

OpenVMS

ユーザーズ・マニュアル

AA-PZWLG-TE

2002 年 10 月

本書は、OpenVMS オペレーティング・システムの使用方法を説明しています。本書の情報はすべての OpenVMS ユーザを対象としており、OpenVMS オペレーティング・システムを実行するすべてのコンピュータに適用されます。

改訂 / 更新情報:	『OpenVMS ユーザーズ・マニュアル』 V7.3 の改訂版です。
ソフトウェア・バージョン:	OpenVMS Alpha V7.3-1 OpenVMS VAX V7.3

コンパックコンピュータ株式会社

© 2002 Compaq Computer K.K.

本書の著作権はコンパックコンピュータ株式会社が保有しており、本書中の解説および図、表はコンパックの文書による許可なしに、その全体または一部を、いかなる場合にも再版あるいは複製することを禁じます。

また、本書に記載されている事項は、予告なく変更されることがありますので、あらかじめご承知おきください。万一、本書の記述に誤りがあった場合でも、コンパックは一切その責任を負いかねます。

本書で解説するソフトウェア (対象ソフトウェア) は、所定のライセンス契約が締結された場合に限り、その使用あるいは複製が許可されます。

コンパックは、コンパックまたはコンパックの指定する会社から納入された機器以外の機器で対象ソフトウェアを使用した場合、その性能あるいは信頼性について一切責任を負いかねます。

以下は、米国 Compaq Computer Corporation の商標です。

Compaq, VAX, VMS および Compaq ロゴ。

OpenVMS は、米国 Compaq Information Technologies Group, L.P. の商標です。

以下は、他社の商標です。

Microsoft, MS および MS-DOS は米国 Microsoft 社の商標です。

Motif, OSF, OSF/1, OSF/Motif および Open Software Foundation は米国 Open Software Foundation 社の商標です。

その他のすべての商標および登録商標は、それぞれの所有者が保有しています。

原典： OpenVMS User's Manual

© 2002 Compaq Computer Corporation

本書は、日本語 VAX DOCUMENT V 2.1を用いて作成しています。

目次

まえがき	xxi
1 OpenVMS オペレーティング・システムの概要	
1.1 ログイン	1-1
1.1.1 ログインの成功	1-3
1.1.2 ログイン・エラー	1-3
1.2 PC からのログイン	1-3
1.3 アカウントのパスワードの選択	1-4
1.3.1 初期パスワードの取得	1-4
1.3.2 初期パスワードの変更	1-5
1.3.3 パスワードの制限事項	1-5
1.3.4 使用するパスワードのタイプ	1-5
1.3.5 システム・パスワードの入力	1-6
1.3.6 2 次パスワードの入力	1-7
1.3.7 各種アカウントのパスワードの要件	1-7
1.4 情報メッセージの読み込み	1-8
1.4.1 メッセージの無表示	1-9
1.4.2 成功ログイン・メッセージ	1-9
1.5 ログインのタイプとログイン・クラス	1-9
1.5.1 会話型ログイン	1-10
1.5.2 非会話型ログイン	1-10
1.6 ログインの失敗	1-11
1.6.1 システム・パスワードを必要とするターミナル	1-12
1.6.2 ログイン・クラスの制限	1-12
1.6.3 勤務時間に関する制限	1-12
1.6.4 勤務時間制限中でのバッチ・ジョブ	1-12
1.6.5 ダイアルアップ・ログインでの失敗	1-13
1.6.6 システム侵入回避プロシージャ	1-13
1.7 パスワードの変更	1-14
1.7.1 ユーザ自身によるパスワードの選択	1-14
1.7.2 生成されたパスワードの使用方法	1-14
1.7.3 生成されたパスワード: 欠点	1-16
1.7.4 2 次パスワードの変更	1-16
1.7.5 ログイン時のパスワードの変更	1-16
1.8 パスワードとアカウントの満了	1-17
1.8.1 満了したパスワード	1-17
1.8.2 2 次パスワードを使用している場合	1-17
1.8.3 パスワードを変更しなかった場合	1-18
1.8.4 満了したアカウント	1-18
1.9 パスワードの保護に関するガイドライン	1-18

1.10	システム応答の認識方法	1-20
1.10.1	省略時のアクション	1-20
1.10.2	情報システム・メッセージ	1-20
1.10.3	システム・エラー・メッセージ	1-20
1.10.4	現在のプロセスのチェック	1-21
1.11	システムのヘルプについて	1-22
1.11.1	オンライン・ヘルプの使用方法	1-22
1.11.2	指定コマンドのヘルプを見る	1-23
1.11.3	システム・メッセージについてのヘルプ	1-24
1.12	システムからのログアウト	1-24
1.12.1	アカウント情報の取得	1-25
1.12.2	リモート・セッションの終了	1-25
1.12.3	ネットワーク接続の消失	1-25
1.13	システム・セキュリティを損なわずにログアウトする	1-26
1.14	ネットワーク	1-26
1.14.1	ネットワーク・ノード	1-27
1.14.2	ネットワークを介してのプログラムの実行	1-27
2	DCL を使用したシステムとの会話	
2.1	コマンドの入力	2-2
2.1.1	使用モード	2-3
2.1.2	DCL コマンドの種類	2-3
2.2	DCL コマンド行	2-3
2.2.1	構文	2-5
2.2.2	コマンドの取消し	2-6
2.2.3	省略時の値の使用	2-6
2.2.4	複数行のコマンドの入力	2-6
2.3	DCL コマンドの入力規則	2-7
2.4	パラメータの入力	2-8
2.5	修飾子の入力	2-9
2.5.1	コマンド修飾子	2-9
2.5.2	定位置修飾子	2-10
2.5.3	パラメータ修飾子	2-10
2.5.4	矛盾する修飾子	2-10
2.5.5	修飾子が取る値	2-11
2.6	値としての日付と時刻の入力	2-11
2.6.1	絶対時刻形式	2-12
2.6.2	デルタ時間の形式	2-13
2.6.3	複合時刻の形式	2-13
2.7	コマンドの再呼び出し	2-14
2.7.1	Ctrl/B の使用	2-15
2.7.2	矢印キーの使用	2-15
2.7.3	RECALL コマンドの使用	2-15
2.8	DCL コマンド行の編集	2-16
2.8.1	SET TERMINAL コマンド	2-17
2.8.2	コマンド行の要素の削除	2-18
2.9	ターミナル・キーの定義	2-18

2.9.1	キー・シーケンス	2-18
3	ファイル情報の格納	
3.1	ファイル名とファイル指定	3-2
3.1.1	完全なファイル指定	3-2
3.1.2	ファイル指定の規則	3-3
3.1.3	DCL コマンドの省略時のファイル・タイプ	3-4
3.1.4	言語ソース・プログラムの省略時のファイル・タイプ	3-5
3.1.5	ファイルのバージョン番号	3-6
3.1.6	ネットワーク・ノード名	3-7
3.1.7	DECnet-Plus の完全なノード名の指定	3-7
3.1.8	TCP/IP 名およびアドレスの指定	3-8
3.1.9	DECnet 使用によるリモート・ノード上のファイルへのアクセス	3-8
3.1.10	TCP/IP 使用によるリモート・ノード上のファイルへのアクセス	3-9
3.1.11	ネットワーク・ファイル指定の使用方法	3-9
3.1.11.1	従来のファイル指定	3-9
3.1.11.2	フォーリン・ファイル指定	3-10
3.1.11.3	タスク指定文字列	3-10
3.1.12	アクセス制御文字列の指定方法	3-10
3.2	ファイル名でのワイルドカードの使用方法	3-11
3.2.1	アスタリスク(*)・ワイルドカード文字	3-11
3.2.2	パーセント記号(%)ワイルドカード文字	3-12
3.3	その他のファイル名	3-13
3.3.1	空のファイル名とファイル・タイプ	3-13
3.3.1.1	空のファイル・タイプによるファイル参照	3-13
3.3.2	磁気テープの代替ファイル名の使用方法	3-14
3.4	ファイルの作成と変更	3-14
3.4.1	ファイルの作成	3-14
3.4.2	ファイルのコピー	3-15
3.4.3	ファイルの連結	3-15
3.4.4	DECnet を使用してリモート・ノードからファイルをコピーする	3-15
3.4.5	DECnet を使用して使用中のノードにあるファイルをリモート・ノードにコピーする	3-16
3.4.6	TCP/IP を使用してリモート・システム上へファイルをコピーする	3-16
3.4.7	ファイルをコピーするためのアクセス制御文字列の使用	3-16
3.4.8	ファイル名の変更	3-17
3.5	ファイルの内容の表示	3-17
3.5.1	TYPE コマンドの使用	3-17
3.5.2	表示内容を制御する	3-17
3.5.3	リモート・ノードのファイルを表示する	3-18
3.5.4	ワイルドカードによるファイルの表示	3-18
3.5.5	複数のファイルの表示	3-18
3.6	ファイルの削除	3-18
3.6.1	PURGE コマンドの使用方法	3-19
3.7	他のユーザからのファイルの保護	3-20
3.7.1	アクセス制御リスト (ACL)	3-20
3.7.2	ファイル保護の種類	3-20
3.8	ファイルの印刷	3-20
3.8.1	印刷ジョブの優先順位	3-21
3.8.2	キュー情報の表示	3-21

3.8.3	プリント・フォーム	3-22
3.8.4	印刷ジョブの停止	3-22
3.8.5	別のノードにあるファイルの印刷	3-22
3.8.6	印刷コマンドの修飾子	3-23
3.8.7	WWPPS ユーティリティ (Alpha のみ)	3-24
3.8.7.1	WWPPS の呼び出し	3-26
3.8.7.2	WWPPS ユーティリティ・コマンド	3-26
4	ディレクトリ・ファイルの編成	
4.1	ディレクトリ構造	4-2
4.2	ディレクトリとは	4-4
4.2.1	ディレクトリの作成	4-4
4.2.2	ディレクトリの表示	4-5
4.2.3	ディレクトリの削除	4-6
4.3	省略時のディレクトリの設定	4-6
4.3.1	省略時の値を存在しないディレクトリに設定する	4-7
4.3.2	SHOW DEFAULT コマンド	4-7
4.3.3	一時的な省略時の値の使用	4-8
4.4	他のユーザからのディレクトリの保護	4-8
4.5	ワイルドカードを使用したディレクトリ構造の検索	4-9
4.5.1	反復記号(...)ワイルドカード文字	4-9
4.5.2	ハイフン(-)・サブディレクトリ文字	4-10
4.6	UIC 形式でのディレクトリの操作	4-11
4.6.1	UIC ディレクトリでのワイルドカードの使用	4-12
4.6.2	UIC 形式から名前形式への変換	4-12
5	拡張ファイル指定	
5.1	ODS-5 ボリューム構造	5-1
5.1.1	長いファイル名	5-2
5.1.2	ファイル名の中で使用できる文字の種類の拡大	5-2
5.1.3	大文字と小文字の区別の保存	5-4
5.1.4	ワイルドカードの使用	5-5
5.1.4.1	ワイルドカード文字	5-5
5.1.4.2	ワイルドカードの構文	5-5
5.2	深いディレクトリ構造	5-6
5.2.1	ディレクトリの命名構文	5-6
5.2.2	ディレクトリ ID およびファイル ID の短縮形	5-7
5.3	DCL での拡張ファイル指定解析機能の使用	5-7
5.4	拡張ファイル指定を使用できる状況	5-8
5.5	拡張ファイル名の表示	5-10
5.5.1	DIRECTORY コマンド	5-10
5.5.2	TYPE コマンド	5-12
5.5.3	DELETE コマンド	5-12
5.5.4	PURGE コマンド	5-13
5.6	拡張ファイル名の端末表示	5-13
5.7	複合環境での作業	5-13

6 ディスクとテープ・ドライブの使用方法

6.1	物理デバイス名	6-2
6.2	デバイス情報の表示	6-2
6.3	論理デバイス名	6-3
6.4	汎用デバイス名	6-3
6.5	OpenVMS Cluster デバイス名	6-3
6.6	ボリュームとボリューム・セット	6-4
6.7	デバイスの管理	6-4
6.7.1	デバイスの割り当て	6-4
6.7.2	ボリュームの初期化	6-5
6.7.3	ボリュームのマウント	6-6
6.7.4	オペレータの手助けが必要な場合	6-7
6.8	プライベート・デバイス上のファイルのアクセス	6-8
6.8.1	ボリュームのディスマウント	6-8

7 Mail を使用して他のユーザと通信する

7.1	Mail の起動と終了	7-2
7.1.1	Mail の起動	7-2
7.1.2	Mail の終了	7-3
7.2	メッセージの開封	7-3
7.2.1	新しいメールを読む	7-3
7.2.2	古いメッセージの開封	7-4
7.2.3	メッセージの検索	7-5
7.3	メッセージの送信	7-5
7.4	ネットワークを介したメールの送信	7-6
7.4.1	ネットワーク・プロトコルの指定	7-6
7.4.2	ノード名の指定	7-7
7.4.3	インターネット・メール・アドレスの使用	7-7
7.4.4	論理ノード名の使用方法	7-7
7.5	複数のユーザへのメッセージの送信	7-8
7.5.1	個人名の使用方法	7-8
7.5.2	配布リストの作成	7-8
7.5.3	配布リストによるメッセージの送信	7-9
7.6	Mail 内でのファイルの操作	7-10
7.6.1	DDIF ファイルの送信	7-11
7.6.2	DCL からのファイルの送信	7-11
7.6.3	メッセージからのファイルの作成	7-12
7.6.4	メッセージへのファイルの追加	7-13
7.7	その他のメッセージの送信方法	7-14
7.7.1	メッセージの返信	7-14
7.7.1.1	入れ子になった引用符を含むアドレスへの返信	7-14
7.7.2	メッセージの転送	7-14
7.7.2.1	SET FORWARD コマンド	7-15
7.8	メッセージの整理	7-16
7.8.1	フォルダの作成	7-16
7.8.2	メール・サブディレクトリの作成	7-16

7.8.3	フォルダへのメッセージの移動	7-17
7.8.4	フォルダ間でのメッセージのコピー	7-17
7.8.5	フォルダの選択	7-17
7.8.6	フォルダの削除	7-18
7.8.7	メール・ファイルの作成とアクセス	7-18
7.8.8	メール・メッセージ・カウントの訂正	7-19
7.9	メッセージの削除	7-20
7.9.1	削除したメッセージの回復	7-20
7.10	メール・メッセージの印刷	7-21
7.11	メール・ファイルの保護	7-21
7.11.1	省略時の保護の設定	7-21
7.11.2	セキュリティの測度	7-21
7.12	Mail でのテキスト・エディタの使用法	7-22
7.12.1	EVE の使用法	7-22
7.12.2	/EDIT 修飾子キーワードの使用法	7-23
7.12.3	エディタの選択	7-23
7.12.4	コマンド・ファイルを使用したメールの編集	7-24
7.12.5	選択したエディタの一時的な無効化	7-24
7.13	メール・キーパッドの使用法	7-24
7.13.1	キーパッド・キーの再定義	7-25
7.13.2	追加キー定義のアサイン	7-26
7.13.3	永久キーの定義	7-26
7.14	Mail コマンドの要約	7-26
7.14.1	メッセージを読む	7-27
7.14.2	メッセージの交換	7-28
7.14.3	メッセージの削除	7-28
7.14.4	メッセージの印刷	7-29
7.14.5	メッセージの整理	7-29
7.14.6	メッセージのマーク付け	7-30
7.14.7	メール環境のカスタマイズ	7-30
7.14.8	終了または制御の移行	7-32
7.14.9	メール・ファイルの圧縮	7-33
7.14.10	システム管理コマンド	7-33
7.15	MIME ユーティリティ	7-33
7.15.1	MIME ユーティリティの起動	7-33
7.15.2	MIME ユーティリティの初期化	7-34
7.15.3	オプションの MIME ユーティリティ・ファイルの作成	7-35
7.15.3.1	MIME\$MAILCAP.DAT ファイル処理	7-36
7.15.3.2	MIME\$FILETYPES.DAT ファイルの処理	7-36
7.15.4	MIME ユーティリティを使用した MIME エンコード・ファイルの抽出	7-37
7.15.5	MIME ユーティリティを使用したファイルのエンコード	7-38
7.15.6	MIME ユーティリティ・コマンド	7-38
7.15.7	エラー処理	7-40

8 EVE エディタによるテキスト・ファイルの編集

8.1	EVE の機能	8-2
8.2	ヘルプ情報の参照	8-3
8.2.1	キーパッド・ヘルプの使用法	8-3
8.2.2	EVE ヘルプの使用法	8-4
8.3	セッションの開始と終了	8-5
8.4	コマンドの入力方法	8-6
8.4.1	コマンドの入力	8-6
8.4.2	定義済みキーによる入力方法	8-7
8.5	編集内容のセーブと EVE の終了	8-8
8.5.1	WRITE FILE コマンドの使用	8-8
8.5.2	EXIT コマンドの使用法	8-9
8.5.3	QUIT コマンドの使用	8-9
8.6	カーソルの移動	8-9
8.7	テキストの入力	8-13
8.7.1	テキストの追加	8-13
8.7.2	ファイルの挿入	8-13
8.7.3	印刷不可能な特殊文字	8-13
8.7.4	EVE のテキスト入力用編集キー	8-14
8.7.5	テキスト入力用 EVE コマンド	8-14
8.7.6	バッファ・モードの設定	8-14
8.8	テキストの削除と復元	8-15
8.9	テキストの移動	8-18
8.10	テキストのコピー	8-21
8.11	ボックス編集	8-22
8.11.1	テキスト・ボックスの選択	8-22
8.11.2	テキスト・ボックスのカットとペースト	8-23
8.11.3	SET BOX SELECT コマンド	8-25
8.12	保留削除の使用法	8-25
8.12.1	保留削除による選択範囲の削除	8-25
8.12.2	保留削除を使用して削除した選択範囲の復元	8-26
8.13	テキストの検索と置換	8-26
8.13.1	テキストの検索	8-27
8.13.1.1	検索文字列が見つかった場合	8-28
8.13.2	大文字と小文字を区別する検索の設定	8-28
8.13.3	ワイルドカードの使用法	8-29
8.13.4	空白を含めて検索する方法	8-30
8.13.5	テキストにマークを付ける方法	8-30
8.13.6	テキストの置換	8-31
8.13.6.1	REPLACE コマンドと大文字と小文字の区別	8-31
8.13.6.2	REPLACE コマンドの応答	8-32
8.14	コマンド行修飾子の使用法	8-32
8.14.1	別の開始位置の指定	8-33
8.14.2	作業ファイルの使用	8-34
8.14.3	Main バッファの変更	8-34
8.15	EVE の別の起動方法	8-35
8.15.1	検索リストからの EVE の起動	8-35

8.15.2	ワイルドカードを使用した EVE の起動	8-35
8.15.3	ワイルドカード・ディレクトリを使用した EVE の起動	8-36
8.15.4	複数の入力ファイルを使用した EVE の起動	8-36
8.16	ジャーナリングと回復	8-37
8.16.1	バッファ・ジャーナリングの使用方法	8-37
8.17	EVE の書式化コマンド	8-41
8.18	バッファの使用方法	8-43
8.18.1	バッファ情報の取得	8-45
8.18.2	バッファの削除	8-46
8.18.3	バッファ状態の変更	8-46
8.18.4	メッセージ・バッファの表示方法	8-47
8.18.5	複数バッファの編集	8-47
8.18.6	EVE へのファイルの読み込み	8-48
8.18.7	EVE からのファイルの書き込み	8-49
8.18.8	ウィンドウの使用方法	8-49
8.18.9	単一バッファの 2 つのセクションの表示	8-51
8.18.10	2 つのバッファの編集	8-51
8.19	サブプロセスの作成	8-52
8.19.1	SPAWN コマンドの使用	8-52
8.19.1.1	DCL から EVE へのサブプロセスの生成	8-52

9 ファイルのソートとマージ

9.1	高性能 Sort/Merge ユーティリティの使用	9-2
9.2	ファイルのソート	9-3
9.2.1	キーの定義	9-5
9.2.2	複数のキー・フィールドの使用方法	9-8
9.2.3	同一キー・フィールドによるレコードのソート	9-9
9.2.4	非文字データ・ファイルのソート	9-10
9.2.5	出力ファイルの編成	9-11
9.2.6	ソートのプロセス	9-12
9.3	照合順序の指定	9-13
9.4	バッチ・ジョブによるソートの実行	9-15
9.4.1	コマンド・プロシージャ	9-15
9.4.2	入力レコードの包含	9-15
9.5	ファイルのマージ	9-16
9.5.1	ソート済みのファイル	9-17
9.5.2	同一キー・フィールドによるレコードのマージ	9-18
9.6	ターミナルからのレコードの入力	9-18
9.7	Sort/Merge 指定ファイルの使用方法	9-19
9.8	ソートまたはマージ操作の最適化	9-24
9.8.1	ソートのプロセス	9-25
9.8.2	レコードとフィールドの排除	9-27
9.8.3	作業ファイルの割り当て	9-27
9.8.4	ワーキング・セット拡張領域の変更	9-28
9.9	Sort/Merge 修飾子の要約	9-28
9.9.1	入力ファイル修飾子	9-32
9.9.2	出力ファイル修飾子	9-33

9.9.3	指定ファイル修飾子	9-37
10	資源へのアクセスの制御	
10.1	プロセスのライト識別子の表示	10-2
10.2	オブジェクトのセキュリティ・プロファイル	10-3
10.2.1	セキュリティ・プロファイルの変更	10-4
10.3	保護コードの解釈	10-4
10.4	省略時のファイル保護	10-5
10.4.1	省略時の UIC 保護	10-6
10.4.2	省略時の ACL 保護	10-6
10.4.3	ファイル名の変更	10-7
10.4.4	明示的なファイル保護	10-7
10.5	ネットワークを介してのファイルへのアクセス	10-7
10.5.1	アクセス制御文字列	10-7
10.5.2	アクセス制御文字列の保護	10-8
10.5.3	代理ログイン・アカウントを使用してパスワードを保護する方法	10-8
10.5.4	汎用アクセス代理アカウント	10-10
10.6	アカウントとファイルへのアクセスの監査	10-11
10.6.1	最終ログイン時間	10-11
10.6.2	監査機能やアラームを起動するイベント	10-12
10.6.3	セキュリティ監査ログ・ファイル	10-12
10.6.4	重要なファイルにアクセス制御リスト・エントリ (ACE) を追加する場合	10-13
11	デバイスとファイルの論理名定義	
11.1	論理名の特徴	11-2
11.2	システムが定義する論理名	11-2
11.3	論理名の作成	11-3
11.3.1	DEFINE コマンドの使用法	11-3
11.3.2	ファイル入出力用コマンド・プロシージャ内の論理名の作成	11-4
11.3.3	論理名の作成規則	11-5
11.3.4	変換属性	11-5
11.3.5	アクセス・モード	11-6
11.3.6	論理ノード名の作成	11-7
11.3.6.1	ファイル指定に論理ノード名を使用	11-8
11.3.6.2	アクセス制御文字列の上書き	11-8
11.3.7	同一オブジェクトに複数論理名を作成	11-9
11.4	論理名の削除	11-9
11.5	論理名の変換	11-9
11.5.1	反復変換	11-10
11.5.2	システムの省略時の設定で補完されるフィールド	11-10
11.5.3	論理名変換に対する省略時の検索順序	11-11
11.6	論理名の表示	11-11
11.6.1	論理名テーブル検索の指定	11-12
11.6.2	変換属性とアクセス・モードの表示	11-13
11.7	検索リストの作成と使用	11-13
11.7.1	ワイルドカードを使用できるコマンドでの検索リストの使用	11-14

11.7.2	SET DEFAULT コマンドによる検索リストの使用	11-15
11.7.3	RUN コマンドによる検索リストの使用	11-16
11.7.4	複数の検索リストの検索順序	11-16
11.8	論理名テーブルの特徴	11-17
11.8.1	論理名テーブルのディレクトリ	11-18
11.8.2	ディレクトリ・テーブルの構造の表示	11-18
11.9	省略時設定の論理名テーブル	11-19
11.9.1	プロセス論理名ディレクトリ	11-20
11.9.2	プロセス論理名テーブル	11-20
11.9.3	システム論理名ディレクトリ	11-21
11.9.4	共用論理名テーブル	11-22
11.9.5	共用論理名テーブルの省略時保護	11-25
11.9.6	共用論理名管理のための特権とアクセス必要条件	11-26
11.10	論理名テーブルの作成	11-27
11.10.1	プロセス固有の論理名テーブルの作成	11-27
11.10.2	共用可能論理名テーブルの作成	11-28
11.10.3	クラスタ単位の論理名テーブルの作成	11-28
11.10.4	特権とアクセスの必要条件	11-28
11.10.5	省略時保護の変更	11-29
11.10.6	論理名テーブルへの制限値の設定	11-29
11.10.6.1	ジョブ・テーブルの制限値の設定	11-30
11.11	論理名変換の順序の変更	11-30
11.12	論理名テーブルの削除	11-31
11.13	プロセスパーマネント論理名	11-32
11.13.1	会話型とバッチ処理の間の等価名の相違点	11-32
11.13.2	プロセスパーマネント論理名を使用するファイル入出力の切り換え	11-33
11.13.2.1	SYSS\$INPUT の再定義	11-33
11.13.2.2	SYSS\$OUTPUT の再定義	11-34
11.13.2.3	SYSS\$ERROR の再定義	11-35
11.13.2.4	SYSS\$COMMAND の再定義	11-36
12	シンボル, コマンド, 式の定義	
12.1	シンボルについて	12-2
12.1.1	論理名とシンボルの違い	12-2
12.2	シンボルの使用	12-3
12.2.1	DCL コマンドを表すシンボルの使用	12-4
12.2.2	シンボルの短縮	12-4
12.2.3	フォーリン・コマンドの定義	12-5
12.2.4	シンボルの置換	12-6
12.2.5	シンボルの削除	12-6
12.3	シンボルの表示	12-6
12.4	別のシンボルの値としてのシンボルの使用	12-6
12.4.1	シンボルの連結	12-7
12.4.2	文字列割り当ての中に含まれるシンボル	12-7
12.5	シンボルによるデータの格納と操作	12-8
12.6	文字列	12-8
12.6.1	文字列の定義	12-9
12.6.2	文字列式	12-10

12.6.3	文字列の操作	12-10
12.6.4	文字列の比較	12-11
12.6.5	部分文字列の置換	12-12
12.7	数値と式の使用	12-14
12.7.1	数値の指定	12-14
12.7.2	数値の内部記憶	12-15
12.7.3	算術演算	12-16
12.7.4	数値の比較	12-17
12.7.5	数値オーバーレイの実行	12-18
12.8	論理値と式の使用	12-19
12.8.1	論理演算	12-19
12.8.2	論理式	12-19
12.8.3	論理演算の結果	12-21
12.8.4	レキシカル関数が戻す値の使用	12-21
12.8.5	演算の順序	12-23
12.8.6	データ・タイプの評価	12-24
12.9	式の値タイプの変換	12-25
12.9.1	文字列を整数に変換する	12-25
12.9.2	整数を文字列に変換	12-26
12.10	シンボル・テーブル	12-26
12.10.1	ローカル・シンボル・テーブル	12-26
12.10.2	グローバル・シンボル・テーブル	12-27
12.10.3	シンボル・テーブル検索順序	12-27
12.11	シンボルの値のマスク	12-28
12.11.1	SET SYMBOL コマンド	12-28
12.11.2	シンボルの有効範囲	12-28
12.12	シンボルの置換	12-29
12.12.1	シンボル置換の強制	12-30
12.12.2	シンボル置換の演算子	12-31
12.13	コマンド処理の3つのフェーズ	12-34
12.13.1	第1フェーズ: コマンド入力検索	12-34
12.13.2	第2フェーズ: コマンドの解析	12-34
12.13.3	第3フェーズ: 式の評価	12-34
12.13.4	繰り返し置換と反復置換	12-35
12.13.5	未定義シンボル	12-37
12.14	シンボルを使用するための別の方法: 自動的なフォーリン・コマンド	12-38
12.14.1	自動的なフォーリン・コマンドの使用	12-40
12.14.2	自動的なフォーリン・コマンドの制限事項	12-41
13	コマンド・プロシージャの概要	
13.1	コマンド・プロシージャを作成するための基礎的な説明	13-2
13.1.1	省略時のファイル・タイプ	13-2
13.1.2	コマンドの作成	13-2
13.1.3	コマンド行の作成	13-3
13.2	コマンド行でのラベルの使用	13-3
13.2.1	ローカル・シンボル・テーブル内のラベル	13-4
13.2.2	重複するラベル	13-4
13.3	コマンド・プロシージャでのコメントの使用	13-4

13.4	コマンド・プロシーダの作成方法	13-5
13.5	コマンド・プロシーダの作成手順	13-5
13.5.1	手順 1: コマンド・プロシーダを設計する	13-6
13.5.2	手順 2: 変数を割り当て、条件をテストする	13-7
13.5.2.1	INQUIRE コマンドの使用	13-7
13.5.2.2	リテラル文字の保存	13-8
13.5.2.3	IF と THEN を使用した条件のテスト	13-8
13.5.2.4	プログラム・スタブの作成	13-8
13.5.3	手順 3: ループを追加する	13-10
13.5.4	手順 4: コマンド・プロシーダを終了する	13-11
13.5.4.1	EXIT コマンドの使用	13-11
13.5.4.2	STOP コマンドの使用	13-12
13.5.5	手順 5: プログラム・ロジックをテストおよびデバッグする	13-12
13.5.5.1	コマンド・プロシーダのデバッグ	13-13
13.5.5.2	実行中にチェック機能をオンにする	13-14
13.5.6	手順 6: クリーンアップ・タスクを追加する	13-14
13.5.6.1	ファイルを閉じる	13-15
13.5.6.2	一時ファイルまたは余分なファイルの削除	13-15
13.5.6.3	一般に変更されるプロセス属性	13-16
13.5.6.4	クリーンアップ操作を確実に実行するには	13-16
13.5.7	手順 7: コマンド・プロシーダを完成する	13-16
13.6	コマンド・プロシーダの実行	13-18
13.6.1	他のコマンド・プロシーダの内部からのコマンド・プロシーダの実行	13-18
13.6.2	リモート・ノードでのコマンド・プロシーダの実行	13-19
13.6.2.1	セキュリティに関する注意	13-19
13.6.3	DCL 修飾子またはパラメータを指定したコマンド・プロシーダの実行	13-20
13.6.3.1	制限事項	13-20
13.6.4	会話形式でのコマンド・プロシーダの実行	13-21
13.6.5	バッチ・ジョブとしてのコマンド・プロシーダの実行	13-22
13.6.5.1	リモート・バッチ・ジョブ	13-22
13.6.5.2	バッチ・ジョブの再起動	13-22
13.6.6	ディスクおよびテープ・ボリュームでのコマンド・プロシーダの実行	13-24
13.6.6.1	個人用ディスクでの実行	13-24
13.6.6.2	テープ・ボリューム上での実行	13-24
13.7	コマンド・プロシーダの終了と割り込み	13-24
13.7.1	終了方法	13-24
13.7.2	終了処理ルーチン	13-25
13.8	エラー処理	13-26
13.8.1	省略時のエラー処理	13-27
13.9	エラー処理のその他の方法	13-27
13.9.1	ON コマンド	13-27
13.10	SET NOON コマンドの使用法	13-29
13.11	Ctrl/Y による割り込み処理	13-30
13.11.1	コマンド・プロシーダの中断	13-31
13.11.2	特権イメージの中断	13-32
13.12	Ctrl/Y アクション・ルーチンの設定	13-32
13.12.1	ON コマンドの使用	13-32

13.12.2	Ctrl/Y の入力結果	13-33
13.13	Ctrl/Y による割り込みの無効化/有効化	13-36
13.13.1	SET NOCONTROL=Y の使用	13-36
13.13.2	SET CONTROL=Y コマンドの使用	13-36
13.14	条件コードを使用しているコマンド・プロシージャでのエラーの検出	13-37
13.14.1	条件コードの表示 (\$STATUS)	13-37
13.14.2	EXIT コマンドを含む条件コード	13-38
13.14.3	重大度レベルの決定	13-39
13.14.4	正常終了のテスト	13-39
13.15	\$STATUS を設定しないコマンドの使用	13-40
13.16	ログイン・コマンド・プロシージャ	13-40
13.16.1	システム全体のログイン・コマンド・プロシージャ	13-40
13.16.2	パーソナル・ログイン・コマンド・プロシージャ	13-41
13.16.3	専用アカウントのログイン・コマンド・プロシージャ	13-41
13.17	拡張ファイル指定と (Extended File Specifications) 解析スタイル	13-41
13.18	DCL コマンド・パラメータでの拡張ファイル名の使用	13-42
13.18.1	コマンド・プロシージャのファイル指定	13-42
13.18.2	大文字と小文字の区別の保存と \$FILE	13-44
13.18.3	アンパサンドと一重引用符の置換	13-44
14	DCL での拡張プログラミング	
14.1	コマンド・プロシージャ実行の実行	14-2
14.1.1	コマンド・プロシージャにデータを登録する場合の制限事項	14-2
14.1.2	その他のデータ入力方式	14-2
14.2	データを渡すためのパラメータの使用	14-3
14.2.1	整数としてのパラメータの指定	14-3
14.2.2	文字列としてのパラメータの指定	14-3
14.2.3	シンボルとしてのパラメータの指定	14-4
14.2.4	ヌル値としてのパラメータの指定	14-4
14.3	バッチ・ジョブにデータを渡すためのパラメータの使用	14-5
14.4	ネスティングしたコマンド・プロシージャにデータを渡すためのパラメータの使用	14-5
14.5	データを要求するプロンプトの表示	14-6
14.6	データを入手するための SYS\$INPUT 論理名の使用	14-7
14.6.1	ターミナルとしての SYS\$INPUT の再定義	14-7
14.6.2	SYS\$INPUT を個別ファイルとして定義する場合	14-8
14.7	コマンド・プロシージャ出力の実行	14-9
14.7.1	データの表示	14-9
14.7.2	コマンドとイメージからの出力先の切り換え	14-10
14.7.3	コマンド・プロシージャからのデータを戻す	14-12
14.7.4	エラー・メッセージの出力先の切り換え	14-13
14.7.4.1	SYS\$ERROR の再定義	14-14
14.7.4.2	システム・エラー・メッセージの禁止	14-14
14.8	ファイルの読み込みと書き込み (ファイル入出力)	14-15
14.9	OPEN コマンドの使用	14-16
14.10	ファイルへの書き込み	14-18

14.10.1 固有の名前を持つファイルの作成	14-19
14.11 WRITE コマンドの使用	14-20
14.11.1 データの指定	14-20
14.11.2 /SYMBOL 修飾子の使用	14-21
14.11.3 /UPDATE 修飾子の使用	14-21
14.12 READ コマンドの使用	14-22
14.12.1 /END_OF_FILE 修飾子の使用	14-23
14.12.2 /INDEX と/KEY 修飾子の使用	14-23
14.12.3 /DELETE 修飾子の使用	14-23
14.13 CLOSE コマンドの使用	14-24
14.14 ファイルの変更	14-24
14.14.1 レコードの更新	14-24
14.14.2 新しい出力ファイルの作成	14-26
14.14.3 ファイルへのレコードの追加	14-28
14.15 ファイル入出力エラーの処理方法	14-29
14.15.1 省略時のエラー・アクション	14-30
14.16 実行フローの制御方法	14-30
14.16.1 IF コマンドの使用	14-31
14.16.2 THEN コマンドの使用	14-31
14.16.3 ELSE コマンドの使用	14-32
14.16.4 コマンド・ブロックの使用	14-32
14.16.5 GOTO コマンドの使用	14-36
14.16.5.1 再実行の回避	14-37
14.16.6 GOSUB と RETURN コマンドの使用	14-38
14.17 新しいコマンド・レベルの作成	14-39
14.17.1 CALL コマンドの使用	14-40
14.17.1.1 CALL コマンドの省力時の設定	14-40
14.17.1.2 サブルーチンの始まりと終わり	14-40
14.18 場合分け文の使用	14-43
14.18.1 ラベルのリスト	14-43
14.18.2 場合分け文の作成	14-43
14.19 ループの作成	14-44
14.20 PIPE コマンドの使用	14-47
14.20.1 条件コマンド実行のための PIPE コマンドの使用	14-48
14.20.2 バイプライン実行のための PIPE コマンドの使用	14-48
14.20.3 サブシェル実行のための PIPE コマンドの使用	14-50
14.20.4 バックグラウンド実行のための PIPE コマンドの使用	14-51
14.20.5 入出力リダイレクトのための PIPE コマンドの使用	14-51
14.20.6 PIPE コマンドへの割り込み	14-52
14.20.7 サブプロセスの性能向上	14-52
15 レキシカル関数を使用しての情報の取得と処理	
15.1 なぜレキシカル関数を使うのか	15-1
15.2 プロセスについての情報	15-2
15.2.1 チェック設定値の変更	15-3
15.2.2 省略時のファイル保護の変更	15-5
15.3 システムについての情報	15-5

15.3.1	OpenVMS Cluster ノード名の決定	15-6
15.3.2	キューについての情報	15-6
15.3.3	プロセスについての情報	15-7
15.3.4	F\$CONTEXT レキシカル関数	15-8
15.4	ファイルとデバイスについての情報	15-9
15.4.1	デバイスの検索	15-9
15.4.2	ディレクトリの中でのファイルの検索	15-10
15.4.3	ファイルの古いバージョンの削除	15-11
15.5	論理名の変換	15-11
15.6	文字列の処理	15-12
15.6.1	文字列または文字の有無の判別	15-13
15.6.2	文字列の一部の取り出し	15-13
15.6.3	出力文字列の書式化	15-15
15.7	データ・タイプの処理	15-17
15.7.1	データ・タイプの変換	15-17
15.7.2	式の評価	15-18
15.7.3	シンボルの有無の判別	15-18
16	プロセスとバッチ・ジョブ	
16.1	プロセス・コンテキストへの割り込み	16-1
16.2	独立プロセスの使用	16-4
16.3	サブプロセスの使用	16-4
16.3.1	Spawn タスクによるサブプロセスの使用	16-5
16.3.2	サブプロセスの使用による複数のタスクの実行	16-5
16.3.3	サブプロセスの作成	16-6
16.3.4	サブプロセスの終了	16-6
16.3.5	サブプロセス・コンテキスト	16-7
16.4	仮想ターミナルでの切断されたプロセスの接続	16-8
16.4.1	ターミナルの切断	16-9
16.4.2	切断されたプロセスの削除	16-9
16.4.3	切断されたプロセスの管理	16-9
16.5	バッチ・ジョブ	16-10
16.5.1	バッチ・ジョブの登録	16-10
16.5.2	バッチ・ジョブへのデータの受け渡し	16-13
16.5.3	バッチ・ジョブ出力の制御	16-14
16.5.4	バッチ・ジョブ属性の変更	16-17
16.5.5	SUBMIT コマンド修飾子	16-18
16.5.6	バッチ・キューの中のジョブの表示	16-19
16.5.7	バッチ・ジョブの削除と終了	16-20
16.5.8	バッチ・ジョブの再開	16-20
16.5.9	バッチ・ジョブ実行の同期化	16-21
16.5.10	WAIT コマンドの使用方法	16-22

A 文字セット

B コマンド・プロシージャの例

B.1	CONVERT.COM コマンド・プロシージャ	B-1
B.2	REMINDER.COM コマンド・プロシージャ	B-5
B.3	DIR.COM コマンド・プロシージャ	B-8
B.4	SYS.COM コマンド・プロシージャ	B-11
B.5	GETPARMS.COM コマンド・プロシージャ	B-13
B.6	EDITALL.COM コマンド・プロシージャ	B-15
B.7	MAILEDIT.COM コマンド・プロシージャ	B-18
B.8	FORTUSER.COM コマンド・プロシージャ	B-19
B.9	LISTER.COM コマンド・プロシージャ	B-23
B.10	CALC.COM コマンド・プロシージャ	B-25
B.11	BATCH.COM コマンド・プロシージャ	B-27
B.12	COMPILE_FILE.COM コマンド・プロシージャ	B-32

用語集

索引

例

7-1	MIME\$MAILCAP.DAT ファイル	7-36
-----	------------------------------	------

図

4-1	ディレクトリ構造	4-3
7-1	メール・ディレクトリ構成	7-19
7-2	Mail ユーティリティ・キーパッド	7-25
8-1	EVE のキー: VT200 シリーズ, VT300 シリーズ, および VT400 シリーズのターミナル	8-7
8-2	EVE のキー: VT100 シリーズ・ターミナル	8-8
9-1	昇順にソートした結果	9-4
9-2	リストの中のレコード・フィールド	9-7
9-3	キー・フィールドによるソート	9-7
9-4	ソート済みリスト (キー・フィールドなし)	9-8
9-5	複数のキー・フィールドによるソート結果	9-9
9-6	昇順および降順キーによるソート結果	9-9
9-7	同じキーを持つ場合のソート結果	9-10
9-8	指定ファイルを使用した出力	9-21

13-1	ON コマンドのアクション	13-29
13-2	Ctrl/Y アクション後の実行フロー	13-34
13-3	ネストされたコマンド・プロシージャ実行中の Ctrl/Y	13-35
16-1	バッチ・ジョブ実行の同期化	16-22
A-1	Differences Between DEC Multinational Character Set and ISO Latin-1 Character Set	A-10

表

2-1	DCL コマンドを入力	2-18
2-2	DCL コマンドに割り込む	2-19
2-3	コマンドを再呼び出しする	2-19
2-4	カーソル位置を制御する	2-19
2-5	画面表示を制御する	2-20
3-1	一般的な言語コードと国コードの組み合わせ	3-28
5-1	ワイルドカードおよびパターン・マッチングの例	5-6
5-2	ODS-5 ボリューム上のディレクトリ名	5-7
5-3	サポートされていない OpenVMS コンポーネント	5-9
7-1	MIME ユーティリティのオプション・ファイル	7-36
8-1	カーソルを移動させる EVE 編集キー	8-10
8-2	EVE のカーソル移動用コマンド	8-10
8-3	テキストの削除および復元に使用する EVE の編集キー	8-16
8-4	テキストの削除および復元に使用する EVE コマンド	8-16
8-5	テキストの移動に使用する EVE 編集キー	8-19
8-6	テキストの移動に使用する EVE コマンド	8-20
8-7	ボックス編集に使用する EVE コマンド	8-23
8-8	SET BOX SELECT コマンド	8-25
8-9	バッファ内のテキスト検索用 EVE コマンド	8-27
8-10	EVE の起動時に指定できる EDIT コマンド行修飾子	8-32
8-11	バッファ・ジャーナリングと回復に使用する EVE コマンド	8-37
8-12	バッファ・ジャーナル・ファイル名	8-39
8-13	EVE の編集キーとその機能	8-41
8-14	EVE のテキスト書式化コマンド	8-42
8-15	バッファを操作するために使用する EVE コマンド	8-44
8-16	ウィンドウの作成と操作に使用する EVE のキー	8-50
8-17	EVE ウィンドウ・コマンド	8-50
9-1	高性能 Sort/Merge ユーティリティ: SORT/MERGE との相違点	9-3
9-2	/KEY 修飾子の値	9-5
11-1	省略時設定の論理名テーブル	11-19
11-2	プロセス論理名ディレクトリの省略時設定の論理名	11-20
11-3	プロセス論理名テーブルの省略時設定の論理名	11-21
11-4	システム論理名ディレクトリの省略時設定の論理名	11-21
11-5	システム論理名テーブルの省略時設定の論理名	11-24
11-6	共用論理名テーブルの省略時保護	11-25
11-7	共用論理名の作業に必要な特権またはアクセス・タイプ	11-26
12-1	文字列比較	12-11

12-2	数値比較	12-17
A-1	DEC 各国語文字セット	A-1
A-2	DCL 文字セット	A-11

対象読者

本書は、OpenVMS オペレーティング・システムのすべてのユーザを対象に書かれています。

システム管理者は、効率的なシステム環境を構築または維持するための管理作業を行います。システム管理者、あるいはシステム管理の概念や手順を理解されたい方は、『OpenVMS システム管理者マニュアル』をお読みください。

本書の構成

随所に、コンピュータによる作業を実行するための概念と手順が説明されています。どの章も基本的な事柄の説明に始まり、より複雑な概念や手順へと続いていきます。

概要

以下の章は OpenVMS オペレーティング・システムをはじめて使用する際に役立つ情報を説明しています。

- 第 1 章 — OpenVMS オペレーティング・システムの概要

システムへのログイン、システムからのログアウト、パスワードの変更、およびヘルプの使用の各方法について説明します。

- 第 2 章 — DCL を使用したシステムとの会話

DIGITAL コマンド言語 (DCL) について説明します。

- 第 3 章 — ファイル情報の格納

情報を格納する場所であるファイルについて説明します。また、ファイルの作成、コピー、名前変更、表示、削除、保護、そして印刷について例を挙げながら説明します。

- 第 4 章 — ディレクトリ・ファイルの編成

ファイルを編成または管理するディレクトリについて説明します。

- 第 5 章 — 拡張ファイル指定

ODS-5 を使用しての OpenVMS Alpha システム用 Extended File Specifications 環境について説明します。

- 第 6 章 — ディスクとテープ・ドライブの使用方法

テープやディスクを個人 (プライベート) 用に予約する方法について説明します。グループで共用されているデバイスと異なり、プライベート・デバイスは、システム管理者が設定や管理を行うことはできません。

- 第7章 —Mail を使用して他のユーザと通信する

Mail ユーティリティ (MAIL) について説明します。Mail は、使用しているシステム上、または DECnet for OpenVMS ネットワークで接続されているシステム上の他のユーザと通信を行うためのユーティリティです。この章では、メール・メッセージの例を挙げ、メール・メッセージの受信、送信、返信、転送、そして編成の方法を手順を追って説明します。章末には、Mail コマンドの要約を示します。

テキストの処理とレコードのソート

以下の章では、テキスト・ファイルの編集とレコードのソートについて説明しています。

- 第8章 —EVE エディタによるテキスト・ファイルの編集

会話型テキスト・エディタ EVE について説明します。EVE は、OpenVMS オペレーティング・システムの標準テキスト・エディタです。この章では、EVE を使用して、新しいファイルを作成し編集する方法、および既存のファイルを編集する方法について説明します。章末には、EVE のコマンド要約を示します。

- 第9章 — ファイルのソートとマージ

Sort/Merge ユーティリティについて説明します。Sort/Merge は、1 つまたは複数の入力ファイルの内容をソートしたり、ソートしたファイルをマージするユーティリティです。章末には、Sort/Merge の修飾子の要約を示します。

セキュリティ

以下の章では、セキュリティについて説明しています。

- 第10章 — 資源へのアクセスの制御

保護されたオブジェクトへのアクセスの制御やリモート・システムのデータへのアクセスなど、セキュリティに関する一般的な問題について説明します。

論理名とシンボルの使用方法

以下の章では、論理名とシンボルの使用方法について説明しています。

- 第11章 — デバイスとファイルの論理名定義

ファイル、ディレクトリ、そしてデバイスを表す論理名の作成と使用方法について説明します。システムが作成する論理名の要約も示します。

- 第12章 — シンボル、コマンド、式の定義

コマンド、文字列、そして数値データを表すシンボルの使用方法について説明します。また、シンボルが表す値を算出する方法についても説明します。

プログラミング

以下の章では、プログラムの作成とプログラミング用関数の使用方法について説明しています。

- 第 13 章 — コマンド・プロシーダの概要

コマンド・プロシーダの作成と使用方法についての導入の章です。DCL コマンドとデータ行からなるファイルである、基礎的なコマンド・プロシーダについて説明します。

- 第 14 章 — DCL での拡張プログラミング

複雑なコマンド・プロシーダの作成と使用方法について説明します。

- 第 15 章 — レキシカル関数を使用する情報の取得と処理

レキシカル関数について説明します。レキシカル関数は、コマンド・プロシーダの中で使用され、ユーザのプロセスやシステム環境についての情報を提供します。

プロセスの管理

以下の章では、プロセスの管理について説明しています。

- 第 16 章 — プロセスとバッチ・ジョブ

プロセスについて説明します。プロセスとは、OpenVMS オペレーティング・システムによって作成される環境のことで、ユーザとシステムの会話はプロセスを介して行われます。この章では、サブプロセス、プログラム、そしてバッチ・ジョブを使用する場面とその方法について説明します。

参照情報

以下の参照情報があります。

- 付録 A — 文字セット

DEC の各国語対応文字セットと DCL 文字セットについて説明します。

- 付録 B — コマンド・プロシーダの例

第 13 章、第 14 章、第 15 章で述べたコマンド・プロシーダの概念と実例を示します。

- 用語集

本書で使用される用語を解説します。

関連資料

OpenVMS 製品およびサービスの詳細についてお知りになりたい場合は、弊社の World Wide Web サイトにアクセスしてください。URL は次のとおりです。

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

本書で使用する表記法

本書では、次の表記法を使用しています。

表記法	意味
Ctrl/x	Ctrl/x という表記は、Ctrl キーを押しながら別のキーまたはポインティング・デバイス・ボタンを押すことを示します。
PF1 x	PF1 x という表記は、PF1 に定義されたキーを押してから、別のキーまたはポインティング・デバイス・ボタンを押すことを示します。
Return	例の中で、キー名が四角で囲まれている場合には、キーボード上でそのキーを押すことを示します。テキストの中では、キー名は四角で囲まれていません。 HTML 形式のドキュメントでは、キー名は四角ではなく、括弧で囲まれています。
...	例の中の水平方向の反復記号は、次のいずれかを示します。 <ul style="list-style-type: none">• 文中のオプションの引数が省略されている。• 前出の 1 つまたは複数の項目を繰り返すことができる。• パラメータや値などの情報をさらに入力できる。
.	垂直方向の反復記号は、コードの例やコマンド形式の中の項目が省略されていることを示します。このように項目が省略されるのは、その項目が説明している内容にとって重要ではないからです。
()	コマンドの形式の説明において、括弧は、複数のオプションを選択した場合に、選択したオプションを括弧で囲まなければならないことを示しています。
[]	コマンドの形式の説明において、大括弧で囲まれた要素は任意のオプションです。オプションをすべて選択しても、いずれか 1 つを選択しても、あるいは 1 つも選択しなくても構いません。ただし、OpenVMS ファイル指定のディレクトリ名の構文や、割り当て文の部分文字列指定の構文の中では、大括弧に囲まれた要素は省略できません。
[]	コマンド形式の説明では、括弧内の要素を分けている垂直棒線はオプションを 1 つまたは複数選択するか、または何も選択しないことを意味します。
{ }	コマンドの形式の説明において、中括弧で囲まれた要素は必須オプションです。いずれか 1 つのオプションを指定しなければなりません。
太字	太字のテキストは、新しい用語、引数、属性、条件を示しています。
<i>italic text</i>	イタリック体のテキストは、重要な情報を示します。また、システム・メッセージ (たとえば内部エラー <i>number</i>)、コマンド・ライン (たとえば <i>PRODUCER=name</i>)、コマンド・パラメータ (たとえば <i>device-name</i>) などの変数を示す場合にも使用されます。

表記法	意味
UPPERCASE TEXT	英大文字のテキストは、コマンド、ルーチン名、ファイル名、ファイル保護コード名、システム特権の短縮形を示します。
Monospace type	モノスペース・タイプの文字は、コード例および会話型の画面表示を示します。 C プログラミング言語では、テキスト中のモノスペース・タイプの文字は、キーワード、別々にコンパイルされた外部関数およびファイルの名前、構文の要約、または例に示される変数または識別子への参照などを示します。
-	コマンド形式の記述の最後、コマンド・ライン、コード・ラインにおいて、ハイフンは、要求に対する引数とその後の行に続くことを示します。
数字	特に明記しない限り、本文中の数字はすべて 10 進数です。10 進数以外 (2 進数、8 進数、16 進数) は、その旨を明記してあります。

OpenVMS オペレーティング・システムの概要

OpenVMS は、会話型の仮想メモリ・オペレーティング・システムです。コンピュータにログインしている間は、DIGITAL コマンド言語 (DCL) を使用してシステムと会話できます。DCL を使用するには、ユーザがキーボードからコマンドを入力すると、システムがそのコマンドを読み取って、解釈します。システムはコマンドを実行するか、または、ユーザが入力したコマンドを解釈できない場合はエラー・メッセージを画面に表示します。この章では、OpenVMS オペレーティング・システムと会話するのに必要な、次の基本的な情報について説明します。

- ログイン
- PC からのログイン
- アカウントのパスワードの選択
- 情報メッセージの読み込み
- ログインのタイプとログイン・クラス
- ログインの失敗
- パスワードの変更
- パスワードとアカウントの満了
- パスワードの保護に関するガイドライン
- システム応答の認識方法
- システムのヘルプについて
- システムからのログアウト
- システム・セキュリティを損なわずにログアウトする方法
- ネットワーク

本章で説明するすべてのコマンドの詳しい説明については、『OpenVMS DCL ディクショナリ』およびオンライン・ヘルプを参照してください。

1.1 ログイン

ログインとは、システムへのアクセス権を取得し、自分自身が登録されたユーザであることをシステムに認識させることです。ログインすると、システムはコマンドを入力できる環境を作成します。この環境をプロセスと呼びます。

OpenVMS オペレーティング・システムへのログイン方法、または OpenVMS オペレーティング・システムからのログアウト方法は、システムの設定によって異なります。本章は、OpenVMS オペレーティング・システムへのログインとオペレーティング・システムからのログアウトについての概論です。個々のシステムに固有な手順については、システム管理者に問い合わせてください。

オペレーティング・システムと会話するには、ユーザ・アカウントにログインしなければなりません。アカウントとは、ログイン時にシステムがユーザを識別するための名前または番号のことです。この名前または番号によって、システムは、ユーザのファイルがどこに格納されているのかを確認し、他のファイルに対するアクセス権を判断します。

通常、アカウントの設定はシステム管理者（または、システムの使用を許可する者）が行い、ユーザの必要に応じた特権を与えます。アクセス権と特権の種類によって、ユーザ・アカウントがアクセスできるファイル、イメージ、またはユーティリティが決まります。これらはシステムの性能や他のユーザにも影響を与えるので重要です。

アカウントにアクセスするには、ユーザ名とパスワードを入力する必要があります。通常、システム管理者がユーザ名と初期パスワードを割り当てます。ユーザ名は、システムがユーザを識別し、他のユーザと区別するためのものです。パスワードは、システム保護のためのものです。パスワードを秘密にすることにより、他人のユーザ名でシステム資源を使用できないようにします。

以下にシステムにログインする手順を示します。

手順	タスク
1	システムがユーザ名を入力するプロンプトが表示される。 Username: ユーザ名をタイプしてから Enter を押す。なお、ユーザ名の入力には約 30 秒以内に終える必要がある。30 秒以上経過すると、「タイムアウト」となり、ログイン操作を最初からやり直さなくてはならない。 スクリーンに入力したユーザ名が表示される。 Username: CASEY 続いて、パスワードを要求するプロンプトが表示される。 Password:
2	パスワードをタイプし、Enter を押す。 パスワードは画面に表示されない。このことを「エコーなし」と呼ぶことがある。
3	システム管理者がアカウントをどのように設定したかに応じて、第 2 パスワードを入力しなければならないことがあり、また、自動的に生成されたパスワードを使用しなければならないことがある (第 1.3.4 項を参照)。

1.1.1 ログインの成功

ログインが成功すると、画面の左端にドル記号(\$)が表示されます。ドル記号は、省略時の DCL プロンプトです。これは、システムが使用可能な状態であることを示します。

次に、ログインの成功例を示します。

```
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
$
```

1.1.2 ログイン・エラー

ユーザ名やパスワードの入力を間違えたり、パスワードの期限が満了している
と、User authorization failureというメッセージが表示されて、ログインできません。入力を間違えた場合には、Enterを押して、もう一度入力してください。パスワードの期限が満了している場合には、パスワードを変更しなければなりません。このため、Set Password: プロンプトが自動的に表示されます。この場合のパスワードの変更についての説明は第 1.7 節を参照してください。ログインに関して他に問題がある場合には、使用しているアカウントの設定者にご相談ください。

1.2 PC からのログイン

以前には、モニタとキーボードで構成されるビデオ・ターミナルからホスト・コンピュータに接続していました。OpenVMS オペレーティング・システムが稼動するホスト・コンピュータにすべての処理能力があり、これらのコンピュータは、しばしば、中央のコンピュータ・ルームに設置されました。今日では、専用の計算機能を備えたパーソナル・コンピュータ (PC) またはワークステーションで作業する方が一般的になっています。この状況で、OpenVMS が稼動するホスト・コンピュータに接続するには、ターミナル・エミュレーション・プログラムを利用します。

ターミナル・エミュレーション・プログラムは、TCP/IP ネットワーク、インターネット、またはイントラネット経由で OpenVMS システムに接続します。オペレーティング・システムとの会話は、ターミナル・エミュレーション・プログラムによって提供されるインタフェースを使用して PC のモニタに表示されます。この方法で OpenVMS に接続するには、ターミナル・エミュレーション・プログラムを起動し、接続先のシステムを選択した後、本章の説明に従って OpenVMS オペレーティング・システムにログインします。

1.3 アカウントのパスワードの選択

安全なパスワードを選択するには、次のガイドラインに従ってください。

- パスワードでは、数字と英字の両方を使用する。英字のみの 6 文字のパスワードもかなり安全であるが、英字と数字の両方を使用した 6 文字のパスワードの方がさらに安全である。
- 6 ~ 10 文字のパスワードを選択する。適切な長さのパスワードを使用すれば、安全性はさらに向上する。最大 32 文字までのパスワードを選択できる。
- 辞書にある言葉や自国語の言葉をパスワードとして選択しない。
- 製品名や車のモデルなど、コンピュータ・システムや自分自身に関連する言葉を安易に選択しない。
- 毎回新しいパスワードを選択し、古いパスワードは再利用しない。

システム管理者やセキュリティ管理者は、たとえば 10 文字未満のパスワードや、パスワードの反復を禁止するなど、それ以外に制約を設定することもできます。

次の表は、安全なパスワードと危険性の高いパスワード (他人が簡単に推測できるパスワード) の例を示しています。

安全なパスワード	危険なパスワード
意味のないつづり: aladaskgam eojfuvcue joxtyois	個人的な関わりの強い言葉: 自分の名前 恋人の名前 ペットの名前 住んでいる町の名前 自分の自動車の名前
混合文字列: 492_weid \$924spa zu_\$rags	仕事に関連する用語: 会社名 特殊プロジェクト名 作業グループ名

1.3.1 初期パスワードの取得

通常、自分のアカウントがシステムですでに作成されているかどうかを調べれば、ユーザ・パスワードが必要かどうかわかります。ユーザ・パスワードが有効な場合には、システム管理者は通常、最初のログインのために特定のパスワードを割り当てます。このパスワードはシステムの利用者登録ファイル (UAF) に登録されており、アカウントの使用方法に関するその他の情報もあわせて登録されています。

他の人が簡単に推測できるようなパスワードは望ましくありません。アカウントを作成する人に、推測しにくいパスワードを指定するように依頼してください。パスワードの作成にまったく関与できない場合には、自分の名前と同じパスワードが指定されてしまう可能性があります。その場合には、ログインした後、ただちにパスワードを

変更してください (パスワードとして自分の名前や姓を使用するのはきわめて一般的であるため、セキュリティの観点からは望ましくありません)。

アカウントを作成するときに、パスワードの最小長と、自分で新しいパスワードを選択できるのか、それともシステムから一方的にパスワードが与えられるのかも知っていなければなりません。

1.3.2 初期パスワードの変更

アカウントが作成された後、そのアカウントにただちにログインし、パスワードを変更してください。アカウントの作成後、初めてログインするまでに時間が経過した場合には、他のユーザがアカウントにログインして、システムに損害を与える可能性があります。同様に、パスワードを変更しなかったり、パスワードを変更できない場合には、システムを安全に維持できません。システムがどのような損害を受けるかは、他にどのようなセキュリティ手段がとられているかに大きく左右されます。パスワードの変更の詳細については、第 1.7 節を参照してください。

1.3.3 パスワードの制限事項

システムはパスワードを受け付けることができるかどうかについて、次のことを調べます。

- 新しいパスワードをシステム辞書と自動的に比較する。これは、パスワードが意味のある言葉にならないようにする点で役立つ。
- 古いパスワードのヒストリ・リストを管理しておき、新しいパスワードを 1 つずつこのリストと比較して、古いパスワードが再利用されないようにする。
- システム管理者が UAF レコードに指定するパスワードの最小長に従うようにする。

システム辞書に登録されているパスワード、以前に使用したことのあるパスワード、UAF に指定されているパスワードの最小長未満のパスワードは指定できません。

1.3.4 使用するパスワードのタイプ

OpenVMS オペレーティング・システムでは、複数のパスワード・タイプが認識されます。

- ユーザ・パスワード

ほとんどのアカウントで必要。ユーザ名を入力した後、パスワードを要求するプロンプトが表示される。アカウントで 1 次パスワードと 2 次パスワードの両方が必要な場合には、2 つのパスワードを入力しなければならない。

- システム・パスワード

特定のターミナルへのアクセスを制御し、セキュリティ管理者が設定する。システム・パスワードは通常、ダイヤルアップ回線やパブリック・ターミナル・ラインなど、システムの不当な使用の対象となるターミナルへのアクセスを制御するのに必要である。

- 1 次パスワード

1 次パスワードと 2 次パスワードの両方を必要とするアカウントで最初に入力しなければならないパスワード。

- 2 次パスワード

1 次パスワードと 2 次パスワードの両方が必要なアカウントで 2 番目に入力しなければならないパスワード。2 次パスワードは、ユーザ・アカウントで補足的なセキュリティ・レベルを提供する。通常、1 次ユーザは 2 次パスワードを知らない。したがって、スーパーバイサなどが 2 次パスワードを指定しなければならない。アプリケーションによっては、アカウントの使用中也スーパーバイザがそのまま残ることがある。したがって、2 次パスワードはログインやログイン後の処理を制御する上で便利である。

2 次パスワードを使用すると、時間がかかり、不便である。したがって、最大のセキュリティ要件を必要とするシステムでのみ使用する。二重パスワードを使用した方がよいアカウントの例としては、通常のアクセス制御を迂回して、データベースを緊急に修復するアカウントがある。

1.3.5 システム・パスワードの入力

使用できる 1 つ以上のターミナルにログインするために、システム・パスワードを指定しなければならないかどうかは、セキュリティ管理者が決定します。現在のシステム・パスワード、パスワードの変更頻度、変更した場合の新しいシステム・パスワードの入手方法については、システム管理者にご質問ください。

システム・パスワードを指定するには、次の操作を実行します。

手順	タスク
1	ターミナルが認識文字を応答するまで、Enter キーを押す。認識文字は通常、ベルである。
2	システム・パスワードをタイプし、Enter を押す。この例に示すように、プロンプトは表示されず、入力した文字も表示されない。正しいシステム・パスワードを指定しなかった場合でも、そのことは通知されない(したがって、そのターミナルでシステム・パスワードが必要であることがわからない場合、最初はシステムが誤動作しているかのように見える)。システムから応答がない場合には、誤ったパスワードを入力したものと解釈し、正しいパスワードを再入力する。
3	正しいシステム・パスワードを入力すると、システム・アナウンスメント・メッセージが表示され(設定されている場合)、その後に Username: プロンプトが表示される。次の例を参照。

MAPLE - A member of the Forest Cluster
Unauthorized Access is Prohibited

Username:

1.3.6 2 次パスワードの入力

2 次パスワードを使用しなければならないかどうかは、アカウントの作成時にセキュリティ管理者が設定します。アカウントで 1 次パスワードと 2 次パスワードが必要な場合には、ログイン時に 2 つのパスワードを指定しなければなりません。セキュリティ管理者が UAF に指定したパスワードの最小長は、両方のパスワードに適用されます。

1 つのパスワードでログインできる場合と同様に、システムはログイン全体に対して特定の時間を割り当てます。その時間内に 2 次パスワードを入力しなかった場合には、ログインは時間切れになります。

次の例は、1 次パスワードと 2 次パスワードを必要とするログインを示しています。

```
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
$
```

1.3.7 各種アカウントのパスワードの要件

OpenVMS システムでは、4 種類のユーザ・アカウントを使用できます。

- ユーザまたはセキュリティ管理者が定期的に変更するパスワードによって保護されるアカウント。このアカウントはもっとも一般的なタイプである。
- 常にパスワードを必要とするが、ユーザがパスワードを変更できないアカウント。セキュリティ管理者はパスワードをロックすることにより (UAF で LOCKPWD フラグを設定する)、パスワードに対するすべての変更を制御する。
- 制限付きアカウントでは、システムの使い方を制限し、場合によってはパスワードを要求する。
- オープン・アカウントでは、パスワードは必要ない。オープン・アカウントにログインする場合には、パスワードを要求するプロンプトは表示されないため、入力する必要がない。ただちにコマンドの入力を開始できる。オープン・アカウントでは、誰もがシステムにアクセスできるため、セキュリティ要件が最低のシステムでのみ使用するようにしなければならない。

1.4 情報メッセージの読み込み

コンピュータに直接接続されているターミナルからログインする場合には、次の例に
しめすように、OpenVMS システムは情報システム・メッセージを表示します。

```
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

$
```

以下は、例についての説明です。

- 1 アナウンスメント・メッセージはノード (および場合によっては OpenVMS Cluster 名) を識別する。また、未登録ユーザに対して、不法なアクセスが禁止されていることも警告する。このメッセージの表示形式と内容は、システム管理者またはセキュリティ管理者が制御できる。
- 2 プロセス切断メッセージは、最後にログインに成功した後、プロセスが切断されたが、まだ存在することを示している。この場合、古いプロセスに再接続すれば、プロセスを切断される前の状態に戻すことができる。

システムは次の条件が満足される場合にのみ、切断メッセージを表示する。

- 割り込みが発生したターミナルが仮想ターミナルとして設定されていること。
- ターミナルが切断可能なターミナルとして設定されていること。
- 最近のセッションで、ログアウトする前に、そのターミナルを通じた CPU への接続が切断されたこと。

一般に、この機能が使用可能であっても、システム・セキュリティに問題はないため、セキュリティ管理者はユーザが再接続できるように設定する。ただし、ターミナルでこの設定を変更し、システムで仮想ターミナルの使用を禁止すれば、この機能を禁止できる (仮想ターミナルの設定と再接続についての説明は、『OpenVMS システム管理者マニュアル』を参照)。

- 3 ウェルカム・メッセージは、実行中の OpenVMS オペレーティング・システムのバージョン番号と、ログインしているノードの名前を示す。システム管理者は別のメッセージを選択したり、メッセージがまったく表示されないように設定できる。
- 4 最後に成功した会話型ログイン・メッセージには、ローカル、ダイアルアップ、リモート・ログインの最後のログイン完了時間が示される (これらのタイプのいずれかを親として持つサブプロセスからのログインはカウントされない)。
- 5 最後に成功した非会話型ログイン・メッセージには、最後の非会話型 (バッチまたはネットワーク) ログインの完了時間が示される。
- 6 ログイン失敗メッセージの数は、ログインの失敗回数を示す (ログインの失敗としてカウントされるのは、誤ったパスワードの入力だけである)。注意を促すために、メッセージを表示した後、ベルが鳴る。
- 7 新規メール・メッセージは、未開封メール・メッセージが到着しているかどうかを示す。

1.4.1 メッセージの無表示

セキュリティ管理者は、ノード名とオペレーティング・システム識別情報を含むウェルカム・メッセージとアナウンスメント・メッセージを表示しないように設定できます。ログイン・プロシージャはオペレーティング・システムに応じて異なっているため、この情報が表示されなければ、ログインが困難になります。

最終ログイン成功メッセージと失敗メッセージは省略可能です。セキュリティ管理者はこれらをまとめて有効にしたり、無効にすることができます。中レベルまたは高いレベルのセキュリティが必要とされるシステムでは、これらのメッセージを表示すれば、不法な侵入 (ブレイクイン) を示すことができます。さらに、システムがログインを監視していることを示すことができるため、これらのメッセージは不当なユーザによるアクセスを抑制する効果があります。

1.4.2 成功ログイン・メッセージ

ログインするたびに、システムは最後に成功したログインとログイン失敗の回数を再設定します。アカウントに会話方式でアクセスするときに、ログイン時に誤ったパスワードを指定しなければ、最後に成功した非会話型ログイン・メッセージとログイン失敗メッセージは表示されません。

1.5 ログインのタイプとログイン・クラス

ログインは会話型と非会話型に分類できます。会話型ログインの場合には、ユーザ名とパスワードを入力します。非会話型ログインでは、システムがユーザの識別と認証を実行します。ユーザ名とパスワードを要求するプロンプトは表示されません。

会話型ログインと非会話型ログインの他に、OpenVMS オペレーティング・システムではさまざまなログイン・クラスも認識されます。システムにログインする方法によって、どのログイン・クラスに属しているかが決定されます。ログイン・クラス、および曜日と時刻によって、システム管理者はシステムへのアクセスを制御します。

1.5.1 会話型ログイン

会話型ログインには、次のログイン・クラスがあります。

- ローカル

中央のプロセッサに直接接続されているターミナル、または中央プロセッサと直接通信するターミナル・サーバからログインします。

- ダイアルアップ

モデムと電話回線を使用してコンピュータ・システムとの接続を確立するターミナルにログインします。システムで使用しているターミナルに応じて、最初に追加操作を実行しなければならないことがあります。必要な操作については、各システムのセキュリティ管理者にご質問ください。

- リモート

DCL の SET HOST コマンドを入力して、ネットワークを介してノードにログインします。たとえば、HUBBUB というリモート・ノードにアクセスするには、次のコマンドを入力します。

```
$ SET HOST HUBBUB
```

HUBBUB ノードのアカウントにアクセスできる場合には、ローカル・ノードからそのアカウントにログインできます。HUBBUB ノードの機能にアクセスすることはできますが、物理的にはローカル・ノードに接続された状態です。

リモート・セッションについての詳しい説明は、第 1.12.2 項を参照してください。

1.5.2 非会話型ログイン

非会話型ログインには、ネットワーク・ログインとバッチ・ログインがあります。

- ネットワーク・ログイン

ディレクトリの内容を表示したり、別のノードにあるディレクトリに格納されたファイルをコピーするなど、リモート・ノードでネットワーク・タスクを開始するときは、システムはネットワーク・ログインを実行します。この場合、現在のシステムとリモート・システムの両方が同じネットワーク内のノードでなければなりません。ファイル・システムにターゲット・ノードを指定し、アクセス制御文字列も指定します。この文字列には、リモート・ノードのユーザ名とパスワードが含まれます。

たとえば、PARIS というリモート・ノードにアカウントを持つ GREG というユーザが次のコマンドを入力すると、ネットワーク・ログインが実行されます。

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

このコマンドは WORK2 ディスクのパブリック・ディレクトリ内のすべてのファイルのリストを表示します。また、パスワードが 8G4FR93A であることもわかります。同じタスクをもっと安全に実行するには、PARIS ノードで代理アカウントを使用します。代理ログインの例については、第 10.5.3 項を参照してください。

- バッチ・ログイン

キューに登録したバッチ・ジョブを実行する場合には、バッチ・ログインが実行されます。ジョブを構築するための許可は、ジョブをキューに登録するときに決定されます。システムがジョブを実行するための準備をするときに、ジョブ・コントローラはアカウントにログインする非会話型プロセスを生成します。ジョブがログインする場合には、パスワードは必要ありません。

1.6 ログインの失敗

ログインはさまざまな理由で失敗します。たとえば、パスワードの 1 つが変更されたり、アカウントの有効期限が満了した場合や、ネットワークを介して、またはモデムからログインしようとしたときに、その操作が登録されていないこともあります。次の表は、ログインの失敗の一般的な理由を示しています。

失敗の症状:	理由:
ターミナルから応答がない	ターミナルに欠陥があるか、システム・パスワードを必要とするターミナルであるか、またはターミナルに電源が投入されていない。
ターミナルから応答がない	システムがダウンしている。
システム・パスワードを入力したが、ターミナルから応答がない。	システム・パスワードが変更された。
システム・メッセージ: "User authorization failure"	ユーザ名またはパスワードを入力ミスした。 アカウントまたはパスワードの有効期限が満了した。
"Not authorized to log in from this source"	特定のログイン・クラス (ローカル、ダイヤルアップ、リモート、会話型、バッチ、ネットワーク) が禁止されている。
"Not authorized to log in at this time"	この曜日またはこの時刻にログインが許可されていない。
"User authorization failure" (かつ、認識されるユーザ障害が発生していない)	ユーザ名を使用してターミナルからシステムへの不法侵入が試みられたため、システムはそのユーザ名によるそのターミナルへのすべてのログインを一時的に禁止した。

この後の項では、ログインの失敗の理由について詳しく説明します。

1.6.1 システム・パスワードを必要とするターミナル

使用するターミナルでシステム・パスワードが必要なときに、そのことを知らないと、ログインできません。システム・パスワードを入力するまで、ログインはすべて失敗します。

