 8까지 증가되고 테스트됩니다. COUNT가 8이면 최대 매개 변수 개수가 입력된 것입니다. null이 아닌 매개 변수가 전달될 때마다 LASTNONNULL이 해당 매개 변수의 번호와 일치됩니다.

IF 명령이 실행될 때마다 기호 COUNT의 값이 달라집니다. 처음 실행될 때는 COUNT 값이 1이고 IF 명령이 P1을 검사합니다. 두 번째 실행될 때는 P2를 검사합니다. 세 번째는 P3를 네 번째는 P4의 순으로 검사합니다.
- [4] 매개 변수 개수가 8에 도달하면 프로시저가 END_COUNT에서 분기됩니다. 기호 LASTNONNULL에는 전달된 null이 아닌 마지막 매개 변수의 개수가 포함됩니다. 이 값은 글로벌 기호 PARMCOUNT에 포함됩니다. PARMCOUNT를 글로벌 기호로 정의해야 이 매개 변수의 값을 호출 명령 수준에서 테스트할 수 있습니다.
- [5] 원본 확인 설정이 복원됩니다.
- [6] 레이블 TELL에서 TYPE 명령은 입력 스트림에 포함되는 데이터를 표시합니다. 명령 프로시저에서 입력 스트림은 명령 프로시저 파일입니다. TYPE 명령은 GETPARMS.COM 사용 방법에 대한 지침을 표시합니다.

GETPARMS.COM 명령 프로시저 보기 실행

프로시저 SORTFILES.COM에서는 사용자가 null이 아닌 세 매개 변수를 전달해야 합니다. SORTFILES.COM 프로시저에는 다음과 같은 행이 포함될 수 있습니다.

```
$ @GETPARMS 'P1' 'P2' 'P3' 'P4' 'P5' 'P6' 'P7' 'P8'
$ IF PARMCOUNT .NE. 3 THEN GOTO NOT_ENOUGH
.
.
.
$NOT_ENOUGH:
$ WRITE SYS$OUTPUT -
  "Three non-null parameters required. Type SORTFILES HELP for info."
$
```

EXIT 프로시저 SORTFILES.COM은 다음과 같이 호출될 수 있습니다.

```
$ @SORTFILES DEF 4
```

Three non-null parameters required. Type SORTFILE HELP for info.

이 프로시저를 올바르게 호출하려면(즉, SORTFILES에 전달되는 매개 변수가

GETPARMS에 그대로 전달되어 처리되게 하려면) 기호 P1에서 P8까지가 단일 물음표로 묶여 있어야 합니다. GETPARMS의 반환 값이 3이 아닐 경우 SORTFILES는 오류 메시지를 출력하고 종료됩니다.

EDITALL.COM 명령 프로시저

이 명령 프로시저는 **EDT** 편집기를 반복적으로 호출하여 파일 그룹을 동일한 파일 유형으로 편집합니다. 이 프로시저에서는 렉시칼 함수를 사용하여 칼럼 형식 출력으로부터 파일 이름을 추출하는 방법을 보여 줍니다. 또한, 명령 프로시저 내에서 호출되는 프로그램에 대한 입력 스트림을 재정의하는 방법을 설명합니다.

보기: EDITALL.COM

```
$ ! Procedure to edit all files in a directory with a
$ ! specified file type. Use P1 to indicate the file type.
$ !
$ ON CONTROL_Y THEN GOTO DONE          ! Ctrl/Y action [1]
$ ON ERROR THEN GOTO DONE
$ !
$ ! Check for file type parameter.  If one was entered, continue;
$ ! otherwise, prompt for a parameter.
$ !
$ IF P1 .NES. " " THEN GOTO OKAY [2]
$ INQUIRE P1 "Enter file type of files to edit"
$ !
$ ! List all files with the specified file type and write the DIRECTORY
$ ! output to a file named DIRECT.OUT
$ !
$ OKAY:
$ DIRECTORY/VERSIONS=1/COLUMNS=1 - [3]
$ /NODATE/NOSIZE -
$ /NOHEADING/NOTRAILING -
$ /OUTPUT=DIRECT.OUT *. 'P1'
$ IF .NOT. $STATUS THEN GOTO ERROR_SEC [4]
$ !
$ OPEN/READ/ERROR=ERROR_SEC DIRFILE DIRECT.OUT [5]
$ !
$ ! Loop to read directory file
$ !
$ NEWLINE: [6]
$     READ/END=DONE DIRFILE NAME
$     DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:    ! Redefine SYS$INPUT
$     EDIT 'NAME'                                ! Edit the file
$     GOTO NEWLINE
$ !
$ DONE: [7]
$     CLOSE DIRFILE/ERROR=NOTOPEN                ! Close the file
$ NOTOPEN:
$     DELETE DIRECT.OUT;*                         ! Delete temp file
$ EXIT
$ !
$ ERROR_SEC:
$     WRITE SYS$OUTPUT "Error:  ",F$MESSAGE($STATUS)
$     DELETE DIRECT.OUT;*
$ EXIT
```

EDITALL.COM 명령 프로시저 설명

- [1] ON 명령은 이 프로시저에 대한 처리 조건을 설정합니다. 이 프로시저를 실행하는 동안에 Ctrl/Y를 누르면 프로시저가 레이블 **DONE**에서 분기됩니다. 오류 또는 심각한 오류가 발생할 경우에도 프로시저가 레이블 **DONE**에서 분기됩니다.
- [2] 이 프로시저는 매개 변수가 입력되었는지 여부를 검사합니다. 매개 변수가 입력되지 않은 경우 프로시저는 파일 형식을 묻는 프롬프트를 표시합니다.
- [3] DIRECTORY 명령은 파일 형식이 **P1**으로 지정된 모든 파일을 나열합니다. 명령 입력은 **DIRECT.OUT**에 기록됩니다. **/VERSIONS=1** 한정자는 각 파일의 최상위 버전만 나열합니다. **/NOHEADING** 및 **/NOTRILING** 한정자는 출력에 제목행 또는 디렉토리 요약 포함하지 않습니다. **/COLUMNS=1** 한정자는 레코드마다 파일 이름을 하나씩 지정합니다.
- [4] IF 명령은 DIRECTORY 명령이 **\$STATUS** 값을 테스트하여 반환되는 값을 검사합니다. DIRECTORY 명령이 성공적으로 완료되지 않으면 **\$STATUS**는 정수 값만 갖고 프로시저가 레이블 **ERROR_SEC**에서 분기됩니다.
- [5] OPEN 명령은 디렉토리 출력 파일을 열고 해당 파일에 논리 이름 **DIRFILE**을 지정합니다.
- [6] READ 명령은 DIRECTORY 명령 출력에서 기호 이름 **NAME**으로 한 행씩 읽습니다. 각 행을 읽은 후 프로시저는 **DEFINE** 명령을 사용하여 편집 세션에 대한 입력 스트림(**SYS\$INPUT**)이 터미널이 되도록 재정의합니다. 그런 다음 기호 **NAME**을 파일 지정 방법으로 지정하여 편집기를 호출합니다. 편집 세션이 완료되면 명령 인터프리터가 명령 프로시저의 다음 행을 읽고 레이블 **NEWLINE**에서 분기됩니다. 프로시저가 디렉토리에서 지정된 파일 형식을 갖는 파일을 모두 편집하면 레이블 **DONE**에서 분기됩니다.
- [7] 레이블 **DONE**은 READ 명령에 대한 **/END** 한정자의 대상 레이블이고 프로시저 시작 시 설정한 **ON CONTROL_Y** 및 **ON ERROR** 조건에 대한 대상 레이블입니다. 이 레이블에서 프로시저는 필요한 정리 작업을 수행합니다.

CLOSE 명령은 **DIRECTORY** 명령 출력 파일을 닫고 **/ERROR** 한정자는 파일에서 그 다음 행에 대한 레이블을 지정합니다. **/ERROR**를 사용하면 디렉토리 파일이 열리지 않는 경우 표시되는 오류 메시지를 무시합니다. 예를 들어, 디렉토리 파일이 열리기 전에 Ctrl/Y를 누를 경우 이 오류 메시지가 발생합니다.

정리의 두 번째 단계에서는 임시 디렉토리 파일을 삭제합니다.

EDITALL.COM 명령 프로시저 보기 실행

```
$ @EDITALL DAT  
* .
```

```
%DELETE-I-FILDEL, device:[directory]DIRECT.OUT;1 deleted (x blocks)
```

프로시저 **EDITALL**은 **P1**을 **.DAT**로 지정하여 호출됩니다. 이 프로시저는 파일 형식이 **.DAT**인 기본 디렉토리에 있는 모든 파일의 디렉토리 목록을 작성하고 각 파일을 편집할 편집기를 호출합니다. 파일 형식이 **.DAT**인 마지막 파일이 편집되면 프로시저는 임시 파일 **DIRECT.OUT**을 삭제하고 터미널에 정보 메시지를 표시합니다.

MAILEDIT.COM 명령 프로시저

이 명령 프로시저는 Mail 유틸리티에서 텍스트 편집기를 호출합니다.

보기: MAILEDIT.COM

```
$ ! Command procedure to invoke an editor for Mail.
$ !
$ ! Inputs:
$ !
$ ! P1 = Input file name.
$ ! P2 = Output file name.
$ !
$ ! If MAIL$EDIT is undefined, Mail will invoke the user's selected
$ ! callable editor set by the mail SET EDITOR command.
$ !
$ ! If MAIL$EDIT is defined to be a command procedure, Mail will create
$ ! a subprocess to edit the mail, but any SET EDITOR command in Mail
$ ! will override the definition of MAIL$EDIT for the remainder of that
$ ! Mail session.
$ !
$ ! Note that this procedure is run in the context of a subprocess.
$ ! LOGIN.COM is not executed. However, all process logical names
$ ! and DCL global symbols are copied. In particular, note that the
$ ! user's individual definition of the symbol EDIT is used if there
$ ! is one. Otherwise, the system default editor is used.
$ !
$ ! The default directory is the same as the parent process
$ !
$ DEFINE /USER SYS$INPUT 'F$TRNLNM("SYS$OUTPUT")' [1]
$ IF P1 .EQS. "" THEN GOTO NOINPUT [2]
$ EDIT /OUTPUT='P2' 'P1'[3]
$ EXIT
$NOINPUT:
$ EDIT 'P2'[4]
$ EXIT
```

MAILEDIT.COM 명령 프로시저 설명

- [1] DEFINE 명령을 사용하면 명령 파일 대신 터미널에서 편집기 입력을 가져올 수 있습니다.
- [2] IF 문은 출력 파일 이름이 다른 파일의 편집과 파일 이름이 동일한 파일의 편집을 구분합니다.
- [3] 이 EDIT 명령은 입력 파일 이름과 출력 파일 이름을 사용하여 편집기를 호출합니다. 이 행을 편집하여 다른 편집기를 호출할 수 있습니다. 예를 들면, 다음과 같습니다.

```
$ RUN XYZ_EDITOR.EXE /INPUT= 'P1' /OUTPUT='P2'
```


- [4] 이 **EDIT** 명령은 단일 파일 이름을 사용하여 편집기를 호출합니다. 이 행을 편집하여 다른 편집기를 호출할 수 있습니다. 예를 들면, 다음과 같습니다.

```
$ RUN XYZ_EDITOR.EXE /INPUT= 'P2' /OUTPUT='P2'
```

MAILEDIT.COM 명령 프로시저 보기 실행

```
$ DEFINE MAIL$EDIT MAILEDIT.COM  
$ MAIL  
MAIL>SHOW EDITOR  
Your editor is defined by the file MAILEDIT.COM.
```

FORTUSER.COM 명령 프로시저

FORTTRAN 프로그램을 작성, 컴파일 및 실행하는 대화형 사용자에게 대한 터미널 환경을 제어하는 시스템 정의 로그인 명령 프로시저 보기를 제공합니다. FORTUSER.COM이 로그인 명령 프로시저로 나열되는 전용 계정에 로그인한 사용자는 FORTUSER.COM에서 수락한 명령만 실행할 수 있습니다. 이 프로시저는 렉시칼 함수를 사용하여 사용자가 입력한 명령을 유효 명령 목록과 비교하여 옵션 테이블을 단계별로 처리하는 방법을 설명합니다.

보기: FORTUSER.COM

```
$ ! Procedure to create, compile, link, execute, and debug
$ ! FORTRAN programs. Users can enter only the commands listed
$ ! in the symbol OPTION_TABLE.
$ SET NOCONTROL=Y [1]
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ OPTION_TABLE =
"EDIT/COMPILE/LINK/RUN/EXECUTE/DEBUG/PRINT/HELP/FILE/DONE/" [2]
$ TYPE SYS$INPUT [3]
    VMS FORTRAN Command Interpreter

    Enter name of file with which you would like to work.

$ !
$ ! Set up for initial prompt
$ !
$ PROMPT = "INIT" [4]
$ GOTO HELP ! Print the initial help message
$ !
$ ! after the first prompting message, use the prompt: Command
$ !
$ INIT:
$ PROMPT = "GET_COMMAND"
$ GOTO FILE ! Get initial file name
$ !
$ ! Main command parsing routine. The routine compares the current
$ ! command against the options in the option table. When it finds
$ ! a match, it branches to the appropriate label.
$ !
$ GET_COMMAND:
$     ON CONTROL_Y THEN GOTO GET_COMMAND ! Ctrl/Y resets prompt [5]
$     SET CONTROL=Y
$     ON WARNING THEN GOTO GET_COMMAND ! If any, reset prompt
$     INQUIRE COMMAND "Command"
$     IF COMMAND .EQS. "" THEN GOTO GET_COMMAND
$     IF F$LOCATE(COMMAND + "/", OPTION_TABLE) .EQ. F$LENGTH(OPTION_TABLE) -[6]
THEN GOTO INVALID_COMMAND
$     GOTO 'COMMAND'
$ !
$ INVALID_COMMAND: [7]
$     WRITE SYS$OUTPUT " Invalid command"
$ !
```

```
$ HELP: [8]
$      TYPE SYS$INPUT
      The commands you can enter are:
      FILE      Name of FORTRAN program in your current
                  default directory. Subsequent commands
                  process this file.
      EDIT      Edit the program.
      COMPILE   Compile the program with FORTRAN.
      LINK      Link the program to produce an executable image.
      RUN       Run the program's executable image.
      EXECUTE   Same function as COMPILE, LINK, and RUN.
      DEBUG     Run the program under control of the debugger.
      PRINT     Queue the most recent listing file for printing.
      DONE     Return to interactive command level.
      HELP     Print this help message.

      Enter Ctrl/Y to restart this session
$ GOTO 'PROMPT' [9]
$ EDIT: [10]
$      DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$      EDIT 'FILE_NAME'.FOR
$      GOTO GET_COMMAND
$ COMPILE:
$      FORTRAN 'FILE_NAME'/LIST/OBJECT/DEBUG
$      GOTO GET_COMMAND
$ LINK:
$      LINK 'FILE_NAME'/DEBUG
$      PURGE 'FILE_NAME'./KEEP=2
$      GOTO GET_COMMAND
$ RUN:
$      DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$      RUN/NODEBUG 'FILE_NAME'
$      GOTO GET_COMMAND
$ DEBUG:
$      DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$      RUN 'FILE_NAME'
$      GOTO GET_COMMAND
$ EXECUTE:
$      FORTRAN 'FILE_NAME'/LIST/OBJECT
$      LINK/DEBUG 'FILE_NAME'
$      PURGE 'FILE_NAME'./KEEP=2
$      RUN/NODEBUG 'FILE_NAME'
$      GOTO GET_COMMAND
$ PRINT:
$      PRINT 'FILE_NAME'
$      GOTO GET_COMMAND
$ BADFILE: [11]
$      WRITE SYS$OUTPUT "File must be in current default directory."
$ FILE:
$      INQUIRE FILE_NAME "File name"
$      IF FILE_NAME .EQS. "" THEN GOTO FILE
$      IF F$PARSE(FILE_NAME,,, "DIRECTORY") .NES. F$DIRECTORY() - [12]
      THEN GOTO BADFILE
$      FILE_NAME = F$PARSE(FILE_NAME,,, "NAME")
```

```
$          GOTO GET_COMMAND
$ DONE:
$ EXIT
```

FORTUSER.COM 명령 프로시저 설명

- [1] SET NOCONTROL=Y 명령은 이 프로시저의 통제에 따라 로그인하는 사용자가 프로시저나 프로시저의 명령 또는 프로그램을 중단하지 못하게 합니다.
- [2] 옵션 테이블에는 사용자가 실행할 수 있는 명령이 나열됩니다. 각 명령은 슬래시로 분리됩니다.
- [3] 이 프로시저는 자동으로 소개됩니다.
- [4] 기호 이름 **PROMPT**에는 프로시저의 레이블 값이 지정됩니다. 프로시저가 처음 호출될 때 이 기호는 값 **INIT**를 갖습니다. **HELP** 명령 텍스트는 레이블 **PROMPT**를 지정하는 **GOTO** 명령으로 종료됩니다. 이 텍스트가 처음으로 표시되면 **GOTO** 명령이 레이블 **HELP**에서 분기됩니다. 그러면 사용자가 입력할 수 있는 명령을 설명하는 **HELP** 메시지가 표시됩니다. 그런 다음 프로시저가 레이블 **INIT**에서 다시 분기되어 **PROMPT** 값이 "**GET_COMMAND**"로 변경됩니다. 마지막으로 프로시저가 레이블 **FILE**에서 분기되어 파일 이름을 가져옵니다. 따라서, 도움말 텍스트가 표시되면 프로시저가 레이블 **GET_COMMAND**에서 분기되어 다음 명령을 가져옵니다.
- [5] Ctrl/Y 조건 동작은 경고 조건 동작처럼 레이블 **GET_COMMAND**로 복귀하도록 설정됩니다. 프로시저는 명령을 묻는 프롬프트를 표시하고 아무것도 입력하지 않더라도 프롬프트를 계속해서 표시합니다. 세션을 종료하고 대화형 명령 수준으로 복귀하려면 명령 **DONE**을 입력합니다.
- [6] 이 프로시저는 **F\$LOCATE** 및 **F\$LENGTH** 텍시칼 함수를 사용하여 명령이 옵션 목록에 포함되는지 여부를 결정합니다. **F\$LOCATE** 함수는 슬래시 뒤에 있는 사용자 입력 명령을 검사합니다. 예를 들어, 사용자가 **EDIT**을 입력하면 프로시저는 **EDIT**을 검사합니다. 명령이 옵션 목록에 포함되지 않을 경우 프로시저가 레이블 **INVALID_COMMAND**에서 분기됩니다. 명령이 유효하면 프로시저가 해당 레이블에서 분기됩니다.
- [7] 레이블 **INVALID_COMMAND**에서 프로시저는 오류 메시지를 기록하고 유효한 명령을 나열하는 도움말 텍스트를 표시합니다.
- [8] 도움말 텍스트에는 유효한 명령이 나열됩니다. 이 텍스트는 처음부터 표시되며, 사용자가 **HELP** 명령 또는 유효하지 않은 명령을 입력할 때마다 표시됩니다.