システム・パスワードがわかっている場合には、第 1.3.5 項で説明した操作を実行してください。その操作を実行してもログインできない場合には、システム・パスワードが変更されている可能性があります。システム・パスワードを知らず、そのことが問題であると考えられる場合には、別のターミナルにログインするか、または新しいシステム・パスワードを要求してください。

1.6.2 ログイン・クラスの制限

UAF レコードで禁止されているクラスのログインを試みても、ログインは成功しません。たとえば、セキュリティ管理者がネットワークを介したログインを制限している場合には、ネットワーク・ログインを試みると、このソースからログインする権限がないことを示すメッセージが出力されます。

セキュリティ管理者は、ローカル、リモート、ダイアルアップ、バッチ、ネットワークの各クラスでのログインを制限している可能性があります。

1.6.3 勤務時間に関する制限

勤務時間に関する制限によりログインできないこともあります。システム管理者またはセキュリティ管理者は、時刻や曜日をもとにシステムへのアクセスを制御することがあります。このような制限はログイン・クラスに影響します。セキュリティ管理者は、同じ作業時間制限をすべてのログイン・クラスに適用したり、ログイン・クラスによって異なる制限を設定することもあります。

該当するログイン・クラスに対して禁止されている時刻にログインしようとしてもログインできません。この場合、この時刻にログインする権限がないことがユーザに通知されます。

1.6.4 勤務時間制限中でのバッチ・ジョブ

勤務時間制限がバッチ・ジョブに適用される場合には、許可されている作業時間以外に実行する予定のジョブは、キューに登録しても実行されません。また、次の作業時間にこのようなジョブをシステムが自動的にキューに再登録することはありません。同様に、何らかのジョブを開始して、許可されている時間外にそのジョブを実行しようとしても、ジョブ・コントローラは、割り当てられた勤務時間が終了すると、未完のジョブを強制終了します。このようなジョブの終了方法は、すべてのジョブに適用されます。

1.6.5 ダイアルアップ・ログインでの失敗

セキュリティ管理者は、ダイアルアップ・ログインで、接続が自動的に切断されるまでの間に、ユーザがパスワードを入力できる回数を制限できます。

ログインが失敗しても、指定された回数に到達していない場合には、Enter キーを押してもう一度ログインを試みてください。ログインが成功するまで、または制限回数に到達するまで、この操作を繰り返すことができます。接続が切断された場合には、アクセス・ラインを再ダイヤルして、もう一度試みてください。

ダイアルアップ・ログインの回数を制限するのは、権限のないユーザがパスワードを知ろうとしても、エラーが発生してあきらめざるを得なくするためです。ダイアルアップ回線では、権限がなくても匿名での操作が可能です。もちろん、各ダイアルアップ回線ごとにログインの試行回数を制限しても、この種のシステム侵入の試みがまったくなくなるわけではありません。この回数制限は、何回もダイヤルしてログインしようとする不法アクセス者からシステムを保護することを目的にしています。

1.6.6 システム侵入回避プロシージャ

同じターミナルで同じユーザ名によってログインしようとしても、何回も失敗すると、システムは侵入の試みが進行中であるかのように応答します。つまり、誰かがこのユーザ名を使用してシステムに対して非合法的なアクセスを試みていると判断します。

セキュリティ管理者は、自分の裁量で侵入回避手段をシステムのすべてのユーザに対して有効に設定できます。セキュリティ管理者は、所定の時間内に何回のパスワード試行を許可するかを制御します。侵入回避手段が一度起動されると、指定された時間内は、正しいパスワードを使用しても、ターミナルにログインできません。再度ログインを試みるまでにどのくらい待たなくてはならないか、あるいは別のターミナルを使用してログインを試みた方がよいのかは、セキュリティ管理者に質問してください。

侵入回避手段によってログインが防止されているだけで、個人的にログインが失敗したわけではないと考えられる場合には、ただちにシステム管理者に連絡してください。同時に、もう一度ログインを試みて、最後のログイン以降のログイン失敗回数を示すメッセージを確認し、侵入の試みについての推測が正しいかどうかを調べてください。通常はログイン・メッセージが表示されないシステムの場合、セキュリティ管理者は Authorize ユーティリティ (AUTHORIZE) を使用して、UAF レコードの中のデータを調べます。誰かが別のターミナルでログインを試みているかどうかは、プロンプトがどのように表示されるかによってわかります。

1.7 パスワードの変更

定期的にパスワードを変更すれば、システム・セキュリティを向上できます。パスワードを変更するには、DCL の SET PASSWORD コマンドを使用します。

システム管理者は、ユーザが自分で自分のパスワードを選択できるように設定でき、また、パスワードを変更するときに自動的なパスワード・ジェネレータを使用するように要求することもできます。ユーザが自分でパスワードを選択できる場合には、パスワードが長さやその他の制限に従っていなければなりません (第 1.3.3 項を参照)。

一定期間内にパスワードを変更できる回数は制限されていません。

たとえば、選択したパスワードが短すぎる場合には、システムは次のメッセージを表示します。

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

1.7.1 ユーザ自身によるパスワードの選択

システム管理者が自動的なパスワード・ジェネレータの使用を要求していない場合には、SET PASSWORD コマンドは新しいパスワードの入力を要求するプロンプトを表示します。その後、次に示すように、確認のためにもう一度新しいパスワードを入力するように要求するプロンプトが表示されます。

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

新しいパスワードとして同じパスワードを 2 回入力しなかった場合には、パスワードは変更されません。同じパスワードを 2 回入力した場合には、画面に何も表示されません。このコマンドはパスワードを変更し、Enter で DCL プロンプトになります。

セキュリティ管理者がパスワード・ジェネレータの使用を要求していない場合でも、パスワード・ジェネレータを使用すれば、システムのセキュリティを向上するのに役立ちます。

1.7.2 生成されたパスワードの使用方法

セキュリティ管理者がシステムで自動的に生成されるパスワードを使用しなければならないと判断した場合には、DCL の SET PASSWORD コマンドを入力したときに、パスワードのリストが表示されます (システムで自動的に生成されたパスワードを使用するように設定されていない場合には、SET PASSWORD コマンド

に/GENERATE 修飾子を指定することにより、自動的に生成されたパスワードを使用できます)。覚えやすいように普通の言語によく似た文字シーケンスになっていますが、外部の人間がパスワードを推測するのは困難です。システムで生成されるパスワードは長さが一定でないため、さらに推測が困難になっています。

注意

パスワード・ジェネレータでは、基本的な音節規則を使用してワードを生成しますが、実際の言語に関する知識があるわけではありません。したがって、偶然に不快な言葉が生成されてしまう可能性があります。

次の例では、文字がランダムに並んだパスワードのリストが自動的に生成されています。ユーザのパスワードの最小長は UAF レコードで 8 文字に設定されています。

```
$ 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
```

以下は、例についての説明です。

- 1 ユーザは古いパスワードを正しく指定し、Enter キーを押す。
- 2 システムは 8 ~ 10 文字の長さの 5 つのパスワードを表示する。通常、発音しやすいパスワードが覚えやすいパスワードである。したがって、そのパスワードが最適である。

OpenVMS VAX システムでは、各パスワードの右側に、同じ単語を音節に区切った表現が表示される (この例を参照)。
- 3 新しいパスワードを要求するプロンプトに対して Enter キーを押すと、新しいリストを要求できることをユーザに示している。
- 4 ユーザは最初に表示された 5 つのパスワードから 1 つを入力して、Enter キーを押す。
- 5 システムは、このパスワードが自動的なパスワード・ジェネレータによって作成されたパスワードであることを認識し、確認プロンプトを応答する。ユーザは新しいパスワードを再入力し、Enter を押す。
- 6 システムはパスワードを変更し、DCL プロンプトに戻る。

1.7.3 生成されたパスワード: 欠点

生成されたパスワードの使用に当たっては、次の2つの欠点があります。

- ユーザが選択したパスワードを忘れてしまう可能性があるという欠点があります。しかし、表示されたどのパスワードも気に入らない場合や、どのパスワードも簡単に覚えられないと思われる場合には、別のリストを要求できます。
- このコマンドを入力したときに、新しいパスワードの候補が表示されてしまうことです。アカウントを保護するためには、誰にも知られないようにパスワードを変更しなければなりません。ビデオ・ターミナルで変更する場合には、コマンドが完了した後、画面をただちに消去してください。印刷ターミナルを使用する場合には、ハードコピー出力をすべて適切に廃棄してください。

このようにしてもパスワードを保護できなかったことが後で判明した場合には、ただちにパスワードを変更してください。各システムの方針に従って、あるいはアカウントのセキュリティが侵された時間の長さからユーザが必要であると判断すれば、自分のアカウントを通じてシステムのセキュリティ侵害が起こりそうであることを、システム管理者に知らせてください。

1.7.4 2次パスワードの変更

2次パスワードを変更するには、DCLのSET PASSWORD/SECONDARY コマンドを使用します。古い2次パスワードと新しい2次パスワードを指定するように要求するプロンプトが表示されます。これは1次パスワードを変更するときと同じ操作です。2次パスワードを削除するには、新しいパスワードと確認パスワードを要求するプロンプトが表示されたときに、Enter キーを押します。

1次パスワードと2次パスワードは別々に変更できますが、どちらのパスワードも有効期限が同じであるため、同じ頻度で変更してください。

1.7.5 ログイン時のパスワードの変更

現在のパスワードの有効期限が満了していなくても、ユーザ名とともに/NEW_PASSWORD 修飾子を指定すれば、システムにログインするときにパスワードを変更できます。ユーザ名の後に/NEW_PASSWORD 修飾子を入力した場合には、ログインの直後に新しいパスワードを設定するように要求するプロンプトが表示されます。

次の例は、ログイン時にパスワードを変更する方法を示しています。

```
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:
```

1.8 パスワードとアカウントの満了

システム管理者は、パスワードまたはアカウント自体が特定の日に自動的に満了するように、アカウントを設定できます。パスワード満了期限が設定されていると、ユーザは定期的にはパスワードを変更しなければならないため、システムのセキュリティが向上します。アカウント満了期限は、アカウントを必要な間だけ使用可能にしたいときに便利です。

1.8.1 満了したパスワード

パスワードの満了期限を設定すると、満了日の 5 日前に警告メッセージが表示され、その後もログインするたびに警告メッセージが表示されます。このメッセージは新規メール・メッセージのすぐ下に表示され、注意を促すためにベルも鳴ります。メッセージには、パスワードが満了することが次のように表示されます。

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

満了前にパスワードを変更しなかった場合には、ログイン時に次のメッセージが表示されます。

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

新しいパスワードを要求するプロンプトが表示されますが、自動パスワード生成機能が有効に設定されている場合には、リストから新しいパスワードを選択するように要求されます。ここで Ctrl/Y を押すと、ログインを強制終了できます。次にログインしようとしたときに、パスワードを変更するように要求するプロンプトが再度表示されます。

1.8.2 2 次パスワードを使用している場合

アカウントで 2 次パスワードを使用している場合には (第 1.3.4 項を参照)、2 次パスワードは 1 次パスワードと同時に満了します。そのとき、2 つのパスワードをどちらも変更するように要求するプロンプトが表示されます。1 次パスワードを変更し、2

次パスワードを変更する前に、Ctrl/Y を押した場合には、ログインは失敗します。その場合、システム・パスワードの変更は記録されません。

1.8.3 パスワードを変更しなかった場合

システム管理者がユーザに対して、ログイン時に満了したパスワードを変更するように強制していない場合には、パスワードの満了時にログインすると、最終警告メッセージが表示されます。次の例を参照してください。

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

この時点で、パスワードを変更しないか、ユーザ・パスワードを変更する前にシステムで障害が発生すると、二度とログインできなくなります。その場合には、再度アクセスする方法についてシステム管理者にご相談ください。

1.8.4 満了したアカウント

一定期間だけ特定の目的でアカウントが必要な場合には、アカウントの作成者がアカウントの有効期限を指定できます。たとえば、大学で使用する学生のアカウントは、1 学期ごとに権限を与えるのが通常です。

満了したアカウントは自動的にログインを拒否します。アカウントが満了する前に警告メッセージは表示されません。したがって、アカウントの有効期限を前もって知っておくことが重要です。アカウントの満了期限は UAF レコードに格納されており、SYSPRV 特権またはそれと同等の権限を持つユーザ (通常はシステム管理者またはセキュリティ管理者) が OpenVMS Authorize ユーティリティ (AUTHORIZE) を使用しなければ、アクセスしたり、表示したりできません。

アカウントが満了した後、次にログインしようとする、登録障害メッセージが表示されます。満了期限の延長が必要な場合には、各システムで定義されている手順に従ってください。

1.9 パスワードの保護に関するガイドライン

正しいパスワードの使用に関して、非合法的なシステム・アクセスを追跡してみると、パスワードの所有者が誤ってパスワードを公開してしまっていることがよくあります。パスワードは誰にも教えないことが何より重要です。

次のようにすれば、パスワードを保護できます。

- 簡単には推測できない長いパスワードを選択する。辞書にあるような自国語の言葉は使用しないようにする。また、パスワードに数字も含むことを検討する。あるいは、パスワードの自動生成機能を使用する。

- パスワードは絶対に書き留めておかない。
- 自分のパスワードは絶対に他人に教えない。パスワードが別のユーザに知られた場合には、ただちに変更する。
- 電子メール・メッセージの本文も含めて、パスワードをどのファイルにも記録しない。(他人がパスワードを知らせてきた場合には、ただちにその情報を削除する。)

実際のパスワードと組み合わせて表示される文字列を見れば、ファイルからパスワードを簡単に検索できる。たとえば、アクセス制御文字列でユーザ名とパスワードの後には、必ず二重引用符と2つのコロンの("::")が続く。したがって、システムに侵入しようとする人は、保護の不十分なファイルからこの文字列を検索することにより、パスワードを知ることができる。また、テキスト・ファイルで“password”という単語を使用することにより、パスワードが知られてしまう可能性もある。次の例を参照。

My password is GOBBLEDYGOOK.

- 異なるシステムのアカウントに対して同じパスワードを使用しないようにする。
未登録のユーザは、アカウントのあるすべてのシステムで同じパスワードを試みる可能性がある。最初にパスワードを公表したアカウントには重要な情報がほとんどない場合でも、別のアカウントでさらに詳しい情報や特権を得れば、最終的にはきわめて大きなセキュリティ侵害になる恐れがある。
- すでに電源が投入されているターミナルにログインする場合には、その前に Break キーを押して安全ターミナル・サーバ機能を起動する(有効な場合)。これは特に、公共のターミナル・ルームで作業する場合に注意しなければならない。
- パスワードは3～6か月ごとに変更する。パスワードの共用は望ましくない。パスワードを共用する場合には、毎月変更することが必要である。
- 何らかの理由でパスワードが外部に知られたと考えられる場合には、ただちにパスワードを変更する。このような場合には、セキュリティ管理者に報告しなければならない。
- 席を外す前にターミナルからログアウトする。
許可されていないユーザがパスワード盗用プログラムのロードなどの悪意ある目的でターミナルを使用する恐れがある。
- 最後のログイン・メッセージを定期的に確認する。ログインの失敗回数に異常がないか注意する。ログイン時に異常な失敗が見られる場合は、ただちにパスワードを変更して、セキュリティ管理者に報告しなければならない。

1.10 システム応答の認識方法

コマンドを入力すると、次に示す 1 つ以上の方法で応答があります。

- コマンドを実行する。通常、入力したコマンドが正しく実行されている場合には、システム・プロンプト (省略時の設定ではドル記号) に戻る。
- コマンドを実行してから、実行内容をメッセージで知らせる。
- 正しく実行できない場合には、エラーが生じたことを知らせる。
- ユーザが値を入力しなかった場合、省略時の値を与えて実行する。

1.10.1 省略時のアクション

省略時の値とは、ユーザが指定しなかったときに、オペレーティング・システムが提供する値のことです。たとえば、PRINT コマンドの修飾子としてコピー部数を指定しないと、省略時の値 1 が使用されます。オペレーティング・システムは、コマンド修飾子やパラメータなど、いくつかのエリアに省略時の値を提供しています。個々のコマンドでオペレーティング・システムが使用する省略時の値については、『OpenVMS DCL デクショナリ』の各コマンドの説明を参照してください。

1.10.2 情報システム・メッセージ

コマンドには、実行内容についての情報をシステム・メッセージに表示するものもあります。たとえば、PRINT コマンドを使用すると、印刷ジョブに割り当てられたジョブ識別番号と、そのジョブが登録されている印刷キューの名前が表示されます。

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

すべてのコマンドが情報メッセージを表示するわけではありません。通常、コマンドが正しく実行された場合にはドル記号のプロンプトに戻ります。コマンドが正しく実行されなかった場合には、常に 1 つ以上のエラー・メッセージが表示されます。

1.10.3 システム・エラー・メッセージ

コマンドを間違えて入力すると、次の例に示すように、エラー・メッセージと、正しいコマンド文字列を求めるプロンプトが表示されます。

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

コード DCL-W-IVVERB は 3 つの部分からなり、次の情報が含まれます。

DCL-W-IVVERB の各部の意味:

DCL	エラーを戻した OpenVMS 機能や構成要素名。この例では、メッセージは省略時のコマンド・インタプリタ DCL から出されたもの。
W	警告を示す重大度。重大度には、この他に S (成功)、I (情報)、E (エラー)、F (回復不可能なエラーまたは重大エラー) がある。
IVVERB	メッセージのタイプ。メッセージは、『OpenVMS System Messages and Recovery Procedures Reference Manual』のニモニック IVVERB、または Help Message ユーティリティ (MSGHLP) (第 1.11.3 項を参照) を使用することによって識別できる。

要求した機能を実行できない場合も、コマンド実行時にエラー・メッセージが表示されます。たとえば、PRINT コマンドは正しく入力したけれども、指定したファイルが存在しない場合、次のようなエラー・メッセージが表示されます。

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

最初のメッセージは PRINT コマンドから出されたもので、指定されたファイルをオープンできないという意味です。2 番目のメッセージは最初のメッセージの理由で、「ファイルが見つからない」という意味です。RMSは、OpenVMS ファイル処理機能であるレコード管理サービスのことで、通常、ファイル処理に関連するエラー・メッセージは OpenVMS の RMS のメッセージです。

1.10.4 現在のプロセスのチェック

プロセスが期待どおりに進んでいないように思われるときには、Ctrl/T を押します。Ctrl/T は、現在のプロセスについての統計情報を 1 行に表示します。統計情報には、ノード名、ユーザ名、現在の時刻、現在のプロセス、中央処理装置 (CPU) の使用率、ページ・フォルト数、入出力処理レベル、CPU 特有のページ番号にリストするメモリの使用率があります。

会話形式のターミナル・セッション中に Ctrl/T を押すと、現在実行中のコマンド、コマンド・プロシージャ、またはイメージに割り込んで統計情報を表示します。Ctrl/T は画面上の文字出力を中断しますが、プロシージャや編集セッションには影響を及ぼしません。たとえば、ノード GREEN のユーザ MCCARTHY が EVE エディタを使用しているときに Ctrl/T を押すと、次のような行が EVE のメッセージ・ウィンドウに表示されます。

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

画面を再表示する場合には、Ctrl/W を押します。

省略時の設定では、Ctrl/Tは無効です。システムの実行中にCtrl/Tを押しても、統計情報が表示されない場合には、DCLのSET CONTROL=Tコマンドを使用してCtrl/Tを有効に設定できます。DCLレベル(ドル記号(\$)プロンプトが表示されるレベル)でこのコマンドを入力し、Ctrl/Tを再度押してください。SET NOCONTROL=Tなどのコマンドやプログラムを使用して無効に設定しない限り、プロセスが終了するまで、Ctrl/Tは有効です。Ctrl/Tを押すことによって画面に情報を表示するには、ターミナルをBROADCASTモードに設定しなければなりません。BROADCASTモードは(MAILやREPLYによって発行されるような)ブロードキャスト・メッセージの受信を可能にするかどうかを制御します。ターミナルをBROADCASTモードに設定するには、DCLプロンプトに対してDCLのSET TERMINAL/BROADCASTコマンドを入力します。

1.11 システムのヘルプについて

オペレーティング・システムにログインすると、HELPコマンドを使用して、システムやコマンドについての情報を得ることができます。第1.11.3項に示すように、HELP/MESSAGEコマンドを入力しても、システム・メッセージについてのヘルプを得られます。

1.11.1 オンライン・ヘルプの使用方法

次に、OpenVMSコマンドとユーティリティについてのヘルプを得る手順を示します。

手順	タスク
1	DCLプロンプトに対してHELPを入力してからEnterを押す。 トピックのリストが表示された後、Topic?プロンプトが表示される。
2	いずれかのトピックについての情報が必要な場合には、プロンプトの後にトピック名を入力してからEnterを押す。
3	いずれかのサブトピックについての情報が必要な場合には、プロンプトの後にサブトピック名を入力してからEnterを押す。 指定したサブトピックについての情報が表示される。
4	SHOW USERSトピックとサブトピックのリストを再表示するには、SHOW USERS Subtopic?プロンプトに対して疑問符(?)を入力する。リストされたすべてのサブトピックを読みたいときには、アスタリスク(*)を入力する。
5	別のトピックについての情報が必要な場合には、Enterを押す。Topic?プロンプトが表示される。
6	HELPを終了する場合には、DCLプロンプトに戻るまでEnterを(何回か)押す。

次の例は、SHOW USERSコマンドについてのヘルプを見るためのコマンドを示しています。


```
$ 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?
$
```

1.11.2 指定コマンドのヘルプを見る

情報が必要なコマンドがわかっている場合には、HELP の後にコマンド名を入力します。たとえば、SHOW USERS コマンドについてのヘルプが必要な場合は、次のコマンドを入力します。

```
$ HELP SHOW USERS
```

どのコマンドやシステム・トピックを指定するかわからない場合には、HELP コマンドのパラメータとして HINTS と入力します。HINTS テキストには、どんなタスクをするときには、どのコマンドを使用するか、またはどのようなシステム情報が必要かが説明されています。

『OpenVMS DCL デクシヨナリ』に HELP コマンドについての詳しい説明があります。

1.11.3 システム・メッセージについてのヘルプ

システム・メッセージについてのオンライン・ヘルプを利用する場合には、Help Message ユーティリティ (MSGHLP) を使用します。最後のコマンドがどのように実行されたかを知りたいときには、次のように入力します。

```
$ HELP/MESSAGE
```

メッセージ識別子やメッセージ・テキストの中の単語を指定すれば、特定のメッセージについての情報も表示できます。たとえば、次のように入力します。

```
$ HELP/MESSAGE BADACP
```

メッセージ状態コードを入力すれば、メッセージとその説明も表示できます。

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

メッセージ状態コードがわからない場合には、SHOW SYMBOL コマンドと\$STATUS グローバル・シンボルを入力することにより表示できます。次の例を参照してください。

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

Help Message ユーティリティを使用すると、メッセージ・データベースをユーザ独自のメッセージで更新したり、既存のメッセージ説明にコメントを追加したりできます。メッセージ・データベースからメッセージの一部分を取り出して、自分に合わせてカスタマイズしたメッセージ・ドキュメントを作成したり印刷したりすることもできます。Help Message 機能の使用方法についての詳細は、『OpenVMS System Messages: Companion Guide for Help Message Users』を参照してください。

1.12 システムからのログアウト

システムを使用し終わったら、必ずログアウトしてください。これは、登録されていないユーザがアカウントやシステムにアクセスできないようにするためです。また、ログアウトすると、必要がなくなった資源を他のユーザが使用できるようになるので、システム資源も無駄になりません。

ログアウトするには、DCL プロンプトに対して LOGOUT コマンドを入力します。

```
$ LOGOUT
```

システムは次のようなメッセージを表示し、ログアウトしたことの確認メッセージを表示します。

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

システムからログアウトできるのは、DCL プロンプト(\$)が出ているときだけです。プログラムをコンパイルしていたり実行しているとき、テキスト・エディタ (EDT や EVE など) を使用しているとき、ユーティリティ (Mail など) を実行しているときには LOGOUT コマンドは入力できません。まず、プログラム、エディタ、ユーティリティを終了し、システムが DCL プロンプトを出してはじめて、ログアウトできます。

1.12.1 アカウント情報の取得

どのくらいターミナルを操作したか (経過時間)、どのくらいコンピュータを使用したか (CPU 使用時間) などのアカウント情報を知るには、DCL プロンプトに対して LOGOUT/FULL を入力します。

```
$ LOGOUT/FULL
```

次のような情報が表示されます。

```
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
```

1.12.2 リモート・セッションの終了

リモート・セッションは次の 2 種類の方法で終了できます。

- リモート・システムのログアウト・プロシージャを使用する (たとえば、OpenVMS システムで LOGOUT コマンドを使用する)。
- Ctrl/Y を 2 回押して、リモート・セッションを強制終了するかどうかを質問するホスト・システムのプロンプトを表示する。リモート・セッションを強制終了するときは、Y を押す。この方法は、リモート・システムで実行しているシステムのタイプとは無関係に機能する。

リモート・セッションを終了すると、“%REM-S-END, control Entered to node NODENAME::”というメッセージが表示され、リモート・ノード接続を確立したシステムのプロセスに戻ります。

1.12.3 ネットワーク接続の消失

リモート・システムとの TCP/IP ネットワーク接続が切断される場合、TCP/IP ではベストエフォート・デリバリ・プロトコルを使用します。これは、回線障害などのエラーが発生した場合に、エラーの回復をせずにデータの受け渡しを試みられるネットワーク・テクノロジーの特徴です。

リモート・システムとの DECnet ネットワーク接続が失われた場合には、DECnet は通信を再確立するときにデータを再送します。DECnet が前もって決められた時間切れの時間内に通信を再確立できない場合には、リモート・システムとの接続は終了し、“Path lost to partner”というメッセージが表示されます。

1.13 システム・セキュリティを損なわずにログアウトする

セッションからログアウトすると、システム資源を節約でき、ファイルを保護できます。ターミナルをオンライン状態のままにしておく、内部的な侵入の最大の原因になります。ターミナルをオンラインにしたまま、オフィスを開放しておくことは、自分のパスワードや特権を人に教えてしまうのと同じであり、自分のファイルやグループの他のメンバのファイルが保護されなくなります。このような場合、アカウントを介してアクセス可能なファイルはすべて、誰もが簡単に転送できます。また、ユーザのファイルや、そのユーザが書き込みアクセス権を持つ他のファイルを、内部の人間が悪意で削除したり、ファイル名を変更することもできます。特殊な特権がある場合、特に Files カテゴリや All カテゴリの特権がある場合には、悪意のあるユーザが大きな損害を与える可能性もあります。

一定時間使用しなくても自動的にロックされないシステム上で作業する場合には、たとえ短時間でもオフィスを離れるときは、ログアウトしてください。リモート・ログインを実行した場合には、それぞれのノードからログアウトしなければなりません。

セキュリティ管理者は、ログアウトするときにダイアルアップ回線への接続を切断するようにユーザに指示することがあります。ダイアルアップ回線への接続を切断しておく理由は次のとおりです。

- オープンされているアクセス回線を誰も使用できません。その回線にアクセスするには、アクセス番号が必要であり、個人的に再ダイアルしなければなりません。
- 使用しているダイアルアップ回線が公共の場所にある場合や、自分が使用した後に他のユーザがターミナルを使用する可能性がある場合には、特に回線接続を切断しておくことが重要です。
- また、この操作を実行すると、必要なダイアルアップ回線の数が少なくなるため、資源も節約できます。

1.14 ネットワーク

複数のコンピュータ・システムを 1 つに連結すると、ネットワークが形成されます。OpenVMS ネットワーク上のオペレーティング・システムは、相互に通信し、情報や資源を共用できます。ネットワーク内の各システムをネットワーク・ノードまたはホストと呼び、固有の名前やアドレスで識別します。ここでホストとノードは同義語であり、ネットワークに接続されたシステムを意味します。

OpenVMS では、ネットワーク・プロトコルを選択できます。1つのネットワークで Compaq TCP/IP Services for OpenVMS 製品または弊社 DECnet 製品のどちらかを使用する方法と、1つのネットワークで両方の製品を使用する方法があります。OpenVMS における弊社の基本ネットワーク戦略は業界標準ネットワーク・プロトコル群である TCP/IP です。

1.14.1 ネットワーク・ノード

ネットワーク・ノードにログインすると、他のネットワーク・ノードと通信できるようになります。ログインしたノードをローカル・ノード、ネットワーク上の他のノードをリモート・ノードと呼びます。リモート・ノード上のアカウントにアクセスすると、自分のログインしたローカル・ノードからそのアカウントにログインでき、自分のローカル・ノードと接続したまま、そのノード上でタスクを実行できます。

第 1.5.2 項で、リモート・ノードにログインする方法を説明しています。リモート・ノードで実行できる操作については、本書の該当する章を参照してください。

1.14.2 ネットワークを介してのプログラムの実行

TCP/IP および DECnet ソフトウェアがサポートする機能によって、ネットワークを介してあたかもローカル・ノードで実行するかのようにリモート・ノード上でプログラムを実行できます。ネットワーク・ソフトウェアはオペレーティング・システム内部に統合されているため、リモート・ファイルにアクセスするプログラムを簡単に作成できます。アプリケーション・プログラムの中でリモート・ファイルにアクセスするには、リモート・ノードの名前と必要なアクセス制御情報をファイル指定の中を含めるだけで実現できます。

すべての TCP/IP または DECnet 環境に共通する機能であるタスク間通信を利用すれば、使用するプログラミング言語にかかわらず、同じまたは異なるオペレーティング・システムで動作する2つのアプリケーション・プログラム同士で通信することができます。ネットワーク・アプリケーションの例としては、分散処理アプリケーション、トランザクション処理アプリケーション、そしてサーバへの接続を行うアプリケーションがあります。

注意

本書のリモート操作の例では、代理アカウントを使用して、リモート・システムで操作できるようにしています。代理アカウントは、ユーザがリモート・システムにアクセスするための方法の1つです。リモート・システムにアクセスする他の方法については、『OpenVMS システム管理者マニュアル』を参照してください。

DCL を使用したシステムとの会話

DIGITAL コマンド言語 (DCL) は、オペレーティング・システムに特定の操作を実行させる命令からなります。DCL には、200 を超えるコマンドと関数があり、これらを使用してオペレーティング・システムと通信し、さまざまなコンピューティング・タスクを実行することができます。DCL コマンドを使用すると、次のことが行えます。

- システムについての情報を得る。
- ファイルを操作する。
- ディスクや磁気テープなどのデバイスを操作する。
- 作業環境を変更する。
- プログラムを開発し実行する。
- 機密保護を行って、資源が効率良く使用されるようにする。

次の表は、いくつかの一般的なシステム操作を実行する場合に使用される DCL コマンドを示しています。

コマンド	操作
COPY	指定されたファイルをコピーする。
COPY/FTP	TCP/IP ネットワーク上のホスト間でファイルを転送する。
CREATE	ファイルまたはディレクトリを作成する。
DELETE	指定されたファイルをディレクトリから削除する。
DIRECTORY	ディレクトリの内容 (ファイルのリスト) を表示する。
EDIT	テキスト・ファイルの内容を調べて変更する。
LOGOUT	セッションを終了する。
PRINT	指定されたファイルをプリンタに送って印刷する。
RENAME	指定されたファイルの名前または位置を変更する。
SET	画面上でのシステムの表示のしかたを制御する。
SHOW	システムの状態を表示する。
TYPE	指定されたファイルの内容を画面に表示する。

本章では、DIGITAL コマンド言語の使用方法について学びます。特に次のことについて説明します。

- DCL コマンドの入力
- DCL コマンド・ライン

- DCL コマンドの入力規則
- パラメータの入力
- 修飾子の入力
- 値としての日付と時刻の入力
- コマンドの再呼び出し
- DCL コマンド行の編集
- ターミナル・キーの定義
- キーの組み合わせ

ローカル環境の違い

本書では、標準的な DCL コマンドについてのみ説明します。システム管理者はローカル環境をサポートするためにシステムを変更できます。システム管理者は次のことができます。

- 異なるコマンド言語インタプリタの使用
- 標準的な DCL コマンドの省略時の動作の変更
- 一部の DCL コマンドの使用の禁止
- DCL プロンプトなど、システムの一部の省略時の設定の変更
- 拡張ファイル指定による環境の構成

本章で説明するコマンド、修飾子、およびパラメータについての詳細は、『OpenVMS DCL ディクショナリ』とオンライン・ヘルプを参照してください。

2.1 コマンドの入力

DCL コマンドを入力するためには、DCL プロンプト(\$)に対してコマンドを入力してから Enter を押します。DCL は大文字と小文字を区別しません。したがって、コマンドは大文字または小文字のどちらでも入力できます。¹

たとえば、DCL コマンド SHOW TIME を使用する場合には、次のコマンドを入力します。

```
$ SHOW TIME
```

現在の日付と時間が表示された後 DCL プロンプトに戻るので、別のコマンドを入力できます。

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

¹ 大文字小文字の区別については第 5 章を参照。

2.1.1 使用モード

DCL は、次の 2 つのモードで使用できます。

- 会話

会話モードでは、ユーザはターミナルからコマンドを入力します。1 つのコマンドの実行が終了すると、次のコマンドを入力できます。

- バッチ

バッチ・モードでは、ユーザの代わりにコマンドを実行する別のプロセスをシステムが作成します。バッチ・ジョブとは、独立したユーザ・プロセスとして実行するためにオペレーティング・システムに発行されるコマンド・プロシージャやプログラムです。コマンド・プロシージャをバッチ実行のために発行した後は、ターミナルとの会話を続けることができます。

バッチ・ジョブとネットワーク・プロセスは、DCL をバッチ・モードで使用します。プロセスについての詳細は、第 16 章を参照してください。

2.1.2 DCL コマンドの種類

DCL コマンドを入力すると、DCL インタプリタがそれを読み込んで解釈します。コマンド・インタプリタのコマンドに対する応答は、入力したコマンドの種類によって異なります。DCL コマンドには、次の 3 つの種類があります。

- 組み込みコマンド

DCL インタプリタに組み込まれているコマンドであり、内部的に実行されます。

- プログラムを起動するコマンド

DCL はこの種類のコマンドを受け取ると内部では実行せず、別のプログラムを呼び出して実行します。コマンドを実行するために起動されたプログラムをコマンド・イメージと呼びます。コマンド・イメージは、会話型のプログラムやユーティリティ (Mail など) のこともあれば、非会話型プログラム (COPY など) のこともあります。

- フォーリン・コマンド

イメージを実行するシンボルをフォーリン・コマンドと呼びます。フォーリン・コマンドは、コマンド・インタプリタが DCL コマンドとして認識しない名前のイメージを実行します。シンボルについての詳細は、第 12 章を参照してください。

2.2 DCL コマンド行

DCL には、他の言語と同様、独自の語彙と使用規則があります。DCL は、語 (語彙) と語順 (構文または形式) から構成されます。ここでは、この 2 つの要素について説明するとともに、有効な DCL コマンドの構成方法について解説します。

次の例は、DCL コマンド行の一般形式と各要素を示しています。

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

DCL コマンド行の各要素について説明します。

1 DCL プロンプト

省略時の DCL プロンプトはドル記号(\$)である。DCL と会話している場合、DCL のコマンド受け入れ準備が整うと、このプロンプトが表示される。

2 DCL コマンド

コマンドの名前を指定する。コマンドは、組み込みコマンド・プログラムを起動するコマンド、フォーリン・コマンドのいずれかである。この例では、DCL コマンドは PRINT である。

3 修飾子

コマンドの処理内容を詳細に指定する。コマンド全体を変更する修飾子もあれば、特定のコマンド・パラメータだけを変更する修飾子もある。また、値を取る修飾子もある。修飾子の前には、常にスラッシュ (/) が付く。この例では、修飾子は /COPIES である。

4 値

修飾子をさらに詳細に指定する。多くの場合、値の前には等号 (=) が付く。値としては、ファイル指定、文字列、数値、DCL キーワードを指定できる。キーワードとは、特定の構文形式で使用するために予約された語のことである。

この例では、値は 5 (5 部) になっている。

5 パラメータ

コマンドの処理内容を指定する。パラメータは、決まった順序でコマンドに指定しなければならない。パラメータ値としては、ファイル指定、キュー名、論理名などを指定できる。

6 Enter キー

Enter キーは、DCL コマンド行の最後にくるもので、「コマンドを実行せよ」とシステムに知らせる。

その他に次の項目も、DCL コマンド行で使用可能です。

- ラベル

コマンド・プロシージャの中の行を識別する。ラベルは、コマンド・プロシージャの内部でだけ使用される。ラベルについては、第 13 章と第 14 章を参照。

- キーワード

キーワードは、特定の構文形式で使用するよう定義された語の 1 つである。キーワードは、指定する DCL コマンドの説明のとおり使用しなければならない。たとえば、SET FILE コマンドの/PROTECTION 修飾子に対する DCL キーワードには、SYSTEM、OWNER、GROUP、WORLD を使用できる。DCL キーワードは、値を取ることもある。

- ワイルドカード文字

ワイルドカード文字はアスタリスク(*)、パーセント記号(%), 反復記号(...), ハイフン(-)である。これらの文字はファイル指定の中でファイル名、ファイル・タイプ・ディレクトリ名、バージョン番号の一部として、またはそのかわりに使用でき、対応するフィールドに対して全部であることを示す。ファイルやディレクトリでのワイルドカード文字の使用方法については、第 3 章、第 4 章、および第 5 章を参照。

2.2.1 構文

話し言葉がその語順によって意味が変わるのと同じように、DCL でも、コマンド行の各要素を特定の語順に並べる必要があります。

次の 2 つの例は、典型的な DCL コマンドの構文、つまり形式を示しています。

ラベル: コマンド/修飾子=値=キーワード

ラベル: コマンド パラメータ/修飾子

DCL コマンドには、いくつかの必須パラメータがあります。これらのパラメータは、必ずコマンド行に入力しなければなりません。これらのパラメータを入力しなければ、パラメータの情報を求めるプロンプトが表示されます。アンダスコア(_)で始まる行は、システムが応答を待っていることを意味します。

任意パラメータの入力を求めるプロンプトが表示された場合、Enter を押せば、そのパラメータを省略できます。どちらのプロンプトの場合でも、必須パラメータを入力した後、残りのパラメータや修飾子を 1 つ以上入力できます。

スラッシュ(/)またはアットマーク(@)を含むパラメータは、二重引用符("")で囲む必要があります。

次の例では、TYPE コマンドはファイル指定を要求しています。ファイル指定は TYPE コマンドの必須パラメータであるため、このパラメータを指定しなかった場合には、システムから要求されます。

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

2.2.2 コマンドの取消し

コマンド・プロンプトの後に Ctrl/Z を押すと、DCL はそのコマンドを無視して DCL プロンプトを再表示します。

2.2.3 省略時の値の使用

省略時の設定と呼ぶ一部の項目は、コマンド行に指定する必要がありません。DCL が省略時の設定によって操作を実行する場合には、コマンドに特定の値を割り当てたり、そのコマンドに関連する特定の機能を実行します。コマンドを入力するときに、これらの値や機能を指定する必要はありません。一般に、値や機能は、ユーザが期待するものまたは典型的であると考えられるものです。

DCL は、コマンド・パラメータや修飾子など、いくつかのエリアで省略時の値を提供します。パラメータの省略時の値については、本書で説明されている特定の DCL コマンドについての節を参照してください。修飾子の省略時の値については、第 2.5 節を参照してください。

PRINT コマンドの修飾子として部数を指定しなかった場合には、DCL は省略時の値である 1 を使用します。次の例では、PRINT コマンド行に/COPIES 修飾子を指定しているため、省略時の値が無効になり、ファイルが 4 部印刷されます。

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

2.2.4 複数行のコマンドの入力

1 行を超えるコマンドを入力する場合には、次の操作を実行して、コマンドを次の行に継続できます。

手順	操作
1	コマンド行の最後にハイフン(-)を指定し、Return を押す。 アンダスコア(_)の後に DCL プロンプト(\$)が表示される。
2	このプロンプトの後にコマンド行の残りの部分を入力する。 アンダスコアから始まる行は、システムがユーザの応答を待っていることを示す。

次のことに注意してください。

- コマンド名とパラメータなどの間には、適切なスペースを挿入しなければならない。
- ハイフンの後に Return を押すと、スペースは追加されない。
- コマンドを入力するときの行数に制限はない。ただし、1024 文字という制限を超えないようにしなければならない。

- ハイフンを指定せずに長いコマンド行を入力することもできる。これは、長いコマンド行のテキストが自動改行されるからである。しかし、コマンド行をハイフンで分割した方が、コマンド行が読みやすくなる。

次の例では、複数の行に継続されるコマンドの入力方法を示しています。

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

DCL コマンドの PIPE コマンドを使用すると、1 つの DCL コマンドから複雑なコマンド処理文を作成することができます。たとえば、同一 DCL コマンド行から、次に示す 1 つまたは複数の操作を実行できます。

- パイプライン (コマンドの連続処理)
- 入出力リダイレクト
- 複数の条件コマンドの実行
- バックグラウンド処理

詳細は、第 14.20 節および『OpenVMS DCL デクシヨナリ: N-Z』の PIPE コマンドの説明を参照してください。

2.3 DCL コマンドの入力規則

DCL コマンドを入力するときには、次の規則が適用されます。DCL コマンドにおける拡張ファイル名の使用については、第 5 章を参照してください。

- 大文字と小文字を組み合わせで使用できます。DCL インタプリタは、小文字を大文字に変換します。パラメータと修飾子値の中の大文字と小文字は、引用符(“ ”)で囲まれている場合を除いて、同じものとみなします。
- コマンド名と最初のパラメータの間には、ブランクまたはタブを少なくとも 1 つ挿入する。
- 2 番目以降のパラメータと前のパラメータや修飾子の間には、ブランクまたはタブを少なくとも 1 つ挿入する。
- 修飾子の先頭にはスラッシュ (/) を付ける。スラッシュは分割文字として働くので、前にブランクやタブを挿入する必要はない。
- パラメータまたは修飾子の値にブランクやタブが含まれる場合には、パラメータまたは修飾子の値を引用符で囲む。
- DCL コマンド行では、空文字 (<NUL>) を指定することはできません。これは、たとえ二重引用符で囲んでも同様です。
- 各コマンド行は、127 要素 (パラメータ、修飾子、修飾値) を超えてはなりません。

コマンドの中の各要素は 255 文字を超えてはなりません。また、すべてのシンボル¹とレキシカル関数²を値に変換した結果得られる、コマンド全体が 1,024 文字を超えないようにすることが必要です。

- 一意的に識別できるのであれば、DCL コマンド名や修飾子を短縮することができます。DCL は初めの 4 文字のみを読みます。

たとえば、次の 2 つのコマンドは等しくなります。

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

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

ただし、コマンド・プロシージャの中ではコマンドを短縮しないでください。これは、コマンド・プロシージャの上位互換性や明確さを維持するためです。コマンド・プロシージャの中でのコマンドの使い方については、第 13 章と第 14 章を参照してください。

2.4 パラメータの入力

ファイル指定は、最も一般的なパラメータです。DCL コマンドは、入力ファイル指定 (コマンドによって処理されるファイル) と出力ファイル指定 (コマンドによって作成されるファイル) を受け入れます。

コマンド行でパラメータを指定するときには、次の規則が適用されます。

- コマンド説明の中で、大括弧 ([]) で囲まれた要素は任意項目を示す。たとえば、次のコマンドでは、ファイル指定を入力しなくてもかまわない。

```
DIRECTORY [file-spec]
```

- コマンド説明の中で、大括弧で囲まれていない要素は必須項目である。たとえば、次のコマンドでは、デバイス名を入力しなければならない。

```
SHOW PRINTER device-name
```

- 通常、入力ファイル・パラメータは出力ファイル・パラメータの前に指定する。たとえば、入力ファイル `LISTS.TXT` を出力ファイル `FORMAT.TXT` にコピーするには、次のように入力する。
- パラメータは、1 つでも複数でも指定できる。パラメータを複数入力する場合には、コンマ(,)かプラス記号(+)で区切る。コンマやプラス記号の前後には、スペースやタブ文字をいくつか入れてもかまわない。コマンドの中には、プラス記号を分割文字としてではなく連結文字として解釈するものもあるので注意すること。

¹ 情報を短縮してシステムに渡す場合には、シンボルを使用します (第 12 章を参照)。

² レキシカル関数は、システムから情報を得ます (第 15 章を参照)。

例

次の例は、入力ファイル `LISTS.TXT` を出力ファイル `FORMAT.TXT` にコピーする方法を示しています。

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

次の例は、ファイル指定の一覧をパラメータとして入力する方法を示しています。

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

次の例は、パラメータを複数指定する方法を示しています。ここでは、3つのファイルが4番目のファイルにコピーされます。3つのファイル指定 (`PLUTO.TXT`、`SATURN.TXT`、`EARTH.TXT`) が最初のパラメータです。`PLANETS.TXT` は2番目のパラメータです。`PLUTO.TXT`、`SATURN.TXT`、`EARTH.TXT` の各ファイル指定の間にはスペースがないことに注意してください。

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

2.5 修飾子の入力

修飾子には、次の3種類があります。

- コマンド修飾子
- 定位置修飾子
- パラメータ修飾子

短縮した名前が、同じコマンドのすべての修飾子名の間で固有の名前として識別される場合には、修飾子名を短縮できます。しかし、コマンド・プロシージャの互換性を維持するには、コマンド・プロシージャの内部でコマンドと修飾子を短縮しないようにしてください。

コマンドには省略時の修飾子が設定されています。コマンドの省略時の設定と異なる場合を除き、修飾子を指定する必要はありません。この後の節では、修飾子のタイプと修飾子の省略時の設定について説明します。各コマンドの省略時の設定については、『OpenVMS DCL デictionary』を参照してください。

2.5.1 コマンド修飾子

コマンド修飾子はコマンドを詳細に指定します。修飾子はコマンド行のどこに指定してもかまいませんが、コマンド名の後に指定するのがよいです。複数の修飾子を指定する場合には、コマンド名の後にまとめて指定します。

次の例では、`/QUEUE` がコマンド修飾子です。ファイル `SATURN.TXT` と `EARTH.TXT` が印刷キュー `LN03_PRINT` に登録されます。

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

2.5.2 定位置修飾子

定位置修飾子は、コマンドやパラメータの意味に変更を加えるためのもので、コマンド文字列内での指定位置によってコマンド全体の意味が異なります。コマンドと最初のパラメータの間に定位置修飾子を指定した場合には、コマンド文字列全体が影響を受けます。パラメータの後に定位置修飾子を指定した場合には、そのパラメータだけが影響を受けます。

次の例では、最初の PRINT コマンドは、ファイル SPRING.SUN と FALL.SUM のコピーを 2 部印刷することを要求しています。2 番目の PRINT コマンドは、ファイル SPRING.SUM のコピーは 2 部要求していますが、FALL.SUM のコピーは 1 部しか要求していません。

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

2.5.3 パラメータ修飾子

パラメータ修飾子は、入力ファイルや出力ファイルなどの特定のタイプのパラメータと一緒に使用します。

たとえば、BACKUP コマンドは、入出力ファイル指定にだけ適用されるいくつかのパラメータ修飾子を受け入れます。

次の例の/CREATED と/BEFORE 修飾子は入力ファイルと一緒にしか指定できない修飾子です。これらの修飾子は、特定の入力ファイルを選択してバックアップ操作を行います。なお、アスタリスク(*)は、ファイル名に置き換わるワイルドカード文字です。BACKUP は、2002 年 12 月 11 日以前に作成されたファイル・タイプ TXT を持つすべてのファイルを選択します。

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

2.5.4 矛盾する修飾子

1 つのコマンド行で矛盾する複数の修飾子を使用すると、最も右側の修飾子が優先されます。

コマンドの中には、同一コマンド行に指定できない相反する修飾子が入っているものもあります。互換性のない修飾子を使用すると、コマンド・インタプリタはエラー・メッセージを表示します。

次の例は矛盾する修飾子を示しています。PRINT コマンドは、/COPIES=2 修飾子と/NOBURST 修飾子だけを受け付けます。これは、これらの修飾子がコマンド行に最後に入力された修飾子だからです。

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

2.5.5 修飾子を取る値

修飾子には、キーワード、ファイル指定、文字列、数値を指定できます。修飾子に値を入力するときには、修飾子と値を等号(=)またはコロン(:)で分けます。

修飾子キーワードの中には、さらに情報が必要なものもあります。この場合には、キーワードと値をコロンまたは等号で分けます。

値を必要とする複数のキーワードを指定するには、キーワードを括弧で囲み、キーワードと値を等号(=)またはコロン(:)で分けます。

例

- この例に示したコマンドはどちらも正しいコマンドである。

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

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

- この例は、追加情報が必要な修飾子を示している。"PROTECTION"キーワードとその値がコロンまたは等号(=)によって区切られている。

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

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

- この例は、複数のキーワードを必要とする修飾子を示しており、各キーワードは複数の値を必要とする。

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

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

2.6 値としての日付と時刻の入力

コマンドや修飾子の中には、PRINT/AFTER コマンドのように日付と時刻を値として取るものがあります。このような値は、次のいずれかの形式で指定します。

- 絶対時刻
- デルタ時間
- 複合時刻

2.6.1 絶対時刻形式

絶対時刻は、特定の日付または時刻を示します。絶対時刻の形式は次のとおりです。

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

各フィールドは、次のような意味を持っています。

dd	日。1 ~ 31 の整数。
mmm	月。JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC。
yyyy	年 (西暦)。整数。
hh	時間。0 ~ 23 の整数。
mm	分。0 ~ 59 の整数。
ss	秒。0 ~ 59 の整数。
cc	1/100 秒。0 ~ 99 の整数。

絶対時刻には、次のような規則が適用されます。

- 日付と時刻の右側部分は省略可能。
- 日付と時刻を両方指定する場合には、日付と時刻の間にコロンを挿入する。
- 日付には、少なくとも 1 つのハイフンを挿入する。
- 各フィールドを分割する句読点を使用すれば、日付と時刻内部のどのフィールドを省略してもかまわない。
- 切り捨てられたまたは省略された日付フィールドの省略時の値は、現在の日付の該当フィールドに設定される。
- 切り捨てられたまたは省略された時刻フィールドの省略時の値はゼロ。
- 現在の時刻または将来の時刻を入力すべきコマンドの中で誤って過去の時刻を指定すると、現在の時刻が使用される。

絶対時刻は、次のいずれかのキーワードとしても指定できます。

TODAY	今日の 00:00:00.00 時
TOMORROW	明日の 00:00:00.00 時
YESTERDAY	昨日の 00:00:00.00 時

次に、絶対時刻の指定の例をいくつか紹介します。

時刻指定	結果
11-DEC-2002:13	2002 年 12 月 11 日午後 1 時
11-DEC	今年の 12 月 11 日の午前 0 時
15:30	今日の午後 3 時 30 分
19-	今年の今月の 19 日の午前 0 時
19-:30	今月の 19 日の午前 0 時 30 分

2.6.2 デルタ時間の形式

デルタ時間とは、現在の日付および時刻から将来の時刻までのオフセット (時間間隔) のことです。次に、デルタ時間の一般形式を示します。

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

各フィールドは、次のような意味を持っています。

dddd	日数。0 ~ 9999 の整数。
hh	時数。0 ~ 23 の整数。
mm	分数。0 ~ 59 の整数。
ss	秒数。0 ~ 59 の整数。
cc	1/100 秒数。0 ~ 99 の整数。

絶対時刻、デルタ時間、または2つの組み合わせとして表現できる値として、修飾子が説明されている場合には、複合時刻の一部であるかのようにデルタ時間を指定しなければなりません。たとえば、現在の時刻から5分間というデルタ時間を指定するには、“+:5”と指定します (“0-0:5”とは指定しません)。

デルタ時間の指定には、次のような規則が適用されます。

- デルタ時間の右側部分は省略可能。
- 日数を指定する場合には、ハイフンを使用する。
- 各フィールドを分割する句読点を使用すれば、時間表記内のフィールドを省略してもよい。
- 時間フィールドを省略したときの省略時の値はゼロ。

次に、デルタ時間の指定の例をいくつか紹介します。

時刻指定	結果
"+3-"	現在から 3 日間 (72 時間)
"+3"	現在から 3 時間
"+:30"	現在から 30 分
"+3-:30"	現在から 3 日と 30 分
"+15:30"	現在から 15 時間 30 分

2.6.3 複合時刻の形式

絶対時刻とデルタ時間を組み合わせる場合には、絶対時刻と正 (+) または負 (-) の符号を付けたデルタ時間を指定します。複合時刻の形式は、次のとおりです。

"[絶対時刻][+デルタ時間]"

[絶対時刻][-デルタ時間]

絶対時刻値とデルタ時間値の可変フィールドと省略時のフィールドには、前の項で説明した規則と同じ規則が適用されます。

複合時刻を指定する場合には、次の規則が適用されます。

- デルタ時間値の前に正または負の符号を挿入する。負の符号は、キーボード上ではハイフンと同じキーになる。
- デルタ時間値の前に正または負の符号が付いている場合には、時間指定全体を引用符で囲む。
- 現在の日付と時刻からデルタ時間をオフセットしたい場合には、絶対時刻値を省略する。
- 日付と時刻の情報は、できるだけ詳しく指定する。

次の表に、複合時刻の指定の例をいくつか紹介します。

時刻指定	結果
"+5"	現在から 5 時間後。
"-1"	現在の時刻から 1 時間を減算した値。負の符号(-)は負のオフセットを示す。1 の後にハイフンがないので、1 は日付ではなく、時間として解釈される。
"+:5"	現在から 5 分後。
"-:5"	現在の時刻から 5 分を減算した値。
"-1-00"	現在の時刻から 1 日を減算した値。負の符号(-)は負のオフセットを示す。(-)ハイフンが日付フィールドと時刻フィールドを分割している。
"31-DEC:+:5"	今年の 12 月 31 日の午前 0 時 5 分。絶対時刻指定 (コロンの前) の省略時の値は、今年の 12 月 31 日の午前 0 時に設定されている。正の符号(+)は、正のオフセットを示す。
31-DEC:-00:10	今年の 12 月 30 日の 23 時 50 分。絶対時刻指定 (コロンの前) の省略時の値は、今年の 12 月 31 日の午前 0 時に設定されている。DEC: の後の負の符号(-)は、負のオフセットを示す。

2.7 コマンドの再呼び出し

DCL レベルでは、前に入力したコマンド行を再呼び出しできるので、長いコマンド行を何度も入力する手間が省けます。コマンドを再呼び出しすれば、再実行や編集をすることができます。

OpenVMS VAX システムでは、再呼び出しバッファには前に入力したコマンドを最大 20 個まで格納できます。

OpenVMS Alpha システムでは、再呼び出しバッファには前に入力したコマンドを最大 254 個まで格納できます。

前に入力したコマンドを表示する場合には、次のいずれかの方法を使用します。

- Ctrl/B を押す。

- 上下の矢印キーを使用する。
- RECALL コマンドを入力する。

2.7.1 Ctrl/B の使用

Ctrl/B を一回押すと、すぐ前のコマンド行が再呼び出しされます。Ctrl/B をもう一度押すと、2 つ前のコマンド行が再呼び出しされ、以下も同様になります。

2.7.2 矢印キーの使用

上向き矢印キーを押すと、直前のコマンドが再呼び出しされ、下向き矢印キーを押すと、直後のコマンドが再呼び出しされます。また、これらの矢印キーを何回も押して、以前に実行したコマンドを順に表示させ選択することができます。

2.7.3 RECALL コマンドの使用

コマンド行を調べるには、RECALL/ALL と入力します。再呼び出しできるコマンドを調べた後、RECALL と入力してから希望のコマンド番号を指定すれば、特定のコマンド行を再呼び出しできます。

RECALL の後に、表示したいコマンド行の最初の文字を指定することもできます。RECALL は、すでに入力したコマンド行を最後から順に検索して、指定した文字で始まる最初のコマンド行を戻します。

例

次の例は、RECALL/ALL と入力することによって生成される表示を示しています。

```
$ 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]
```

次の例は、4 番目のコマンド行を再呼び出しする方法を示しています。

```
$ RECALL 4
```

Enter を押すと、DCL プロンプトに続いてリストの中の 4 番目のコマンドが表示されます。なお、RECALL コマンド自体は、バッファには登録されません。

たとえば、以前に入力したコマンド EDIT ACCOUNTS.COM を再呼び出しするには、次のコマンドを入力します。

```
$ RECALL E
```

Enter を押すと、次のコマンド行が表示されます。

```
$ EDIT ACCOUNTS.COM
```

注意

OpenVMS 画面管理ソフトウェアを使用するユーティリティやアプリケーション・プログラムを実行している場合には、Ctrl/B と上下の矢印キーを使用してコマンドの再呼び出しを行うことができます。この場合、行編集機能が使用可能でなければなりません。このような機能を持つユーティリティには、MAIL、OpenVMS デバッガ、SHOW CLUSTER、システム・ダンプ・アナライザ (SDA)、EVE エディタがあります。

再呼び出しバッファの内容を削除する場合には、次に示すように、RECALL コマンドに ERASE 修飾子を指定します。

```
$ RECALL/ERASE
```

機密保護上の理由から、パスワードを含むコマンドを入力した後は、再呼び出しバッファの内容を削除するとよいでしょう。

2.8 DCL コマンド行の編集

DCL コマンド・レベルでは、1 つ 1 つのキーやキーの組み合わせを使用して、入力内容を変更することができます。ターミナルのタイプが異なると操作特性も異なりますが、大半のターミナルは標準ファンクション・キーとライン・エディタで使用できるキーを備えています。

ターミナル上で行編集機能が使用可能かどうかを調べる場合には、SHOW TERMINAL コマンドを入力します。行編集機能の現在の状態が "Terminal Characteristics" の最初の桁に表示されます。

次の例では、行編集は有効ではありません。

```
$ 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	Fulldup
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 Pasthru
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			

2.8.1 SET TERMINAL コマンド

SET TERMINAL コマンドを使用すると、ターミナルが DCL コマンド行を編集する方法を変更することもできます。省略時の設定では、SET TERMINAL コマンドによる編集内容の変更は、現在のセッションのみに適用されます。ログインするたびにターミナルを設定する場合には、LOGIN.COM ファイルに SET TERMINAL コマンドを入れるようにします。

この例では、行編集機能はできません。行編集機能を使用可能にする場合には、SET TERMINAL/LINE_EDIT コマンドを入力します。

```
$ SET TERMINAL/LINE_EDIT
```

挿入モードと上書モード

コマンド行は、挿入モードまたは上書モードで編集できます。挿入モードでは、入力した文字がカーソルの左側に挿入されます。上書モードでは、入力した文字がカーソルの上に示された文字に重ね書きされます。

コマンド行の編集モードを変更する場合には、Ctrl/A を押します (Ctrl/A がスイッチの働きをします)。セッションの編集モードを変更するには、SET TERMINAL/INSERT または SET TERMINAL/OVERSTRIKE コマンドを入力します。

テキストのラップ

SET TERMINAL/WRAP コマンドを使用すると、画面の 1 行に収まらない数の文字を入力する場合に、テキストは次の行に自動改行されます。SET TERMINAL/NOWRAP コマンドで、ターミナル画面の 1 行に限度以上に文字を入力すると、その行の最後の文字の位置に上書きされます。

編集できるのは、カーソルが存在する行だけです。テキストを自動改行するときは、上向き矢印キーを使用してカーソルを上に移動して、前の行を編集することはできません。カーソルを前の行に移動するには、Delete キーを使用して現在の行にあるすべての文字を削除します。

2.8.2 コマンド行の要素の削除

Backspace キーを押すと、カーソルが後退し、その場所にある文字が削除されます。行編集機能が使用可能な場合には、Ctrl/U を使用して、行の先頭から現在のカーソル位置までの文字を削除できます。行編集機能が使用できない場合には、Ctrl/U を使用すると行全体が取り消されます。このとき、システムはその行を無視して、DCL プロンプトを再表示します。

2.9 ターミナル・キーの定義

キー定義とは、特定のターミナル・キーに文字列を割り当てることです。DEFINE /KEY コマンドを使用します。キーを定義しておくで、定義したキーを押すだけで文字列を入力する代わりになります。キー定義には、通常、コマンド行全体またはその一部が入っています。キー定義を使用すると、キーストロークを減らして DCL コマンドを入力できるようにキーボードをカスタマイズできます。定義したキーを押すと、コマンドを/TERMINATE 修飾子で定義したかどうかにより、コマンドがターミナルに表示されたり実行されたりします。

省略時には、ターミナルは数値キーパッド・モードに設定されています。SET TERMINAL コマンドを使用して、数値キーパッドのキーを再定義できます。詳細は、『OpenVMS DCL ディクショナリ』の SET TERMINAL/APPLICATION_KEYPAD、SET TERMINAL/NUMERIC、および DEFINE/KEY コマンドの説明を参照してください。

2.9.1 キー・シーケンス

入力した DCL コマンドに加えて、キーの組み合わせを使用してもタスクを実行できます。キーの組み合わせは、別のコマンドを処理しながらシステムを使用するための手段です。

キーの組み合わせを入力するためには、Ctrl キーを押しながらもう 1 つのキーを押します。次の表は、機能ごとにキー・シーケンスをまとめたものです。

表 2-1 DCL コマンドを入力

キー・シーケンス	機能
Ctrl/Z および F10	ターミナルから入力されたデータのファイルの終端を知らせる。
Enter	現在行をシステムに送信して処理させる。ログインしていない場合には、Enter はログイン・シーケンスを開始する。

表 2-2 DCL コマンドに割り込む

キー・シーケンス	機能
Ctrl/T	<p>ターミナル出力に割り込んで、現在のプロセスについての統計情報を表示する。ノード名、ユーザ名、時間、実行中のイメージの名前、現在のターミナル・セッションで使用されているシステム資源についての情報が表示される。</p> <p>Ctrl/T キーを使用すると、システムが、動作中かどうかとも判別できる。システムが一時的に応答不能な場合や、ターミナルが NOBROADCAST に設定されている場合には情報は表示されない。Ctrl/T を使用するには、最初に (システム・ログイン・コマンド・プロシージャや個人ログイン・コマンド・プロシージャ、または会話の中で) SET CONTROL=T コマンドを入力しなければならない。</p>
Ctrl/Y, Ctrl/C およ び F6	<p>コマンド処理に割り込む。Ctrl/Y を使用不能にするには、SET NOCONTROL=Y コマンドを使用する。</p> <p>ほとんどの場合は、DCL プロンプトが戻る。このとき、プログラムは実行中である。いずれかの組み込みコマンドを入力すれば、CONTINUE コマンドでプログラムの実行を継続できる。なお、CONTINUE コマンドを入力した後で画面を消去する場合には、Ctrl/W を押す。</p>

表 2-3 コマンドを再呼び出しする

キー・シーケンス	機能
Ctrl/B, 上向き矢印	以前に入力した最大 20 個 (VAX) または 254 個 (Alpha) のコマンドを再呼び出しする。
下向き矢印	再呼び出しバッファにある次の行を表示する。

表 2-4 カーソル位置を制御する

キー・シーケンス	機能
Backspace	最後に入力された文字を削除する。
Ctrl/A, F14	上書モードと挿入モードを切り替える。省略時のモード (SET TERMINAL/LINE_EDITING コマンドで設定) は、各行の冒頭で再設定される。
Ctrl/D, 左矢印	カーソルを 1 文字だけ左に移動する。
Ctrl/E	カーソルを行末に移動する。
Ctrl/F, 右矢印	カーソルを 1 文字だけ右に移動する。
Ctrl/H, F12	カーソルを行頭に移動する。
Ctrl/I, Tab	カーソルをターミナル上の次のタブ・ストップに移動する。タブ・ストップは、1 行の中で 8 文字ごとに設定されている。タブ設定値はハードウェア・ターミナル特性で、通常はユーザが変更可能。Tab キーは、行編集機能が使用不能の場合も動作する。
Ctrl/J	カーソルの左にある単語を削除する。
Ctrl/K	現在行を次の垂直タブ・ストップに進める。
Ctrl/L	カーソルを次のページの冒頭に移動する。行編集機能が使用可能な場合には、このキーの機能は無視される。

(次ページに続く)

表 2-4 (続き) カーソル位置を制御する

キー・シーケンス	機能
Ctrl/R	現在のコマンド行を繰り返す。カーソルは、Ctrl/R を押したときの位置に残る。
Ctrl/U	現在の入力行のカーソルの左にあるテキストを削除する。
Ctrl/V	行編集ファンクション・キーの一部をオフにする。たとえば、Ctrl/V の後に Ctrl/D を押すと、カーソルが 1 文字左に移動する代わりに、Ctrl/D が生成される。Ctrl/D は、DCL レベルでの行終了文字。 Ctrl/V と組み合わせると、行終了文字でない文字は影響を及ぼさない。Ctrl/H や Ctrl/J などがこの例である。ただし、Ctrl/U などの制御キーは、行編集機能を保持したままである。
Ctrl/X	現在行を取り消して、先読みを可能にするバッファにあるデータを削除する。
F7 , F8 , F9 , F11	予備。

表 2-5 画面表示を制御する

キー・シーケンス	機能
Ctrl/O	ターミナルへの表示を一時停止したり再開したりする。Ctrl/O を押すと、Output off または Output on と表示される。
Ctrl/S	Ctrl/Q が押されるまで、ターミナル出力を一時停止する。
Ctrl/Q	Ctrl/S によって一時停止されていたターミナル出力を再開する。
Ctrl/W	画面表示をリフレッシュする。

ファイル情報の格納

ファイルは、情報を含むシステム・オブジェクトです。この情報は、コンピュータが理解できる機械可読データのこととあれば、ユーザが入力して操作するテキストのこともあります。ファイルの内容には、ドキュメント、プログラム、アドレス・リストなどがあります。テキスト・ファイルの内容は、オンラインで表示したり、印刷することによって調べることができます。

プログラムは、イメージまたは実行可能イメージとも呼ばれ、命令やデータを機械が読み取り可能な形式で含んでいるファイルのことです。一部のプログラムは、DCL コマンドに関連付けられています。たとえば、DCL の COPY コマンドを入力すると、システムは SYS\$SYSTEM:COPY.EXE というプログラムを実行します。また、DCL の RUN コマンドの後にプログラム名を入力して起動するプログラムもあります。

イメージ・ファイルには、オペレーティング・システムが提供するものと、ユーザが作成するものがあり、通常、イメージ・ファイルのファイル・タイプは EXE です。イメージ・ファイルは ASCII 文字では構成されていないので、DCL コマンドの TYPE、PRINT、または EDIT によって内容を見ることはできません。(一方、テキスト・ファイルは、英字、句読点、数字などの特殊記号を表す標準的な方式である ASCII 文字で構成されています。)

本章では、ファイルを作成したり、処理する方法について、ローカルで作業する場合と、TCP/IP または DECnet for OpenVMS ネットワークを介して作業する場合について説明します。特に、次のことについて説明します。

- ファイル名とファイル指定
- ファイル名でのワイルドカードの使用方法
- その他のファイル名
- ファイルの作成と変更
- ファイルの内容の表示
- ファイルの削除
- 他のユーザからのファイルの保護
- ファイルの印刷

詳細は、以下のマニュアルを参照してください。

- extended file specifications 環境でのファイル名については、第 5 章

- 本章で説明しているコマンドについては、『OpenVMS DCL デictionary』およびオンライン・ヘルプ
- リモート・ノードへのアクセスについては、『OpenVMS システム管理者マニュアル』
- TCP/IP ユーザ・ユーティリティおよびコマンドの使用については、『Compaq TCP/IP Services for OpenVMS User's Guide』
- DECnet ネットワークについては、『DECnet for OpenVMS Networking Manual』
- DECnet Phase V ネットワークについては、『DECnet-Plus for OpenVMS Introduction and User's Guide』

3.1 ファイル名とファイル指定

ファイルとは、人間と機械の双方が扱えるデータを格納するために、OpenVMS オペレーティング・システムが使用する単位のことです。ファイルに名前を付ける場合には、システムがそのファイルの格納場所や内容を使用できる情報を指定する。

ファイル名は、ファイルの名前とファイル・タイプで構成されます。名前とファイル・タイプはピリオド(.)で区切られます。また、ファイルにはバージョン番号が付けられ、複数のファイル・バージョンのファイルを扱うことができます。バージョン番号を指定しなければ、既存のファイル番号で最も大きな値のバージョン番号が使用されます。ファイルを編集すると、元のバージョンは変更されず、新しい出力ファイルが作成されます。省略時の設定では、出力ファイルの名前とファイル・タイプは元のファイルと同じものが使用されますが、バージョン番号は同じ名前の既存のファイルより1つ大きな値になります。

ファイルの名前、ファイル・タイプ、およびバージョン番号を合わせてファイル指定と呼びます。

3.1.1 完全なファイル指定

ファイルは、ネットワーク上の特定のコンピュータ(すなわちノード)、そのコンピュータに接続されている特定のデバイスまたはデバイスのセット(ボリュームと呼ばれます)、そのボリューム上の特定のディレクトリにあります。完全なファイル指定とは、次のとおりです。

- システムがファイルを配置したり識別したりするためのアクセス・パスをもらさず記述している。
- ファイル名に加えてファイルが格納されているディレクトリと、ファイルが存在するネットワーク・ノードを指定する。
- 完全なファイル指定のことを、ネットワーク・ファイル指定と呼ぶこともある。

完全なファイル指定を指定する必要はありませんが、システムとユーザの両方が識別できるようなファイル名あるいはファイル・タイプを指定しなければなりません。ただし、省略時の構成要素と組み合わせる場合、システムがファイルの位置付けや識別をするためのファイル指定をする必要があります。¹

システムの省略時の設定を無効にする場合、またはネットワークを介してファイル操作を行う場合には、完全なファイル指定が必要です。完全なファイル指定とは、次の形式のファイル指定のことです。

ノード:: デバイス:[ルート].[ディレクトリ]ファイル名. ファイル・タイプ; バージョン

次に、各構成要素について説明します。

ノード	ネットワーク・ノードまたはホスト名。TCP/IP または DECnet をサポートするシステムのものに適用。磁気テープに保存するファイルには適用しない。ログインしたシステムと同じシステム上のファイルの指定には使用しない。
デバイス	OpenVMS オペレーティング・システムを実行しているコンピュータに接続されたディスク・ドライブ、テープ・ドライブ、またはその他の周辺機器を指す用語。各デバイスには、そのデバイスの種類と位置を示す一意の名前が付けられる。ディスクは、ODS-2 (省略時の値) または ODS-5 (OpenVMS Alpha のみ) としてフォーマットできる。
ディレクトリ	ファイルが格納されるディレクトリの名前。ディレクトリは、大括弧 ([]) またはアングル括弧 (<>) で区切る。保存するファイルには適用しない。
ファイル名	ファイルの名前。
ファイル・タイプ	ファイル構造やファイル・タイプの識別。
バージョン	ファイルのバージョン番号。バージョンは 10 進数で表され、ファイルの新しいバージョンを作成するたびに 1 ずつ大きくなる。ユーザが指定しない場合には、システムが自動的にバージョン番号を割り当てる。

3.1.2 ファイル指定の規則

ファイル指定の各要素を指定する場合には、次の規則に従います。

- わかりやすいファイル名を付けます。ODS-2 ディスクを備えた OpenVMS Alpha システムと OpenVMS VAX システムでは、ファイル名には A から Z (大文字または小文字)、数字の 0 から 9、()、(-)、チルド (~)、(\$) からなる 39 文字を指定できます。
- ファイル名の先頭にハイフンは使用しないでください。OpenVMS の一部の古いバージョンでは、これをサポートしていない形式のファイル指定があるためです。
- ファイル・タイプの先頭はピリオド (.) とします。ODS-2 ディスクを備えた Alpha システムと VAX システムにおけるファイル・タイプには、文字の A から Z (大文字と小文字のどちらでも指定可)、数字の 0 から 9、アンダースコア (_)、ハイフン (-)、ドル記号 (\$) から選ばれた文字を最高で 39 文字 (ピリオドを含む) 指定できます。

¹ レコード・マネージメント・サービス (RMS) は、ファイルの処理と管理においてアプリケーション・プログラムを支援する OpenVMS の機能です。RMS は、ファイル指定解析の規則を保持します。RMS による部分ファイル指定への省略時の設定の適用方法については、『Guide to OpenVMS File Applications』を参照してください。

- バージョン番号の先頭文字はセミコロン (;) またはピリオド (.) です。システムがファイル指定を表示するとき、バージョン番号にはセミコロンが表示されます。
- 磁気テープ上のファイルの参照にディレクトリ・フィールドは使用しないでください。(ディレクトリを使用できるのはディスク上のファイルだけです。)
- ノード名はシステムがネットワークの一部で、目的のファイルがログインしたノードにある場合に使用します。
- ODS-2 ディスクの OpenVMS Alpha システムと OpenVMS VAX システムでは、UFD(ユーザ・ファイル・ディレクトリ) 名またはサブディレクトリ名には、文字の A から Z(大文字と小文字のどちらでも指定可)、数字の 0 から 9、アンダースコア (_), ハイフン (-), ドル記号 (\$) から選ばれた文字を最高で 39 文字指定できます。サブディレクトリ名の先頭にハイフンは使用できません。
- OpenVMS Alpha バージョン 7.2 およびそれ以降では、ディレクトリとルート名のサブディレクトリ内の文字の合計数(かぎ括弧と区切り記号のピリオドを除く)は最高で 512 文字です。また、UFD とサブディレクトリ名にもファイル名、ファイル・タイプ、バージョン番号の制限が適用され、ディレクトリは <directory-name>.DIR;1 形式のファイルとして保存されます。
- 拡張ファイル指定をサポートするシステムとサポートしないシステムが混在する環境では、制限の多いシステムからは、その能力を越える名前を持つファイルやディレクトリにアクセスできません。

詳細については『Guide to OpenVMS File Applications』を参照してください。

注意

Extended File Specifications を使用する環境の場合、これらの規則は異なります。拡張ファイル名について詳しくは、第 5 章を参照してください。

3.1.3 DCL コマンドの省略時のファイル・タイプ

コマンドによっては、ファイル・タイプを省略した場合に、システムが省略時の値を適用することがあります。次の表に、DCL コマンドが使用する一般的な省略時のファイル・タイプの一部を示します。

ファイル・タイプ	内容
.CLD	コマンド定義ファイル
.COM	コマンド・プロシージャ・ファイル
.DAT	データ・ファイル
.DIF	DIFFERENCES コマンドによって作成される出力ファイル
.DIR	ディレクトリ・ファイル
.DIS	MAIL コマンドの配布リスト・ファイル

ファイル・タイプ	内容
.EXE	リンカによって作成される実行可能プログラム・イメージ・ファイル
.HLB	ヘルプ・テキスト・ライブラリ・ファイル
.HLP	ヘルプ・ライブラリの入力ソース・ファイル
.INI	初期化ファイル
.LIS	言語コンパイラまたはアセンブラによって作成されるリスト・ファイル, または PRINT コマンドと TYPE コマンドの省略時の入力ファイル
.LOG	バッチ・ジョブ出力ファイル
.MAI	MAIL メッセージ・ファイル
.PS	PostScript 形式のファイル
.SYS	システム・イメージ
.TJL	DECTPU と ACL エディタによって作成されるジャーナル・ファイル
.TLB	テキスト・ライブラリ・ファイル
.TMP	一時的ファイル
.TPU	EVE エディタのコマンド・ファイル
.TPU\$JOURNAL	EVE エディタによって作成されるジャーナル・ファイル
.TXT	テキスト・ライブラリの入力ファイルまたは MAIL コマンド出力

3.1.4 言語ソース・プログラムの省略時のファイル・タイプ

次の表に, 高級言語ソース・プログラムの省略時のファイル・タイプを示します。

ファイル・タイプ	内容
.ADA	Compaq Ada コンパイラの入力ソース・ファイル
.BAS	BASIC コンパイラの入力ソース・ファイル
.B32	VAX BLISS-32 コンパイラの入力ソース・ファイル
.C	Compaq C コンパイラの入力ソース・ファイル
.COB	Open VMS VAX システム上の VAX COBOL コンパイラおよび OpenVMS Alpha システム上の Compaq COBOL コンパイラの入力ソース・ファイル
.FOR	Compaq Fortran (OpenVMS VAX システム用 Compaq Fortran はこれまで VAX Fortran だった) の入力ソース・ファイル
.M64	OpenVMS Alpha MACRO-64 アセンブラの入力ソース・ファイル
.MAP	リンカ・ユーティリティによって作成されるメモリ割り当てマップ
.MAR	OpenVMS Alpha 用 VAX MACRO アセンブラまたは MACRO-32 コンパイラの入力ソース・ファイル
.MLB	MACRO アセンブラのマクロ・ライブラリ
.MSG	メッセージのテキストを指定するソース・ファイル
.OBJ	言語コンパイラまたはアセンブラによって作成されるオブジェクト・ファイル
.OLB	オブジェクト・モジュール・ライブラリ
.OPT	LINK コマンドへの入力用オプション・ファイル
.PAS	Pascal コンパイラの入力ソース・ファイル

ファイル・タイプ	内容
.PLI	PL/I コンパイラの入力ソース・ファイル
.STB	リンカ・ユーティリティによって作成されるシンボル・テーブル・ファイル
.UPD	VAX MACROソース・プログラムの変更用更新ファイル (SUMSLP エディタへの入力ともなる)

3.1.5 ファイルのバージョン番号

すべてのファイルには、ファイル名とファイル・タイプに加えて、バージョン番号があります。バージョン番号は、ファイルのバージョンを表す 1 ~ 32,767 の 10 進数です。ファイルを作成すると、ファイルに 1 というバージョン番号が割り当てられます。

1 つのファイルに対して複数のバージョンが存在することもあります。バージョン番号の指定がない場合には、バージョン番号の最も大きいファイルが使用されます。バージョン番号として 0 を指定した場合には、既存の最大バージョンが使用されます。ファイルの新しいバージョンを作成するコマンド、アプリケーション、テキスト・エディタ (EVE など) を使用してファイルを変更した場合、ファイル名は変更されませんが、バージョン番号は 1 だけ大きくなります。

バージョン番号の前には、セミコロンまたはピリオドを入れます。ファイル指定を表示する場合は、ファイル・バージョン番号の前にセミコロンが表示されます。

ゼロまたは負のバージョン番号を指定すると、ファイルのバージョンを相対的に表すことができます。ゼロを指定すると、ファイルの最新 (最も大きい) バージョンが使用されます。-1 を指定すると最新バージョンの前のバージョン、-2 を指定するとその前のバージョンが使用されます。以下同様です。ファイルの最も古い (最も小さい) バージョンを探す場合には、バージョン番号として -0 を指定します。バージョン番号が 32767 より大きいファイルを作成することはできません。バージョン番号が 32767 より大きい新しいファイルを作成しようとした場合には、エラー・メッセージが表示されます。

CREATE/DIRECTORY、SET DIRECTORY、または SET FILE コマンドで /VERSION_LIMIT 修飾子を指定すると、ファイルのバージョン番号を制御できます。バージョンの上限値を超えると、最も小さいバージョン番号のファイルが自動的に削除されます。たとえば、バージョンの上限値が 5 の場合には、ファイルの 6 番目のバージョン (ACCOUNTS.DAT;6) を作成すると、ファイルの最初のバージョン (ACCOUNTS.DAT;1) が削除されます。DIRECTORY/FULL コマンドを実行すると、作成できるバージョンの制限を確認できます。作成できるバージョンの制限は、File attributes: フィールドに表示されます。

3.1.6 ネットワーク・ノード名

ノードとは、コンピュータ・ネットワークを構成する個々のシステムのことです。使用しているシステムがネットワークの一部である場合は、ログインしたときにアクセスするノードのことをローカル・ノードと呼びます。そして、ネットワークの中のこれ以外のノードをリモート・ノードと呼びます。リモート・ノードのファイルを指定する場合には、ノード名を使用します。

ノード指定の形式は、次のとおりです。

ノード["アクセス制御文字列"]::

ファイル指定の一部としてノード名を入力する場合には、次の規則に従います。

- ノード名には 1 ~ 6 文字の英数字を指定できるが、少なくとも 1 文字の英字が必要である。次の例を参照。

AFTP1
F2OTR2
MYNODE

- ノード名の後には、アクセス制御文字列の有無にかかわらず必ずダブルコロンを (::) を付ける。
- ノード名を指定する場合には、0 ~ 42 文字のアクセス制御文字列を含めることができる。アクセス制御文字列には、リモート・ノードに送信するログイン情報を含める。アクセス制御文字列については、第 3.1.12 項を参照すること。

アクセス制御文字列の後に、ダブルコロンを付けること。

- ノード名の代わりに論理ノード名を使用できる。論理ノード名については、第 11 章を参照。

3.1.7 DECnet-Plus の完全なノード名の指定

OpenVMS システムでは、完全なノード名を指定できます。しかし、完全なノード名を認識するためには、DECnet-Plus ソフトウェアがインストールされていなければなりません。

完全なノード名の長さは最大 255 文字であり、次の文字を除き、他の文字はすべて使用できます。

- スペース
- タブ
- 特殊文字: カンマ(,), 引用符(" "), スラッシュ(/), 感嘆符(!), 等号(=), プラス記号(+), アットマーク(@), 一重引用符('), 括弧(()), 二重コロンの (::)
- 最初の文字または最後の文字として使用する場合は 1 つのコロン(:)

完全なノード名が引用符(" ")で囲まれている場合には、単独で使用する引用符を除き、他のすべての文字を使用できます。ノード名の内部で引用符を使用するときは、2つの連続する引用符を指定し、引用符も含めて文字列全体を引用符で囲まなければならない。

OpenVMS ソフトウェアでは、ノード名の構文に関していくつかの規則を設定していますが、実際の正しいノード名は、システムで実行されている DECnet ソフトウェアによって制限されます。完全なノード名についての詳しい説明は、DECnet-Plusに関するマニュアルを参照してください。(有効な文字コードを含め) 構文規則についての詳細は、『DECnet-Plus DECdns Management Guide』を参照してください。

次の例では、ノード名の内部で引用符を使用しているため、文字列全体が引用符に囲まれています。

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

次の例は正しい完全なノード名を示しています。