- [9] **HELP** 텍스트의 마지막 부분에서 **GOTO** 명령은 기호 이름 **PROMPT**를 지정합니다. 이 프로시저가 처음 호출될 때 기호는 값 **INIT**를 가지고, 그 이후에는 값 **GET_COMMAND**를 가집니다.
- [10] 목록에 있는 각 유효 명령은 옵션 테이블에 해당 엔트리가 있고 명령 프로시저에 해당 레이블이 있습니다. 터미널에서 입력을 읽는 명령(예: **EDIT**)의 경우 프로시저는 입력 스트림을 **SYS\$COMMAND**로 정의하는 **DEFINE** 명령을 포함합니다.
- [11] 레이블 **BADFILE**에서 프로시저는 파일이 현재 디렉토리에 있어야 함을 나타내는 메시지를 표시합니다. 그런 다음 다른 파일 이름을 묻는 프롬프트를 표시합니다.
- [12] 파일 이름을 입력하면 프로시저는 현재 기본 디렉토리나 다른 디렉토리를 지정하지 않았는지 확인합니다. 그런 다음 프로시저는 **F\$PARSE** 함수를 사용하여 파일 이름을 추출합니다. 각 명령은 해당 기본 파일 형식을 제공합니다. 파일을 처리할 명령을 가져올 수 있도록 프로시저가 레이블 **GET_COMMAND**에서 다시 분기됩니다.

FORTUSER.COM 명령 프로시저 보기 실행

다음 보기에서는 이 명령 프로시저를 전용 명령 프로시저로 사용하는 방법을 설명합니다.

```
Username:CLASS30
Password:
OpenVMS Version 7.1

OpenVMS FORTRAN Command Interpreter
Enter name of file with which you would like to work.
The commands you can enter are:
FILE Name of FORTRAN program in your current
default directory. Subsequent commands
process this file.
EDIT Edit the program. COMPILE
Compile the program with VAX FORTRAN.
LINK Link the program to produce an executable image.
RUN Run the program's executable image.
EXECUTE Same function as COMPILE, LINK and RUN.
DEBUG Run the program under control of the debugger.
PRINT Queue the most recent listing file for printing.
DONE Return to interactive command level.
HELP Print this help message.

Enter Ctrl/Y to restart this session
File name: AVERAGE
Command: COMPILE
Command: LINK
Command: RUN
Command: FILE
File name: READFILE
Command: EDIT
```

이 보기 실행에서는 이름이 **CLASS30**인 사용자가 **FORTUSER** 명령 프로시저에 의해 제어되는 계정에 로그인하는 세션을 보여 줍니다. **FORTUSER** 명령 프로시저는 사용자가 실행할 수 있는 명령과 세션을 다시 시작하기 위한 명령을 표시합니다. 사용자가 파일 **AVERAGE**를 지정하고, 해당 파일을 컴파일, 링크 및 실행합니다. 그런 다음 사용자는 **FILE** 명령을 입력하여 다른 파일 작업을 시작합니다.

LISTER.COM 명령 프로시저

입력 데이터를 묻는 프롬프트를 표시하고, 열에 있는 데이터를 서식화한 다음 출력 파일로 정렬합니다. 이 프로시저에서는 **READ** 및 **WRITE** 명령과 할당문의 문자 하위 문자열 오버레이 서식을 보여 줍니다.

보기: LISTER.COM

```
$ ! Procedure to accumulate programmer names and document file names.
$ ! After all programmer names and file names are entered, they are
$ ! sorted in alphabetic order by programmer name and printed on
$ ! the system printer.
$ !
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE") [1]
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ !
$ OPEN/WRITE OUTFILE DATA.TMP          ! Create output file [2]
$ !
$ ! Loop to obtain programmers' last names and file names,
$ ! and write this data to DATA.TMP.
$ !
$ LOOP: [3]
$     INQUIRE NAME "Programmer (press Return to quit)"
$     IF NAME .EQS. "" THEN GOTO FINISHED
$     INQUIRE FILE "Document file name"
$     RECORD[0,20]:='NAME' [4]
$     RECORD[21,20]:='FILE'
$     WRITE OUTFILE RECORD
$     GOTO LOOP
$ FINISHED:
$     CLOSE OUTFILE
$ !
$ DEFINE/USER_MODE SYS$OUTPUT: NL:      ! Suppress sort output $
SORT/KEY=(POSITION:1,SIZE=20) DATA.TMP DOC.SRT [5]
$ !
$ OPEN/WRITE OUTFILE DOCUMENT.DAT [6]
$ WRITE OUTFILE "Programmer Files as of ",F$TIME()
$ WRITE OUTFILE ""
$ RECORD[0,20]:="Programmer Name"
$ RECORD[21,20]:="File Name"
$ WRITE OUTFILE RECORD
$ WRITE OUTFILE ""
$ !
$ CLOSE OUTFILE [7]
$ APPEND DOC.SRT DOCUMENT.DAT
$ PRINT DOCUMENT.DAT

$ !
$ INQUIRE CLEAN_UP "Delete temporary files [Y,N]" [8]
$ IF CLEAN_UP THEN DELETE DATA.TMP;*,DOC.SRT;*
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(SAVE_VERIFY_PROCEDURE,SAVE_VERIFY_IMAGE)
$ EXIT
```

LISTER.COM 명령 프로시저 설명

- [1] LISTER.COM은 현재 확인 설정을 저장하고 확인 기능을 비활성화합니다.
- [2] OPEN 명령은 기록할 임시 파일을 작성합니다.
- [3] INQUIRE 명령은 프로그래머 이름과 파일 이름을 묻는 프롬프트를 표시합니다.
Return 키를 눌러 INQUIRE 명령 프롬프트에 대한 응답으로 null 행을 입력하면 프로시저는 더 이상의 데이터가 입력되지 않는 것으로 간주하고 레이블 FINISHED에서 분기됩니다.
- [4] 기호 NAME 및 FILE에 값을 할당한 후 프로시저는 할당문의 문자열 오버레이 서식을 사용하여 기호 RECORD에 대한 값을 구성합니다. RECORD의 1열부터 21열까지에 NAME의 현재 값이 기록됩니다. 명령 인터프리터는 공백을 사용하여 NAME 값을 패딩하여 지정된 20자 길이를 채웁니다.

마찬가지로 RECORD의 다음 20열에 FILE 값이 채워집니다. 그런 다음 RECORD 값이 출력 파일에 기록됩니다.
- [5] 파일이 닫힌 다음 프로시저는 출력 파일 DATA.TMP를 정렬합니다. DEFINE 명령은 null 파일 NL에 SORT 명령 출력을 지시합니다. 그렇지 않으면 다음과 같은 통계가 터미널에 표시됩니다.

처음 20열에 대해 프로그래머 이름 순으로 정렬이 수행됩니다.

정렬된 출력 파일의 이름은 DOC.SRT입니다.
- [6] 이 프로시저는 OPEN 명령을 사용하여 최종 출력 파일 DOCUMENT.DAT를 작성합니다. 파일에 기록된 첫째 행은 머리글 행으로 해당 열의 머리글, 날짜와 시간, 제목 등이 제공됩니다.
- [7] 프로시저는 파일 DOCUMENT.DAT를 닫고 해당 파일에 정렬된 출력 파일 DOC.SRT를 첨부합니다. 그런 다음 출력 파일을 시스템 프린터의 대기열에 등록합니다.
- [8] 프로시저는 중간 파일을 삭제할지 여부를 결정하라는 프롬프트를 표시합니다. INQUIRE 명령 프롬프트에 참에 해당하는 응답(T, t, Y, y)을 입력하면 파일 DATA.TMP와 DOC.SRT가 삭제됩니다. 그렇지 않으면 해당 파일이 유지됩니다.

LISTER.COM 명령 프로시저 보기 실행

```
$ @LISTER
Programmer:WATERS
Document file name:CRYSTAL.CAV
Programmer:JENKINS
Document file name:MARIGOLD.DAT
Programmer:MASON
Document file name:SYSTEM.SRC
Programmer:ANDERSON
Document file name:JUNK.J
Programmer:Return
Delete temporary files [Y,N]:y
```

이 프로시저 실행의 결과 출력 파일에는 다음이 포함됩니다.

Programmer Files as of 31-DEC-1999 16:18:58.79

Programmer Name	File Name
ANDERSON	JUNK.J
JENKINS	MARIGOLD.DAT
MASON	SYSTEM.SRC
WATERS	CRYSTAL.CAV

CALC.COM 명령 프로시저

산술 계산을 수행하고 결과 값을 16진수 및 10진수 값으로 변환합니다.

보기: CALC.COM

```
$ ! Procedure to calculate expressions.  If you enter an
$ ! assignment statement, then CALC.COM evaluates the expression
$ ! and assigns the result to the symbol you specify.  In the next
$ ! iteration, you can use either your symbol or the symbol Q to
$ ! represent the current result.
$ !
$ ! If you enter an expression, then CALC.COM evaluates the
$ ! expression and assigns the result to the symbol Q.  In
$ ! the next iteration, you can use the symbol Q to represent
$ ! the current result.
$ !
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(0)
$ START:
$     ON WARNING THEN GOTO START [1]
$     INQUIRE STRING "Calc" [2]
$     IF STRING .EQS. "" THEN GOTO CLEAN_UP
$     IF F$LOCATE("=",STRING) .EQ. F$LENGTH(STRING) THEN GOTO EXPRESSION
$ !
$ ! Execute if string is in the form symbol = expression
$ STATEMENT: [3]
$     'STRING' ! Execute assignment statements
$     SYMBOL = F$EXTRACT(0,F$LOCATE("=",STRING)-1,STRING) ! get symbol name
$     Q = 'SYMBOL' ! Set up q for future iterations
$     LINE = F$FAO("Decimal = !SL      Hex = !-!XL      Octal = !-!OL",Q)
$     WRITE SYS$OUTPUT LINE
$     GOTO START
$ !
$ !
$ ! Execute if string is an expression
$ EXPRESSION: [4]
$     Q = F$INTEGER('STRING') ! Can use Q in next iteration
$     LINE = F$FAO("Decimal = !SL      Hex = !-!XL      Octal = !-!OL",Q)
$     WRITE SYS$OUTPUT LINE
$     GOTO START
$ !
$ CLEAN_UP:
$ SAVE_VERIFY_PROCEDURE = F$VERIFY(SAVE_VERIFY_PROCEDURE,SAVE_VERIFY_IMAGE)
$ EXIT
```

CALC.COM 명령 프로시저 참고

- [1] 이 프로시저는 프로시저를 다시 시작하는 오류 처리 조건을 설정합니다. 심각도가 높은 오류 또는 경고가 발생하면 프로시저가 시작 부분에서 분기되어 ON 조건을 초기화합니다.

이 기술은 사용자가 식을 잘못 입력하더라도 프로시저가 종료되지 않게 합니다.

- [2] INQUIRE 명령은 산술식을 묻는 프롬프트를 표시합니다. 프로시저는 다음과 같은 형식으로 식을 적용합니다.

name = expression

expression

Return 키를 누르면 프로시저는 CALC 세션이 끝난 것으로 간주하고 종료합니다.

"name = expression" 형식으로 입력하면 프로시저가 레이블 STATEMENT에서 계속 실행됩니다. 그렇지 않으면 프로시저가 레이블 EXPRESSION에서 분기됩니다.

- [3] 프로시저는 할당문을 실행하고 식의 결과를 기호로 지정합니다. 그런 다음 기호 이름을 추출하고 기호 값을 Q로 지정합니다. 이렇게 하면 다음에 프로시저를 반복하는 동안 Q 또는 기호를 사용할 수 있습니다. 그런 다음 결과를 표시하고 레이블 START에서 다시 분기됩니다.

- [4] 프로시저는 식을 평가하고 결과를 기호 Q로 지정합니다. 이렇게 하면 다음에 프로시저를 반복하는 동안 Q를 사용할 수 있습니다. 그런 다음 결과를 표시하고 레이블 START에서 다시 분기됩니다.

CALC.COM 명령 프로시저 보기 실행

```
$ @CALC
Calc:2 * 30
Decimal = 60 Hex = 0000003C Octal = 000000000074
Calc:Q + 3 Decimal = 63 Hex = 0000003F Octal = 000000000077
Calc:TOTAL = Q + 4 Decimal = 67 Hex = 00000043 Octal = 000000000103
Calc:5 + 7
Decimal = 12 Hex = 0000000C Octal = 000000000014
Calc:Return
$
```

프로시저에서 각 프롬프트 다음에 사용자는 산술식을 입력합니다. 이 프로시저는 결과를 10진수, 16진수 및 8진수로 표시합니다. 데이터가 없는 행에서 Return 키를 눌러 null 행을 지정하면 CALC 세션이 끝납니다.

BATCH.COM 명령 프로시저

명령 문자열, 명령 프로시저 또는 명령 목록을 적용한 다음 해당 명령을 일괄 처리 작업으로 실행합니다.

보기: BATCH.COM

```
$ VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ VERIFY_PROCEDURE = F$VERIFY(0)
$!
$! Turn off verification and save current settings.
$! (This comment must appear after you turn verification
$! off; otherwise it will appear in the batch job log file.)
$!
$!
$! If this is being executed as a batch job,
$! (from the SUBMIT section below) go to the EXECUTE_BATCH_JOB section
$! Otherwise, get the information you need to prepare to execute the
$! batch job.
$!
$ IF F$MODE() .EQS. "BATCH" THEN GOTO EXECUTE_BATCH_JOB [1]
$!
$!
$! Prepare to submit a command (or a command procedure) as a batch job.
$! First, determine a mnemonic process name for the batch job. Use the
$! following rules:
$!
$! 1) If the user is executing a single command, then use the verb name.
$! Strip off any qualifiers that were included with the command.
$! 2) If the user is executing a command procedure, then use the file name.
$! 3) Otherwise, use BATCH.
$!
$ JOB_NAME = P1 [2]
$ IF JOB_NAME .EQS. "" THEN JOB_NAME = "BATCH"
$ IF F$EXTRACT(0,1,JOB_NAME) .EQS. "@" THEN JOB_NAME =
F$EXTRACT(1,999,JOB_NAME)
$ JOB_NAME = F$EXTRACT(0,F$LOCATE("/",JOB_NAME),JOB_NAME)
$ JOB_NAME = F$PARSE(JOB_NAME,, "NAME", "SYNTAX_ONLY")
$ IF JOB_NAME .EQS. "" THEN JOB_NAME = "BATCH"
$!
$!
$! Get the current default device and directory.
$!
$ ORIGDIR = F$ENVIRONMENT("DEFAULT")
$!
$!
$! Concatenate the parameters to form the command string to be executed.
$! If the user did not enter a command string, the symbol COMMAND will have
$! a null value.
$!
$ COMMAND = P1 + " " + P2 + " " + P3 + " " + P4 + " " + - [3]
P5 + " " + P6 + " " + P7 + " " + P8 $!
```

```

$!
$! If the user is executing a single command and if both the command and the
$! original directory specification are small enough to be passed as
$! parameters to the SUBMIT command, then submit the batch job now.
$!
$ IF (P1 .NES. "") .AND. (F$LENGTH(COMMAND) .LE. 255) .AND. - [4]
(F$LENGTH(ORIGDIR) .LE. 255) THEN GOTO SUBMIT
$!
$!
$! If the single command to be executed in the batch job is very large, or
$! if you have to prompt for commands to execute in the batch job, then
$! create a temporary command procedure to hold those commands and get the
$! fully expanded name of the command procedure.
$!
$ CREATE_TEMP_FILE:
$   ON CONTROL_Y THEN GOTO CONTROL_Y_HANDLER [5]
$   OPEN/WRITE/ERROR=FILE_OPEN_ERROR TEMPFILE SYS$SCRATCH:'JOB_NAME'.TMP [6]
$   FILESPEC = F$SEARCH("SYS$SCRATCH:" + JOB_NAME + ".TMP")
$!
$! By default, have the batch job continue if it encounters any errors.
$!
$   WRITE TEMPFILE "$ SET NOON"
$!
$! Either write the single large command to the file, or prompt for
$! multiple commands and write them to the file.
$!
$   IF COMMAND .NES. " " THEN GOTO WRITE_LARGE_COMMAND
$
$   LOOP:
$     READ /END_OF_FILE=CLOSE_FILE /PROMPT="Command: " SYS$COMMAND COMMAND
$     IF COMMAND .EQS. " " THEN GOTO CLOSE_FILE
$     WRITE TEMPFILE "$ ",COMMAND
$     GOTO LOOP
$
$ WRITE_LARGE_COMMAND:
$   WRITE TEMPFILE "$ ",COMMAND
$
$!
$! Finish the temporary file by defining a symbol so that you will know
$! the name of the command procedure to delete and then close the file.
$! Define the symbol COMMAND to mean "execute the command procedure
$! you have just created." Then submit the batch job and execute
$! this command procedure in the batch job.
$!
$ CLOSE_FILE: [7]
$   WRITE TEMPFILE "$ BATCH$DELETE_FILESPEC == "",FILESPEC,"""
$   CLOSE TEMPFILE
$   ON CONTROL_Y THEN EXIT
$   COMMAND = "@ " + FILESPEC
$!
$!
$! Submit BATCH.COM as a batch job, and pass it two parameters.
$! P1 is the command (or name of the command procedure) to execute.
$! P2 is the directory from which to execute the command.
$!
$ SUBMIT: [8]

```

```

$   SUBMIT/NOTIFY/NOPRINT 'F$ENVIRONMENT("PROCEDURE")' /NAME='JOB_NAME' -
/PARAMETERS=("'COMMAND'", "'ORIGDIR'")
$   GOTO EXIT
$!
$!
$! The user pressed Ctrl/Y while the temporary command procedure was open.
$! Close the command procedure, delete it if it exists, and exit.
$!
$ CONTROL_Y_HANDLER: [9]
$   CLOSE TEMPFILE
$   IF F$TYPE(FILESPEC) .NES. "" THEN DELETE/NOLOG 'FILESPEC'
$   WRITE SYS$OUTPUT "Ctrl/Y caused the command procedure to abort."
$   GOTO EXIT
$!
$!
$! The temporary command procedure could not be created.
$! Notify the user and exit.
$!
$ FILE_OPEN_ERROR: [10]
$   WRITE SYS$OUTPUT "Could not create sys$scratch:", job_name, ".tmp"
$   WRITE SYS$OUTPUT "Please correct the situation and try again."
$!
$!
$! Restore the verification states and exit. $!
$ EXIT: [11]
$   VERIFY_PROCEDURE = F$VERIFY(VERIFY_PROCEDURE, VERIFY_IMAGE)
$   EXIT
$!
$!
$! BATCH.COM was invoked as a batch job. P1 contains the command
$! to execute and P2 the default directory specification.
$! Return a status code that indicates the termination status of P1.
$!
$ EXECUTE_BATCH_JOB: [12]
$   SET NOON
$   VERIFY_PROCEDURE = F$VERIFY(VERIFY_PROCEDURE, VERIFY_IMAGE)
$   SET DEFAULT 'P2'
$   'P1'
$   IF F$TYPE(BATCH$DELETE_FILESPEC) .EQS. "" THEN EXIT $STATUS
$   STATUS = $STATUS
$   DELETE /NOLOG 'BATCH$DELETE_FILESPEC'
$   EXIT STATUS

```

BATCH.COM 명령 프로시저 설명

- [1] 이 IF 문은 BATCH.COM이 일괄 처리 모드에서 실행 중인지 여부를 테스트합니다. BATCH.COM을 대화형으로 호출할 경우 일괄 처리 작업으로 실행할 명령 프로시저 또는 명령 문자열을 매개 변수로 제공합니다. 매개 변수를 제공하지 않으면 BATCH.COM은 명령을 묻는 프롬프트를 표시하고, 파일에 이 명령을 기록한 다음 이 명령 프로시저를 일괄 처리 작업으로 실행합니다. BATCH.COM은 일괄 처리 작업에서 실행할 명령을 준비한 다음 SUBMIT 명령을 사용하여 일괄 처리 작업으로 실행하고 이 작업에서 명령을 실행합니다(설명 8 참조). BATCH.COM을 일괄 처리 작업으로 호출하면 프로시저가 레이블 EXECUTE_BATCH_JOB에서 분기됩니다. BATCH.COM을 일괄 처리 모드에서 실행할 경우 실행할 명령 또는 명령 프로시저를 P1으로 지정하고 기본 디렉토리를 P2로 지정해야 합니다.
- [2] 이 명령은 일괄 처리 작업을 실행과 비교합니다. 먼저, 프로시저는 일괄 처리 작업에 대한 이름을 구성합니다. 명령 문자열이 전달되면 BATCH.COM은 동사 이름을 작업 이름으로 사용합니다. 명령 프로시저가 전달되면 BATCH.COM은 파일 이름을 사용합니다. 입력이 전달되지 않으면 BATCH.COM은 작업 이름을 BATCH로 지정합니다.
- [3] 매개 변수를 연결하여 실행할 명령 문자열을 형성합니다. 명령 문자열은 기호 COMMAND로 지정됩니다.
- [4] SUBMIT 명령은 255자를 초과하는 매개 변수를 전달할 수 없습니다. 따라서, 이 프로시저는 명령 문자열과 디렉토리 이름이 255자를 넘지 않는지 테스트하여, 명령 문자열과 디렉토리 이름이 모두 255자를 넘지 않고 사용자가 명령 문자열을 전달한 경우 레이블 SUBMIT에서 분기됩니다.
- [5] 이 프로시저는 이 명령 프로시저 섹션을 수행하는 동안 사용자가 Ctrl/Y를 누를 경우 정리 작업이 수행되도록 Ctrl/Y 처리기를 구성합니다.
- [6] 이 프로시저는 실행할 명령을 포함시킬 임시 파일을 작성합니다. 사용자가 긴 명령 문자열을 입력하면 프로시저가 WRITE_LARGE_COMMAND에서 분기되고 임시 파일에 이 명령을 기록합니다. 그렇지 않을 경우 프로시저는 실행할 명령을 묻는 프롬프트를 표시합니다. 각 명령은 임시 파일에 기록됩니다.
- [7] 임시 파일을 닫기 전에 파일에 기호 할당문을 기록하십시오. 이 할당문은 임시 파일에 대한 파일 이름을 기호 BATCH\$DELETE_FILESPEC으로 지정합니다. 임시 파일을 닫은 다음 프로시저는 기본 Ctrl/Y 처리기를 초기화합니다. 그런 다음 기호 COMMAND가 실행될 때 임시 명령 파일을 호출하도록 기호 COMMAND를 정의합니다.

- [8] 이 프로시저는 정의된 작업 이름을 사용하여 일괄 처리 작업으로 자동 실행됩니다(설명 2 참조). 이 프로시저는 실행할 명령 또는 명령 프로시저와 명령을 실행할 디렉토리라는 두 매개 변수를 전달합니다. 그런 다음 레이블 **EXIT**에서 분기됩니다(설명 11 참조).
- [9] 이 섹션에서는 임시 파일이 작성되는 동안 사용자가 **Ctrl/Y**를 입력하면 정리 작업을 수행합니다.
- [10] 이 섹션에서는 임시 파일을 작성할 수 없을 경우 오류 메시지를 기록합니다.
- [11] 프로시저는 원본 확인 설정을 초기화한 다음 종료됩니다.
- [12] **BATCH.COM**이 일괄 처리 모드에서 실행될 때 이 명령이 실행됩니다. 먼저, **ON** 오류 처리가 해제되고 사용자의 기본 확인 설정이 설정됩니다. 그런 다음 기본값이 **P2**에서 나타내는 디렉토리로 설정되고 **P1**에서 나타내는 명령 또는 명령 프로시저가 실행됩니다. 임시 파일이 작성된 경우 이 파일은 삭제됩니다. **BATCH\$DELETE_FILESPEC**을 삭제하기 전에 **P1**의 완료 상태가 저장됩니다. 이 완료 상태는 **EXIT** 명령에 의해 반환됩니다.

BATCH.COM 명령 프로시저 보기 실행

```
$ @BATCH RUN MYPROG
Job RUN (queue SYS$BATCH, entry 1715) started on SYS$BATCH
```

이 보기에서는 **BATCH.COM**을 사용하여 일괄 처리 작업에서 프로그램을 실행합니다.

COMPILE_FILE.COM 명령 프로시저

Pascal 또는 FORTRAN으로 작성된 파일을 컴파일, 링크 및 실행합니다. 이 프로시저는 처리할 파일을 묻는 프롬프트를 표시하고 파일 형식이 .PAS인지 .FOR인지를 확인합니다. 파일 형식이 .PAS 또는 .FOR이 아니거나 현재 기본 디렉토리에 파일이 없을 경우 명령 프로시저는 해당 오류 메시지를 출력합니다. 이 프로시저에서는 IF-THEN-ELSE 언어 구조의 사용 방법을 보여 줍니다.

보기: COMPILE_FILE.COM

```
$! This command procedure compiles, links, and runs a file written in Pascal
$! or FORTRAN.
$!
$ ON CONTROL_Y THEN EXIT
$!
$ TOP:
$   INQUIRE FILE "File to process"
$   IF F$SEARCH(FILE) .NES. "" [1]
$ THEN
$   FILE_TYPE = F$PARSE(FILE,,,"TYPE") [2] ! determine file type
$   FILE_TYPE = F$EXTRACT(1,F$LENGTH('FILE_TYPE'),FILE_TYPE) ! remove
period
$! Remove type from file specification
$   PERIOD_LOC = F$LOCATE(".",FILE)
$   FILE = F$EXTRACT(0,PERIOD_LOC,FILE)
$   ON WARNING THEN GOTO OTHER
$   GOTO 'FILE_TYPE'
$ ELSE [3]
$   WRITE SYS$OUTPUT FILE, "does not exist"
$ ENDIF [4]
$!
$ GOTO END $!
$!
$!
$ FOR: [5]
$ ON ERROR THEN GOTO PRINT
$ FORTRAN/LIST 'FILE'
$ GOTO LINK
$!
$ PAS:
$   ON ERROR THEN GOTO PRINT
$   PASCAL/LIST 'FILE'
$   GOTO LINK
$!
$ OTHER:
$   WRITE SYS$OUTPUT "Can't handle files of type '.'FILE_TYPE'"
$   GOTO END
$!
$ LINK: [6]
$   ON ERROR THEN GOTO END
```

```

$ WRITE SYS$OUTPUT "Successful compilation ...."
$ LINK 'FILE'
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$ RUN 'FILE'
$ GOTO CLEANUP
$! $ PRINT: [7]
$ WRITE SYS$OUTPUT "Unsuccessful compilation, printing listing file ...."
$ PRINT 'FILE'
$!
$ CLEANUP:
$ DELETE 'FILE'.OBJ;
$ DELETE 'FILE'.LIS;
$!
$ END:
$ INQUIRE/NOPUNCTUATION ANS "Process another file (Y or N)? "
$ IF ANS THEN GOTO TOP
$ EXIT

```

COMPILE_FILE.COM 명령 프로시저 설명

- [1] IF 명령은 F\$SEARCH 렉시칼 함수를 사용하여 디렉토리를 검사하고 파일이 있는지 확인합니다.
- [2] THEN 명령 다음의 명령 블록은 다음을 수행합니다.
 - F\$LENGTH 렉시칼 함수를 사용하여 파일 형식의 길이 결정
 - 파일 형식 결정
 - 파일 형식에서 기간 제거
 - 파일 지정 방법에서 파일 형식을 제거하여 파일 이름 결정
 - 파일 이름에서 기간 제거
 - 오류가 발생할 경우 수행할 작업 정의
 - 기호 FILE_TYPE에 의해 정의된 레이블에서 분기
- [3] "File to process:" 프롬프트에 입력한 파일이 디렉토리에 없으면 ELSE 명령 다음에 이 명령이 실행됩니다.
- [4] ENDIF 명령은 IF-THEN-ELSE 명령 구조를 종료합니다.
- [5] 프로시저는 FORTRAN 프로그램을 컴파일하고 LINK 레이블에서 분기됩니다. 컴파일하는 동안 오류가 발생하면 프로시저가 PRINT 레이블에서 분기됩니다.
- [6] 이 프로시저는 올바르게 컴파일된 프로그램을 표시하고, 프로그램을 링크 및 실행한 다음 CLEANUP 레이블에서 분기됩니다. 오류가 발생하면 프로그램이 END 레이블에서 분기됩니다.
- [7] 프로시저는 기본 인쇄 대기열에 프로그램 목록 파일을 입력합니다.

COMPILE_FILE.COM 명령 프로시저 보기 실행

```
$ @COMPILE_FILE
File to process: RAND.PAS
Successful compilation
%DELETE-I-FILDEL,WORK:[DESCH]RAND.OBJ;1 deleted (3 blocks)
%DELETE-I-FILDEL,WORK:[DESCH]RAND.LIS;1 deleted (9 blocks)
Process another file (Y or N)? N Return
```

주석 명령 프로시저

COMPILE_FILE.COM 명령 프로시저

가

개인 로그인 명령 프로시저 계산 환경을 사용자 정의할 수 있는 명령 프로시저입니다. 명령 프로시저에 포함된 명령은 사용자가 로그인할 때마다 실행됩니다.

개인 볼륨 독점적으로 사용하기 위해 프로세스에서 할당한 대용량 저장 매체입니다.

객체 시스템에서 액세스를 제어하는 정보의 수동 저장소입니다. 객체에 대한 액세스는 객체가 포함하는 정보에 대한 액세스를 의미합니다.

계정 모든 사용자에게는 시스템을 사용할 계정이 있어야 합니다. 계정은 사용자의 사용자 이름으로 식별됩니다. 다른 계정은 시스템에서 다른 수준의 서비스(예: 사용자가 소유한 특권, 사용자가 로그인할 수 있는 시간 등)를 허용합니다.

계층 디렉토리 구조 일대다 관계를 기반으로 트리형 구조로 정렬된 여러 수준이 있는 디렉토리의 구조입니다.

고성능 Sort/Merge 유틸리티 OpenVMS Alpha 시스템에서 사용할 수 있는 Sort/Merge 유틸리티의 버전입니다.

구문 한정자 및 매개 변수의 철자 및 순서를 비롯한 명령의 특정 형식입니다. 잘못 철자한 단어가 일반적인 대부분의 구문 오류입니다.

구문 분석 다음 중 하나입니다.

1. 명령 문자열을 해석하기 위해 해당 요소로 구분하는 것입니다.
2. OpenVMS RMS(레코드 관리 서비스)에서 수행되는 것처럼 파일 지정 방법을 해석하는 것입니다.

구분 기호 문자열, 문 또는 프로그램의 요소를 구분, 종류 또는 구성하는 문자입니다.

글로벌 기호 다음 중 하나입니다.

1. 다른 모듈에서 잠재적으로 참조할 수 있는 프로그램의 모듈에 정의된 기호입니다. 링커는 글로벌 기호를 확인합니다. 참조를 정의와 일치시킵니다. 로컬 기호와 비교하십시오.
2. 모든 명령 수준에서 액세스할 수 있는 명령 언어 기호입니다.

기능 키 특수 신호를 운영 체제로 보내는 키보드 키입니다. 함수 키는 Fn이라고 합니다. 여기서 n은 해당 키와 연결된 숫자입니다. 예를 들어, Mail에서 F9를 누르면 메시지를 전달하려고 한다는 것을 시스템에 알리는 것입니다.

기본 디렉토리 사용자가 디렉토리 지정 방법을 제공하지 않는 경우 OpenVMS 운영 체제에서 사용하는 디렉토리입니다.

기본 디스크 사용자가 만든 모든 파일을 시스템에서 기본적으로 읽고 기록하는 디스크입니다. 명령의 파일 지정 방법이 명시적으로 장치를 명명하지 않은 경우 기본 값이 사용됩니다.

기본 암호 시스템이 사용자에게 요청하는 첫 번째 사용자 암호인 사용자 암호의 한 유형입니다. 시스템에서 보조 암호도 요청하는 경우도 있습니다. 기본 암호는 사용자 이름과 관련된 암호여야 합니다.

기본값 사용자가 지정하지 않는 경우 자동으로 명령에 포함되는 값이나 작업입니다. 대부분의 경우, 기본 설정은 “일반적”이거나 “예상되는” 것입니다.

기호 정의할 때 특정 컨텍스트에서 특정 함수나 엔터티(예: 명령 문자열, 디렉토리 이름 또는 파일 이름)를 나타내는 엔터티입니다.

기호 범위 명령 프로시저 집합으로 이 범위 내에서 기호를 액세스할 수 있습니다.

나

네트워크 교환된 개별 컴퓨터 시스템의 모음입니다.

노드 다음 중 하나입니다.

1. 네트워크의 다른 컴퓨터 시스템과 통신할 수 있는 네트워크의 개별 컴퓨터 시스템입니다.
2. OpenVMS VAX 시스템에서 VAXBI 버스에 있는 16개의 논리 위치 중 하나를 사용하는 VAXBI 인터페이스(예: 중앙 처리 장치, 제어 장치 또는 메모리 하위 시스템)입니다.
3. OpenVMS VAX 시스템에서 SCS(시스템 통신 서비스) 소프트웨어가 인식하는 VAX 프로세서 또는 HSC입니다.

노드 지정 방법 파일 지정 방법의 첫 번째 필드입니다. 이 필드는 네트워크에서 컴퓨터 시스템 위치를 식별합니다.

논리 이름 파일, 디렉토리, 장치 및 대기열과 같은 시스템 객체를 나타내는 데 다른 문자열 대신 사용할 수 있는 사용자 정의 이름입니다. 논리 이름 할당은 논리 이름 표에서 유지됩니다.

논리 이름 표 논리 이름과 해당 등가 문자열 집합을 포함하는 표입니다. 논리 이름은 프로세스-개인이거나 공유 가능합니다. 기본 공유 논리 이름 표는 작업, 그룹, 시스템, 클러스터 전체 시스템 및 클러스터 전체 상위 표입니다.

논리 장치 이름 다소 복잡한 장치 이름을 간단하고 의미 있는 이름과 일치시키는 문자열입니다.

논리식 True 또는 False 값이 있는 식입니다.

다

단순 문자 버전을 제외한 파일 지정 방법의 모든 구성 요소에서 사용할 수 있는 기본 문자 집합입니다.

대기열 다음 중 하나입니다.

1. 일괄 처리 작업 대기열이나 프린터 작업 대기열과 같은 처리해야 할 작업 라인입니다. 처리는 주로 선입선출(FIFO) 순서로 이루어지지만 작업을 제출한 프로세스의 우선 순위를 반영하기도 합니다. 인쇄 대기열을 참조하십시오.
2. INSQUE 명령을 사용하여 목록이나 표에 항목을 추가하기 위한 것입니다.

대용량 저장 장치 데이터 및 기타 파일 유형이 사용되지 않는 동안 저장되는 입력/출력 장치입니다. 일반 저장 장치에는 디스크, 자기 테이프 및 플로피 디스크가 포함됩니다.

대화형 모드 사용자가 명령을 입력하면 시스템에서 이 명령을 실행하고 응답하는 운영 체제와 통신하는 모드입니다. 한 명령이 실행을 끝내야 다른 명령을 입력할 수 있습니다.

데이터 통신, 해석 또는 처리에 적합한 형식으로 사실, 개념 또는 지시 사항 표현에 대한 일반 용어입니다.

도움말 HELP 명령과 함께 사용하기에 적합한 형식의 텍스트 파일입니다. 온라인 도움말은 9개의 검색 수준까지 제공할 수 있습니다.

등가 문자열 논리 이름 표의 논리 이름과 연결된 문자열입니다. 예를 들어, 등가 문자열은 장치 이름, 다른 논리 이름 또는 파일 지정 방법의 한 부분과 연결된 논리 이름일 수 있습니다.

디렉토리 디스크나 테이프에 저장된 파일 집합을 간단하게 카탈로그로 만드는 파일입니다. 디렉토리에는 집합에 있는 각 파일의 이름, 유형 및 버전 번호와 파일의 실제 위치를 식별하고 파일 속성의 목록을 가리키는 고유 번호가 들어 있습니다. 하위 디렉토리를 참조하십시오.

디스크 고속의 임의 액세스 장치입니다. 디스크의 종류는 여러 가지입니다. 플로피 디스크는 작고 다루기 쉬운 디스크입니다. 하드 디스크는 한 위치에 고정되어 있거나 이동 가능합니다. 이동식 디스크 유형에는 보호 케이스 안에 있는 단일 하드 디스크와 보호 케이스 안에 있는 디스크의 누적 집합이 포함됩니다.

라

라인 프린터 파일을 한 번에 한 라인씩 인쇄하는 출력 장치입니다. 다른 경우에는 더 느린 장치를 연결하는 많은 양의 출력을 인쇄하는 데 사용합니다. 거의 모든 시스템에는 라인 프린터로 지정된 장치가 있습니다. 일부 경우 “라인 프린터”가 실제로는 고속 터미널입니다.

레코드 관리 서비스(RMS) 파일과 파일 내의 레코드를 처리하기 위해 프로그램에서 호출하는 운영 체제 프로시저 집합입니다. RMS를 사용하면 프로그램은 레코드 수준(레코드 입출력)과 읽기 및 쓰기 블록(블록 입출력)에서 GET 및 PUT 요청을 생성할 수 있습니다. RMS는 시스템 소프트웨어의 필수적인 부분입니다. RMS 프로시저는 실행 모드에서 실행됩니다.

레코드 정렬 레코드는 그대로 유지하면서 완전한 레코드로 구성된 출력 파일을 만드는 정렬 프로세스입니다.

레코드 지향 장치 터미널, 라인 프린터 또는 카드 판독기 같은 장치입니다. 레코드 지향 장치의 물리적 레코드는 프로그램에서 한 번의 입출력 작업으로 액세스할 수 있는 가장 큰 데이터 단위입니다.

레코드 파일 주소(RFA) 파일에 있는 레코드의 고유 주소입니다. RFA를 사용하면 이전에 액세스한 레코드를 다음에 임의로 액세스할 수 있습니다. 이러한 액세스는 파일 구성에 관계없이 발생합니다.