```
MYNODE
MASSACHUSETTS:.BUSINESS.YOURNODE
A.B;C
```

3.1.8 TCP/IP 名およびアドレスの指定

TCP/IP では、特に指定のない限り、コマンド行でホストを指定する場合、ホスト名、完全修飾ドメイン名、またはホストの IP アドレスを使用できます。ホストの相対名は、完全修飾ドメイン名を含まない単純名です。すなわち、ピリオド(.)を含みません。TCP/IP 構文規則については、『Compaq TCP/IP Services for OpenVMS User's Guide』を参照してください。

3.1.9 DECnet 使用によるリモート・ノード上のファイルへのアクセス

リモート・ノードのファイルにアクセスすると、DECnet はリモート・ノードでログインします。このためには、そのノードについてのログイン情報が必要になります。ノードについてのログイン情報は、アクセス制御文字列で指定できます。アクセス制御文字列を省略した場合には、リモート・ノードに送られるログイン情報は次のようにして判別されます。

- リモート・ノードに代理ログイン・アカウントがある場合には、システムはそのアカウントを使用してログインする。代理ログイン・アカウントを使用すると、指定されたユーザだけがノードにログインできる。
- 代理ログイン・アカウントが存在しない場合には、リモート・ノードのシステム管理者によって指定された、省略時の DECnet アカウントが使用される。

アクセス制御文字列を指定すると、システムはその文字列を使用してリモート・ノードにログインします。ファイル指定の残りの部分は、リモート・ノードに渡されてから、そこで解釈されます。

ファイル指定の一部としてローカル・ノードを指定すると、ローカル・ノードにファイルが存在していても、システムはネットワークを介してログインしてファイル操作を行います。リモート・システムへのアクセス方法については、『OpenVMS システム管理者マニュアル』を参照してください。

注意

これ以降、この章のノード名を指定する例には、必ずしもアクセス制御文字列は入っていません。これは、代理アカウントを使用すれば、ユーザはリモート・システムで操作を実行できるからです。

3.1.10 TCP/IP 使用によるリモート・ノード上のファイルへのアクセス

Compaq TCP/IP Services for OpenVMS では、ネットワークの他のホストにあるファイルのアクセスや転送に FTP(ファイル転送プロトコル) を使用します。FTP を使用するには、Compaq TCP/IP Services for OpenVMS へのアクセス許可がある OpenVMS システムのアカウントと、リモート FTP ホスト上のユーザ・アカウントが必要です。場合によっては、TCP/IP でアカウントとパスワードを指定せずにリモート・ホストに接続できます。その機能が有効でなければ、リモート・ホスト用のユーザ認証情報が必要です。FTP コマンドの使用方法については、『Compaq TCP/IP Services for OpenVMS User's Guide』を参照してください。

3.1.11 ネットワーク・ファイル指定の使用方法

ネットワーク・ファイル指定には、次の 3 つの形式があります。

- 一般的なファイル指定
- フォーリン・ファイル指定
- タスク・ファイル指定

どの形式でも、ノード指定にアクセス制御文字列を入れることができます。詳細は、『DECnet User's Manual』および『Compaq TCP/IP Services for OpenVMS User's Guide』を参照してください。

3.1.11.1 従来のファイル指定

ファイルの一般的な形式を以下に示します。

ノード:: デバイス:[ディレクトリ]ファイル名. タイプ; バージョン

3.1.11.2 フォーリン・ファイル指定

フォーリン・ファイル指定とは、OpenVMS 構文に従わないファイルのことである。たとえば、次のようなものがある。

ノード::"フォーリン・ファイル指定文字列"

次の例では、ファイル名に、疑問符 (?) という有効なファイル名文字として認識されない文字が含まれているため、ファイル名は引用符 (" ") で囲まなくてはなりません。また、アクセス中のリモート・ノードのオペレーティング・システムが認識する形式になっていなければなりません。

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

3.1.11.3 タスク指定文字列

タスク指定文字列は、リモート・ノードで実行されるプログラムを識別します。プログラムの内部でタスク指定文字列を使用すれば、プログラムはリモート・ノードの他のプログラムと通信できます。タスク指定文字列の形式は次のとおりです。

ノード::"タスク指定文字列"

この例のタスク指定は、リモート・ノード BOSTON のプログラム TEST2 を表しています。

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

注意

ファイルを UNIX システムにコピーしたり、ファイルを UNIX システムからコピーしたりする場合には制限事項があります。詳細は、『OpenVMS Record Management Utilities Reference Manual』を参照してください。

3.1.12 アクセス制御文字列の指定方法

アクセス制御文字列は、リモート・ノード上でログインできるアカウントを指定します。アクセス制御文字列を持つノード名は、次のような形式になります。

ノード"アクセス制御文字列"::

アクセス制御文字列は引用符 (" ") で囲み、その後にダブルコロンの (::) を付けます。

OpenVMS システムの場合、アクセス制御文字列は、ユーザ名と、それに続く 1 つ以上のスペースまたはタブとパスワードから構成されます。ACL についての詳しい説明は、第 10 章を参照してください。

次の例では、BOSTON はネットワーク・ノード名です。"HIGGINS ETUHCARAP" はアクセス制御文字列です。

- HIGGINS は BOSTON ノードでのユーザ名である。

- ETUHCARAP は、その名前に割り当てたパスワードである。

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

3.2 ファイル名でのワイルドカードの使用方法

1 つの DCL コマンドを、一度に 1 つのファイルではなく、複数のファイルに適用する場合には、ワイルドカード文字を使用します。入力されたファイル指定の一部に一致するファイルであれば、コマンドが適用されます。

本章で示す例の多くは、ファイル操作でのワイルドカード文字の使い方を示しています。DCL コマンドでのワイルドカード文字の使い方は、コマンドによって異なります。

多くの DCL コマンドでは、ディレクトリ名、ファイル名、またはファイル・タイプを指定する場合、ワイルドカードとしてアスタリスク(*)またはパーセント記号(%)の両方を使用できます (ディレクトリに対して使用できるワイルドカードについての説明は、第 4.5 節を参照してください)。バージョン番号はアスタリスク(*)を使用できますが、パーセント記号、ワイルドカードと数詞の組み合わせは使用できません。

OpenVMS バージョン 7.2 およびそれ以降を実行する Alpha システムでは、パーセント記号 (%) のかわりに疑問符 (?) を使用できます。

Extended File Specifications を使用するシステムで作業をしている場合、別のワイルドカード・オプションについては、第 5 章を参照してください。

3.2.1 アスタリスク(*)・ワイルドカード文字

アスタリスク(*)・ワイルドカード文字を使用する場合には、次の条件を満たしてください。

- ディレクトリ名、ファイル名、ファイル・タイプの各フィールドのフィールド全体またはその一部になっていること。
- バージョン番号フィールドの全体。一部のみを指定することはできない。

アスタリスク(*)・ワイルドカード文字は次の場合に使用できます。

- 個々のファイル名を指定せずに、多くのファイルを処理する場合
- 選択されるファイルを特定のグループに制限する場合
- ディレクトリ指定の内部で使用する場合

例

次の例では、ファイル指定は、[FROGMAN]ディレクトリの中のすべてのファイルのすべてのバージョンを選択します。

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

次の例では、現在の省略時のディレクトリの中で、ファイル・タイプ.DAT を持つファイルだけが表示されます。

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

次の例は、[FROGMAN]から 1 レベル下のサブディレクトリの中で、ファイル・タイプ.DAT を持つすべてのファイルを選択します。

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

次の例では、ワイルドカード文字をディレクトリ指定に使用しています。

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

このファイル指定は、現在の省略時のディスク上の 2 番目のレベルのサブディレクトリの中で、任意のファイル・タイプの AVERAGE というファイル名のすべてのファイルの全バージョンを選択します。たとえば、このファイル指定では、[A.B.C]AVERAGE.DAT は選択しますが、[X.Y]AVERAGE.DAT は選択しません。

3.2.2 パーセント記号(%)ワイルドカード文字

パーセント記号(%)ワイルドカード文字は、ファイル指定の中の任意の 1 文字の代わりとして使用します。パーセント記号は、ディレクトリ、ファイル名、ファイル・タイプの各フィールドで使用できます。ただし、バージョン番号フィールドや ANSI 磁気テープ・ファイル指定では使用できません。パーセント記号は、1 つのフィールドの中の 1 文字と置き換えられるので、置き換える対象となる文字が必要だからです。

パーセント記号はいくつでも指定でき、他のワイルドカード文字と組み合わせることもできます。

次の例は、DISTRICT の後に 1 文字が続くファイル名で、ファイル・タイプが.DAT であるファイルの最新バージョンを表示します。

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

この例で表示されるのは、ファイル DISTRICT1.DAT , DISTRICT2.DAT , DISTRICT3.DAT などです。ファイル DISTRICT4_5.DAT は、DISTRICT の後に 2 文字以上続いているので表示されません。また、ファイル DISTRICT.DAT も表示されません。

たとえば、次のような指定も可能です。

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

3.3 その他のファイル名

これ以降の節では、OpenVMS 環境でサポートされるファイル名の他のタイプについて説明します。

3.3.1 空のファイル名とファイル・タイプ

ファイル名やファイル・タイプなどのファイル指定の構成要素を指定しない場合、その多くは DCL コマンドやユーティリティによる (組み込み) 構文解析時に省略時の値に置き換えられます。たとえば、FORTRAN コマンドでは省略時のファイル・タイプまたは.FOR を使用します。以下のコマンドを実行すると、FORTRAN コンパイラはファイル FILE.FOR をコンパイルします。

```
$ FORTRAN FILE
```

また、DIRECTORY コマンドでは欠落した要素のかわりにアスタリスク・ワイルドカードが使用されます。たとえば以下のコマンドでは、ファイル・タイプに関係なく (ピリオド (.) を含む) ファイル名 FILE を持つすべてのファイルが表示されます。

```
$ DIRECTORY FILE
```

ファイルには空 (ヌル値) のファイル名、または区切りピリオドだけのファイル・タイプ (空のファイル・タイプともいう) を指定できます。たとえば、以下のファイル名は有効です。

```
.TMP  
TEMP.
```

3.3.1.1 空のファイル・タイプによるファイル参照

以下に示すように、区切り記号のピリオドだけのタイプによるファイル参照を使用できます。

```
$ DIRECTORY TEMP.      !
```

ファイル名区切り記号がないので、空のファイル名によるファイル参照はできません。ファイル名がないファイル参照は、ファイル名がないものと解釈されます。

以下のコマンドでは、ファイル.TMP だけでなく、タイプ.TMP のすべてのファイルのリストが表示されます。ディレクトリ・ユーティリティが、欠落しているファイル名を "*" に自動的に置き換えるからです。

```
$ DIRECTORY .TMP
```

3.3.2 磁気テープの代替ファイル名の使用方法

標準 (ODS-2 互換) ファイル名に加えて、オペレーティング・システムは ANSI ラベルの磁気テープの代替ファイル命名規則もサポートします。次の形式をとります。

"ファイル名"; バージョン

ファイル名には、ASCII "a"文字セットの 1 ~ 17 文字を指定できます。この文字セットには、数字、英大文字、スペースに加えて、次の文字も含まれます。

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

また、ANSI 磁気テープ・ファイル名では、アスタリスク(*)文字も使用できます。

詳細は、『Guide to OpenVMS File Applications』を参照してください。

3.4 ファイルの作成と変更

これ以降の節では、OpenVMS 環境でサポートされるツールやコマンドを使用して、ファイルを作成および変更する方法を説明します。

テキスト・ファイルを作成および変更できる会話型エディタです。OpenVMS に標準装備されているテキスト・エディタは、EVE と EDT ですが、これ以外のエディタも使用できます。

また、DCL のコマンド CREATE、COPY、RENAME を使用しても、ファイルを作成したり変更したりできます。この節では、これらのコマンドを使用して、ファイルを作成したり変更したりする方法について説明します。

Extended File Specifications を使用する環境で作業を行っている場合、その環境におけるファイルの作成、コピーについての詳しい情報については、第 5 章を参照してください。

3.4.1 ファイルの作成

CREATE コマンドはテキスト・ファイルを作成します。CREATE コマンドでファイルを変更することはできません。また、Enter を押した後は、前の行に戻って変更することもできません。CREATE コマンドで作成したファイルを変更するには、テキスト・エディタを使用します。Ctrl/Z はファイルの終端を知らせるもので、これを押すと、DCL コマンド・レベルに戻ります。

次の例では、CREATE コマンドを入力し、テキスト行を入力することにより、TEST.TXT という名前のファイルを作成します。

```
$ CREATE TEST.TXT
```



```
this is a test  
12345678  
Ctrl/Z
```

3.4.2 ファイルのコピー

COPY コマンドを使用すると、次のものを複製 (コピー) できます。

- 既存のファイルの内容を新しいファイルに複製できる。
- 一度に多くのファイルを複製できる。
- COPY コマンドに/SINCE 修飾子を指定すると、指定した条件を満足するファイルだけを複製できる。

例

次の例では、FEES.DAT ファイルが RECORDS.DAT にコピーされる。

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

次の例では、省略時のディレクトリ内のすべての.TXT ファイルが別のディレクトリにコピーされる。

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

次の例では、[JONES.LICENSES.DOG]ディレクトリ内のファイルのうち、1999 年 12 月 11 日以降に変更されたファイルだけが省略時のディレクトリにコピーされる。

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

3.4.3 ファイルの連結

COPY コマンドで、ファイルを連結することができます。たとえば、省略時のディレクトリで FEES1.DAT を FEES.DAT に追加する (FEES.DAT の新しいバージョンを作成する) 場合には、次のように入力します。

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

FEES.DAT の後のコンマとファイル名 FEES1.DAT の間に、スペースは必要ありません。

3.4.4 DECnet を使用してリモート・ノードからファイルをコピーする

別のノードにあるファイルを使用中のノードにコピーするには、COPY コマンドを使用します。たとえば、ノード CHAOS 上のディレクトリ DISK2:[PUBLIC]にあるすべてのファイルの最新バージョンを、省略時のディレクトリに同じ名前でコピーする場合には、次のように入力します。

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

3.4.5 DECnet を使用して使用中のノードにあるファイルをリモート・ノードにコピーする

使用中のノードにあるファイルをリモート・ノードにコピーする場合には、COPY コマンドを使用します。システム間でファイルをコピーしようとしたときに、保護違反または DECnet エラー・メッセージが表示された場合には、メールでファイルをコピーするか、または、アクセス制御文字列を使用して回避することができます。

たとえば、省略時のディレクトリにあるすべてのファイルの最新バージョンを、ノード CHAOS 上のディレクトリ DISK2:[STAFF_BACKUP]に同じ名前でコピーする場合には、次のように入力します。

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

3.4.6 TCP/IP を使用してリモート・システム上へファイルをコピーする

TCP/IP では、ネットワークの他のホストにあるファイルのアクセスや転送に FTP(ファイル転送プロトコル) サービスを使用します。リモート・ホストからローカル・ホストにファイルを転送するには、GET コマンドを使用します。ローカル・ホストからリモート・ホストにファイルをコピーするには、PUT コマンドを使用します。これらのコマンドを使用するには、リモート・ホストとのアクティブな FTP セッションが必要です。セッション時には、いくつでも FTP コマンドを入力できます。FTP コマンドの使用方法については、『Compaq TCP/IP Services for OpenVMS User's Guide』を参照してください。

たとえば、次のコマンドは、ファイル FEES.DAT をノード CHAOS の JONES アカウントに送信します。

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

3.4.7 ファイルをコピーするためのアクセス制御文字列の使用

保護違反を受信した後、ファイルをコピーするには、ファイル指定のノード名の後にアクセス制御文字列を指定できます(第 3.1.12 項を参照)。

次の例では、ユーザは CHAOS ノードに、ユーザ名が SMITH で、パスワードが SPG96PRT であるアカウントを持っています。ユーザは、省略時のディレクトリのすべてのファイルの最新バージョンを CHAOS のアカウントにコピーします。

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

3.4.8 ファイル名の変更

RENAME コマンドは、ファイルに新しい名前を付けます。また、場合によっては、そのファイルを異なるディレクトリに格納します。ファイルの変更後は、ファイル FEES.DAT;4 は省略時のディレクトリには存在しなくなることに注意してください。RENAME コマンドを使用するときは、入力位置と出力位置が同じデバイス上になければなりません。

次の例では、ファイル FEES.DAT が新しい名前 RECORDS.DAT に変更され、省略時のディレクトリから [SAVETEXT]ディレクトリに移動されます。

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

3.5 ファイルの内容の表示

これ以降の節では、OpenVMS 環境でサポートされるツールやコマンドを使用して、ファイルの内容を表示する方法を説明します。

3.5.1 TYPE コマンドの使用

ファイルの内容を画面に表示する場合には、DCL プロンプトに対して TYPE コマンドとファイル名を入力します。省略時の設定では、ファイルの最新バージョンが表示されるので、ファイル指定にバージョン番号を指定する必要はありません。

次の例では、STAFF_VACATIONS.TXT ファイルの最新のバージョンが表示されます。

```
$ TYPE STAFF_VACATIONS.TXT
```

3.5.2 表示内容を制御する

TYPE コマンドに /PAGE 修飾子を指定すると、一度に 1 画面分の情報だけを表示することができます。Enter を押せば、次の画面を見ることができます。

/READ_ONLY 修飾子を付けて会話型のテキスト・エディタ (たとえば、EVE や EDT) を起動した場合も、会話形式の編集コマンドを使用して、ファイルの中で移動したり特定の文字シーケンスを検索することができます。READ_ONLY 修飾子を指定すれば、ファイルの変更ができないので会話型のエディタを終了するときにファイルを誤って変更してしまう恐れがなくなります。

3.5.3 リモート・ノードのファイルを表示する

DECnet を使用している場合、リモート・ノード上のファイルの内容を表示するには、ファイル指定の中にノード名、ディスク、そしてディレクトリを指定します。

次の例では、COMPANY_HOLIDAYS.TXTファイル (リモート・ノード CHAOS に存在するファイル) が表示されます。

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

TCP/IP でファイルやリモート・ノードの内容を表示するには、FTP VIEW コマンドを使用し、ファイル名を指定します。ファイルが現在の作業ディレクトリにない場合、ディレクトリ名を加えてファイル名を指定してください。FTP VIEW コマンドについては、『Compaq TCP/IP Services for OpenVMS User's Guide』を参照してください。

3.5.4 ワイルドカードによるファイルの表示

アスタリスク (*)・ワイルドカードを使用すると、特定のファイルの全バージョンを表示できます。

たとえば、ディレクトリ[JONES]のファイル LOGIN.COM の全バージョンを表示する場合には、次のように入力します。

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

ディレクトリ[JONES]の中で、STAFF で始まるあらゆるファイル・タイプを持つすべてのファイルの全バージョンを表示する場合には、次のように入力します。

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

3.5.5 複数のファイルの表示

TYPE コマンド行に 2 つ以上のファイルを指定すると、指定した順序でファイルが表示されます。ワイルドカード文字を使用すると、ファイルはアルファベット順に表示されます。

3.6 ファイルの削除

DELETE コマンドは、ファイルをディレクトリから削除し、そのファイルが占有していたディスク領域を解放して他のファイルが使用できるようにします。DELETE コマンドを使用するときは、各ファイル指定にバージョン番号を指定するか、バージョン番号としてアスタリスク (*)・ワイルドカード文字を指定しなければなりません。

たとえば、ファイル POUND.LIS のバージョン 17 を削除する場合には、次のように入力します。

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

POUND.LIS のバージョン 16 と 17 を削除する場合には、次のように入力します。

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

ファイル POUND.LIS のすべてのバージョンを削除する場合には、次のように入力します。

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

ワイルドカード文字で複数のファイルを削除する場合には、/CONFIRM 修飾子を使用して、削除するたびに確認するとよいでしょう。また、削除する場合には、ファイルの名前を表示するとよいでしょう。この場合は、DELETE コマンドと一緒に/LOG 修飾子を指定します。

たとえば、サブディレクトリ[JONES.LICENSES.DOG]のすべてのファイルを削除してよいかどうかを確認する場合には、次のコマンドを入力します。

```
$ 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
```

次の例では、コマンドを入力すると、ファイルを削除した後でそのファイルの名前が表示されます。

```
$ 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)
```

3.6.1 PURGE コマンドの使用方法

PURGE コマンドは、省略時のディレクトリまたは指定されたディレクトリに存在する指定されたファイル(または全ファイル)の最新バージョン以外のバージョンをすべて削除します。ファイルを更新した後でファイルの古いバージョンをパージすると、ディスク上に未使用領域を確保できます。

たとえば、省略時のディレクトリにある各ファイルの2つの最新バージョンを残してすべてのバージョンをパージする場合には、次のように入力します。

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

3.7 他のユーザからのファイルの保護

これ以降の節では、ファイルを保護する方法を説明します。詳細は、以下を参照してください。

- ディレクトリの保護については第 4 章
- ファイル保護の変更については第 10 章

3.7.1 アクセス制御リスト (ACL)

自分以外のユーザがファイルをアクセスできないようにするには、保護を設定するか、ファイルの ACL を変更します。保護を設定したりファイルの ACL を変更する場合には、ファイルを所有しているか、ファイルの制御アクセス権を持っているか、GRPPRV, SYSPRV, BYPASS, READALL 特権を持っていないければなりません。

3.7.2 ファイル保護の種類

ファイル保護には、省略時の保護と明示的な保護の 2 種類があります。ファイルを作成すると、そのファイルには通常、親ディレクトリと同じ保護が割り当てられます。これが省略時の保護です。CREATE/PROTECTION コマンドを使用してファイルを作成した場合や、SECURITY/PROTECTION コマンドを使用して既存のファイルの保護を変更する場合には、明示的なファイル保護を使用します。

ファイルを完全に保護するには、そのファイルが存在するディレクトリにも同等の保護またはそれ以上の保護を適用しなければなりません。

3.8 ファイルの印刷

これ以降の節では、ファイルを印刷する手順を説明します。

1 つまたは複数のファイルを印刷するには、PRINT コマンドを使用します。PRINT コマンドは、印刷キューといわれる印刷するジョブのリストに印刷ジョブ (印刷するすべてのファイル) を登録します。PRINT コマンドに指定されるファイルの省略時のファイル・タイプは、.LIS または最後に明示的に指定されたファイル・タイプになります。システムは、ジョブ名、キュー名、ジョブ番号を表示するとともに、ジョブの状態も示します。

省略時の設定では、ジョブ名は、PRINT コマンドの最初の (または唯一の) ファイル指定の名前になります。ジョブをキューに登録した後は、ジョブ番号を使用してそのジョブを指定します。ジョブをキューに登録すると、他のジョブがそれ以前にキューに登録されていないくて、プリンタが物理的に印刷を開始できる状態であれば、ジョブが印刷されます。

次の例では、3つのファイルを含むプリント・ジョブが省略時のプリント・キューである SYSS\$PRINT に登録されます。

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

PRINT コマンドの省略時のファイル・タイプは.LIS であるため、POUND.LIS、MALE.LIS、FEES.DAT ファイルがキューに登録されます。ジョブ名は POUND であり、キュー名は SYSS\$PRINT であり、ジョブ番号は 202 です。

3.8.1 印刷ジョブの優先順位

印刷キューは、一度に1つのジョブしか実行できません。印刷ジョブは、その優先順位に従って印刷順がスケジューリングされ、最も高い優先順位を持つジョブが最初に印刷されます。同じ優先順位を持つジョブが2つ以上あるときには、普通は最も小さいジョブが最初に印刷されます。優先順位が同じでサイズも等しいジョブが2つ以上あるときは、キューへの登録順に印刷されます。優先順位は、システム管理者が決定するか、PRINT コマンドに/PRIORITY 修飾子を入力することによって決定できます。優先順位のスケジューリングについての詳しい説明は、『OpenVMS システム管理者マニュアル』を参照してください。

3.8.2 キュー情報の表示

通常、省略時の印刷キュー SYSS\$PRINT は、ユーザのシステムに固有なスタートアップ・プロシージャの一部として起動されます。次の表は、キューに関する情報を表示するために使用できるコマンドを示しています。

表示対象	入力するコマンド
サイトのキュー	SHOW QUEUE
プリント・ジョブの状態	SHOW ENTRY
他のユーザがキューに登録したジョブ	SHOW ENTRY/USER_NAME=username
特定の1つ以上のジョブに関する情報	SHOW ENTRY job-name
	SHOW ENTRY entry-number

次の例では、SHOW ENTRY コマンドを使用して、キューに登録されているプリント・ジョブに関する情報を表示します。

```
$ SHOW ENTRY

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

3.8.3 プリント・フォーム

プリント・フォームは次の機能を実行します。

- 特定のページ形式属性 (マージンやページ長など) を指定する。
- フォームに指定された用紙ストックに応じて、ジョブが印刷可能であるかどうかを判断する。

印刷に対する要件が限られている場合には、特殊なフォームを使用する必要はありません。コンパックはすべてのキューに対して、システム全体で有効な省略時のフォーム (DEFAULT という名前) を提供しています。また、システム管理者もプリント・フォームを作成できます。出力を書式化しなければならない場合や、特定のプリント・ジョブで特殊な用紙が必要な場合には、システム管理者にご相談ください。

3.8.4 印刷ジョブの停止

印刷ジョブを停止して、印刷キューからその印刷ジョブを削除する場合には、DELETE/ENTRY コマンドにエントリ番号パラメータを入力します。

たとえば、エントリ 202 を削除する場合には、次のように入力します。

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

3.8.5 別のノードにあるファイルの印刷

DECnet や TCP/IP Services では、別のシステムにファイルの印刷ができます。

システム・マネージャは、TCP/IP を使用してリモート・ネットワーク・ホスト上のプリント・キューに DCL PRINT コマンドでプリント・ジョブを転送するための LPR(Line Printer Remote) ネットワーク・サービスと LPD(Line Printer Daemon) ネットワーク・サービスをシステムに構成します。リモート・ホストは UNIX システムでも、LPR/LPD を実行する別の OpenVMS システムでもかまいません。LPR/LPD ネットワーク・サービスでは、以下の操作ができます。

- リモート・ネットワーク・ホストに接続したプリンタにプリント・ジョブを送信
- プリント・キュー・ステータスを表示
- プリント・ジョブの取り消し
- UNIX システムで開始したプリント・ジョブをローカル OpenVMS システムのプリント・キューで受信
- SMTP メールで「終了」通知を受信

LPR/LPD コマンドによるファイルの印刷方法については、『Compaq TCP/IP Services for OpenVMS User's Guide』を参照してください。

DECnet では、別のシステム上のファイルを印刷できます。また、そのファイルをリモート・ノードにコピーしたり、PRINT コマンドに/REMOTE 修飾子を指定することができます。

次の例では、COMPANY_HOLIDAYS.TXT ファイルがローカル・ノードからリモート・ノード CHAOS にコピーされ、ファイルは印刷のために、CHAOS ノードの省略時のシステム・プリント・キュー (SYSS\$PRINT) に登録されます。

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

この例では、アクセス制御文字列は、ユーザ JONES が、ノード CHAOS 上のディレクトリ[JONES]にファイルをコピーする権限を持つことを示しています。ファイル指定の最後のアスタリスク(*)・ワイルドカードは、ファイルをリモート・ノードにコピーする場合に、ファイル名 COMPANY_HOLIDAYS.TXT を使用するようにシステムに指示します。

注意

PRINT コマンドのすべての修飾子が/REMOTE 修飾子と互換性があるとは限りません。たとえば、印刷キューの中にはジョブを特定のキューに登録できないものがあり、ジョブはすべて省略時のシステム印刷キュー (SYSS\$PRINT) に登録されます。/REMOTE 修飾子と互換性のある PRINT コマンドの修飾子のリストについては、『OpenVMS DCL ディクショナリ』の PRINT コマンドの/REMOTE 修飾子の説明を参照してください。

3.8.6 印刷コマンドの修飾子

プリント・ジョブは、PRINT コマンドに対する修飾子を使用して、さまざまな方法で制御できます。たとえば、印刷部数を指定したり、印刷ジョブが完了したときに通知を出すようにシステムに要求したりできます。

ここに示した修飾子の他に、システムで DECprint Supervisor ソフトウェアを実行している場合には、ランドスケープ印刷や両面印刷など、さまざまな方法で印刷するために/PARAMETER 修飾子を使用できます。システムで使用できるプリント・オプションの一覧については、システム管理者に質問してください。

次の表は PRINT コマンドの修飾子を示しています。PRINT コマンドについての詳細は、『OpenVMS DCL ディクショナリ』またはオンライン・ヘルプを参照してください。

印刷操作	印刷ジョブ・コマンドと修飾子
コピー部数 ジョブ別 ファイル別 指定されたファイルのみ	PRINT/JOB_COUNT=n ¹ PRINT/COPIES=n ¹ ファイル指定/COPIES=n ¹
ページ数	PRINT/PAGES= ¹
印刷機能 フラグ・ページ 用紙タイプ 特殊機能 ダブルスペース ページ見出し	PRINT/FLAG= ¹ PRINT/FORM= ¹ PRINT/CHARACTERISTICS= ¹ PRINT/SPACE ¹ PRINT/HEADER ¹
ジョブ実行の通知	PRINT/NOTIFY
ジョブ実行の延期 実行時刻指定 無期限	PRINT/AFTER PRINT/HOLD
延期しているジョブの解放	SET QUEUE/ENTRY/RELEASE
印刷ジョブの表示	SHOW ENTRY
印刷ジョブの停止 ジョブの削除 現在印刷しているジョブを 停止して、キューの中の次 のジョブの印刷を開始する	DELETE/ENTRY=ジョブ番号 STOP/ABORT
現在印刷しているジョブを 停止して、再登録して印刷 し直す	STOP/REQUEUE
ジョブ完了後、ジョブをキューに残す	PRINT/RETAIN

¹SET QUEUE/ENTRY コマンドに修飾子を使用すると、キューに登録されているが、まだ印刷されていない印刷ジョブにこれらの操作を指定できる。

3.8.7 WWPPS ユーティリティ (Alpha のみ)

WWPPS(World-Wide PostScript Printing Subsystem) は、さまざまな言語の文字で構成されたテキスト・ファイルを任意の PostScript プリンタで印刷するためのユーティリティです。PostScript 印刷可能ファイルにフォント・データを組みこめば、ローカル言語フォントがないプリンタにもその言語文字を印刷できます。

注意

PostScript 印刷可能ファイルにフォント・データを組みこむと、プリンタ・メモリがサポートできるサイズを超える場合があります。その場合、WWPPS では印刷された出力の最後にエラー・ページを追加してファイル・サイズがプリンタの容量を越えたことを知らせます。

中国語、韓国語、日本語などのローカル言語文字を印刷する場合、少なくとも 24MB のメモリを備えたプリンタを使用してください。

サポートされる言語

WWPPS は以下の言語をサポートしています。

- キリル語 (ISO8859-5)
- ギリシア語 (ISO8859-7)
- ヘブライ語 (ISO8859-8)
- 日本語 (Super DEC Kanji)
- 韓国語 (DEC Korean)
- Latin 1 (ISO8859-1)
- Latin 2 (ISO8859-2)
- Latin 4 (ISO8859-4)
- 簡体字中国語 (DEC Hanzi)
- タイ語 (TACTIS)
- 繁体字中国語 (Taiwanese EUC/DEC Hanyu)
- トルコ語 (ISO8859-9)
- Unicode

文字の処理では、WWPPS は文字を現在のロケールで印刷できるかどうかを確認します。ロケール設定は、OpenVMS のインストール時に Compaq C for OpenVMS Run-Time Library(RTL) で提供されます。16 ビット Unicode や ISO 10646(USC-4) 形式以外のファイルでは、英語以外の言語文字によるファイルを印刷する場合、あらかじめ適切なロケールを設定する必要があります。プロセスのロケール設定が入力ファイルに適していない場合、/LOCALE 修飾子でロケールをそのプリント・ジョブに指定できます。

サポートされるコードセット

OpenVMS システムでは、以下のコードセットをサポートしています。

コードセット	コードセット名
DECHANYU	繁体字中国語用の DECHanyu (プレーン 1 とプレーン 2 のみ)
DECHANZI	簡体字中国語用の DECHanzi
DECKOREAN	韓国語用の DECKorean
GB18030	簡体字中国語および繁体字中国語用の 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	日本語用の Super DEC Kanji
TACTIS	タイ語用の TIS-620

上記のコードセットは、いずれも WWPPS でサポートしていますが、1 度に関連付けることができるフォントは各コードセットごとに 1 言語だけです。

WWPPS では、タイ語を除く上記のすべてのコードセットについて、Unicode 文字変換機能もサポートしています。Unicode 文字は上記のコードセットのどれかの文字に変換され、PostScript ファイルでは、そのコードセットをサポートするフォントがその文字に適用されます。文字を変換できない場合、その位置にはスペースが印刷されます。

3.8.7.1 WWPPS の呼び出し

WWPPS 用のフォーリン・コマンドがシステム・マネージャによって設定されていない場合、以下の行を LOGIN.COM に追加して設定します。

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

DCL プロンプトから WWPPS ユーティリティを呼び出すには、以下のように入力します。

```
$ WWPPS
```

3.8.7.2 WWPPS ユーティリティ・コマンド

以下に、WWPPS ユーティリティで使えるコマンド、パラメータ、修飾子を説明します。説明の後に例を示します。

EXIT

WWPPS セッションを終了し、DCL コマンド・レベルに戻ります。Ctrl/Z または Ctrl/C を入力しても WWPPS セッションは終了します。

```
WWPPS> EXIT
```

HELP

WWPPS (World-Wide PostScript Printing Subsystem) に関する情報を入手できます。

```
WWPPS> HELP PRINT
```

個々のコマンドやトピックに関する情報を入手するには、コマンド名やトピック名を指定して HELP コマンドを入力します。

```
HELP [topic]
```

PRINT

1 度にテキスト・ファイルを 1 つずつ、印刷可能な PostScript ファイルに変換し、プリンタ・キューに発行します。文字は標準フォントまたは太字で印刷できます。

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

/QUEUE 修飾子は、file-specによって指定されたテキスト・ファイルを送るキューの名前を指定するために、すべての PRINT コマンドに必要です。たとえば、次のコマンドは、ファイル REPORT.TXT を PRT_QUEUE プリンタ・キューに発行して、American English (/LOCALE 修飾子で指定された言語) で印刷します。

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

PRINT コマンドのオプションの修飾子は以下のとおりです。

- /COPIES

印刷するコピー数を指定します。省略時のコピー数は 1 です。

- /INDENTATION

左マージンからのインデントを文字数で指定します。省略時の設定は /INDENTATION=0(インデントなし) です。使用できる最大値は/PAPER_SIZE と/ORIENTATION の指定 (または省略時の設定) 値によって異なります。

/PAPER_SIZE	/ORIENTATION	/INDENTATION の最大値
LETTER	PORTRAIT	39
A4	PORTRAIT	38
LETTER	LANDSCAPE	65
A4	LANDSCAPE	67

- /LENGTH

行数でページ長を指定します。省略時の長さは LETTER サイズで 66 行, A4 サイズで 68 行です。

- /LOCALE

WWPPS で入力ファイルを変換するときのロケール設定を指定します。Unicode 形式 (UTF-20) のテキスト・ファイルに/LOCALE は指定する必要はありません。

ロケールは以下の規則で構成されています。

language_country_codeset.LOCALE

言語と国はいずれも OSF 命名規則で定義した 2 文字で表します (使用できる値については/LOCALE サブトピックを参照)。たとえば, EN_US_ISO8859-1 は米国で話される英語のロケールを表します。

省略時の設定で, WWPPS はシステム指定ロケールまたはプロセス指定ロケールを使用します。システム指定ロケールまたはプロセス指定ロケールがない場合, 省略時の設定は/LOCALE=C です。

システムで指定したロケールを表示するには, 以下のコマンドを入力します。

\$ LOCALE SHOW PUBLIC

表 3-1 は, 一般に組み合わせとなっている言語コードと国コードを整列します。

表 3-1 一般的な言語コードと国コードの組み合わせ

言語コード	言語	国コード	国
CA	カタロニア語	ES	スペイン
ES	スペイン語		
CS	チェコ語	CZ	チェコ共和国
DA	デンマーク語	DK	デンマーク
DE	ドイツ語	CH	スイス
		DE	ドイツ
EL	ギリシア語	GR	ギリシア
EN	英語	GB	イギリス
		US	アメリカ合衆国
FI	フィンランド語	FI	フィンランド
FR	フランス語	BE	ベルギー
		CA	カナダ
		FR	フランス
HE	ヘブライ語	IL	イスラエル
IW	ヘブライ語		
HU	ハンガリー語	HU	ハンガリー
IS	アイスランド語	IS	アイスランド
IT	イタリア語	IT	イタリア
JA	日本語	JP	日本
KO	韓国語	KR	韓国
LT	リトアニア語	LT	リトアニア
NL	オランダ語	NL	オランダ
NO	ノルウェー語	NO	ノルウェー
PL	ポーランド語	PL	ポーランド
PT	ポルトガル語	PT	ポルトガル
RU	ロシア語	RU	ロシア
SK	スロヴァキア語	SK	スロヴァキア
SL	スロヴェニア語	SI	スロヴェニア
SV	スウェーデン語	SE	スウェーデン
TH	タイ語	TH	タイ
ZH	中国語	HK	香港
		TW	台湾
		CN	中華人民共和国

OpenVMS システムでサポートされるコードセットの一覧は、第 3.8.7 項のサポートされるコードセットにあります。

- /ORIENTATION

論理ページ上の印刷出力の向きを PORTRAIT(省略時の設定) または LANDSCAPE で指定します。

- /PAPER_SIZE

LETTER(省略時の設定) または A4 で用紙のサイズを指定します。

- /RANGE

開始ページ番号mと終了ページ番号nで、印刷するページを指定します。または、ページを指定して印刷するかわりに ODD と指定すると奇数ページが印刷され、EVEN と指定すると偶数ページが印刷されます。省略時の設定では、ドキュメント全体が印刷されます。

- /VERTICAL

中国語、韓国語、日本語のマルチバイト文字用に縦書きモードを指定します。
/VERTICAL を指定すると、マルチバイト文字が反時計回りに 90 度回転し、左から右に行が印刷されます。印刷するページを時計回りに 90 度回転すると、文字は右から左に縦書きで読み取られます。縦書きモードでも、英語などの言語のシングルバイト文字は左から右に横に印刷されます。

- /WIDTH

ページ幅をカラム数で指定します。有効値は以下のとおりです。

- 80 (LETTER サイズ用紙と PORTRAIT 方向用)
- 132 (LETTER サイズ用紙と LANDSCAPE 方向用)
- 78 (A4 サイズ用紙と PORTRAIT 方向用)
- 136 (A4 サイズ用紙と LANDSCAPE 方向用) 省略時の値は/WIDTH=80 です。

ディレクトリ・ファイルの編成

ディレクトリは、ファイルの名前と位置を含む特殊なファイルです。たとえば、システム管理者がユーザ・アカウントを作成すると、そのユーザ名と同じ名前を持つディレクトリが作成されます。ユーザ名が JONES の場合、ディレクトリは[JONES]になります。

サブディレクトリは、別のディレクトリ・ファイルやサブディレクトリ・ファイルの中にあるディレクトリ・ファイルです。サブディレクトリを使用すれば、ファイルを用途ごとに編成することができます。たとえば、メモ用のサブディレクトリと状態レポート用のサブディレクトリを作成することができます。

ディレクトリと同じように、サブディレクトリにも、その中に登録されているファイルの名前とポインタが格納されます。サブディレクトリの中にも別のサブディレクトリを登録することができます。このような構造 (最上位ディレクトリとサブディレクトリ) を階層ディレクトリ構造と呼びます。

ユーザがアクセスするファイルは、ディスクに格納されています。各ディスクにはメイン・ディレクトリがあり、これをMFD (マスタ・ファイル・ディレクトリ)といいます。MFD には、UFD (ユーザ・ファイル・ディレクトリ)を格納します。UFD をユーザの最上位ディレクトリといいます。ほとんどの場合、システム上の各ユーザについて1つのUFDが存在します。UFD には、ユーザのディレクトリに登録されているファイルの名前とポインタが格納されます。最上位ディレクトリは、通常はプロセス・デフォルト・ディレクトリです。アカウントを特に変更しない限り、システムは自動的にユーザの最上位ディレクトリをログイン時のプロセス・デフォルト・ディレクトリにします。

完全なファイル指定のデバイス (ディスク) およびディレクトリ部分を一般にファイル・パスと呼びます。パスは、ファイル名とファイル・タイプ (およびバージョン) とともに、完全なファイル指定を構成します。完全なファイル指定には、システムがファイルを検索して識別するために必要なすべての情報が含まれています。¹

部分的なファイル指定に対するシステムによる省略時の設定の適用方法についての詳細は、『Guide to OpenVMS File Applications』を参照してください。

本章では、ディレクトリを使用してファイルを編成し、管理する方法を説明します。以下のことについて説明します。

¹ ファイルは磁気テープに格納することもできますが、磁気テープはディレクトリ構造ではありません。テープに格納されたファイルにアクセスするには、ファイル情報だけを含んだファイル指定を使用します。

- ディレクトリ構造
- ディレクトリについて
- 省略時の設定
- 他のユーザからのディレクトリの保護
- ワイルドカードを使用したディレクトリ構造の検索
- UIC 形式でのディレクトリの操作

注意

この例ではノード名を指定するときに、アクセス制御文字列を指定していないこともあります。このような例では、代理アカウントを使用すれば、リモート・システムで操作を行えるからです。

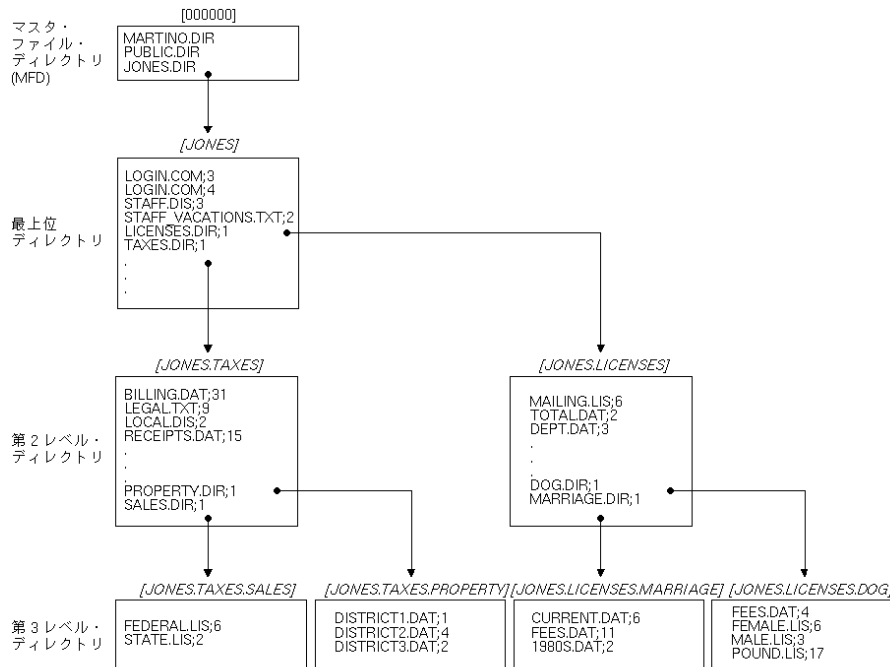
Extended File Specifications を使用する環境で作業を行っている場合、ディレクトリ構造と構文は、ここで説明している従来からの構造とは異なります。このような環境でのディレクトリ操作については、第 5 章を参照してください。

4.1 ディレクトリ構造

図 4-1 に、ディレクトリ階層のサンプルを示します。このディレクトリ構造の最上部は、マスタ・ファイル・ディレクトリ (MFD) です。このマスタ・ファイル・ディレクトリには、[000000]という名前が付けられています。図 4-1 に示す MFD には、ユーザ・ファイル・ディレクトリ MARTINO.DIR, PUBLIC.DIR, JONES.DIR のエントリが登録されています。最上位ディレクトリ[JONES]は、マスタ・ファイル・ディレクトリ[000000]中の JONES.DIR;1 という名前のユーザ・ファイル・ディレクトリのことです。

図 4-1 に示したサンプル・ディレクトリ構造は、本章の多くの例で使われます。

図 4-1 ディレクトリ構造



JRD-1746-GE

以下は、このディレクトリ構造についての説明です。

- ここでは、ユーザが[JONES]である場合を想定します。ログインすると、ユーザは省略時のディレクトリである[JONES]に入ります。

- [JONES]には、次の4つのディレクトリでないファイルが入っています。

LOGIN.COM;3
LOGIN.COM;4
STAFF.DIS;3
STAFF_VACATIONS.TXT;2

- [JONES]には、次の2つのディレクトリ・ファイルも入っています。

LICENSES.DIR;1
TAXES.DIR;1

- ディレクトリ・ファイル LICENSES.DIR;1 は、[JONES.LICENSES]サブディレクトリを指しています。
- TAXES.DIR;1 は、[JONES.TAXES]サブディレクトリを指しています。
- [JONES.LICENSES]サブディレクトリには、3つの非ディレクトリ・ファイルと2つのディレクトリ・ファイルが入っています。
- ディレクトリ・ファイル DOG.DIR;1 は、[JONES.LICENSES.DOG]サブディレクトリを指しています。

- MARRIAGE.DIR は、[JONES.LICENSES.MARRIAGE]サブディレクトリを指しています。

4.2 ディレクトリとは

ファイル指定のディレクトリの内容は最上位ディレクトリ名 (UFD など) からなり、最上位ディレクトリ名の後には多くのサブディレクトリ名を続けて指定できます。サブディレクトリ名はピリオド (.) で区切ります。

バージョン 7.2 より前のバージョンの OpenVMS Alpha および OpenVMS VAX のすべてのバージョンは、UFD と最大 7 つのサブディレクトリ名をサポートします。OpenVMS Alpha バージョン 7.2 およびそれ以降では、ディレクトリに最大 255 の名前 (UFD とサブディレクトリ) をサポートします。

ディレクトリ指定は、次の形式になります。

[ディレクトリ. サブディレクトリ]

1 つ以上のサブディレクトリ・レベルを追加するには、各サブディレクトリごとに、ピリオドとサブディレクトリ名を 1 つ追加します (限度まで)。別のサブディレクトリのサブディレクトリを指定するには、階層で 1 つ上のサブディレクトリにそのサブディレクトリ名 (前にピリオドを指定) を結合します。

OpenVMS Alpha バージョン 7.2 より前のバージョン、OpenVMS VAX のすべてのバージョン、および ODS-2 ディスクを使用する OpenVMS Alpha システムでは、サブディレクトリ名は最長 39 文字です。

ODS-5 ディスクを使用する OpenVMS Alpha バージョン 7.2 およびそれ以降では、サブディレクトリ・ファイルが <subdirectory-name>.DIR;1 の形式で保存されるため、サブディレクトリ名はファイル名の制限を受けます。ファイル指定におけるディレクトリとルート名の合計文字数 (区切り記号の括弧とピリオドを除く) は 512 文字までです。

4.2.1 ディレクトリの作成

ディレクトリを作成する場合、CREATE/DIRECTORY コマンドを使用します。現在のディレクトリの下にサブディレクトリを作成する場合には、現在のディレクトリ名の代わりにピリオドを指定できます。

たとえば、[JONES.TAXES]ディレクトリを作成する場合には、次のように入力します。

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

現在の省略時のディレクトリが[JONES]の場合、次のように入力します。

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

4.2.2 ディレクトリの表示

ディレクトリの中にあるファイルの名前を表示する場合は、`DIRECTORY` コマンドを使用します。サブディレクトリの中のファイルを表示する場合には、`DIRECTORY` コマンドにサブディレクトリ名を指定します。

`DIRECTORY` コマンドに特定のコマンド修飾子を指定すると、ファイルの名前以外の情報を検索することができます。`DIRECTORY` コマンドで使用できる修飾子については、『OpenVMS DCL デクショナリ』またはオンライン・ヘルプを参照してください。

たとえば、[JONES]の中のファイルを表示する場合には、次のように入力します。この例では、[JONES]の中に2つのサブディレクトリ([JONES.LICENSES]と[JONES.TAXES])と4つの非ディレクトリ・ファイル(STAFF.DIS、STAFF_VACATIONS.TXT、およびLOGIN.COMの2つのバージョン)が入っています。

```
$ 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.
```

省略時のディレクトリが[JONES]である場合、[JONES.LICENSES]サブディレクトリの内容を表示するには、次のように入力します。

```
$ 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.
```

4.2.3 ディレクトリの削除

ディレクトリを削除する場合は、次の手順に従ってください。

手順	操作
1	ディレクトリにファイルが登録されていないことを確認する。ディレクトリにファイルが登録されているかどうかを調べる場合は、 <code>DIRECTORY</code> コマンドを入力する。 ディレクトリの中にファイルがない場合には、次のようなメッセージが表示される。 <code>%DIRECT-W-NOFILES, no files found</code>
2	ディレクトリの中にファイルがある場合には、それを別のディレクトリにコピーして保存する。ファイルを保存しない場合には、ファイルを削除する。ディレクトリにサブディレクトリがある場合には、そのサブディレクトリを調べ、中にあるファイルをコピーまたは削除してから、そのサブディレクトリを削除する。
3	削除したいディレクトリより 1 レベル上のディレクトリに移動する。サブディレクトリは、ディレクトリにファイルとして存在する。ディレクトリを削除するには、そのディレクトリを指すファイルを削除する。
4	ファイルの削除が許容されるように、ディレクトリのファイル保護を変更する。マスタ・ファイル・ディレクトリに登録されているディレクトリ・ファイルを削除するためには、 <code>SYSPRV</code> 特権が必要。ファイル保護についての詳細は、第 3 章を参照のこと。
5	<code>DELETE</code> コマンドを使用してディレクトリ・ファイルを削除する。

次の例に、`[JONES.LICENSES]` サブディレクトリを削除する方法をまとめて示します。

```
$ 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
```

4.3 省略時のディレクトリの設定

省略時のディレクトリを変更する場合は、`SET DEFAULT` コマンドを使用します。新しく変更した省略時の値は、別の `SET DEFAULT` コマンドを入力するか、システムからログアウトするまで有効です。省略時のサブディレクトリを設定する場合には、それより 1 レベル上のディレクトリの名前に、サブディレクトリ名を追加します。

次の例では、省略時の設定はディレクトリ `[JONES]` に設定され、ファイル `[JONES]STAFF_VACATIONS.TXT` が表示されます。

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

次の例では、サブディレクトリ[JONES.TAXES]に格納されているファイル BILLING.DAT が表示されます。

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

4.3.1 省略時の値を存在しないディレクトリに設定する

このオペレーティング・システムでは、省略時の値を存在しないディスクまたはディレクトリに設定できます。省略時の値を存在しないディレクトリに設定した後、ファイルを操作しようとする、ディレクトリが存在しないことを示すメッセージが表示されます。現在の設定が存在しないディスクまたはディレクトリであるため、必要な操作を実行できない場合には、省略時の値を既存のディスクまたはディレクトリに設定してください。

4.3.2 SHOW DEFAULT コマンド

現在の省略時のディレクトリを表示する場合には、次の例に示すように、SHOW DEFAULT コマンドを入力します。

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

SET DEFAULT コマンドを使用すると、省略時のデバイスも変更できます。省略時の値は、別の SET DEFAULT コマンドを入力するか、システムからログアウトするまで有効です。コマンドの中にディレクトリを指定しなくても、省略時に設定したいデバイスを指定することができます。

次の例に、省略時のデバイスの変更方法を示します。

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

次の例では、[JONES]というディレクトリが、DISK1 と DISK2 の両方のデバイスに存在することを示しています。

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

4.3.3 一時的な省略時の値の使用

複数のファイルを入力する場合、それぞれのファイルに完全なファイル指定を指定しないと、ノード名、デバイス名、ディレクトリ名に対して一時的な省略時の値が適用されます。現在の省略時のディレクトリを一時的な省略時の値として使用する場合、空の大括弧を使用します。複数のファイル指定の1つにノード名を指定した場合には、ダブルコロンのを使用して、一時的な省略時の値を無効にできます。

たとえば、次のコマンドは、[STATS]ディレクトリにある A.LIS と B.LIS ファイルを[RESULTS]ディレクトリにコピーします。

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

この例において、[STATS]は B.LIS ファイルに対する一時的な省略時のディレクトリとなります。システムは、[STATS]A.LIS の前半のファイル指定を使用して、省略時のディレクトリを判別します。

次のコマンドは、1つの一時的な省略時のデバイスと2つのディレクトリを使用しています。

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

この例では、3つのファイル (A.LIS, B.LIS, C.LIS) がすべて BASE デバイスからコピーされています。A.LIS ファイルは[STATS]ディレクトリからコピーされ、残りの2つのファイルは[TIME]ディレクトリからコピーされます。

現在の省略時のディレクトリが[BETA]である場合、次のコマンドは、[ALPHA]TEST.DAT と[BETA]FINAL.DAT ファイルを[RESULTS]ディレクトリにコピーします。

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

4.4 他のユーザからのディレクトリの保護

ファイルを完全に保護するためには、ファイルが格納されているディレクトリにも同じ保護が必要です。たとえば、あるファイルへのアクセスがすべて拒否されているのに、そのファイルのディレクトリへの読み込みアクセスが許可されている場合には、ファイルは存在することを確認できるが中身を見ることができない、という現象が起きます。逆に、あるファイルへのアクセスは許容されているのに、そのファイルのディレクトリ (またはその親ディレクトリ) へのアクセスが拒否されている場合には、そのファイルが存在することさえわかりません。

注意

重要なファイルを保護するためには、ディレクトリ保護だけでは不十分です。ディレクトリの中のそれぞれのファイルも保護しなければなりません。

省略時の設定において、最上位ディレクトリは UIC に基づく保護 S:RWE,O:RWE,G:RE,W:E を受け、ACL に基づく保護は受けません。サブディレクトリは、親ディレクトリから UIC に基づく保護を受けます。保護コードについての詳細は、第 10.3 節を参照してください。

ディレクトリを作成するときに UIC に基づく保護を明示的に指定する場合は、CREATE/DIRECTORY コマンドに /PROTECTION 修飾子を指定します。ディレクトリの ACL はディレクトリ作成時に指定することはできません。既存のディレクトリに対する UIC ベースの保護を変更するには、SET SECURITY/PROTECTION コマンドをディレクトリ・ファイルに適用します。

ディレクトリへのアクセスを禁止せずにある制限を加えるためには、読み込みアクセスを指定せずに実行アクセスを指定します。ディレクトリに対する実行アクセス権がある場合には、そのディレクトリに格納されているファイルを調べたり、読み込むことができます。つまり、ファイル指定がわかっている場合には、そのファイルを調べることができますが、ディレクトリ内のファイルの一覧を表示することはできません。セキュリティについての詳細は、『OpenVMS Guide to System Security』を参照してください。

4.5 ワイルドカードを使用したディレクトリ構造の検索

ディレクトリ構造のどこからでも、そのディレクトリ構造の中の別のディレクトリやサブディレクトリを参照することができます。この場合、参照したいディレクトリやサブディレクトリを名前で直接指定するか、反復記号(...)やハイフン(-)のワイルドカード文字を使用します。ワイルドカードについての詳細は、第 3.2 節を参照してください。

Extended File Specifications を使用する環境で作業を行う場合、ワイルドカード文字によるディレクトリ構造の検索については、第 5 章を参照してください。

4.5.1 反復記号(...)ワイルドカード文字

ディレクトリ構造を下向きに検索する場合、反復記号(...)ワイルドカード文字を使用します。現在のディレクトリとその中のすべてのサブディレクトリの中を検索する場合には、次に示すように反復記号だけを使用します。

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

ディレクトリ指定の先頭に反復記号を置くと、現在のディレクトリから検索が開始されます。ただし、ディレクトリ指定の先頭にピリオドを付けると、現在のディレクトリより 1 レベル下のサブディレクトリだけが検索されます。

ディレクトリ構造の中から、すべての最上位ディレクトリとそのサブディレクトリを検索する場合は、アスタリスク(*)の後に反復記号(...)を使用します。

たとえば、現在のディレクトリが[JONES]の場合、[JONES]ディレクトリと[JONES]のすべてのサブディレクトリの中にある FEES.DAT という名前を持つすべてのファイルの最新バージョンを表示するためには、次のように入力します。

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

たとえば、現在の省略時のディレクトリが[JONES]の場合、ディレクトリ指定の末尾が.SALES であるすべてのサブディレクトリにおいて、FEDERAL.LIS というファイルの最新バージョンを表示するには、次のように入力します。

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

次のコマンドは、[JONES]と[JONES]のすべてのサブディレクトリにおいて DEPT.DAT という名前を持つすべてのファイルの最新バージョンを表示します。

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

現在のディレクトリが[JONES]の場合、次のコマンドは、[JONES]より 1 レベル下の[.LETTERS]サブディレクトリにおいて、MAILING.LIS という名前を持つすべてのファイルを検索します。このとき、[JONES.LICENSES]サブディレクトリは検索されますが、[JONES.LICENSES.MARRIAGE]は検索されません。

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

現在のディレクトリが[JONES]の場合、次のコマンドは、[JONES]ディレクトリの[.LICENSES]サブディレクトリと、[.LICENSES]サブディレクトリの中のすべてのサブディレクトリにおいて、DEPAT.DAT という名前を持つすべてのファイルの最新バージョンを表示します。

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

次のコマンドは (READALL 特権が必要)、最大 8 つのレベルのディレクトリ名 (最上位ディレクトリと 7 つのサブディレクトリ) を検索します。MFD は検索しません。

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

4.5.2 ハイフン(-)・サブディレクトリ文字

ハイフン文字は、現在のデフォルト・ディレクトリより上の[サブ]ディレクトリの短縮指定方法として使用します。ハイフンはそれぞれが 1 つのレベルを表します。ハイフンの後にはサブディレクトリ名 (区切りはピリオド) を続けてディレクトリ階層の下のパスを指定します。

多くのハイフンを入力して参照が最上位ディレクトリより上を指定した場合は、エラー・メッセージが表示されます。

次の例では、現在のデフォルト・ディレクトリは[JONES.LICENSES]です。以下のコマンドでは[JONES]内の STAFF.DIS の最新バージョンが表示されます。

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

現在のディレクトリが[JONES.LICENSES]の場合、[JONES.TAXES]ディレクトリにおいて、BILLING.DAT の最新バージョンを表示するためには、次のように入力します。

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

以下の例では、表示されるコマンドがデフォルト・ディレクトリからディレクトリ階層で現在のレベルから 2 レベル上のディレクトリに変更されます。

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

ODS-5 ディスクを使用する OpenVMS Alpha バージョン 7.2 またはそれ以降では、ハイフンだけでファイル名とサブディレクトリ名を指定できます。ハイフンと相対指定による(サブ)ディレクトリと区別するため、前者は少なくとも 1 つの RMS エスケープ文字 (^) を指定します。以下の指定では、現在のプロセス・デフォルトより 3 つ上のレベルのディレクトリを参照します。

```
[---]
```

以下の指定では、ディレクトリ (UFD)"-"を参照します。

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

4.6 UIC 形式でのディレクトリの操作

本章では、ディレクトリを名前で指定する方法 (名前形式と呼ぶ) について説明してきました。ただし、ディレクトリは UIC 形式で指定することもできます。UIC 形式は、2 つの 8 進数から構成されます。この 2 つの 8 進数がユーザ・ファイル・ディレクトリ (UFD) を指す UIC (利用者識別コード) を表します。ファイル指定が可能な DCL コマンドのほとんどが、UIC 形式のディレクトリ名を認識できます。リアルタイム Resource Sharing Executive (RSX) オペレーティング・システムを操作する場合を除いて、通常、この形式を使用する必要はありません。

UIC ディレクトリ指定の形式は、次のとおりです。

[グループ,メンバ]

たとえば、[122,1]は、グループ 122 のメンバ 1 を表す UIC ディレクトリ指定です。通常、UIC 形式のディレクトリ名はディレクトリの所有者の UIC に対応しますが、必ず対応するわけではありません。

UIC ディレクトリを表す場合には、次の規則に従います。

- グループの指定には、1 ~ 37776 の範囲の 8 進数を使用する。
- メンバの指定には、0 ~ 177776 の範囲 8 進数を使用する。
- ハイフン(-)または反復記号(...)ワイルドカードは、UIC ディレクトリ指定において使用できない。

4.6.1 UIC ディレクトリでのワイルドカードの使用

アスタリスク(*)ワイルドカードはUIC ディレクトリ指定に使用できます。たとえば、[*,6]は、任意のグループ番号において、メンバ番号 6 を持つすべてのディレクトリを示します。検索されるのは、UIC 形式のディレクトリだけです。[*,*]というディレクトリ指定は、UIC 形式ですべてのディレクトリを検索します。UIC 形式と同様、名前形式のすべてのディレクトリを検索する場合、[*]と指定します。

4.6.2 UIC 形式から名前形式への変換

UIC 形式のディレクトリ名は名前形式のディレクトリ名に変換できます。必要であれば、グループ番号とメンバ番号の左にゼロをいくつか追加して 6 文字の名前を作成すれば、名前形式のディレクトリ名となります。

UIC 形式と名前形式を組み合わせることはできません。UIC 形式の名前を持つディレクトリのサブディレクトリの 1 つを指定する場合には、UIC 形式を名前形式に変換します。

UIC ディレクトリ指定[122,1]と次の名前形式とは同じです。

[122001]

[122,1]SUB.DIR というサブディレクトリを表す場合には、[122001.SUB]という名前形式のディレクトリを使用します。

拡張ファイル指定

OpenVMS Alpha バージョン 7.2 で、次の 2 つの構成要素から成る拡張ファイル指定がインプリメントされました。

- 幅広い範囲の文字を含む長いファイル名をサポートするオプションのボリューム構造 On-Disk Structure Level 5 (ODS-5)
- 深いディレクトリ

これらの構成要素を利用することによって、OpenVMS Alpha システムは (Advanced Server for OpenVMS を使用して)、Windows 環境と同様の名前を持つファイルの格納、管理、サービスの提供、およびアクセスを、これまでよりもはるかに柔軟に行うことができます。

深いディレクトリと拡張ファイル名には、次の利点があります。

- OpenVMS のユーザは、長いファイル名、新しい文字のサポート、および小文字によるファイル名と大文字小文字が混在したファイル名の両方の処理機能を利用することができる。これらの新しい機能により、OpenVMS ファイル・サーバ上のファイルの使用状況が、Windows ユーザにとってより透過的になる。
- OpenVMS システム管理者は、Windows ユーザが指定した名前を使用して、OpenVMS システム上のファイルを表示することができる。
- アプリケーション開発者は、深いディレクトリをサポートしている他の環境からアプリケーションを移植する場合に、OpenVMS 上でも同等の構造を使用できる。

5.1 ODS-5 ボリューム構造

On-Disk Structure (ODS) とは、ディスクに格納された情報に与えられる論理構造を指します。ODS-2 は、OpenVMS オペレーティング・システムの省略時のディスク構造です。ODS-5 は ODS-2 のスーパーセットであり、特にマルチプラットフォーム環境で便利です。ODS-5 ボリューム構造には、次の機能があります。

- 長いファイル名
- ファイル名の中で使用できる文字の種類の拡大
- ファイル名の中での大文字と小文字の区別の保存

5.1.1 長いファイル名

従来 (ODS-2) のファイル指定の形式は 39.39 であり、ファイル名とファイル・タイプを区切るには 1 個のピリオド (.) しか使用できません。

ODS-5 ボリュームでは、ファイル名とファイル・タイプを合わせた長さの上限は、8 ビット文字では 236、16 ビット文字では 118 です。¹たとえば、次のようなファイル名を使用することができます。

```
$ 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.
```

5.1.2 ファイル名の中で使用できる文字の種類の拡大

従来 (ODS-2 準拠) のファイル名では、英数字 (A-Z、a-z、0-9)、ドル記号 (\$)、アンダースコア (_)、およびハイフン (-) を使用することができます。ODS-5 では、より多くの種類の文字セットをファイル名に使用することができます。

ISO LATIN-1 および Unicode (UCS-2) 文字セット

ODS-5 は、8 ビットの ISO Latin-1 文字セットと 16 ビットの Unicode (UCS-2) 文字セットを使用したファイル名をサポートしています。ISO Latin-1 各国語文字セットは、従来の ASCII 文字セットのスーパーセットです。拡張ファイル指定では、8 ビットの ISO Latin-1 各国語文字セットにあるすべての文字をファイル指定の中で使用することができますが、アスタリスク (*) と疑問符 (?) は使用できません。