렉시칼 함수 DCL(DIGITAL Command Language) 명령 인터프리터가 명령 문자열을 구문 분석하기 전에 평가하고 대체하는 명령 언어 구문입니다. 렉시칼 함수는 현재 프로세스(예: 사용자 식별 코드(UIC) 또는 기본 디렉토리)와 문자열(예: 문자열의 길이 또는 하위 문자열의 위치)에 대한 정보를 반환합니다.

로그아웃 DCL(DIGITAL Command Language) 명령 LOGOUT을 입력하는 과정으로 사용자가 특정 터미널을 사용하여 작업을 마치면 운영 체제에 알려 줍니다.

로그인 시스템에 대한 사용자 식별입니다. 사용자가 로그인할 때 시스템의 프롬프트에 응답하여 사용자 이름과 암호를 입력합니다. 사용자 이름과 암호가 시스템의 계정과 일치하면 사용자가 시스템에 액세스할 수 있습니다.

로그인 디렉토리 사용자가 로그인할 때 LOGINOUT에서 설정하는 기본 디렉토리입니다.

로그인 명령 프로시저 로그인 시 및 일괄 처리 작업 시작 시 자동으로 실행되는 명령 프로시저입니다.

로그인 클래스 사용자의 시스템 로그인 방법입니다. 시스템 관리자는 로컬, 전화 접속, 원격, 일괄 처리 또는 네트워크 같은 로그인 클래스에 따라 시스템 액세스를 제어할 수 있습니다.

로컬 기호 다음 중 하나입니다.

1. 기호를 정의한 모듈에서만 의미가 있는 기호입니다. 언어 프로세서에 대해 글로벌 기호로 표시되지 않은 기호를 로컬 기호로 간주합니다. 언어 프로세서는 기호를 확인합니다(참조를 정의와 일치). 로컬 기호는 링커에 알려져 있지 않으므로 다른 객체 모듈에서 사용할 수 있도록 할 수 없습니다. 하지만 링커를 통해 디버거로 전달할 수 있습니다. 글로벌 기호와 비교하십시오.
2. 현재 명령 수준 및 후속 호출 수준에서만 액세스할 수 있는 명령 언어 기호 이름입니다. 로컬 기호가 정의되어 있는 명령 수준이 종료되면 삭제됩니다.

로컬 노드 사용자가 실제로 있는 네트워크 노드입니다.

마

마스터 파일 디렉토리(MFD) 디스크의 주 디렉토리를 포함하는 파일입니다.

매개 변수 다음 중 하나입니다.

1. 범위가 P1에서 P8까지인 기호와 일치하는 명령 프로시저에 전달된 값입니다. 명령 매개 변수를 참조하십시오.

2. 네트워크 관리 구성 요소에 대해 휘발성 또는 영구 데이터베이스에 있는 항목입니다.

메모리 이진 단어 형식으로 데이터나 지시 사항이 있을 수 있는 일련의 물리적 위치입니다. 메모리의 각 위치를 지정할 수 있고 해당 내용을 변경할 수 있습니다. 메모리를 대용량 저장 장치와 혼동하지 마십시오.

명령 DCL(DIGITAL Command Language)에서 사용자가 터미널에서 입력하거나 명령 프로시저에 포함된 지시 사항입니다. 일반적으로 영어 단어입니다. 명령은 터미널을 모니터링하거나 명령 프로시저를 읽는 소프트웨어가 몇 가지 잘 정의된 작업을 수행하도록 요청합니다. 예를 들어, COPY 명령을 입력하면 시스템이 한 파일의 내용을 다른 파일로 복사하도록 요청합니다.

명령 매개 변수 파일 지정 방법, 옵션 또는 상수와 같이 공백으로 구분되는 명령의 위치 피연산자입니다.

명령 문자열 명령 및 명령을 수정하는 정보(옵션)를 포함하는 행(또는 연속된 줄의 집합)입니다. 명령 문자열은 명령, 명령 한정자, 명령 매개 변수(예: 파일 지정 방법) 및 매개 변수 한정자로 구성됩니다. 일반적으로 Return 키를 누르면 명령 문자열이 종료됩니다.

명령 수준 명령 인터프리터의 입력 스트림입니다. 초기 입력 스트림은 항상 명령 수준 0입니다. 대화형 명령 프로시저는 명령 수준 1에서 실행됩니다. 일괄 처리 작업 명령 프로시저는 명령 수준 0에서 실행됩니다. 명령 프로시저에서 실행 프로시저(@) 명령이나 CALL 명령을 사용하여 중첩된 명령 수준을 32개까지 만들 수 있습니다.

명령 이미지 DCL 명령과 연결되고 호출되는 프로그램입니다.

명령 인터프리터 사용자가 터미널에서 입력하거나 명령 파일로 제출한 명령을 받고, 구문을 검사하고 명령을 구문 분석하는 프로세스의 컨텍스트에서 감독자 모드로 실행되는 프로시저 기반 시스템 코드입니다.

명령 프로시저 명령 인터프리터에서 사용할 수 있는 명령과 데이터가 들어 있는 파일입니다. 명령 프로시저는 자동으로 명령을 운영 체제로 전달하는 방법을 제공하므로 사용자가 터미널에서 이러한 명령을 수동으로 입력할 필요가 없습니다. 또한 명령 프로시저는 사용자의 조작을 통해 변경할 수 있는 복잡한 명령 순서를 설정하도록 루프, 카운터, 레이블 및 기호 대체와 같은 프로그래밍 방법을 사용하도록 허용합니다. 명령 프로시저는 일괄 처리 작업으로 처리되도록 시스템에 전송될 수도 있습니다.

문자열(character string) 인쇄 가능한 문자의 연속 집합입니다.

문자열(string) 연결된 문자의 순서입니다. 텍스트 편집기가 텍스트 파일에서 단어나 구를 검색할 때 문자열을 찾습니다. 명령을 이루는 문자열 순서를 명령 문자열이라고 하기도 합니다.

물리적 장치 이름 시스템에 대한 물리적 장치(예: 저장 디스크 또는 터미널)를 고유하게 확인하는 문자열입니다.

바

반복적 변환 논리 이름 정의에 다른 논리 이름이 포함된 경우 발생하는 논리 이름의 반복적 변환입니다.

버전 번호 파일 지정 방법의 숫자 구성 요소입니다. 파일을 편집하면 버전 번호가 하나씩 증가합니다.

버퍼 입력이나 출력 작업 중에 데이터 레코드의 임시 저장소로 사용되는 내부 메모리 영역입니다.

보안 터미널 서버 이미 로그아웃한 터미널로만 사용자가 로그인할 수 있게 설계된 OpenVMS 소프트웨어입니다. 사용자 터미널에서 **Break** 키를 누르면 보안 터미널 서버(사용 가능한 경우)가 로그인 프로세스 연결을 먼저 끊은 다음 로그인을 시작하여 응답합니다. 터미널에서 로그인된 프로세스가 없는 경우 로그인이 즉시 진행될 수 있습니다.

보조 암호 로그인 시 기본 암호가 올바르게 전송된 다음 바로 입력해야 하는 사용자 암호입니다. 로그인 시 두 명 이상의 사용자가 있는지 확인하기 위해 기본 암호와 보조 암호를 개별 사용자에게 알릴 수 있습니다. 자주 사용되는 방법은 아니지만 암호 길이를 늘리기 위한 수단으로 보조 암호를 요구하는 경우가 있습니다. 이러한 총 문자 조합 수는 암호 추측을 어렵게 하고 시간이 많이 소모되게 합니다.

보호 객체 시스템에서 액세스를 제어하는 공유 정보를 포함하는 객체입니다. 객체를 참조하십시오.

보호 코드 다른 범주의 시스템 사용자가 파일이나 다른 보호 객체에 대해 가질 수 있는 액세스 유형 그리고 액세스한 경우 수행할 수 있는 작업을 지정하는 일련의 문자입니다.

복합 문자 단순 문자와 확장 문자 집합 문자와의 조합입니다.

볼륨 디스크 팩이나 자기 테이프 릴과 같은 대용량 저장 매체입니다. 볼륨은 파일 구조에서 가장 큰 논리 단위입니다.

볼륨 집합 한 개 이상의 대용량 저장 매체에 있는 파일 구조의 데이터 모음입니다.

분리된 프로세스 소유자가 없는 프로세스입니다. 사용자가 시스템에 로그인하면 작업 제어 장치는 분리된 프로세스를 만듭니다. 또한 일괄 처리 작업을 시작하거나 논리 링크 연결에 대한 요청을 처리할 때마다 분리된 프로세스를 만듭니다. 작업 제어 장치는 자신이 만든 프로세스를 소유하지 않으므로 이러한 프로세스는 분리된 것으로 여겨집니다. DCL 명령 RUN/UIC와 Create Process 시스템 서비스(UIC 지정)를 사용하면 적절한 특권이 있는 프로세스에서 분리된 프로세스 만들기를 요청할 수 있습니다.

비디오 터미널 운영 체제와 사용자의 상호 작용을 표시하는 키보드 및 비디오 화면 또는 모니터입니다. 터미널을 참조하십시오.

사

사용 제어 목록(ACL) 보호된 시스템 객체에 대해 사용자나 그룹이 갖고 있는 액세스 권한을 정의하는 항목 모음입니다.

사용 제어 항목(ACE) 사용 제어 목록의 항목입니다. 사용 제어 항목은 ID 소유자, 디렉토리에 대한 기본 보호 또는 보안 경보 세부 사항에 부여되거나 거부되는 ID와 액세스 권한을 지정합니다.

사용자 식별 코드(UIC) 사용자, 파일, 글로벌 섹션, 명령 이벤트 플러그 클러스터 및 우편함에 할당된 숫자 쌍입니다. UIC는 owner, group, world 및 system에서 사용할 수 있는 액세스 유형(읽기, 쓰기 또는 읽기/쓰기)을 지정합니다.

사용자 암호 사용자와 연결된 암호입니다. 사용자가 시스템에 액세스할 수 있도록 승인을 받으려면 로그인을 시도할 때 이 암호를 제대로 입력해야 합니다. 사용자 암호의 두 가지 유형은 주 암호와 보조 암호이며 이 용어는 암호가 입력되는 순서를 나타내기도 합니다.

사용자 인증 파일(UAF) 시스템의 각 계정에 대한 세부 사항을 보관하는 파일입니다. UAF에는 각 계정에 할당된 사용자 이름, 암호, 사용자 식별 코드(UIC), 할당량, 한계 및 특권이 들어 있습니다.

사용자 파일 디렉토리(UFD) 디스크나 테이프에 저장된 파일 집합을 간단하게 카탈로그로 만드는 파일입니다. UFD에는 해당 집합에 있는 각 파일의 이름, 유형, 버전 번호가 들어 있습니다. 또한 파일의 실제 위치를 식별하고 파일 속성 목록을 가리키는 고유한 숫자가 들어 있습니다. 디렉토리를 참조하십시오.

상대적 파일 구성 각 레코드가 버킷 내에서 같은 길이의 셀을 사용하는 파일의 레코드 배열입니다. 각 셀에는 연속된 숫자가 할당됩니다. 이 숫자는 파일의 시작 부분에 대해 상대적인 위치를 나타냅니다.

서브 루틴 다른 프로그램에서 호출할 때 실행되는 보조 루틴입니다. 특정 조건이 만족될 때까지 서브 루틴이 반복하여 호출되곤 합니다.

소프트웨어 특정 컴퓨터 시스템 작업과 연결된 이미지, 프로시저, 규칙 및 문서의 모음입니다. 예를 들어, 운영 체제는 소프트웨어입니다.

순차적 액세스 모드 프로그램이 파일의 임의 지점에서 시작하고 끝내면서 레코드가 표시되는 순서대로 레코드를 하나씩 읽거나 쓰는 레코드 검색 또는 저장입니다.

순차적 파일 구성 레코드가 원래 쓰여진 순서대로 표시되는 파일 구성입니다. 레코드는 고정 길이 또는 가변 길이일 수 있습니다. 레코드 주소를 사용하여 레코드를 순차적으로 또는 임의로 액세스할 수 있습니다. 상대적 레코드 번호를 사용하여 고정 길이 레코드를 임의로 액세스할 수도 있습니다.

숫자식 산술 연산자로 연결된 피연산자 모음으로 구성된 수치 문입니다.

스크롤 수직으로 이동하여 두 개 이상의 텍스트 화면을 표시할 수 있게 하는 비디오 터미널의 기능입니다. 예를 들어, **TYPE** 명령을 입력하면 맨 위에 있는 가장 오래된 출력이 사라지면서 화면의 맨 아래에 새 출력이 나타납니다.

시간 종료 장치에서 입출력 전송을 완료해야 할 시간 제한이 만료된 것입니다.

시스템 관리자 사용자가 리소스를 사용할 수 있게 하고 이러한 리소스 사용에 관련된 제한을 설정하는 사람입니다.

시스템 로그인 명령 프로시저 사용자가 로그인할 때 특정 명령이 항상 실행되는지 시스템 관리자가 확인할 수 있게 해주는 프로시저입니다.

시스템 암호 로그인을 시작할 수 있기 전에 터미널에서 요구하는 암호입니다.

시작 위치 한정자 EVE에서 사용자가 지정한 버퍼에 커서가 처음으로 표시되는 열과 행을 확인하기 위해 사용할 수 있는 한정자입니다.

식 컴퓨터가 값을 구해 결과를 산출할 수 있도록 변수, 상수 또는 두 가지 모두를 연산자와 조합한 것입니다.

실행 가능 이미지 프로세스에서 실행할 수 있는 이미지입니다. 실행 시 실행 가능 이미지를 프로세스의 실행 파일로부터 읽습니다.

쓰기 데이터를 보내는 이미지의 작업 또는 기능입니다. 예를 들어, **PRINT** 명령이 실행되면 지정한 파일이 저장되고 프린터로 쓰여질 때마다 이 파일을 읽습니다. 읽기를 참조하십시오.

아

암호 ID를 확인하고 인증 증명 형식으로 사용자 계정에 액세스하기 위해 사용자가 로그인 시 입력하는 문자열입니다. 암호는 시스템 암호 및 사용자 암호라는 두 가지 유형이 있습니다. 사용자 암호는 기본 암호와 보조 암호 모두를 포함합니다.

액세스 제어 문자열 원격 노드로 보낼 로그인 정보를 포함하는 0-42자입니다. **OpenVMS** 시스템에서 액세스 제어 문자열은 일반적으로 사용자 이름, 공백이나 탭 및 암호로 구성됩니다.

역방향 비디오 기본 비디오 대비를 역방향으로 바꾸는 비디오 터미널의 기능입니다. 기본 표시가 흰색 배경에 검정 그림이었다면 역방향 비디오는 검정 배경에 흰색 그림으로 표시합니다.

연결 연속하여 파일을 연결하는 작업입니다.

연산자 컴퓨터에게 피연산자 조작 방법을 알려 주는 표현식의 부분입니다. 예를 들어, 더하기 기호(+)는 컴퓨터에게 더하기를 수행하라고 알려 주는 연산자입니다.

열린 계정 암호가 필요하지 않은 계정입니다.

오류 메시지 사용자가 요청한 일부 작업이 실패한 경우 시스템에서 보내는 메시지입니다. 각 오류 메시지는 오류를 탐지했던 운영 체제의 특정 부분을 식별합니다. 대부분의 오류 메시지는 입력 실수나 구문 지정 실수로 인해 발생합니다. 명령을 제대로 다시 입력하면 오류를 수정할 수 있습니다.

와일드카드 문자 주어진 필드에 대해 “모두” 표시하기 위해 파일 지정 방법의 파일 이름, 파일 유형, 디렉토리 이름 또는 버전 번호 내에서 사용되거나 이를 대신하여 사용되는 별표(*) 또는 백분율 기호(%)와 같은 영숫자가 아닌 문자입니다.

외부 명령 명령 인터프리터가 이미지 이름을 DCL 명령으로 인식하지 못하는 이미지를 실행하는 기호입니다.

외부 지정 방법 파일 지정 방법 파일이 OpenVMS 구분이나 형식에 맞지 않는 파일입니다.

우선 순위 프로세스가 실행 중인 경우 시스템 리소스를 얻는 우선권을 결정하기 위해 프로세스에 할당된 순위입니다.

운영 체제 컴퓨터 프로그램 실행을 제어하고 시스템 기능을 수행하는 통합 프로그램 모음입니다.

원격 노드 사용자가 현재 로그인한 노드가 아닌 네트워크의 노드입니다.

유틸리티 프로그램 개발 유틸리티(편집기, 링커), 파일 관리 유틸리티(파일 복사 또는 파일 형식 변환 프로그램) 또는 작업 관리 유틸리티(디스크 할당량, 진단 프로그램)와 같은 관련된 일반 용도의 함수 집합을 제공하는 프로그램입니다.

이미지 실행 가능 프로그램을 구성하기 위해 링커에서 함께 결합한 프로시저 및 데이터입니다. 이 실행 가능 프로그램은 프로세스에서 실행합니다. 이미지 유형에는 실행, 공유 및 시스템 등 세 가지가 있습니다.

인덱스 순차 파일 각 레코드에 포함된 데이터 키가 한 개 이상 있는 레코드 파일입니다. 파일의 레코드 각각은 레코드와 연결된 키를 지정하여 액세스할 수 있습니다.

인쇄 대기열 인쇄를 기다리는 파일의 목록입니다.

인쇄 양식 인쇄하기 위해 페이지 설정 및 스톱을 정의하는 속성 집합입니다.

일괄 처리 작업 일괄 처리 하위 시스템의 제어 하에 예약되고 실행되는 프로그램입니다. 일괄 처리 작업의 제어 입력을 디스크에 저장된 명령 프로시저에서 가져오며 출력은 디스크 파일로 지정됩니다.

일반 장치 이름 특정 장치가 아닌 장치 유형을 식별하는 장치 이름입니다. 특정 제어 장치나 장치 번호가 생략된 장치 이름입니다.

읽기 데이터를 받아들이는 이미지의 동작이나 기능입니다. 예를 들어, **TYPE** 명령이 실행되면 시스템은 디스크에서 지정된 파일을 읽어 터미널에 씁니다. 쓰기를 참조하십시오.

임의 액세스 검색하거나 쓸 데이터의 위치가 이전에 검색했거나 쓴 데이터의 위치와 독립적인 데이터를 검색하거나 쓰는 방법입니다. 임의 액세스는 모든 정보를 동등하게 액세스할 수 있는 메모리나 대용량 저장 장치와 관련됩니다.

입력 스트림 명령 및 날짜의 소스입니다. 사용자 터미널이나 일괄 처리 스트림, 명령 프로시저가 될 수 있습니다.

입력 파일 컴퓨터로 전송할 데이터를 포함하는 파일입니다. 입력 파일과 출력 파일이 혼동되는 경우가 있습니다. DCL에서는 대개 이러한 파일에 대한 메시지를 표시하지만 대부분의 시스템 유틸리티의 경우에는 명령행의 위치로 사용자 입력 파일과 출력 파일을 식별해야 합니다. 사용하고 있는 명령의 구문이나 형식을 확인해야 합니다.

자

자기 테이프 데이터를 저장하고 액세스할 수 있는 매체입니다.

작업 프로세스 및 하위 프로세스(있을 경우) 및 이 하위 프로세스에서 만드는 모든 하위 프로세스에 해당하는 계정 단위입니다. 작업은 일괄 처리 및 대화형으로 분류됩니다. 예를 들어, 작업 제어 장치는 사용자가 시스템에 로그인하는 경우 대화형 작업을 만들어 사용자 요청을 처리하고 심비언트 관리자가 명령 입력 파일을 시스템에 전달하는 경우 일괄 처리 작업을 만듭니다.

작업 트리 주 프로세스가 맨 위에 있는 모든 프로세스와 하위 프로세스의 계층입니다.

장치 데이터 받기, 저장 또는 전송이 가능한 프로세서에 연결된 주변 장치의 일반적 이름입니다. 카드 판독기, 라인 프린터 및 터미널은 레코드 지향 장치의 예입니다. 자기 테이프 장치 및 디스크 장치는 대용량 저장 장치의 예입니다. 터미널 회선 인터페이스 및 인터프로세서 링크는 통신 장치의 예입니다. 장치가 항상 하드웨어인 것은 아닙니다.

장치 이름 파일이 저장된 장치 단위를 식별하는 파일 지정 방법의 필드입니다. 또한 장치 이름에는 데이터 전송 요청에서 입출력 주변 장치를 식별하는 니모닉이 포함됩니다.

니다. 장치 이름은 니모닉, 제어 장치 ID 문자(해당 사항이 있을 경우), 장치 번호(해당 사항이 있을 경우) 및 콜론 순으로 구성됩니다.

전용 계정 사용자 활동을 제한하는 OpenVMS 계정의 한 유형입니다. 일반적으로 사용자는 특정 명령 프로시저와 명령을 사용하도록 제한됩니다. 예를 들어, 사용자가 Ctrl/Y 키 시퀀스를 사용하는 것이 허용되지 않을 수 있습니다. 이 계정 유형은 토큰(일괄 공급)나 제한된 계정의 동의어입니다.

전체 이름 DECdns 네임스페이스에 있는 이름에 대한 완전한 지정 방법으로 루트 디렉토리에서 이름이 지정된 객체, 디렉토리 또는 소프트웨어 링크에 이르기까지 경로의 모든 상위 디렉토리가 포함됩니다. 또한 네임스페이스 이름도 포함할 수 있지만 네트워크에 한 네임스페이스만 있는 경우에는 필요하지 않습니다.

제한 계정 보안 로그인 프로시저가 있는 OpenVMS 계정의 유형입니다. 사용자는 시스템 또는 프로세스 로그인 명령 프로시저 중에 Ctrl/Y 키 시퀀스를 사용할 수 없습니다. 로그인 명령 프로시저의 실행에 따라 이러한 제어가 사용자에게로 넘어갈 수 있습니다.

조합 순서 차례로 나열하기 위해 사용하는 문자 집합(예: ASCII, 다국적 또는 EBCDIC)의 문자에 할당된 순서입니다.

중앙 처리 장치(CPU) 실행 프로그램과 입력 및 출력의 모든 계산 및 라우팅을 처리하는 하드웨어입니다. 즉, CPU는 실제로 계산하는 컴퓨터의 부분입니다.

증명서 권한 데이터베이스에 기록되고 액세스 요청 확인에서 시스템이 사용하는 사용자나 사용자 그룹을 나타내는 영숫자 문자열입니다. 증명서 유형은 환경, 기능, 일반 및 사용자 식별 코드(UIC) 등 네 가지입니다.

지정 방법 파일 정렬 작업을 완료하는 데 필요한 명령 및 한정자를 지정하기 위해 **Sort/Merge** 유틸리티에서 사용하는 명령 파일입니다.

차

최적 배달 데이터 배달을 시도하지만 라인 오류와 같은 오류가 있는 경우 복구를 시도하지 않는 네트워크 프로토콜입니다.

출력 파일 정렬되거나 편집된 파일처럼 처리 작업의 결과가 들어 있는 파일입니다.

침입 시도 시스템에 액세스 하기 위해 권한이 없는 소스에 의해 이루어지는 시도입니다. 첫 번째 시스템 액세스는 로그인을 통해 이루어졌으므로 침입 시도는 대개 불법 로그인 시도를 의미합니다. 이러한 시도는 주로 정보를 근거로 추측하거나 기타 시행 착오 방법을 통해 시스템에 계정이 있는 것으로 알려진 사용자의 암호를 입력하는 것입니다.

카

커서 화면의 위치를 가리키기 위해 모니터 화면에서 사용하는 표시기입니다.

키 다음 중 하나입니다.

1. 인덱스 파일의 경우 인덱스 파일에서 각 데이터 레코드 내에 있는 문자열, 팩트형 십진수, 2바이트나 4바이트 부호 없는 이진수 또는 2바이트나 4바이트 부호형 이진수입니다. 사용자가 레코드 내에서 길이와 위치를 정의합니다. **OpenVMS RMS**(레코드 관리 서비스)는 인덱스를 구성하는 데 키를 사용합니다.

2. 상대적 파일의 경우 데이터 파일에 있는 각 데이터 레코드의 상대적 레코드 번호입니다. **OpenVMS RMS**(레코드 관리 서비스)는 임의 액세스 모드에서 상대적 파일의 데이터 레코드를 식별하고 액세스하는 데 상대적 레코드 번호를 사용합니다.

3. **Sort/Merge** 유틸리티의 경우 사용자가 레코드를 정렬하기 위한 기준으로 사용하는 정보를 포함하는 레코드의 데이터 필드입니다.

키보드 타자기와 유사하게 작동되는 입력 장치입니다.

키워드 주로 명령 문자열이나 문 등 특정 지정 구문 형식에서 사용하도록 예약한 단어입니다.

키패드 터미널의 주 키보드 옆에 있는 키의 작은 집합입니다.

타

터미널 키보드와 비디오 화면 또는 프린터가 있는 주변 장치의 일반적 이름입니다. 프로그램 제어 하에 터미널을 사용하여 사용자는 키보드로 명령과 데이터를 입력하고 비디오 화면이나 프린터에서 메시지를 받을 수 있습니다.

파

파일 사용자에게 중요한 구조로 정렬되어 있는 데이터 요소 집합입니다. 파일은 명명되고 저장된 프로그램이나 데이터 또는 두 가지 모두로 시스템에서 액세스할 수 있습니다. 액세스 유형에는 읽기 전용과 읽기/쓰기 두 가지가 있습니다. 읽기 전용은 파일을 변경할 수 없다는 의미이고 읽기/쓰기는 파일의 내용을 수정할 수 있다는 의미입니다. 볼륨을 참조하십시오.

파일 경로 파일 지정 방법의 디스크 및 디렉토리 부분입니다.

파일 유형 마침표 다음에 0-39자 ID로 구성되는 파일 지정 방법의 필드입니다. 규칙에 따라 이 필드는 컴파일러 및 어셈블러 목록 파일, 이진 객체 파일 등 같은 용도나 특성을 갖는 파일의 일반 클래스를 식별합니다.

파일 이름 파일 지정 방법에서 파일 유형 앞에 오는 것으로 파일에 대한 1-39자의 이름을 포함하는 필드입니다.

파일 지정 방법 대용량 저장 매체에 있는 파일에 대한 고유한 이름입니다. 파일이 저장된 노드, 장치, 디렉토리 이름, 파일 이름, 파일 유형 및 버전 번호를 식별합니다.

편집기 컴퓨터 파일에서 텍스트를 만들거나 수정하는데 사용되는 프로그램입니다.

폴더 메일 메시지를 저장할 수 있는 파일의 하위 부분입니다.

프로그램 특정 결과를 목표로 하는 일련의 지시 사항입니다. 프로그래밍 언어는 컴퓨터에서 프로시저를 수행할 수 있도록 프로시저를 설명하는 수단입니다. 이미지를 참조하십시오.

프로그램 스텝 명령 프로시저를 작성하는 테스트 단계에서 사용되는 코드의 임시 섹션입니다. 프로그램 스텝은 일반적으로 대체 중인 프로시저를 알려주는 메시지를 출력합니다.

프로세스 시스템 소프트웨어에서 예약한 기본 엔터티입니다. 프로세스는 이미지가 실행될 컨텍스트를 제공합니다. 프로세스는 주소 공간과 하드웨어 및 소프트웨어 컨텍스트로 구성됩니다.

프로세스 기본 디렉토리 시스템에서는 사용자가 로그인하면 자동으로 최상위 디렉토리를 프로세스 기본 디렉토리로 만듭니다.

프로시 로그인 사용자가 로컬 노드의 계정을 소유한 것처럼 원격 노드의 사용자가 효과적으로 로컬 노드에 로그인할 수 있게 해주는 로그인 유형입니다. 하지만 사용자가 액세스 제어 문자열에 암호를 지정하지 않습니다. 원격 사용자는 계정을 소유하거나 다른 사용자와 계정을 공유할 수 있습니다.

프롬프트 사용자가 입력해야 한다는 것을 알려 주는 문자열로 터미널 화면에 나타납니다.

피연산자 값이 들어 있는 표현식의 부분입니다. 결과를 산출하기 위해 표현식 계산 중에 연산자에 의해 피연산자가 작동합니다.

필드 논리 레코드에 있는 연속 바이트 집합입니다.

하

하드웨어 장치 라인 프린터, 터미널, 대용량 저장 장치와 같은 기계 장치를 비롯한 물리적 컴퓨터 장비입니다.

하드카피 터미널 출력을 용지에 인쇄하는 터미널입니다. 터미널을 참조하십시오.

하위 디렉토리 디렉토리의 소유자에게 속한 추가 파일을 나열하는 디렉토리 파일로 더 높은 수준 디렉토리에 서 카탈로그로 만들어집니다.

하위 프로세스 다른 프로세스에서 만든 보조 프로세스입니다. 하위 프로세스를 만든 프로세스가 소유자입니다. 프로세스와 그 하위 프로세스는 할당량 및 한계 풀을 공유합니다. 시스템에서 소유자 프로세스가 제거되면 모든 해당 하위 프로세스 및 그 하위 프로세스도 제거됩니다.

한정자 여러 가지 옵션 중 하나를 선택하여 명령 동사나 명령 매개 변수를 수정하는 명령 문자열의 일부입니다. 있을 경우 한정자는 자신이 적용될 명령 동사나 매개 변수 다음에 /qualifier[=option] 형식으로 옵니다. 예를 들어, 명령 문자열 “PRINT filename /COPIES=3”에서 COPIES 한정자는 사용자가 주어진 파일의 사본 3부를 인쇄하려고 한다는 것을 표시합니다.

할당문 DCL에서 문자열이나 숫자 값과 기호 이름의 연결입니다. 기호는 시스템 명령의 동의어를 정의할 수 있으며 명령 프로시저의 변수로 사용될 수 있습니다.

행 편집기 행 단위로 파일에 추가하고 삭제할 수 있게 하는 프로그램입니다.

호스트 네트워크에 연결된 시스템입니다. 노드를 참조하십시오.

확장 파일 지정 방법 OpenVMS에서 이전에 부여한 디렉토리 및 파일 명명 제한 중 많은 부분을 제거하는 옵션 기능입니다. 깊은 수준의 디렉토리 및 확장 파일 이름을 허용합니다.

A

American Standard Code for Information

Interchange 텍스트 표시 및 통신 프로토콜에서 사용되는 알파벳, 구두점, 숫자 및 기타 특수 기호를 나타내는 8비트 이진수 집합입니다.

ASCII American Standard Code for Information Interchange를 참조하십시오.

C

CPU 중앙 처리 장치(CPU)를 참조하십시오.

D

DCL(DIGITAL Command Language) 사용자와 운영 체제 간의 통신 수단을 제공하는 OpenVMS 시스템의 명령 인터프리터입니다.

DECnet-Plus OSI와 DNA 프로토콜을 통합하는 DNA(디지털 네트워크 아키텍처) Phase V를 구현하는 Compaq 하드웨어 및 소프트웨어 제품군입니다.

DECnet-Plus는 OSI를 따르며 DECnet Phase IV 및 TCP/IP와 호환됩니다.

DIGITAL Command Language(DCL)

DCL(DIGITAL Command Language)을 참조하십시오.

M

MIME(Multipurpose Internet Mail Extension)

비텍스트 파일을 메일 메시지에 첨부하는 데 사용되는 표준입니다. 그래픽이나 사운드 파일과 같은 비텍스트 파일은 인코딩되어 일반 텍스트로 전송됩니다. 하지만 이 텍스트는 읽을 수 없습니다. 수신인은 **MIME** 인터프리터 유틸리티를 사용하여 파일의 원래 형식으로 텍스트를 디코딩할 수 있습니다.

N

null 값 두 개의 인용 부호(“)로 명령 프롬프트에 표시되는 문자가 없는 문자열입니다.

R

RMS 레코드 관리 서비스(RMS)를 참조하십시오.

T

time-stamp 데이터와 날짜를 정확하게 지정하는 텍스트 문자열입니다. 예를 들면, 11-DEC-1996 17:13:21입니다.

U

UAF 사용자 인증 파일(UAF)을 참조하십시오.

기호

\$CREPRC 시스템 서비스, 443
 \$GETDVI 렉시칼 함수, 646
 \$RESTART 기호
 명령 프로시저에서 사용, 530
 .DDIF 파일
 Mail, 216
 .DIS 파일 유형
 우편 그룹, 214

숫자

8진수
 UIC 디렉토리 지정 방법, 160

A

ACE(엑세스 제어 항목), 404
 ACL(엑세스 제어 목록)
 기본 보호, 396
 ALLOCATE 명령, 190
 형식, 193
 ASCII, 125
 ASCII 문자 집합, 166, 686
 ASSIGN 명령, 409
 엑세스 모드, 412
 ATTACH 명령, 324
 AUTO_PURGE 명령
 Mail, 236

B

BATCH\$RESTART 기호
 명령 프로시저에서 사용, 530
 BATCH.COM 명령 프로시저, 736
 보기 실행, 740
 BUFFER 명령, 320

C

CALC.COM 명령 프로시저, 734
 보기 실행, 735
 CC_PROMPT 명령, 239
 CDU(Command Definition Utility), 558
 CLEANUP.COM 명령 프로시저, 524
 CLOSE 명령
 명령 프로시저
 논리 이름 삭제, 410
 COMPILE_FILE.COM 명령 프로시저, 741
 보기 실행, 743
 CONNECT 명령
 /CONTINUE 한정자, 669
 CONTINUE 명령, 664

CONVERT.COM 명령 프로시저, 703
 보기 실행, 707
 COPY 명령, 126
 /SINCE 한정자, 126
 Mail, 223
 CREATE 명령, 126
 CREATE/DIRECTORY 명령, 152, 157
 CREATE/NAME TABLE 명령
 /PROTECTION 한정자, 437
 CREATE/PROTECTION 명령, 133
 Ctrl 키
 일반, 105
 Ctrl/A 키 시퀀스, 105
 Ctrl/B 키 시퀀스, 100, 398
 명령 다시 호출, 100, 105
 Ctrl/C 키 시퀀스, 105
 Mail 메시지 취소, 208, 216
 Ctrl/E 키 시퀀스, 105
 Ctrl/T 키 시퀀스
 DCL 명령 중단, 105
 프로세스 상태 확인, 71
 Ctrl/U 키 시퀀스, 103
 Ctrl/W 키 시퀀스
 화면 표시 새로 고침, 664
 Ctrl/Y 키 시퀀스
 DCL 명령 중단, 105
 DCL 명령 취소, 105
 PIPE 명령 중단, 628
 동작 루틴 설정, 543
 명령 프로시저 중단, 541
 명령 프로시저 취소, 541
 비활성화, 547
 원격 세션 중단, 76
 이미지 중단, 664
 입력 효과, 543
 종료, 542
 활성화, 547
 Ctrl/Z 키 시퀀스
 Mail 메시지 보내기, 208
 Mail에서 파일 보내기, 216
 파일 끝 종료자, 105

D

DBG\$INPUT 논리 이름, 435
 DBG\$OUTPUT 논리 이름, 435
 DCL 명령
 null 문자 사용, 90
 PIPE 명령, 89
 간략화, 90
 명령 프로시저, 90
 간략화된 한정자, 92

색인

- 구문, 86, 87
- 구성, 86
- 기본 한정자, 92
- 기본값, 88
- 다시 호출
 - 아래쪽 화살표 사용, 105
- 대/소문자 구분, 84
- 대문자, 90
- 동사, 86
- 매개 변수, 87
- 매개 변수 지정, 91
- 매개 변수 한정자, 93
- 명령 인터프리터에서 수행, 576
- 명령 프로시저에 포함, 505
- 명령 프로시저에서의 순서, 512
- 명령 한정자, 92
- 명령행 편집, 102
- 소문자, 90
- 여러 행 명령, 89
- 여러 행 입력, 89
- 위치 한정자, 92
- 입력, 84
- 입력 규칙, 90
- 자동 외부 명령, 500
 - 제한 사항, 502
- 중단, 105
- 최대 요소 수, 90
- 출력 재지정, 575
- 충돌하는 한정자, 93
- 취소, 105
- 프롬프트, 86
- 필요한 공백, 90
- 필요한 문장 부호, 90
- DCL\$PATH 논리 이름, 499, 500
- DCL(DIGITAL Command Language)
 - 정의, 82
- DEALLOCATE 명령, 193
- DEASSIGN 명령, 412, 416, 442, 446
- DEC 다국적 문자 집합, 686
- DECnet
 - 원격 시스템 연결 손실, 76
 - 전송 지정, 209
 - 전체 노드 이름 지정, 117
 - 파일 조작, 127
- DECTPU(DEC Text Processing Utility), 251
- DEFINE 명령, 409, 415
 - /USER_MODE 한정자, 449
 - Mail의 /KEY 한정자, 233, 239
 - 초기화 파일, 234
 - 논리 이름, 409
 - 액세스 모드, 412
- DEFINE/KEY 명령, 104
- DELETE BUFFER 명령, 317
- DELETE 명령, 131
 - /ENTRY 한정자, 681
 - F\$SEARCH 렉시컬 함수를 사용, 647
 - Mail, 226
- DELETE/SYMBOL 명령
 - /GLOBAL 한정자, 459
- DIR.COM 명령 프로시저, 712
 - 보기 실행, 714
- DIRECTORY 명령, 153
 - Mail, 206, 224
- DISMOUNT 명령, 198
 - /NOUNLOAD 한정자, 198
- E**
- EDIT 명령
 - /EDT 한정자
 - /READ_ONLY 한정자, 129
 - /RECOVER 한정자, 306
 - /TPU 한정자
 - /READ_ONLY 한정자, 129
 - EVE를 실행, 258
- EDITALL.COM 명령 프로시저, 721
 - 보기 실행, 723
- Enter 키, 105
- EVE
 - EDIT 명령행 한정자, 299
 - /READ_ONLY 또는 /NOWRITE 무시, 301
 - 버퍼 수정용, 301
 - 시작 위치용, 300
 - 작업 파일용, 300, 301
 - EVE 및 DCL 간에 전환, 324
 - Mail에서 사용, 229
 - 겹쳐쓰기 모드, 273
 - 상자 편집, 290
 - 기능 요약, 254
 - 기본 편집기, 254
 - Mail, 230
 - 도움말, 256
 - HELP 명령, 257
 - 키패드를 사용하여 액세스, 256
- 명령
 - BUFFER, 320
 - DELETE BUFFER, 317
 - EXIT, 263
 - FIND, 292
 - GET FILE, 319
 - GOTO, 296
 - HELP, 257