特殊文字

一部の ISO Latin-1 文字をファイル指定の中で使用して正しく解釈されるためには、エスケープ文字を前に付ける必要があります。RMS および DCL は、拡張ファイル名に含まれているサーカンフレックス (^) をエスケープ文字として解釈します。次のリストは、エスケープ文字を使用する場合の規則を示しています。

- エスケープ文字 (^) の後にアンダースコア (_) またはスペースを続けると、スペースを表す。
- エスケープ文字 (^) の後に次の文字を続けると、続けた文字がファイル指定の中で持つ特別な意味ではなく、ファイル名の一部として使用されることを意味する。

. , ; [] % ^ &

¹ ODS-5 用に変更されていないプログラムやユーティリティでは、完全なファイル指定の長さが合計で 255 バイト以内に制限されたり、省略される可能性があります。

- ユーザがファイル名の中でリテラル文字としてのピリオド (.) を使用する場合は、エスケープ文字 (^) を使用してもしなくてもよい。ピリオドがファイル・タイプとバージョン番号の間の区切り文字として機能する場合を除き、システムはすべてのピリオドに対してエスケープ文字を追加する。ディレクトリ名の中で使用するリテラル文字としてのピリオド (.) の場合は、その前に必ずエスケープ文字を付ける必要がある。
 - エスケープ文字の後に 16 進数が続く場合には、2 番目の 16 進数が必要になる。その後の 2 文字が、任意の 1 バイト文字の 16 進数値として解釈される。たとえば、^20 はスペースを表す。
 - ファイル指定の中でエスケープ文字の後に“U”が続く場合には、その後の 4 つの 16 進数が Unicode として解釈されることを表す。たとえば、^U012F。
- ファイル指定の中で前にエスケープ文字 (^) が付いていないすべての文字は、ISO Latin-1 と想定される。

注意

ファイル名に特殊文字が含まれている場合には、VAX システムからアクセスすることができません。複合アーキテクチャ環境についての詳細は、第 5.7 節を参照してください。

ピリオド (.) の解釈

拡張ファイル名の中でピリオド (.) をリテラル文字として使用するには、ピリオドがファイル名文字であるかまたは区切り文字であるかを、RMS が判断できなければなりません。

拡張ファイル名の中で使用されているピリオド (.) が 1 つだけの場合には、そのピリオドは区切り文字として解釈されます。以前のバージョンの OpenVMS と同様に、1 つのピリオドの後に数字が続いた場合にも、この処理が実行されます。

```
$ CREATE Test.1
```

このコマンドによって、次のファイルが作成されます。

```
Test.1;1
```

バージョン番号の判断

1 つのファイル名の中に複数のピリオド (.) がある場合には、RMS は、最後のピリオドの後のすべての文字を調べます。

最後のピリオドの後の文字	判断
すべて数値	バージョン番号であると判断される。
すべて数値であり、その前にマイナス記号 (-) が付いている	バージョン番号であると判断される。
6 個以上の数字	RMS はこのファイル名を無効なファイル名と判断し、受け付けない。

最後のピリオドの後の文字	判断
数字でない文字	最後のピリオドはファイル・タイプ区切り文字として解釈される。

たとえば、次のコマンドを使用したとします。

```
$ CREATE Test4.3.2.1
```

この結果、次のファイルが作成されます。

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

このとき、2はファイル・タイプ、1はファイル・バージョンです。

バージョン番号がセミコロン (;) によって明示的に区切られている場合には、5文字以下の数値文字でなければならない、前にマイナス記号 (-) を付けることができます。

5.1.3 大文字と小文字の区別の保存

以前のバージョンの OpenVMS では、DCL および RMS は、すべてのファイル指定を大文字に変換していました。

ODS-5 ボリュームでは、すべてのファイル名の大文字と小文字の区別は、ユーザが作成したときの状態のまま保存されます。次に例を示します。

```
$ CREATE KitContents.Txt
$ DIRECTORY
Directory DISK1:[USER1]
KitContents.Txt;1
```

大文字と小文字の区別が異なるだけで、同じ名前のファイル名を複数作成すると、DCL は後でできたファイルを新しいバージョンとして扱い、大文字と小文字の区別を元のファイルと同じ状態に変換します。次に例を示します。

```
$ CREATE CaPri
$ CREATE CAPRI
$ CREATE capri
$ DIRECTORY
Directory DISK1:[USER1]
CaPri.;1 CaPri.;2 CaPri.;3
```


5.1.4 ワイルドカードの使用

ワイルドカードは、単一文字の場合でも複数文字の場合でも、ODS-5 ファイルでは予想どおりに動作します。単一文字のワイルドカードは、ファイル名またはファイル・タイプの中の特定の 1 文字だけを表しますが、ファイル・バージョン文字列の中で使用することはできません。複数文字のワイルドカードは、ファイル名またはファイル・タイプの中の (0 個も含む) 任意の数の文字を表します。複数文字のワイルドカードは、バージョン文字列の代わりに使用することができます。

5.1.4.1 ワイルドカード文字

次の文字は、常に有効なワイルドカード文字です。

- アスタリスク (*): 複数文字のワイルドカード
- パーセント記号 (%): 単一文字のワイルドカード
- 疑問符 (?): 単一文字のワイルドカード

パーセント記号 (%) は、既存のアプリケーションとの互換性を保つため、引き続き単一文字のワイルドカードとして使用されます。パーセント記号 (%) の前にサーカンフлекс (^) を付けるとリテラル文字として使用することができます。また、Windows ファイル名の中では、リテラル文字として使用されます。RMS は、パーセント記号の他に疑問符 (?) も単一文字のワイルドカードとして認識します。疑問符は、OpenVMS 7.2 以降では、パーセント記号とまったく同様に、ワイルドカード文字として機能します。パーセント記号と疑問符は共に、検索パターンの中の 1 文字だけとマッチします。

注意

エスケープ文字 (^ など) またはエスケープ・シーケンス (^EF や ^U0101 など) は、ワイルドカードのマッチングで使用するための単一文字と考えられています。

5.1.4.2 ワイルドカードの構文

DCL では拡張ファイル名の大文字と小文字の区別が保存されますが、ワイルドカードのマッチングでは、大文字と小文字は区別されません。

ワイルドカードを含む検索処理では、引き続き、対象となるファイルの同じ部分にある対応する文字だけとマッチします。表 5-1 は、ワイルドカードを使用した検索の例を示しています。

表 5-1 ワイルドカードおよびパターン・マッチングの例

パターン...	マッチする例...	マッチしない例...
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

5.2 深いディレクトリ構造

ODS-2 および ODS-5 ボリューム構造はどちらも，OpenVMS Alpha 上で深いディレクトリのネスティングをサポートしています。

- 255 レベルまでのディレクトリ
- ODS-2 では，ディレクトリ名の形式は 39.39
- ODS-5 では，各ディレクトリ名は 8 ビット文字で 236，16 ビット文字で 118 まで

たとえば，次のような深いネスティング構造を持つディレクトリを作成することができます。

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

ODS-5 ボリューム上では，次のように長いファイル名を持つディレクトリを作成することができます。

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

完全なファイル指定が 255 バイトより長い場合，変更されていないアプリケーションで表示するときには，完全なファイル指定は RMS によって短縮されます。

5.2.1 ディレクトリの命名構文

ODS-5 ボリュームでは，ディレクトリ名は，ISO Latin-1 を使用した場合のファイル名と同じ規則に準拠します。ピリオドと特殊文字は，ディレクトリ名の中で使用することができますが，リテラル文字として認識されるためには，表 5-2 に示すように，エスケープ文字としてのサーカンフレックス (^) を前に付けなければなりません。

表 5-2 ODS-5 ボリューム上のディレクトリ名

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

5.2.2 ディレクトリ ID およびファイル ID の短縮形

状況によっては、完全なファイル指定に、変更されていないアプリケーションで許可されている 255 バイトより多くの文字が含まれている場合があります。そのようなアプリケーションが必要とするファイル指定の長さが 255 バイトを超えている場合、RMS はディレクトリをディレクトリ ID(DID) に短縮し、ファイル名をファイル ID(FID) に短縮することによって、より短いファイル指定を生成します。

ファイル指定が長すぎる場合、RMS はまずディレクトリをそのディレクトリ ID に変換することにより、より短いディレクトリを生成しようとします。この短い指定は、DID と呼ばれます。

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

UIC 形式のディレクトリ名との混同を避けるため、この形式のディレクトリ名には、3 つの数字と 2 つのコンマがなければならないことに注意してください。DIRECTORY コマンドを使用すると、ファイル指定の短いバージョンのほか、完全なバージョンも表示することができます。

5.3 DCL での拡張ファイル指定解析機能の使用

ファイル名に対する省略時の DCL 解析スタイルは、ODS-2 スタイルのファイル名です。

拡張ファイル名を DCL コマンド行で使用する場合には、拡張ファイル指定を受け付けて表示できるように、解析スタイルを EXTENDED に設定する必要があります。解析スタイルを設定するには、次のコマンドを入力します。

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

このコマンドは、OpenVMS VAX システムにはまったく影響を与えないことに注目してください。

このコマンドを入力すると、DCL は、次のようなファイル名を受け付けるようになります。

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

詳細は、『OpenVMS DCL ディクショナリ: N-Z』の SET PROCESS/PARSE_STYLE コマンドに関する説明を参照してください。

DCL を省略時の解析スタイルに再設定するには、次のコマンドを入力します。

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

このコマンドを入力すると、DCL は、ODS-2 のファイル名形式だけを受け付けるようになります。

5.4 拡張ファイル指定を使用できる状況

一部の DCL コマンドと OpenVMS ユーティリティは、拡張ファイル指定を完全にサポートしています。これらのコマンドおよびユーティリティは、拡張ファイル名のすべての特徴を活用できるように変更されており、拡張ファイル指定をエラーなしに、また、大文字と小文字の区別を変更せずに受け付け、処理することができます。さらに、ディレクトリ ID(DID) またはファイル ID(FID) 形式に短縮せずに、従来課せられていた 255 バイトの制限を超える長いファイル指定を受け付け、また生成することができます。¹

省略時サポート・レベルの DCL コマンドおよび OpenVMS ユーティリティには、拡張ファイル名を活用するための変更がほとんど、またはまったく加えられていません。これらのユーティリティおよびコマンドは、拡張ファイル指定のほとんどの属性(新しい文字や深いディレクトリ構造など)を正しく処理します。ただし、ファイル名を作成したり表示するときに、大文字と小文字の区別に誤りが生じる可能性があります。

完全サポート・レベルのユーティリティとは異なり、省略時サポート・レベルのユーティリティは、長いファイル指定を扱うために RMS が提供する DID および FID の短縮機能に依存しています。このため、これらのユーティリティには、DID および FID の短縮に関連した次の制限事項があります。

- FID の短縮が使用されている環境でのマッチング操作は、必ずしも期待どおりに動作しません。たとえば、ワイルドカードを使用してのマッチング操作は、長いファイル名が数値 FID 短縮形式で表現されている可能性があるため、必ずしもすべてのターゲット・ファイル名が操作対象として認識されないことがあります。この制限事項は、RMS の外部で実行されるマッチング操作に適用されます。
- ワイルドカードや省略時の値は、FID の短縮では使用できません。たとえば、次のコマンドは無効です。

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

¹ DCL コマンド行に長いファイル指定を入力する場合でも、コマンド行の長さは 255 バイトに制限されます。

FID の短縮形は 1 つのファイルの一意の数値表現であるため、これを使って他のファイルを表現したり、マッチングを実行することはできません。

- FID の短縮形を使ってファイルを作成することはできません。

DID および FID の短縮形についての詳細は、『Guide to OpenVMS File Applications』を参照してください。

特定のコマンドまたはユーティリティについての詳細は、OpenVMS ドキュメント・セットの中の該当するマニュアルを参照してください。

拡張ファイル名非サポート

拡張ファイル名をサポートしていない OpenVMS ユーティリティおよびコマンドは、ODS-5 ポリウム上で機能することができるものの、処理できる対象が従来のファイル指定に限られます。これらのユーティリティおよびコマンドは、拡張ファイル指定を処理するときに正しく動作することが保証されていないため、ODS-5 ポリウム上では慎重に使用する必要があります。

ODS-5 非サポート

ODS-5 ポリウム構造をサポートしていない OpenVMS ユーティリティおよびコマンドは、拡張ファイル名を処理することができません。これらのユーティリティおよびコマンドは、従来のファイル指定を処理するときでも正しく動作することが保証されていないため、ODS-5 ポリウム上では慎重に使用する必要があります。

表 5-3 には、拡張ファイル名または ODS-5 構造の処理に制限があるため、Extended File Specifications をサポートしていない OpenVMS ユーティリティおよびコマンドが示されています。

表 5-3 サポートされていない OpenVMS コンポーネント

コンポーネント	注意事項
ODS-5 非サポート	
ディスク・デフラグメンタ	特定のデフラグメンテーション・ツールが ODS-5 ポリウムをサポートするように更新されたという記述がある場合を除き、サポートされない。 ¹
拡張ファイル名非サポート	
コード・コンパイラ	拡張ファイル名をオブジェクト・ファイル名として使用することはできない。ただし、コード・コンパイラが、拡張ファイル名をサポートするアプリケーションを作成することはできる。
INSTALL 既知イメージ	拡張ファイル名を持つイメージを、既知イメージとしてインストールしてはならない。
LINK	拡張ファイル名を持つイメージを出力することはできない。

¹DFO は ODS-5 ポリウムをサポートするように変更されていることに注意してください。

(次ページに続く)

表 5-3 (続き) サポートされていない OpenVMS コンポーネント

コンポーネント	注意事項
拡張ファイル名非サポート	
MONITOR	拡張ファイル名を信頼の置ける形で処理することができない。
ネットワーク・ファイル (NET*.DAT)	拡張ファイル名を使った名前に変更してはならない。
オブジェクト・モジュール (.OBJ)	拡張ファイル名を使った名前に変更してはならない。
ページ・ファイルとスワップ・ファイル	拡張ファイル名を使用してはならない。
SYSGEN	拡張ファイル名を使ってパラメータ・ファイルを作成してはならない。
システム・スタートアップ・ファイル	拡張ファイル名を使った名前に変更してはならない。

5.5 拡張ファイル名の表示

一部の DCL コマンドでは、次のように新しい修飾子を使用して拡張ファイル名の表示を制御することができます。

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

この修飾子を使用すると、更新された DCL コマンドが拡張ファイル名およびそれらに関連するプロンプトを表示する方法を制御することができます。

キーワード CONDENSED を使用すると、多くのユーティリティで 255 バイトと定められている文字列の上限以内に収まるように生成されたファイル指定が表示されます。必要に応じて、このファイル指定には、DID 短縮形または FID 短縮形が含まれている場合があります。キーワード EXPANDED を使用すると、ディスク上に格納されているファイル指定が完全な形で表示され、DID 短縮形や FID 短縮形は含まれません。

この後の項では、DIRECTORY、TYPE、PURGE、および DELETE コマンドに /STYLE 修飾子を使用した例が示されています。

5.5.1 DIRECTORY コマンド

DIRECTORY コマンドを使用すると、ディレクトリの内容を表示するときに、ファイル名を表示する形式を選択することができます。

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

DIRECTORY コマンドを使用すると、省略時の設定では、次の例のように、必要に応じて DID を使用し、DID が不要な場合には完全なディレクトリ指定に切り替わって、ファイル名が表示されます。

```
$ DIRECTORY
Directory TEST$ODS5:[23,1,0]
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
NOPQRSTUVWXYZ.abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZNOPQRSTUVWXYZabcde
fghijklmnopqrst;2
Total of 1 file.
Directory TEST$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.
```

DIRECTORY コマンドで/STYLE 修飾子を使用し、両方のキーワードを指定すると、2 列から成るディレクトリ・リストが表示されます。それぞれの列には、すべてのファイル名が含まれています。CONDENSED の列には、必要に応じて DID および FID が含まれ、EXPANDED の列には、完全なディレクトリ名および完全なファイル名が含まれています。ファイル・エラーがあると、CONDENSED 列に表示されません。次の例は、DIRECTORY コマンドで/STYLE 修飾子を使用し、両方のキーワードを指定した結果を示しています。

```
$ DIRECTORY/STYLE=(CONDENSED,EXPANDED)
Directory TEST$ODS5:[23,1,0] TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ KLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
KLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz KLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
uvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZfghi
fghiijklmnopqrstuvwxyzABCDEFGHIJKLMNO fghiijklmnopqrstuvwxyzABCDEFGHIJKLMNO
PQRSTUVWXYZ.abcdefghijklmnopqrstuvwxyzPQRSTUVWXYZ.abcdefghijklmnopqrstuvwxyz
yABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
ijklmnopqrst;2 jklmnopqrst;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.
```

DIRECTORY コマンドでは、/STYLE 修飾子に一方または両方のキーワードを使用することができます。

5.5.2 TYPE コマンド

TYPE コマンドは、/STYLE 修飾子を受け付けます。この修飾子を使用すると、ファイルやプロンプトを入力するときにシステム・メッセージに表示されるファイル名形式を選択することができます。

```
$ TYPE/STYLE=(keyword)
```

次の例では、TYPE コマンドに修飾子 TYPE=EXPANDED および CONFIRM を使用しています。

```
$ TYPE/CONFIRM/STYLE=EXPANDED abc*.*rst;2
TYPE TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]abcdefghijklmnopqrstuvwxyzABCDEF
GHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNPOQRSTUVWXYZabc
defghijklmnopqrstuvwxyzGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz;2 ? [N]: Y

[System outputs contents of file]
```

5.5.3 DELETE コマンド

DELETE コマンドは/STYLE 修飾子を受け付けます。この修飾子を使用すると、コマンドを実行するときに表示されるファイル名形式を選択することができます。

```
$DELETE/STYLE=(keyword)
```

次の例の反復記号 (...) は、ファイル名の中に多くの文字が含まれていることを示します。これらの例では、CONFIRM 修飾子を使用して、システム・メッセージを生成しています。

省略時の値 (CONDENSED) を使用した DELETE:

```
$ 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
```

完全なファイル指定が必要な場合には、DELETE コマンドで/STYLE 修飾子と EXPANDED キーワードを使用します。

```
$ DELETE/CONFIRM/STYLE=EXPANDED abc*.*.*
DELETE TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]abcAlphabet.stuff;1 ? [N]: Y
DELETE TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]abcdefg. . .QRSTUVWX
Y.abcdefg. . .tuvwxyz;1 ? [N]: Y
```


5.5.4 PURGE コマンド

PURGE コマンドは/STYLE 修飾子を受け付けます。この修飾子を使用すると、コマンドを実行するときに表示されるファイル名形式を選択することができます。

```
$ PURGE/STYLE=(keyword)
```

次の例の反復記号 (...) は、ファイル名の中に多くの文字が含まれていることを示します。これらの例では、CONFIRM 修飾子を使用して、システム・メッセージを生成しています。

省略時の値 (CONDENSED) を使用した PURGE:

```
$ PURGE/CONFIRM
DELETE TEST$ODS5:[23,1,0]abcdefg. . .QRSTUVWXY.abcdefg. . .tuvwxy;1
? [N]: Y
```

完全なファイル指定が必要な場合には、PURGE コマンドで/STYLE 修飾子と EXPANDED キーワードを使用します。

```
$ PURGE/CONFIRM/STYLE=EXPANDED
DELETE TEST$ODS5:[TEST.RANDOMTESTING.RANDOM]abcdefg. . .QRSTUVWXY.ab
cdefg. . .tuvwxy;1 ? [N]: Y
```

5.6 拡張ファイル名の端末表示

端末に拡張ファイル名を表示するには、端末で表示する文字セットとして ISO Latin-1 を指定しなければなりません。このように設定しないと、画面に表示される文字は、PC で表示される文字と一致しくくなります。端末に表示される文字セットを確認または変更するには、端末セッティング・ダイアログ・ボックスを使用します。表示する文字セットを選択するためのオプションは、通常 General タブにあります。

付録 A には、DEC MCS と ISO Latin-1 文字セットの間で異なる文字のリストが示されています。

5.7 複合環境での作業

OpenVMS Alpha バージョン 7.2 またはそれ以降のバージョンを実行しているシステムでは、ODS-5 ボリューム上で拡張ファイル指定機能のすべてを利用することができます。また、バージョン 7.2 より前のファイルおよびディレクトリも、引き続き扱うことができます。たとえば、次のようなことができます。

- ODS-2 ボリューム上に深いディレクトリ構造を作成し、アクセスする。

- 以前のバージョンの OpenVMS で作成された BACKUP セーブセットを読み込む。
- ODS-5 形式の名前を持つファイルを、以前のバージョンの OpenVMS を実行しているシステム上の ODS-2 形式の名前のファイルにコピーする。

複数のバージョンまたは複数のアーキテクチャが混在する OpenVMS Cluster で作業する場合は、いくつかの制限事項があります。以前のバージョンの OpenVMS を実行しているシステムは、ODS-5 ボリュームをマウントできず、拡張ファイル名を正しく扱えず、拡張ファイル名を表示することもできません。バージョン 7.2 より前のバージョンの OpenVMS 上のユーザは、ODS-5 ボリューム上のファイルにアクセスできません。ボリュームが CI または SCSI バスに物理的に接続されている場合でも、MSCP または QIO サーバによって接続されている場合でも同様です。また、これらのユーザは ODS-5 イメージ・セーブセットを作成したり復元することもできません。ただし、ODS-5 セーブセットから ODS-2 準拠のファイル名を復元することはできます。

OpenVMS バージョン 7.2 VAX システムで使用可能な拡張ファイル指定機能は、次のものに限られます。

- ODS-5 ボリュームをマウントする。
- ODS-5 ボリューム上に ODS-2 準拠ファイルを書き込み、管理する。
- ODS-5 ファイル指定にアクセスしたときには、擬似名 (\pISO_LATIN\.??? または \pUNICODE\.??? が表示される

OpenVMS Alpha システムと OpenVMS VAX システムの両方が含まれている環境で作業を行う場合には、ユーザが次のことを認識していることが重要です。

- システム・タイプとオペレーティング・システムのバージョン
- 省略時のディレクトリは ODS-2 と ODS-5 のどちらか
- 新しいファイルを作成するのは ODS-2 と ODS-5 のどちらのボリュームか

OpenVMS バージョン 7.2 を使用すると、VAX システムから ODS-5 ボリュームをマウントすることができます。ただし、OpenVMS VAX システム上のユーザがアクセスできるのは、ODS-2 準拠のファイル名を持つファイルだけです。

OpenVMS Alpha システム上のボリュームを ODS-5 に変換するかどうかを選択することができます。ODS-2 ボリュームと ODS-5 ボリュームの複合環境で作業を行う場合には、ODS-5 ボリューム上でファイルを作成するときの ODS-2 ファイル名の制限事項に注意してください。ファイル名に特殊文字が含まれている ODS-5 ボリューム上のファイルを ODS-2 ボリューム上にコピーするには、ODS-2 準拠の名前を付ける必要があります。

ディスクとテープ・ドライブの使用方法

この章では、OpenVMS システムでのディスク・ドライブとテープ・ドライブの使用に関する一般的な概念を説明します。OpenVMS システムに接続された周辺機器は、ディスク・ドライブとテープ・ドライブも含めて、デバイスと呼ばれます。ユーザがログインすると、ユーザの省略時のデバイスとディレクトリへのアクセスが自動的に許可されます。また、パブリック・デバイスおよびディレクトリにアクセスすることもできます。ほとんどの場合、複数のユーザが共用するデバイスの設定と管理はシステム管理者が行います。

個人的に使用できるドライブがある場合には、そのようなデバイスの割り当て、初期化、およびマウントの方法を知っておく必要があります。本章では、個人専用のディスク・ドライブおよびテープ・ドライブへのアクセスをインプリメントするユーザのために、以下の概念を説明します。

- 物理デバイス名
- デバイス情報の表示
- 論理デバイス名
- 汎用デバイス名
- OpenVMS Cluster デバイス名
- ボリュームとボリューム・セット
- デバイス管理

補足説明については、以下を参照してください。

- 本章で説明したコマンドについての詳細は、『OpenVMS DCL デictionary』またはオンライン・ヘルプを参照。
- デバイスの使用方法是、『OpenVMS システム管理者マニュアル (上巻)』を参照。
- MOUNT コマンドについての詳細は、『OpenVMS システム管理ユーティリティ・リファレンス・マニュアル』を参照。
- OpenVMS Cluster 環境内でのデバイスについての詳細は、『OpenVMS Cluster システム』を参照。

6.1 物理デバイス名

システムは各物理デバイスを物理デバイス名によって一意に識別します。物理デバイス名によって、ディスク・ドライブかターミナルかなど、デバイスの種類がわかります。

ほとんどの物理デバイス名は、次の項目からできています。

- ハードウェアの種類を表すデバイス・コード
- デバイスが接続されているハードウェア・コントローラを識別するコントローラ指示子
- 特定のコントローラ上のデバイスを識別するユニット番号

VTA12, FX09, および DAD44 は、デバイス名の一例です。

デバイス名の具体的な形式については、『OpenVMS システム管理者マニュアル』を参照してください。

6.2 デバイス情報の表示

システム上にあるボリュームについての情報を表示するには、SHOW DEVICES コマンドを使用します。さらに詳しい情報や特定のデバイスについての情報を得るには、次のいずれかの方法で SHOW DEVICES コマンドを入力します。

- 記録密度、ボリューム・ラベル、UIC、マウントされたボリュームの相対ボリューム番号を調べるには、SHOW DEVICES/FULL コマンドを入力する。
- システムで構成されている特定のタイプのすべてのドライブについての情報を表示するには、汎用デバイス・コード (たとえば、SHOW DEVICES DK) を指定する。
- 特定のドライブにマウントされているボリュームの情報を表示するには、物理デバイス名 (たとえば、SHOW DEVICES DKA1) を指定する。

次の例では、SHOW DEVICES コマンドは DAD40: についての情報を表示しています。

```
$ SHOW DEVICES DAD40
```

Device	Device	Error	Volume	Free	Trans	Mnt
Name	Status	Count	Label	Blocks	Count	Cnt
DAD40:	Mounted wrt1ck	0	CHICAGO	540088	1	1

6.3 論理デバイス名

システム管理者は、システム上のデバイスを表す論理デバイス名を設定できます。論理デバイス名を使用すると、複雑なデバイス名を短くて分かりやすい名前に対応させることができます。そして、物理デバイス名の代わりにこの論理デバイス名を使用してデバイスを参照できます。

論理名の使用法についての詳細は、第 11 章を参照してください。

6.4 汎用デバイス名

汎用デバイス名はデバイス・コードからできており、個々のコントローラやユニット番号は省略されています。MOUNT コマンドや ALLOCATE コマンドで汎用デバイス名を使用した場合、システムは、指定された汎用デバイス名の部分を満足する物理名を持つ最初に利用できるコントローラまたはデバイス・ユニットを探します。

他のコマンドで汎用デバイス名を指定する場合、次の省略時の値が使用されます。

- コントローラ指示子を省略した場合、A が使用される。
- ユニット番号を省略した場合、0 が使用される。

6.5 OpenVMS Cluster デバイス名

OpenVMS Cluster デバイス名には、デバイスが接続されているノードの名前と物理デバイス名を指定します。2 つの名前はドル記号(\$)で区切ります。たとえば、ROXXY\$DUA1 は、ノード ROXXY 上のディスク DUA1 を表します。

デュアル・パスの OpenVMS Cluster ディスクには、割り当てクラスのデバイス名を使用するのが普通です。このデバイス名は、すべての OpenVMS Cluster ノードが常に認識できる唯一の名前です。

OpenVMS Cluster の環境におけるデバイス名の形式についての詳細は、『OpenVMS Cluster システム』を参照してください。

デバイスがデュアル・パスの (2 つのノードに接続されている) 場合、次の形式の OpenVMS Cluster デバイス名を指定します。

\$ノード割り当てクラス\$ddcu

要素の意味は次のとおりです。

ノード割り当て クラス	デュアル・パス・デバイスに接続されているノードに割り当てられた値。 たとえば、\$1\$DJA16 は、両方ともクラス値 1 を持つ 2 つのノードに接続されたディスクを識別する。
----------------	---

dd	ハードウェア・デバイス・タイプのデバイス・コードを表す。たとえば、デバイス・コード DK は RZ23 ディスクを表す。
c	デバイスが接続されているハードウェア・コントローラを識別する。コントローラ・デスティネーションは、ユニット番号といっしょにシステムのハードウェア構成内のデバイスの場所を識別する。コントローラは英字 A から Z によって指示される。
u	個々のコントローラ上のデバイスのユニット番号を識別する。ユニット番号は、0 から 65535 までの 10 進数であり、重複しない値を持つ。

6.6 ボリュームとボリューム・セット

OpenVMS オペレーティング・システムは、ディスクとテープをそれらが存在する実際のハードウェア・ドライブと区別して、ボリュームとして認識します。ボリュームは、編成されたデータの集まりです。OpenVMS オペレーティング・システムは、ボリューム・セットも認識します。ボリューム・セットは、2 つ以上の関連するボリュームから成ります。ボリュームを結合してボリューム・セットにすると、複数の新しいボリュームを定義しなくても、1 つのセットにボリュームを追加できるので、ファイルに使用できる領域を拡張することができます。単一のボリュームではなくボリューム・セットを作成する手順については、『OpenVMS システム管理者マニュアル』を参照してください。

6.7 デバイスの管理

個人的に使用できるディスク・ドライブがある場合は、ディスク・ドライブの設定手順を知っておく必要があります。

手順	操作
1	DCL の ALLOCATE コマンドを使用して、ディスク・ドライブをユーザのプロセスに割り当てる。
2	DCL の INITIALIZE コマンドを使用して、ディスク・ボリュームをフォーマットしてから、必要があれば、そのボリュームに識別用のラベルを書き込む。
3	DCL の MOUNT コマンドを使用して、ボリュームまたはボリュームに含まれているファイルやデータをプロセスがアクセスできるようにする。

6.7.1 デバイスの割り当て

デバイスを割り当てると、そのデバイスはそのユーザ・プロセス専用となります。このデバイスは、DCL コマンドの DEALLOCATE によって明示的にデバイスの占有を解除するまで、またはログアウトするまでプロセスに割り当てられたままになります。

DCL コマンドの ALLOCATE を使用してディスクやテープ・ドライブを自分のプロセスに割り当てる (ローカル割り当て)。ALLOCATE コマンドの形式は、次のとおりです。

ALLOCATE デバイス名[:][...] [論理名[:]]

要素の意味は次のとおりです。

デバイス名	ボリュームをロードするドライブを表す。デバイス名には物理名、汎用名、または論理名を指定できる。
論理名	デバイスに関連付ける論理名を指定する。この名前は省略可能である。

6.7.2 ボリュームの初期化

ディスクまたは磁気テープ・ボリュームを初期化すると、そのボリュームはフォーマットされます。ボリュームを使用するたびに初期化する必要はありません。初期化は、ボリュームを最初に使用する前と、ボリュームを完全に消去したいときに行います。ボリュームを初期化するには、DCL コマンドの INITIALIZE を使用します。INITIALIZE コマンドは次の処理を行います。

- ボリュームに新しいファイル構造を作成する。初期化時にディスクに格納されていたデータは、初期化処理中に削除される。
- ボリューム上に識別のためのラベルを付ける。
- 所有者 UIC とボリュームの保護を定義する。

注意

INITIALIZE コマンドを使用すると、別のユーザのボリュームを初期化することも可能です。このため、このコマンドを使用するときには、あらかじめ目的のデバイスを割り当ててから個人専用のボリュームを初期化するようにしてください。

別のユーザにボリュームを初期化してもらう場合 (たとえば、十分な特権が自分にはない場合) には、そのボリュームのボリューム・ラベル、所有者 UIC、保護コードをそのユーザに知らせなければなりません。

INITIALIZE コマンドの形式は、次のとおりです。

INITIALIZE デバイス名[:] ボリューム・ラベル

フィールドの意味は次のとおりです。

デバイス名	ボリュームを物理的にマウントするデバイスの名前を表す。
ボリューム・ラベル	ボリュームを識別するためのものである。ディスク・ボリュームには最大 12 文字の英数字、磁気テープ・ボリュームには最大 6 文字の英数字のラベルを指定できる。

ディスク・ボリュームの初期化

省略時の設定では、INITIALIZE コマンドは新しいボリューム上に Files-11 構造を作成します。OpenVMS オペレーティング・システム用に、または OpenVMS オペレーティング・システムによって初期化されたディスク・ボリュームの省略時の形式を、Files-11 ディスク構造レベル 2 と呼びます。INITIALIZE コマンドは、Files-11 ディスク構造レベル 1 形式でディスク・ボリュームを初期化することもできます。

ブランクのディスク・ボリューム (すなわち、書き込みが一切行われていないボリューム) や現在の UIC または UIC [0,0] によって所有されたディスク・ボリューム上で論理保護を変更するには、特別な特権は必要ありません。これ以外の場合、ディスク・ボリュームを初期化するには、ユーザ特権 VOLPRO が必要です。

次の例は、DKA300 上でボリュームを初期化し、そのボリュームに ACCOUNTS というラベルを付けています。

```
$ INITIALIZE DKA300: ACCOUNTS
```

6.7.3 ボリュームのマウント

割り当てたディスク・ボリューム上のファイルを使用するためには、そのボリュームをマウントする必要があります。DCL の MOUNT コマンドは、ボリュームとボリュームに格納されているファイルがプロセスにアクセスできるようにします。

MOUNT コマンドを入力すると、システムは、次の条件が満足されているかどうかを検証します。

- デバイスが別のユーザによって割り当てられていない。
- デバイスを割り当てることができるようにデバイス保護が設定されている。
- ボリュームが物理的に指定されたデバイスにロードされている。
- ボリューム上のラベルが指定されたラベルと一致している。

単独のボリュームをマウントすることもできますし、ボリューム・セットをマウントすることもできます。ボリューム・セットの作成とマウントの手順については、『OpenVMS システム管理者マニュアル』を参照してください。

MOUNT コマンドの形式は、次のとおりです。

MOUNT デバイス名[:...] [ボリューム・ラベル[:...]] [論理名[:]]

要素の意味は次のとおりです。

デバイス名	ボリュームをマウントするデバイスの物理デバイス名または論理名を表す。
-------	------------------------------------

ボリューム・ラベル	ボリュームを初期化するときを使用したラベルを表す。MOUNT 修飾子の/FOREIGN, /NOLABEL, /OVERRIDE=IDENTIFICATION のいずれかを使用する場合には、ボリューム・ラベルを指定する必要はない。
論理名	デバイスに関連付ける名前を表す。論理名を省略すると、MOUNT コマンドは、ディスク・ドライブとテープ・ドライブに省略時の論理名 DISK\$ボリューム・ラベルと TAPE\$ボリューム・ラベルをそれぞれ割り当てる。

6.7.4 オペレータの手助けが必要な場合

オペレータは、システム・ボリュームとプライベート・ボリュームの両方の物理マウント (およびディスマウント) を行うことができます。使用するドライブにすでにボリュームが存在する場合は、オペレータの手助けは必要ありません。

MOUNT メッセージは、TAPE および DISK メッセージを受信する権限のあるすべてのオペレータに送信されます。たとえば、ディスク・デバイスをマウントするのにオペレータの手助けが必要な場合には、ディスク・オペレータにメッセージを送信します。MOUNT 要求を受信してそれに応答できるオペレータがいない (オペレータにその権限がない) 場合には、その状況を知らせるメッセージが表示されます。オペレータの手助けが必要なければ、/NOASSIST 修飾子を指定することもできます。

MOUNT コマンドを出すと、オペレータにマウント要求が通知され、ターミナルにメッセージが表示されます。

```
$ MOUNT DKA300: DISK VOL1
%MOUNT-I-OPRQST, PLEASE MOUNT DEVICE _MARS$DKA300:
```

デバイスが正しくマウントされると、次のメッセージが出されます。

```
%MOUNT-I-MOUNTED, DISK mounted on _DKA300:
```

次の例は、ディスク・ボリュームを割り当てて、初期化し、マウントする方法を示しています。

```
$ 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]
```

ボリューム上にファイルを格納するには、上記の例の CREATE/DIRECTORY コマンドの部分に示されているように、ディレクトリを作成しなければなりません。

フォーリン・ディスク・ボリュームのマウント

フォーリン・ディスク・ボリューム (すなわち、Files-11 以外のファイル構造を持つディスク・ボリューム) をマウントするには、次のように、/FOREIGN 修飾子を指定します。

```
$ MOUNT/FOREIGN DISK  
%MOUNT-I-MOUNTED, BACKUP_FILE      mounted on DISK$DMA2:
```

MOUNT/FOREIGN コマンドは、フォーリン・ディスクのボリュームの内容をシステムでできるようにしますが、ファイル構造に関しては特別な処理は行いません。前の例では、MOUNT はボリューム・ラベルを表示して、フォーリン・デバイスとしてマウントされているにもかかわらず、ディスクが Files-11 構造であることを示しています。ディスクが認識されたファイル構造を持たない場合には、MOUNT はラベルを表示しません。

/FOREIGN 修飾子を指定して Files-11 構造のディスクをマウントするには、所有者 UIC が自分の UIC と一致する場合を除いて、ユーザ特権 VOLPRO が必要です。

6.8 プライベート・デバイス上のファイルのアクセス

プライベート・デバイス上にあるファイルをアクセスするには、デバイス名を指定するか、SET DEFAULT コマンドを使用してそのデバイスを省略時のデバイスおよびディレクトリ名に設定しなければなりません。

デバイスを参照するには、物理名、論理名、または汎用名が使用できます。さらにシステムが OpenVMS Cluster の一部である場合、OpenVMS Cluster のすべてのメンバが特定のデバイスをアクセスできます。テープ・ボリューム・セット上のファイルにアクセスするには、割り当てられたデバイスを指定します。

個人用のボリュームからファイルを印刷することができますが、印刷するファイルを含んでいるボリュームは、ファイルの印刷が完了するまでマウントしたままにしておく必要があります。

出力ファイル指定を使用できるコマンドもあります。この場合、出力ファイル指定をプリンタやターミナルなどのレコード単位取り扱いデバイスの名前に置き換えることができます。たとえば、次のように指定します。

```
$ COPY DFILE.DAT TTB4:
```

この COPY コマンドは、ファイル DFILE.DAT を TTB4 というターミナルに送ります。ターミナルはファイルを受け取り、1 レコードずつ表示します。ファイル指定としてデバイス名を使用する場合、デバイス名の後にコロン (:) を付けます。

6.8.1 ボリュームのディスマウント

ディスクやテープ・ボリュームでのファイル操作が終了したら、DISMOUNT コマンドを使用して、ボリュームをディスマウントできます。DISMOUNT コマンドは、ボリュームをディスマウントする前に、ディスマウント操作を妨げる条件がないかどうかをチェックします。たとえば、ボリュームにインストール済みのスワップ・ファイルやページ・ファイル、インストール済みのイメージ、オープンされたユーザ・ファ

イルが含まれる場合には、ボリュームをディスマウントできないことを知らせるエラー・メッセージを出します。

省略時の設定では、DISMOUNT コマンドは、ドライブからボリュームを自動的にアンロードします。ボリュームをもう一度マウントしたり初期化したりする予定があるときには、次に示すように、ボリュームをディスマウントした後で、/NOUNLOAD 修飾子を使用すれば、ボリューム処理に要する無駄な時間を節約できます。

```
$ DISMOUNT/NOUNLOAD MTA1:
```

この例では、磁気テープ・ボリュームは論理的にディスマウントされ、テープが巻き戻されますが、物理的にはテープは MTA1 ドライブにロードされたままです。

ボリュームを物理的にアンロードする場合には、DISMOUNT コマンドによってボリュームを明示的にディスマウントしなければなりません。ドライブがアンロードされるのを待ってから、ボリュームを取り出すようにしてください。DCL コマンド SHOW DEVICES を入力すれば、ディスマウント操作が完了したかどうかを確認できます。

ボリュームをマウントしたジョブからログアウトすれば、ボリュームは自動的にディスマウントされてアンロードされます。ただし、システムに障害が生じた場合には、ボリュームは自動的にディスマウントされません。ディスマウントしているデバイスが、ALLOCATE コマンドによって割り当てられたデバイスである場合には、DISMOUNT コマンドを使用してディスマウントしても、割り当ては解除されません。MOUNT コマンドによってデバイスを暗黙に割り当てた場合には、DISMOUNT コマンドによって割り当てが解除されます。

Mail を使用して他のユーザと通信する

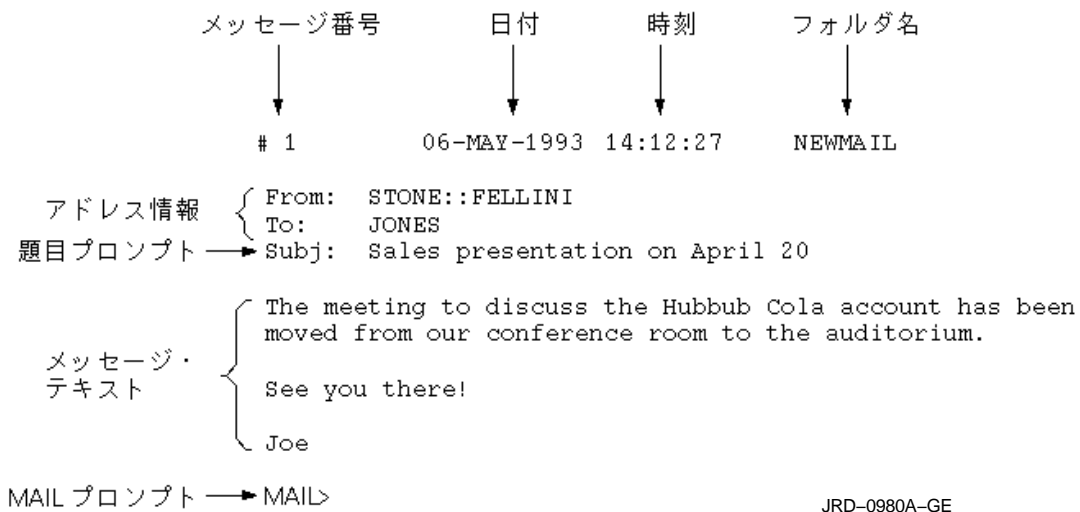
OpenVMS Mail ユーティリティ (MAIL) は、メッセージをシステム上の他のユーザや、Compaq TCP/IP for OpenVMS または DECnet ネットワークで接続されている他のコンピュータのユーザに送ることができるようにします。本章では、以下のことについて説明します。

- Mail の起動と終了
- メッセージの開封
- メッセージの送信
- ネットワークを介したメールの送信
- 複数のユーザへのメッセージの送信
- メールでのファイルの操作
- メッセージを送信するための他の方法
- メッセージの整理
- メッセージの削除
- メール・メッセージの印刷
- メール・ファイルの保護
- Mail でのテキスト・エディタの使用法
- Mail 環境のカスタマイズ
- Mail コマンドの要約
- MIME ユーティリティの使用

詳細は、以下を参照してください。

- Mail コマンドと修飾子の詳細情報については、HELP MAIL コマンドを入力するか、MAIL>プロンプトに対して HELP コマンドを入力。
- ユーザ・アカウントを通じて Mail の使用を制御する方法の説明については、『OpenVMS システム管理者マニュアル』を参照。
- TCP/IP メール・コマンドと修飾子については、DCL プロンプトに HELP TCPIP_SERVICES を入力します。
- TCP/IP Services によるメールの送受信の詳細については、『Digital TCP/IP Services for OpenVMS User's Guide』を参照してください。

次の図は、メールのメッセージと構成を示しています。



7.1 Mail の起動と終了

これ以降の節では、Mail の起動方法および終了方法を説明します。

7.1.1 Mail の起動

Mail ユーティリティを起動するには、MAIL と入力します。

```
$ MAIL
MAIL>
```

いったん Mail ユーティリティに入り、MAIL>プロンプトで適切なコマンドを入力し、Enter キーを押すと、次の操作を実行できます。

- メール・メッセージの開封
- メール・メッセージの送信
- メール・メッセージの返信
- メール・メッセージの転送
- フォルダとファイルによるメール・メッセージの整理
- メール・メッセージの削除
- メール・メッセージの印刷

7.1.2 Mail の終了

Mail を終了するには、MAIL>プロンプトに対して EXIT コマンドを入力します。

```
MAIL> EXIT
$
```

Ctrl/Z を押すか、QUIT コマンドを使用しても Mail を終了できます。

7.2 メッセージの開封

Mail は、受け取ったメッセージをメール・ファイル(省略時のファイル・タイプ:MAI)に格納します。省略時の設定では、このファイルには、古いメッセージと新しいメッセージを格納する 2 つのフォルダがあります。新しいメッセージは自動的に NEWMAIL と呼ばれるフォルダに格納され、古いメッセージは MAIL と呼ばれるフォルダに格納されます。

通常、新しいメッセージを読み終えると、そのメッセージは自動的に NEWMAIL フォルダから MAIL フォルダに移されます。また、FILE コマンドや MOVE コマンドを使用して格納するフォルダを変更したり、DELETE コマンドを使用してメッセージを削除することもできます。新しいメール・メッセージをすべて読み終えた後、別のフォルダを選択するか、Mail を終了すると、NEWMAIL フォルダは削除されます。

7.2.1 新しいメールを読む

自分のアカウントにログイン中にメール・メッセージを受信すると、新しいメッセージを受信したという通知が画面に表示されます。たとえば、FELLINI というユーザから送信されたメッセージを受信すると、次のような通知が表示されます。

```
New mail on node DOODAH from STONE::FELLINI      (10:02:23)
```

新しいメッセージを読む場合には、Mail を起動して、MAIL>プロンプトに対して Enter キーを押します。

```
$ MAIL
You have 1 new message.
MAIL>
```

新しいメッセージが複数ある場合には、MAIL>プロンプトに対して Enter キーを押すと次のメッセージを読むことができます。新しいメッセージをすべて読み終えると、"%MAIL-E-NOMOREMSG, no more messages"というメッセージが表示され、コマンドの入力待ち状態になります。

Mail コマンドの実行中にメール・メッセージを受け取った場合には、READ/NEW コマンドを入力すれば、新しいメッセージを読むことができます。

7.2.2 古いメッセージの開封

省略時の MAIL フォルダにある古いメール・メッセージを読む場合には、次の手順に従ってください。

手順	操作
1	MAIL>プロンプトに対して SELECT コマンドを入力する。 MAIL> SELECT MAIL MAIL フォルダに移動する。
2	省略時の MAIL フォルダにある最初のメッセージを読むには、MAIL>プロンプトに対して Enter を押すか、READ コマンドを入力する。 省略時のメール・ファイルにある最初のメッセージ(1)が表示される。
3	次のメッセージを表示するには、Enter を押す。 メッセージが長すぎて 1 つの画面に収まらない場合には、Enter を押せば、メッセージの次の部分が表示される。 メッセージの残りの部分をスキップして次のメッセージを表示する場合には、NEXT コマンドを入力する。

省略時の MAIL フォルダにある特定のメッセージを読む場合には、次の手順に従ってください。

手順	操作
1	MAIL>プロンプトに対して DIRECTORY コマンドを入力する。 >From や Subj に特定の文字列を含むメッセージのみのリストを表示するには、DIRECTORY コマンドと一緒に /FROM または /SUBJECT 修飾子を使用する。
2	MAIL>プロンプトに対して、読みたいメッセージの番号を入力する。 選択したメッセージが表示される。

次の例では、DIRECTORY コマンドを使用して古いメッセージを表示し、その後、開封するメッセージとして、2 というラベルの付いたメッセージを選択しています。

```
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
```


7.2.3 メッセージの検索

複数のメッセージがあるときは、SEARCH コマンドを使用すれば特定のメッセージを見つけることができます。SEARCH コマンドは、1 つ以上のメッセージの中から特定の文字列を探し出すコマンドです。

新しい文字列を検索する場合には、SEARCH コマンドのパラメータとして別の文字列を指定します。新しい文字列を指定するたびに、SEARCH コマンドはメッセージ番号 1 から検索を開始します。同じ文字列を検索し続ける場合には、パラメータを指定せずに SEARCH コマンドを使用します。異なるフォルダから同じ文字列を検索する場合には、SELECT または SET FOLDER フォルダ名のコマンドを入力してから、パラメータを指定せずに SEARCH コマンドを使用します。

次の例では、現在のフォルダの中から、文字列 *appointment* が含まれている最初のメッセージが選択され、表示されています。

```
MAIL> SEARCH "appointment"
```

7.3 メッセージの送信

メール・メッセージを同じシステムの他のユーザに送信する場合には、次の手順に従ってください。

手順	操作
1	MAIL>プロンプトに対して SEND を入力する。 メッセージを受け取るユーザの名前を求めるプロンプトが表示される。
2	メッセージを受け取るユーザの名前を入力して Enter を押す。 メッセージの題目を求めるプロンプトが出される。
3	メッセージの題目を入力して Enter を押す。この情報は省略することもできる。 メッセージのテキストを求めるプロンプトが表示される。
4	メッセージのテキストを入力して Enter を押す。この情報は省略することもできる。
5	Ctrl/Z を押してメッセージを送信する。メッセージを送信しない場合には、Ctrl/C を押す。Ctrl/C は、Mail を終了せずに送信操作を取り消す。

次の例では、メッセージは THOMPSON という名前のユーザに送信されます。

```
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
```

7.4 ネットワークを介したメールの送信

これ以降の節では、ネットワークを介してメールを送信する方法を説明します。

7.4.1 ネットワーク・プロトコルの指定

メッセージを受信すると、以下のように Mail は指定アドレスを解釈します。

- アドレスのノード名にピリオド(.)が含まれる場合、アドレスはインターネット・アドレスに解釈されます。MAIL\$INTERNET_TRANSPORT 論理名で代替プロトコルを定義して別のインターネット・プロトコルを使用するようあらかじめシステムを設定しておかない限り、Mail は省略時の設定で SMTP プロトコルを使用します。
- アドレスのノード名にピリオドがなければ、アドレスは DECnet アドレスとして解釈されます。

ただし、特定のプロトコルを選択するよう Mail 環境をカスタマイズできます。このオプションは、インターネット・プロトコルと DECnet プロトコルのどちらでもメール・アドレスを有効に解釈できる場合に便利です。

プロトコルを指定するには、以下のように MAIL\$INTERNET_MODE 論理名を指定します。

- HYBRID (省略時の設定)

アドレスのノード名にピリオド(.)を使用すると、Mail ではインターネット・プロトコルを使用します。ピリオドがなければ、Mail では DECnet プロトコルを使用します。

- DECNET

Mail では、つねにアドレスのノード名を DECnet ノード指定として解釈します。

- SMTP

Mail では、つねにアドレスのノード名をインターネット・アドレス指定として解釈します。省略時の伝送プロトコルは、MAIL\$INTERNET_TRANSPORT 論理名で代替インターネット伝送プロトコルを使用しない限り SMTP です。

以上のどの方法でも Mail 環境を変更する場合、Compaq では LOGIN.COM ファイルに MAIL\$INTERNET_MODE 論理名と MAIL\$INTERNET_TRANSPORT 論理名を指定することを推奨します (論理名の使用方法と定義方法については第 11 章を参照)。

たとえば、システムが省略時の設定 (HYBRID) を使用するセットアップの場合、Mail アドレス smith@pluto は DECnet アドレスとして解釈されます。これは、アドレスにピリオドがないためです。しかし、Mail で DECnet ではなく SMTP を使用したい場合は、LOGIN.COM ファイルに以下の行を追加します。

```
$ DEFINE MAIL$INTERNET_MODE SMTP
```

smith@pluto を指定すると、Mail はこのアドレスを Internet アドレスとして解釈し、SMTP プロトコルを使用します (例; SMTP%"smith@pluto.xyz.dec.com")。

7.4.2 ノード名の指定

ユーザのコンピュータ・システムがネットワークに接続されている場合には、ネットワーク上の他のユーザにメールを送信できます。別のノード上のユーザにメールを送信する場合には、To: プロンプトに対してユーザのノード名とユーザ名を入力します。ユーザ名に特殊文字やスペースが含まれている場合は、ユーザ名を二重引用符(" ")で囲む必要があります。次の形式を使用します。

ノード名:: ユーザ名

リモート・ノードへネットワーク接続できない場合には、メッセージが表示されません。少し待ってからメッセージの送信を再び試みてください。

ノード名の指定についての詳細は、第 3.1.6 項を参照してください。

たとえば、CHEETA ノード上の HIGGINS というユーザにメッセージを送信する場合には、次のように入力します。

```
MAIL> SEND  
To: CHEETA::HIGGINS
```

7.4.3 インターネット・メール・アドレスの使用

ネットワーク経由でユーザにメールを送信するために、完全な形式のインターネット・メール・アドレスも使用することができます。特に組織の外へメールを送信する場合、このアドレスは一般的なものです。

username@company.com

To: プロンプトに対して、メールを送信したいユーザのインターネット・アドレスを完全な形式で入力します。アドレスは、ほとんどの場合、大文字小文字が区別されません。

```
MAIL> SEND  
To: J_SMITH@COMPANYNAME.COM, Kate.Muir@school.edu
```

7.4.4 論理ノード名の使用法

論理名を使用して、ユーザ名とノードを表すこともできます。Mail は、ノード名や論理名の中のアクセス制御情報を無視するので注意してください。

次の例では、CHEETA::HIGGINS のかわりに HENRY が使用されています。まず、論理名 (HENRY) が定義され、次にユーザ名とノードのかわりに、その論理名使用されます。

```
$ DEFINE HENRY CHEETA::HIGGINS
$ MAIL
MAIL> SEND
To: HENRY
```

7.5 複数のユーザへのメッセージの送信

これ以降の節では、複数のユーザにメールを送信する方法を説明します。

7.5.1 個人名の使用方法

同時に複数のユーザにメールを送信する場合には、To: プロンプトに対して個々のユーザ名を入力するか、配布リストを使用します。ユーザ名を使用して複数のユーザに同じメッセージを送信するには、To: プロンプトに対して各ユーザ名を入力します。このとき、ユーザ名はコンマかスペースで区切ります。

たとえば、メッセージを Thompson, Jones, Barney の 3 人に送信する場合には、次のように入力します。

```
MAIL> SEND
To: THOMPSON,JONES,BARNEY
Subj: Meeting on January 9
```

7.5.2 配布リストの作成

配布リストとは、ユーザ名とそのノード名のリストが収められたファイルのことです。テキスト・エディタを使用して、配布リストを作成しなければなりません。配布リストは、Mail ユーティリティ内では作成できません。

オープン・ファイル・クォータ (アカウントに割り当てられた制限値) によって、メールを 1 度に送信できるノード数と、配布リストをネストできる深さが決定されます。クォータを超えると、エラー・メッセージが表示されます。このような場合は、システム管理者に相談してクォータを大きくするか、一度にメールを送信するノード数を少なくしてバッチ形式で送信します。

省略時の設定では、検索される配布リスト・ファイルのファイル・タイプは.DIS です。配布リスト・ファイルがこれと異なるファイル・タイプを持つ場合には、To: プロンプトに対してファイル名とファイル・タイプを指定します。配布リスト・ファイルが Mail を起動したときのディレクトリとは別のディレクトリに登録されている場合には、To: プロンプトに対して配布リストの完全なディレクトリ名を入力します。

配布リストを作成するには、次の手順で行います。

手順	操作
1	テキスト・エディタを使用して、ファイル・タイプ.DIS の配布リスト・ファイルを作成する。
2	配布リスト・ファイルの 1 行ごとにユーザ名を 1 つずつ入力する。
3	ファイルに他の配布リストの名前を含める (リストを「ネスト」する) 場合には、アットマーク(@)の後に配布リストの名前を指定する。
4	ファイルにコメントをいれるには、コメントの前に感嘆符(!)を入力する。

次に、配布リスト・ファイルの例を示します。

```
! ALLBUDGET.DIS
!  
! Budget Committee Members  
@BUDGET      ! listed in BUDGET.DIS.  
! Staff  
  Thompson  
  BRUTUS::JONES  
  PORTIA::BARNEY
```

BUDGET.DIS ファイルが、作成中の新しい配布リスト・ファイル (ALLBUDGET.DIS) と同じディレクトリにない場合には、BUDGET.DIS のファイル指定を新しい配布リストにする必要があります。ALLBUDGET.DIS を作成する場所によっては、BUDGET.DIS を登録するデバイスとディレクトリを指定しなければならないこともあります。ファイル指定についての詳細は、第 3 章を参照してください。

7.5.3 配布リストによるメッセージの送信

配布リストを使用して複数のユーザにメールを送信する場合には、次の手順で行います。

手順	操作
1	Mail を起動する。
2	MAIL>プロンプトに対して SEND と入力してから Enter を押す。
3	To: プロンプトに対してアットマーク(@)と配布リストのファイル名を入力してから、Enter を押す。
4	Subj: プロンプトに対してメッセージの題目を入力してから Enter を押す。
5	テキスト・プロンプトに対してメッセージのテキストを入力する。

次の例では、メッセージは配布リスト ALLBUDGET.DIS に送信されます。

Mail を使用して他のユーザと通信する

7.5 複数のユーザへのメッセージの送信

```
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 レベルから配布リストにファイルを送信することもできます。ファイル・タイプ (.DIS) を省略する場合には、アットマークとファイル名の前後に二重引用符を付けて、ファイルを配布リストとして識別します。題目をいれるには、MAIL コマンドで /SUBJECT 修飾子を使用します。

次の例は、MEETING.TXT ファイルを THOMAS というユーザと FRIENDS.DIS 配布リスト内のユーザに送信します。

```
$ MAIL/SUBJECT="update" MEETING THOMAS, "@FRIENDS.DIS"
```

次の例は、NOTICE.TXT ファイルを WRITERS.DIS 配布リスト内のユーザに送信します。ここでは、/SUBJECT 修飾子が入っていないので、メッセージの送信時の題目は省略されます。

```
$ MAIL NOTICE "@WRITERS"
```

7.6 Mail 内でのファイルの操作

ファイルの送信は、Mail 内部からでも、または DCL レベルからでもできます。Mail 内部からファイルを送信する場合には、次の手順で行います。

手順	操作
1	MAIL>プロンプトに対して、SEND と、送信したいファイルの名前を入力する。
2	To: プロンプトに対して、ファイルの受信者の名前を入力する。
3	Subj: プロンプトに対して、ファイルの題目を入力する。
4	ファイルを送信する場合には、Enter を押す。ここで、送信操作を取り消す場合には、Ctrl/C または Ctrl/Y を押す。Ctrl/C を押すと MAIL プロンプトに戻り、Ctrl/Y を押すと DCL レベルに戻る。

次の例では、ファイル MEMO.TXT がユーザ EDGELL に送信されます。

```
MAIL> SEND MEMO.TXT
To: EDGELL
Subj: Another memo
```

メールによってファイルを送信するときには、次の制限事項に注意してください。

- COPY コマンドを使用してファイルをコピーするときには、オペレーティング・システムによるデータ整合性チェックが行われます。このチェックはメールによ

ってファイルを送信するときには行われないので、フォーリンファイル (実行可能ファイルなど) を送信するときにファイルが破損することがあります。

- 大きなファイルを送信するときには注意が必要です。一部のシステムのユーザは、大きなファイル (POSTSCRIPTファイルなど) を受信できない場合があります。

7.6.1 DDIF ファイルの送信

ファイルが DIGITAL Document Interchange Format (DDIF) 仕様に従って作成された複合文書である場合には、OpenVMS AXP Version 1.0 または VAX VMS Version 5.2-2、またはそれ以降のシステムの場合にだけ、Mail は OpenVMS RMS ファイル・タグと DDIF セマンティックを保存します。DDIF ファイルを含むメール・メッセージを、OpenVMS AXP Version 1.0 または VAX VMS Version 5.2-2 より前の OpenVMS システムに送信した場合や、OpenVMS 以外のオペレーティング・システムに送信した場合には、Mail はエラー・メッセージを戻します。

7.6.2 DCL からのファイルの送信

DCL レベルからファイルを送信する場合にも Mail は起動されますが、会話型セッションは開始されず、MAIL>プロンプトも表示されません。ファイルが送信されると、自動的に DCL レベルに戻ります。MAIL コマンドと適切な修飾子を入力した後、ファイルを送信するときは Enter を押し、送信操作を取り消すときは Ctrl/C を押します。

以下の点にも注意してください。

- ファイル指定では、ワイルドカード文字を使用できません。ファイル・タイプを省略すると、省略時のファイル・タイプは.TXT になります。
- ファイル指定として SYSS\$INPUT を指定した場合には、DCL レベルから直接、メッセージ文の入力を求めるプロンプトを表示させることができます。SYSS\$INPUT の使用についての詳細は、第 11 章を参照してください。
- DCL レベルからファイルを送信する場合、省略可能な/SUBJECT 修飾子への引数にスペースや英数字以外の文字が含まれているときには、引数を二重引用符で囲まなくてはなりません。

次の例では、DCL レベルで MEMO.TXT ファイルを CHEETA ノードの EDGELL というユーザに送信しています。

```
$ MAIL/SUBJECT="Another memo" MEMO.TXT CHEETA::EDGELL
```

次の例では、ファイル指定として SYSS\$INPUT を指定した場合に、DCL レベルから直接、メッセージ文の入力を求めるプロンプトが表示されています。

Mail を使用して他のユーザと通信する

7.6 Mail 内でのファイルの操作

```
$ 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
$
```

7.6.3 メッセージからのファイルの作成

メッセージからテキスト・ファイルを作成する場合には、メッセージの読み込み中(選択中)に MAIL>プロンプトに対して EXTRACT コマンドとファイル名を入力します。Mail を終了すると、ファイルは現在のディレクトリに格納されます。別のディレクトリを指定すれば、そのディレクトリに格納されます。ファイルが DDIF ファイルの場合には、OpenVMS RMS ファイルのタグと DDIF のセマンティクスが保持されます (VAX/VMSバージョン 5.2-2 またはそれ以降)。

メールのヘッダは、From:、To:、Subj: の各行から構成されていますが、ヘッダ情報を含まないファイルを作成するには、EXTRACT コマンドに/NOHEADER 修飾子を指定します。メッセージにヘッダが複数ある場合には (たとえば、転送されたメッセージなど)、一番上のヘッダだけが削除されます。

メッセージを既存のファイルの終端にコピーするには、EXTRACT コマンドに/APPEND 修飾子を指定します。/ALL 修飾子を使用すると、現在のフォルダにあるすべてのファイルが既存のファイルにコピーされます。

次の例では、DEC_MEETINGS.TXT という名前のファイルがメール・メッセージから作成されています。

```
#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
```

次の例に、メッセージ番号 3 のテキストを JANUARY_MEETINGS.TXT というファイルに格納する方法を示します。


```
MAIL> READ 3
.
.
.
MAIL> EXTRACT/NOHEADER JANUARY_MEETINGS.TXT
%MAIL-I-CREATED, DISK1:[JONES]JANUARY_MEETINGS.TXT;1 created
MAIL>
```

7.6.4 メッセージへのファイルの追加

メール・メッセージの最後に小さなファイルを自動的に追加するときは、SET SIGNATURE_FILE コマンドを使用します。このコマンドで指定したファイルは、ANSWER、FORWARD、MAIL、REPLY、SEND の各コマンドを使用して送信するすべてのメール・メッセージに自動的に追加されます (省略時の場合)。署名ファイルの例として、ユーザの会社名、住所、電話番号、Internet アドレスを名刺形式で編集したテキスト・ファイルなどが考えられます。

ファイルを特定のメッセージのみに追加したり、省略時の署名ファイル設定を指定変更したりする場合は、ANSWER、FORWARD、MAIL、REPLY、SEND の各コマンドに/SIGNATURE_FILE[=ファイル名]修飾子を付けて使用します。

省略時の署名ファイルを指定しているかどうか確認するときは、SHOW SIGNATURE_FILE コマンドを使用します。また、SHOW ALL コマンドを使用する場合も、署名ファイル情報が表示されます。

DCL コマンド MAIL に/SIGNATURE_FILE[=ファイル名]修飾子を付けて使用すると、省略時の署名ファイルを DCL レベルで設定することも可能です。

署名ファイルを含むメール・メッセージを作成するときには、通常のメッセージの場合よりも大きな一時ディスク領域が必要になります。これは、オペレーションの際に一時的ファイルが作成されるためです。メッセージが送られたら、これらの一時的ファイルは削除されます。

署名ファイルの名前を指定するとき、次の事項に注意してください。

- ファイル・タイプを指定していない場合、省略時の値は.SIG になります。
- ディレクトリを指定していない場合、Mail ユーティリティは、ユーザのメール・ディレクトリで署名ファイルを検索します。

次の例では、FORWARD、MAIL、REPLY、SEND の各コマンドを使用してメール・メッセージを送信する場合、すべてのメッセージに自動的に追加される省略時ファイルとして、ファイルBUSINESS_CARD.SIGが指定されます。

```
MAIL> SET SIGNATURE_FILE BUSINESS_CARD.SIG
```

次の例では、省略時の署名ファイルの代わりに、特定の応答に自動的に追加されるファイルとして、ファイルGREETINGS.SIGが指定されます。

```
MAIL> REPLY/SIGNATURE_FILE=GREETINGS.SIG
```

7.7 その他のメッセージの送信方法

これ以降の節では、Mail ユーティリティを使用してメッセージを送信する他の方法を説明します。

7.7.1 メッセージの返信

受信したメッセージに返信する場合には、次の手順で行います。

手順	操作
1	MAIL>プロンプトに対して REPLY を入力してから Enter を押す。
2	メッセージを入力した後、Ctrl/Z を押すとメッセージが送信される。または、Ctrl/C を押すと、送信が取り消される。

次の例では、返信が STONE::THOMPSON に送信されます。返信コマンドを入力した後、Mail から To: プロンプトと Subj: プロンプトが自動的に表示されます。

```
To: STONE::THOMPSON
Subj: RE: Budget Meeting
Enter your message below. Press CTRL/Z when complete. CTRL/C to quit:
```

7.7.1.1 入れ子になった引用符を含むアドレスへの返信

ほとんどの場合は、Mail の REPLY コマンドを使用して、入れ子になった引用符を含むアドレスにメッセージを返信できます。ただし、使用しているシステムで返信できない場合は、システム管理者に相談してください。

7.7.2 メッセージの転送

メール・メッセージを他のユーザに転送する場合には、メッセージの読み込み中 (選択中) に MAIL>プロンプトに対して FORWARD コマンドを入力します。受信者名と題目行を求めるプロンプトが表示されます。必要な情報を入力した後、Enter を押してメッセージを送信します。

DDIF ファイルから構成されるメッセージを転送すると、DDIF でのセマンティクスと DDIF タグを含む DDIF ファイル全体が受信者に送信されます。

次の例では、メッセージがユーザ STONE::JONES に転送されます。

```
MAIL> FORWARD
To: STONE::JONES
Subj: FYI - Status of proposed budget meeting
```

7.7.2.1 SET FORWARD コマンド

送信されたすべてのメッセージを、別の OpenVMS クラスタ上の、またはまったく異なるシステム上の別のアカウントに転送するために、SET FORWARD コマンドを使用することができます。基本的に、このコマンドは電子的な転送アドレスを作成します。定期的にチェックしたくないアカウントについてのみ、転送アドレスを設定してください。たとえば、OLD クラスタ上のメール・アカウントから、STAR クラスタ上のメール・アカウントへすべてのメールを転送したいとします。OLD にログインしてから、Mail ユーティリティを起動して、次のコマンドを入力してください。

```
MAIL> SET FORWARD STAR::SMITH
```

OLD::SMITH に送信されたすべてのメッセージは、STAR ノード上のメール・アカウントに自動的に転送されます。転送アドレスとして、インターネット・メール・アドレスを設定することもできます。

```
MAIL> SET FORWARD SMITH@Company.com
```

この場合、OLD::SMITH に送信されたすべてのメールは、SMITH@Company.com に送信されます。

常に、古いアカウントに対してテストメッセージを送信して、そのアカウントが転送を正しく行っていることを確認してください。転送が無限に行われて、決して届かない転送ループを作るのを避けるために、自分自身宛てに、または別の転送アカウント宛てに転送を行うアカウントを設定しないでください。OLD::SMITH から OLD::SMITH に転送してはいけません。OLD::SMITH から STAR::SMITH へ転送して、さらに STAR::SMITH から OLD::SMITH へ転送してもいけません。

どのアカウントが転送を行っているかをチェックするには、次のコマンドを入力してください。

```
MAIL> SHOW FORWARD  
Your mail is being forwarded to STAR::SMITH.
```

転送アドレスを削除するには、次のコマンドを入力してください。

```
MAIL> SET NOFORWARD  
MAIL> SHOW FORWARD  
You have not set a forwarding address.
```

転送アドレスが削除されたことを確認して、そのアカウントにテスト・メッセージを送信してください。

注意

以前のバージョンの OpenVMS オペレーティング・システムでは、SET FORWARD コマンドで引用符を指定する場合には、引用符を 2 つ指定しなければなりません。これは、SET FORWARD コマンドが自動的に引用符を 1 つ削除してしまうためでした。OpenVMS V7.0 以降では、SET

FORWARD コマンドが引用符を削除しないようになったので、引用符を 2 つ指定する必要がなくなりました。

7.8 メッセージの整理

これ以降の節では、メッセージを整理する方法を説明します。

7.8.1 フォルダの作成

メール・メッセージを整理する場合には、自分独自のメール・ファイルとフォルダを作成します。メール・ファイルにはフォルダが入り、フォルダにはメール・メッセージが入ります。それぞれのファイルとフォルダには任意の数のメッセージを入れることができます。

通常、メッセージを編成する場合には、メール・ファイルを作成するのではなく、フォルダを作成します。省略時のメール・フォルダ (NEWMAIL, MAIL, WASTEBASKET) と同様に、通常、作成したフォルダはメール・ファイル MAIL.MAI に格納されます。現在のフォルダの名前は、READ または DIRECTORY コマンドを入力するたびに、画面の右上角に表示されます。操作が可能なメッセージは、現在のフォルダにあるメッセージだけです。

メール・ファイルがきわめて大きい (500 ブロックを超える) 場合には、別のメール・ファイルを作成し、フォルダを大きくしてください。Mail ユーティリティの性能が向上します。

7.8.2 メール・サブディレクトリの作成

受信したメール・メッセージは、省略時の設定では、最上位ディレクトリにある MAIL\$xxxxxxxxxx.MAI というファイルに書き込まれます。ここで、x は、任意のファイル指定です。省略時のメール・ファイルである MAIL.MAI は、初めてメール・メッセージを受信したときに、最上位ディレクトリに作成されます。

.MAI ファイルが最上位ディレクトリに表示されないようにするには、Mail コマンドの SET MAIL_DIRECTORY を使用します。このコマンドは、メール・サブディレクトリを作成して、.MAI ファイルをすべてそのサブディレクトリに移動するコマンドです。.MAI ファイルをサブディレクトリから最上位ディレクトリに戻すには、SET NOMAIL_DIRECTORY コマンドを使用します。

すべての.MAI ファイルが登録されているサブディレクトリ名を表示するには、MAIL>プロンプトに対して SHOW MAIL_DIRECTORY コマンドを入力します。

次の例では、ユーザ (FRED) が .MAIL ディレクトリを作成しています。

```
MAIL> SET MAIL_DIRECTORY [.MAIL]
MAIL> SHOW MAIL_DIRECTORY
Your mail file directory is SY$LOGIN:[FRED.MAIL]
```

7.8.3 フォルダへのメッセージの移動

現在選択されているメッセージを別のフォルダに格納する場合には、FILE コマンドまたは MOVE コマンドのいずれかを使用します。フォルダが存在しない場合には、フォルダを作成するかどうかを尋ねるメッセージが表示されます。メッセージが指定されたフォルダに格納されると、自動的に現在のフォルダから削除されます。

7.8.4 フォルダ間でのメッセージのコピー

Mail の COPY コマンドは、現在選択されているメッセージを指定したフォルダにコピーします。フォルダが存在しない場合には、作成するかどうかを尋ねるメッセージが表示されます。

次に示すコマンドは、MEETING という文字列が入っているすべてのメッセージを、現在のフォルダから SCHEDULE というフォルダにコピーします。コマンドの実行後、メッセージごとにコピーが 2 部作成され、1 部は現在のフォルダに、もう 1 部は SCHEDULE フォルダに格納されます。

```
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
```

次のコマンドは、*meeting* という文字列が入っている次のメッセージを選択して表示します。

```
MAIL> SEARCH

MAIL> COPY SCHEDULE
MAIL> SEARCH
%MAIL-E-NOTFOUND, no messages containing 'MEETING' found
```

7.8.5 フォルダの選択

現在選択されているメール・ファイルにあるフォルダのリストを表示する場合には、DIRECTORY /FOLDER コマンドを入力します。新しいフォルダを選択する場合には、次のいずれかのコマンドを使用します。

- SELECT フォルダ名

指定されたフォルダを現在のフォルダとして選択します。これ以降に入力する READ, DIRECTORY などの Mail コマンドは, 指定されたフォルダを使用します。コマンドごとにフォルダ名を指定する必要はありません。

- SET FOLDER フォルダ名

SELECT コマンドと同じように動作します。

- DIRECTORY フォルダ名

指定されたフォルダを現在のフォルダとして選択し, そのフォルダの中のメッセージをリストします。

- READ フォルダ名

指定されたフォルダを現在のフォルダとして選択し, 指定されたメッセージ (省略時の設定では, フォルダの中の最初のメッセージ) を表示します。

次の例では, MEMOS フォルダが選択されます。