-
- INCLUDE FILE, 320
 - MARK, 296
 - OPEN SELECTED, 319
 - QUIT, 263
 - RECOVER BUFFER, 306
 - REPLACE, 296, 298
 - RESTORE BOX SELECTION, 289
 - RESTORE SELECTION, 289
 - SET BUFFER, 318
 - SET FIND CASE EXACT, 293, 297
 - SET PENDING DELETE, 289
 - SHOW BUFFERS, 319
 - SPAWN, 324, 325
 - SPLIT WINDOW, 323
 - WRITE FILE, 263, 320
 - 텍스트 형식화, 310
 - 형식화, 310
 - 명령 입력, 260
 - 미리 정의된 키 사용, 260
 - 정의된 키 사용, 260
 - 모드 변경, 273
 - 버퍼
 - BUFFER 명령, 320
 - GET FILE 명령, 319
 - OPEN SELECTED 명령, 319
 - SET BUFFER 명령, 318, 319
 - 두 개 편집, 323
 - 메시지 표시, 319
 - 삭제, 314, 317, 318
 - 상태 변경, 318, 319
 - 섹션 두 개 보기, 323
 - 여러, 319
 - 작성, 314
 - 정보 표시, 317
 - 정의, 314
 - 조작, 314
 - 창 분할, 323
 - 버퍼 변경 저널링, 304
 - /RECOVER 한정자, 306
 - RECOVER BUFFER 명령, 307
 - 명령, 304
 - 비활성화, 304, 308
 - 정의, 304
 - 파일, 305
 - 파일 이름, 305, 306
 - 편집 내용 복구, 306, 308
 - 활성화, 309
 - 삽입 모드, 273
 - 상자 편집, 290
 - 상자 편집, 285
 - 겹쳐쓰기 모드, 290
 - 명령, 286, 288
 - 삭제 보류의 효과, 289
 - 삽입 모드, 290
 - 연습, 287
 - 텍스트 복원, 289
 - 텍스트 붙여넣기, 286
 - 텍스트 선택, 285
 - 텍스트 잘라내기, 286
 - 세션 종료, 263
 - 실행, 258
 - 검색 목록, 302
 - 여러 입력 파일 사용, 302, 303
 - 와일드카드 디렉토리 이름, 303
 - 와일드카드 디렉토리 이름 사용, 302, 303
 - 와일드카드 사용, 302
 - 일괄 처리 작업 로그 파일 읽기, 675
 - 정의, 251
 - 종료, 263
 - 창, 321
 - 명령, 321
 - 키, 321
 - 커서 이동, 265
 - 명령, 265
 - 연습, 269
 - 키, 265
 - 키
 - 정의
 - VT100 계열 터미널, 261
 - VT200 계열 터미널, 260
 - VT300 계열 터미널, 260
 - VT400 계열 터미널, 260
 - 키 이름 규칙, 253
 - 텍스트 바꾸기, 296
 - 텍스트 복사, 284
 - 연습, 284
 - 텍스트 복원, 275
 - 명령, 275
 - 삭제 보류 작업 이후, 289
 - 상자 편집, 289
 - 연습, 278
 - 키 편집, 275
 - 텍스트 이동, 279
 - 명령, 279
 - 연습, 283
 - 편집 키, 279
 - 텍스트 입력, 271
 - 겹쳐쓰기 모드, 273
 - 명령, 272
 - 삽입 모드, 273
-

색인

연습, 273
키 편집, 272
특수 문자, 271
파일 포함, 271
텍스트 지우기, 275
명령, 275
삭제 오류 사용, 289
연습, 278
키 편집, 275
텍스트 찾기, 291
 FIND 명령 사용, 292
 검색 방향, 292
 대/소문자 일치, 293
명령, 291
 와일드카드 사용, 295
 위치 표시, 296
파일 백업, 304
파일 쓰기, 320
파일 읽기, 320
편집 내용 저장, 263
 EXIT 명령 사용, 263
 WRITE FILE 명령 사용, 263
편집 키, 310
하위 프로세스 만들기, 324
하위 프로세스 생성, 324, 325
EVE 명령
 ATTACH, 324
EXIT 명령
 Mail, 204
명령 프로시저, 517
명령 프로시저에 사용, 533
사용할 시기, 518
조건 코드, 550
EXTRACT 명령
 Mail, 217

F

F\$CONTEXT 렉시칼 함수, 644
F\$CVTIME 렉시칼 함수, 651
F\$DIRECTORY 렉시칼 함수, 480
F\$ELEMENT 렉시칼 함수, 651
F\$ENVIRONMENT 렉시칼 함수
 기본 파일 보호 변경, 640
F\$EXTRACT 렉시칼 함수, 651
F\$FAO 렉시칼 함수, 653
F\$GETQUI 렉시칼 함수
 대기열 정보 얻기, 642
F\$GETSYI 렉시칼 함수
 정보 얻기
 시스템, 642
F\$INTEGER 렉시칼 함수, 656

 데이터 유형 변환, 483, 655
F\$LENGTH 렉시칼 함수
 F\$LOCATE와 같이 사용, 650
F\$LOCATE 렉시칼 함수
 F\$LENGTH와 같이 사용, 650
F\$LOGICAL 렉시칼 함수, 648
F\$MODE 렉시칼 함수
 로그인 프로시저, 672
F\$PARSE 렉시칼 함수, 651
F\$PID 렉시칼 함수, 643
 프로세스 정보 얻기, 643
F\$SEARCH 렉시칼 함수
 DELETE 명령과 함께 사용, 647
 파일 검색, 647
F\$STRING 렉시칼 함수
 데이터 유형 변환, 483, 655
F\$STRNLNM 렉시칼 함수, 648
 LNMS\$DCL LOGICAL 논리 이름, 432
 논리 이름 변환, 648
F\$TYPE 렉시칼 함수
 기호 식별, 483, 656
F\$VERIFY 렉시칼 함수, 638
FILE 명령
 Mail, 223
Files-11 On-Disk 구조, 194
FORM 명령
 Mail, 237
FORTUSER.COM 명령 프로시저, 726
 보기 실행, 729
FORWARD 명령
 Mail, 219

G

GET FILE 명령, 319
GETPARMS.COM 명령 프로시저, 718
 보기 실행, 719

H

HELP 명령, 72
 EVE, 256
HELP/MESSAGE 명령, 74

I

IF 명령
 명령 프로시저에서 사용, 513
INCLUDE FILE 명령, 320
INITIALIZE 명령, 194
 Files-11 On-Disk 구조, 194
 형식, 194
INQUIRE 명령

READ 명령과 비교, 570
 기호 및 일괄 처리 작업 사용, 570
 명령 프로시저 사용, 513
 ISO Latin-1 문자 집합, 686

J

JTQUOTA 값, 443

L

LINK 명령, 175
 LISTER.COM 명령 프로시저, 731
 보기 실행, 733
 LNM\$CLUSTER 논리 이름, 432
 LNM\$CLUSTER_TABLE 논리 이름, 433
 LNM\$DCL LOGICAL 논리 이름
 LNM\$FILE_DEV, 432
 SHOW LOGICAL 명령, 432
 LNM\$FILE_DEV 논리 이름
 검색 목록, 419, 432
 프로세스-개인 정의, 444
 LNM\$GROUP 논리 이름, 432, 433
 LNM\$JOB 논리 이름, 432, 434
 LNM\$PERMANENT_MAILBOX 논리 이름, 432
 LNM\$SYSCLUSTER 논리 이름, 432, 433
 LNM\$SYSCLUSTER_TABLE 논리 이름, 433
 LNM\$SYSTEM 논리 이름, 432, 435
 LNM\$TEMPORARY_MAILBOX 논리 이름, 432
 LOCKPWD 플래그, 53
 LOGIN.COM 파일
 논리 이름 사용, 409
 LOGOUT 명령, 75
 /FULL 한정자, 75

M

MAIL 명령, 204
 /EDIT 한정자, 229
 /SUBJECT 한정자, 216
 Mail 유틸리티(MAIL)
 EVE 사용, 229, 230
 기본 편집기 설정, 230
 네트워크를 통해 메시지 보내기, 209
 메시지 교환, 236
 메시지 삭제, 226
 메시지 읽기, 205, 235
 메시지 제거, 236
 메시지를 파일로 추출, 217
 메시지에 파일 추가, 218
 명령 요약, 235
 문제 및 제한 사항
 중첩된 인용 부호가 있는 주소에 응답, 219

변경 및 향상/추가 적용
 SET FORWARD 명령, 220
 보안 단위, 228
 우편 그룹, 212
 전송, 209
 초기화 파일, 234
 키패드 명령, 232
 텍스트 편집기 사용, 229
 파일 보내기, 215
 DCL 수준, 216
 호출, 204
 Mail 폴더
 MAIL, 206
 NEWMAIL, 205
 WASTEBASKET, 226
 만들기, 223
 목록 표시, 223
 삭제, 224
 선택, 223
 Mail 하위 디렉토리
 만들기, 222
 MAIL\$INIT 파일, 234
 MAIL\$INTERNET_MODE 논리, 209
 MAIL\$INTERNET_TRANSPORT 논리, 209
 MAIL\$KEYDEF.INI 파일, 234
 보기, 234
 MAIL\$SYSTEM_FLAGS 논리, 219
 MAIL.MAI 파일, 222
 MAIL DIRECTORY 명령
 Mail, 240
 MAILEDIT.COM 명령 프로시저, 230, 724
 보기 실행, 725
 MARK 명령, 239
 MERGE 명령, 349
 /KEY 한정자, 350, 351
 /NODUPPLICATES 한정자, 351
 /STABLE 한정자, 351
 Sort/Merge 유틸리티 참조, 327
 MIME 유틸리티, 243
 주석, 248
 파일 인코딩, 247
 파일 추출, 246
 MIME\$FILETYPES.DAT, 244
 MIME\$MAILCAP.DAT, 244
 MOUNT 명령, 190, 195
 /FOREIGN 한정자, 197
 형식, 195
 MOVE 명령
 Mail, 223

색인

N

SET, 236, 237, 239, 240

Null 값

파일 유형, 124

파일 이름, 124

null 문자

DCL 명령, 90

O

ON CONTROL_Y 명령

Ctrl/Y 활성화, 547

ON 명령

Ctrl/Y 동작 루틴 설정, 543

OPEN SELECTED 명령, 319

OPEN 명령

명령 프로시저

논리 이름 만들기, 410

영구 프로세스 논리 이름, 448

영구 프로세스 파일, 448

OpenVMS Cluster 시스템

장치 이름, 191

node-allocation-class 필드, 191

이중 경로, 191

형식, 191

OpenVMS 화면 관리 소프트웨어

명령 다시 호출, 101

P

PERSONAL_NAME 명령

Mail, 240

PID 번호, 716

F\$PID 렉시칼 함수를 사용하여 얻기, 643

프로세스 컨텍스트, 660

PIPE 명령, 89, 448, 624, 631

중단, 628

PQL\$ JTQUOTA 할당량 목록 값, 443

PQL DJTQUOTA 시스템 매개 변수, 443

PQL MJTQUOTA 시스템 매개 변수, 443

PRINT 명령, 134, 137

Mail, 227

한정자, 138

PURGE 명령, 132

Mail, 226, 236

Q

QUEUE 명령

Mail, 237

QUIT 명령, 204

EVE, 263

R

READ 명령

INQUIRE 명령과 비교, 570

Mail, 205

Mail에서의 /NEW 한정자, 205

Mail의 /NEW 한정자, 225

명령 프로시저

논리 이름 사용, 410

명령 프로시저에 데이터 읽기, 570

RECALL 명령, 100

/ERASE 한정자, 100, 398

REMINDER.COM 명령 프로시저, 708

보기 실행, 711

RENAME 명령, 128

REPLY 명령

Mail, 219

RMS

Mail의 파일 태그, 216

RSX 시스템

UIC 형식 디렉토리 이름 지정 방법, 160

RUN 명령

/JOB_TABLE_QUOTA 한정자, 443

검색 목록, 424

프로세스와 함께 사용, 663

S

SEARCH 명령

Mail, 207

SELECT 명령

Mail, 206, 223

SEND 명령

Mail, 208, 215

/EDIT 한정자, 230

SET CONTROL=T 명령, 71

SET CONTROL=Y 명령, 547

SET COPY_SELF 명령, 239

SET DEFAULT 명령

기본 디렉토리 설정, 155

기본 장치 설정, 155

SET EDITOR 명령

Mail, 230, 240

SET ENTRY 명령, 677

한정자, 677

SET FILE 명령, 238

Mail, 224

SET FOLDER 명령, 238

Mail, 223

SET FORWARD 명령, 240

Mail, 220

SET HOST 명령, 56

SET NOCONTROL=Y 명령

- Ctrl/Y 비활성화, 547
- SET NOON 명령
 - 명령 수준, 540
 - 명령 프로시저, 540
- SET ON 명령
 - 명령 수준, 540
- SET OUTPUT RATE 명령, 675
- SET PASSWORD 명령, 61
 - /GENERATE 한정자, 62
 - /SECONDARY 한정자, 63
 - 자동 암호 생성, 62
- SET PREFIX 명령, 675
- SET PROCESS 명령
 - /NAME 한정자, 660
- SET PROTECTION 명령
 - /DEFAULT 한정자, 396
- SET QUEUE 명령
 - /ENTRY 한정자, 677
 - Mail, 227
 - 한정자, 677
- SET SECURITY 명령
 - /ACL 한정자, 396
 - /PROTECTION 한정자, 133, 157, 392, 397
 - 객체 보안 프로파일 변경, 392
- SET SIGNATURE_FILE 명령
 - Mail, 218
- SET SYMBOL 명령, 487
 - 동사 문자열 변환, 487
- SET TERMINAL 명령, 102, 104
 - /BROADCAST 한정자, 71
 - /INSERT 한정자, 103
 - /LINE_EDIT 한정자, 102
 - /OVERSTRIKE 한정자, 103
 - /WRAP 한정자, 103
- SET VERIFY 명령, 675
- SHOW ALL 명령
 - Mail, 240
- SHOW AUTO_PURGE 명령
 - Mail, 236
- SHOW BUFFERS 명령, 319
- SHOW COPY_SELF 명령, 240
- SHOW DEFAULT 명령, 155
- SHOW DEVICES 명령, 188
- SHOW EDITOR 명령
 - Mail, 240
- SHOW ENTRY 명령, 135, 671, 679
- SHOW FILE 명령, 238
 - Mail, 224
- SHOW KEY 명령, 239
- SHOW LOGICAL 명령, 420
 - LNMSDCL_LOGICAL 논리 이름, 432
 - 논리 이름 표
 - 표시, 420
- 디렉토리 표 구조
 - 표시, 427
- 액세스 모드
 - 표시, 421
- 영구 프로세스 파일 이름
 - 표시, 420
- SHOW PROCESS 명령, 390
 - /ALL 한정자, 659
- SHOW QUEUE 명령, 135, 679
 - /FULL 한정자, 674
- SHOW SECURITY 명령
 - 객체의 보안 프로파일 표시, 392
 - 새 보호 코드 확인, 392
- SHOW SYMBOL 명령, 460, 479
- SHOW TERMINAL 명령, 102
- SHOW TRANSLATION 명령, 420, 432
- SMTP(단순 메일 전송 프로토콜)
 - 전송 지정, 209
- SORT 명령
 - Sort/Merge 유틸리티 참조, 327
- Sort/Merge 유틸리티(SORT/MERGE)
 - /KEY 한정자, 337, 338
 - /NODUPPLICATES 한정자, 340
- SORT 명령, 332, 347
 - Sort/Merge 유틸리티(고성능) 참조, 329
- STABLE 한정자, 340
- 기본 키 사용, 332
- 레코드 병합
 - 동일한 키 필드, 351
- 성능 향상, 360
 - /STATISTICS 한정자, 360
 - 작업 집합 범위 수정, 363
 - 작업 파일 관리, 362
 - 정렬 작업 동안, 361
 - 정렬하는 동안, 362
- 일괄 처리 작업으로 실행, 347
 - 명령 프로시저, 347
 - 입력 레코드 포함, 347
- 정렬
 - 다중 키 필드 사용, 338
 - 동일한 키 필드 사용, 340
 - 레코드, 342
 - 비문자 데이터 파일, 340
- 조합 순서, 343
 - /COLLATING_SEQUENCE 한정자, 343
 - ASCII, 344
 - EBCDIC, 344
 - NCS, 344
 - 기본값, 344
 - 다국적, 344