```
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
```

7.8.6 フォルダの削除

メール・フォルダを削除するには, フォルダの中のすべてのメッセージを削除するか, 別のフォルダに移動します。フォルダ内のすべてのメッセージを削除すると, 別のフォルダを選択したときに, 空のフォルダは自動的に削除されます。

次の例では, MUSIC フォルダの中のメッセージを削除する場合には, 次のコマンドを入力しています。

```
MAIL> SELECT MUSIC
%MAIL-I-SELECTED, 2 messages selected
MAIL> DELETE/ALL
```

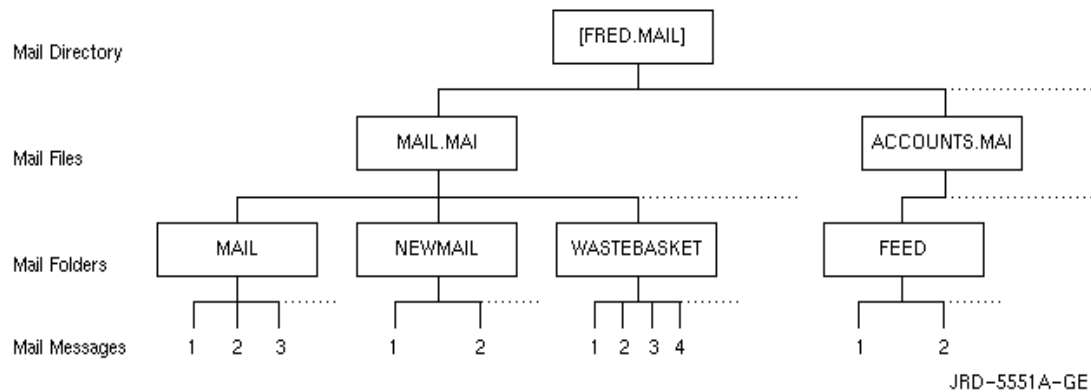
7.8.7 メール・ファイルの作成とアクセス

メール・メッセージを整理するファイルを作成することもできます。メール・ファイルを作成する場合には, フォルダと同様, COPY, MOVE, FILE のいずれかのコマンドを使用します。フォルダの名前を求めるプロンプトの後に, ファイル名を求めるプロンプトも表示されます。File: プロンプトに対して新しいファイル名を入力すると, そのファイルが新規に作成されます。

省略時のメール・ファイル以外のメール・ファイルを操作する場合には、Mail コマンドの SET FILE を使用して別のファイルを指定します。SHOW FILE コマンドを使用すると、現在選択されているメール・ファイルの名前が表示されます。メール・ファイルを変更すると、現在選択されているメール・ファイルの WASTEBASKET フォルダが空にされてから削除され (AUTO_PURGE が設定されている場合)、メール・ファイルはクローズされます。

図 7-1 にメールの一般的な構成例を示します。

図 7-1 メール・ディレクトリ構成



たとえば、現在選択されているメッセージを ACCOUNTS ファイルの中の FEED というフォルダに移動する場合には、次のように入力します。MOVE コマンドは、メール・ファイル ACCOUNTS.MAI を作成して、現在選択されているメッセージを FEED フォルダに移動し、現在選択されているフォルダとファイルからメッセージを削除します。

```
MAIL> MOVE
_Folder: FEED
_File: ACCOUNTS
```

次の例では、ACCOUNTS ファイル内にある FEED フォルダが選択されています。

```
MAIL> SET FILE ACCOUNTS
MAIL> SET FOLDER FEED
MAIL> SHOW FILE
Your current mail file is SYS$LOGIN:[FRED.MAI]ACCOUNTS.MAI;1.
```

7.8.8 メール・メッセージ・カウントの訂正

画面に表示される新しい(未読の)メール・メッセージの数が実際の新しいメッセージ数と一致しない場合には、新しいメールがなくなったときに、READ/NEW コマンドを入力します。READ/NEW コマンドを入力して次のいずれかのシステム・メッセージが表示された場合には、新しいメールがないことを示しています。

```
"%MAIL-W-NONEWMAIL, no new messages"  
"%MAIL-E-NOMOREMSG, no more messages"
```

7.9 メッセージの削除

現在選択されているフォルダからメール・メッセージを削除する場合には、メッセージの読み込み中 (選択中) に DELETE コマンドを入力するか、DELETE コマンドの後に削除したいメッセージの番号 (または番号の範囲) を入力します。削除するメッセージの範囲は、ハイフン(-)またはコロン(:)を使用して指定します。

たとえば、メッセージ 4, 5, 6, 11, 12, 14, 15, 16, 17 を削除するには、MAIL>プロンプトに対して、次のように入力してから Return を押します。

```
MAIL> DELETE 4-6,11,12,14:17
```

7.9.1 削除したメッセージの回復

メッセージを削除すると、メッセージは WASTEBASKET フォルダに移されます。削除されたメッセージは、Mail を終了するか、別のメール・ファイルを指定するかして現在のメール・ファイルを終了するまで WASTEBASKET フォルダに残されます。SET AUTO_PURGE コマンドを実行した場合は、現在のメール・ファイルを終了すると、WASTEBASKET は空になり、WASTEBASKET フォルダ自体が削除されます。つまり、Mail ユーティリティを会話形式で使用している間は、メッセージを WASTEBASKET フォルダから移動すれば、削除したメッセージを回復することができます。PURGE コマンドを入力して、WASTEBASKET フォルダを空にすることもできます。

次の例では、番号 12 によって識別されるメール・メッセージが削除され、その後、WASTEBASKET フォルダから回復されます。

```
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
```

7.10 メール・メッセージの印刷

メール・メッセージを印刷する場合には、MAIL>プロンプトに対して PRINT コマンドを入力します。省略時の設定では、メッセージは SYS\$PRINT キューに送信されます。メール・ファイルは、Ctrl/Z を押すか、EXIT と入力するか、または PRINT/PRINT コマンドを入力するまで印刷キューに送信されません。

別のキューを指定する場合には、PRINT コマンドの/QUEUE 修飾子を使用します。SET QUEUE キュー名コマンドを使用すれば、別のキューを選択することもできます。このキューは、Mail を終了しても、別の SET QUEUE コマンドを入力するまでは省略時の印刷キューのままです。

次の例では、メール・メッセージは AK34\$PRINT 印刷キューに発信されています。

```
MAIL> PRINT/QUEUE=AK34$PRINT
```

次の例では、省略時の印刷キューが SYS\$PRINT から AK34\$PRINT に変更されています。

```
MAIL> SET QUEUE AK34$PRINT
```

7.11 メール・ファイルの保護

これ以降の節では、メール・ファイルの保護について説明します。

7.11.1 省略時の保護の設定

メール・ファイル (たとえば、MAIL.MAI) は、他のユーザが読んだり、間違って削除してしまうことがないように保護されています。.MAI ファイルの保護コードは (S:RW,O:RW,G:;,W:) です。システム (Mail 自体も含む) と所有者は、ファイルを読み込んだりファイルに書き込んだりできますが、グループとワールドはすべてのアクセスが拒否されます。

Mail ユーティリティには、メールへの不正アクセスを防止するための省略時のファイル保護も設定されています。ただし、Mail は完全に不正アクセスから保護されるわけではなく、十分な特権をもっていれば、保護を変更してメール・ファイルにアクセスすることも可能です。

7.11.2 セキュリティの測度

メール・ファイルはそれぞれユーザのディレクトリに格納されていますので、第 10 章に説明するようなファイル保護手法を重要なファイルに適用することもできます。加えて、次の点にも注意してください。

- メール要求に応える際にも、ユーザの適切な判断が要求されます。ノードがローカルな OpenVMS Cluster の外部にある場合には、そのノードが故意にまたは偶然に間違えて識別される可能性もあります。
- メール・メッセージの内容の決定や送信相手の選択にあたっては、慎重さが要求されます。
- パスワードやアカウントの詳細な使用法は、決して漏らさないようにしてください。メール・メッセージの情報は、一旦送信してしまうと、後で取り消すことはできません。

7.12 Mail でのテキスト・エディタの使用方法

これ以降の節では、Mail でテキスト・エディタを使用する方法を説明します。

7.12.1 EVE の使用方法

テキスト・エディタを使用してメッセージを作成してからメッセージを送信することもできます。この場合、SEND コマンドに/EDIT 修飾子を指定します。To: プロンプトと Subj: プロンプトに応答すると、テキスト・エディタが起動されます。別のエディタを選択しない限り、DECTPU に基づく EVE エディタが起動されます。

[End of file]マーカーは、テキストを入力すると下に移動していきます。EVE エディタについての詳細は、第 8 章を参照してください。メッセージを送信する場合には、Do キーを押してから EXIT コマンドを入力します。送信を取り消す場合には、Do キーを押してから QUIT コマンドを入力します。

次の例では、メール・メッセージの作成に EVE が使用されています。

```
MAIL> SEND/EDIT
```

```
[End of file]
```

```
Buffer:  MAIN                |  Write  | Insert  | Forward
```

注意

.DDIF メール・ファイルは編集しないでください。一度編集してしまうと.DDIF ファイルとして使用できなくなるためです。.DDIF メール・ファイルを編集してしまうと、以後そのファイルのテキストにしかアクセスできなくなります。

7.12.2 /EDIT 修飾子キーワードの使用方法

Mail を起動するときに /EDIT 修飾子を指定すると、Mail を終了するまで、エディタを使用した送信、返信、転送操作を行うことができます。/EDIT と一緒にキーワードを使用すれば、Mail の省略時の値を設定することもできます。

メッセージを返信するときにだけエディタを起動する場合には、MAIL/EDIT コマンドに REPLY キーワードを指定します。エディタを起動して、返信先のメッセージを表示する場合には、=EXTRACT オプションとともに REPLY キーワードを指定します。/EDIT にキーワードを指定しない場合、省略時の値は /EDIT=(SEND,REPLY) になります。

メッセージを送信または返信する場合には、Do キーを押して EXIT コマンドを入力してエディタを終了します。SEND または REPLY 操作を取り消す場合には、Do キーを押して QUIT コマンドを入力してエディタを終了します。

例

次の例では、送信または転送される各メール・メッセージに対して、エディタが起動されます。

```
$ MAIL/EDIT=(SEND,FORWARD)
```

次の例では、返信される各メッセージに対して、エディタが起動されます。

```
$ MAIL/EDIT=(REPLY)
```

次の例では、エディタが起動され、メッセージに返信するたびに、返信の対象となっているメッセージがテキストとして含まれます。

```
$ MAIL/EDIT=(REPLY=EXTRACT)
```

7.12.3 エディタの選択

省略時の設定では、Mail コマンドの SEND/EDIT を指定すると、DECTPU に基づく EVE エディタが起動されます。Mail コマンドの SET EDITOR を入力すれば、EVE の代わりのエディタを起動するように指定できます。たとえば、EDT エディタを選択する場合には、SET EDITOR EDT と入力します。SET EDITOR コマンドを入力するまでは（たとえ一度ログアウトしても）、EDT エディタが省略時の Mail エディタになります。

選択された Mail エディタの名前を表示するには、SHOW EDITOR を入力します。

7.12.4 コマンド・ファイルを使用したメールの編集

Mail を入力する前に、論理名 MAIL\$EDIT をコマンド・ファイルとして定義することができます。ここでエディタを起動する Mail コマンドを実行すると、コマンド・ファイルが呼び出されて編集操作が実行されます。コマンド・ファイルでは、スペル・チェッカなど、コマンド・ファイルで実行できるユーティリティや機能を指定することができます。MAILEDIT.COM コマンド・プロシージャの注釈付きの例については、付録 B を参照してください。コマンド・ファイルについての詳細は、第 13 章および第 14 章を参照してください。

7.12.5 選択したエディタの一時的な無効化

選択したエディタを一時的に無効にする場合は、文字列 "CALLABLE_" の後に使用するエディタ名を追加したものを MAIL\$EDIT として定義します。たとえば、EVE の代わりに EDT を使用する場合には、次のコマンドを入力します。

```
$ DEFINE MAIL$EDIT CALLABLE_EDT
```

MAIL\$EDIT が定義された状態で起動されたセッションにおいて SET EDITOR コマンドを実行すると、永久エディタと現在のエディタの設定値が両方とも無効になります。MAIL\$EDIT によって定義されたコマンド・ファイルをもう一度使用するには、Mail を起動し直さなくてはなりません。

7.13 メール・キーパッドの使用法

Mail では、キーボード上にある数値キーパッドを使用してコマンドを実行できます。ほとんどのキーパッド・キーには、2 つのコマンドが割り当てられています。

図 7-2 に示す各キーの上側のコマンドを入力する場合には、該当キーだけを押しします。下側のコマンドを入力する場合には、PF1 キーを押してから該当キーを押しします。

図 7-2 Mail ユーティリティ・キーパッド

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

たとえば、Mail コマンドの SEND を実行する場合には、KP7 を押します。これに対して、Mail コマンドの SEND/EDIT を実行する場合には、最初に PF1 キーを押してから、KP7 を押します。

7.13.1 キーパッド・キーの再定義

キーパッド・キーの定義を変更して、Mail コマンドを実行させることもできます。キーを再定義する場合、それまでのキー定義は新しい定義で置き換えられることに注意してください。

Mail でのキーパッド・キーの定義は、DCL コマンドでのキーパッド・キーの定義と似ています。

次の例では、KP2 キーは Mail コマンド PRINT/PARAM=PAGE_ORIENT=LANDSCAPE として定義されます。このように定義しておけば、KP2 を押すだけで PRINT/PARAM=PAGE_ORIENT=LANDSCAPE コマンドを実行することができます。

```
MAIL> DEFINE/KEY KP2 "PRINT/PARAM=PAGE_ORIENT=LANDSCAPE"
```

7.13.2 追加キー定義のアサイン

/STATE 修飾子を使用すれば、ターミナルで利用できるキー定義数を増やすことができます。状態 (STATE) を設定することによって、同じキーにも複数の機能を定義できます。状態名には、任意の英数字文字列を指定できます。状態を指定することによって、キーを 1 回目に押したときはコマンドを入力し、2 回目に押したときは修飾子を入力するようにできます。

次の例では、PF1 (2 回押す) は DIRECTORY/FOLDER として定義されます。

```
MAIL> DEFINE/KEY PF1 "DIRECTORY"/SET_STATE=FOLDER /NOTERMINATE
MAIL> DEFINE/KEY PF1 "/FOLDER" /IF_STATE=FOLDER /TERMINATE
```

PF1 を 2 回押して、DIRECTORY/FOLDER コマンドを入力します。/TERMINATE 修飾子がコマンド行を終了するので、Enter キーを押す必要はありません。

7.13.3 永久キーの定義

Mail セッション時に定義したキーパッド・キーは、Mail を終了すると無効になります。Mail セッションを終了してもキーパッド・キーの定義が保持されるようにする場合には、キー定義を格納するファイル (たとえば、MAIL\$KEYDEF.INI) を最上位ディレクトリに作成します。たとえば、次のサンプル MAIL\$KEYDEF.INI には、6 つのキー定義が格納されています。

```
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
```

Mail を起動するたびにこれらのコマンドを実行するには、ログイン・コマンド・ファイル (LOGIN.COM) に次のコマンド行を入力します。

```
$ DEFINE MAIL$INIT SYS$LOGIN:MAIL$KEYDEF.INI
```

7.14 Mail コマンドの要約

この節では、すべての Mail ユーティリティ・コマンドの要約を示します。これらのコマンドで使用される修飾子についての詳細は、オンライン・ヘルプを参照してください。

MIME ユーティリティを使用しての、MIME でエンコードされたメッセージの読み込み、編集については、第 7.15 節を参照してください。

7.14.1 メッセージを読む

メッセージを読むには、次のコマンドを使用します。

- BACK

最後のコマンドが READ の場合、現在または最後に読んだメッセージの前のメッセージを表示する。最後のコマンドが DIRECTORY の場合、ディレクトリ・リストの前の画面を表示する。

- CURRENT

現在読んでいるメッセージの冒頭を表示する。

- DIRECTORY [フォルダ名]

現在のメール・ファイルにあるメッセージのリストを表示する。メッセージ番号、送信者の名前、日付、題目も表示する。

- ERASE

ターミナル画面を消去する。

- EXTRACT

現在のメッセージのコピーを、指定されたファイルに出力する。メール・メッセージをメール・ファイルの中のフォルダにコピーする場合、COPY、FILE、MOVE コマンドのいずれか 1 つを使用する。

- FIRST

現在のフォルダの最初のメッセージを表示する。

- LAST

現在のフォルダの最後のメッセージを表示する。

- NEXT

次のメッセージにスキップし表示する。

- READ [フォルダ名] [メッセージ番号]

メッセージを表示する。Enter キーだけの入力、パラメータなしの READ コマンドと同じ。

- SEARCH検索文字列

現在選択されているフォルダから、指定されたテキスト文字列を含むメッセージを検索する。

- SHOW NEW_MAIL_COUNT

未読のメール・メッセージ数を表示する。

7.14.2 メッセージの交換

メッセージを交換するには、次のコマンドを使用します。

- ANSWER [ファイル指定]
REPLY [ファイル指定]

現在または最後に読んだメッセージの送信者に、メッセージを送信する。

- FORWARD

現在または最後に読んだメッセージのコピーを、1 人以上のユーザに送信する。

- MAIL [ファイル指定]
SEND [ファイル指定]

メッセージを 1 人以上のユーザに送信する。

7.14.3 メッセージの削除

メッセージを削除するには、次のコマンドを使用します。

- DELETE [メッセージ番号]

現在読んでいるメッセージ、ある範囲のメッセージ、または最後に読んだメッセージを削除して、WASTEBASKET フォルダに移動する。

- PURGE

WASTEBASKET フォルダにあるすべてのメッセージを削除する。Mail を終了するか、SET FILE コマンドを入力 (新しいメール・メッセージを選択する) した場合、暗黙にパーズが実行されて WASTEBASKET フォルダが空にされる。ただし、あらかじめ SET NOAUTO_PURGE コマンドを実行している場合、WASTEBASKET フォルダの内容は保持される。

- SET [NO]AUTO_PURGE

EXIT または SET FILE コマンドの入力時に WASTEBASKET フォルダを空にするかどうかを決定する。SET NOAUTO_PURGE コマンドを使用する場合、定期的に PURGE コマンドを入力して、WASTEBASKET フォルダの中のメッセージを削除しなければならない。

- SHOW AUTO_PURGE

EXIT または SET FILE コマンドの入力時に WASTEBASKET フォルダを空にするかどうかを表示する。

7.14.4 メッセージの印刷

メッセージを印刷するには、次のコマンドを使用します。

- PRINT

現在読んでいるメッセージのコピーを印刷キューに追加する。PRINT コマンドによって作成されたファイルは、Mail を終了すると印刷キューに渡される。つまり、1つの印刷ジョブに複数のメッセージが連結される (/NOW または /PRINT 修飾子を使用する場合を除く)。

- SET [NO]FORM形式名
SHOW FORM

Mail から PRINT コマンドを実行する場合の省略時の印刷形式を設定する。SET NOFORM コマンドは、省略時の印刷形式をクリアする。SHOW FORM コマンドは、省略時の印刷形式を表示する。

- SET [NO]QUEUEキュー名
SHOW QUEUE

Mail から PRINT コマンドを入力する場合の省略時の印刷キューを設定する。SET NOQUEUE は、定義された印刷キューをクリアし、キューを省略時の印刷キュー SYSSPRINT に設定する。SHOW QUEUE コマンドは、省略時の印刷キューを表示する。

7.14.5 メッセージの整理

メッセージを整理するには、次のコマンドを使用します。

- COPYフォルダ名[ファイル名]

メッセージを現在のフォルダから別のフォルダにコピーする。指定されたフォルダが存在しない場合、フォルダを作成する。

- FILEフォルダ名[ファイル名]
MOVEフォルダ名[ファイル名]

メッセージを現在のフォルダから指定されたフォルダに移動する。

- SELECT [フォルダ名]
SET FOLDER [フォルダ名]

複数のメッセージをグループとして設定する。これらのメッセージは、フォルダ間でのコピー/移動、表示、削除、検索、抽出ができる。加えて、SELECT と SET FOLDER コマンドは、フォルダの変更にも使用できる。

- SET FILEファイル名

現在のメール・ファイルとして別のファイルを設定 (またはオープン) する。省略時のメール・ファイルは MAIL.MAI。COPY, FILE, MOVE のいずれか 1 つの

コマンドを使用して別のメール・ファイルを作成すると、SET FILE コマンドでメール・ファイルをオープンできる。

- SHOW FILE

現在オープンされているメール・ファイルの名前を表示する。

- SHOW FOLDER [フォルダ名]

現在のフォルダ名を表示する。

- SET WASTEBASKET_NAME フォルダ名

WASTEBASKET フォルダの名前を変更する。このフォルダには、削除するメッセージが格納される。PURGE コマンドを入力すると、WASTEBASKET フォルダの中のすべてのメッセージが削除される。AUTO_PURGE が設定されている場合、EXIT コマンドを入力すると、WASTEBASKET フォルダの中のメッセージが削除される。AUTO_PURGE が設定されている場合、QUIT コマンドを入力して WASTEBASKET フォルダの中のメッセージの削除を行わないこともできる。

- SHOW WASTEBASKET_NAME

WASTEBASKET フォルダの名前を表示する。

- SHOW DELETED

現在のメール・ファイルにおいて、削除されたメッセージ・スペースの量を表示する。

7.14.6 メッセージのマーク付け

メッセージにマークを付けるには、次のコマンドを使用します。

- MARK [メッセージ番号]

現在のメッセージまたは指定されたメッセージにマークを設定する。ディレクトリ・リストにおいて、マークが設定されたメッセージの左端はアスタリスク(*)が付けられる。マークが設定されたメッセージを選択したり編成したりする場合、SELECT コマンドに/MARKED 修飾子を指定する。

- UNMARK [メッセージ番号]

現在のメッセージまたは指定されたメッセージのマークの設定を解除する。ディレクトリ・リストの左端のアスタリスク(*)は削除される。

7.14.7 メール環境のカスタマイズ

メール環境をカスタマイズするには、次のコマンドを使用します。

- DEFINE/KEY キー名文字列

Mail コマンドを実行するキーを定義する。コマンド名を入力しなくても、定義したキーを押すだけでコマンドを実行できる。

- SHOW KEY [キー名]
DEFINE/KEY コマンドによって作成されたキー定義を表示する。
- EDIT [ファイル名]
選択したエディタを起動して、メッセージを編集する。エディタの終了後、メッセージは送信される。
- HELP [トピック]
Mail についての情報を表示する。個々のコマンドやトピックについての情報を得るには、HELP の後にコマンド名またはトピック名を入力する。
- SET [NO]CC_PROMPT
メッセージの送信時にカーボン・コピー・プロンプト (CC:) が現れるようにするかを決める省略時の値を設定する。
- SET COPY_SELF コマンド[, コマンド]
SEND, REPLY, FORWARD のいずれか 1 つを入力した場合、送信中のメッセージのコピーを送信者に戻すかどうか、省略時の値を設定する。
- SHOW COPY_SELF
どのコマンド (SEND, REPLY, FORWARD) によってメッセージのコピーが自動的に送信されているのかを表示する。
- SET [NO]SIGNATURE_FILE
ANSWER, FORWARD, MAIL, REPLY, SEND の各コマンドを使用するとき、メール・メッセージの最後に署名テキスト・ファイルが自動的に追加されるよう、Mail ユーティリティを設定する。
- SHOW SIGNATURE_FILE
省略時の署名ファイルを指定したかどうかを示し、指定している場合はそのファイルの名前も表示する。SHOW ALL コマンドを使用する場合も、署名ファイル情報が表示される。
- SET EDITOR エディタ名
SEND/EDIT などによってメッセージを編集するときに使用されるテキスト・エディタを選択する。システム上で使用できるエディタを指定する。このコマンドは、コマンド・プロシージャ MAIL\$EDIT が設定した定義を無効にする。
- SHOW EDITOR
選択したテキスト・エディタの名前を表示する。
- SET [NO]FORWARD アドレス
メールの転送先アドレスを設定する。
- SHOW FORWARD
現在転送中のアドレス名を表示する。
- SET [NO]MAIL_DIRECTORY[, サブディレクトリ名]

SYSSLOGIN ディレクトリ中のすべてのメール・ファイル(ファイル・タイプ.MAI) を、指定されたサブディレクトリに移動するかどうか設定する。

- SHOW MAIL_DIRECTORY

すべての. MAI ファイルが格納されているデバイスとディレクトリの名前を表示する。

- SET [NO]PERSONAL_NAME"テキスト文字列 "

メール・メッセージの From: フィールドに表示するテキスト文字列を指定する。このフィールドには、ユーザの本名やニックネームなどを入力する。ただし、文字列は英字で始まり、スペースを 2 つ以上続けてはならないことに注意。

- SHOW PERSONAL_NAME

SET PERSONAL_NAME コマンドによって設定されたテキスト文字列を表示する。

- SHOW ALL

現在の Mail 設定値についての詳しい情報を表示する。

7.14.8 終了または制御の移行

Mail を終了したり、制御を移行するには、次のコマンドを使用します。

- ATTACH [プロセス名]

現在のプロセスを中断せず、ターミナルの制御を別のプロセスに切り替える。たとえば、ファイルの編集中、SPAWN コマンドを使用すれば、サブプロセス(Mail) に移動して新しいメール・メッセージを読み込むことができる。この後、ATTACH を入力すれば、編集セッションに戻ることができる。

- EXIT

Mail を終了する。EXIT コマンドを入力すると、SET NOAUTO_PURGE コマンドを入力した場合を除いて、WASTEBASKET フォルダの中のメッセージが削除される。Ctrl/Z を押しても Mail を終了できる。

- QUIT

Mail を終了する。しかし、WASTEBASKET フォルダは空にならない(EXIT コマンドを入力するか、Ctrl/Z を押さない限り、削除されたメッセージは破壊されない)。QUIT は Ctrl/Y と同じ働きをする。

- SPAWN [コマンド]

現在のプロセスのサブプロセスを作成する。SPAWN コマンドを使用すると、Mail を一時的に終了して他の機能(ディレクトリ・リストの表示やファイルの印刷など)を実行してから、Mail に戻ることができる。

7.14.9 メール・ファイルの圧縮

メール・ファイルを圧縮するには、次のコマンドを使用します。

- COMPRESS [ファイル指定]

指定されたメールを圧縮する。ファイル名が指定されないと、現在オープンされているメール・ファイルを圧縮する。オープンされているメール・ファイルがない場合には、省略時のメール・ファイル (MAIL.MAI) を圧縮する。

7.14.10 システム管理コマンド

次のコマンドはシステム管理のために使用します。

- REMOVEユーザ名

システムのメール・プロフィール、データ・ファイル
SYS\$SYSTEM:VMSMAIL_PROFILE.DATA からユーザのレコードを削除する。SYSPRV 特権が必要。

- SHOW FORWARD

ユーザの現在転送中のアドレス名を表示する。

- SHOW PERSONAL_NAME

ユーザが SET PERSONAL_NAME コマンドで設定したテキスト文字列を表示する。

7.15 MIME ユーティリティ

MIME (多目的インターネット・メール拡張機能) は、テキスト以外のファイルをメール・メッセージに添付するために使用される標準規則です。MIME ユーティリティにより、MIME エンコードしたメール・メッセージを作成、あるいは読み込むことができます。MIME を使用すると、グラフィックスや音声ファイルなどのテキスト以外のファイルを、プレーン・テキストとしてエンコードして送信することができます。ただし、このテキストは読めません。MIME ユーティリティは、MIME ファイルを元の形式にデコードすることができ、また MIME エンコード・ファイルを作成できます。このファイルを OpenVMS Mail ユーティリティを使用してメール・メッセージとして送信することができます。

7.15.1 MIME ユーティリティの起動

MIME のためのフォーリン・コマンドは、システム管理者によって設定済みのはずですが、設定されていない場合は、LOGIN.COM: に次の行を追加することによって設定できます。

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

MIME では、MIME エンコード・テキスト・ファイルのみを開きます。まず Mail で MIME エンコード・メッセージをテキスト・ファイルに抽出する必要があります (詳細については第 7.6.3 項を参照)。

DCL プロンプトから MIME ユーティリティを起動するには、次のように入力します。

```
$ MIME ファイル名.TXT
```

ファイル名修飾子はオプションです。指定したファイルが存在する場合、そのファイルは、READ_ONLY で開けられます。

- /READ_ONLY は、そのファイルが、デコードしようとしているユーザが受信したメッセージの中身であることを示します。これは省略時の設定です。
- /DRAFT は、先のセッションで作成されたメッセージがファイルにあって、それが WRITE_ACCESS で開かれていることを示します。

MIME ユーティリティでは、To: フィールドや From: フィールドなどのヘッダ情報が作成されません。MIME ユーティリティでは MIME ヘッダとメッセージの本体テキストのみが作成され、後でテキストは Mail が送信するファイルに保存されます。指定したファイルに認識可能なヘッダまたは RFC822 ヘッダがあると、ファイルは開かれ、省略時のアクセス許可は/READ_ONLY となります。

指定ファイルに認識可能なヘッダがない場合やそのファイル自体がない場合、OPEN FILE ERROR メッセージが表示されます。

MIME エンコード・メッセージを表示するシステム単位の省略時の設定は、MIME\$MAILCAP.DAT と MIME\$FILETYPES.DAT という 2 つのファイルを作成して定義できます。

MIME\$MAILCAP.DAT には、それぞれローカル認識されるコンテンツ・タイプの MIME エンコード・ファイルを定義するアプリケーションが組み込まれています。MIME\$FILETYPES.DAT では、それぞれのコンテンツ・タイプをファイル拡張子に関連付けます。ユーザは、SYS\$LOGIN にこれらのファイルを作成して省略時の設定に上書きできます。

7.15.2 MIME ユーティリティの初期化

MIME ユーティリティを開始すると、初期化プロセスが以下のステップで行われます。

1. ユーザの VMSmail プロファイルで、MIME ユーティリティは、MIME ユーティリティで使用するためのユーザのメール・ディレクトリと省略時のエディタを探します。

2. MIME ユーティリティは、ファイル
MIMESMAILCAP.DATとMIMESFILETYPES.DATを読み取ります。
3. MIME ユーティリティは、以下の内部省略時の設定リストを参照します。

- コンテンツ・タイプ

MIME ユーティリティは、入力メッセージを表示する前にコンテンツ・タイプ・リストを参照します。このリストは、MIME ユーティリティが認識するコンテンツ・タイプと、各コンテンツ・タイプを元の形式にデコードするときに必要な情報がおさめられています。

以下に示すのは、RFC 1524 で作成した MAILCAP エントリの例です。

```
image/*; xview %s
```

コンテンツ・タイプは、MIMESMAILCAP.DAT ファイルを作成してリスト
MIME 認識に追加できます。(例 7-1 は MIMESMAILCAP.DAT ファイルの例
です。)

- ファイル拡張子

MIME ユーティリティは、出力メッセージの構成時にファイル拡張子リストを
参照します。このリストには OpenVMS ファイル拡張子と、各拡張子に関連付
けられたコンテンツ・タイプがおさめられています。MIME ユーティリティ
では、構成する MIME フォーマットのメッセージ本体に、これらの拡張子が
必要です。

ファイル拡張子リストの各行は、以下の項目からなります。

拡張子,コンテンツ・タイプ/サブタイプ,(オプションで)Content-Transfer-Encoding文字列

以下に示すのは、ファイル拡張子リストの行の例です。

```
doc, application/ms-word, base64
```

表 7-1 で説明した MIMESFILETYPES.DAT ファイルを作成すれば、MIME
ユーティリティが認識するファイル拡張子と照合用のコンテンツ・タイプをリ
ストに追加できます。

7.15.3 オプションの MIME ユーティリティ・ファイルの作成

表 7-1 に、システム上の MIME ユーティリティをカスタマイズするときを作成する
可能性のあるファイル名と説明を示します。

表 7-1 MIME ユーティリティのオプション・ファイル

ファイル	目的
MIME\$MAILCAP.DAT	入力メッセージの表示と構文解析用
MIME\$FILETYPES.DAT	出力アタッチ・ファイルへのコンテンツ・タイプ割り当て用

以上のファイルはSYS\$LOGINディレクトリに配置します。

7.15.3.1 MIME\$MAILCAP.DAT ファイル処理

MIME\$MAILCAP ファイル形式は、RFC 1524、1993 年 9 月発行の N. Borenstein 著、『A User Agent Configuration Mechanism for Multimedia Mail Format Information』に端を発します。MIME ユーティリティでは、このファイルの命令でメッセージとアタッチメントを解釈、表示します。これらの命令により、MIME ユーザ・エージェントは外部プログラムを呼び出して MIME メッセージであるコンテンツ・タイプを表示します。

MIME\$MAILCAP.DAT ファイルをカスタマイズして、特定のコンテンツ・タイプの FDL(File Descriptor Language) を指定すれば、特定のコンテンツ・タイプでシステムのメッセージ・パーツを抽出できます。例 7-1 は、MIME\$MAILCAP.DAT ファイルの例です。

注意

プログラム名の参照は、論理名が有効なファイル指定とします。

例 7-1 MIME\$MAILCAP.DAT ファイル

```
#
# 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
#
```

7.15.3.2 MIME\$FILETYPES.DAT ファイルの処理

オプションの MIME\$FILETYPES.DAT ファイルには、互いに関連付けられた OpenVMS ファイル拡張子と MIME コンテンツ・タイプのリストがおさめられています。ADD コマンドの処理では、構成するメッセージにアタッチする OpenVMS ファイルのコンテンツ・タイプを FILETYPE 構造で指定します。

ファイル形式の構文は、MIME\$MAILCARDAT ファイルの構文に似ており、コメントは“#”文字で表します。ファイルの各行には拡張子を 1 つ (前に ‘.’ なし)、続けてコンテンツ・タイプ、拡張子を使用するファイルに関連付けられたサブタイプを指定します。

オプションで、行には Content-Transfer-Encoding 文字列 (7 ビット, 8 ビット, Base64 または引用符で囲んだ印刷可能文字列) を追加できます。これは、メッセージの伝送字にファイル・コンテンツをエンコードする文字列です。7 ビット, 8 ビット, Base64 または引用符で囲んだ印刷可能文字列は、標準 MIME エンコードであり、これ以外は使用できません。エンコードを指定しないと、MIME ユーティリティは 7 ビットが使用されます。

7.15.4 MIME ユーティリティを使用した MIME エンコード・ファイルの抽出

MIME ユーティリティを使用して MIME エンコード・ファイルを抽出するには、まずデコードしたいファイルをオープンします。ファイル名を指定して MIME ユーティリティを起動するか、または MIME ユーティリティでファイルをオープンするといういずれかの方法で、ファイルをオープンすることができます。EXTRACT コマンドは、ネイティブ・ファイル形式または/FDL 修飾子で指定した別の形式で指定アタッチメントを抽出します。

以下に示すのは、メッセージ・ファイルを開き、読み取り可能なテキストでメッセージを表示し、メッセージ属性をリストするための代表的な MIME ユーティリティ・コマンドです。

```
MIME> OPEN file-name
MIME> READ
MIME> LIST
```

アタッチメントを抽出するには、以下のコマンドを入力します。

```
MIME> EXTRACT /ATTACHMENT=n destination-file-name
```

アタッチメントを単独で指定するには、/ATTACHMENT=*n* 修飾子を追加します。これで、抽出するアタッチメント番号を指定します。また指定アタッチメントを出力ファイルに変換するときに使用する FDL (File Descriptor Language) 定義ファイルを指定する /FDL=ファイル名も使用できます。各アタッチメントの番号は、LIST コマンドで表示します。

MIME ユーティリティで使用するコマンドの全リストは、第 7.15.6 項を参照してください。

7.15.5 MIME ユーティリティを使用したファイルのエンコード

アタッチメントとして送信するファイルをエンコードするには、まず MIME ユーティリティを呼び出して NEW コマンドを指定して新しいファイルを作成します。ファイル名を指定しないと、NEW のプロンプトでファイル名が要求されます。

```
$ MIME NEW new-file-name
```

また、MIME ユーティリティでは OPEN コマンドでドラフト・メッセージ・ファイルを開くことができます。

```
MIME> OPEN/DRAFT file-name
```

以前のセッションで作成したファイルをオープンするには、コマンドに修飾子 /DRAFT を指定します。

ファイルに添付ファイルを追加するには、次のコマンドを入力します。

```
MIME> ADD ファイル名
```

このコマンドのオプションの修飾子の全リストは、第 7.15.6 項を参照してください。

このファイルに現在の情報を書き込むには、SAVE コマンドを使用します。いったんセーブすると MIME エンコード・ファイルは、OpenVMS Mail ユーティリティを使用してファイルとして送信することができます。

MIME ユーティリティを終了するには、QUIT または EXIT コマンドを入力します。

MIME ユーティリティで利用できるコマンドの完全なリストについては、第 7.15.6 項を参照してください。

7.15.6 MIME ユーティリティ・コマンド

以下のリストに、コマンド、パラメータ、MIME ユーティリティで利用できる修飾子について説明します。説明の後にそれぞれ例を示します。

ADD — 新しい本体部分または添付ファイルを編集しているメッセージに追加します。ADD コマンドには、パラメータとして添付したいファイルの名前が必要です。オプションの修飾子は次のとおりです。

- /BINARY — コンテンツ・タイプを "application/octet-stream" に指定し、コンテンツ転送エンコードを "base64" に指定します。この形式で任意バイナリ・データ・ストリームを表現します。
- /CONTENT-TYPE=タイプ — 省略時の設定のコンテンツ・タイプを指定された文字列に変更します。たとえば "IMAGE/JPEG." です。

- /ENCODING_TYPE={7Bit | 8Bit | Base64 | Quoted-Printable} — 省略時設定のエンコード・タイプを指定されたエンコード・タイプに変更します。
- /MESSAGE — 添付ファイルはメッセージ・ファイルです (標準 RFC822)。
- /TEXT — 添付ファイルはテキストのコンテンツタイプです。

MIME> ADD ファイル名/TEXT

CLOSE — 現在のメッセージ・ファイルをクローズします。最新の変更内容を保存していない場合には、クローズする前に MIME ユーティリティがプロンプトで知らせます。ファイルが/READ_ONLY である場合は、ファイルは変更されません。

MIME> CLOSE

EDIT — 指定された添付ファイルに対するユーザの省略時設定のテキスト・エディタを起動します。

MIME> EDIT attachment-number

EXIT — 処理中の作業を保存して、MIME エディタを終了します。

MIME> EXIT

EXTRACT - ネイティブ・ファイル形式でファイルに指定アタッチメントを抽出します。

- /ATTACHMENT=n — 抽出するアタッチメント番号を指定します。
- /FDL=filename — 指定アタッチメントを出力ファイルに変換するときに使用する FDL(File Descriptor Language) 定義ファイルを指定します。

MIME> EXTRACT ファイル名/ATTACHMENT=n

HELP — MIME ユーティリティのヘルプ・ファイルを表示します。

MIME> HELP

LIST — 本体部分およびアタッチメント番号のような属性のリストを含む現在のメッセージに関する情報を表示します。

MIME> LIST

NEW — 新しいメッセージを作成します。

MIME> NEW ファイル名

OPEN — 指定されたファイル名を持つメッセージをオープンします。使用できる修飾子は次のとおりです。

- /DRAFT — メッセージ・ファイルは以前のセッションで作成されたドラフトです。

- /READ — メッセージは読み込み専用で更新することができません。

MIME> OPEN ファイル名/NEW

QUIT — 現在のメッセージを保存せずに、現在の MIME 編集セッションを強制終了します。

MIME> QUIT

READ — 読み取り可能なテキストで現在のメッセージを表示します。必要に応じてアタッチメントを表示します。

MIME> READ

REMOVE — 現在のメッセージから指定された添付ファイルを削除します。

MIME> REMOVE 1

SHOW — 指定したオプションにもとづいて、MIME 環境に関する情報を表示します。使用できるオプションは、CONTENT_TYPE、FILE_TYPES、VERSION です。

MIME> SHOW option

SAVE — 現在のメッセージをファイルに書き込みます。ファイル名が指定されている場合は、それを使用します。

MIME> SAVE ファイル名

7.15.7 エラー処理

エラー条件は、OpenVMS シグナリング・サブシステム、具体的には lib\$signal() および lib\$stop() を使用して報告されます。エラー条件には、回復不可能、エラー、警告という 3 つのレベルの重大度があります。これらのレベルは、条件からどのような結果が予測されるかを示します。重大度と対応する結果については、以下のリストで説明します。

- Fatal (-F-) は、プログラムを即時終了させます。
- Error (-E-) は、現在アクティブなコマンドを終了させ、既存のメッセージ・コンテンツは保持します。
- Warning (-W-) は、MIME 編集セッションに割り込むことなく、現在のコマンドを終了させます。ただし、これはコマンドがそのタスクを正常にすべて完了したということではありません。エラーがないかチェックしてください。

EVE エディタによるテキスト・ファイルの編集

テキスト・エディタを使用すると、テキスト・ファイルを作成したり変更したりできます。テキスト・エディタで、キーボードから入力したテキストをテキスト編集コマンドを使用して変更できます。たとえば、レポート用のデータを入力した後で、段落を編成したり、情報をコピーしたり、句を置換したり、テキストを編集したりすることが可能です。また、テキスト・エディタは、プログラミング言語のソース・ファイルの作成や変更にも使用することもできます。OpenVMS オペレーティング・システムは、複数のテキスト・エディタをサポートしています。

Extensible Versatile Editor (EVE) は、DEC Text Processing Utility (DECTPU) を基盤とする汎用テキスト・エディタです。DECTPU は高性能のプログラム可能なテキスト・プロセッサです。本章では、次のことについて説明します。

- EVE の特徴
- ヘルプ情報
- 編集セッションの開始
- コマンドの入力
- 編集内容のセーブと EVE の終了
- カーソルの移動
- テキストの入力
- テキストの削除と復元
- テキストの移動
- テキストのコピー
- ボックス編集
- 保留削除の使用法
- テキストの検索と置換
- コマンド行修飾子の使用法
- EVE の別の起動方法
- ジャーナリングと回復
- EVE 書式化コマンド
- バッファの使用法
- サブプロセスの作成

EVE について詳しくは、EVE のオンラインヘルプおよび『Extensible Versatile Editor Reference Manual』を参照してください。

EDT について詳しくは、『OpenVMS EDT Reference Manual』を参照してください。

表記法

本章では、EVE のキー名の表記法として (SHOW KEY コマンドまたは HELP KEYS コマンドを使用したときの) 次のように表示します。すなわち、コントロール・キー、シフト・ファンクション・キー、および Alt キーの組み合わせにはスラッシュを使用し、GOLD キー・シーケンスにはスペースまたはダッシュを使用します。したがって、1 つのキー (Ctrl など) を押したまま別のキーを押す必要がある場合は、スラッシュで示します。また、1 つのキーを押した後に別のキーを押す場合 (GOLD-Help など) は、スペースまたはダッシュで示します。

8.1 EVE の機能

DECTPU は高性能のプログラム可能なテキスト・プロセッサです。EVE を使用すれば、ビジネス文書、技術文書、およびプログラム・ソース・ファイルなどのテキスト・ファイルを作成および編集できます。

EVE は、OpenVMS オペレーティング・システムの省略時のエディタです。他の省略時のエディタを指定しないかぎり、EDIT コマンドを入力すると EVE が起動されます。

EVE を使用すると、次のことができます。

- 手紙、報告書、プログラム・ソースなどのテキスト・ファイルを作成および編集できる。
- 削除、カット、ペースト、行詰め、検索、置換、およびページ付けなどの整形作業を実行できる。
- 複数のバッファおよびウィンドウを使用して、同一の編集セッション内で異なるファイルを表示および編集できる。
- 編集用にキー操作を定義できる。たとえば、学習シーケンス (いくつかのコマンドやキーストロークを 1 つのキーに対応づける) および EDT キーパッドまたは WPS-PLUS キーパッドの設定などがある。
- カット・アンド・ペーストやその他の編集用に、ボックスで囲むかまたは行単位で範囲を選択できる。
- 文字列パターンの検索にワイルドカードが使用できる。
- エディタ内で DCL コマンド (DIRECTORY など) が実行できる。
- DECspell を実行して、選択範囲または全バッファのスペルチェックができる。
- サブプロセスを生成したり、他のプロセスに接続することができる。

- DECTPU プロシージャをコンパイルおよび実行して、EVE を拡張できる。
- DECwindows インタフェースのメニュー項目を追加および削除できる。
- コンパイルされたプロシージャ、メニュー定義、キー定義、およびその他のカスタマイズ内容を、後のセッションで使用するために保存できる
- スタートアップ時または編集セッション中に初期化ファイルを使用できる。
- システムの障害によって編集セッションが中断したとき、キーストロークまたはバッファ・ジャーナリングを使用して作業内容を復元できる。
- EVE のコマンド、キー、メニュー項目、および DECTPU の組み込みプロシージャを含むその他のトピックに関するオンライン・ヘルプを参照できる。

EVE の起動方法とコマンドの入力方法がわかると、各種の EVE コマンドを使用してファイルを作成したり編集したりできます。編集キーや編集コマンドを使用すれば、カーソルを移動したり、バッファ・モードを設定したり、テキストの入力、削除、復元、移動などの編集を行うことができます。

8.2 ヘルプ情報の参照

編集セッション中はいつでもオンライン・ヘルプを参照できます。EVE エディタでは、2 種類のオンライン・ヘルプを使用できます。

- キーパッド・ヘルプ：ターミナル上の Help キーを使用してアクセスする。
- EVE ヘルプ：EVE のコマンド・プロンプトに対して HELP コマンドを入力することによってアクセスする。

8.2.1 キーパッド・ヘルプの使用方法

キーパッド・ヘルプにアクセスするには、次のようにします。

1. Help キーを押す。

Help ユーティリティがキーパッドの図を表示する。

2. 画面上の指示に従って必要な情報を得る。

- EVE コマンド

EVE コマンドのヘルプを参照するには、コマンド名または疑問符(?)を入力して Enter キーを押す。

- 定義済みのキー

ユーザが定義したキーのヘルプを参照するには、そのキーを押すか、または SHOW KEY コマンドを使用する。

- キー定義の一覧表示

すべてのキー定義の一覧を表示するには、keysと入力して Enter キーを押すか、または GOLD HELP キーを押す。GOLD キーは、数値キーパッド上の PF1 キーまたは NumLock キーである。

3. Help を終了するには、Enter キーを押す。

8.2.2 EVE ヘルプの使用方法

HELP コマンドを使用して EVE ヘルプにアクセスするには、次のようにします。

1. Do キーを押す。

2. HELP コマンドを入力する。

Prev Screen キーまたは Next Screen キーを使用して、使用可能なヘルプ・トピックの一覧をスクロールする。

3. Help を終了するには、Enter キーを押す。

特定のコマンドに関する情報を得るには、HELP の後にそのコマンド名を入力してから Enter キーを押します。ヘルプ・テキストが画面に現れます。HELP TPU コマンドを入力することによって、DECTPU 組み込みプロシージャのヘルプ情報を得ることもできます。

次の例では、MOVE BY LINE コマンドのヘルプ・テキストを示しています。

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
------------------	-------------	------	---------------

8.3 セッションの開始と終了

EVE は、OpenVMS オペレーティング・システムの省略時のエディタです。ユーザまたはシステム管理者が他に省略時のエディタを指定しない限り、次のように EDIT コマンドを入力すると EVE エディタが起動します。

```
$ EDIT
```

EVE が省略時のエディタになっていないシステムで EVE を起動するには、EDIT /TPU コマンドを使用します。編集セッションが開始されたら、既存ファイルの名前または作成する新しいファイルの名前を入力します。編集セッションの開始時にファイル名を指定しないと、EVE は、編集セッションの終了時に Main と呼ばれる省略時のバッファにテキストが追加されているかどうかを確認し、追加されている場合はファイル名の入力を要求します。バッファの使用については、第 8.18 節を参照してください。

次の例は、EVE を起動して、NEWFILE.DAT という名前の新しいファイルを作成します。

```
$ 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 の画面表示を確認する場合には、次のことに注意してください。

1 [End of file]マーカ

EVE バッファの終わりを示している。これは、画面上に見えるだけで、ファイルの一部ではない。バッファにテキストを追加すると、[End of file]マーカが下に移動する。ターミナル画面の長さによっては、大量のテキスト行を含むバッファの冒頭を表示すると、このマーカが見えなくなることがある。

2 ウィンドウ

バッファの内容を表示する画面領域である。EVE のバッファは編集セッションの間だけ存在する。編集セッションの終了後に、ユーザは、バッファの内容を保存するか、または破棄するかを選択できる。

3 ステータス・ライン

EVE ウィンドウの一番下に強調表示され、ウィンドウに表示中のバッファについての情報が示される。ステータス・ラインには、バッファ名、編集状態 (書き込み可または読み込み専用)、現在のモード (挿入または上書)、現在の方向 (順方向または逆方向) が示される。

4 コマンド行

行モードコマンドを入力するために使用する (第 8.4 節を参照)。コマンド行は、Do キーを押すと表示される。

5 メッセージ・ウィンドウ

EVE を起動して、コマンド行にファイル名を指定すると、強調表示されたステータス・ラインの下に表示される。このウィンドウには、最初、新規ファイルであること、または既存のファイルから一定の行数が読み込まれたことのどちらかを示すメッセージが入っている。編集セッションの間、EVE はメッセージ・ウィンドウ内にさまざまなメッセージを表示する。

8.4 コマンドの入力方法

EVE コマンドには、次の 2 つの入力方法があります。

- コマンド行にコマンドを入力する。
- EDT キーパッドまたは WPS キーパッド上で、定義済みキーを使用する。

8.4.1 コマンドの入力

コマンドをコマンド行に入力するには、次のようにします。

1. Do キーを押す。

カーソルがコマンド・ウィンドウに移動し、EVE がコマンドの入力を要求してくる。

2. コマンドを入力する。

コマンドの最初の数文字だけを使用して、コマンド名を省略することもできる。EVE は、大文字と小文字を区別しない。コマンド行には、FIND コマンドまたは REPLACE コマンドに文字列を指定する場合を除いて、大文字と小文字の任意の組み合わせを使用できる。

3. Do キーまたは Enter キーを押す。

EVE はコマンドを実行するか、より詳細な情報の入力を要求する。

8.4.2 定義済みキーによる入力方法

定義済みキーを使用して EVE コマンドを入力することもできます。各定義済みキーは、1 つの編集コマンドを実行します。EVE 関数を実行するためにユーザ独自のキーを定義することもできます。

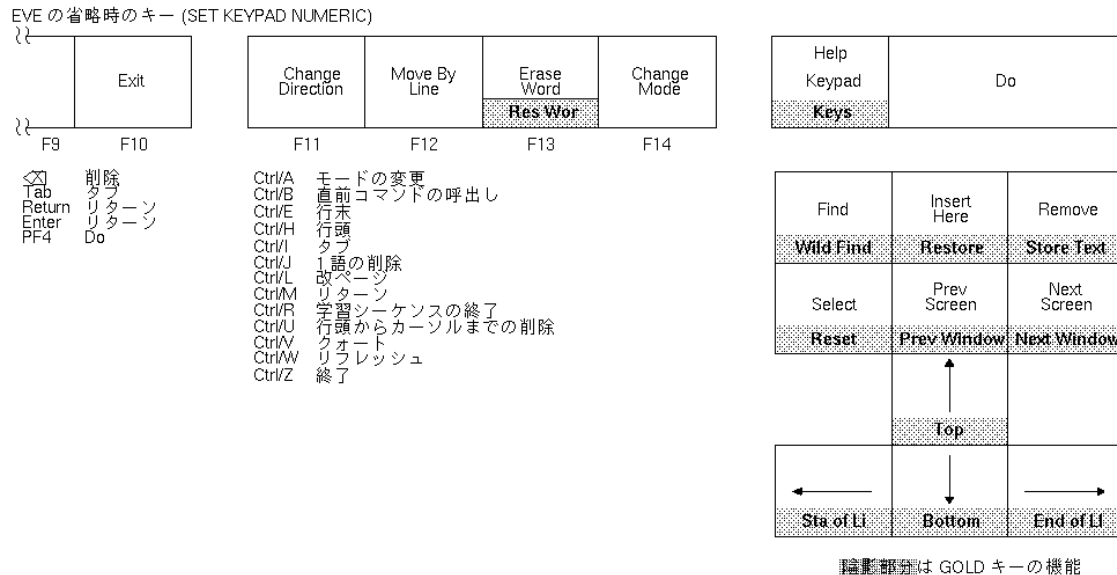
EVE では、いくつかのキーが省略時の値として設定されています。VT200、VT300、および VT400 の各シリーズのターミナルには、次のキーがあらかじめ定義されています。

- ミニキーパッド (メイン・キーボードのキーと数値キーパッドの間の、矢印キーの上にある)
- ファンクション・キー
- コントロール・キーのシーケンス

コントロール・キー・シーケンス、矢印キー、Tab、Return、Delete の各キーは、3 つのタイプのターミナルで同じように定義されています。

図 8-1 は、VT200、VT300、および VT400 の各シリーズのターミナルの定義済みキーを示しています。

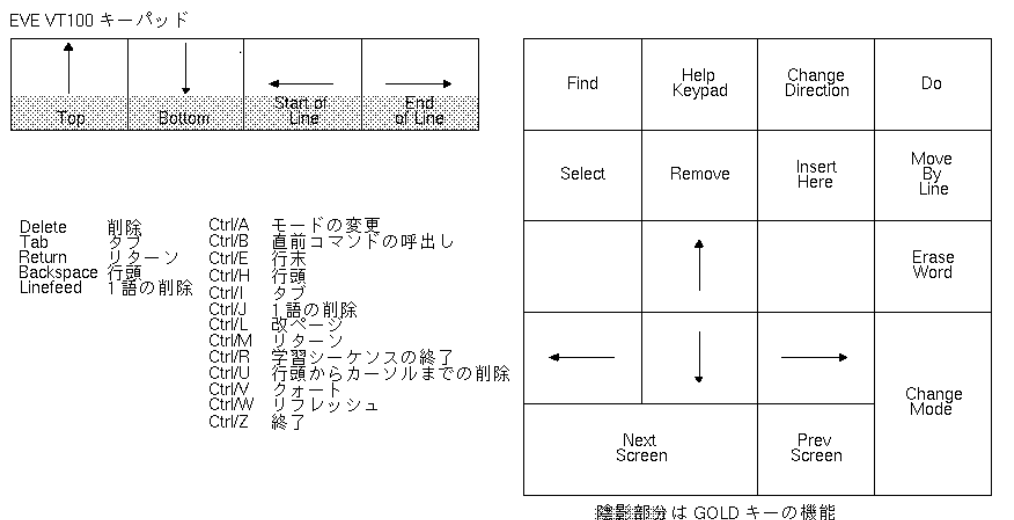
図 8-1 EVE のキー: VT200 シリーズ、VT300 シリーズ、および VT400 シリーズのターミナル



JRD-6300-GE

VT100 シリーズ・ターミナルでは、EVE は、ほとんどの数値キーパッド・キー、4 つの矢印キー、およびコントロール・キーのシーケンスを自動的に定義します。図 8-2 は、VT100 シリーズ・ターミナルの定義済みキーを示しています。

図 8-2 EVE のキー: VT100 シリーズ・ターミナル



JRD-6301-GE

8.5 編集内容のセーブと EVE の終了

編集内容をセーブして EVE を終了するには、次の 3 つの方法があります。

- WRITE FILE コマンド
編集セッションを終了することなくファイルをセーブする。
- EXIT コマンド
変更内容をファイルにセーブして、編集セッションを終了する。
- QUIT コマンド
変更内容をファイルにセーブしないで、編集セッションを終了する。

8.5.1 WRITE FILE コマンドの使用

EVE を終了することなく、バッファ内のテキストをファイルに書き出してセーブするには、WRITE FILE コマンドを使用します。バッファがファイルと関連付けられていない場合、EVE は、次のようにファイル名の入力を要求します。

Type filename for buffer Main (press RETURN to not write it):

バッファの内容をファイルに書くには、ファイル名を入力して Enter キーを押します。

8.5.2 EXIT コマンドの使用方法

編集したテキストをセーブするには、EXIT コマンドを使用します。EXIT コマンドは、F10 キーまたは Ctrl/Z を押すことによって実行できます。

現在のバッファを変更した場合は、元のバージョンと同じファイル名と同じファイル・タイプを持ち、バージョン番号が 1 だけ大きい新しいバージョンのファイルが作成されます。たとえば、FUN.DAT;1 というファイルを変更した後に EXIT コマンドを使用すると、FUN.DAT;2 という名前のファイルが作成されます。

8.5.3 QUIT コマンドの使用

編集内容をセーブせずにセッションを終了するには、QUIT コマンドを入力します。編集内容をセーブせずにセッションを終了する場合は、Y を入力してから Enter キーを押します。編集内容をセーブすることにした場合には、N を入力して Enter キーを押してから、EXIT コマンドを使用してバッファを終了します。

現在のバッファ以外のバッファを変更した場合には、EVE は、そのバッファの内容をセーブするかどうかを聞いてきます。Y を入力すると、バージョン番号が 1 だけ大きい既存ファイルの新しいバージョンが作成されます。既存ファイルが存在しない場合は、ファイル名の入力を求めるプロンプトが表示されます。

どのバッファの内容も変更されていない場合には、EXIT と QUIT は同じになります。たとえば、ファイルを編集するのではなく参照するために EVE を使用する場合は、Ctrl/Z を押して終了できます。

次の例では、FUN.DAT という名前のバッファが変更されており、QUIT コマンドが入力されています。

```
Command: QUIT  
Buffer modifications will not be saved, continue quitting (Y or N)?
```

8.6 カーソルの移動

EVE を使用してファイルを編集する場合、編集を行う場所までカーソルを移動します。テキスト内でより速くかつより効率的にカーソルを移動できれば、それだけ編集必要な時間を節約できます。カーソルの移動には、キーボードまたはコマンドを使用できます。

表 8-1 は、カーソル移動に使用する EVE 編集キーの一覧です。GOLD キーとの組み合わせについての詳しい説明は、オンライン・ヘルプの「GOLD」というトピックを参照してください。

表 8-1 カーソルを移動させる EVE 編集キー

キーまたは キー・シーケンス	機能
上向き矢印キー	MOVE UP と同じ。カーソルを 1 行上に移動する。VT100 シリーズ・ターミナルでは、KP5 も MOVE UP として定義されている。
下向き矢印キー	MOVE DOWN と同じ。カーソルを 1 行下に移動する。VT100 シリーズ・ターミナルでは、KP2 も MOVE DOWN として定義されている。
左向き矢印キー	MOVE LEFT と同じ。カーソルを 1 文字左に、つまり 1 桁左に移動する。VT100 シリーズ・ターミナルでは、KP1 も MOVE LEFT として定義されている。
右向き矢印キー	MOVE RIGHT と同じ。カーソルを 1 文字右に、つまり 1 桁右に移動する。VT100 シリーズ・ターミナルでは、KP3 も MOVE RIGHT として定義されている。
Ctrl/E または GOLD 右向き矢印キー	END OF LINE と同じ。カーソルを現在行の行末まで移動する。
Ctrl/H または GOLD 左向き矢印	START OF LINE と同じ。カーソルを現在行の行頭まで移動する。
GOLD 上向き矢印	TOP と同じ。カーソルを現在のバッファの先頭まで移動する。
GOLD 下向き矢印	BOTTOM と同じ。カーソルを現在のバッファの終わりに移動する。
GOLD Next Screen	NEXT WINDOW と同じ。2 つ以上のウィンドウを使用している場合に、カーソルを画面上の次のウィンドウに移動する。カーソルは、そのウィンドウで最後に置かれていた場所に現れる。
GOLD Prev Screen	PREVIOUS WINDOW と同じ。2 つ以上のウィンドウを使用している場合に、カーソルを画面上の前のウィンドウに移動する。カーソルは、そのウィンドウで最後に置かれていた場所に現れる。

表 8-2 は、カーソル移動に使用する EVE コマンドの一覧です。

表 8-2 EVE のカーソル移動用コマンド

コマンド	機能
BOTTOM	カーソルを現在のバッファの終わりに移動する。省略時の設定では、GOLD 下向き矢印が BOTTOM として定義されている。
CHANGE DIRECTION	現在のバッファの方向を変更する。バッファの方向はステータス・ラインに示される。
END OF LINE	カーソルを現在行の行末まで移動する。省略時の設定では、Ctrl/E と GOLD 下向き矢印が END OF LINE として定義されている。
FORWARD	省略時の設定である。現在のバッファの方向を順方向 (右下方向) に設定する。バッファの方向はステータス・ラインに示される。
GO TO	MARK コマンドによってラベルが付けられた場所にカーソルを移動する。
LINE	行番号によって指定された行の先頭にカーソルを移動する。

(次ページに続く)

表 8-2 (続き) EVE のカーソル移動用コマンド

コマンド	機能
MARK	見えない現在の位置にマーカを置き、指定された名前とそのマーカを関連づける (マーカ自体は画面に表示されない)。後で、GO TO コマンドを使用して、マークの付いた位置に戻ることができる。
MOVE BY LINE	順方向では、カーソルを現在行の行末まで移動する。カーソルがすでに行末にある場合には、次の行の行末に移動する。逆方向では、カーソルを現在行の先頭に移動する。カーソルがすでに行の先頭にある場合には、前の行の先頭に移動する。VT200, VT300, および VT400 の各シリーズのターミナルでは、F12 キーが MOVE BY LINE として定義されている。VT100 シリーズのターミナルでは、キーボード上のマイナス・キー (-) が MOVE BY LINE として定義されている。
MOVE BY PAGE	現在の方向に応じて、カーソルを前後のページ・ブレイク (改ページ) に移動する。現在の方向にページ・ブレイクがない場合には、カーソルはバッファの終わりまたは先頭に移動する。
MOVE BY WORD	順方向では、カーソルを次の単語の先頭に移動する。カーソルがすでに行末にある場合には、次の行の先頭に移動する。逆方向では、カーソルを前の単語の先頭に移動する。カーソルがすでに行の先頭にある場合には、前の行の行末に移動する。
NEXT SCREEN	現在のバッファの中で現在のウィンドウの行数から 1 を引いた行数だけ順方向にスクロールする。たとえば、現在のウィンドウが 12 行の場合、NEXT SCREEN コマンドは、カーソルを順方向に 11 行スクロールする。VT200, VT300, および VT400 の各シリーズのターミナルでは、E6 キー (Next Screen) が NEXT SCREEN として定義されている。VT100 シリーズのターミナルでは、キーボード上の KP0 キーが NEXT SCREEN として定義されている。
NEXT WINDOW または OTHER WINDOW	画面上に次のウィンドウがあれば、カーソルをそのウィンドウに移動する。カーソルは、そのウィンドウで最後に置かれていた場所に現れる。GOLD Next Screen が NEXT WINDOW として定義されている。
PREVIOUS SCREEN	現在のバッファの中で現在のウィンドウの行数から 1 を引いた行数だけ逆方向にスクロールする。たとえば、現在のウィンドウが 12 行の場合、PREVIOUS SCREEN コマンドは、カーソルを 11 行だけ逆方向にスクロールする。VT200, VT300, および VT400 の各シリーズのターミナルでは、E5 キー (Prev Screen) が PREVIOUS SCREEN として定義されている。VT100 シリーズのターミナルでは、キーボード上のピリオド・キー (.) が PREVIOUS SCREEN として定義されている。
PREVIOUS WINDOW	画面上に前のウィンドウがあれば、カーソルをそのウィンドウに移動する。カーソルは、そのウィンドウで最後に置かれていた場所に現れる。GOLD Prev Screen が PREVIOUS WINDOW として定義されている。
REVERSE	現在のバッファの方向を逆方向、すなわち左上方向に設定する。バッファの方向はステータス・ラインに示される。
SET CURSOR BOUND	カーソルがテキストの流れに沿って移動するようにする。カーソルは、バッファの未使用部分には入れない。EDT, WPS, その他エディタのカーソル動作と似ている。
SET CURSOR FREE	省略時の設定値である。カーソルをバッファ内のどこにでも置くことができ、その場所にテキストを入力することができる。

(次ページに続く)

表 8-2 (続き) EVE のカーソル移動用コマンド

コマンド	機能
SET SCROLL MARGINS	カーソルの上下の移動につれて自動的にスクロールを開始する、ウィンドウの最上部または最下部からの距離を設定する。この距離は、行数またはウィンドウ・サイズのパーセンテージで指定する。省略時の設定値は 0 である。すなわち、ウィンドウの最上部または最下部を超えなければ、スクロールしない。
SHIFT LEFT	EVE の現在のウィンドウを指定の桁数だけ左に移動する。SHIFT RIGHT コマンドと SHIFT LEFT コマンドを使用すれば、ウィンドウ幅を変更せず、132 桁モードも使用しなくても、長いテキスト行の未表示部分を見ることができる。SHIFT LEFT コマンドは、SHIFT RIGHT コマンドを使用してウィンドウを右に移動していた場合に、ウィンドウ位置を元に戻すために使用する。
SHIFT RIGHT	EVE の現在のウィンドウを指定の桁数だけ右に移動する。SHIFT RIGHT コマンドと SHIFT LEFT コマンドを使用すれば、ウィンドウ幅を変更しなくても、長いテキスト行の未表示部分を見ることができる。
START OF LINE	カーソルを現在の行の先頭に移動する。省略時の設定では、Ctrl/H と GOLD 左向き矢印キーの両方が、START OF LINE として定義されている。
TOP	カーソルを現在のバッファの先頭 (左上隅) に移動する。省略時の設定では、GOLD 上向き矢印キーが TOP として定義されている。

操作手順: EVE でのカーソルの移動

次の操作手順は、バッファ内でカーソルを移動する方法を示しています。

1. EVE を起動し、次のコマンドを使用してバッファ SCHEDULE.DAT を作成する。

```
$ EDIT SCHEDULE.DAT
```

EVE は、バッファの先頭にカーソルを位置づけ、入力待ちの状態になっている。

2. 次のテキストを入力する。

```
Schedule for 1 July  
10:00 AM meeting with supervisor  
Read and review memo from Sally  
Work on Pascal program
```

テキストを入力するにつれて[End of file]マーカがバッファ内を下へ移動し、入力したテキストの最後にカーソルが位置づけられる。

3. TOP コマンドを入力して、カーソルをファイルの先頭に移動する。
4. Ctrl/E を押して、カーソルを (テキストの最初の行の) 行末に移動する。Ctrl/E は、DCL 内と同じように働く。
5. BOTTOM コマンドを入力して、カーソルをバッファの終わりに移動する。
6. 上向き矢印キーを押して、カーソルを 1 行上、すなわちテキストの 4 行目に移動する。
7. Change Direction キーを押して、現在のバッファの方向を逆方向に変更する。

8. Move by Line キーを押して、カーソルを 3 行目の行頭に移動する。
9. コマンド LINE 1 を入力して、カーソルをバッファの 1 行目の行頭に移動する。
10. Ctrl/Z を押して EVE を終了する。

8.7 テキストの入力

ユーザは、現在編集中のバッファに、キーボードの文字、ファイル全体、および印刷不可能な特殊文字 (制御文字など) を入力できます。テキストを入力するには、キーボードまたはコマンドを使用します。また、バッファにテキストやファイル、特殊文字を追加することもできます。

8.7.1 テキストの追加

キーボードから文字を入力し、バッファの現在のカーソルの位置の後に追加することができます。入力した文字は、バッファが挿入モードであるのか、上書モードであるのかに応じて、既存の文字の前に挿入されるか、または既存の文字の上に上書きされます。

8.7.2 ファイルの挿入

ファイル全体を追加するには、Do キーを押してから EVE コマンドの INCLUDE FILE を入力します。File to include: プロンプトに対してファイル指定を入力してから Enter キーを押します。バッファの現在のモード (挿入または上書き) にかかわらず、指定されたファイルの全内容が現在のカーソル行の前に挿入されます。

ファイルを指定するには、ワイルドカードが使用できます。あるワイルドカードのファイル指定に一致するファイルが複数存在する場合には、それらがすべて表示され、より完全なファイル指定の入力を求めるプロンプトが表示されます。指定されたファイルが存在しない場合は、そのファイルを取り込めなかったことを知らせるメッセージが表示されます。

8.7.3 印刷不可能な特殊文字

QUOTE コマンドを使用すれば、Ctrl/V を押してから特殊文字を入力することによって、印刷不可能な特殊文字を追加できます。たとえば、バッファ内にエスケープ文字を追加するには、Ctrl/V を押してから Ctrl/[を押します。特殊文字は、バッファの現在のモード (挿入または上書き) に応じて追加されます。

8.7.4 EVE のテキスト入力用編集キー

次の表は、テキスト入力に使用する EVE の編集キーの一覧です。

キーまたは キー・シーケンス	機能
Ctrl/A	CHANGE MODE コマンドと同じ。現在のバッファの編集モードを変更する。編集モードは強調表示のステータス・ラインに示される。挿入モードでは、テキストは現在の文字位置に挿入され、既存のテキストは右に移動する。上書モードでは、テキストは現在の位置で上書きされる。VT200, VT300, および VT400 の各シリーズのターミナルでは、F14 キーが CHANGE MODE として定義されている。VT100 シリーズのターミナルでは、キーボード上の Enter キーが CHANGE MODE として定義されている。
Ctrl/V	QUOTE コマンドと同じ。印刷不可能な文字や制御コードを挿入できるようにする。特殊文字を検索するには、最初に Find キーを押してから、Ctrl/V と検索対象の特殊文字のキーを押す。次に、Enter キーを押して、検索を実行する。

8.7.5 テキスト入力用 EVE コマンド

次の表は、テキストの入力に使用するコマンドの一覧です。

コマンド	機能
CHANGE MODE	Ctrl/A と同じ。現在のバッファの編集モードを変更する。編集モードは強調表示のステータス・ラインに示される。挿入モードでは、テキストは現在の文字位置に挿入され、既存のテキストは右に移動する。上書モードでは、テキストは現在の位置で上書きされる。VT200, VT300, および VT400 の各シリーズのターミナルでは、F14 キーが CHANGE MODE として定義されている。VT100 シリーズのターミナルでは、キーボード上の Enter キーが CHANGE MODE として定義されている。
INCLUDE FILE	指定されたファイルの内容を現在のバッファのカーソル位置の上にある行に挿入する。このコマンドはファイルのマージのときに役立つ。
INSERT MODE	現在のバッファのモードを、上書モードではなく挿入モードに設定する。挿入モードでは、現在位置にテキストが挿入され、既存のテキストは右に移動する。
OVERSTRIKE MODE	現在のバッファのモードを、挿入モードではなく上書モードに設定する。上書モードでは、現在位置でテキストが上書きされる。
QUOTE	Ctrl/V と同じ。あるキーを押すことによって、そのキーの印刷不可能な文字または制御コードを入力する。FIND コマンドまたは REPLACE コマンドに文字列を指定する場合に、制御コードまたはその他の文字をクォートできる。たとえば、Tab キーをクォートすればタブ文字を検索できる。

8.7.6 バッファ・モードの設定

テキストを入力する前に、バッファの現在のモードが挿入モードか上書モードかを確認してください。

バッファのモードを確認するには、強調表示のステータス・ラインを見ます。挿入モードでは、テキストはカーソル位置に挿入され、既存のテキストは右に移動します。上書モードでは、キーボードから入力したテキストがカーソル位置で上書きされ、バッファ内の既存のテキストはカーソルが移動するにつれて順に上書きされます。

モードを変更するには、Ctrl/A を押してください。

操作手順: 挿入、または上書モードでのテキストの追加

次の操作手順に従って、挿入モードと上書モードの両モードでファイルにテキストを追加してみてください。

1. EVE を起動して、既存のファイル SCHEDULE.DAT を編集する。
2. 強調表示のステータス・ラインを見て、挿入モードであることを確認する。
3. 上書モードの場合は、Ctrl/A を押して挿入モードに変更する。
4. カーソルを *supervisor* という単語の先頭の文字 *s* に移動してから *Engineering* と入力してスペース・バーを押す。

Engineering という単語がテキスト・バッファに追加され、同じ行の残りの部分は右に移動する。

```
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 を押して、上書モードに変更する。
6. カーソルを *Sally* という単語の先頭の文字 *S* に移動してから *Peggy* と入力する。

バッファ内の *Sally* という単語が、*Peggy* という単語に置き換わる。

```
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. Ctrl/Z を押して EVE を終了する。

8.8 テキストの削除と復元

EVE を使用すると、容易にテキストを削除したり、編集中的間違いを修正したりできます。誤ってテキストを削除してしまった場合でも、最後に削除したテキストを元の位置に復元したり、またはカーソルを移動することによって異なる位置に復元することができます。

バッファからテキストを削除するには、削除したいテキストにカーソルを移動してから適切な編集キーを押すか、または適切な EVE コマンドを入力します。

テキストの削除および復元に使用する編集キーの一覧を表 8-3 に示します。

表 8-3 テキストの削除および復元に使用する EVE の編集キー

キーまたは キー・シーケンス	機能
Delete キーまたは Delete	カーソルの左の 1 文字を削除する。DELETE コマンドと同じ。保留削除が使用可能な場合は、選択範囲のテキストを削除し、Restore Selection バッファ内に格納する。保留削除についての詳しい説明は、第 8.9 節を参照。
Ctrl/J	ERASE WORD と同じ。現在の単語を削除する。カーソルが単語間にあるときは、次の単語を削除する。VT200、VT300、および VT400 の各シリーズのターミナルでは、F13 キーが ERASE WORD として定義されている。VT100 シリーズのターミナルでは、キーボード上のコンマ・キー (,) が ERASE WORD として定義されている。
Ctrl/U	ERASE START OF LINE と同じ。カーソルの左から行の先頭までの文字を削除する。
GOLD Insert Here	RESTORE と同じ。EVE コマンドや編集キーによって最後に削除した単語、行、文を現在のカーソル位置に挿入し直す。
GOLD F13	RESTORE WORD (WPS キーボードの場合は除く) と同じ。最後に削除した単語を現在のカーソル位置に挿入し直す。

テキストの削除と復元に使用する EVE コマンドの一覧を表 8-4 に示します。

表 8-4 テキストの削除および復元に使用する EVE コマンド

コマンド	機能
DELETE	カーソルの左の 1 文字を削除する。挿入モードでは、行の残りの部分が左に移動して削除された文字を埋める。上書モードでは、削除された文字はスペースに置き換えられる。行の先頭では、DELETE は、モードにかかわらず、前の行のキャリッジ・リターンを削除して、現在の行が上に移動する。保留削除が使用可能な場合は、選択範囲のテキストを削除し、Restore Selection バッファ内に格納する。保留削除についての詳しい説明は、第 8.9 節を参照。
ERASE CHARACTER	カーソル位置にある文字を削除する。挿入モードでは、行の残りの部分が左に移動して削除された文字を埋める。上書モードでは、削除された文字はスペースに置き換えられる。カーソルが行末にある場合は、モードにかかわらず、キャリッジ・リターンが削除されて次の行が上に移動する。
ERASE LINE	現在の文字から行末までを削除して、次の行を現在の行の終わりに追加する。カーソルが行末にある場合は、キャリッジ・リターンだけが削除されて、次の行が上に移動する。

(次ページに続く)

表 8-4 (続き) テキストの削除および復元に使用する EVE コマンド

コマンド	機能
ERASE PREVIOUS WORD	前の単語またはカーソル位置にある単語を削除する。カーソルが単語間にあるときや単語の最初の文字の上にある場合は、前の単語が削除される。カーソルが単語の途中にあるときは、その単語全体が削除される (ERASE WORD と同じ)。カーソルが行の先頭にあるときは、前の行の行末にあるキャリッジ・リターンが削除されて、現在の行が上に移動する。
ERASE START OF LINE	カーソルの左から行の先頭までの文字を削除する。カーソルがすでに行の先頭にある場合は、何も削除されない。
ERASE WORD	現在の単語を削除する。カーソルが単語間にあるときは、次の単語を削除する。Ctrl/J と同じ。VT200, VT300, および VT400 の各シリーズのターミナルでは、F13 キーが ERASE WORD として定義されている。VT100 シリーズのターミナルでは、キーボード上のコンマ・キー (,) が ERASE WORD として定義されている。カーソルが行末にあるときは、キャリッジ・リターンだけが削除されて、次の行が上に移動する。
RESTORE	EVE コマンドや編集キーによって最後に削除した単語、行、文を現在のカーソル位置に挿入し直す。RESTORE は 1 文字だけの復元は行わない。GOLD Insert Here キーが RESTORE として定義されている。
RESTORE CHARACTER	EVE コマンドや編集キーによって最後に削除した 1 文字を現在のカーソル位置に挿入し直す。上書モードでは、復元された文字はカーソル位置の文字の代わりに置き換えられる。挿入モードでは、復元された文字がカーソル位置に挿入され、したがって既存のテキストが右に移動する。
RESTORE LINE	EVE コマンドや編集キーによって最後に削除した 1 行を現在のカーソル位置に挿入し直す。
RESTORE SELECTION	保留削除を使って、最後に削除したテキストを現在のカーソル位置に挿入し直す。保留削除についての詳しい説明は、第 8.12 節を参照。
RESTORE WORD	EVE コマンドや編集キーによって最後に削除した単語を現在のカーソル位置に挿入し直す。GOLD F13 キーが RESTORE WORD として定義されている (WPS キーボードの場合は除く)。

作成手順: テキストの削除と復元

次の作成手順を使用して、テキストを削除および復元してみてください。

1. EVE を起動してバッファ RHYMES.DAT を作成し、次のテキストを入力する。

```
She rhymes with tree,
also with bee,
and this one makes three.
```

2. カーソルを *also* という単語の文字 *l* に移動してから、ERASE LINE コマンドを入力する。

*also*の*l*から行末までのすべての文字が削除され、次の行が現在の行の終わりに追加される。

```
She rhymes with tree,  
aand this one makes three.
```

3. カーソルを*rhymes*という単語の文字*y*に移動してから、ERASE WORD コマンドを入力する。

*rhymes*という単語が削除され、行の残りのテキストが左に移動する。

```
She with tree,  
aand this one makes three.
```

4. カーソルを2行目の2番目の文字*a*に移動し、RESTORE LINE コマンドを入力する。

最後に削除された行、すなわち*lso with bee*,が復元される。

```
She with tree,  
also with bee,  
and this one makes three.
```

5. カーソルを最初の行にある*with*という単語の文字*w*に移動してから、RESTORE WORD コマンドを入力する。

最後に削除された単語、すなわち*rhymes*が復元される。

```
She rhymes with tree,  
also with bee,  
and this one makes three.
```

6. Ctrl/Z を押して、EVE を終了する。

第 8.9 節に、SELECT コマンドと REMOVE コマンドの機能を示します。これらのコマンドをともに使用すると、バッファからテキストを削除できます。

8.9 テキストの移動

EVE コマンドを使用すると、テキストの一部を選択して、コピー、移動、削除、およびその他の編集作業を行うことができます。ここではテキストを移動する方法について説明します。

データ移動の際のバッファ操作についての詳細は、第 8.18 節を参照してください。

また、行単位ではなく長方形の領域 (ボックス) 単位で、テキストを移動、削除、およびコピーすることもできます。ボックス編集コマンドについての詳細は、第 8.11 節を参照してください。

テキストを移動するには、次の手順に従ってください。

手順	操作
1	EVE でファイルをオープンし、移動するテキストの最初の文字にカーソルを移動する。
2	Select キーを押す。
3	移動するテキストの最後の文字の次の文字位置までカーソルを移動する。逆方向の場合は、最後の文字の次の文字位置までではなく、最後の文字まで移動する。移動するテキストが反転属性によって強調表示される。テキストをバッファに移動しないことにした場合は、再度 Select キーを押して、選択を取り消す。
4	Remove キーを押す。強調表示されたテキストが画面から削除され、Insert Here バッファに格納される。
5	Insert Here キーを押して、テキストを挿入する。 カーソル位置にテキストが挿入される。Insert Here バッファにいったん格納されたテキストは、新しいテキストが選択されて Insert Here バッファに格納されるまで、何度でもどのカーソル位置にも挿入できる。Insert Here バッファには、最後にコピーまたは削除されたテキストが格納されている。

次の表に、テキストの移動に使用する EVE 編集キーを示します。

表 8-5 テキストの移動に使用する EVE 編集キー

キーまたは キー・シーケンス	機能
Insert Here	INSERT HERE コマンドまたは PASTE コマンドと同じ。削除またはコピーしたテキストを現在のカーソル位置に挿入する。
Remove	REMOVE コマンドまたは CUT コマンドと同じ。SELECT コマンドによってマークされたり、FIND コマンドによって強調表示されたテキストを削除して、Insert Here バッファに格納する。
Select	最初のカーソル位置からカーソルを移動した先までのテキストにマークをつける (反転属性で強調表示する)。強調表示されたテキストを選択範囲と呼ぶ。選択範囲を取り消すには、Select キーをもう一度押すか、RESET を使用する。
GOLD Select	RESET と同じ。次のいずれかを取り消して、現在のバッファの方向を順方向に再設定する。 <ul style="list-style-type: none"> 選択範囲または検出範囲の強調表示。 GOLD キーの押下 (または、繰り返し回数に対応する GOLD キーと数字キー <i>n</i> の組み合わせ)。 不完全なコマンド行または再呼び出しされたコマンド行、あるいは Choices バッファ表示。 SHOW, SHOW DEFAULTS BUFFER, SHOW SUMMARY, SHOW WILDCARDS の出力。作業していたバッファに戻る。
GOLD Remove	STORE TEXT コマンドまたは COPY コマンドと同じ。SELECT または FIND によってマークされたテキストをコピーして、Insert Here バッファに格納する。コピーされたテキストは元の位置から削除されない。

次の表に、テキストの移動に使用する EVE コマンドを示します。

表 8-6 テキストの移動に使用する EVE コマンド

コマンド	機能
INSERT HERE または PASTE	コピーまたは削除したテキストを挿入する。省略時の設定では、E2 キー (VT200, VT300, および VT400 の各シリーズのターミナルのミニキーパッド上の Insert Here) または KP9 キー (VT100 シリーズ・ターミナルの場合) が INSERT HERE として定義されている。
REMOVE または CUT	SELECT によってマークされたり、FIND によって強調表示されたテキストを削除し、それを Insert Here バッファに格納する。省略時の設定では、E3 キー (VT200, VT300, および VT400 の各シリーズのターミナルのミニキーパッド上の Remove) または KP8 キー (VT100 シリーズ・ターミナルの場合) が REMOVE として定義されている。
RESET	次のいずれかを取り消して、現在のバッファの方向を順方向に再設定する。 <ul style="list-style-type: none"> • 選択範囲または検出範囲の強調表示。 • GOLD キーの押下 (または、繰り返し回数に対応する GOLD キーと数字キー <i>n</i> の組み合わせ)。 • 不完全なコマンド行または再呼び出しされたコマンド行、あるいは Choices バッファ表示。 • SHOW, SHOW DEFAULTS BUFFER, SHOW SUMMARY, SHOW WILDCARDS の出力。作業していたバッファに戻る。
RESTORE SELECTION	保留削除で削除されたテキストを挿入し直す。保留削除についての詳しい説明は、第 8.12 節を参照。
SELECT	最初のカーソル位置からカーソルを移動した先までのテキストを反転属性で強調表示する。強調表示されたテキストを選択範囲と呼ぶ。選択範囲を取り消すには、SELECT コマンドをもう一度入力するか、RESET を使用する。省略時の設定では、E4 キー (VT200, VT300, および VT400 の各シリーズのターミナルのミニキーパッド上の Select) または KP7 キー (VT100 シリーズ・ターミナルの場合) が SELECT として定義されている。
SELECT ALL	カーソル位置にかかわらず、現在のバッファにあるすべてのテキストを反転属性で強調表示する。強調表示されたテキストを選択範囲と呼ぶ。選択範囲を取り消すには、SELECT コマンドを入力するか、RESET を使用する。SELECT ALL コマンドは、すべてのバッファ内容を間違えて削除しないよう、保留削除を一時的に使用不能にする。
SET NOPENDING DELETE	省略時の設定である。Delete キーを使用したり新しいテキストを入力したときに、選択されたテキストが削除されないようにする。バッファ中にテキストを選択してある場合に、新しいテキストを入力すると文字が選択範囲に追加され、Delete キーを使用するとカーソルの左にある 1 文字だけが削除される。

(次ページに続く)

表 8-6 (続き) テキストの移動に使用する EVE コマンド

コマンド	機能
SET PENDING DELETE	保留削除に設定し、テキスト・ブロックを迅速に削除できるようにする。最初に保留削除を使用可能にし、次に SELECT コマンドを使用して削除したいテキストを選択する。Delete キー (または英数字キーボード上の任意のキー) を押してテキストを削除する。削除した内容を挿入し直すには、テキストを挿入したい位置までカーソルを移動して、RESTORE SELECTION コマンドを入力する。省略時の設定は、SET NOPENDING DELETE である。
STORE TEXT または COPY	SELECT または FIND によってマークされたテキストをコピーして、Insert Here バッファに格納する。コピーされたテキストは元の位置から削除されない。

操作手順: テキストの移動

ある場所からテキストを選択し、削除し、それを別の場所に挿入するには、次の手順に従ってください。

1. ファイル RHYMES.DAT を編集するために EVE を起動する。
2. RHYMES.DAT の 2 行目の先頭にカーソルを移動してから、Select キーを押す。
3. 下向き矢印キーを 1 回押す。

テキストの 2 行目が強調表示される。

4. Remove キーを押す。

現在のバッファからテキストの 2 行目が削除される。

```
She rhymes with tree,  
and this one makes three.  
[End of file]
```

5. Enter キーを 2 回押してから、Insert Here キーを押す。

Insert Here バッファ内のテキストが、現在のカーソル位置に挿入される。

```
She rhymes with tree,  
  
also with bee,  
and this one makes three.  
[End of file]
```

6. Ctrl/Z を押して、EVE を終了する。

8.10 テキストのコピー

COPY コマンドを使用すれば、テキストを任意の場所へコピーできます。STORE TEXT コマンドは COPY コマンドと同じ働きをします。次の例で使用されている COPY コマンドはすべて、STORE TEXT コマンドで置き換えることができます。

操作手順: テキストのコピー

バッファを順方向に設定しているときに、テキストをコピーするには、次の手順に従ってください。

1. ファイル RHYMES.DAT を編集するために EVE を起動する。
2. テキストの最初の行にカーソルを移動する。
3. Select キーを押す。
4. Ctrl/E を押して、カーソルを最初の行の行末に移動する。
5. COPY コマンドを入力する。これで、選択されたテキストのコピーが、Insert Here バッファ内に格納される。
6. カーソルを *also with bee* の 1 行上に移動する。
7. Insert Here キーを押す。バッファは次のようになる。

```
She rhymes with tree,  
She rhymes with tree,  
also with bee,  
and this one makes three.  
[End of file]
```

8. カーソルをテキストの最初の行の行頭に移動する。Select キーと Remove キーを使用して、テキストの最初の行を削除する。
9. Ctrl/Z を押して、EVE を終了する。

8.11 ボックス編集

ユーザは、標準の行単位の範囲だけでなく、長方形の領域、すなわちボックス内のテキストも編集できます。たとえば、リストまたは表内の列を含むボックスを選択してから、それをカットおよびペーストしたり、その他の編集を実行したりできます。

8.11.1 テキスト・ボックスの選択

テキスト・ボックスを選択するには、次の手順に従ってください。

1. 選択開始位置、通常はボックスの左上隅にカーソルを置く。
2. BOX SELECT コマンドを入力する。
3. 通常は左上から右下に向かってカーソルを移動し、ボックスの対角線上の反対側の頂点を決める。

カーソルを移動するにつれて、カーソル通過部分のテキストがボールド体属性で強調表示されます (通常の選択の場合は反転属性を使用します)。ボックスは、対角線上の頂点の組によって定義されます。左上から右下に向かってカーソルを移動する場合

は、カーソル位置の文字はボックスの外側になります。すなわち、カーソルのすぐ左側がボックスの右下隅になります。

テキストを選択したら、通常の行単位またはボックス単位の編集に使用する任意のコマンドを使用してボックスを編集できます。キーを再定義する必要はありません。詳細は、『Extensible Versatile Editor Reference Manual』を参照してください。

検索対象の選択範囲が複数行にまたがらなければ、FIND SELECTED や OPEN SELECTED も使用できます。また保留削除も使用できます。

複数のボックス編集を実行する場合、たとえば、複数カラムの表とリストを編集する場合には、SET BOX SELECT コマンドを使用します。SET BOX SELECT は、複数のコマンドとキーを対応する BOX コマンドとして再定義し、他の編集操作を行単位ではなく、ボックスに対して実行するようにします。

ボックスの選択を取り消すには、再度 SELECT または BOX SELECT を入力するか、RESET を使用します。

8.11.2 テキスト・ボックスのカットとペースト

ボックスをカットすると、カットされた部分に通常はスペースが埋め込まれるので、ボックスの右側にあったテキストが左に動くことはありません。したがって、列の位置も揃ったままです。ボックスをペーストすると、通常、そのボックスは既存のテキストの上に上書きされます。ボックス内またはボックスにまたがるタブ文字は、テキストの左右の位置が変わらないようにするためにスペースに置き換えられます。

次の表は、ボックス編集に使用する EVE コマンドの一覧です。

表 8-7 ボックス編集に使用する EVE コマンド

コマンド	機能
BOX COPY	別の場所にペーストできるよう、テキスト・ボックスを削除せずにコピーする。
BOX CUT	テキスト・ボックスをカットして、別の場所にペーストできるようにする。通常は、ボックスのあった領域にはスペースが埋め込まれるので、ボックスの右側にあったテキストが左に動くことはない。
BOX CUT INSERT	ボックスをカットし、ボックスの右側にあったテキストを左に“ずらして”すき間を埋める。
BOX CUT OVERSTRIKE	ボックスをカットして該当領域にスペースを埋め込み、ボックスの右側にあったテキストが左に動かないようにする。
BOX PASTE	コピーまたはカットしたテキスト・ボックスをペーストする。通常は、既存のテキストに上書きする。

(次ページに続く)

表 8-7 (続き) ボックス編集に使用する EVE コマンド

コマンド	機能
BOX PASTE INSERT	ボックスをペーストして、既存のテキストを右に動かす。
BOX PASTE OVERSTRIKE	ボックスをペーストして、既存のテキストに上書きする。
BOX SELECT	テキスト・ボックスを選択する。一般には、ボックスの左上隅を決めてから、カーソルを右下方向に必要なだけ移動する。
RESTORE BOX SELECTION	保留削除で削除されたボックスを元に戻す (削除を取り消す)。通常は、既存のテキストに上書きする。
SET BOX NOPAD	バッファが上書モードの場合を除いて、ボックス編集のスペース埋込みや上書きを禁止する。
SET BOX NOSELECT	省略時の設定である。ボックスの選択、カット、ペーストを行えないようにする。SELECT、COPY、REMOVE などのコマンドは、標準の行単位の範囲を使用する。ボックスを編集するには、BOX の各種コマンドを使用する。
SET BOX PAD	省略時の設定である。バッファ・モードにかかわらず、ボックス編集で自動スペース埋込みと上書きを有効にする。
SET BOX SELECT	ボックス選択を行えるようにする。SELECT、REMOVE、INSERT HERE などのコマンドを BOX コマンドの同じようなコマンドと対応させることができるので、キーを再定義する必要がない。

操作手順: テキストの切り取りと貼り付け

テキストのボックスを選択して、切り取り、貼り付けるには、次の手順に従ってください。

1. EVE を起動してバッファ CITIES.DAT を作成し、次のテキストを入力する。

```
Rome   Paris   New York
London Tunis   Boston
Tokyo  Bonn    Lisbon
```

2. *Paris* という単語の文字 *P* の左にカーソルを移動する。BOX SELECT コマンドを入力する。
3. *Bonn* という単語の 2 番目の *n* の 2 文字右側、すなわちボックスの対角線上の反対側の頂点にカーソルを移動する。選択範囲のテキストがボールド体で強調表示される。BOX CUT コマンドを入力する。
テキスト・ボックスが削除される。
4. *New York* という単語で始まる列の右側にカーソルを移動する。
5. BOX PASTE コマンドを入力する。

次のように、新しい列にテキスト・ボックスがペーストされる。

```
Rome      New York  Paris
London    Boston   Tunis
Tokyo     Lisbon   Bonn
[End of file]
```

8.11.3 SET BOX SELECT コマンド

次の表は、SET BOX SELECT コマンドの一覧です。

表 8-8 SET BOX SELECT コマンド

コマンド	対応する BOX コマンド
INSERT HERE または PASTE	BOX PASTE
REMOVE または CUT	BOX CUT
RESTORE SELECTION	RESTORE BOX SELECTION
SELECT	BOX SELECT
STORE TEXT または COPY	BOX COPY

SET BOX SELECT を使用すれば、キーを再定義せずに、Select、Remove、および Insert Here の各キーを使用してボックスを選択、カット、ペーストできるようになります。

8.12 保留削除の使用方法

ユーザは保留削除を使用して、選択したテキストを削除できます。保留削除とは、新しいテキストまたはスペースを入力するか、または削除を実行する (通常は Delete キーを押す) ことによって、選択範囲を削除することをいいます。

ボックス選択を使用した場合、保留削除は BOX CUT のように働き、通常ボックスのあった場所にはスペースが埋め込まれるので、ボックスの右側にあったテキストは左に動きません。したがって、列の位置も揃ったままです。

保留削除は Insert Here バッファを使用しないので、テキストをカットしたりペーストするための別の方法になります。保留削除についての詳しい説明は、EVE オンライン・ヘルプの「Pending Delete」というトピックを参照してください。

8.12.1 保留削除による選択範囲の削除

保留削除を使用して選択範囲を削除するには、次の手順に従ってください。

1. EVE でファイルをオープンする。
2. 保留削除を使用可能にするために SET PENDING DELETE コマンドを使用する。省略時の設定は、SET NOPENDING DELETE である。

3. 削除するテキストを選択する。SELECT または BOX SELECT を使用する (SELECT ALL は使用できない)。
4. 新しいテキストを入力するか、または DELETE コマンドを入力する。

8.12.2 保留削除を使用して削除した選択範囲の復元

保留削除を使用して削除した選択範囲を元に戻す (復元する) には、次の手順に従ってください。

1. テキストを復元する位置にカーソルを置く。ボックス選択範囲を復元する場合は、ボックスの左上隅に当たる部分にカーソルを置く。
2. RESTORE SELECTION を使用する。保留削除によってボックス選択範囲を削除した場合には、RESTORE BOX SELECTION を使用する。SET BOX SELECT を使用した場合は、そのまま RESTORE SELECTION を使用できる (キーを再定義する必要はない)。

ボックスの復元は BOX PASTE のように働き、通常、既存のテキストに上書きします。SET BOX NOPAD コマンドを使用した場合、ボックス編集の効果は、そのときのバッファ・モード (ステータス・ラインに表示される挿入モードまたは上書モード) によって決まります。

- 挿入モードでは、ボックスをカットすると、ボックスの右側にあったテキストが左に“ずれて”すき間が埋まる。ボックスの右側にあったタブ文字は、スペースに変換されるので、テキストが左へ移動しても、列の位置は揃ったままである。この方法は、たとえば 4 列の表を 2 列の表に変換する場合など、表またはリストから列を削除する場合に役立つ。ボックスをペーストすると、既存のテキストが右に移動する。これは、表の途中に列を追加する場合に役立つ。
- 上書モードでは、ボックスをカットすると、ボックスのあった位置にスペースが埋め込まれるので、ボックスの右側にあったテキストは左に動かない。ボックスをペーストすると、既存のテキストが上書きされる。これは、省略時の設定値である SET BOX PAD と同じである。

保留削除を使用してボックスを削除する場合、また削除したボックスを復元する場合の動作も、バッファ・モードによって決まります。

8.13 テキストの検索と置換

EVE コマンドを使用すれば、バッファ内の特定のテキストを検索できます。ユーザは、特定のテキストの存在位置をすべて検索したり、2 行にまたがる検索文字列を検索したりできます。また、ワイルドカードを使用して検索することもできます。この節では、テキストの検索と置換について説明します。

表 8-9 に、バッファ内のテキストの検索に使用する EVE コマンドを示します。

表 8-9 バッファ内のテキスト検索用 EVE コマンド

コマンド	機能
FIND	現在のバッファから指定の文字列を検索して、見つかったら強調表示する。強調表示された文字列を検出範囲と呼ぶ。
FIND NEXT	FIND、REPLACE、WILDCARD FIND のうちいずれか 1 つのコマンドによって最後に指定された文字列を検索する。
FIND SELECTED	入力された文字列ではなく、選択された文字列を検索する。選択範囲は、2 行以上にまたがることはできない。
SET FIND CASE EXACT	大文字、小文字を区別して検索するように設定する。このコマンドは、小文字だけの検索文字列を見つけて置換する場合に役立つ。
SET FIND CASE NOEXACT	省略時の設定である。大文字、小文字を区別しないで検索するように設定する。検索文字をすべて小文字で入力したとしても、大文字、小文字に関係なく該当するすべての文字列が検索される。
SET FIND NOWHITESPACE	省略時の設定である。タブとスペースが検索文字列の指定と正確に一致し、全体が 1 行に収まる文字列を検索するように FIND コマンドと WILDCARD FIND コマンドを設定する。
SET FIND WHITESPACE	スペース、タブ、改行記号 (改行記号は 1 回のみ) を“空白”として処理するように FIND コマンドと WILDCARD FIND コマンドを設定する。2 語以上の単語からなる検索文字列を分割の仕方にかかわらず検索できるようになる。
SET WILDCARD VMS	OpenVMS の省略時の設定である。OpenVMS のワイルドカード・パターンを有効にする。
SHOW WILDCARDS	WILDCARD FIND コマンドで使用できるワイルドカードのパターンを一覧表示する。
WILDCARD FIND	ワイルドカードを使用して、テキスト・パターンを検索する。

8.13.1 テキストの検索

現在のバッファ内の特定のテキストを検索するには、FIND コマンドを使用します。省略時の設定では、E1 キー (VT200、VT300、および VT400 の各シリーズのターミナルの Find キー、VT100 シリーズ・ターミナルの PF1 キー) が FIND コマンドとして定義されています。

検索文字列がすべて小文字の場合、大文字と小文字の区別は無視され、一致するすべての文字列が検索されます。したがって、検索文字列 *the* は、*the*、*THE*、*The*、および *thE* のいずれにも一致します。検索文字列に大文字が含まれている場合は、大文字と小文字が正確に一致する文字列だけを検索します。したがって、検索文字列 *tHis* と一致するのは *tHis* だけです。次に例を示します。

1. FIND コマンドを入力する。
2. 検索したいテキスト (検索文字列と呼ぶ) を入力する。

バッファの現在の方向によって、順方向の検索か、逆方向の検索かが決まります。

検索文字列が現在の方向にはないが、逆方向にはある場合は、方向の変更を求めるプロンプトが出されます。

逆方向の検索を行う場合は、YES (Y) と入力して Enter キーを押します。こうすると、逆方向に最初に現れた検索文字列にカーソルが移動します。ただし、この場合、強調表示されたステータス・ラインの現在の方向は変わりません。

8.13.1.1 検索文字列が見つかった場合

検索文字列が見つかったと、その文字列が強調表示され、その最初の文字にカーソルが移動します。強調表示された検索文字列に対して実行できる編集コマンドの一覧については、『Extensible Versatile Editor Reference Manual』を参照してください。

強調表示を消すには、カーソルを検索文字列の外に移動するか、RESET コマンドを使用します。

その検索文字列が現れる次の位置を検索するには、Find キーを 2 回押すか、FIND NEXT コマンドを入力します。

8.13.2 大文字と小文字を区別する検索の設定

小文字の文字列の検索時に、検索文字列の大文字と小文字を正確に区別したい場合は、SET FIND CASE EXACT コマンドを入力します。以後、すべて小文字で検索文字列を入力すると、小文字だけの文字列が検索され、大文字を含む文字列は無視されます。

この設定は、FIND、REPLACE、および WILDCARD FIND の各コマンドにも適用されます。この設定は、自分のセクション・ファイルまたはコマンド・ファイルに保存しておけば、後の編集セッションで使用できます。省略時の設定は、SET FIND CASE NOEXACT です。

EVE は発音区別 (アクセント) 符号の有無を区別するので、発音区別符号がまったく同じに付けられている文字列だけを検索します。たとえば、*e*を検索するとき、*e*以外の文字 (*e*など) はどれも無視されます。

たとえば、次のコマンドは大文字と小文字を区別する検索を設定していますから、小文字だけの *digital* が見つかり、*Digital* や *DIGITAL* は無視されます。

```
Command: SET FIND CASE EXACT  
Command: FIND digital
```


操作手順: テキストの検索

既存のファイル RHYMES.DAT に対して FIND コマンドを使用するには、次の手順に従ってください。

1. RHYMES.DAT を編集するために EVE を起動する。カーソルがバッファ内の最初の行の先頭文字に位置づけられ、現在の方向が順方向に設定される。
2. Find キーを押して、*ree*という文字列を入力してから Enter キーを押す。カーソルが、*tree*という単語の文字*r*に移動し、文字列*ree*が強調表示される。
3. Find キーを 2 回押して、次に現れる文字列*ree*を検索する。カーソルが、*three*という単語の文字*r*に移動し、文字列*ree*が強調表示される。

検索文字列が見つかると、その文字列が強調表示され、選択範囲または検出範囲に対して実行できる任意のコマンド (SPELL を除く) が使用できるようになる。また、検出範囲に対しては保留削除を実行することもできない。

4. UPPERCASE WORD コマンドを入力する。

UPPERCASE WORD コマンドは、次の例に示すように、強調表示された文字列を小文字から大文字に変更する。

```
She rhymes with tree,  
also with bee,  
and this one makes thREE.  
[End of file]
```

操作手順: FIND SELECTED コマンドの使用法

特に複雑な文字列やスペリングや入力を間違えやすい文字列を検索するために FIND SELECTED を使用するには、次の手順に従ってください。

1. テキストをコピーして、バッファ内に同じテキストを 2 つ表示する。
2. 最初の行の文字列*rhymes with tree*の先頭にカーソルを移動する。
3. SELECT コマンドを入力する。
4. カーソルを移動し文字列を強調表示させることによって、テキストを選択する。選択範囲は 2 行以上にまたがることができないことに注意する。
5. FIND SELECTED コマンドを入力する。

カーソルは次に現れる文字列*rhymes with tree*に移動する。選択範囲は取り消され、検出文字列がボールド体属性で表示される。

8.13.3 ワイルドカードの使用法

テキストの検索にはワイルドカードを使用することができます。SHOW WILDCARDS コマンドを使用すれば、現在のワイルドカード設定のパターンを表示できます。

操作手順: ワイルドカードの使用

ワイルドカードを使用するには、次の手順に従ってください。

1. バッファの先頭にカーソルを位置づける。
2. コマンド WILDCARD FIND *eeを入力して、eeで終わる文字列を検索する。

```
She rhymes with tree,  
also with bee,  
and this one makes thREE.  
[End of file]
```

先頭行の *tree* という文字列の *r* にカーソルが位置づけられる。

8.13.4 空白を含めて検索する方法

SET FIND WHITESPACE コマンドおよび SET FIND NOWHITESPACE コマンドを使用すると、WILDCARD FIND コマンドと FIND コマンドが、スペース、タブ、改行記号などの単語の間の空白を処理する方法を指定できます。

SET FIND NOWHITESPACE コマンドを使用すると、1 行以内に収まる複数の単語からなる文字列を、文字列中のスペースやタブが現れるとおりに照合しながら検索できます。SET FIND NOWHITESPACE は、省略時の検索方法です。

SET FIND WHITESPACE コマンドを使用すると、WILDCARD FIND コマンドと FIND コマンドは、2 つ以上の単語からなる文字列を分割の仕方に関係なく検索できます。これによって、FIND の各コマンドは、単語の間に 1 つ以上のスペースまたはタブを含む、2 行にまたがる文字列を検索できます。

8.13.5 テキストにマークを付ける方法

MARK コマンドおよび GO TO コマンドは、大きなファイルの編集時に、特定の位置にマークを付けておいて、後で編集セッション中にその位置に戻るときに役立ちます。次の表は、MARK コマンドおよび GO TO コマンドについて説明します。

コマンド	機能
MARK	マークを現在のカーソル位置に置く。このマークは表示されない。このマークは、編集セッションの間、またはマークを変更するまで変わらない。マークは終了時に保存されない。
GO TO	カーソルを MARK コマンドによってマークされた場所に移動する。マークが別のバッファにある場合には、カーソルをそのバッファに移動して、そのバッファを現在のウィンドウに表示する。

マークを付けるには、MARK コマンドの後にラベル名を入力します。ラベル名には、英数字、区切り文字、スペース、およびタブ文字を含む 1 つ以上の印刷可能な文字が使用できます。マークの付いた位置に戻るには、GO TO コマンドの後にラベル名を入力します。

8.13.6 テキストの置換

REPLACE コマンドを使用すれば、現在のバッファ内の文字列を別の文字列に置換することができます。このコマンドは、長いファイル全体で特定の単語のスペルを間違えて入力したとき、それを全部訂正するような場合に便利です。

8.13.6.1 REPLACE コマンドと大文字と小文字の区別

REPLACE コマンドは大文字と小文字を区別します。検索文字列に大文字が含まれている場合、大文字、小文字が正確に一致する文字列が検索されます。検索文字列がすべて小文字の場合は、大文字と小文字に関係なく一致するすべての文字列が検索されます。置換文字列に大文字が含まれている場合は、その新しい文字列にそのまま正確に置換されます。置換文字列もすべて小文字の場合は、次の規則に従って文字列の置換が行われます。

- 検索文字列の先頭文字だけが大きく残り、残りは小文字である文字列が見つかったら、置換文字列も先頭文字だけが大きく置換される。
- 検索文字列のすべてが大文字である文字列が見つかったら、置換文字列もすべてが大文字になって置換される。大文字と小文字が混在している文字列が見つかったら、新しい文字列はすべて小文字で置換される。

次の表に、EVE が大文字と小文字を処理する方法を示します。

検索文字列	置換文字列	強調表示される文字列	実際に置き換えられる文字列
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

小文字の文字列だけを検索または置換する場合は、SET FIND CASE EXACT コマンドを入力します。このコマンドを入力してから、すべて小文字の検索文字列を入力すると、小文字だけの文字列が検索され、大文字を含む文字列は無視されます。この設定は、FIND、REPLACE、および WILDCASE FIND のコマンドに適用されます。

次の表は，SET FIND CASE EXACT コマンドを入力したときに，小文字だけの文字列が検索および置換される場合を示しています。

検索文字列	置換文字列	強調表示される文字列	実際に置き換えられる文字列
butter	margarine	butter	margarine

省略時の設定は，SET FIND CASE NOEXACT です。

8.13.6.2 REPLACE コマンドの応答

次の表に，REPLACE コマンドに対する応答とその機能を要約します。

応答	機能
Yes	この文字列を置換して次の文字列を検索する。省略時の設定である。Enter キーを押すだけでよい。
No	この文字列は置換しないで次の文字列を検索する。
All	該当するすべての文字列を置換する。ここ以降，反対方向で文字列が見つかった場合を除き，確認を求めるプロンプトは出されない。
Last	この文字列を置換して検索を中止する。
Quit	この文字列を置換しないで検索を中止する。

8.14 コマンド行修飾子の使用方法

EVE の起動時には，コマンド行修飾子を使用して EVE の高度な編集機能を指定することができます。文字セル更新画面を使用している場合は，ターミナルの設定によって省略時に挿入モードになるか，または上書モードになるかが決まります。

次の表に，EVE の起動時に EDIT コマンドとともに指定できる修飾子の一覧を示します。

表 8-10 EVE の起動時に指定できる EDIT コマンド行修飾子

修飾子	省略時の設定
コマンド・ファイル	/COMMAND=TPU\$COMMAND.TPU
ファイル作成	/CREATE
パッケージのデバッグ	/NODEBUG
表示モードの指定	/DISPLAY=CHARACTER_CELL
ファイルの初期化	/INITIALIZATION=EVE\$INIT.EVE
ジャーナリング	/JOURNAL
Main バッファの変更	/MODIFY
出力ファイルの指定	/OUTPUT=output-file

(次ページに続く)

表 8-10 (続き) EVE の起動時に指定できる EDIT コマンド行修飾子

修飾子	省略時の設定
読み込み専用アクセス	/NOREAD_ONLY
回復	/NORECOVER
セクション・ファイル	/SECTION=TPU\$SECTION
開始位置	/START_POSITION=(1,1)
作業ファイル	/WORK=SYSS\$SCRATCH:TPU\$WORK.TPU\$WORK

8.14.1 別の開始位置の指定

開始位置修飾子は、コマンド行に指定したバッファを開いたときに、最初にカーソルが表示される行および桁の位置を決定します。

省略時の開始位置は、1,1 – すなわち 1 行 1 桁で、バッファの左上隅に設定されています。開始位置修飾子を指定しても、編集セッション中に作成した別のバッファの初期カーソル位置は変わりません。また、バッファ・サイズが制限されることもありません。

開始位置修飾子の書式は次のとおりです。

/START_POSITION=(row[,column])

各フィールドの意味は次のとおりです。

/START_POSITION	EDIT コマンドには、/START_POSITION=修飾子を使用しなければならない。
row	EVE の起動時の初期カーソル位置の行。
column	EVE 起動時の初期カーソル位置の桁。

開始位置修飾子は、特定の行または特定の桁から編集を開始する場合に使用します。たとえば、バッチ・ログ・ファイルやエラー・メッセージによって、プログラム内の特定の行にエラーが存在することが分かっている場合や、ファイル内の標準見出しをスキップする場合は、その行を無視するように新しい開始行位置を指定すれば、プログラム・ソース・ファイルの編集開始時にカーソルが直接指定の開始行に移動します。次のコマンドは、test.com というファイルの編集開始時に、カーソルを 10 行 5 桁に表示します。

```
$ EDIT TEST.COM /START_POSITION=(10,5)
```

ファイル内の特定の行の 1 桁目から編集を開始する場合は、2 番目のパラメータ (桁) は省略できます。

8.14.2 作業ファイルの使用

作業ファイル修飾子は、非常に大きなファイルを編集するときのメモリ領域のスワップ用に使用される作業ファイルを決定します。作業ファイルとは、編集セッションごとに1つ存在し、終了時に自動的に削除される一時ファイルのことです。

省略時の作業ファイルの名前は、TPU\$WORK.TPU\$WORK です。作業ファイルは、ユーザがその位置を指定しなければ SYSS\$SCRATCH 内に作成されます。

省略時の設定と異なる作業ファイルを指定するには、次の2つの方法があります。

- 論理名 TPU\$WORK を定義する。

この方法は、SYSS\$SCRATCH 以外の場所、たとえば大容量のディスク上に作業ファイルを作成する場合に便利である。論理名定義は、LOGIN.COM ファイル内に格納できる。

- /WORK=修飾子を使用して作業ファイルを指定する。

/WORK=修飾子を指定すると、TPU\$WORK 論理名の定義は無効になる。たとえば、次のコマンドは、EVE を起動して SYSS\$SCRATCH:MYWORK.TPU\$WORK という作業ファイルを作成する。

```
$ EDIT /WORK=MYWORK
```

SYSS\$SCRATCH 以外の場所に作業ファイルを作成したい場合は、デバイス名 (ディスク名) とディレクトリを含む完全なファイル指定を使用します。作業ファイルの指定にワイルドカードは使用できません。

8.14.3 Main バッファの変更

変更修飾子は、コマンド行に指定したバッファを変更できるかどうかを決定します。ただし、編集セッション中に作成した他のバッファには影響を与えません。

省略時の設定では、バッファ内のテキストを編集することによってバッファの内容を変更できます。編集を終了すると、バッファ内容が変更されている場合は、バッファがファイルに書き込まれます。

ファイルを変更せずに内容を調べるだけの場合には、/NOMODIFY を使用します。/NOMODIFY を指定すると、カーソル移動コマンドは使用できますが、テキストは変更できません。

/MODIFY と /NOMODIFY のどちらも指定しない場合と、バッファを変更できるかどうかはアプリケーションが決定します。省略時の動作では、バッファを変更します。

/MODIFY を指定すると、/READ_ONLY や/NOWRITE の指定が無効になります。編集操作の練習をするだけで、終了時にファイルに書き込まない場合は、/READ_ONLY または/NOWRITE とともに/MODIFY を指定します。たとえば、次のコマンドは EVE を起動し、コマンド行に指定したバッファを読み込み専用 (書き込み不可) にして、変更可能にします。

```
$ EDIT /READ_ONLY /MODIFY
```

SET BUFFER コマンドを使用すれば、バッファの変更属性を設定または変更できます。

8.15 EVE の別の起動方法

EVE は 4 種類の方法で起動できます。それは、検索リストから起動する方法、ワイルドカードを使用する方法、ワイルドカード・ディレクトリ名を使用する方法、複数の入力ファイルを使用する方法です。

8.15.1 検索リストからの EVE の起動

EVE を起動する際に、検索リストを指定して編集を開始することができます。

```
$ DEFINE STAFFMEMOS HIRING.DAT,PROMOTION.LIS,SALARY.TXT  
$ EDIT STAFFMEMOS
```

次の例では、検索リスト内の最初のファイルが存在する場合、そのファイル (HIRING.DAT) がバッファ内にコピーされ、そのファイル名とファイル・タイプがバッファ名として使用されます。最初のファイルが見つからない場合は、2 番目のファイル (PROMOTION.LIS) が検索され、それも見つからなければ、その次の 3 番目のファイルという具合に検索されます。検索リスト内のすべてのファイルが存在しない場合は、空のバッファが作成され、検索リスト内の最初のファイル名である HIRING.DAT がバッファ名として使用されます。

8.15.2 ワイルドカードを使用した EVE の起動

既存のファイルを編集するために EVE を起動する場合、ワイルドカード文字であるアスタリスク(*)をファイル名やファイル・タイプの一部または全部の文字の代わりに使用することができます。EVE でワイルドカードを使用するには、DCL でワイルドカードを使用するときと同じ規則に従います。パーセント記号(%)は 1 つで 1 文字の代わりに、反復記号([...]) はディレクトリ指定の代わりに使用できます。該当するファイルが 1 つしかない場合は、そのファイルが画面に表示されます。該当するファイルが 2 つ以上ある場合は、それらのリストが表示され、より詳しいファイル指定を求めるプロンプトが出されます。該当するファイルが存在しない場合は、Main という名前のバッファが作成されます。

ワイルドカード要求に一致するファイルが2つ以上存在する場合は、表示されたリストの中から必要なファイルを選択できます。

一致するファイルが存在しない場合は、Main という名前の空のバッファが作成されます。検索リストまたはワイルドカード・ディレクトリを使用して入力ファイルを指定した場合は、\$CHOICES\$バッファは表示されずに、最初に見つかった一致ファイルが使用されます。\$CHOICES\$バッファについての詳細は、オンライン・ヘルプの「Choices Buffer」というトピックを参照してください。

次の例では、ファイル・タイプが.TXT であるすべてのファイルの一覧が表示されます。

```
$ EDIT *.TXT
```

*.TXT を指定すると、EVE はワイルドカード要求に合ったファイルをシステム・バッファの\$CHOICES\$という名前の第2ウィンドウにリストします。

8.15.3 ワイルドカード・ディレクトリを使用した EVE の起動

ディレクトリ名にワイルドカード([...])を使用して EVE を起動するには、各システムの構文を使用します。ディレクトリ名にワイルドカードを使用すると、現在のディレクトリまたは現在のディレクトリのサブディレクトリが検索対象になります。

このような検索リストとワイルドカード・ディレクトリの処理方法は、EDIT コマンドだけでなく、パラメータにファイル指定を使用する EVE コマンドにも適用されます。次の EVE コマンドは、パラメータにファイル指定を使用します。

```
@ (at sign)
GET FILE
INCLUDE FILE
OPEN
OPEN SELECTED
RECOVER BUFFER
```

次の例では、ディレクトリ・ツリーが上から下に検索され、最初に見つかった PINK.TXT という名前のファイルが使用されます。

```
$ EDIT [...]PINK.TXT
```

8.15.4 複数の入力ファイルを使用した EVE の起動

EVE の起動コマンド行には、複数の入力ファイルを指定できます。各ファイルは、コンマ (空白は入れても入れなくてもよい) で区切る必要があります。ファイル名にワイルドカード文字が含まれている場合は、2つ以上のファイルに一致する最初のワイルドカード・ファイル名について、一致ファイルが表示されます。その他のあいまいなファイル名については、警告メッセージが表示されます。

8.16 ジャーナリングと回復

バッファ・ジャーナリングは、各テキスト・バッファごとに異なるジャーナル・ファイルを作成します。

このジャーナリング方式は、EVE の省略時の設定です。バッファ・ジャーナリングは、DECwindows と文字セル・ターミナルの両方で機能します。通常は、EVE の RECOVER BUFFER コマンドを使用して、1 度に 1 つずつバッファを回復します。異なる編集セッションのバッファも回復できます。回復されるのはテキストだけで、設定値、キー定義、システム障害が生じる前のシステム・バッファ (Insert Here バッファなど) の内容は回復されません。

EVE の起動時にジャーナル機能を使用不能にするには、コマンド行に/NOJOURNAL 修飾子を指定します。この修飾子は、ファイルを編集せずに調べるだけの場合、またはデモ用のセッションに EVE を使用する場合に便利です。

OpenVMS のファイル・システムがバージョン番号を管理しているため、EVE のファイル・バックアップは使用不可能になっており、使用可能にはできません。したがって、EVE のバックアップ機能は必要ありません。

8.16.1 バッファ・ジャーナリングの使用方法

バッファ・ジャーナリングは、各テキスト・バッファごとにジャーナル・ファイルを作成します。Insert Here バッファ、DCL バッファ、\$RESTORE\$ バッファなどのシステム・バッファのバッファ・ジャーナル・ファイルは作成されません。バッファを編集すると、ジャーナル・ファイルにはテキストの削除、挿入、整形などの変更内容が記録されます。EVE を終了するか、バッファを削除すると、ジャーナル・ファイルは削除されます。編集セッション中にシステム障害が生じると、ジャーナル・ファイルはセーブされます。ただし、システム障害が生じる直前のいくつかのキーストロークは消失することがあります。

次の表に、バッファ・ジャーナリングと回復に使用する EVE コマンドをまとめます。

表 8-11 バッファ・ジャーナリングと回復に使用する EVE コマンド

コマンド	機能
RECOVER BUFFER	指定されたバッファのジャーナル・ファイルを使用してバッファを回復する。回復したいバッファまたはファイルの名前か、バッファのジャーナル・ファイルの名前を指定する。

(次ページに続く)

表 8-11 (続き) バッファ・ジャーナリングと回復に使用する EVE コマンド

コマンド	機能
RECOVER BUFFER ALL	バッファのジャーナル・ファイルが存在する場合に、それを使用して一度に 1 つずつすべてのテキスト・バッファを回復する。
SET JOURNALING	指定されたバッファでバッファ変更ジャーナリング機能が実行されるようにする。
SET JOURNALING ALL	すべてのバッファでバッファ・ジャーナリング機能が実行されるようにする。省略時の設定である。
SET NOJOURNALING	指定されたバッファでバッファ変更ジャーナリング機能が実行されないようにする。
SET NOJOURNALING ALL	すべてのバッファでバッファ・ジャーナリング機能が実行されないようにする。

バッファ・ジャーナル・ファイルは、論理名 TPU\$JOURNAL によって定義されたディレクトリ内に作成されます。省略時のディレクトリは通常、ログイン時の最上位ディレクトリである SYSSCRATCH です。論理名 TPU\$JOURNAL を再定義すれば、異なるディレクトリにジャーナル・ファイルを作成できます。たとえば、次のコマンドは[USER.JOURNAL]という名前のサブディレクトリを作成し、TPU\$JOURNAL をこのサブディレクトリに再定義しています。

```
$ CREATE/DIRECTORY [USER.JOURNAL]
$ DEFINE TPU$JOURNAL [USER.JOURNAL]
```

論理名の定義は、LOGIN.COM ファイル内に格納できます。

バッファ・ジャーナル・ファイルは、非常に大きなファイルになることがあります。編集時のテキスト・ファイルよりも大きくなるかもしれません。バッファ・ジャーナル・ファイルのサイズはかなり大きくなる可能性があること、またジャーナル・ファイルは各テキスト・バッファごとに存在することを考慮して、TPU\$JOURNAL は、SYSSCRATCH ではなく、大容量ディスク上のディレクトリまたはサブディレクトリに再定義することをおすすめします。

バッファ変更ジャーナル名の命名

バッファ・ジャーナル・ファイル名は、編集時のファイルまたはバッファの名前から取られ、オペレーティング・システムの省略時のファイル・タイプがつけられます。現在のバッファのジャーナル・ファイル名を確認するには、EVE プロンプトに SHOW コマンドを入力します。SHOW コマンドは、入力ファイル、出力ファイル、ジャーナル・ファイル、および現在のバッファに関するその他の情報を表示します。

表 8-12 に、バッファ・ジャーナル・ファイル名を示します。

表 8-12 バッファ・ジャーナル・ファイル名

テキスト・バッファ名	バッファ・ジャーナル・ファイル
JABBER.TXT	JABBER.TXT.TPU\$JOURNAL
GUMBO_RECIPE.RNO	GUMBO_RECIPE_RNO.TPU\$JOURNAL
MAIN	MAIN.TPU\$JOURNAL
LATEST NEWS	LATEST_NEWS.TPU\$JOURNAL

バッファ・ジャーナリングを使用した回復方法
バッファ・ジャーナル・ファイルを使用して編集内容を回復するには、次の 2 つの方法があります。

- EVE の起動時に EDIT コマンド行に/RECOVER 修飾子を指定する。
- EVE 内で RECOVER BUFFER コマンドを使用する。

次の例では、JABBER.TXT という名前のファイルを編集中にシステム障害が発生して、編集セッションが中断しています。その後、システム回復機能を使用して編集内容を回復しています。

```
$ EDIT JABBER.TXT
.
.
.
*** system failure ***
.
.
.
$ EDIT JABBER.TXT/RECOVER
```

RECOVER BUFFER コマンドの使用方法
バッファ回復コマンドを使用するには、次のようにします。

手順	操作
1	EVE を起動し、次のコマンドを入力してテキストを回復する。

Command: RECOVER BUFFER **file-name.txt**

バッファ・ジャーナル・ファイルが使用できる場合は、次の情報が表示され、そのバッファを回復するかどうか尋ねられる。

バッファの名前
バッファの元の入力ファイル (あれば)
バッファの出力ファイル (あれば)
回復のためのソース・ファイル (あれば)
編集セッションの開始日時
ジャーナル・ファイルの作成日時

手順	操作
2	<p>バッファを回復したい場合には、Enter キーを押す。</p> <p>バッファを回復したくない場合は、No と入力してから Enter キーを押す。回復しようとするソース・ファイルを削除したり、名前を変更したりすると、回復は失敗する。回復するソース・ファイルは、バッファに最初に読み込まれたファイルか、システム障害が発生する直前に書き出されたファイルである。</p> <p>回復しようとするバッファが存在すると (通常は Main バッファ)、最初にそのバッファが削除されてから回復が行われる。回復しようとするバッファに変更が加えられている場合には、回復する前にバッファを削除するかどうか確認するプロンプトが出される。</p>

ファイル名が分からないときの回復

バッファ名やジャーナル・ファイル名が分からないときには、次のようにワイルドカード文字であるアスタリスク(*)を指定します。

Command: RECOVER BUFFER *

この場合、使用可能なすべてのジャーナル・ファイルのリストが表示されるため、そこからいずれかを選択します。このリストは、2 番目のウィンドウの\$CHOICES\$という EVE システム・バッファに現れます。このバッファについての詳細は、EVE オンライン・ヘルプの「Choices Buffer」というトピックを参照してください。

すべてのバッファの回復

すべてのテキスト・バッファを 1 つずつ回復するには、RECOVER BUFFER ALL コマンドを使用します。バッファ・ジャーナル・ファイルが使用可能な各テキスト・バッファについて回復が行われます。この RECOVER BUFFER ALL コマンドは、バッファ名やジャーナル・ファイル名を何度も指定しながら RECOVER BUFFER コマンドを繰り返す場合と同じ結果になります。各テキスト・バッファごとに、バッファ名、バッファに関連付けられているファイル名、ジャーナル・ファイルが作成された日時などの情報が表示されます。そして、次のうちの 1 つを選択するよう求めるプロンプトが出されます。

応答	機能
Yes	バッファを回復してから、次のバッファが存在する場合には、そのバッファを回復するかどうか聞いてくる。省略時の設定である。Enter キーを押すだけでよい。
No	回復を行わない。回復する別のバッファがある場合は、そのバッファについて聞いてくる。
Quit	取り消し。バッファを回復しないで回復操作を中止する。

バッファ・ジャーナリングを使用不可能にする方法

特定のバッファのバッファ・ジャーナリングを使用不能にするには、SET NOJOURNALING コマンドを使用します。すべてのバッファのバッファ・ジャーナリングを使用不能にするには、SET NOJOURNALING ALL コマンドを使用します。

バッファ・ジャーナリングを使用可能にする方法
バッファ・ジャーナリングが使用不能になっている場合は、SET JOURNALING コマンドを使用してジャーナリングを使用可能にできます。次のコマンドは、JABBER.TXT という名前のバッファのジャーナリングを使用可能にします。

Command: SET JOURNALING JABBER.TXT

ジャーナリングを使用せずに EVE を起動した後、編集セッション中にバッファ・ジャーナリングを使用可能にする場合は、SET JOURNALING ALL コマンド (省略時の設定) を使用します。

バッファに変更が加えられている場合は、バッファ・ジャーナリングを使用可能にできません。その場合は、次のメッセージが表示されます。

Command: SET JOURNALING MEMO.TXT
Buffer MEMO.TXT is not safe for journaling

このような場合は、まず WRITE FILE コマンドまたは SAVE FILE コマンドを使用してバッファの内容を書いて (セーブして) から、ジャーナリングを使用可能にします。

8.17 EVE の書式化コマンド

EVE には、マージン、タブ、ワード・ラップを設定するテキストを書式化するコマンドがあります。この機能には、行のセンタリング、テキストからの余分な空白の削除、改ページの挿入などがあります。

次の表に、EVE の編集キーとその機能を説明します。

表 8-13 EVE の編集キーとその機能

キーまたは キー・シーケンス	機能
Return または Ctrl/M	キャリッジ・リターンを現在位置に挿入して、新しい行を開始するか、コマンド入力を終了する。VT200、VT300、および VT400 の各シリーズのターミナルでは、Enter キーも Return として定義されている。
Tab または Ctrl/I	タブ・モードに応じた現在位置と、現在設定されているタブ・ストップにタブ文字を挿入する。
Ctrl/L	現在位置に改ページ文字を挿入して、新しいページの開始位置を示す。改ページは小さな F 2 つ (F_2) で表され、常にそれだけで 1 行を占有する。INSERT PAGE BREAK と同じ。

次の表に、EVE のテキスト書式化コマンドの一覧を示しその機能を説明します。

表 8-14 EVE のテキスト書式化コマンド

コマンド	機能
CAPITALIZE WORD	単語の先頭文字を大文字にし、残りの文字を小文字に変更する。選択範囲、ボックス、単語に対して機能する。
CENTER LINE	現在の行を左右のマージンの中央にセンタリングする。カーソルは行とともに移動し、行が移動しても同じ文字の上に留まる。
CONVERT TABS	ボックス、選択範囲、全バッファ内のタブ文字を適切な数のスペースに変換する。
FILL	バッファのマージンに応じて、現在のパラグラフ、選択範囲、またはボックス内の書式を変更して、1 行に収まる単語数を最大にする。選択範囲または検出範囲に対して実行するとき、FILL コマンドまたは FILL RANGE コマンドは、改ページ記号、DIGITAL Standard Runoff (DSR) コマンド、または DOCUMENT タグで始まる行の書式は変更せずに、範囲内の残りの行の書式を変更する。範囲に FILL コマンドを使用しても、空白行は削除されない。選択範囲についての詳細は、第 8.9 節を参照。
FILL PARAGRAPH	バッファに設定されているマージンに応じて、カーソルが現在あるパラグラフの書式を変更する。パラグラフの場合、FILL コマンドは、改ページ記号、DSR コマンド、または DOCUMENT タグで始まる行の書式は変更せずに、パラグラフ内の残りの行の書式を変更する。
FILL RANGE	現在のマージン設定値に応じて、選択範囲またはボックスの書式を変更する。選択範囲または検出範囲の場合には、FILL コマンドまたは FILL RANGE コマンドは、改ページ記号、DSR コマンド、または DOCUMENT タグで始まる行の書式は変更せずに、範囲内の残りの行の書式を変更する。範囲に FILL RANGE コマンドを使用しても、空白行は削除されない。
INSERT PAGE BREAK	現在の位置に改ページ文字を挿入して、新しいページの始まりを示す。改ページは小さな F 2 つ (F_F) で表され、常にそれだけで 1 行を占有する。省略時の設定では、Ctrl/L が INSERT PAGE BREAK として定義されている。
LOWERCASE WORD	現在の単語、選択範囲、またはボックスを小文字に変更する。
PAGINATE	54 行ごとに“ソフト”改ページを 1 つ挿入する。ソフト改ページは、後ろに空文字が続く改ページ文字 (F_{NL}) として表示される。PAGINATE コマンドを入力すると、直前の改ページから次の 54 行の中で改ページの有無がチェックされる。この 54 行の中でソフト改ページが見つかった場合は、それを削除し、54 行下に移動してソフト改ページを挿入する。カーソルは次の行に置かれる。ソフト改ページは、それだけで 1 行を占有する。54 行の中にハード改ページ (改ページのみ) が見つかった場合、ハード改ページの後の行で停止する。このとき、必要であればこの改ページを削除できる。
SET LEFT MARGIN	現在のバッファの左マージンを設定する。左マージンは 0 より大きく、右マージンより小さくなくてはならない。省略時の設定では、左マージンは 1 (左端の桁である)。

(次ページに続く)

表 8-14 (続き) EVE のテキスト書式化コマンド

コマンド	機能
SET RIGHT MARGIN	現在のバッファの右マージンを設定する。右マージンは、左マージンより大きい値でなければならない。省略時の設定では、右マージンは画面幅より 1 桁だけ小さい。通常の幅は 80 なので、省略時の右マージンは 79 になる。
SET PARAGRAPH INDENT	作成または書式変更したパラグラフの最初の行に追加する、または最初の行から差し引くスペース数を指定する。省略時の値は 0(インデントなし) である。
SET TABS AT	指定された桁にタブ・ストップを設定する。桁番号は昇順に設定し、スペースで区切る。省略時の設定では、タブ・ストップは 8 桁ごとに設定されている。ターミナルのハードウェア・タブ設定値には影響を及ぼさない。
SET TABS EVERY	指定された間隔でタブ・ストップを設定する。省略時の設定では、タブ・ストップは 8 桁ごとに設定されている。ターミナルのハードウェア・タブ設定値には影響を及ぼさない。
SET TABS INSERT	省略時の設定である。Tab キーを押すと、現在の桁にタブ文字が挿入されるようにタブ・モードを変更する。カーソルとテキストは次のタブ・ストップに移動する。
SET TABS MOVEMENT	Tab キーがカーソル移動キーになるようにタブ・モードを変更する。Tab キーを押すと、カーソルは次のタブ・ストップに移動するが、タブ文字は挿入されない。
SET TABS SPACES	Tab キーを押したときに、タブ文字ではなく、該当する数のスペースが挿入されるようにタブ・モードを変更する。以前から存在していたタブ文字は影響を受けない。
SET TABS INVISIBLE	省略時の設定値である。タブ文字は画面上では見えず、空白となる。
SET TABS VISIBLE	タブ文字を画面上で見えるようにし、 H_T (水平タブ) として表示する。
SET NOWRAP	バッファの右マージンで単語を自動改行できないようにする。新しい行を開始するときには、Enter キーを押すか、FILL コマンドを使用しなくてはならない。
SET WRAP	省略時の設定値である。バッファの右マージンで単語が自動改行されるようにする。Enter キーを押したり FILL コマンドを使用しなくても、新しい行が開始される。
UPPERCASE WORD	現在の単語、選択範囲、ボックスを大文字に変更する。

8.18 バッファの使用方法

バッファは、編集セッションの間だけ存在する記憶領域です。既存のファイルを編集する場合は、そのファイルの内容がバッファに読み込まれます。強調表示されたステータス・ラインには、バッファ名、編集状態 (読み込み専用または書き込み)、編集モード (挿入または上書)、方向 (順方向または逆方向) が表示されます。

次の表に、バッファを作成、操作、および削除するために使用する EVE コマンドを説明します。

表 8-15 バッファを操作するために使用する EVE コマンド

コマンド	機能
BUFFER	指定されたバッファを現在のウィンドウに置いて、カーソルをそのバッファの中で最後に置かれていた場所に移動する。指定されたバッファが存在しない場合は、新しいバッファを作成する。
DELETE BUFFER	名前で指定したバッファを削除する。
GET FILE または OPEN	必要なら新しいバッファを作成して、指定されたファイルを現在のウィンドウに表示する。ファイルが存在する場合は、そのファイルを現在のウィンドウ内の新しいバッファにコピーする。ファイルが存在しない場合は、新しい空のバッファを作成し、指定されたファイル名とファイル・タイプをバッファ名にする。同じ名前のバッファが既に存在する場合は、異なる名前の入力及要求される。
GO TO	カーソルを MARK コマンドによってラベルが付けられた場所に移動する。ラベルが付いた場所が別のバッファにある場合は、カーソルをそのバッファに移動して、そのバッファを現在のウィンドウに表示する。編集セッションで複数のバッファを使用する方法については、第 8.18.5 項を参照。
INCLUDE FILE	指定されたファイルの内容を現在のバッファのカーソル位置の上にある行に挿入する。このコマンドはファイルをマージする場合に便利である。
NEW	Main という名前の新しいバッファを作成して現在の EVE ウィンドウに表示する。Main というバッファがすでに存在する場合には、新しいバッファ名の入力を求めるプロンプトが出される。
NEXT BUFFER	次のバッファがあれば、それを現在のウィンドウに表示し、カーソルをそのバッファの中で最後に置かれていた場所に移動する。このコマンドを使用すれば、バッファ名を入力しなくてもバッファ間を移動できる。
OPEN SELECTED	選択または検索された名前を持つファイルをオープンする。このコマンドは、GET FILE コマンドまたは OPEN コマンドと同じであるが、ファイル名を入力する必要がない。
REMOVE または CUT	Buffer List バッファにいる場合は、DELETE BUFFER と同じ。バッファ名を入力せずにバッファを削除するには、次のように REMOVE コマンドを使用する。SHOW BUFFERS コマンドを入力して (Buffer List バッファに移動する)、削除するバッファ名にカーソルを移動してから、REMOVE コマンドを入力する。
SAVE FILE	編集セッションを終了せずに、現在のバッファの内容をバッファに関連付けられているファイルに書き込む。SAVE FILE コマンドでファイル名を指定しないと、出力ファイル指定を求めるプロンプトが出される。WRITE FILE と似ている。
SAVE FILE AS	編集セッションを終了せずに、現在のバッファの内容を指定のファイルに書き込む。たとえば、FIRST.DAT というファイルを編集の場合、それを SECOND.TXT としてセーブできる。このコマンドはバッファの名前は変更しない。ただし、バッファを指定のファイルに関連付けるので、これ以降に SAVE FILE コマンド、WRITE FILE コマンド、または EXIT コマンドを使用すると、バッファはその指定のファイルに書き込まれる。SAVE FILE AS コマンドには出力ファイルを指定する必要がある。

(次ページに続く)

表 8-15 (続き) バッファを操作するために使用する EVE コマンド

コマンド	機能
SELECT または RETURN	Buffer List バッファにいる場合は、指定されたバッファを選択する。バッファ名を入力せずにバッファを選択するには、次のように SELECT コマンドを使用する。SHOW BUFFERS コマンドを入力し、選択するバッファ名にカーソルを移動してから、SELECT コマンドを入力する。
SET BUFFER	バッファの編集状態、すなわちバッファを変更可能にするか、また EVE の終了時にバッファをファイルに書き込むかどうかを指定できるようにする。
SHOW	編集セッション中に作成されたバッファについての情報を表示する。編集セッション時に 2 つ以上のバッファが処理されている場合には、現在編集中のバッファについての情報を表示する。他のバッファについては、Do キーを押す。編集を再開するには、それ以外のキーを押す。
SHOW BUFFERS	編集セッション中に作成されたバッファのリストを表示する。リスト内でカーソルを移動して、特定のバッファを指定してから Select キーを押せば、そのバッファを表示できる。
SHOW DEFAULTS BUFFER	マージン、タブ・ストップ、方向、モード、最大行数などの EVE システム・バッファ \$DEFAULTSS についての情報を表示する。これらは、新しいバッファの作成時に使用される省略時の設定である。
SHOW SYSTEM BUFFERS	Message バッファ、Help バッファ、Insert Here バッファ、\$RESTORES バッファなどの EVE によって作成されたシステム・バッファのリストを表示する。リスト内でカーソルを移動して、特定のバッファを指定してから Select キーを押せば、そのバッファを表示できる。
WRITE FILE	編集セッションを終了せずに、現在のバッファの内容をバッファに関連付けられているファイルか、またはコマンド行に指定されたファイルに書き込む。現在のバッファに関連付けられているファイル指定が存在しない場合は、出力ファイル指定を求めるプロンプトが出される。SAVE FILE と似ている。

8.18.1 バッファ情報の取得

現在のバッファについての詳しい情報を表示するには、SHOW コマンドを入力します。SHOW コマンドは、次の情報以外に、バッファが変更されたかどうかを示します。

- バッファの名前
- 入力ファイル、出力ファイル、およびバッファ・ジャーナル・ファイルの名前
- 現在のモードと方向
- 行数
- マージンと画面幅の設定
- パラグラフ・インデント
- WPS ワード・ラップ
- ラップ・インデント

- タブ・ストップ

編集セッション中に2つ以上のバッファが処理されている場合は、現在のバッファ以外のバッファの情報を表示するには、Do キーを押すよう求めるプロンプトが出されます。

8.18.2 バッファの削除

バッファを削除するには、DELETE BUFFER コマンドを入力し、削除するバッファの名前を指定します。バッファが空であるか変更されていない場合は、そのバッファは削除されます。しかし、バッファが変更されている場合は、選択を求めるプロンプトが出されます。バッファ名には、完全な名前を指定しなければなりません。省略はできません。削除しようとするバッファを表示中の場合は、そのバッファが、編集セッション中で最も古いバッファに置き換えられます。

次の表に、入力できる選択肢を示します。

キーワード	機能
DELETE_ONLY	指定されたバッファを削除する。
WRITE_FIRST	指定されたバッファを書き出して (セーブして) から、削除する。
QUIT	省略時の選択である。バッファを削除しない。

たとえば、次のコマンドは、変更されたバッファ MYFILE.TXT の削除を要求します。

```
Command: DELETE BUFFER MYFILE.TXT
That's a modified buffer. Type delete_only, write_first, or quit:
```

8.18.3 バッファ状態の変更

バッファの編集状態、すなわちバッファが変更可能かどうか、また終了時にバッファをファイルに書き込むかどうかを変更するには、SET BUFFER コマンドを使用します。

各コマンドには、次のいずれかの SET BUFFER キーワードを指定できます。

キーワード	機能
MODIFIABLE	省略時の設定である。バッファを変更できるようにする。また、前のバッファ・モード (挿入または上書) を復元する。
READ_ONLY	バッファが変更された場合でも、終了時にバッファをセーブしない (書き出さない)。WRITE の反対である。バッファを変更不可能にすることもできる。ただし、変更可能にできる。
UNMODIFIABLE	バッファを変更できなくする。バッファ・モード (挿入または上書) を無効にする。

キーワード	機能
WRITE	省略時の設定である。バッファが変更された場合に、バッファをセーブする (書き出す)。READ_ONLY の反対である。バッファが読み込み専用または変更不可能の場合は、SET BUFFER WRITE コマンドによって変更可能にし、前のモード (挿入または上書) を復元する。

省略時の設定では、バッファ状態は MODIFIABLE かつ WRITE に設定されており、バッファの内容は変更可能で、変更されたバッファはファイルにセーブされます。

バッファ状態を変更してバッファの内容を誤って変更することがないようにするには、次のコマンドを使用してバッファを READ_ONLY (変更不可能) に設定します。

Command: SET BUFFER READ_ONLY

バッファ状態を変更して一時格納領域“スクラッチパッド”として使用するには、次のコマンドを使用してバッファを READ_ONLY と MODIFIABLE に設定します。

Command: SET BUFFER READ_ONLY
Command: SET BUFFER MODIFIABLE

上のコマンドによって、バッファは編集できるようになりますが、終了時にセーブされません。

8.18.4 メッセージ・バッファの表示方法

メッセージ・ウィンドウは、画面の最下部に表示され、編集セッション中にエラー・メッセージおよび情報メッセージを通知します。メッセージ・ウィンドウには、Messages バッファ内の最新のメッセージが表示されます。

これらのメッセージは、BUFFER コマンドを使用して表示できます。Messages バッファの内容を表示するには、Do キーを押してから BUFFER MESSAGES コマンドを入力します。編集していたバッファに戻るには、Do キーを押して BUFFER コマンドの後に戻り先のバッファ名を入力します。

SHOW BUFFERS コマンドを入力すると、作成したバッファを表示できます。そこで、Select キーを押せばバッファを選択できます。

8.18.5 複数バッファの編集

2 つ以上のファイルを編集する場合、または複数のテキスト・ブロックを操作する一時格納領域が必要な場合には、複数のバッファを使用できます。新しいバッファを作成するには、GET FILE、OPEN、OPEN SELECTED、および BUFFER の各コマンドのうちいずれか 1 つを使用できます。

GET FILE コマンドの使用方法

新しいバッファを作成するには、GET FILE、OPEN、OPEN SELECTED、および BUFFER の各コマンドのうちいずれか 1 つを使用できます。既存のファイルを使用して新しいバッファを作成するには、GET FILE (または OPEN) コマンドの後に、バッファにコピーしたいファイルの名前を入力します。アスタリスク(*)のワイルドカード文字を使用すれば、ファイル名およびファイル・タイプの全部または一部を代用できます。また、パーセント記号(%)のワイルドカード文字はファイル名およびファイル・タイプ内の 1 文字の代わりに、反復記号([...])のワイルドカードはディレクトリ指定の代わりに使用できます。

OPEN SELECTED コマンドの使用方法

次のように OPEN SELECTED コマンドを使用して、新しいバッファを作成することもできます。

1. オープンするファイルの名前にカーソルを移動する。
2. OPEN SELECTED コマンドを入力する。

BUFFER コマンドの使用方法

現在の EVE ウィンドウ内に特定のバッファを表示するには、BUFFER コマンドの後に、現在のウィンドウ内に表示したいバッファの名前を入力します。バッファ名にワイルドカード文字を使用してはなりません。アスタリスク(*)とパーセント記号(%)は、バッファ名の中リテラル文字と見なされます。指定されたバッファが存在しない場合は、新しいバッファが作成されます。

指定されたファイルが既に存在する場合は、そのファイルの内容が新しいバッファに読み込まれ、現在のウィンドウにバッファが表示されます。ワイルドカードを使用したファイル指定に一致するファイルが 2 つ以上存在する場合は、\$CHOICES\$バッファに選択リストが表示され、より詳しいファイル指定を求めるプロンプトが出されます。検索リストまたは反復記号([...])のワイルドカードを使用した場合は、最初に一致したファイルがオープンされます。一致するファイルが存在しない場合は、空のバッファが作成され、現在のウィンドウに表示されます。

現在のウィンドウ内のバッファを変更するには、Do キーを押して、BUFFER コマンドの後に、表示したいバッファの名前を入力してから、Enter キーを押します。バッファ名を忘れた場合は、SHOW BUFFERS コマンドを入力して、編集セッション内で処理中のバッファの名前を表示し、Select キーを使用してバッファを指定します。

8.18.6 EVE へのファイルの読み込み

EVE バッファにファイルを読み込むには、次の 4 つの方法があります。

- ファイル指定を使用して EVE を起動する。
- INCLUDE FILE コマンドと組み込みファイルの名前を入力する。

ファイルの全内容がカーソル行の直前のバッファに読み込まれます。INCLUDE FILE コマンドを使用しても、ステータス・ラインのバッファの名前は変わりません。

- GET FILE コマンドまたは OPEN コマンドと使用ファイルの名前を入力する。
どちらのコマンドを入力しても、新しいバッファが作成され既存のファイルの内容がそのバッファに読み込まれます。ステータス・ラインのバッファの名前は、GET FILE コマンドまたは OPEN コマンドで指定するファイル名と同じです。第 8.18.5 項を参照してください。
- ファイル名を選択するか検索してから、OPEN SELECTED コマンドを入力する。

8.18.7 EVE からのファイルの書き込み

現在のバッファの内容をファイルに書き込むには、WRITE FILE コマンドを入力します。WRITE FILE コマンドには、ファイル指定を使用できます。ファイル指定を使用しないと、入力ファイル指定がファイルの書き込みに使用されます。BUFFER コマンドまたは NEW コマンドで現在のバッファを作成した場合には、ファイルの書き込み先のファイル指定を求めるプロンプトが表示されます。

次の例は、バッファに関連付けられている出力ファイルを使用して、ファイルへバッファを書く方法を示しています。

```
Command: GET FILE RHYMES.DAT
.
.
.
Command: WRITE FILE
3 lines written to WORKDISK:[USER]RHYMES.DAT;2
```