색인

- 사용자 정의, 344
 - 지정 방법 파일, 353
 - 명령 무시, 353
 - 주석 포함, 354
 - 한정자 순서, 354
 - 형식, 353
 - 출력 파일, 341
 - 터미널에서 레코드 입력, 352
 - 파일 병합, 349
 - 키 필드에 따라 정렬된 경우, 350
 - 파일 정렬, 332
 - 한정자, 364, 385
 - SORT 및 MERGE 명령 사용, 364
 - 입력 파일 지정 방법, 369
 - 지정 방법 파일, 374
 - 출력 파일 지정 방법, 370
 - Sort/Merge 유틸리티(고성능)
 - 루틴, 329
 - SPAWN 명령, 324, 664
 - /NOLOGICAL_NAMES 한정자, 668
 - /NOSYMBOL 한정자, 668
 - /NOWAIT 한정자, 664
 - SPLIT WINDOW 명령, 323
 - STOP 명령
 - 명령 프로시저, 518
 - SUBMIT 명령, 671
 - /LOG 한정자, 675
 - /RESTART 한정자, 681, 682
 - 명령 프로시저에 매개 변수 전달, 568
 - 명령 프로시저와 함께 사용, 529
 - 여러 명령 프로시저 지정, 673
 - 한정자, 678
 - SUBMIT/AFTER 명령, 672
 - SYLOGIN.COM 명령 프로시저, 671
 - SYNCHRONIZE 명령, 682, 683
 - SYS\$COMMAND 논리 이름, 430
 - 명령 프로시저에서 사용, 449
 - 영구 프로세스, 447
 - 재정의, 451
 - SYS\$COMMON 논리 이름, 435
 - SYS\$DISK 논리 이름, 428, 430
 - SYS\$ERROR 논리 이름, 430
 - 영구 프로세스, 447
 - 재정의, 450
 - 출력, 로그 파일, 675
 - SYS\$ERRORLOG 논리 이름, 435
 - SYS\$EXAMPLES 논리 이름, 435
 - SYS\$HELP 논리 이름, 435
 - SYS\$INOUT 논리 이름
 - 재정의, 568
 - SYS\$INPUT 논리 이름, 428, 430, 572
 - 별개의 파일로 정의, 573
 - 영구 프로세스, 447
 - 영구 프로세스 파일, 674
 - 재정의, 448, 572
 - SYS\$INSTRUCTION 논리 이름, 435
 - SYS\$LDR 논리 이름, 436
 - SYS\$LIBRARY 논리 이름, 428, 435
 - SYS\$LOADABLE_IMAGES 논리 이름, 435
 - SYS\$LOGIN 논리 이름, 428, 434
 - SYS\$LOGIN_DEVICE 논리 이름, 434
 - SYS\$MAINTENANCE 논리 이름, 436
 - SYS\$MANAGER 논리 이름, 436
 - SYS\$MESSAGE 논리 이름, 436
 - SYS\$NET 논리 이름, 430
 - SYS\$NODE 논리 이름, 436
 - SYS\$OUTPUT 논리 이름, 430
 - 영구 프로세스, 447
 - 재정의, 449
 - 출력, 로그 파일, 675
 - SYS\$PRINT 논리 이름, 135
 - SYS\$PROC DMP 논리 이름, 436
 - SYS\$REM_ID 논리 이름, 434
 - SYS\$REM_NODE 논리 이름, 434
 - SYS\$SCRATCH 논리 이름, 428, 434
 - SYS\$SHARE 논리 이름, 436
 - SYS\$SPECIFIC 논리 이름, 436
 - SYS\$STARTUP 논리 이름, 436
 - SYS\$SYSDEVICE 논리 이름, 435, 436
 - SYS\$SYSROOT 논리 이름, 435, 437
 - SYS\$SYSTEM 논리 이름, 428, 437
 - SYS\$TEST 논리 이름, 437
 - SYS\$UPDATE 논리 이름, 437
 - SYS.COM 명령 프로시저, 715
 - 보기 실행, 717
 - SYSGEN(System Generation Utility), 176
 - SYSUAF.DAT 파일
 - JTQUOTA 값, 443
- ## T
- TCP/IP(전송 제어 프로토콜/인터넷 프로토콜)
 - 원격 시스템 연결 손실, 76
 - 이름 및 주소 지정, 118
 - 파일 복사, 128
 - 파일 액세스, 119
 - 파일 인쇄, 136
 - THEN 명령
 - 명령 프로시저에서 사용, 513
 - time-stamp
 - 확인 설정, 639
 - Time-stamping
 - SET PREFIX 명령 사용, 675
 - TYPE 명령
 - 와일드카드 문자 사용, 130
 - 원격 노드에서 파일 표시, 129

파일 표시, 129

U

UAF(사용자 인증 파일)
로그인 클래스 제한, 59
마지막 로그인 레코드, 402
UIC 디렉토리 지정
이름 형식으로 변환, 160
UIC 디렉토리 지정 방법, 160
UIC 증명서
보기, 391
UIC(사용자 식별 코드)
기본 보호, 396
UNMARK 명령, 239

V

VOLPRO(볼륨 보호 무시), 195

W

WAIT 명령
명령 프로시저 동기화, 682, 683
WASTEBASKET 폴더
Mail에서 비우기, 226
WRITE FILE 명령, 320
EVE, 263
WRITE 명령
데이터 쓰기, 574
레코드에 문자열 기록, 653
명령 프로시저
논리 이름 사용, 410
WWPPS 유틸리티, 139
WWPPS(World-Wide 포스트스크립트 인쇄 하위 시스템), 139

가

가상 터미널
사용 불가, 55
연결이 끊긴 프로세스, 669, 670
연결이 끊긴 프로세스 관리, 670
연결이 끊긴 프로세스에 다시 연결, 669
제한 사항, 670
감탄 부호(!)
우편 그룹, 213
값
DCL 명령행, 86
객체
보안 프로파일, 392
보안 프로파일 변경, 392
보안 프로파일 표시, 392

객체 소유권
변경, 392
검색 목록, 407
RUN 명령 사용, 424
기본값
수정, 419, 444
논리 이름, 422
와일드카드, 423
변환, 422
여러
검색 순서, 425
접촉쓰기 모드, 103
경보
보안에 사용, 402
이벤트 트리거, 402
계정
개방, 53
만료, 65
만료된 계정 업데이트, 66
보조 암호, 51
액세스 감사, 402
유형, 53
제한, 53
처음 로그인, 50
초기 암호, 50
계정 정보
로그아웃 시 얻기, 75
글뱅이 기호
Mail에서 우편 그룹과 함께 사용, 214
글뱅이 기호(@)
Mail에서 우편 그룹과 함께 사용, 213
실행 프로시저 명령, 526
구문
DCL 명령, 86
OpenVMS Cluster 장치 사양, 191
UIC 디렉토리 지정 방법, 160
노드 지정 방법, 116
테이프 볼륨의 파일 지정 방법, 125
하위 디렉토리, 152
기본값
DCL 명령, 88
명령 프로시저에서 복원, 522
명령 프로시저에서 저장, 522
시스템에 의해 제공된 값, 88
정의, 69
파일 보호, 396
기호
DCL 명령으로 사용, 458
F\$TYPE 렉시칼 함수를 사용하여 식별, 656
간략화, 458
값 표시, 460

색인

글로벌, 457
논리 데이터, 476
논리 이름과 비교, 456
대체, 459, 489
강제, 490
단계, 494
데이터 행, 494
명령 구문 분석, 494
명령 입력 스캔, 494
반복적인, 495
순서, 490
앰퍼샌드(&) 사용, 492, 493
어포스트로피 사용, 491
연산자, 491
자동, 489
표현식 평가, 494
로컬, 457
문자 유형, 464
문자열, 464
연산, 466
표현식, 465
문자열 할당, 462
및 숫자 식, 471
변수로 사용, 463
별표(*), 458
삭제, 459
산술 연산, 472
숫자 값 표시, 470, 482
숫자 비교, 473
숫자 식, 471
숫자 오버레이, 474, 475
앰퍼샌드(&), 493
앰퍼샌드(&), 492
어포스트로피, 491
연결, 461
외부 명령, 458, 459
정의, 453, 457
기호, 461
렉시칼 함수, 479
렉시칼 함수로 사용, 479, 480
문자열, 465
정의되지 않음, 497
평가, 482
표현식, 463
할당문으로 만들기, 457
기호 대체
문자열, 575
기호 범위
글로벌, 488
로컬, 487

정의, 487
기호 표, 485
검색 순서, 486
글로벌, 485
\$RESTART 기호, 486
\$SEVERITY 기호, 486
로컬, 485
매개 변수, 485
기호표
글로벌
\$STATUS 기호, 485

나

날짜
변경 날짜 및 시간 지정, 96
절대 날짜/시간 지정, 95
네트워크
로그아웃, 76
우편 보내기, 209
원격 시스템 연결 손실, 76
네트워크 노드
로컬, 119
원격, 118
네트워크 액세스 제어 문자열, 398
네트워크 파일 지정 방법
ULTRIX 제한, 120
기본 형식, 120
외부 파일 형식, 120
작업 지정 방법 문자열, 120
노드
로컬, 119
노드 이름
논리 이름 만들기, 413
입력 규칙, 116
전체, 117
정의, 112
파일 지정 방법의 형식, 116
논리 식, 476, 477
논리 이름, 420
CONCEALED 속성, 412
DCL\$PATH, 500
TERMINAL 속성, 412
검색 목록, 444
RUN 명령, 424
검색 목록 만들기, 422
와일드카드, 423
기호와 비교, 456
노드
지정 방법 파일에서 사용, 414
논리 이름 디렉토리에 추가, 444

- 등가 문자열, 407
 - 만들기, 409
 - 노드, 413
 - 만들기 규칙, 410
 - 명령 프로시저, 407
 - 명령 프로시저에서 사용, 410
 - 변환, 417
 - 검색 목록, 422, 444
 - 대화형 프로세스, 447
 - 명령 프로시저, 447, 448
 - 반복적, 417
 - 속성, 411
 - 수정, 444
 - 시스템 기본값, 418
 - 일괄 처리 프로세스, 447
 - 삭제, 416, 446
 - 시스템, 408, 432, 435
 - 엑세스 모드, 412
 - 표시, 421
 - 영구 프로세스, 447
 - 명령 프로시저, 448
 - 재정의, 447, 448, 449, 451
 - 파일 지정 방법, 448
 - 우편함, 432
 - 유효한 문자, 410
 - 정의, 405
 - 최대 길이, 410
 - 파일 입력, 410
 - 파일 출력, 410
 - 표시, 420
 - 한 객체에 여러 이름 정의, 415
 - 논리 이름 표
 - 검색 순서
 - 수정, 444
 - 공유, 426, 428, 440
 - 그룹, 428, 433
 - 기본 검색 순서, 444
 - 기본값, 428
 - 논리 이름 디렉토리에 추가, 444
 - 디렉토리, 426
 - 만들기, 440
 - 보호, 442
 - 보호 정의, 437
 - 사용 제어 항목, 442
 - 삭제, 446
 - 시스템, 428, 435
 - 시스템 디렉토리 표의 내용, 431
 - 엑세스 요구 사항, 441
 - 작업, 428, 434
 - 할당량 지정, 443
 - 클러스터 전체, 428, 432, 433
 - 만들기, 441
 - 클러스터 전체 상위, 433
 - 특권 요구 사항, 441
 - 특성, 426
 - 표시, 420, 427
 - 프로세스, 426, 428, 430, 440
 - 프로세스 디렉토리, 429
 - 할당량 지정, 442
 - 논리 이름 표 참조, 433
 - 논리식, 476
 - 논리적 장치 이름, 189
- 다**
- 다국적 문자 집합
 - DEC 다국적 문자 집합 참조, 686
 - 다중 파일 지정 방법
 - 매개 변수 목록, 156
 - 달러 기호(\$)
 - DCL 프롬프트, 46, 84
 - OpenVMS Cluster 장치 사양, 191
 - 명령 프로시저, 506
 - 파일 이름, 112
 - 대/소문자 구분
 - DCL 명령행, 84
 - EVE에서 REPLACE 명령 사용, 297
 - SET FIND CASE EXACT 명령 사용, 297
 - 대화형 모드
 - 로그인, 56
 - 데이터
 - WRITE 명령을 사용하여 쓰기, 574
 - 논리, 476, 477
 - 명령 프로시저에 포함, 563
 - 터미널에 쓰기, 574
 - 데이터 유형
 - 렉시칼 함수와 같이 사용, 655
 - 데이터 파일
 - 프로그램 포함, 573
 - 도움말
 - 명령, 74
 - 시스템 메시지, 74
 - 도움말 메시지 유틸리티(MSGHLP)
 - 호출, 74
 - 등가 문자열, 407, 412
 - DEFINE 명령, 409
 - 변환 속성, 411
 - 유효한 문자, 410
 - 최대 길이, 410
 - 디렉토리
 - 내용 표시, 153
 - 논리 이름 표

색인

- 시스템, 426
- 프로세스, 426
- 삭제, 154
- 엑세스, 157
- 정의, 148
- 디렉토리 구조
 - 샘플, 150
- 디렉토리 내의 파일 목록 표시, 153
- 디렉토리 이름
 - UIC 형식을 이름 형식으로 변환, 161
- 대체
 - 줄임표(...) 와일드카드 문자, 158
 - 하이픈(-) 와일드카드 문자, 159
- 파일 지정 방법의 형식, 152
- 디렉토리 지정 방법
 - 정의, 152
 - 형식, 152
- 디렉토리 파일
 - 기본, 150, 155
 - 기본값 설정, 155
 - 보호, 157
 - 생성, 152
 - 최상위, 150
 - 형식, 152
- 디렉토리 필드
 - 백분율 기호(%) 와일드카드 문자 사용, 123
 - 별표(*) 와일드카드 문자 사용, 122
 - 정의, 112
- 디렉토리에 있는 파일 표시, 153
- 디버거
 - 기본 입력 스트림, 435
 - 기본 출력 스트림, 435
- 디버깅
 - 명령 프로시저, 518
 - SET PREFIX 명령, 519
 - SET VERIFY 명령, 519
 - SHOW SYMBOL 명령, 519
- 디스크
 - 마운트, 196
- 디스크 조각 모음, 175
- 라
- 레이블
 - 로컬 기호 레이블, 507
 - 명령 프로시저, 87, 507
 - 명령 프로시저의 중복, 507
- 레코드
 - Sort/Merge 유틸리티 참조, 327
 - 정렬, 342
 - 필드, 332
 - 터미널에서 레코드 입력, 352
 - 레코드 정렬, 332
 - 레코드 지정 장치
 - 출력 파일 지정으로 사용, 198
 - 렉시칼 함수
 - FSDIRECTORY, 480
 - 구문, 479
 - 기호 식별, 656
 - 논리 이름 변환, 648
 - 데이터 유형 변환, 655
 - 데이터 유형 조작, 655
 - 데이터 평가, 656
 - 명령 프로시저에서 사용, 479
 - 문자열 조작, 650
 - 문자열 확인, 650
 - 문자열의 일부 추출, 651
 - 사용, 479, 480
 - 장치 검색, 646
 - 정보 얻기
 - OpenVMS Cluster 노드, 644
 - 대기열, 642
 - 시스템, 641, 642
 - 파일 및 장치, 646
 - 프로세스, 636, 643
 - 정의, 633
 - 출력 문자열 서식 지정, 653
 - 파일 검색, 647
 - 프로세스 특성 변경, 636
 - 프로세스 특성 저장, 637
 - 로그 파일
 - 내용, 675
 - 명령 출력 포함, 676, 677
 - 일괄 처리 작업, 675
 - 일괄 처리 작업 실행 중 점검, 676
 - 저장, 676
 - 로그아웃
 - 보안 고려 사항, 77
 - 연결이 끊긴 프로세스, 670
 - 운영 체제, 75
 - 원격 세션, 76
 - 로그인
 - 네트워크, 57
 - 대화형, 56
 - 로컬, 56
 - 마지막 모니터링, 402
 - 만료 계정, 66
 - 비대화형, 56, 57
 - 사용 불가능
 - 침입 방지, 60
 - 암호 변경, 64
 - 운영 체제
 - 오류, 47

- 원격, 56
 - 원격 시스템, 56
 - 일괄 처리, 57
 - 전화 접속, 56
 - 암호 제공 기회, 60
 - 제어, 51
 - 프록시, 57
 - 허가된 시간, 59
 - 로그인 메시지, 54
 - 생략, 55
 - 알림, 54
 - 암호 만료, 47
 - 로그인 명령 프로시저
 - 개인, 553
 - 일괄 처리 작업으로 실행, 675
 - 일괄 처리 작업을 위한 실행, 672
 - 전용 계정, 554
 - 정의, 553
 - 로그인 실패
 - 다시 시도, 60
 - 메시지, 55
 - 원인, 47, 58, 60
 - 로그인 오류
 - 메시지, 402
 - 로그인 클래스, 56
 - 네트워크, 57
 - 대화형, 56
 - 로컬, 56
 - 비대화형, 57
 - 원격, 56
 - 일괄 처리, 57
 - 전화 접속, 56
 - 제한, 59
 - 로컬 기호 테이블
 - 레이블, 507
 - 루프
 - 작성, 515
 - 정의, 515
 - 리터럴 문자
 - 명령 프로시저에 포함, 513
 - 리터럴 텍스트
 - 문자열, 574
- 마**
- 마스터 파일 디렉토리(MFD), 150
 - 마운트 요청, 196
 - 매개 변수
 - DCL 명령행, 86
 - 명령 프로시저에 데이터 전달, 565
 - 일괄 처리 작업에 데이터 전달, 568
 - 중첩된 명령 프로시저에 데이터 전달, 569
 - 지정
 - null 값, 567
 - 기호, 566
 - 문자열, 565
 - 정수, 565
 - 매개 변수 목록
 - 다중 파일 지정 방법, 156
 - 다중 파일 지정 방법 기본값, 156
 - 매개 변수 한정자, 93
 - 메시지
 - Mail에서 복사, 223
 - 감사, 403
 - 로그인 동안, 54
 - 로그인 실패, 55
 - 로그인 오류, 402
 - 명령에 대한 시스템 응답, 69
 - 명령행 오류 표시, 70
 - 생략, 55
 - 시스템 오류, 70
 - 시스템 표시에 사용된 규칙, 70
 - 알림, 54
 - 정보, 69
 - 메시지 개수
 - Mail에서 수정, 225
 - 메시지 읽기
 - Mail, 235
 - 명령
 - Mail에서 입력, 203
 - 간략화, 90
 - 날짜 형식, 95
 - 시간 형식, 95
 - 자동 외부, 499
 - 취소, 88
 - 명령 다시 호출, 100
 - RECALL 명령 사용, 100
 - 화살표 키 사용, 100
 - 명령 수준
 - SET NOON 명령, 540
 - 정의, 533
 - 명령 순서
 - PIPE 명령을 사용하여 만들기, 624, 627
 - 명령 인터프리터
 - 명령 수행, 576
 - 명령 처리
 - 3 단계, 494
 - 명령 프로시저
 - BATCH.COM, 736
 - CALC.COM, 734
 - CLEANUP.COM, 524
 - COMPILE_FILE.COM, 741
 - CONVERT.COM, 703

색인

- Ctrl/Y를 사용하여 중단, 541
- Ctrl/Y를 사용하여 취소, 541
- DCL 명령 포함, 505
- DCL 명령의 순서, 512
- DIR.COM, 712
- EDITALL.COM, 721
- EXIT 명령, 517, 533
- FORTUSER.COM, 726
- GETPARMS.COM, 718
- IF 명령, 513
- LISTER.COM, 731
- LOGIN.COM, 672, 675
- MAILEDIT.COM, 724
- REMINDER.COM, 708
- STOP 명령, 518
- SYLOGIN.COM, 671
- SYS\$INPUT 재정의, 568
- SYS.COM, 715
- THEN 명령, 513
- 개인 로그인, 553
 - 전용 계정, 554
- 기본 오류 동작, 536
- 기본 파일 유형, 505
- 기본값 복원, 522
- 기본값 저장, 522
- 달러 기호(\$), 506
- 디버깅, 518
 - SET PREFIX 명령, 519
 - SET VERIFY 명령, 519
 - SHOW SYMBOL 명령, 519
- 디자인, 510
- 레이블, 507
- 레이블 오류 처리, 535
- 로그인, 553
- 리터럴 문자 포함, 513
- 만들기, 505
- 매개 변수를 사용하여 데이터 전달, 565
- 명령행 포함, 505
- 변수 결정, 511
- 변수 사용, 648
- 변수 할당, 512
- 사용자 입력 프롬프트 표시, 570
- 실행, 526
 - 개인 디스크, 532
 - 기호를 사용하지 않음, 499
 - 다른 명령 프로시저, 526
 - 대화형, 529
 - 대화형 출력 재지정, 529
 - 원격 노드, 526
 - 원격 일괄 처리 작업, 530
- 일괄 처리 작업, 529
- 일괄 처리 작업 다시 시작, 530
- 자동 외부 명령, 499
- 테이프 볼륨, 532
- 명령 한정자, 92
- 명령행
 - Mail, 204
 - 다시 호출, 100
 - 명령 프로시저에 포함, 505
 - 문자 삭제, 103
- 완료, 524
- 유형, 504
- 일괄 처리 작업에서 지정, 673
- 입력, 563
- 작성, 509
- 작성 단계, 510
- 정리 작업, 521, 523
- 정의, 503
- 조건 결정, 511
- 조건 테스트, 512
- 종료, 517, 533
- 주석, 508
- 중복 레이블, 507
- 중첩
 - 매개 변수를 사용하여 데이터 전달, 569
- 출력, 574
- 테스트, 518
- 파일 닫기, 522
- 파일 삭제, 522
- 프로그램 스텝, 514
- 프로그램 스텝 바꾸기, 524
- 확인 설정 변경, 638
- 확인 설정 비활성화, 638
- 명령 프로시저 실행, 526
- 개인 디스크, 532
- 기호를 사용하지 않음, 499
- 다른 명령 프로시저, 526
- 대화형, 529
- 대화형 출력 재지정, 529
- 원격 노드, 526
- 원격 일괄 처리 작업, 530
- 일괄 처리 작업, 529
- 일괄 처리 작업 다시 시작, 530
- 자동 외부 명령, 499
- 테이프 볼륨, 532