8.18.8 ウィンドウの使用方法

EVE の編集セッション中は、編集中のバッファが画面上のウィンドウに表示されます。強調表示されたステータス・ラインがウィンドウの最下部に現れ、バッファ名、現在の編集モード、および現在のバッファ方向が表示されます。

ターミナル画面上に 2 つ以上のウィンドウを同時に表示することもできます。たとえば、画面上に 2 つのウィンドウをオープンして、同じバッファ内の異なる部分を表示および編集できます。

次の表では、ウィンドウの作成と操作に使用する EVE のキーについて説明します。

表 8-16 ウィンドウの作成と操作に使用する EVE のキー

キーまたは キー・シーケンス	ウィンドウ環境での機能
GOLD Next Screen	カーソルを次の (もう一方の) ウィンドウに移動する。NEXT WINDOW コマンドと同じ。
GOLD Prev Screen	カーソルを前の (もう一方の) ウィンドウに移動する。PREVIOUS WINDOW コマンドと同じ。

次の表は、ウィンドウを作成および操作するために使用される EVE コマンドを示しています。

表 8-17 EVE ウィンドウ・コマンド

コマンド	ウィンドウ環境での機能
DELETE WINDOW	2 つ以上のウィンドウを使用している場合に、現在のウィンドウを削除する。
ENLARGE WINDOW	現在のウィンドウを指定された行数だけ拡大する。たとえば、ENLARGE WINDOW 5 はウィンドウを 5 行だけ拡大する。このとき、隣接するウィンドウは拡大された行数分だけ縮小される。
NEXT WINDOW または OTHER WINDOW	カーソルを次の (もう一方の) ウィンドウに移動する。
ONE WINDOW	現在のウィンドウを単独の大きなウィンドウに戻す。他のウィンドウはすべて画面から削除される。ただし、それらのウィンドウに関連するバッファは削除されない。
PREVIOUS WINDOW	カーソルを前の (もう一方の) ウィンドウに移動する。
SET WIDTH	画面上に表示される行の幅を設定する。幅は正の整数で指定する。省略時の設定では、ターミナルごとに設定されている画面幅 (通常は 80 桁) が使用される。80 より大きい幅が設定されると、現在の編集セッションでターミナルが 132 桁モードに設定される。EVE を終了すると、ターミナルは省略時の設定値に戻る。幅を設定すると、すべてのウィンドウのテキスト表示が変更される。
SHIFT LEFT	現在のウィンドウを指定された桁数だけ左に移動する。SHIFT LEFT コマンドは、SHIFT RIGHT コマンドの効果を逆に戻すときにだけ使用する。
SHIFT RIGHT	現在のウィンドウを指定された桁数だけ右に移動して、現在ターミナルの画面に現れていない文字桁を表示できるようにする。
SHRINK WINDOW	現在のウィンドウを指定された行数だけ縮小する。たとえば、SHRINK WINDOW 5 は、ウィンドウを 5 行だけ縮小する。このとき、隣接するウィンドウは縮小された行数だけ拡大される。
SPLIT WINDOW	現在のウィンドウを分割して、2 つの小さいウィンドウを表示する。このコマンドとともに数を指定すれば、ウィンドウを 3 つ以上に分割できる。たとえば、SPLIT WINDOW 3 は、ウィンドウを 3 つのウィンドウに分割する。

(次ページに続く)

表 8-17 (続き) EVE ウィンドウ・コマンド

コマンド	ウィンドウ環境での機能
TWO WINDOWS	SPLIT WINDOW 2 コマンドと同じ。

8.18.9 単一バッファの 2 つのセクションの表示

1 つのファイル内の 2 つの異なる部分を同時に表示するには、SPLIT WINDOW コマンドを使用します。SPLIT WINDOW コマンドは、画面を分割して、2 つの同じウィンドウを作成します。カーソルはバッファ内の同じ場所に置かれますが、下のウィンドウにしか現れません。どちらのウィンドウのステータス・ラインにも、同じバッファ名が表示されます。

大きなファイルの 2 つの異なる部分を同時に表示すれば、ファイル内で効率的にテキストを移動することができます。ファイル内のある部分のテキストを選択および削除して、もう一方の部分に挿入できます。カーソルをもう一方のウィンドウに移動するには、NEXT WINDOW コマンドを使用します。

2 番目のウィンドウを画面から削除して、現在のウィンドウを編集領域いっぱいに拡大するには、Do キーを押して ONE WINDOW コマンドを入力してから、Enter キーを押します。

8.18.10 2 つのバッファの編集

次の手順は、異なるファイルを含む 2 つのバッファの編集方法について説明しています。

手順	操作
1	SPLIT WINDOW コマンドを入力して、画面上に 2 つのウィンドウを作成する。 画面が 2 つに分割され、2 つのウィンドウが作成される。カーソルはバッファ内の同じ場所に置かれるが、下のウィンドウにしか現れない。各ウィンドウの強調表示されたステータス・ライン内には、同じバッファ名が表示される。
2	GET FILE、OPEN、または OPEN SELECTED のうちいずれかのコマンドを使用して、2 番目のファイルを現在のウィンドウに表示する。 今まで編集セッション中に作成したバッファを現在のウィンドウに表示するには、BUFFER コマンドと表示バッファの名前を入力する。 ターミナルの画面には、2 つの異なるバッファが表示される。一方のバッファ内のテキストを選択および削除して、もう一方のバッファに挿入できる。もう一方のウィンドウにカーソルを移動するには、NEXT WINDOW コマンドを入力する。

8.19 サブプロセスの作成

サブプロセスを作成すれば、編集セッションを終了せずに、EVE 編集セッションと DCL コマンド・レベルとの間を切り替えることができます。サブプロセスを作成するには、SPAWN コマンドを入力します。SPAWN コマンドは、現在の編集セッションを中断して新しいサブプロセスにターミナルを接続します。ターミナル画面には、DCL プロンプト (\$) が現れます。

8.19.1 SPAWN コマンドの使用

サブプロセスは主に、Mail ユーティリティを起動したり、画面指向のプログラムを実行するために生成しますが、任意の OpenVMS ユーティリティを起動したり、DCL コマンドを実行することもできます。

編集セッションに戻るには、DCL コマンドの LOGOUT を入力して、サブプロセスからログアウトします。編集セッションが再開され、カーソルは、サブプロセスを生成する前に置かれていた位置に現れます。SPAWN コマンドのパラメータに DCL コマンドを指定すれば、特定のサブプロセスを生成できます。

次の例では、Mail ユーティリティが EVE から生成されます。

[End of file]

```
Buffer: MAIN                               | Write | Insert | Forward
Command: SPAWN MAIL
```

画面には、Mail ユーティリティのプロンプト (MAIL>) が現れます。Mail を終了すると、自動的にサブプロセスからログアウトされ、編集セッションが再開されます。

8.19.1.1 DCL から EVE へのサブプロセスの生成

DCL を使用するためにサブプロセスを生成する代わりに、EVE の編集セッションを実行するためにサブプロセスを生成して、親の DCL プロセスにアタッチすれば、DCL コマンドとユーティリティを使用できます。

DCL コマンド・レベルに戻りたいときは、EVE コマンドの ATTACH を使用して親プロセスに戻ります。

編集セッションを再開するには、DCL コマンドの ATTACH にサブプロセス名 (SMITH_1) を指定して実行し、編集サブプロセスに再接続します。編集セッションが再開され、カーソルが親プロセスにアタッチする前に置かれていた位置に現れます。次に例を示します。

次の例では、DCL コマンド SPAWN を使用してサブプロセスが生成されます。SPAWN コマンドは SMITH_1 というサブプロセスを生成します。サブプロセス・レベルで EVE が起動され、編集セッションが実行されます。編集セッションの最後に ATTACH コマンドが入力され、DCL に戻ります。その後、編集セッションを再

開するには、サブプロセスSMITH_1をプロセス名として使用して、DCL コマンド ATTACH を入力します。

```
$ 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
```

ファイルのソートとマージ

本章では、OpenVMS の Sort/Merge ユーティリティ (SORT/MERGE) の使用方法について説明します。Sort/Merge ユーティリティは、次の 2 種類の作業を実行します。

- 選択したフィールドに従い、1 つまたは複数の入力ファイルからレコードを読み込んでソートし、順序変更した出力ファイルを生成する。
- あらかじめ同じキー・フィールドでソートした入力ファイルを、最高で 10 ファイルをマージし、1 個の出力ファイルを生成する (高性能 Sort/Merge ユーティリティの場合、最高で 12 個の入力ファイルがサポートされている)。

Alpha システムでは、高性能 Sort/Merge ユーティリティを選択することもできます。このユーティリティでは、Alpha アーキテクチャを活用することにより、ほとんどの Sort 操作および Merge 操作で性能を向上させることができます。詳細は第 9.1 節を参照してください。

本章では、次のことを説明します。

- 高性能 Sort/Merge
- ファイルのソート
- 照合順序の指定
- バッチ・ジョブとしてのソートの実行
- ファイルのマージ
- ターミナルからのレコードの入力
- Sort/Merge 指定ファイルの使用方法
- ソートまたはマージ操作の最適化
- Sort/Merge の修飾子の要約

その他の情報については、次のマニュアルを参照してください。

- この章で使用するコマンドについては、『OpenVMS DCL デクショナリ』を参照してください。
- Sort/Merge ユーティリティ使用時のシステム管理者による効率化については、『OpenVMS システム管理者マニュアル』を参照してください。

9.1 高性能 Sort/Merge ユーティリティの使用

Alpha システムでは、高性能 Sort/Merge ユーティリティを選択することもできます。このユーティリティでは、Alpha アーキテクチャを活用することにより、ほとんどの Sort 操作および Merge 操作で性能を向上させることができます。

高性能 Sort/Merge ユーティリティは、SORT/MERGE と同じコマンド行インターフェースを使用します。高性能 Sort/Merge ユーティリティと SORT/MERGE との違いについては、本章で紹介します。

高性能 Sort/Merge ユーティリティを使用するときは、論理名 SORTSHR を使用します。次のように SORTSHR を定義して、SYS\$LIBRARY にある高性能ソート実行可能プログラムを指定します。

```
$ define sortshr sys$library:hypersort.exe
```

SORT/MERGE に戻るときは、SORTSHR の割り当てを解除します。SORTSHR が定義されていない場合、SORT/MERGE ユーティリティが省略時の値として設定されます。

注意

メモリ割り当ての差分により、高性能 Sort/Merge ユーティリティで実行するときの性能は、Sort/Merge ユーティリティと同サイズの仮想メモリで実行可能なソート操作数と同数の操作回数に制限されることがあります。

この場合、プロセスに使用できる仮想メモリのサイズを増やすか、ワーキング・セットの範囲を減らします。システム・パラメータにより、仮想メモリのサイズ増加やワーキング・セット範囲の削減については、『OpenVMS システム管理ユーティリティ・リファレンス・マニュアル』を参照してください。

高性能 Sort/Merge ユーティリティの動作は、ほとんど SORT/MERGE ユーティリティの動作と同じです。相違点は、表 9-1 に示します。

サポートされていない修飾子を使用しようとしたり、サポートされていない値を修飾子に指定しようとしたりすると、高性能 Sort/Merge ユーティリティはエラーを表示します。

表 9-1 高性能 Sort/Merge ユーティリティ: SORT/MERGE との相違点

キー・データ・タイプ	<p>H-FLOATING および ZONED 10 進データ・タイプはサポートされていません。</p> <p>BINARY データ・タイプ・キーのサイズは、1 バイト、2 バイト、4 バイト、8 バイトのいずれかでなければなりません。16 バイトのバイナリ・キーはサポートされていません (これは OpenVMS Alpha の将来のリリースでサポートされる予定です)。</p>
照合順序	<p>NCS(National Character Set) 照合順序はサポートされていません (これは OpenVMS Alpha の将来のリリースでサポートされる予定です)。/COLLATING_SEQUENCE 修飾子に NCS 照合順序を指定しないでください。ASCII, EBCDIC, および MULTINATIONAL 照合順序はサポートされています。省略時の設定は ASCII です。</p> <p>指定ファイルを使用して、照合順序を定義したり変更したりすることはできません (これは OpenVMS Alpha の将来のリリースで解決される予定です)。</p>
指定ファイル	<p>指定ファイルはサポートされていません (これは OpenVMS Alpha の将来のリリースでサポートされる予定です)。</p> <p>/SPECIFICATION 修飾子は使用しないでください。</p>
内部ソート・プロセス	<p>レコードのソート・プロセスのみサポートされています (これは OpenVMS Alpha の将来のリリースで解決される予定です)。/PROCESS=RECORD 修飾子を指定することも、または /PROCESS 修飾子を省略することもできます。/PROCESS 修飾子で TAG, ADDRESS, および INDEX を指定しないでください。</p>
統計情報	<p>現在サポートされているのは次の統計情報です。</p> <p>Records read (読み込まれたレコード数) Records sorted (ソートされたレコード数) Records output (出力されたレコード数) Input record length (入力レコード長)</p> <p>次の統計情報は利用できません。</p> <p>Internal length (内部長) Output record length (出力レコード長) Sort tree size (ソート・ツリー・サイズ) Number of initial runs (初期実行回数) Maximum merge order (最大マージ順) Number of merge passes (マージ・パス回数) Work file allocation (ワーク・ファイル割り当て)</p> <p>この機能の完全な実装は将来の OpenVMS Alpha リリースで行われる予定です。</p>

9.2 ファイルのソート

ファイルをソートするときは、DCL の SORT コマンドを使用します。ソートする入力ファイルを複数指定するときは、それぞれのファイルをコンマで区切り、最終的に作成されるソート済みの出力ファイルの名前をその後に続けます。

オプションとして、ソートに使用する各フィールドのキーを指定することができます。それぞれのキーには、次の情報が含まれます。

- レコード内にあるキー・フィールドの開始位置 (必須)
- キーのサイズ (必須)
- キーのデータ型
- レコードのソート順
- キーの優先順位

キーを指定しないでソートを行う場合、キーが1つだけ存在しており、このキー・フィールドが次のようになっているものと想定されます。

- レコードの最初の位置から開始する。
- すべてのレコードが含まれる。
- 文字データが含まれる。
- 昇順にソートが行われる。

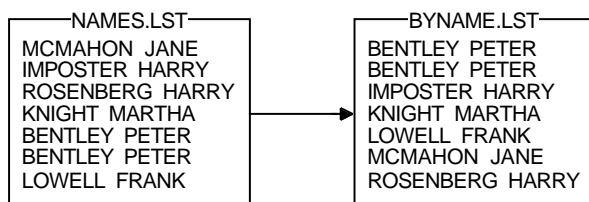
次の2つの例は、省略時のキーが使用されています。

1. 次の例では、NAMES.LST が昇順にソートされます。

```
$ SORT NAMES.LST BYNAME.LST
```

このコマンドは、図 9-1 に示すように、出力ファイル BYNAME.LST を作成します。

図 9-1 昇順にソートした結果



ZK-5055A-GE

2. この例では、ファイル NAMES.LST と NAMES2.LST をソートして、出力ファイル BYNAME.LST に出力します。2つのファイルは、1つの大きなファイルとして扱われ、ソートされます。

```
$ SORT NAMES.LST,NAMES2.LST BYNAME.LST
```

すべての SORT の修飾子の一覧については、第 9.9 節を参照してください。

9.2.1 キーの定義

キーを定義するときは、/KEY 修飾子を使用します。複数のキーを指定するときは、それぞれのキーごとに/KEY 修飾子を使用します。

表 9-2 は、キーを構成する 5 つの要素について説明しています。

表 9-2 /KEY 修飾子の値

キー要素	値	説明															
キーの位置	POSITION:n	レコード内にあるキー・フィールドの 1 バイト目の位置を表す。レコードの 1 バイト目が 1 になる。POSITION:nは必ず指定しなければならない。															
キー・サイズ	SIZE:n	<p>キー・フィールドの長さを表す。浮動小数データの場合を除き、SIZE:nは必ず指定しなければならない。</p> <p>キーに指定するデータ型により、サイズを指定するときに受け付けられる値が決まる。次の表は、それぞれのデータ型に対応する値と、キーのサイズを指定するときに使用する単位を表す。</p> <table> <tr> <th>データ</th><th>有効な範囲</th><th>単位</th></tr> <tr> <td>文字</td><td>1 ~ 32,767</td><td>文字</td></tr> <tr> <td>バイナリ</td><td>1, 2, 4, 8, 16(高性能 Sort /Merge ユーティリティでは、バイナリ・データ・タイプのサイズは 1 バイト, 2 バイト, 4 バイト, または 8 バイトでなければならぬ。16 バイトのバイナリ・データは OpenVMS Alpha の将来のリリースでサポートされる予定である。)</td><td>バイト</td></tr> <tr> <td>10 進</td><td>1 ~ 31</td><td>桁</td></tr> <tr> <td>浮動小数点</td><td>値は必要なし</td><td></td></tr> </table>	データ	有効な範囲	単位	文字	1 ~ 32,767	文字	バイナリ	1, 2, 4, 8, 16(高性能 Sort /Merge ユーティリティでは、バイナリ・データ・タイプのサイズは 1 バイト, 2 バイト, 4 バイト, または 8 バイトでなければならぬ。16 バイトのバイナリ・データは OpenVMS Alpha の将来のリリースでサポートされる予定である。)	バイト	10 進	1 ~ 31	桁	浮動小数点	値は必要なし	
データ	有効な範囲	単位															
文字	1 ~ 32,767	文字															
バイナリ	1, 2, 4, 8, 16(高性能 Sort /Merge ユーティリティでは、バイナリ・データ・タイプのサイズは 1 バイト, 2 バイト, 4 バイト, または 8 バイトでなければならぬ。16 バイトのバイナリ・データは OpenVMS Alpha の将来のリリースでサポートされる予定である。)	バイト															
10 進	1 ~ 31	桁															
浮動小数点	値は必要なし																
データ型	CHARACTER BINARY F_FLOATING D_FLOATING G_FLOATING H_FLOATING S_FLOATING	<p>10 進データでは、10 進記号が別のバイトに格納される場合、そのバイトはデータのサイズにカウントされない。</p> <p>指定したキーがレコードの終わりを越えた場合、失われた文字は空文字として扱われる。</p> <p>文字データ。CHARACTER が省略時のデータ型になる。</p> <p>バイナリ・データ。</p> <p>SIGNED— 符号付きバイナリ・データまたは符号付き 10 進データ。バイナリ・データおよび 10 進データの場合、SIGNED が省略時の値になる。</p> <p>UNSIGNED— 符号なしバイナリ・データまたは符号なし 10 進データ。</p> <p>F_FLOATING 形式のデータ。</p> <p>D_FLOATING 形式のデータ。</p> <p>G_FLOATING 形式のデータ。</p> <p>VAX システムでは、H_FLOATING 形式のデータになる (現在、高性能 Sort/Merge ユーティリティではサポートされていない)。</p> <p>Alpha システムでは、IEEE S_FLOATING 形式のデータになる。</p>															

(次ページに続く)

表 9-2 (続き) /KEY 修飾子の値

キー要素	値	説明
ソート順	T_FLOATING	Alpha システムでは、IEEE T_FLOATING 形式のデータになる。
	DECIMAL	10 進データ。 TRAILING_SIGN— 後続符号 10 進データ。10 進データの場合、TRAILING_SIGN が省略時の値になる。 LEADING_SIGN— 先行符号 10 進データ。先行符号はフィールドの最初になければならず、フィールドの残りのスペースにはゼロが埋め込まれる。 OVERPUNCHED_SIGN— オーバパンチ 10 進データ。10 進データの場合、OVERPUNCHED_SIGN が省略時の値になる。 SEPARATE_SIGN— 分割符号 10 進データ。
	ZONED	ゾーン 10 進データ (現在、高性能 Sort/Merge ユーティリティではサポートされていない)。
	PACKED_DECIMAL	パック 10 進データ。
	ASCENDING	ソート操作を、英数字の昇順でソートする。ASCENDING が省略時の順序になる。
	DESCENDING	ソート操作を、英数字の降順でソートする。
キーの優先順位	NUMBER:n	複数のキーが優先順位に従ってリストされていない場合、それぞれのキーの優先順位を指定する。1 ~ 255 までの値が指定できる。

キー・フィールドのデータが文字データでない場合は、データ型を指定する必要があります。Sort/Merge ユーティリティでは、次のデータ型が認識されます。

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 システムのみ)
T_FLOATING, IEEE (Alpha システムのみ)
PACKED_DECIMAL
ZONED

[]かっこ内の項目は省略時の値になるため、指定する必要はありません。

注意

10 進文字列データの場合、VAX システムや Alpha システムと違って、Sort/Merge ユーティリティが、入力文字列の無効な桁について報告します。VAX システムの場合、比較のために無効な桁 (または予約オペランド) が有効な 10 進文字列に変換されたというメッセージが表示されます。Alpha システムの場合、Sort/Merge ユーティリティによって同じ変換が行われますが、メッセージが表示されません。どちらの場合も、入力ファイルのデータがそのまま出力ファイルに記述されます。

各レコードが(1)部門名，(2)アカウント番号，(3)従業員名の3つのフィールドから構成される EMPLOYEE.LST ファイルを例として考えてみます。図 9-2 にこの3つのフィールドを示します。

図 9-2 リストの中のレコード・フィールド

EMPLOYEE.LST		
①	②	③
BST	7828	MCAHON 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

次に，EMPLOYEE.LST のレコードをソートする例を示します。

1. 次の例では，EMPLOYEE.LST はアカウント番号によってソートされ，アカウント番号フィールドを記述するために/KEY 修飾子を使用しています。

```
$ SORT/KEY=(POSITION:5,SIZE:4,DECIMAL) EMPLOYEE.LST BILLING1.LST
```

このコマンドは，キー・フィールド (アカウント番号) が位置 5 から始まること，長さが 4 文字であること，10 進データが格納されていること，昇順 (省略時の設定) でソートすることを指定しています。ソート結果を図 9-3 に示します。

図 9-3 キー・フィールドによるソート

EMPLOYEE.LST			BILLING1.LST		
BST	7828	MCAHON JANE	ANS	5243	BENTLEY PETER
ADM	7933	IMPOSTER HARRY	BST	7828	MCAHON 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

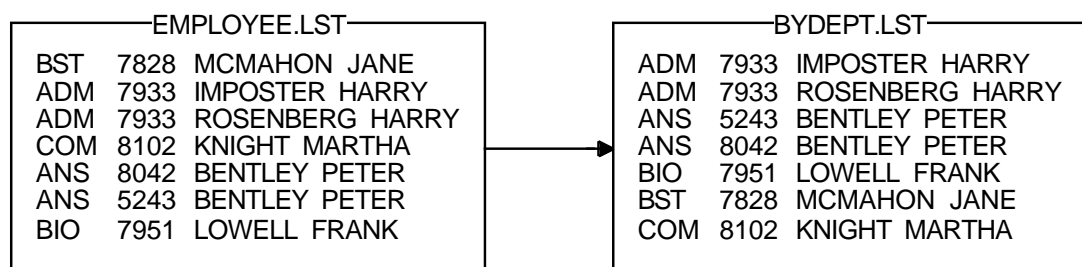
ZK-5058A-GE

2. 次の例は，キー・フィールドを指定せずに，ファイル EMPLOYEE.LST をソートする方法を示しています。

```
$ SORT EMPLOYEE.LST BYDEPT.LST
```

キーを指定していないので省略時の特性が使用されます。ソートの結果を図 9-4 に示します。

図 9-4 ソート済みリスト (キー・フィールドなし)



ZK-5057A-GE

Sort は、EMPLOYEE.LST 中のそれぞれレコードを 1 つの文字データ・キーとして使用します。この例では、各レコードは、部門名、アカウント番号、従業員名で構成されています。重複する部門名がある場合、それらの部門名をアカウント番号の昇順にソートします。さらにアカウント番号もいくつか重複していた場合は、従業員名のアルファベット順にソートします。ここでアカウント番号はレコードの一部であることに注意してください。特に指定しない限り、アカウント番号は文字データとして処理されます。

9.2.2 複数のキー・フィールドの使用方法

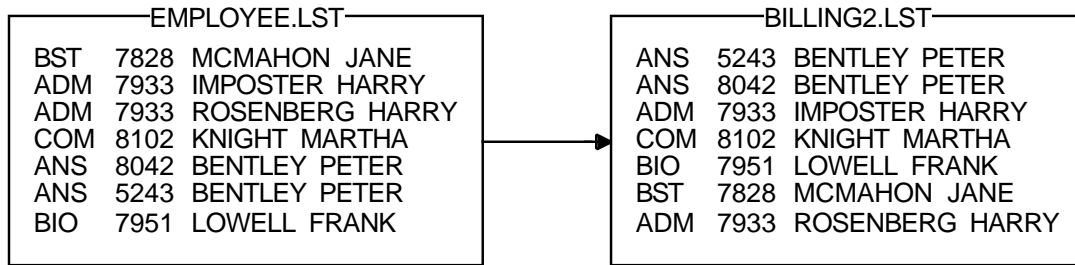
2 つ以上のキー (255 キーが上限) でソートを行うことができます。この場合には、複数のキーを優先順位順に指定します。すなわち、最初に 1 次キーを、次に 2 次キーを指定し、以下も同様になります。また別の方法として、NUMBER:n を使用して、キーの優先順位を指定する方法もあります。それぞれのキーは昇順でも降順でもかまいません。

たとえば、次のコマンドは、EMPLOYEE.LST を最初に従業員名をキーとしてソートし、次に (同一名があれば) アカウント番号順にソートします。

```
$ SORT /KEY=(POSITION:10,SIZE:15,CHARACTER) -  
_$/KEY=(POSITION:5,SIZE:4,DECIMAL) EMPLOYEE.LST BILLING2.LST
```

ソート結果を図 9-5 に示します。

図 9-5 複数のキー・フィールドによるソート結果



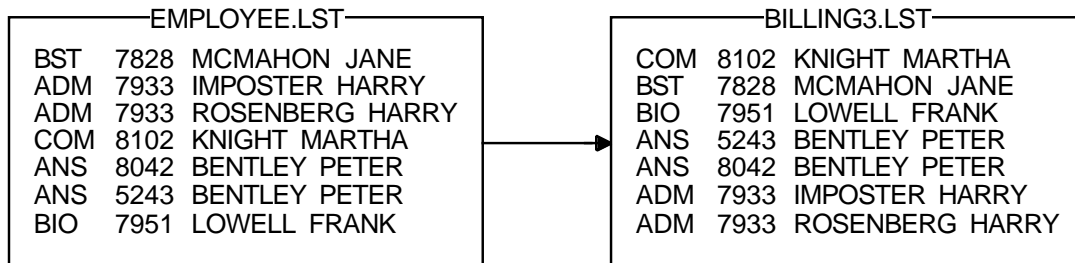
ZK-5059A-GE

それぞれのキーは昇順でも降順でもかまいません。たとえば、次のコマンドは、最初にレコードを部門名別に降順にソートしてから、次に従業員名別に昇順にソートします。

```
$ SORT/KEY=(POSITION:1,SIZE:3,DESCENDING) -  
_$/KEY=(POSITION:10,SIZE:15) -  
_$/ EMPLOYEE.LST BILLING3.LST
```

ソート結果を図 9-6 に示します。

図 9-6 昇順および降順キーによるソート結果



ZK-5060A-GE

9.2.3 同一キー・フィールドによるレコードのソート

省略時の設定では、Sort/Merge は同一キー・フィールドでレコードを保持しますが、それらのレコードを、入力ファイル中で現れた順序と同じ順序で保持するとは限りません。このようなレコードが存在すると思われる場合には、次の修飾子のいずれかを使用してその扱い方を指定することができます。

- /STABLE

同じキーを持つレコードを入力順序で残す。複数の入力ファイルをソートするときこの修飾子を使用すると、同じキーを持つレコードが最初のファイルにあれば

ば，2 番目のファイルの前に並べられることになる。2 番目以降についても同様の方法で処理される。

- /NODUPPLICATES

1 つを残し，他の重複レコードをすべて排除する。そのまま残す重複レコードを指定する場合，プログラム・レベルで Sort を起動し，同一キー・ルーチンを指定する。

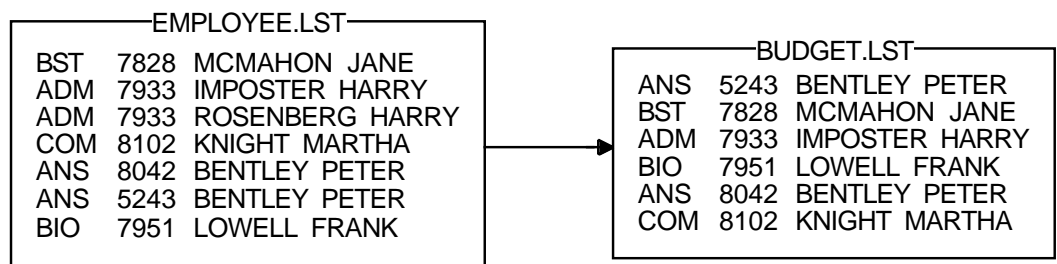
/STABLE 修飾子と/NODUPPLICATES 修飾子は互換性がないため，同一コマンド行で両方の修飾子を指定することはできません。

次の例では，アカウント番号が重複するレコードは，ファイル EMPLOYEE.LST から削除されます。

```
$ SORT /KEY=(POSITION:5,SIZE:4)/NODUPPLICATES EMPLOYEE.LST BUDGET.LST
```

図 9-7 はこのソート操作の結果を示しています。

図 9-7 同じキーを持つ場合のソート結果



ZK-5061A-GE

9.2.4 非文字データ・ファイルのソート

文字データ以外の項目を含むレコードをソートする場合は，それぞれのキーでデータ型を指定します。また，比較する項目が2 バイト以上占有することもあるため，開始位置とサイズを注意深く計算するようにします。

20 個の文字の後に，F_floating 形式の浮動小数を3 つ含むファイルをソートする場合，位置は次のようになります。

- 文字データは，1 から 20 までの位置を占有する (20 文字)。
- 最初の F 浮動小数は，21 から 24 までの位置を占有する。
- 2 番目の F 浮動小数は，25 から 28 までの位置を占有する。
- 3 番目の F 浮動小数は，29 から 32 までの位置を占有する。

3 番目の浮動小数でファイルをソートするときは、キー・フィールドを次のように指定します。

```
$ SORT/KEY=(POSITION:29,F_FLOATING) STATS.RAW STATS.SOR
```

浮動小数のサイズは 4 バイトに固定されているため、サイズを指定する必要はありません。

9.2.5 出力ファイルの編成

省略時の設定では、Sort は、最初の入力ファイルと同じファイル編成の出力ファイル生成します。入力ファイルと異なる出力ファイル編成を指定するときは、Sort のコマンド行で出力ファイルを指定した後に次のいずれかの修飾子を指定します。

- /FORMAT (レコード形式)

この出力修飾子を使用すると、ファイル・レコードの形式、長さ、ブロック・サイズを定義できます。

- /INDEXED_SEQUENTIAL

この修飾子を使用すると、出力を索引順編成ファイル編成に定義できます。出力ファイル編成として索引順編成を指定する場合は、次の操作も行わなければなりません。

- ソート操作を行う前に、出力ファイルとして使用する空のファイルを作成しておかなければなりません。これは、ソート操作では、既に存在する空の出力ファイルが必要だからです。
- SORT コマンド行で、出力ファイル名の後に/OVERLAY 修飾子を指定しなければなりません。/OVERLAY 修飾子は、既存のファイルの上に入力ファイルのソートしたレコードをオーバーレイするよう指定します。

- /RELATIVE

この修飾子を使用すると、出力を相対ファイル編成に定義できます。

- /SEQUENTIAL

この修飾子を使用すると、出力を順ファイル編成に定義できます。

次の例では、索引付き順編成ファイル EMPLOYEE.LST をソートした後、順編成ファイルが作成されます。

```
$ SORT/KEY=(POSITION:10,SIZE:15) -  
_ $ EMPLOYEE.LST BYNAME.LST/SEQUENTIAL
```

9.2.6 ソートのプロセス

Sort は、レコード、タグ、アドレス、索引などの内部プロセスを 1 つ使用して、ファイルの整列を行います。(高性能 Sort/Merge ユーティリティでは、レコード・プロセスのみがサポートされます。タグ、アドレス、索引の各プロセスについては、OpenVMS Alpha の将来のリリースでサポートされる予定です。) 指定するプロセスによっては、Sort 操作の効率が変わることもあります。Sort 操作、Merge 操作の最適化については、第 9.8 節を参照してください。

次の表は、プロセスの 4 つのタイプを示しています。ソート・プロセスを指定するときは、/PROCESS=タイプ修飾子を使用します。

ソート・プロセス	タイプ	説明
レコード	RECORD	ソートを行うときレコードを完全な状態のまま残し、完全なレコードで構成される出力ファイルを作成する。レコードが省略時のソート・プロセスになる。
タグ	TAG	キー・フィールドのみをソートしてから、入力ファイルを読み込み直し、完全なレコードの出力ファイルを作成する。実際の結果は、完全なレコード・ソートの場合と同じになる。 タグ・ソートは通常、ソート時に作業ファイルの領域をあまり使用しないため、ディスク領域が少ない場合に使用するのに適している。ほとんどの場合、タグ・ソートは、レコード・ソートよりも処理速度が遅くなる。これは、入力ファイルを読み込み直すときに余分の時間が必要になるためである。
アドレス	ADDRESS	キー・フィールドのみをソートし、レコード・ファイル・アドレス (RFA) の索引になる出力ファイルを、バイナリ形式で作成する。 アドレス・ソートは、レコード・ソートよりも高速であるが、レコード・アドレスを入力ファイルのレコードと対応させるプログラムを記述する必要がある。
索引	INDEX	キー・フィールドのみをソートし、キーと RFA の出力ファイルを (バイナリ形式で) 作成する。 アドレス・ソートの場合と同様に、索引ソートもレコード・ソートよりも高速であるが、レコード・アドレスを入力ファイルのレコードと対応させるプログラムを記述する必要がある。

9.3 照合順序の指定

文字は、照合順序の順番に従って、ソートされます。文字データを含むファイルの場合は、`/COLLATING_SEQUENCE=sequence`修飾子を使用して、照合順序を指定します。次の表は、照合順序オプションについて説明しています。

照合順序	順序	説明
ASCII	ASCII	文字データの場合、省略時の照合順序になる。ASCII 順序では、数字 (0 ~ 9)、大文字 (A ~ Z)、小文字 (a ~ z) の順番になる。
EBCDIC	EBCDIC	EBCDIC 順序で並べられた出力ファイルを生成する。データは ASCII 表現のままになる。EBCDIC 順序では、小文字 (a ~ z)、大文字 (A ~ Z)、数字 (0 ~ 9) の順番になる。
DEC で定義している文字セット	MULTINATIONAL	MULTINATIONAL 照合順序では、DEC で定義している文字セットに従って文字を整列させる (付録 A を参照)。MULTINATIONAL の文字順序では、文字は次の規則で並べられる。 <ul style="list-style-type: none"> • 発音区別形式のすべての文字には、その文字の照合値が指定される (A', A", A' の場合 A で照合)。 • 小文字には、それに相当する大文字の照合値が指定される (a では A, a" では A")。 • 2 つの文字列が比較されて同じものと判断された場合、タイプブレークが実行される。文字列が比較され、発音区別符号、無視される文字、実際には異なるにもかかわらず同じものとして扱われる文字などにより、両者の違いが検出される。2 つの文字列がそれでも同じものと判断される場合、文字の数値コードに基づいた比較が再度行われる。この最終的な比較では、小文字が大文字の前に並べられる。
国別文字セット (NCS)	照合順序名	指定する照合順序は、NCS ライブラリで定義されていなければならない。詳細は、『OpenVMS National Character Set Utility Manual』を参照。 (高性能 Sort/Merge ユーティリティは、NCS 照合順序をサポートしていない。OpenVMS Alpha の将来のリリースでサポートされる予定である。)
ユーザ定義順序	(順序文字列)	ユーザ定義照合順序を指定する。ユーザ定義照合順序は、指定ファイルでのみサポートされており、コマンド行インタフェースではサポートされていない。 (高性能 Sort/Merge ユーティリティは、ユーザ定義順序をサポートしていない。OpenVMS Alpha の将来のリリースではサポートされる予定である。)

照合順序	順序	説明
		<p>単一文字または二重文字の文字列、あるいは単一の文字を複数集めた文字の集まりを指定することによって、照合順序を定義する。二重文字は、単一の文字を 2 つ集めて 1 文字のように扱う。たとえば、整列するときに "CH" を "C" として扱うよう定義することができる。この文字列は括弧で囲んで指定する。</p> <p>基数演算子 %O, %D, %X を使用して、それぞれ 8 進、10 進、16 進の値で、文字を表現することもできる。</p> <p>照合順序を定義するとき、次の規則を守らなければならない。</p> <ul style="list-style-type: none">• 文字を二重引用符 (") で囲む。• それぞれの文字と一連の文字をコンマ (,) で区切り、リスト全体を括弧で囲む。• Sort 操作、Merge 操作の文字キーで使用するすべての文字に照合値を付ける。照合値の付いていない文字は、FOLD オプションまたは MODIFICATION オプションが指定されていない限り無視される。• 文字を複数回定義しないようにする。• 空文字を指定するとき、二重引用符 (") を使用しないようにする。二重引用符ではなく、%X0 などの基数演算子を使用する。• 二重引用符を指定するときは、別の二重引用符で囲む (" " ") か、基数演算子を使用する。 <p>次の文字列は、二重文字 LL が L と M の間の単一文字として照合される照合順序を定義する。</p> <p>("A"- "L", "LL", "M"- "Z")</p>

注意

DEC で定義している照合順序を使用して、ファイルをソートまたはマージするときには注意が必要です。ほとんどのプログラミング言語のシーケンス・チェック・プロシージャは、数字を比較します。DEC で定義している照合順序は、グラフィック文字を表すコードを比較するのではなく、実際のグラフィック文字を比較するため、通常シーケンス・チェックは正しく動作しません。

次の例は、指定ファイルで使用するために、ユーザ定義照合順序を作成する方法を示しています。指定ファイルについての詳細は、第 9.7 節を参照してください。

1.

```
(/COLLATING_SEQUENCE=(SEQUENCE=ASCII,IGNORE=(-," ")))
```

このように、/COLLATING_SEQUENCE 修飾子に IGNORE オプションを指定した場合、次のフィールドが比較されるとき同じものとして扱われ、タイプブレークになります。

```
252-3412
252 3412
2523412
```


2.

```
/COLLATING_SEQUENCE=(SEQUENCE=("A"- "L", "LL", "M"- "R", "RR", "S"- "Z"))
```

この/COLLATING_SEQUENCE 修飾子は、二重文字 LL が L と M の間の単一文字として照合され、二重文字 RR が R と S の間の単一文字として照合される順序を定義します。このような定義がなければ、これらの二重文字は、通常のアルファベット順と同様になります。省略時の場合、このユーザ定義順序によって、小文字の a から z のような、他の文字が定義されることはありません。

9.4 バッチ・ジョブによるソートの実行

バッチ・ジョブとは、現在のセッションとは独立して実行するプログラムや DCL コマンド・プロシージャのことです。大きなファイルをソートする場合には、ソート操作を終了するまでに時間がかかるので、ソート操作をバッチ・ジョブとして登録することを検討してみてください。バッチ・ジョブは、現在のセッションとは独立してバッチ・ジョブとコマンド・プロシージャについての詳細は、第 16 章、第 13 章、および第 14 章を参照してください。

9.4.1 コマンド・プロシージャ

コマンドを画面に記述するのと同様の方法で、コマンド・プロシージャに SORT コマンドを指定します。省略時のディレクトリにソートするファイルがない場合は、コマンド・プロシージャで省略時のディレクトリを明示的に設定するか、コマンド・ファイル指定にディレクトリを入れるようにします。

次の例では、DCL コマンド・プロシージャ SORTJOB.COM をバッチ・ジョブとして登録しています。コマンド・プロシージャのテキストをコマンド行の後に示します。

```
$ 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
```

9.4.2 入力レコードの包含

バッチ・ジョブには、入力レコードを含めておくことができます。これには、SORT コマンドの後に入力レコードを 1 行に 1 つずつ指定します。ただし、個々のソート・レコードは 2 行以上になってもかまいません。

ファイルのソートとマージ

9.4 バッチ・ジョブによるソートの実行

レコードをターミナルから入力する場合と同様に、入力ファイル・パラメータには `SYSS$INPUT` を指定します。その後に `/FORMAT` 修飾子を使用して、レコード・サイズ (バイト数) とおおよそのファイル・サイズ (ブロック数) を指定します。80 文字の行が 6 行で 1 ブロックとほぼ等しくなります。

次の例は、入力レコードを含めたコマンド・プロシージャを示しています。

```
$ SUBMIT SORTJOB

! SORTJOB.COM
!
$ SET DEFAULT [USER.PER]
$ SORT/KEY=(POSITION:10,SIZE:15) -
SYSS$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
```

9.5 ファイルのマージ

`MERGE` コマンドは、最大で 10 個 (高性能 Sort/Merge ユーティリティの場合は、最高で 12 個) のソート済みのファイルをまとめて、レコードが順に並べられた 1 つの出力ファイルを作成します。マージする入力ファイルは、同じ形式を持ち、同じキー・フィールドに基づいてソートされていなければなりません。

省略時の設定では、Merge は、入力ファイルの中のレコードをチェックして、正しい順序で並べられているかどうかを確認します。Merge で順序をチェックするのが必要がなければ、`/NOCHECK_SEQUENCE` 修飾子を指定します。`/CHECK_SEQUENCE` 修飾子を指定し、レコードの順序が正しくない場合、(たとえば、入力ファイルのどれかをソートしていなかった場合)、Merge では以下のエラーをレポートします。

```
%SORT-W-BAD_ORDER, merge input is out of order
```

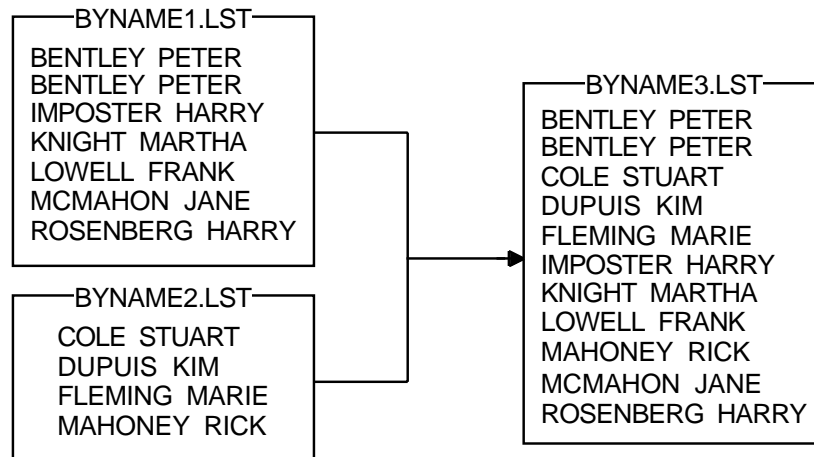
`MERGE` コマンドと `SORT` コマンドは、同じ修飾子を使用できます。ただし、次の 2 つの例外があります。

- マージ操作にはプロセス (`/PROCESS`) を指定できない。
- `/CHECK_SEQUENCE` 修飾子はマージ操作にだけ使用される。

次の例では、ファイル BYNAME1.LST と BYNAME2.LST は、従業員名によってすでに昇順にソートされています。ここに示したコマンドは、これらをマージします。

```
$ MERGE BYNAME1.LST,BYNAME2.LST BYNAME3.LST
```

出力ファイル BYNAME3.LST には、次の図に示すように、BYNAME1.LST と BYNAME2.LST の両方のファイルのすべてのレコードが含まれます。



ZK-5062A-GE

9.5.1 ソート済みのファイル

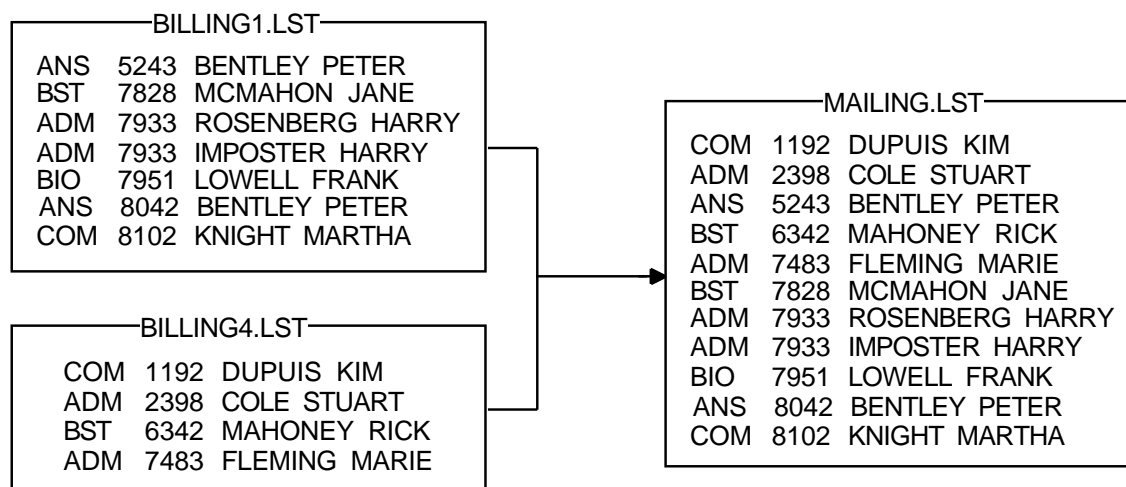
特定のキーを使用してソートしたファイルをマージする場合は、MERGE コマンド行の/KEY 修飾子で同じキーを指定する必要があります。

キーを指定しない場合は、Merge は第 9.2 節で説明している省略時設定のキーを使用します。

たとえば、ファイル BILLING1.LST と BILLING4.LST がアカウント番号によってソート済み (/KEY=POSITION:5,SIZE:4,DECIMAL) であるとします。この 2 つのファイルを出力ファイル MAILING.LST にマージするには、次のコマンド行を入力します。

```
$ MERGE/KEY=(POSITION:5,SIZE:4,DECIMAL) -  
_$ BILLING1.LST,BILLING4.LST MAILING.LST
```

マージ結果は次のとおりです。



ZK-5063A-GE

マージしたいファイルがソート順になっていることを知っている場合は、
/NOCHECK_SEQUENCE 修飾子を指定することにより、順序チェックを省略することができます。

9.5.2 同一キー・フィールドによるレコードのマージ

ソート操作の場合と同様に、入力ファイルに同じキー・フィールドを持つレコードが2つ以上ある場合、レコード順序が入力ファイルと同じになるとはかぎりません。同じキーを持つレコードの入力順序をそのまま残すには、MERGE コマンド行に/STABLE 修飾子を指定します。同じキーを持つレコードのうち1つだけを残すには、/NODUPLICATES 修飾子を指定します。

9.6 ターミナルからのレコードの入力

ソートまたはマージしようとするレコードは、ファイルに格納されている必要はありません。SORT または MERGE コマンドを入力するときに、ターミナルからレコードを直接入力することができます。次の表に、その手順を示します。

手順	操作
1	SORT または MERGE コマンド行で、入力ファイルとして SYSS\$INPUT を指定する。 入力ファイル修飾子/FORMAT を使用して、最も長いレコードのサイズ (バイト数) とそれに対応する入力ファイルのサイズ (ブロック数) を指定する。
2	後続の行にレコードを入力する。 Return を押して各レコードを終了させる。
3	Ctrl/Z を押してファイルを終了させる。

次の例は、ソートしようとする入力レコードを、ターミナルから直接に入力する場合のソート操作を示しています。

```
$ 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
```

このコマンド・シーケンスは、レコードをソートし、その結果を BYNAME.LST という名前のファイルに格納します。

9.7 Sort/Merge 指定ファイルの使用方法

Sort/Merge では、指定ファイルを使用してソート定義を格納しておき、必要があればさらに複雑なソート条件を指定することができます。(高性能 Sort/Merge ユーティリティでは、指定ファイルはサポートされていません。この機能は、OpenVMS Alpha の将来のリリースでサポートされる予定です。) 指定ファイルを作成するときは、任意の標準エディタや DCL の CREATE コマンドが使用できます。

Sort/Merge 指定ファイルを使用すると、次の操作を行えます。

- ソート/マージの対象とするレコードの選択
- 出力ファイルに格納されるレコードのフォーマットの変更
- 条件付きキーまたはデータの使用
- 複数のレコード形式の指定
- 照合順序シーケンスの作成や変更
- 作業ファイルの再割り当て
- 頻繁に使用するソート/マージ操作をファイルに格納

指定ファイルを作成したら、/SPECIFICATION 修飾子を使用してファイル名を指定します。指定ファイルの省略時のファイル・タイプは.SRT です。

指定ファイル内では、各コマンドの先頭にスラッシュ (/) を付けます。コマンドが 2 行以上にわたる場合でも、行継続文字は必要ありません。

注意

指定ファイルで使用する修飾子の多くは、Sort/Merge コマンド行で使用する DCL 修飾子と似ています。ただし、これらの修飾子の構文は異なる場合があります。たとえば、DCL レベルの/KEY 修飾子は、指定ファイルの/KEY 修飾子とは構文が異なります。指定ファイルの修飾子については、第 9.9.3 項を参照してください。

コマンド行で指定する DCL コマンド修飾子は、指定ファイル中の対応する項目を無効にします。たとえば、DCL コマンド行で/KEY 修飾子を指定しても、Sort/Merge は指定ファイル中の/KEY 修飾子を無視します。

一般に、指定ファイル中の修飾子はどのような順序で指定してもかまいません。ただし、次のような場合には、順序が重要になります。

- NUMBER:n キー要素を指定しない場合に、複数のキー・フィールドでソートする。
- 出力形式を記述する。
- 複数のレコード・タイプを定義する。

/COLLATING_SEQUENCE 修飾子と同時に FOLD, MODIFICATION, および IGNORE キーワードを指定する場合には、MODIFICATION 句と IGNORE 句をすべて指定してから FOLD 句を指定します。/COLLATING_SEQUENCE 修飾子についての詳細は、第 9.9.3 項を参照してください。

指定ファイルの中にコメントを挿入するには、各コメント行の先頭に感嘆符(!)を付けます。DCL のコマンド行と異なり、指定ファイルは行を継続するときにハイフン(-)を必要としません。

例

1. 次の例は、負と正のデータを昇順にソートするための指定ファイルです。

```
! 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)
```

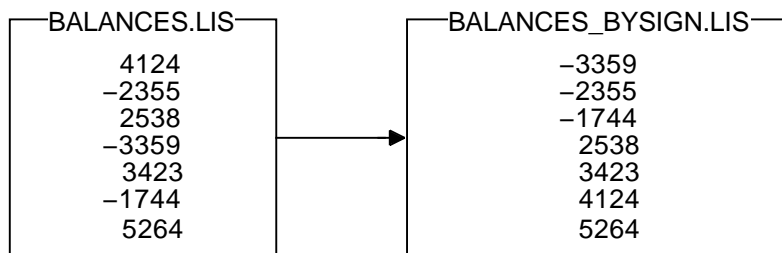
指定ファイルを確認する際には、次の点に注意してください。

- 1 このコマンド行は、レコードの第 1 バイトで始まる、長さ 1 バイトのフィールドを定義し、このフィールドに名前 SIGN を割り当てる。
- 2 このコマンド行は、レコードの第 2 バイトで始まる、長さ 4 バイトのフィールドを定義し、このフィールドに名前 AMT を割り当てる。

- 3 これは条件文である。SIGN バイトに負符号 (-) がある場合、条件 CHECK1 を満たす。
- 4 これは条件文である。SIGN バイトがブランクの場合、条件 CHECK2 を満たす。
- 5 条件 CHECK1 を満たす場合、レコードを降順にソートする。
- 6 条件 CHECK2 を満たす場合、レコードを昇順にソートする。

図 9-8 は、上の指定ファイルの例を BALANCES.LIS という入力ファイルに使用した結果を示しています。

図 9-8 指定ファイルを使用した出力



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)

```

この例では、不動産代理店の2つの支社から送られた2つの入力ファイルが、指定ファイルの指示に従ってソートされます。1つ目のファイルのレコードは、最初の位置に A がついており、次の形式になっています。

ファイルのソートとマージ

9.7 Sort/Merge 指定ファイルの使用方法

A	PRICE	TAXES	STYLE	ZIP
1	2	10	15	25

2 目目のファイルのレコードは、最初の位置に B がつき、style フィールドと zip code フィールドが、次のように逆になっています。

B	PRICE	TAXES	ZIP	STYLE
1	2	10	15	20

これらの 2 つのファイルを、レコード A の形式の zip code フィールドでソートするとき、最初に両方のレコードのフィールドを、/FIELD 修飾子で定義します。その後、/CONDITION 修飾子で、2 つのタイプのレコードを区別するためのテストを指定します。最後に/INCLUDE 修飾子を指定して、タイプ B のレコード形式をタイプ A のレコード形式に変更して出力します。

/INCLUDE 修飾子でキー・フィールドまたはデータ・フィールドのいずれかを指定する場合、/INCLUDE 修飾子の Sort 操作で明示的にすべてのキー・フィールドとデータ・フィールドを指定する必要があります。

また、タイプ A でもタイプ B でもないレコードは、ソート時に排除されます。

3.

```
/COLLATING_SEQUENCE=(SEQUENCE=
("AN","EB","AR","PR","AY","UN","UL",
"UG","EP","CT","OV","EC","0"- "9"),
MODIFICATION=("'"="19"),
FOLD)
```

この/COLLATING_SEQUENCE 修飾子では、ユーザ定義順序を指定します。この指定により、それぞれの月に、日付順の固有の値が定義されます。たとえば、SEMINAR.DAT というファイルを日付順に並べたい場合、SEMINAR.DAT ファイルは次のようになっています。

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

一次キーが year フィールドで、二次キーが month フィールドです。month フィールドは数値ではありませんが、日付順に並べたいため、独自の照合順序を定義する必要があります。この場合、それぞれの月に固有のキー値を指定して、それぞれの月の 2 番目と 3 番目の文字を (日付順に) ソートすることにより、これを行います。

MODIFICATION オプションでは、アポストロフィ (') が 19 と同等になるよう指定していますが、これによって、'83 と 1984 が比較できるようになります。また FOLD オプションは、大文字と小文字が同等に扱われるよう指定します。

この Sort 操作の出力は、次のようになります。

```
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
```

ユーザ定義照合順序の他の例については、第 9.3 節を参照してください。

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)
```

この例では、2つの不動産ファイルをソートします。1つのファイルは、ある代理店が Real-T Trust と呼ぶもので、もう1つのファイルは Realty Trust と呼ばれます。/CONDITION 修飾子と/DATA 修飾子により、Realty Trust のソート出力ファイルに AGENT フィールドがリストされることになります。

5.

```
/FIELD=(NAME=ZIP,POSITION:60,SIZE:6)
/CONDITION=(NAME=LOCATION,
            TEST=(ZIP EQ "01863"))
/KEY=(IF LOCATION THEN 1
      ELSE 2)
```

この例では、郵便番号 01863 を持つすべてのレコードがソート出力ファイルの最初にリストされます。条件テストは、/FIELD 修飾子で定義されているように ZIP フィールドで行われます。条件名は LOCATION になっています。/KEY 修飾子にある値 1 と 2 は、条件を満たすレコードと条件を満たさないレコードの相対順序を表すものです。

6.

```
/FIELD=(NAME=ZIP,POSITION:60,SIZE:6)
/CONDITION=(NAME=LOCATION,
            TEST=(ZIP EQ "01863"))
/DATA=(IF LOCATION THEN "NORTH CHELMSFORD"
      ELSE "Outside district")
```

この例では、/CONDITION 修飾子により、郵便番号 01863 についてテストが行われます。/DATA 修飾子は、テスト結果に従って、town フィールドの名前が出力レコードに追加されるよう指定します。

7.

```
/FIELD=(NAME=FFLOAT,POS:1,SIZ:0,F_FLOATING)
/CONDITION=(NAME=CFFLOAT,TEST=(FFLOAT GE 100))
/OMIT=(CONDITION=CFFLOAT)
```

この例では、/FIELD 修飾子でフィールド FFLOAT が F_FLOATING と定義されているため、数値 100 が F_FLOATING データ型とみなされます。

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
```

この/FIELD 修飾子は、次の形式を持つ入力ファイルのレコードのフィールドを定義します。

```
AGENT ZIP STYLE CONDITION PRICE TAXES
```

/DATA 修飾子では、/FIELD 修飾子で定義されているフィールド名を使用しますが、レコードを再フォーマットして、次の形式の出力レコードを作成します。

```
PRICE TAXES STYLE ZIP AGENT
```

9.8 ソートまたはマージ操作の最適化

ソート操作またはマージ操作の効率は、ソート環境により、いくつかの方法で向上させることができます。SORT コマンドまたは MERGE コマンドに/STATISTICS 修飾子を使用すると、ソート環境の変数を表示できます。

この統計情報を調べてから、この後に説明する最適化オプションの使用を検討してみてください。

SORT コマンドまたは MERGE コマンドに/STATISTICS 修飾子を指定すると、次のような出力が表示されます。

```
$ 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
```

この統計表示から次のことが分かります。

1 Records read

ソート操作中に読み込まれたレコードの数を示す。ソート操作で特定のレコードを省略する方法についての詳細は、第 9.8.2 項を参照。

2 Working set extent

ソート操作を実行するために確保されたブロックの数を示す。ワーキング・セットを大きくする方法についての詳細は、第 9.8.4 項を参照。

3 Page faults

オペレーティング・システムが、物理メモリからページング・デバイスへプロセスを転送した回数を示す。ページングをしないようにする方法についての詳細は、第 9.8.4 項を参照。

4 Work file allocation

作業ファイルのために確保されたディスク領域のサイズを示す。作業ファイルについての詳細は、第 9.8.3 項を参照。

5 Elapsed CPU

オペレーティング・システムがソート操作の処理に費やした CPU 時間を示す。別のソート方法を選択して時間を節約する方法についての詳細は、第 9.8.1 項を参照。

9.8.1 ソートのプロセス

Sort では、内部でデータをソートするときに使用する、レコード、タグ、アドレス、索引の 4 つのプロセスを定義します。(高性能 Sort/Merge ユーティリティでは、レコード・プロセスのみがサポートされます。タグ、アドレス、索引の各プロセスについては、OpenVMS Alpha の将来のリリースでサポートされる予定です。) RECORD が省略時のプロセスになります。指定するプロセスのタイプによっては、必要な記憶域の量と同様に、Sort 操作の効率も変わることがあります。その他のソート・プロセスについての詳細は、第 9.2.6 項を参照してください。

どのタイプのソート方法を使用するかは、次の点を考慮して決定してください。

- 出力ファイルをどのように使用するか
 - レコード・ソートとタグ・ソートは、ソート済みの全レコードを含むファイルを生成するため、これらのファイルはいつでも使用できる。
 - アドレス・ソートと索引順編成ソートの出力ファイルは、Pascal、FORTRAN、MACRO、Cなどのプログラミング言語で作成されたプログラムによって処理できる。
 - アドレス・ソートは、入力ファイルの中のレコードに対するポインタのリストを作成する。このリストは、バイナリ形式のRFAと、複数の入力ファイルをソートするときのファイル番号から構成される。プログラムは、ポインタを使用してレコードにアクセスする。
 - 索引順編成ソートは、RFAとキー・フィールドに加えて、複数のファイルをソートするときのファイル番号が含まれる出力ファイルを作成する。これらのキー・フィールドの形式は入力ファイルと同じ。キー・フィールドの内容に従ってそれ以降の処理を決定する必要があるプログラムの場合には、アドレス・ソートではなく、索引順編成ソートを選択する。

あるファイルのレコードを様々な目的で並べ換える必要がある場合には、アドレス・ソートまたは索引順編成ソートによって生成された複数の出力ファイルを格納する。望ましい順序で並べられたメイン・ファイルの中のレコードにアクセスするには、この出力ファイルを使用する。

- ソートに使用可能な一時的な格納領域
タグ・ソートは、レコード・ソートに比べて、使用する一時格納領域が少なくてすむ。レコード・ソートは、ソート中はレコードに手を加えないので、ファイルが大きい場合には使用する作業領域が多くなる。アドレス・ソートと索引順編成ソートは、一時格納領域をほとんど使用しない。
- 使用する入力デバイスと出力デバイスのタイプ
カード、磁気テープ、ディスクから入力を受け付けるプロセスは、レコード・ソートだけである。タグ・ソートとレコード・ソートの結果はどの出力デバイスにも出力できるが、アドレス・ソートと索引順編成ソートの結果は、バイナリ・データを受け付けるデバイスに出力しなければならない。
- 処理速度の相違
ソートしたレコードを操作の最中に検索する予定がある場合、通常、レコード・ソートが最も処理速度が速くなる。それ以外の場合には、アドレス・ソートと索引順編成ソートの処理速度が速くなる。

9.8.2 レコードとフィールドの排除

Sort の効率化は、指定ファイルを使用して高めることもできます。/CONDITION , /INCLUDE , /OMIT 修飾子を使用すると、必要なレコードだけを選択して出力ファイルに格納できます。(高性能 Sort/Merge ユーティリティでは、指定ファイルはサポートされていません。この機能は、OpenVMS Alpha の将来のリリースでサポートされる予定です。) また、指定ファイル修飾子を使用すれば、レコードのフォーマットを変更して、不必要なフィールドを出力ファイルに出さないようにすることもできます。これらの修飾子は、コマンド行修飾子として使用することはできません。

9.8.3 作業ファイルの割り当て

ソート中、入力ファイルのレコードはメモリに読み込まれています。Sort では、割り当てられたメモリにすべてのレコードを収めることができない場合は、ソート済みのデータが、1 つまたは複数の一時的作業ファイルに転送されます。Merge の場合は、作業ファイルが使用されません。

作業ファイルの数を変更し、それらのファイルを特定のデバイスに割り当てることにより、ソートの効率を向上させることができます。

- コマンド行修飾子/WORK_FILES=nを使用することにより、割り当てられた作業ファイルの数が指定変更される。
- 通常、作業ファイルはデバイス SYS\$SCRATCH に置かれ、任意の順序でアクセスされる。作業ファイルを特定のデバイスに割り当てる方法として、次の 2 つの方法がある。
 - 指定ファイルで/WORK_FILES=(device,...) 修飾子を使用すると、作業ファイルが、指定されたデバイスに置かれる。指定ファイル内で/WORK_FILES 修飾子を使用する方法についての詳細は、
 - 指定ファイルを使用しない場合、DCL コマンドの ASSIGN を使用することによって、作業ファイルを特定のデバイスに割り当てる。

作業ファイルに対応するユーザ指定デバイス名を識別するときは、SORTWORKn論理名が使用される。この場合nは0から9までのいずれかの値になる。(高性能 Sort/Merge ユーティリティでは、nは0から254までのいずれかの値になる)。SORTWORKn論理名を次のように定義する。

ASSIGN デバイス: SORTWORKn

例

```
$ ASSIGN WORK$2: SORTWORK1
$ ASSIGN WORK$3: SORTWORK2
```

この例では、SORTWORK1 がデバイス WORK\$2: として定義され、SORTWORK2 がデバイス WORK\$3: として定義される。論理名についての詳細は、第 11 章を参照。

作業ファイルをデバイスに割り当てるとき、次の事項に注意します。

- RAM や大容量ディスクのような、できる限り高速なデバイスに、作業ファイルを割り当てるようにする。
- あまり使用しないデバイスで、もっとも容量のあるものを選択する。
- それぞれの作業ファイルを異なる物理デバイスに割り当て、入力と出力のオーバーラップを最大限にする。

9.8.4 ワーキング・セット拡張領域の変更

Sort が作業ファイルを必要とする場合 (たとえば、大きなファイルをソートする場合) には、ワーキング・セットを大きくすると、ソート効率が向上します。ただし、システムの負荷が大き過ぎると、ワーキング・セット拡張領域の全ページをプロセスに割り当てられないこともあります。このような場合、オペレーティング・システムが物理メモリとページング・デバイス上のメモリの間でプロセスの一部を転送するとき、ページングが起こることがあります。そのため、プロセスのアクティブな部分だけが物理メモリ内に残ります。過度なページングの発生を避けるためには、プロセスのワーキング・セット拡張領域を小さくします。ワーキング・セット拡張領域を小さくするときは、SET WORKING_SET コマンドを使用します。

9.9 Sort/Merge 修飾子の要約

ここでは、SORT コマンドと MERGE コマンドで使用されるコマンド修飾子について説明します。コマンド修飾子を使用するには、SORT または MERGE コマンドのすぐ後に修飾子を指定します。

/[NO]CHECK_SEQUENCE

MERGE コマンドのみに適用。MERGE 入力ファイル内のレコードの順序を検査する。省略時の設定では、レコードの順序がチェックされる。

また/CHECK_SEQUENCE 修飾子を使用すると、1 つまたは複数 (最高で 10 ファイル、高性能 Sort/Merge ユーティリティの場合は 12 ファイルまでサポート) のファイルのレコードがソートされているかどうかチェックすることができる。この場合もレコードは出力ファイルに送られるが、そのファイルは指定する必要がある。レコードが、レコード全体以外のキー・フィールドでソートされているかどうかチェックする場合、順序の他に、キー情報も指定する必要がある。

マージ操作で、レコードの順序をチェックしないようにするには、/NOCHECK_SEQUENCE 修飾子を使用する。

例

```
$ MERGE/KEY=(SIZE:4,POSITION:3)/NOCHECK_SEQUENCE -  
_ $ PRICE1.DAT,PRICE2.DAT PRICE.LIS
```

/NOCHECK_SEQUENCE 修飾子により、入力ファイル PRICE1.DAT および PRICE2.DAT の順序についてはチェックする必要のないことが指定されている。

/COLLATING_SEQUENCE=sequence

文字キー・フィールドに対して、定義されている 3 つの照合順序から 1 つを選択するか、文字キーの比較に使用する国別文字セット (NCS) 照合順序の名前を指定する。(高性能 Sort/Merge ユーティリティでは、NCS 照合順序はサポートされていない。NCS 照合順序は、OpenVMS Alpha の将来のリリースでサポートされる予定。) 文字は ASCII(省略時)、EBCDIC、MULTINATIONAL のいずれかの順序で整列される。

例

```
$ SORT/COLLATING_SEQUENCE=MULTINATIONAL -  
_ $ NAMES.DAT,NOM.DAT LIST.LIS
```

この SORT コマンドでは、MULTINATIONAL 照合順序で入力ファイル NAMES.DAT および NOM.DAT が整列され、出力ファイル LIST.LIS が作成される。

/[NO]DUPLICATES

省略時の設定では、重複キーを持つ複数レコードがすべて保持される。/NODUPLICATES 修飾子を使用すると、重複キーを持つ複数レコードのうち 1 つを除いてすべてが削除される。保持されるレコードは、入力ファイルの順序と同じ順序にならない場合がある。重複レコードの順序を保持するように指定したい場合は、プログラム・レベルで Sort を起動し、同一キー・ルーチンを指定する。

/STABLE 修飾子と/NODUPLICATES 修飾子は、相互に排他的な関係にある。

例

```
$ SORT/KEY=(POSITION:3,SIZE:5,DECIMAL)/NODUPLICATES -  
_ $ ACCT1,ACCT2 ACCT.LIS
```

この SORT コマンドにより、指定されたキーに従って、2 つの入力ファイルが整列され、同一のキーを持つ複数レコードは、1 つのレコードを除いてすべて消去される。

/KEY=(POSITION:n,SIZE:n[,field,...])

位置、サイズ、ソート順 (ASCENDING または DESCENDING)、優先順位 (NUMBER:n)、データ型 (文字、バイナリ、h_floating など) を含むキー・フィールドを記述する。省略時の設定では、文字データを持つレコード全体を昇順にソートすることにより、ファイルの並べ替えが行われる。

/KEY 修飾子についての詳細は、第 9.2.1 項を参照。

/PROCESS=type

(SORT コマンドのみに適用。) 内部ソート・プロセスを定義する。/PROCESS 修飾子により、レコード、タグ、アドレス、索引の4つのプロセスのうち1つを選択することができる。(高性能 Sort/Merge ユーティリティでは、レコード・プロセスのみがサポートされる。タグ、アドレス、索引の各プロセスについては、OpenVMS Alpha の将来のリリースでサポートされる予定。)

/PROCESS 修飾子についての詳細は、第 9.2.6 項を参照。

例

```
$ SORT/KEY=(POS:40,SIZ:2,DESC)/PROCESS=TAG YREND AVG.DAT -  
_ $ DESCYRAVG.LIS
```

この Sort 操作では、タグ・ソート・プロセスが使用され、出力ファイル DESCYRAVG.LIS が作成される。

/SPECIFICATION=filespec

(高性能 Sort/Merge ユーティリティでは、この修飾子はサポートされていない。この機能は、OpenVMS Alpha の将来のリリースでサポートされる予定。)

Sort 操作または Merge 操作で使用する Sort 指定ファイルまたは Merge 指定ファイルを識別する。省略時のファイル・タイプは.SRT になる。

指定ファイルの使用方法については、第 9.7 節および第 9.9.3 項を参照。

/[NO]STABLE

省略時の設定では、同一のキーを持つレコードは、出力ファイルで入力ファイルの順序と同じになるとは限らない。/STABLE 修飾子を使用すると、レコードが同じ順序のまま保持される。

/STABLE 修飾子と/NODUPLICATES 修飾子は、相互に排他的な関係にある。

例

```
$ SORT/KEY=(POS:1,SIZ:5,DECIMAL)/STABLE PRICESA.DAT, -  
_ $ PRICESB.DAT,PRICESC.DAT SUMMARY.LIS
```

この Sort 操作では、同一のキーを持つレコードは、PRICESA.DAT にあるものが最初にリストされ、その後 PRICESB.DAT のレコード、さらに PRICESC.DAT のレコードがリストされる。

/[NO]STATISTICS

統計情報の要約が SYS\$OUTPUT に出力され、これを最適化に使用することができる。これらの統計情報をファイルに保存するときは、次のコマンドを使用する。

```
$ DEFINE/USER SYS$ERROR output-file
```


統計情報の要約には、次の情報が含まれる。

統計情報	説明
Records read	Sort 操作または Merge 操作によって読み込まれたレコードの数。
Records sorted	Sort を使用したときに処理されたレコードの数。Sort 操作または Merge 操作で、指定ファイルを使用して特定のレコードのみを選択した場合、この数値が、読み込まれたレコードの数より少なくなることもある。
Records output	出力ファイルに書き込まれたレコードの数。/NODUPPLICATES が選択された場合や、出力レコードが書き込まれたときに入出力エラーが発生した場合、この数値が、ソートされたレコードの数より少なくなることもある。
Working set extent	プロセスのワーキング・セット拡張領域のページの数。この値は、ソート・データ構造体のサイズの上限に使用される。この値を調節することにより、Sort 操作の効率を向上させることができる。
Virtual memory	データを保持するソート・イメージに追加される仮想記憶のページ数。
Direct I/O + buffered I/O	この合計値は、データの読み込みおよび書き込みに必要な入出力移動の数になる。この合計値を小さくするほど、整列操作の効率は向上する。
Page faults	データがメモリに収まる度合いを示す値。この値が大きいくほど、整列操作の効率は低下する。
Elapsed time	Sort 操作または Merge 操作で使用される、時間、分、秒、1/100 秒単位の合計クロック時間。
Input record length	この値は、ユーザが指定しない場合レコード管理サービス (OpenVMS RMS) から取得される。
Internal length	内部フォーマット・ノードの、バイト単位のサイズ。これには、キー、データ、長さを保管するワード、長さ、レコード・ファイル・アドレス (RFA)、変換されたキーなどが含まれる。
Output record length	出力レコードの長さ。この長さは入力レコードの長さ、ソート・プロセス、必要なレコードの再フォーマットにより計算される。
Sort tree size	Sort の内部データ構造体に収まるレコードの数。
Number of initial runs	データがメモリに収まる程度を示す 1 つの指針。
Maximum merge order	同時にマージされるソート文字列の最大数。
Number of merge passes	ソートされた出力文字列が作成されるまで、Sort ユーティリティが文字列をマージする回数。Number of initial runs と Number of merge passes により、データがメモリに収まる程度が示される。これらの値が大きいくほど、ワーキング・セットのサイズにデータが収まらなくなり、同時にソートに時間がかかるようになる。
Work file allocation	作業ファイルに使用されるブロックの数。マージ・パスが複数必要な場合、このサイズは、入力ファイル割り当てのサイズの約 2 倍になる。
Elapsed CPU	整列操作で使用される CPU 時間。入出力オペレーションが終了するのを待つ時間や、別のプロセスが動作しているときにそれを待っている時間などは含まれない。

例

\$ SORT/STATISTICS PRICE1.DAT,PRICE2.DAT PRICE.LIS

SORT /STATISTICS コマンドを使用すると、次のような統計情報が表示される。

```
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 コマンドのみに適用。) 1 から 10(高性能 Sort/Merge ユーティリティの場合は、255 ファイルまでサポート) までの任意の数だけ、Sort の作業ファイルの数を増加させ、それぞれの作業ファイルを小さくする。使用可能なディスクが小さすぎる場合、または作業ファイルの大きさと比べてあまりに余裕がない場合、ファイルの数を増やすことにより、Sort 操作の効率を向上させることができる。

作業ファイルは、必要にならない限り作成されない。作業ファイルが必要になった場合は、省略時の設定として、2つのファイル (SORTWORK0, SORTWORK1) が作成され、SYS\$SCRATCH ディレクトリに置かれる。

例

```
$ ASSIGN DRA5: SORTWORK0
$ ASSIGN DB0: SORTWORK1
$ ASSIGN DB1: SORTWORK2
$ SORT/KEY=(POS:1,SIZ:80)/WORK_FILES=3 -
_$ STATS1,STATS2,STATS3,STATS4 SUMMARY.LIS
```

この Sort 操作では、入力ファイルが大きなファイルであるため、作業ファイルを 3 つ指定することにより、ソート操作の効率を向上させることができる。

ディレクトリ名を中に含めることにより、作業ファイルをデバイス上の特定のディレクトリに割り当てることもできる。たとえば、SORTWORK0 を DRA5 の [WORKSPACE] ディレクトリに割り当てるとき、次のコマンドを入力する。

```
$ ASSIGN DRA5:[WORKSPACE] SORTWORK0
```

9.9.1 入力ファイル修飾子

次の入力修飾子は、SORT または MERGE コマンド行に入力ファイル指定を指定した直後に指定しなければなりません。

/FORMAT=(RECORD_SIZE:n,FILE_SIZE:n)

入力ファイルの特性を定義する。レコード・サイズやファイル・サイズを指定または変更できる。入力ファイル修飾子は、Sort または Merge コマンド行の出力ファイル指定の直後に指定します。

Sort では、必要なメモリ量を判断するとき、Sort 操作に使用する作業ファイルのサイズの他に、入力ファイル・サイズの情報が使用される。このファイル・サイズが未知の場合 (たとえばディスクや標準 ANSI 磁気テープ上にないファイルをソートしている場合)、非常に大きなファイル・サイズが想定される。

次の修飾子の値を指定することができる。

RECORD_SIZE:n	入力ファイルの最長レコード長 (LRL) をバイト単位で指定する。指定できる最長のレコード長は、ファイル編成により異なる。
	順次 32,767
	相対 16,383
	索引順次 16,362
	これらの値には、固定長制御付き可変 (VFC) 形式レコードの制御バイトが含まれる。
FILE_SIZE:n	入力ファイル・サイズをブロック単位で指定する。指定できる最大ファイル・サイズは、4,294,967,295 ブロックになる。

出力ファイルの修飾子としても/FORMAT を使用することができる。詳細は第 9.9.2 項を参照。

例

```
$ SORT/KEY=(POS:40,SIZ:2,DESC) -  
_$_CRA0:YREND AVG.DAT/FORMAT=(RECORD_SIZE:41,FILE_SIZE:3) -  
_$_DESCYRAVG.LIS
```

入力ファイル YREND AVG.DAT は、ディスク・デバイス上にも ANSI 磁気テープ上にもないため、ファイル編成を/FORMAT 修飾子で記述する必要がある。

9.9.2 出力ファイル修飾子

次の出力修飾子は、SORT および MERGE コマンドに対して使用できます。出力ファイル修飾子を使用するには、SORT または MERGE コマンド行の出力ファイル指定の直後に指定します。

/ALLOCATION=n

最適化のための出力ファイルに事前に割り当てておくブロック数に、1 から 4,294,967,295 の値を指定する。出力ファイルの割り当てと入力ファイルの割り当てとが大きく異なることが分かっている場合 (たとえば、データのフォーマットを変更する場合やレコードを省略する場合) にこの修飾子を使用する。

/CONTIGUOUS 修飾子如果使用されている場合、/ALLOCATION 修飾子が必要になる。

例

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT -  
_ $ SUMMARY.LIS/ALLOCATION=1000/CONTIGUOUS
```

この SORT コマンドにより、出力ファイル SUMMARY.LIS に 1000 の連続ブロックが割り当てられる。

/BUCKET_SIZE=n

最適化のために、相対編成出力ディスク・ファイルや索引順編成出力ディスク・ファイルで使用する、OpenVMS RMS バケット・サイズ (バケットあたりの 512 バイト・ブロック数) を指定する。1 から 32 までの値が指定できる。

出力ファイルの編成が入力ファイルと同じ場合、省略時の値は、最初の入力ファイルのバケット・サイズと同じ値になる。出力ファイルの編成が異なる場合、省略時の値は 1 になる。

例

```
$ SORT/KEY=(POS:1,SIZ:80) STATS1.DAT,STATS2.DAT -  
_ $ SUMMARY.LIS/BUCKET_SIZE=16/RELATIVE
```

この SORT コマンドにより、バケット・サイズが 16 の出力ファイル SUMMARY.LIS が、相対編成で作成される。

/CONTIGUOUS

出力ファイルを連続するディスク・ブロックに格納してアクセス時間を短縮することを要求する。/ALLOCATION 修飾子と一緒に使用しなければならない。省略時の設定では、Sort/Merge は、出力ファイルに連続ディスク・ブロックを割り当てない。

例

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT -  
_ $ SUMMARY.LIS/ALLOCATION=1000/CONTIGUOUS
```

この SORT コマンドにより、出力ファイル SUMMARY.LIS に 1,000 の連続ブロックを割り当てる。

/FORMAT=(type:n[,...])