배치, 103
 편집, 102
 명령행 편집, 102
 겹쳐쓰기 모드, 103
 문자 삭제, 103
 삽입 모드, 103
 행 편집 활성화, 102
 모드 액세스 모드 참조, 412
 문자
 영숫자, 464
 인쇄할 수 없음, 464
 특수, 464
 문자 집합, 171
 DEC 다국적 문자 집합 참조, 686
 ISO Latin-1 다국적, 166
 문자열
 교체, 468
 기호, 464
 기호 대체, 575
 대체, 468, 469
 렉시칼 함수 추출, 651
 렉시칼 함수를 사용하여 변환, 655
 렉시칼 함수를 사용하여 확인, 650
 렉시칼 함수와 같이 사용, 650
 리터럴 텍스트, 574
 비교, 466, 467
 연산, 466
 인용 부호(" "), 574
 정수 값으로 변환, 483
 정의, 465
 평가, 482
 표현식
 기호, 465
 피연산자, 465
 문자열 할당
 기호 포함, 462

바

반복
 정의, 509
 백분율 기호(%)
 와일드카드 문자, 123
 백스페이스 키, 103, 105
 버전 번호, 확장 파일 지정 방법, 167
 버퍼
 DCL 명령 다시 호출, 100
 변경 시간
 구문, 96
 기본값, 97
 입력 규칙, 97
 절대 시간과 조합, 98

변수

INQUIRE 명령을 사용하여 할당, 513
 명령 프로시저에서 결정, 511
 명령 프로시저에서 할당, 512
 정의, 509
 별표(*)
 기호 정의, 458
 와일드카드 문자
 UIC 형식 디렉토리 지정 방법, 160
 디렉토리 지정 방법, 122
 사용 규칙, 122

보안

감사 로그 파일, 403
 관리자, 51
 높은 수준, 402
 데이터 보호 메커니즘, 392
 보안 감사, 402
 계정 및 파일 액세스, 402
 메시지, 403
 사용 시기 결정, 404
 파일에 ACE 추가, 404
 보안 감사 로그 파일, 403
 보안 감사 이벤트, 402
 보안 경고, 402
 보안 기능
 계정 지속 기간, 65, 66
 로그인 클래스 제한, 59
 보안 경고, 402
 암호 만료, 65
 이동 제한, 59
 전화 접속 다시 시도, 60
 침입 방지, 60

보안 제한

로그인 클래스, 59
 시간, 59
 이동, 59

보안 프로파일

객체, 392
 객체 변경, 392
 객체 표시, 392
 사용자
 표시, 390
 사용자 표시, 390
 프로세스
 표시, 390
 프로세스 표시, 390

보조 암호

길이, 52

보호

객체, 392
 기본값, 396

색인

디렉토리, 157
파일, 133
 Mail, 228
보호 코드, 394
범주, 394
엑세스 유형, 395
볼륨
 마운트, 195
 외부, 197
 분리, 198
 할당된 장치, 199
 운영자 지원, 196
 정보 표시, 188
 초기화, 194
볼륨 다시 초기화, 198
볼륨 마운트
 보안 감사, 403
 외부, 197
볼륨 집합
 마운트, 195
 정의, 187
분리
 볼륨, 198
 할당된 장치, 199
비디오 터미널, 48

사
사용 목록, 394
사용 제어 항목(ACL)
 논리 이름 표, 442
 파일 보호, 133
사용자
 프로세스 권한 증명서 표시, 390
사용자 인증 파일(UAF), 50
 LOCKPWD 플래그, 53
 계정 만료, 66
삭제
 명령행 문자, 103
삽입 모드
 정의, 103
서명 파일
 Mail 메시지에 추가, 218
서브 루틴, 제어 전달, 612
서브셀
 PIPE 명령, 627
숫자
 DCL에서 인식하는 정수 값, 470
 내부 저장, 471
 렉시칼 함수를 사용하여 변환, 655
 문자열 값으로 변환, 484
 비교, 473

 평가, 482
숫자 식, 471
 정수 피연산자, 471
숫자 오버레이, 474, 475
시간
 변경 날짜 및 시간 지정, 96
 절대 날짜/시간 지정, 95
시간 초과 기간, 46
시스템
 사용 제어, 51
시스템 권한
 파일 보호, 133
시스템 메시지
 메시지 참조, 403
시스템 보안
 감사 로그 파일, 403
시스템 예제
 논리 이름, 435
실행
 여러 명령 문자열, 624
실행 프로시저(@) 명령, 526

아

아래쪽 화살표 키
 명령 다시 호출, 100, 105
악센트 기호 문자, 171, 556, 557
암호
 개방 계정, 53
 기본, 51, 52
 길이, 49, 50, 61
 다시 시도, 60
 만료, 65
 변경, 61
 /NEW_PASSWORD 한정자 사용, 64
 로그인 시, 64
 로그인할 때, 50
 보조, 63
 빈도 지침, 50, 67
변경 실패, 66
변경 이유, 402
변경 확인, 61
보조, 51, 65
 입력, 52
보호, 67
사용자, 51
새 암호, 64
새 암호 설정, 61
생성, 61, 62
선택, 49
 선택을 위한 지침, 49
시스템, 51

- 로그인 실패 발생, 59
- 입력, 51
- 안전한, 49
- 위험도 높음, 49
- 이중, 51
- 자동 생성, 62
- 작성, 63
- 잘못된, 55
- 잠김, 53
- 재사용, 49, 50
- 적합한, 50
- 전화 접속 시 제공, 60
- 제한, 50
- 초기, 50
- 추측, 50
- 프록시 로그인에서 생략, 399
- 암호 보호, 67
- 전화 접속 다시 시도, 20
- 암호변경
 - /NEW_PASSWORD 한정자, 64
- 액세스 모드, 412
- 논리 이름
 - 표시, 421
 - 논리 이름 표, 426
- 정의, 449
- 액세스 유형
 - 보안 감사, 403
- 액세스 제어
 - 객체 보안 프로파일, 392
- 논리 이름 표
 - 공유, 426
- 액세스 제어 문자열, 117, 398
- 노드 간 파일 복사, 128
- 노드 이름의 형식, 121
- 명령 프로시저 내, 398
- 무시, 414
- 입력 규칙, 121
- 정보 보호, 398
- 형식, 398
- 앰퍼샌드(&)
 - 기호 정의, 492, 493
- 에포스트로피
 - 기호 정의, 491
- 연결, 파일, 127
- 연산자(수치)
 - 계산 순서, 481
- 영구 프로세스, 420
- 오류
 - 레이블, 535
 - 로그인 하는 중, 47
 - 명령 프로시저에서 감지, 549
 - 명령 프로시저에서 처리, 535
 - 오류 검사 비활성화, 540
 - 오류 처리
 - ON 명령 동작, 537
 - SET NOON 명령, 540
- 오른쪽 화살표 키
 - 커서 이동, 105
- 와일드카드 문자
 - DCL 명령행, 87
 - 백분율 기호(%), 123
 - 별표(*), 122
 - 줄임표(...), 158, 159
 - 파일 이름, 122
 - 하이픈(-), 159
- 외부 명령, 458
- 문자 제한, 457, 458
- 자동, 499
- 외부 파일 지정 방법
 - 네트워크, 120
- 우편 그룹
 - DCL 수준에서 우편 보내기, 214
 - 편집기를 사용하여 만들기, 212
- 원격 노드
 - F\$CONTEXT를 사용하여 정보 얻기, 644
 - 액세스 제어 문자열, 118
- 원격 세션
 - 중단, 76
- 위쪽 화살표 키
 - 명령 다시 호출, 100, 105
- 위치 한정자
 - 정의, 92
- 이동 제한, 59
- 이미지, 175
 - 기호를 사용하지 않고 호출, 499
 - 자동 외부 명령으로 호출, 499
 - 출력 재지정, 575
- 이스케이프 문자, 166, 557
- 인쇄
 - 가로, 137
- 인쇄 대기열
 - 제어, 137
- 인쇄 작업, 134, 137
 - 대기열 정보, 135
 - 실행, 134
 - 여러 사본 얻기, 137
 - 우선 순위, 134
 - 지연, 137
 - 함께 사용할 DCL 명령 목록, 137
- 인용 부호(" ")
 - 문자열, 574
- 인터넷
 - 전송 지정, 209

색인

일괄 처리 작업

SUBMIT 명령
 한정자, 678
 다시 시작, 681, 682
 로그 파일, 675
 사용, 671
 삭제, 681
 상태 표시, 679
 시작 시간 지정, 672
 여러 명령 프로시저 지정, 673
 여러 프로시저 동기화, 682, 683
 이동 제한에 영향받음, 59
 인증, 57
 입력 제공, 674
 작업 항목, 671
 정의, 347
 제출, 671
 명령 프로시저, 529, 671
 중지, 681
 출력, 675
 특성 변경, 677
 항목 변경, 677
일괄 처리 프로세스
 영구 프로세스 논리 이름, 447
일반 장치 이름, 190
일반 증명서
 보기, 391
입력
 명령 프로시저에서 프롬프트 표시, 570
 일괄 처리 작업, 674
입력 스트림, 572
재지정, 627
입력 파일
 매개 변수 목록의 임시 기본값, 156
입력 파일 목록의 임시 기본값, 156

자

자동 외부 명령, 499
작업, 57
작업 종료
 이동 제한에 의해 부과됨, 59
작업 지정 방법 문자열
 네트워크, 120
작업 컨트롤러
 이동 제한에 영향 받음, 59
작업 트리
 정의, 664
장치
 F\$DEVICE 렉시컬 함수를 사용하여 정보 얻기, 646
 개인용, 198
 블록 집합 액세스, 187

파일 액세스, 198
할당, 193
할당 분리, 199
장치 번호 필드, 191
 기본값, 190
장치 이름, 156
 OpenVMS Cluster 시스템, 191
 논리적, 189
 일반, 190
장치 필드
 정의, 112
전송
 Mail에서 지정, 209
전용 계정, 554
전체 이름, 117
전화 접속
 로그인 실패, 60
전화 접속 회선
 액세스 제어, 51
절대 시간
 구문, 95
 기본값, 95
 변경 시간과 조합, 98
 입력 규칙, 95
 키워드 지정, 96
정렬에 사용할 수 있는 임시 저장 공간, 362
정리 작업, 523
 명령 프로시저, 521
제어 문자
 기타 문자, 686
제어 문자, 목록, 686
조건
 명령 프로시저에서 결정, 511
 명령 프로시저에서 테스트, 512
 정의, 509
조건 코드
 EXIT 명령 사용, 550
 보안 수준, 551
 정의, 549
 표시, 549
조합 시간
 구문, 98
 입력 규칙, 98
종료
 명령 프로시저, 533
주석
 감탄 부호(!) 사용, 213
 명령 프로시저, 508
 우편 그룹, 213
줄임표(...)
 디렉토리 이름의 와일드카드 문자, 158, 159
증명서

UIC, 391
 일반, 391
 프로세스 표시, 390
 환경, 391

차

초기화
 디스크 볼륨, 194
 볼륨
 디스크 볼륨, 194
 초기화 파일, 234
 최상위 디렉토리, 150
 출력
 명령 및 이미지에서 재지정, 575
 명령 프로시저, 574
 출력 문자열
 F\$FAO 렉시칼 함수를 사용하여 서식 지정, 653
 출력 스트림
 재지정, 627
 출력 재지정, 198
 침입 시도, 60
 감사, 402
 방지, 60

카

컨트롤러 지정 필드
 기본값, 190
 정의, 191
 코드 컴파일러, 175
 클러스터 전체 시스템 표, 433
 키 시퀀스, 105
 키 이름
 Mail, 232
 키 정의
 Mail, 233
 지정, 104
 키 필드, 327
 키(키보드)
 Mail에서 정의, 232
 키보드 키
 DCL 명령 다시 호출, 105
 DCL 명령 입력, 105
 DCL 명령 중단, 105
 정의, 104
 커서 위치 제어, 105
 화면 표시 제어, 108
 화살표
 명령 다시 호출, 100, 105
 키워드
 DCL 명령행, 87

키패드
 Mail에서 정의, 234
 Mail의 기본값, 232

타

탭 정지
 Ctrl/K를 사용하여 이동, 105
 탭 키, 105
 터미널
 가상, 55, 669, 670
 데이터 쓰기, 574
 시스템 암호
 요구 사항, 51
 시스템 암호 요청, 59
 액세스 제어, 51
 연결 끊김, 669
 전화 접속 로그인, 56
 화면 지우기, 398
 화면 표시 중단 및 시작, 108
 터미널 제어 문자
 숫자 값, 686
 테스트
 명령 프로시저, 518
 테이프
 ANSI 볼륨 파일 지정 방법 형식, 125
 테이프 볼륨
 파일 지정 방법, 125
 텍스트 편집기
 파일 표시, 129
 특권
 VOLPRO, 195

파

파이프라인, 624
 파일
 /VERSION_LIMIT 한정자 사용, 116
 F\$SEARCH 렉시칼 함수를 사용하여 찾기, 647
 Mail 메시지에 추가, 218
 MAIL.MAI, 222
 경보 적용, 404
 기밀 보호, 404
 만들기, 126
 COPY 명령 사용, 126
 CREATE 명령 사용, 126
 Mail 메시지, 217
 명령 프로시저에서 닫기, 522
 명령 프로시저에서 삭제, 522
 버전 번호, 112, 116
 버전 번호 제어, 116
 보안 감사 사용 시기 결정, 404

색인

- 보안 감사를 위한 ACE 추가, 404
- 보호, 133, 228
- 복사, 126
 - 노드 간, 127
 - 원격 계정, 401
- 삭제, 131
- 수정, 126
- 액세스
 - 개인용 장치, 198
 - 블록 집합, 187
- 액세스 감사, 402, 404
- 연결, 127
- 영구 프로세스, 448
- 와일드카드 문자 지정, 122
- 우편 보내기
 - Mail 유틸리티 사용, 215
- 우편으로 보내기
 - DCL 수준, 216
- 이름 바꾸기, 128
- 인쇄, 134
- 정의, 111
- 제거, 132
- 초기화, 234
- 표시
 - 내용, 129, 130
 - 와일드카드 문자 사용, 130
- 프록시 액세스에 필요한 보호, 401
- 파일 I/O
 - 재지정, 448
- 파일 버전 번호
- 별표(*) 와일드카드 문자를 사용, 122
- 파일 보내기
 - Mail, 215
- 파일 보호, 133, 228
 - F\$ENVIRONMENT 렉시칼 함수를 사용하여 기본값 변경, 640
 - 기밀 파일, 404
 - 프록시 액세스에 필요, 401
- 파일 복사, 126
 - 노드 간, 127
- 파일 브라우저, 404
- 파일 액세스
 - 원격 노드, 119
 - 제어 문자열 사용, 121
- 파일 유형, 112
 - 기본값 목록, 114, 115
 - 입력 규칙, 112
- 파일 유형 필드
 - null 값 사용, 124
 - 백분율 기호(%) 와일드카드 문자 사용, 123
 - 별표(*) 와일드카드 문자 사용, 122
- 정의, 112
- 파일 지정 방법, 112
 - null 값 사용, 124
 - 네트워크, 119
 - 노드 이름, 116, 127
 - 목록 지정 방법, 156
 - 백분율 기호(%) 와일드카드 문자 사용, 123
 - 별표(*) 와일드카드 문자 사용, 122
 - 유효한 문자, 112
 - 입력 규칙, 112
 - 자기 테이프의 대체 형식, 125
 - 포함된 필드의 목록, 112
- 페이지 및 스왑 파일, 176
- 표시, 420
- 표현식
 - DCL 평가, 482
 - 값 데이터 유형 변환, 483
 - 기호, 463
 - 논리, 476, 477
 - 연산자 우선 순위, 481
- 프로그램
 - 데이터 파일에 포함, 573
- 프로그램 스텝
 - 명령 프로시저, 514
 - 명령으로 바꾸기, 524
- 프로세스
 - Ctrl/T로 상태 확인, 71
 - 다시 연결, 669
 - 렉시칼 함수를 사용하여 특성 변경, 636
 - 만들기, 658
 - 분리, 663
 - 연결, 669
 - 연결 끊김, 669
 - 연결되어 있음, 54
 - 연결이 끊김, 54
 - 로그아웃, 670
 - 일반적으로 변경되는 특성, 523
 - 작업 트리, 664
 - 정의, 658
 - 특성 저장, 637
 - 프로세스 권한 증명서 표시, 390
- 프로세스 컨텍스트
 - 특성 목록, 659, 660
- 프록시 계정, 118, 399
 - 기본값, 401
 - 단일 사용자, 400
 - 명명, 401
 - 여러 계정 중 선택, 401
 - 여러 사용자, 401
 - 일반 액세스, 401
 - 최대 허용 개수, 399

프록시 로그인, 57, 399

보안 이점, 399

주요 특성, 400

프롭포트, 87

DCL, 46

피연산자

문자열 표현식, 465

정수, 471

필드, 332

Sort/Merge 유틸리티 참조, 327

하

하위 디렉토리

구문, 152

기본값 설정, 155

목록 표시, 153

생성, 152

하위 프로세스

만들기, 664, 665

여러, 664

상위 프로세스에서 상속된 특성, 667

상위 프로세스의 특성 제외, 667

작업 트리, 664

정의, 664

제어권 전송, 668

종료, 666

컨텍스트, 667

하이픈(-)

디렉토리 이름, 159

한정자

DCL 명령행, 86

값 지정, 93

날짜 형식, 95

매개 변수, 93

명령, 92

시간 형식, 95

위치, 92

충돌, 93

할당

클래스 필드, 191

할당량

논리 이름 표, 426, 442

작업, 443

할당문

기호 만들기, 457

행 편집, 102

호출 버퍼, 398

지우기, 101

화살표 키

DCL 명령행을 사용하여 커서 이동, 105

명령 다시 호출, 100, 105

확인

명령 프로시저 실행 중 활성화, 521

확장 파일 지정 방법

파일 이름

DCL 명령 매개 변수에 사용, 556

환경 증명서

보기, 391