出力ファイルのレコード形式が入力ファイル形式と異なるときに、出力ファイルのレコード形式 (FIXED:n, VARIABLE:n, CONTROLLED:n) を指定する。ファイル・レコードのサイズ (SIZE:n) またはブロック・サイズ (BLOCK_SIZE:n) も指定できる。

Sort 操作がレコード・ソートまたはタグ・ソートの場合、省略時の出力レコードの形式は、最初の入力ファイル・レコードの形式と同じになる。Sort 操作がアドレス・ソートまたは索引ソートの場合、省略時の出力レコードの形式は、固定レコード形式になる。入力ファイルにさまざまなレコード形式がある場合、出力レ

コード・サイズは、入力ファイルにある最大のレコードを収められるだけの大きさになる。

次の修飾子の値を指定することができる。

BLOCK_SIZE:n	ファイルを磁気テープに出力する場合、出力ファイルのブロック・サイズをバイト単位で指定する。入力ファイルがテープ・ファイルの場合、出力ファイルのブロック・サイズは、省略時の設定で入力ファイルと同じサイズになる。それ以外の場合、出力ファイルのブロック・サイズは、省略時の設定でテープがマウントされたときに使用されるサイズになる。 nの値は、20 から 65,532 のいずれかになる。ただし、弊社の他のシステムとデータ交換を正しく行えるようにするためには、512 バイト以上のブロック・サイズは指定できない。また弊社製以外のシステムと互換性を保つためには、そのブロック・サイズに、2,048 バイト以下の値を指定するようにする。
CONTROLLED:n	出力ファイルに、固定長制御付き可変 (VFC) レコードを指定する。
FIXED:n	出力ファイルに固定長レコードを指定する。
SIZE:n	VFC (CONTROLLED) レコードの固定部分のサイズを、255 バイト以下のバイト数で指定する。SIZE を指定していない場合、省略時の値として、最初の入力ファイルの固定部分のサイズになる。サイズに 0 を指定した場合、OpenVMSRMS は、省略時の値として、2 バイトになる。
VARIABLE:n	出力ファイルに可変長レコードを指定する。

オプションとして、上記の修飾子の値に、出力レコードの最大レコード・サイズnを (バイト単位で) 指定することができる。指定できる最大レコード・サイズは、ファイル編成によって異なる。

順次ファイル	32,767
相対ファイル	16,383
索引順次ファイル	16,362

これらの最大レコード・サイズの値には、固定長制御付き可変 (VFC) 形式レコードの制御バイトが含まれる。

例

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT SUMMARY.LIS/FORMAT=FIXED:80
```

入力ファイル STATS.DAT は、長さ 80 バイトの可変長のレコードで構成される。/FORMAT 修飾子には、固定長レコードで構成される出力ファイル SUMMARY.LIS を指定する。

/INDEXED_SEQUENTIAL

出力ファイルのファイル編成を索引順編成として定義する。出力ファイルはあらかじめ存在し、かつ空でなければならない。また、空のファイルは、/OVERLAY 修飾子を使用してソート済みレコードが上書きされるように指定しなければならない。

例

```
$ CREATE/FDL=NEW.FDL AVERAGE.DAT  
$ SORT/KEY=(POS:1,SIZ:80) DATA.DAT,STATS.DAT -  
_$ AVERAGE.DAT/INDEXED_SEQUENTIAL/OVERLAY
```

CREATE/FDL コマンドを使用することにより、空のファイル AVERAGE.DAT が作成される。SORT コマンドは、出力ファイルが索引順次編成になり、空のファイル AVERAGE.DAT に書き出されることを指定する。

/OVERLAY

出力ファイルによってオーバーレイされたり、直接書き込まれたりする、既存の空のファイルを指定する。/INDEXED_SEQUENTIAL 修飾子を使用するとき、/OVERLAY 修飾子が必要になる。

入力ファイル編成が索引順次になっている場合、出力ファイルがすでに存在しており、空になっている必要がある。出力ファイルが空でない場合も、/OVERLAY により、そのファイルが上書きされることはない。その代わりに、ソートの結果が既存の出力ファイルに追加される。

CREATE/FDL ユーティリティを使用すると、空のデータ・ファイルを作成することができる。空のファイルを作成するときに指定する属性は、その後 Sort の出力ファイルの属性になる。

例

```
$ CREATE/FDL=NEW.FDL AVERAGE.DAT  
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT AVERAGE.DAT/OVERLAY
```

FDL ファイル NEW.FDL は、ファイル AVERAGE.DAT に対して特殊な属性を指定する。そのファイルに出力が書き込まれるとき、Sort の出力ファイルは、FDL ファイルによって指定された属性になる。

/RELATIVE

出力ファイルのファイル編成を相対編成として定義する。

例

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT SUMMARY.LIS/RELATIVE
```

入力ファイル STATS.DAT が相対ファイルでなく、出力ファイル SUMMARY.LIS が相対ファイルになるため、/RELATIVE で、出力ファイル指定を修飾する。

/SEQUENTIAL

出力ファイルのファイル編成を順編成として定義する。アドレス・ソート操作および索引ソート操作の場合、これが省略時の設定になる。レコード・ソート操作およびタグ・ソート操作の省略時の設定は、最初の入力ファイルの編成になる。

例

```
$ SORT/KEY=(POS:1,SIZ:80) STATS.DAT SUMMARY.LIS/SEQUENTIAL
```

入力ファイル STATS.DAT が順次ファイルでなく、出力ファイル SUMMARY.LIS が順次ファイルになるため、/SEQUENTIAL で、出力ファイル指定を修飾する。

9.9.3 指定ファイル修飾子

次の修飾子は指定ファイルの中で使用されます。(高性能 Sort/Merge ユーティリティでは、指定ファイルはサポートされていません。この機能は、OpenVMS Alpha の将来のリリースでサポートされる予定です。)これらの修飾子は Sort/Merge 指定ファイルの中でのみ有効であることに注意してください。

/CDD_PATH_NAME="cdd-path-name"

CDD/Repository コマンドを使用して、共通データ・ディクショナリ (CDD/Plus) で使用するために定義されたフィールドと属性を識別する。一度フィールドを識別したら、/KEY、/CONDITION、/INCLUDE、/OMIT などの他の指定ファイル修飾子でもそのフィールドを使用できる。

/CDD_PATH_NAME は、/FIELD 文の代わりに使用したり、/FIELD 文と一緒に使用したりすることができる。

"cdd-path-name"の値は、CDD/Plus 内の CDD/Plus レコード定義になる。
/CDD_PATH_NAME 修飾子は、システムに CDD/Plus がインストールされている場合のみ使用できる。

例

```
/CDD_PATH_NAME="employee"
```

/CDD_PATH_NAME 修飾子は、前に CDD/Plus で定義されている employee レコードを識別する。

/[NO]CHECK_SEQUENCE

(MERGE コマンドでのみ適用。)入力ファイル内のレコードの順序をチェックするかどうか指定する。省略時の設定では、レコードの順序をチェックする。

例

```
/NOCHECK_SEQUENCE
```

/NOCHECK_SEQUENCE 修飾子は、Merge ユーティリティの省略時の動作を指定変更する。

```
/COLLATING_SEQUENCE=(SEQUENCE=sequence-type  
[,MODIFICATION=("char1" operator "char2")]  
[,IGNORE=文字または文字範囲,...]  
[,FOLD]
```

[,[NO]TIE_BREAK))

文字キー・フィールドに定義済みの 3 つの照合シーケンス ASCII, EBCDIC, MULTINATIONAL のうちいずれか 1 つ, またはユーザが定義するシーケンスを指定する。定義済みの照合シーケンスまたは前もって定義されたユーザ定義シーケンスを変更できるようにする。

ASCII, EBCDIC, MULTINATIONAL の各照合順序についての詳細は, 第 9.3 節を参照。

次の修飾子の値を指定することができる。

SEQUENCE	指定ファイルは, ASCII, EBCDIC, MULTINATIONAL の各照合順序の他, ユーザ定義照合順序もサポートする。これらの照合順序についての詳細は, 第 9.3 節を参照。
MODIFICATION	<p>SEQUENCE オプションで指定した照合順序を変更するよう指定する。ASCII, EBCDIC, MULTINATIONAL の各照合順序の他, ユーザ定義照合順序も変更することができる。たとえ順序が省略時の値 (ASCII) であっても, 変更する順序は, SEQUENCE 修飾子で指定されているものでなければならない。</p> <p>character 照合順序の文字を指定する。</p> <p>operator 文字を比較するときに使用する演算子を指定する。大なり (>), 小なり (<), 等号 (=) を指定することができる。</p> <p>MODIFICATION オプションでは, 次の種類の変更が認められている。</p> <ul style="list-style-type: none">- 単一文字または二重文字が, すでに照合値の割り当てられている単一文字と等しくなる ("a"="A")。- 単一文字または二重文字が, すでに照合値の割り当てられている単一文字の後に照合される ("CH">"C")。- 単一文字または二重文字が, すでに照合値の割り当てられている単一文字の前に照合される ("D"<"A")。- 二重文字が, 前に定義されている二重文字と等しくなる ("CH"="SH")。- 単一文字が, 二重文字の順序と等しくなる ("C" = "CH")。
IGNORE	最初に比較を行うとき, 照合順序内の 1 つまたは複数の文字が無視されるよう指定する。タイブレークが発生するとき, IGNORE 値で指定されている文字についても考慮される。
FOLD	<p>すべての小文字に対して, 大文字に相当する照合値が定義されるよう指定する。ASCII, EBCDIC, ユーザ定義順序の場合, 小文字は a から z になる。</p> <p>MULTINATIONAL 順序の小文字には, 大文字に相当する照合値がすでにあるため, FOLD を使用する必要はない。</p>
[NO]TIE_BREAK	<p>同等の値を持つ文字でタイブレークの比較を行うとき, 数値を使用するかどうか指定する。省略時の場合, MULTINATIONAL 順序ではタイブレークが発生する。NOTIE_BREAK を指定すると, この省略時の設定が指定変更され, 最初の比較が行われた後は比較が行われなくなる。</p> <p>ASCII, EBCDIC, ユーザ定義順序でタイブレークが発生するようにしたい場合, TIE_BREAK オプションを指定する必要がある。これらの順序で FOLD 値または MODIFICATION 値を指定するとき, TIE_BREAK を使用する。</p>

例

指定ファイルにおける照合順序の使用例については、第 9.3 節および第 9.7 節を参照。

```
/CONDITION=(NAME=condition-name,  
TEST=(field-name operator test-condition  
[logical-operator...]))
```

レコードの相対順序を変更したり、レコード内の特定のフィールドの内容を変更したりするときは、指定ファイルを使用することができる。最初に/CONDITION 修飾子で条件テストを定義する。/CONDITIONAL 修飾子を使用してテストを定義したら、レコードの順序を変更するときに、/KEY 修飾子や/DATA 修飾子でその同じテストを使用することができる。またレコードの内容を変更するときに、/OMIT 修飾子や/INCLUDE 修飾子でこのテストを使用することもできる。

出力ファイルのレコードの順序を変更したい場合、最初に/CONDITION 修飾子で条件名を指定し、その条件に適合するかどうかを試すテストを設定する。次にその形式の/KEY 修飾子で相対順序を指定する。

```
/KEY=(IF condition-name THEN value ELSE value)
```

レコードの相対順序を指定するとき、任意の値を使用することができる。

/CONDITION 修飾子を使用すると、出力レコードのフィールドの内容も変更することができる。最初に条件名を指定し、次にその条件に適合するかどうかを試すテストを設定する。その形式の/DATA 修飾子で、フィールドに入れる内容を指定する。

```
/DATA=(IF condition-name THEN "new-contents"  
ELSE "new-contents")
```

次の修飾子の値を指定することができる。

NAME テストする条件の名前を指定する。この条件名は、/CONDITION 修飾子で定義した後、/KEY、/DATA、/OMIT、/INCLUDE のそれぞれの修飾子で使うことができる。

TEST	条件テストを指定する。
field-name	テストするフィールドの名前を指定する。このフィールド名は、/FIELD 修飾子であらかじめ定義しておく必要がある。
operator	条件テストで使用する論理演算子または相対演算子を指定する。使用できる論理演算子には、AND と OR がある。指定できる相対演算子には、次のようなものがある。 EQ =等しい NE =等しくない GT =より大きい GE =より大きいか等しい LT =より小さい LE =より小さいか等しい
test-condition	テストを行う、定数またはフィールド名を指定する。定数は次の形式で指定する。 Decimal_digits (省略時) %Ddecimal_digits %Ooctal_digits %Xhexadecimal_digits "character" 基数演算子 (%D) の場合、通常指定する必要はない。ただしテスト条件では、フィールド名と同じデータ型が想定される。

例

指定ファイルにおける/CONDITION 修飾子の使用例については、第 9.7 節を参照。

```
/DATA=field-name
/DATA=(IF condition THEN "new contents"
ELSE "new contents")
```

出力レコードからフィールドを除去したり、並べ替え直したりするときは、/DATA 修飾子を使用する。出力レコードに表示させたい順序でデータ・フィールドを指定する。/DATA 修飾子では、出力フィールドに指定する、レコード内のすべてのフィールドを識別する必要がある。/DATA 修飾子で識別されたフィールドのみが、出力ファイルに出力される。

/CONDITION 修飾子で設定した条件を指定することにより、出力レコードのフィールドの内容を条件によって変更することができる。/DATA 修飾子のフィールドに設定したい内容を指定する。

```
/DATA=(IF condition-name THEN "new-contents" ELSE "new-contents")
```

次のような修飾子の値を設定できる。

field-name	レコードのフィールド名を指定する。フィールド名は事前に/FIELD 修飾子で定義されていないなければならない。
condition-name	事前に/CONDITION 修飾子で定義されている条件名を指定する。
new-contents	レコードをどのように変更するかを指定する。定数値でも/FIELD 修飾子で定義されたフィールド名でも可。

例

指定ファイル内での/DATA 修飾子の使用例については、第 9.7 節を参照。

```
/FIELD=(NAME=field-name,POSITION:n,SIZE:N, [DIGITS:n,]data-type)
```

```
/FIELD=(NAME=field-name,VALUE:n,SIZE:N,[DIGITS:n,] data-type)
```

出力レコードのフィールドの順序を変更したりフォーマットを変更する場合は、入力ファイルのフィールドを定義する。これらのフィールドには、キー・フィールド、比較されるフィールド、出力ファイルに転送されるフィールドがある。それぞれのフィールドは、その名前、レコード上での位置とサイズ、データ・タイプを指定することにより識別する。

フィールド名は一意でなければならない。また、フィールド定義の数は 255 以内でなければならない。

また、/FIELD 修飾子を使用して定数値を定義し、/CONDITION、/DATA、および/KEY 文で使用するデータ・タイプの値に割り当てることができる。

次のような修飾子の値を指定できる。

NAME	フィールド名を指定する。フィールド名にはスペースを含めることはできない。英字で始まり、31 文字以内でなければならない。
POSITION:n	レコード内でのこのフィールドの位置を指定する。
VALUE:n	/CONDITION、/DATA、/KEY 文で使われる定数フィールドに値を割り当てる。VALUE:n を指定した場合、/POSITION:n を指定してはいけない。これは、このフィールドは定数フィールドであり、入力レコードのフィールドではないからである。
SIZE:n	文字あるいはバイナリ・データを含むフィールドのサイズをバイト数で指定する。データ・タイプにより、以下のように指定可能なサイズが決まっている。 <ul style="list-style-type: none">文字データ — 32,767 以内バイナリ・データ — 1, 2, 4, 8, 16浮動小数点データ — サイズは指定しない
DIGITS:n	10 進データを含むフィールドのサイズを指定する。ここで指定するサイズは 31 以内でなければならない。DIGITS:n は 10 進データを含んだフィールドを表現するときのみ使用する。
data-type	データ・タイプを指定する。文字型の場合は指定する必要はない。省略時の設定では、Sort は文字データ・タイプを想定する。Sort/Merge により認識されるデータのタイプについては第 9.2.1 項を参照。

例

```
/FIELD=(NAME=SALARY,POSITION:10,DIGITS:8,DECIMAL)
```

この/FIELD 修飾子により、レコード内のこのフィールドは SALARY により認識され、フィールドは 10 バイト目から 8 桁の長さ、10 進データであることが指定される。

```
/INCLUDE=(CONDITION=condition[,KEY=...] [,DATA=...])
```

条件により、あるフィールドを出力レコードに出力することを指定することができる。/CONDITION 修飾子により条件を定義したのち、/INCLUDE 修飾子で選

択するレコードを定義する。/INCLUDE 修飾子は、出力ファイルに出力すべき条件を満たすレコードを選択する。

1つのファイルに対して、複数の/INCLUDE、/OMIT 修飾子を指定することができる。それらを指定した順序により、入力レコードがテストされる。最後の/INCLUDE 修飾子のあとで、選択対象とならなかったレコードまたは明示的に削除されたレコードがすべて削除される。

事前に削除されなかったレコード、または条件を指定せずに/INCLUDE 修飾子を指定することにより選択されたレコードを、出力に含めることもできます。

複数の形式のレコードをソートする場合は、ソートするそれぞれの形式のレコード形式に対して/INCLUDE 修飾子を指定する。/INCLUDE 修飾子で KEY オプションを指定しない場合は、Sort は省略時のキー定義を使用する。/INCLUDE 修飾子で KEY オプションを指定した場合は、省略時のキー定義は使用されない。/INCLUDE 修飾子の KEY フィールドの順序により、ソートのための内部キーが作成される。/INCLUDE 修飾子の DATA フィールドの順番により、出力レコードのフォーマットが決定される。/INCLUDE 修飾子で1つでも KEY フィールドまたは DATA フィールドを指定した場合は、出力したいすべての KEY フィールドまたは DATA フィールドを指定する必要がある。

以下のような修飾子の値を指定できる。

CONDITION	/CONDITION 修飾子で既に定義されている条件名を指定する。
KEY	/KEY 修飾子で定義された省略時のレコード・タイプが使用されないで、キー・フィールドを定義する。
DATA	/DATA 修飾子で定義された省略時のレコード・タイプが使用されないで、データ・フィールドを定義する。

例

```
/FIELD=(NAME=ZIP,POSITION:20,SIZE:6)
/CONDITION=(NAME=LOCATION,
            TEST=(ZIP EQ "01863"))
/INCLUDE=(CONDITION=LOCATION)
```

/CONDITION および/INCLUDE 修飾子を使用して、出力ファイルに zip コードが 01863 のレコードを含めることを指定している。

```
/KEY=field-name
/KEY=(field-name,order)
/KEY=([IF condition THEN value ELSE]...) value [,order]
```

ソート処理で使用するキー・フィールドを指定する。文字データを使用しているレコードすべてをソートする場合は、キー・フィールドを指定する必要はない。そうでない場合は、その優先順位に従って、それぞれのキーを/KEY 修飾子で指定する。最大 255 キーでソートすることができる。

3種類の/KEY 修飾子の使用方法がある。

- キー・フィールド名を指定する。

- キー・フィールド名，およびソート順を指定する。この場合，フィールド名およびソート順は括弧で囲む。
- 条件修飾子のように，出力ファイルのレコード順序を変更する。まず /CONDITION 修飾子で条件名を指定し，この条件を照合するための準備をする。その後，/KEY 修飾子でソートの前後関係を指定する。

```
/KEY=(IF condition-name THEN value ELSE value)
```

レコードの相対順序を指定するするためには，任意の値を使用することができる。

以下の修飾子の値を指定できる。

field-name	キー・フィールド名を指定する。フィールド名は，事前に/FIELD 修飾子で定義されている。
order	ソート順を指定する。ASCENDING は昇順，DESCENDING は降順にソートする。省略時の設定は ASCENDING。
value	キーを指定する。値は，定数値でも/FIELD 修飾子で定義されたフィールド名でも可。

例

1.

```
/FIELD=(NAME=SALARY,POSITION:10,DIGITS:8,DECIMAL)
/KEY=(SALARY,DESCENDING)
```

この/KEY 修飾子は，キーは SALARY で，降順にソートすることを示す。

2.

```
/FIELD=(NAME=ZIP,POSITION:20,SIZE:6)
/CONDITION=(NAME=LOCATION,
            TEST=(ZIP EQ "01863"))
/KEY=(IF LOCATION THEN 1
      ELSE 2)
```

この例では，zip コードが 01863 のすべてのレコードがソート済みの出力ファイルの最初にくる。/CONDITION で定義された条件 LOCATION は，ZIP フィールドが判定の対象であることを示す。この/KEY 句における 1 と 2 は，この条件を満たすレコードとそうでないレコードの順序を明示する。

```
/OMIT=(CONDITION=condition-name)
```

/CONDITION 修飾子で定義された条件に基づいて，出力ファイルから除外するレコードを指定する。

はじめに/CONDITION 修飾子で条件を定義しなければならない。ソート処理から除外される条件を満たすレコードを指定するために，/OMIT 修飾子を指定する。省略時の設定では，条件に該当しないすべてのレコードは出力ファイルに出力される。

指定ファイルで、複数の/OMIT と/INCLUDE 修飾子を指定できる。指定の順序により、除外のための入力レコードのチェックの順序が決定される。まだ出力されていないすべてのレコード、または最後の/OMIT 修飾子の後まで除外されていないすべてのレコードが出力される。/OMIT 修飾子のみを指定することにより、除外されていなかったレコードを無条件に取り除くこともできる。

例

```
/FIELD=(NAME=ZIP,POSITION:20,SIZE:6)
/CONDITION=(NAME=LOCATION,
            TEST=(ZIP EQ "01863"))
/OMIT=(CONDITION=LOCATION)
```

この/CONDITION と/OMIT 修飾子は、zip コードが 01863 のレコードを出力ファイルに出力しないことを指定している。

/PAD=single-character

レコード形式を変更したり、長さの等しくないレコードを比較する際に、領域を拡張するために埋め込む文字を指定する。省略時の設定では、前バージョンとの互換性を保証するために空 (null) を使用する。1 文字として定義可能な 2 文字 ("ch" > "c") は、埋めこみ文字として使用できない。文字、10 進数、8 進数、16 進数が使用可。

埋めこみ文字は以下のように指定する。

- 文字に対しては引用符を使用する。たとえば " # " は番号記号を指定する。
- 10 進数には 10 進記号を使用する。たとえば %D35 は 10 進数の 35 を示す。
- 8 進数には 8 進記号を使用する。たとえば %O043 は 8 進数の 043 を示す。
- 16 進数には 16 進記号を使用する。たとえば %X23 は 16 進数の 23 を示す。

例

```
/PAD="."
```

この/PAD 修飾子の例では、レコードをピリオドで埋め込むことを指定している。

/PROCESS=type

(SORT コマンドにのみ適用される。) ソート・プロセスの方法 (レコード、タグ、アドレス、または索引) を指定する。出力されたレコードを再書式化する場合、アドレス・ソートやインデックス・ソートを使用することはできない。プロセス・タイプには RECORD、TAG、ADDRESS、INDEX のいずれかを指定する。

4 タイプのプロセスの比較については第 9.8.1 項を参照。

例

```
/PROCESS=tag
```

この/PROCESS 修飾子の例では、タグ・ソート・プロセスを使用することを指定している。

/[NO]STABLE

同じキーのレコードがあった場合、出力ファイルに出力する順序を指定する。省略時の設定は/NOSTABLE。

省略時の設定では、同一キーでレコードがソートされた場合、出力ファイルでのレコードの順序は、必ずしも入力ファイルでの順序と同じではない。指定ファイルで/STABLE 修飾子を指定すると、キーの同じレコードがあった場合、入力ファイルの順序で出力ファイルに出力される。複数の入力ファイルがある場合にこの修飾子を使用した場合、キーが同じレコードは、1 番目の入力ファイルが最初に、次に 2 番目の入力ファイルのレコードが出力される。

例

```
/STABLE
```

この/STABLE 修飾子の例では、キーが同じレコードがあった場合、出力ファイルには入力ファイルと同じ順序で整列されることが保証される。

/WORK_FILES=(device[,...])

(SORT コマンドにのみ適用される。) 処理速度を向上させるために、異なったディスク構造のデバイスに作業ファイルを割り当てる。作業ファイルで/WORK_FILES 修飾子を使用することにより、コマンドあるいはプログラムレベルでソートを呼び出す前に論理名を割り当てる必要がなくなる。

DCL での修飾子/WORK_FILES=n と異なり、指定ファイル修飾子/WORK_FILES=(device[,...]) は、作業ファイルの数ではなく作業ファイルの割り当てを指定する。

作業ファイルについての詳細は、第 9.8.3 項を参照。

例

```
/WORK_FILES=( "WRKD$:" )
```

この/WORK_FILES 修飾子の例では、ソートの作業ファイルの 1 つをデバイス WRKD\$を割り当てる。これは、このデバイスがもっとも使用可能スペースが多いからである。

資源へのアクセスの制御

システムにはそれぞれ、独自のセキュリティに関する要件があります。このため、システムごとに、システム管理者とユーザに対する物理的およびソフトウェア的なセキュリティ要件を示すシステム・セキュリティ方針を明らかにしておく必要があります。システム・セキュリティを確実にするために、OpenVMS オペレーティング・システムでは、システムへのアクセスと、共用可能情報を登録したオブジェクトへのアクセスの両方を制御します。デバイスやボリューム、論理名テーブル、ファイル、キューなどのオブジェクトを保護されたオブジェクトと呼びます。保護されたオブジェクトはすべて、誰がそのオブジェクトに特定の方法でアクセスできるかを指定するアクセス条件一覧を定義しています。

オペレーティング・システムで使用可能なセキュリティ機能と、アカウントとシステムのセキュリティを保つためにシステム管理者が実行可能なタスクについては、『OpenVMS Guide to System Security』を参照してください。本章では、OpenVMS がシステム・リソースを保護し、監査する方法について説明します。特に次のことについて説明します。

- プロセスのライト識別子の表示
- オブジェクトのセキュリティ・プロファイル
- 保護コードの解釈
- 省略時のファイル保護
- ネットワークを通してのファイルのアクセス
- アカウントとファイルへのアクセスの監査

セキュリティについての補足説明は、以下の資料を参照してください。

- オブジェクトの保護およびシステム・セキュリティの一般的な説明は、『OpenVMS Guide to System Security』
- この章で説明するコマンドについての詳しい説明は、『OpenVMS DCL ディクショナリ』またはオンライン・ヘルプ

セキュリティ機能

たとえば、次の操作を通じてこれらの機能を理解することができます。

- プロセスに割り当てられているライト識別子を知ること。ライト識別子は、どの資源にアクセスできるかを決定する。プロセスに適切な識別子が割り当てられていない場合には、特定の保護されたオブジェクトにアクセスできないことがある。

ライト識別子の表示についての説明は第 10.1 節を参照。

- 保護されたオブジェクトのセキュリティ・プロファイルを表示すること。セキュリティ・プロファイルには、保護されたオブジェクトに関する情報が格納されている。自分で所有しているオブジェクトのセキュリティ・プロファイルは、他のユーザからアクセス可能またはアクセス不可能に変更できる。

セキュリティ・プロファイルについての説明は第 10.2 節を参照。

- ネットワークを介してファイルにアクセスする方法を知ること。このためには、アクセス制御文字列または代理ログイン・アカウントを使用する。

リモート・ファイルへのアクセスについての説明は、第 10.5 節を参照。

- アカウントとファイルへのアクセスを監査すること。このためには、ログイン・メッセージを詳しく調べ、システム管理者と協力してファイルを監査する。

アカウントとファイルへのアクセスの監査についての説明は第 10.6 節を参照。

10.1 プロセスのライト識別子の表示

保護されたオブジェクトにアクセスしようとするプロセスはすべて、ライト識別子と呼ぶ保護機能を実行します。保護されたオブジェクトはすべて、特定の方法でそのオブジェクトに誰がアクセスできるかを指定できるアクセス条件を指定しています。アクセスするプロセスのライト識別子がオブジェクトのライト識別子と一致しない場合には、アクセスは拒否されます。

次の例では、SHOW PROCESS コマンドを使用して、現在のプロセスの識別子を表示する方法を示しています。

```
$ 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
```

ライト識別子には、UIC、環境、汎用の3種類があります。SHOW PROCESS コマンドからの出力はこの3種類の識別子をすべて示しています。

- 1 UIC 識別子は、Greg というユーザが DOC グループのメンバであることを示している。
- 2 環境識別子は、Greg というユーザが会話型ユーザであることを示している。
- 3 環境識別子は、Greg というユーザがローカルからログインしたことを示している。
- 4 汎用識別子は、Greg というユーザがセールス・グループのメンバでもあることを示している。
- 5 汎用識別子は、Greg が資源属性を持つ MINDCRIME 識別子を保有しているため、識別子に対してディスク容量を要求できることを示している。
- 6 環境識別子は、Greg というユーザが ACCOUNTS ノードから作業していることを示している。

10.2 オブジェクトのセキュリティ・プロファイル

オペレーティング・システムは多くのユーザを同時にサポートするため、1人のユーザの操作が別のユーザによって妨害されないように保護するためのセキュリティ・メカニズムが組み込まれています。保護コード、アクセス制御、ハードウェア設計の組み合わせにより、多くのユーザがシステムを共用できるように、メモリ、共用可能装置、データの使用が保護されます。オブジェクトのセキュリティ・プロファイルは利用者識別コード (UIC)、ACL、そのオブジェクトに割り当てられた保護コードで構成されます。自分で所有しているオブジェクトのセキュリティ・プロファイルは表示したり、変更することができます。

保護されたオブジェクトのセキュリティ・プロファイルを表示するには、DCL の SHOW SECURITY コマンドを使用します。たとえば、次のコマンドは 95_FORECAST.TXT ファイルに関するセキュリティ情報を要求しています。

```
$ 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>
```

このコマンドからの表示を見ると、95_FORECAST.TXT ファイルが Greg というユーザによって所有されていることがわかります。また、ファイルの保護コードも示されます。保護コードはシステム・ユーザと所有者に対して読み込み、書き込み、実行、削除アクセス権を与えています。また、グループ・ユーザに対して読み込みアクセス権と実行アクセス権を与え、ワールド・ユーザに対してはアクセス権を与えていません（詳細は第 10.3 節を参照してください）。このファイルに対して ACL は設定されていません。

10.2.1 セキュリティ・プロファイルの変更

保護されたオブジェクトの所有者、保護コード、ACL に対して新しい値を指定でき、また、SET SECURITY コマンドを使用して 1 つのオブジェクトから別のオブジェクトにプロファイルをコピーすることもできます。

たとえば、第 10.2 節に示した SHOW SECURITY の表示結果では、95_FORECAST.TXT ファイルが Greg というユーザによって所有されていることがわかります。このユーザは所有者として、ファイルの保護コードを変更できます。もともと、この保護コードでは、ワールド・ユーザに対してアクセス権が与えられていませんでした。ここで Greg は保護コードを変更し、ワールド・ユーザに対して読み込みアクセス権と書き込みアクセス権を許可します。

```
$ SET SECURITY/PROTECTION=(W:RW) 95_FORECAST.TXT
```

SHOW SECURITY コマンドはファイルの新しい保護コードを確認します。

```
$ 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>
```

10.3 保護コードの解釈

保護コードは、特定のユーザまたはユーザ・グループに対して許可（または禁止）されるアクセス・タイプを制御します。保護コードの形式は次のとおりです。

[カテゴリ: 許可されるアクセスのリスト (, カテゴリ: 許可されるアクセスのリスト,...)]

カテゴリは、system (S)、owner (O)、group (G)、world (W) のいずれかです。各カテゴリは最初の 1 文字に短縮できる。各カテゴリの定義は次のとおりです。

System	UIC が 1 ~ 10 (8 進数) であるか、SYSPRV 特権を持つか、または所有者と同じグループに属し、GRPPRV を保有するユーザ・プロセスまたはアプリケーション。
Owner	オブジェクトの UIC と等しい UIC を持つユーザ・プロセスまたはアプリケーション。
Group	オブジェクトのグループ UIC と等しいグループ UIC を持つユーザ・プロセスまたはアプリケーション。
World	システムの任意のユーザ・プロセスまたはアプリケーション。

複数のユーザ・カテゴリを指定するときは、各カテゴリをコンマで区切り、コード全体を括弧で囲みます。ユーザ・カテゴリとアクセス・タイプはどの順序で指定してもかまいません。

アクセス指定としてヌルを指定した場合には、アクセスを許可しないことを示します。したがって、ユーザ・カテゴリに対してアクセス・タイプを省略すると、そのユーザ・カテゴリはそのタイプのアクセスを実行できません。ユーザ・カテゴリに対してすべてのアクセスを禁止するには、ユーザ・カテゴリだけを指定し、アクセス・タイプを省略します。ユーザ・カテゴリに対してアクセスを禁止するときは、ユーザ・カテゴリの後のコロンを省略します。

ファイルに対して、read (R)、write (W)、execute (E)、delete (D) を指定する。アクセス・タイプは各カテゴリに割り当てられます。カテゴリとアクセス・タイプの間はコロン (:) で区切ります。ファイル・アクセス・タイプの意味は次のとおりです。

Read	ディスク・ファイルの読み込み、印刷、コピーを許可する。ディレクトリ・ファイルに対して読み込みアクセス権がある場合には、ファイルを読み込むか、またはファイル・リストを表示でき、ワイルドカード文字を含むファイル名を使用してファイルを検索できる。読み込みアクセス権があるときは、実行アクセス権も与えられる。
Write	ファイルに書き込むか、またはファイルの内容を変更することはできるが、ファイルを削除することはできない。書き込みアクセス権がある場合には、ファイルの内容を記述するファイル属性を変更できる。ディレクトリ・ファイルに対する書き込みアクセス権がある場合には、ファイル・カタログにエントリを作成したり、エントリを削除することができる。
Execute	実行可能プログラム・イメージまたは DCL コマンド・プロシージャを格納したファイルを実行することができる。ディレクトリ・ファイルに対して実行アクセス権が割り当てられている場合には、名前がわかっているファイルを検索できる。
Delete	ファイルを削除することができる。ファイルを削除するには、ファイルへの削除アクセス権と、そのファイルが格納されているディレクトリへの書き込みアクセス権が必要である。

10.4 省略時のファイル保護

新しいファイルには省略時の UIC ベースの保護が与えられ、その親ディレクトリの省略時のアクセス制御リスト (ACL) が与えられます。ACL は、ユーザまたはユーザ・グループがファイルやディレクトリ、装置などの特定の保護されたオブジェクトに対して実行できるアクセス権を定義したエントリの集合です。

新しいファイルに割り当てられる省略時の UIC ベースの保護を変更するには、次のいずれかの操作を実行します。

10.4.1 省略時の UIC 保護

オペレーティング・システムは各プロセスに対して次の UIC ベースの保護を割り当てます。

```
(S:RWED, O:RWED, G:RE, W)
```

省略時の設定では、システム UIC を持つユーザとオブジェクトの所有者は、そのオブジェクトに対して完全なアクセスを実行でき、オブジェクト所有者と同じ UIC グループに属すユーザは、そのオブジェクトに対して読み込みアクセスと実行アクセスを実行でき、他のすべてのユーザはオブジェクトにアクセスできません。作成するファイルの省略時の保護を変更するには、SET PROTECTION コマンドと/DEFAULT 修飾子を使用します。たとえば、ログイン・コマンド・プロシージャに次のコマンドを登録しておけば、すべてのプロセスに対して作成するファイルへの読み込みアクセス権と実行アクセス権を割り当てることができます(このコマンドを実行するには、ログイン・コマンド・プロシージャを実行しなければなりません)。

```
$ SET PROTECTION = (S:RWED,O:RWED,G:RE,W:RE)/DEFAULT
```

10.4.2 省略時の ACL 保護

指定ディレクトリやサブディレクトリの省略時の UIC 保護を無効にするには、適切なディレクトリ・ファイルの ACL に省略時の保護アクセス許可制御エントリ (ACE) を配置します。ACE に指定する省略時の保護は、指定したディレクトリまたはディレクトリのサブディレクトリに作成される新しいファイルにすべて適用されます。後続の ACE (ディレクトリ・ファイルの ACL 内に指定しなければならない) は、そのディレクトリおよびディレクトリのサブディレクトリの省略時の保護により、システムおよび所有者プロセスに対して完全なアクセス権を割り当て、グループ・プロセスに対して読み込みアクセス権と実行アクセス権を割り当て、ワールド・ユーザに対してアクセス権を割り当てないことを指定します。

```
$ SET SECURITY/ACL = (DEFAULT_PROTECTION,S:RWED,O:RWED,G:RE,W:)  
[JONES]PERSONAL.DIR
```

ディレクトリにこの後作成されるファイルの ACL にコピーする省略時の識別子 ACE を指定するには、ディレクトリ・ファイルの識別子 ACL に DEFAULT オプションを指定します。

たとえば、次の ACE はディレクトリ・ファイルに適用され、ネットワーク・ユーザが、ディレクトリに作成されたすべてのファイルにアクセスすることを禁止します。

```
$ SET SECURITY/ACL = (IDENTIFIER=NETWORK,OPTIONS=DEFAULT,ACCESS=NONE) -  
_ $ [JONES]PERSONAL.DIR
```

10.4.3 ファイル名の変更

ファイル名を変更しても、そのファイルの保護は変更されません。既存のファイルの新しいバージョンには、前のバージョンと同じ ACL および UIC ベースの保護が割り当てられます (省略時の UIC ベースの保護を変更するには、BACKUP、COPY、CREATE、SET FILE コマンドの/PROTECTION 修飾子を使用します)。

10.4.4 明示的なファイル保護

/PROTECTION 修飾子を使用すれば (この修飾子は BACKUP、COPY、CREATE コマンドで使用可能)、新しいファイルに対して UIC ベースの保護を明示的に指定できます。

既存のファイルに対する UIC ベースの保護を変更するときは、SET SECURITY /PROTECTION コマンドを使用します。

ファイルを作成し、そのファイルの ACL を作成した後、ACL を変更し、必要な数だけ ACL にエントリを追加できます。ACL によって指定される保護は、ファイルのユーザ識別コード保護より優先します。

次の例では、UIC ベースの保護を指定しています。

```
$ CREATE MAST12.TXT/PROTECTION=(S:RWED,O:RWED,G,W)
```

次の例では、ファイル MAST12.TXT で UIC ベースの保護を変更しています。

```
$ SET SECURITY/PROTECTION=(S:RWED,O:RWED,G:RE,W) MAST12.TXT
```

10.5 ネットワークを介してのファイルへのアクセス

これ以降の節では、ネットワークを介してのファイルへのアクセスについて説明します。

10.5.1 アクセス制御文字列

DECnet for OpenVMS ネットワークで利用できる DCL コマンドのファイル指定に、ネットワーク・アクセス制御文字列を取り込むことができます。アクセス制御文字列を使用すると、ローカル・ノードのユーザがリモート・ノード上のファイルにアクセスすることができます。

アクセス制御文字列では、次に示すように、リモート・アカウントのユーザ名とそのパスワードが二重引用符で囲まれています。

NODE"ユーザ名パスワード ":: ディスク:[ディレクトリ]ファイル・タイプ

注意

アクセス制御文字列に収められている情報を知ることができれば、誰でもリモート・アカウントに侵入できるため、重大なセキュリティの侵害が発生しかねません。

10.5.2 アクセス制御文字列の保護

アクセス制御文字列の情報を保護するには、次のようにします。

- ハードコピー・ターミナルやビデオ・ターミナルで情報を漏らさないようにする。
ハードコピー・ターミナルを使用する場合は、ハードコピー出力を正しく処置する。ビデオ・ターミナルを使用する場合は、ネットワーク・ジョブが完了したら画面を消去し、DCL の RECALL/ERASE コマンドによって再呼バッファを空にする。こうしておけば、別のユーザが Ctrl/B によってコマンド行を表示したり DCL の RECALL/ALL コマンドを使用してもパスワードを見ることはできない。
- アクセス制御文字列を含むネットワーク・コマンドをコマンド・プロシージャに指定すると発見されやすいので、指定しないようにする。
- アクセス制御文字列をコマンド・プロシージャに指定しなければならない場合には、それらのファイルに最適ファイル保護を設定する。

10.5.3 代理ログイン・アカウントを使用してパスワードを保護する方法

アクセス制御文字列を使用しなくてもすむように、代理ログイン・アカウントを使用するとよいでしょう。代理ログインを使用すると、アクセス制御文字列にユーザ名やパスワードを指定しなくても、ネットワーク内でファイルをアクセスできます。代理ログインには、次のようなセキュリティ上の利点があります。

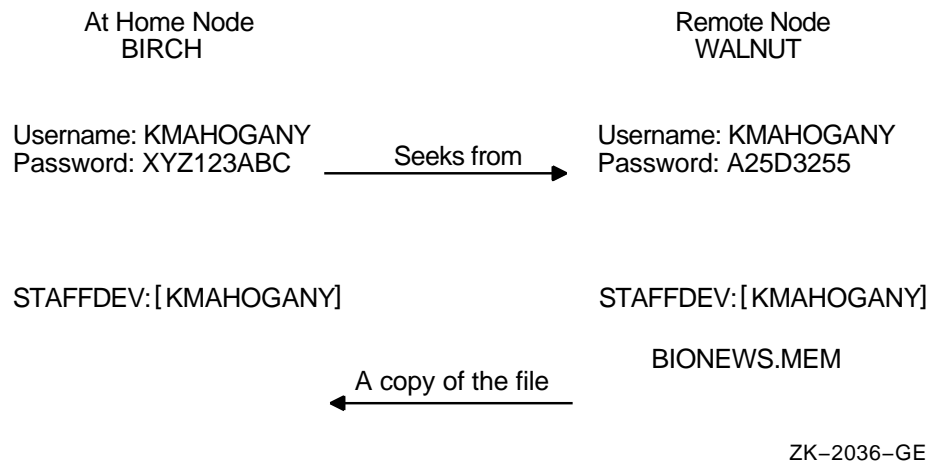
- 要求を出すターミナル上でパスワードがエコー表示されない。
- パスワードが暗号化されない形式で傍受されることのあるシステム間でのパスワードの受け渡しが無い。
- リモート・アクセスを行うコマンド・ファイルでパスワードが必要ない。

ユーザが代理ログインを開始するには、リモート・ノードのシステムまたはセキュリティ管理者が代理アカウントを作成していなければなりません。代理アカウントは、通常のアカウントと同様、OpenVMS Authorize ユーティリティ (AUTHORIZE) によって作成されます。通常、特権なしのアカウントです。ユーザは、1 つの省略時の代理アカウントと最大 15 個の省略時以外の代理アカウントにアクセスできます。代理ログインを使用すると、システム管理者にとっては設定に手間がかかりますが、より安全なネットワーク・アクセスができ、ユーザがアクセス制御文字列を入力する手間も省けます。

次の例は、通常のネットワーク・ログイン要求と代理ログイン要求の違いを示しています。次の条件を想定しています。

- KMAHOGANY ユーザは、次の 2 つのユーザ・アカウントを持っている。
 - パスワードが "XYZ123ABC" の BIRCH ノード上のアカウント
 - パスワードが "A25D3255" の WALNUT ノード上のアカウント
- KMAHOGANY は BIRCH ノードにログインした。
- KMAHOGANY は、WALNUT ノード上のアカウントの省略時のデバイスとディレクトリにある BIONEWS.MEM ファイルをコピーしようとしている。

次の図は、これらの条件を示しています。



- KMAHOGANY ユーザは、アクセス制御文字列を使用して、BIONEWS.MEM ファイルをコピーできます。

```
$ COPY WALNUT"KMAHOGANY A25D3255"::BIONEWS.MEM BIONEWS.MEM
```

A25D3255 というパスワードはエコー表示されるので、画面を見ればパスワードが分かります。

- これとは対照的に、KMAHOGANY が BIRCH ノードから WALNUT ノードのアカウントに代理アクセスを行う場合には、BIONEWS.MEM ファイルをコピーするためのコマンドは、次のようになります。

```
$ COPY WALNUT::BIONEWS.MEM BIONEWS.MEM
```

KMAHOGANY がアクセス制御文字列にパスワードを指定しなくても、システムが BIRCH ノードのアカウントから WALNUT ノードのアカウントに代理ログインを行います。このとき、パスワードの交換は行われません。

10.5.4 汎用アクセス代理アカウント

セキュリティ管理者が、フォーリン・ノードのユーザのグループに汎用アクセス代理アカウントを共用する権限を付与することもあります。たとえば、WALNUT ノードのセキュリティ管理者が次の条件で汎用アクセス・アカウントを作成するとします。

- ユーザ名は GENACCESS。
- アクセスはネットワーク・ログインに制限される。
- パスワードはアカウントの所有者しか知らない。リモート・ユーザはパスワードを知る必要はなく、アカウントの保護に役立つ。
- 省略時のデバイスとディレクトリは STAFFDEV:[BIOSTAFF]。

セキュリティ管理者が GENACCESS アカウントへの BIRCH::KMAHOGANY 代理アクセスを許可すれば、KMAHOGANY ユーザは、次のコマンドによって BIONEWS.MEM ファイルをコピーできます。

```
$ COPY WALNUT::[KMAHOGANY]BIONEWS.MEM BIONEWS.MEM
```

BIONEWS.MEM ファイルは GENACCESS アカウントの省略時のデバイスとディレクトリ (STAFFDEV:[BIOSTAFF]) がないため、KMAHOGANY は [KMAHOGANY] ディレクトリを指定しなければなりません。また、BIONEWS.MEM ファイルの保護は、GENACCESS アカウントへのアクセスを許可するものでなければなりません。そうでないと、コマンドは正しく実行されません。

特定のノード上の複数の代理アカウントにアクセスするときに、省略時の代理アカウントを使用したくない場合には、代理アカウントの名前を指定します。たとえば、GENACCESS アカウント (省略時の値) の代わりに PROXY2 という代理アカウントを使用するには、次のコマンドを入力します。

```
$ COPY WALNUT"PROXY2"::[KMAHOGANY]BIONEWS.MEM BIONEWS.MEM
```

このコマンドは、PROXY2 アカウントを使用して、WALNUT ノード上の [KMAHOGANY] ディレクトリにある BIONEWS.MEM ファイルをコピーします。

10.6 アカウントとファイルへのアクセスの監査

侵入の試みがないかどうかシステムを監視するのはセキュリティ管理者の仕事ですが、ユーザがアカウントやファイルへのアクセスの監査を手伝うこともできます。

10.6.1 最終ログイン時間

OpenVMS システムは、ユーザが最後にアカウントにログインした時間についての情報を UAF レコードに保管します。ログイン時にこの情報を表示するかどうかは、セキュリティ管理者が決定します。中程度から高水準のセキュリティ要件を持つシステムでは、頻繁にこの情報を表示して、異常な成功ログインや説明のつかない成功ログインがないかどうか、説明のつかない障害ログインがないかどうかをユーザがチェックします。

ユーザがログインしなかったのに会話型または非会話型ログインの報告があれば、ただちにセキュリティ管理者に知らせるとともに、パスワードを変更します。セキュリティ管理者は、システム・アカウント・ログと監査ログを使用して詳しく調査します。

ログイン障害メッセージを受け取ったが、障害が生じたとは考えられない場合には、誰か他のユーザがアカウントにアクセスしようとして、失敗したのかもしれませんが。パスワードをチェックして、第 1.9 節に説明するパスワード保護のガイドラインに準拠しているかどうかを調べます。ガイドラインに反することがあれば、ただちにパスワードを変更します。

ログイン障害メッセージが表示されるはずなのに、表示されなかったり、障害数が少なすぎる場合には、パスワードを変更し、ログイン障害の問題がありそうなことをセキュリティ管理者に報告します。

セキュリティ管理者が、発生した場合には特別な注意を払う必要のあるイベント・タイプを指定していることがあります。このようなイベントが検出された場合には、セキュリティ管理者は、システム・セキュリティ監査ログ・ファイルに監査結果を送信するか、セキュリティ・オペレータ・ターミナルとして使用可能なターミナルにアラームを送信するようにシステムに指示します。たとえば、セキュリティ管理者が書き込みアクセスが禁止されている 1 つ以上のファイルを指定している場合に、監査機能を使用可能にするか、アラームを設定しておけば、このようなファイルへのアクセスが発生したことが分かります。

アカウントへの侵入があると考えられる場合には、パスワードを変更します。重要なファイルに監査機能を適用するように、セキュリティ管理者に要求してもよいでしょう。

10.6.2 監査機能やアラームを起動するイベント

監査機能やアラームを起動するイベントには、次のものがあります。

セキュリティ監査またはアラームを開始するイベント	
イメージのインストール	システム・パスワードとユーザ・パスワード、システム登録ファイル、ネットワーク代理ファイル、または権利データベースの変更
ファイル・アクセスのタイプ	
ボリュームのマウントとディスマウント	
ACL ファイルまたはグローバル・セクションが要求したアクセス・イベント	ログイン、ログアウト、ログイン障害、侵入の試み

たとえば、CONFIDREVIEW.MEM というファイルを監査するとします。ABADGUY というユーザが CONFIDREVIEW.MEM というファイルにアクセスするときに削除アクセス権を持っていると、次のような監査レコードがシステム・セキュリティ監査ログ・ファイルに書き込まれます。

```
%%%%%%%% 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
```

監査メッセージには、不法アクセス者の名前、アクセス方法 ([SYSEXEC]DELETE.EXE プログラムを使用して行われた削除操作)、アクセス時間 (9:21 A.M)、ファイルにアクセスするのに使用した特権 (SYSPRV) が示されています。セキュリティ管理者は、これらの情報に基づいて処置をします。

10.6.3 セキュリティ監査ログ・ファイル

いずれかのファイルがアクセスされ、そのファイルの ACL の監査エントリ (第 10.6.4 項を参照) に指定された条件が満たされるたびに、セキュリティ監査メッセージが監査ログ・ファイルに書き込まれます。CONFIDREVIEW.MEM ファイルにアクセスがあると、セキュリティ監査機能によって保護されているシステム上のファイルにアクセスがあった場合と同様、監査レコードをセキュリティ監査ログ・ファイルに書き込むようにプロンプトが出されます。

監査機能を導入したら、セキュリティ管理者とともに、別の侵入が生じていないかどうかを定期的にチェックします。

10.6.4 重要なファイルにアクセス制御リスト・エントリ (ACE) を追加する場合

不正にアクセスされた重要ファイルがあれば、セキュリティ管理者に相談して、それらのファイルへのアクセスを監査する戦略を立てるとよいでしょう。

状況を調べて、標準保護コードや汎用 ACL (『OpenVMS Guide to System Security』を参照) を使用してファイルを保護するためのすべての手段を実行しつくした場合には、セキュリティ監査が必要であると判断してよいでしょう。

セキュリティ監査機能を指定するには、所有しているファイルまたはアクセス制御権を持つファイルに特別なアクセス制御リスト・エントリ (ACE) を追加します。ただし、監査ログ・ファイルはシステム全般に適用されるので、システムごとのセキュリティ管理者がファイル監査の使い方を制御することをおすすめします。制御権を持つファイルにはユーザでも監査 ACE を追加できますが、セキュリティ管理者の場合は、システム・レベルでファイルの監査機能を使用可能にしなければなりません。

アカウントに侵入の試みがあったと考えられる場合には、セキュリティ管理者は、すべてのファイル・アクセスの監査機能を一時的に使用可能にすることができます。また、監査機能を使用可能にすれば、ファイルに対する読み込みアクセスを監視して、ファイルにアクセスして内容を閲覧したユーザを見つけることもできます。

1 つのファイルにアクセス違反があると、他のファイルにもアクセス上の問題が存在することがよくあります。したがって、セキュリティ管理者は、セキュリティ監査 ACL を持つすべての重要ファイルに対するアクセスを監視した方がよいでしょう。重要ファイルに望ましくないアクセスがあった場合には、ただちに処置を講じなければなりません。

たとえば、ユーザ RWOODS とセキュリティ管理者が、極秘ファイルである CONFIDREVIEW.MEM をいつアクセスしたらよいかを知らなければならないと同意した場合には、ユーザ RWOODS は、次のようにして CONFIDREVIEW.MEM ファイルの既存の ACL にエントリを追加します。

```
$ SET SECURITY/ACL=(ALARM=SECURITY,ACCESS=READ+WRITE-  
_$ +DELETE+CONTROL+FAILURE+SUCCESS) CONFIDREVIEW.MEM
```

デバイスとファイルの論理名定義

論理名とは、ファイル、ディレクトリ、デバイス、またはキューなどのシステム・オブジェクトを表す名前の代わりに使用できる名前をいいます。たとえば、省略時のディスクやディレクトリに論理名を割り当てることが可能です。論理名は、読みやすさとファイルの独立性の2つの働きをします。

頻繁に使用するファイル、ディレクトリ、またはデバイスに、短く分かりやすい論理名を定義することができます。こうした論理名は、完全なファイル指定より覚えやすく、入力も簡単です。使用頻度の高い名前は、ログイン・コマンド・プロシージャの中で定義します。ユーザが頻繁に使用する名前は、システム管理者がシステム・スタートアップ・コマンド・プロシージャの中で定義します。

論理名を使用すれば、プログラムやコマンド・プロシージャを物理的なファイル指定を意識せずに扱うことができます。たとえば、あるコマンド・プロシージャで論理名 ACCOUNTS を定義しておけば、どのディスク上のどのファイルに対しても ACCOUNTS を使用するだけでコマンド・プロシージャを実行できます。本章では、次のことについて説明します。

- 論理名の特徴
- システム定義論理名の使い方
- 論理名の作成
- 論理名の削除
- 論理名の変換
- 論理名の表示
- 検索リストの作成と使用
- 論理名テーブルの特徴
- 省略時の論理名テーブル
- 論理名テーブルの作成
- 論理名変換の順序の変更
- 論理名テーブルの削除
- プロセス永久論理名の使用

本章で説明するコマンドについての詳細は、『OpenVMS DCL デictionary』またはオンライン・ヘルプを参照してください。

11.1 論理名の特徴

論理名は次のような特徴を持っています。

- 1つの文字列 (等価文字列または等価名と呼ばれる) である場合と、等価文字列のリスト (検索リストと呼ばれる) である場合がある。論理名を使用すると、その論理名は解釈時に対応する等価文字列に置き換えられる。
- 省略時の論理名テーブル、またはユーザが作成した論理名テーブルに格納される。
- 長いファイル指定の入力の手間を軽減する目的で使用できる。
- ユーザが定義することもできるし、システムによって定義することもできる。
- プログラムやコマンド・プロシージャを、物理的なファイル指定から独立させることもできる。たとえば、コマンド・プロシージャに論理名 ACCOUNTS を指定しておくと、コマンド・プロシージャの実行前に ACCOUNTS の定義を変更することにより、任意のディスク上の任意のファイルを実行することができる。

一般に、コマンドはシステム・オブジェクトを受け取ると、その名前が論理名かどうかをチェックします。論理名の場合には、論理名が実際の値に置き換えられてからコマンドが実行されます。

たとえば、DISK7:[WALSH.COMMAND_PROC]ディレクトリを指す論理名 COMS を定義しておけば、DCL コマンド行での入力が次のように簡単になります。

```
$ DEFINE COMS DISK7:[WALSH.COMMAND_PROC]
```

次のように、論理名は DCL コマンドの中でも使用することができます。

```
$ SET DEFAULT COMS
```

```
$ TYPE COMS:PAYROLL.COM
```

11.2 システムが定義する論理名

システムを起動してログインすると、そのシステムで使用される論理名が作成されます。これらの論理名を使用すると、物理デバイス名を使用しないで、頻繁に使用されるファイルやデバイスを参照することができます。これらの論理名一覧は第 11.9.3 項を参照ください。

ログインするたびに、ユーザ・プロセス用の論理名グループが作成され、それらの論理名がプロセス・テーブルに置かれます。これらの論理名一覧は第 11.9.1 項を参照ください。

たとえば、オペレーティング・システムのプログラムの一覧を表示するには、次のように、プログラムが実際に格納されているディスクやディレクトリの名前ではなく、論理名 SYSS\$SYSTEM を使用できます。

```
$ DIRECTORY SYSS$SYSTEM
```

たとえば、論理名 SYSS\$LOGIN は、ログインしたときの省略時のデバイスとディレクトリを表しています。SET DEFAULT コマンドを使用して現在の省略時の値を変更した場合には、次のコマンドを使用すれば、当初の省略時のディレクトリにあるファイルを表示できます。

```
$ TYPE SYSS$LOGIN:DAILY_NOTES.DAT
```

11.3 論理名の作成

論理名は、ASSIGN コマンドまたは DEFINE コマンドで作成します。本章では、例のなかで DEFINE コマンドを使用することとします。

通常、論理名はログイン・コマンド・プロシージャ (LOGIN.COM) で定義するので、ログインするたびに論理名を使用することができます。論理名は会話形式でも作成できます。ただし、この場合には、現在のプロセスが終了した時点でそれらの論理名は使用できなくなります。

ユーザが自分のプロセス・テーブルに作成した論理名は、他のユーザのプロセスでは使用することができません。システム管理者、または特権をもったユーザは、システムにログインするあらゆるユーザが使用できる論理名を共用テーブル上に作成できます。グループ、またはシステム論理名テーブルは共用テーブルの例です。

共用テーブルの詳細については、第 11.9.4 項を参照してください。

11.3.1 DEFINE コマンドの使用方法

DEFINE コマンドで論理名を定義するときの形式は、次のとおりです。

```
DEFINE 論理名 等価名[...]
```

同じ形式を使用して、ノード名、ファイル指定、デバイス名、アプリケーション固有情報、他の論理名も作成できます。

省略時の設定では、DEFINE コマンドは、ユーザ・プロセスの論理名テーブルに論理名を登録します（第 11.8 節を参照）。これらの論理名は、そのプロセスとサブプロセスだけで使用できるものです。他の論理名テーブルに論理名を追加する場合は、/JOB、/GROUP、/SYSTEM、または/TABLE=テーブル名のいずれかの修飾子を 1 つ付けて別のテーブルを指定します。この修飾子のうち、最初の 3 つは、それぞれ、省略時設定のジョブ、グループ、システムの論理名テーブルを指定しま

す。/TABLE=テーブル名は、どのような種類のテーブルでも指定でき、また、クラスタ単位のテーブルを指定する際に使用できる唯一の修飾子です。

たとえば、次の例では、論理名 WORKFILE を
等価文字列 DISK2:[WALSH.REPORTS]WORK_SUMMARY.DAT と対応づけるよう
定義しています。

```
$ DEFINE WORKFILE DISK2:[WALSH.REPORTS]WORK_SUMMARY.DAT
```

WORKFILE を論理名として定義しておけば、等価文字列の代わりに論理名を使用できます。

たとえば、次のコマンドは、印刷キュー BLDGC_LPS20_ANSI に対応する論理名 MY_Q を作成します。

```
$ DEFINE MY_Q BLDGC_LPS20_ANSI
```

以後は、次のコマンドによってファイル FABLES.TXT を印刷キュー BLDGC_LPS20_ANSI にプリントできます。

```
$ PRINT/QUEUE=MY_Q FABLES.TXT
```

次の例は/TABLE=テーブル名修飾子の使用例であり、プロセス論理名テーブル以外のテーブルに論理名を作成します。LNM\$SYSCUSTER を指定すると、論理名は省略時のクラスタ単位テーブルである LNM\$SYSCUSTER_TABLE に置かれるので、クラスタ内のすべてのユーザがこの論理名にアクセスできます。

```
$ DEFINE/TABLE=LNM$SYSCUSTER CUSTOMERS DISK1:[CUSTOMER_VISITS]CUSTOMERS.TXT
```

11.3.2 ファイル入出力用コマンド・プロシージャ内の論理名の作成

コマンド・プロシージャ内で論理名を使用してファイル入出力を行うことができます。OPEN コマンドを使用してファイルをオープンする際には、そのファイルの論理名も作成することができます。これに続く READ, WRITE, CLOSE コマンドでは、このファイルを参照するために実際のファイルを指定せずに論理名を使用することができます。

次の例では、OPEN コマンドは論理名 INFILE を作成し、CLOSE コマンドはこの論理名を削除しています。

```
$ OPEN INFILE DISK3:[WALSH]DATA.DAT  
$ READ INFILE RECORD  
$ CLOSE INFILE
```

11.3.3 論理名の作成規則

DEFINE コマンドで論理名を作成するときには、次の規則に従います。

- 等価名と論理名はともに、255 文字以内とする。論理名には、英数字、アンダスコア(_), ドル記号(\$), ハイフン(-)を使用できる。
- 等価名を指定する場合には、ファイル指定に必要な句読点(コロン, 大括弧, ピリオド)を使用する。たとえば、デバイス名の終わりにはコロンを付け、ディレクトリ指定は大括弧で囲む。また、ファイル・タイプの前にはピリオドを付ける。
- 論理名がファイル指定の一部分だけを表す場合には、論理名とファイル指定の残りの部分とをコロンで区切る。ただし、論理名が完全なファイル指定を表すときには、最後のコロンは必要ない。

また、論理名は、ファイル指定の左端の構成要素になるようにすること。

- 場合によっては、論理名の終わりにコロンを付ける。

ASSIGN コマンドは、コロンを削除してから論理名を論理名テーブルに登録するが、DEFINE コマンドは、論理名の一部としてコロンをセーブすることに注意。

- 論理名をある等価文字列に対応づけてから、同じ論理名を別の等価文字列に対応づける場合、両者を異なる論理名テーブルの中で定義するか、異なるアクセス・モードで定義しないかぎり、1 番目の定義ではなく 2 番目の定義が有効になる。

次のコマンドは、DISK1:[SALES_STAFF]PAYROLL.DAT:ファイルを表示します。

```
$ 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
```

11.3.4 変換属性

論理名を作成する場合には、システムが等価名をどのように解釈するかを決定する変換属性を指定できます。

2 つの変換属性を等価名に適用するには、DEFINE コマンドに/TRANSLATION_ATTRIBUTES 修飾子を使用します。これは、定位置修飾子で、この修飾子をコマンド行のどこに置くかによって、すべての等価名に変換属性が適用されたり、特定の等価名だけに変換属性が適用されたりします。

次の例では、デバイス名 DJA3: は論理名 DISK で隠されます。

```
$ DEFINE/TRANSLATION_ATTRIBUTES=CONCEALED DISK DJA3:
$ SHOW DEFAULT
  DISK:[SAM.PUP]
$ SHOW LOGICAL DISK
  "DISK" = "DJA3" (LNM$PROCESS_TABLE)
```

論理名 DISK は物理装置 DJA3 を表わします。したがって、SHOW DEFAULT コマンドは、物理デバイス DJA3 ではなく、論理名 DISK を表示します。SHOW LOGICAL コマンドは論理名 DISK が実際に指しているものを明らかにします。

CONCEALED 属性を指定すると、システム・メッセージにデバイスの物理名ではなく、論理名が表示されるようになります。普通、CONCEALED 属性は物理デバイスを表す論理名と一緒に使用します。隠されたデバイスを使用すると、どの物理デバイスにディスクやテープが保持されているかを考えずに、プログラムやコマンド・プロシージャを作成したり、他の操作を実行したりできます。また、物理デバイス名より分かりやすい名前を使用できます。

TERMINAL 属性は、論理名の反復変換が行われないようにします。すなわち、等価名が論理名であるかどうかを調べなくなり、最初の変換で「終わり」になります。

11.3.5 アクセス・モード

OpenVMS は次のアクセス・モードを持っています。

- ユーザ・モード (最も外側で、最も特権の少ないモード)
- スーパーバイザ・モード
- エグゼクティブ・モード
- カーネル・モード (最も内側で最も特権を持つモード)

DCL コマンド DEFINE または ASSIGN を使用して、最初の 3 つのモード (ユーザ、スーパーバイザ、エグゼクティブ) で論理名を作成することができます。各論理名の定義に異なるアクセス・モードを指定して、同じ論理名を同じ論理名テーブルの異なる等価文字列に等しいと定義することができます。論理名テーブルにエグゼクティブ・モードで論理名を作成するには、SYSNAM または SYSPRV 特権を持つ必要がありますので注意してください。

ユーザ・モード

ユーザ・モードで作成される論理名は一時的なものです。次のコマンドやイメージの実行のためにだけ論理名を使用したい場合には、ユーザ・モードで論理名を定義します。

次の例では、PAYABLE プログラムの実行後、論理名 ADDRESSES が自動的に削除されます。

```
$ DEFINE/USER_MODE ADDRESSES DISK1:[SAM.ACCOUNTS]OVERDUE.LIS  
$ RUN PAYABLE
```

スーパーバイザ・モード

モードを指定せずに DEFINE コマンドを使用すると、スーパーバイザ・モードで論理名が作成されます。

次の例では、コマンドは論理名 ACCOUNTS をプロセス論理名テーブルの中の 2 つの異なる等価名に、1 つはスーパーバイザ・モードで、もう 1 つはエグゼクティブ・モードで定義しています。

```
$ DEFINE ACCOUNTS DISK1:[ACCOUNTS]CURRENT.DAT  
$ DEFINE/EXECUTIVE_MODE ACCOUNTS DISK1:[JANE.ACCOUNTS]OBSOLETE.DAT
```

エグゼクティブ・モード

すべての特権イメージおよび LOGINOUT のようなユーティリティは、論理名を検索する際に、ユーザ・モードとスーパーバイザ・モードの名前とテーブルを参照しません。論理名を、ユーティリティを含む特権イメージで使用する場合には、これはエグゼクティブ・モード・テーブルまたはカーネル・モード・テーブルに、エグゼクティブ・モードまたはカーネル・モードで定義する必要があります。エグゼクティブ・モードで定義される論理名の候補としては他に、プリント・キューやシステム・ディスクなどの、作業グループとシステム資源が使用する公用ディレクトリの名前があります。

カーネル・モード

カーネル・モードで論理名を作成できるのは、オペレーティング・システムと特権プログラムだけです。

11.3.6 論理ノード名の作成

ネットワーク・ノード名の代わりに、またはノード名とアクセス制御文字列の代わりに論理ノード名を使用することができます。論理ノード名を定義しておくと、これを使用して画面上でユーザ名とパスワードを入力(して表示)する手間を省くことができます。

論理ノード名を定義する場合には、次の規則にしたがってください。

- 論理名の始まりはアンダスコア(_)にしない。
- 等価文字列の終わりはダブル・コロンの(::) にし、二重引用符(" ")で囲む。
- アクセス制御文字列で二重引用符を表示する場合は、二重引用符を 2 組("" "") 使用する。

(アクセス制御文字列については、本書の第 3.1.6 項、第 3.1.12 項、第 10.5 節を参照してください)。

- 1 ~ 255 の文字を含む論理名を指定する。

注意

ファイルにパスワードを含んだ DEFINE コマンドをファイルに記述しないでください(ログイン・コマンド・プロシージャなど)。他の人がファイルを読み込むと、パスワードが知られてしまいます。

次のコマンドは、論理名 BOS をノード名 BOSTON とアクセス制御文字列に等しいと定義しています。ここで ADAMS はユーザ名で、OLMEKIKI はパスワードです。

```
$ DEFINE BOS "BOSTON" "ADAMS OLMEKIKI" "::"
```

11.3.6.1 ファイル指定に論理ノード名を使用

ファイル指定には、論理ノード名(ローカル・ノードでシステムが変換する)と論理装置名(リモート・ノードでシステムが変換する)をともに含むことができます。論理名を使用してノード名だけを表す場合、ファイル指定のノード位置に論理名を使用するときにダブル・コロン(::)を記述する必要があります。

システムは、ローカル・ノードで論理ノード名を変換した後に、ファイル指定の残りの部分を解析してその形式が有効であるかどうか判断します。

次の例では、システムがローカル・ノードで論理ノード名 NYC を変換します。システムは、リモート・ノード(NEWYRK)で論理装置名(DOC:)を変換します。

```
$ DEFINE NYC NEWYRK::  
$ TYPE NYC::DOC:[PERKINS]TERM_PAPER.DAT
```

11.3.6.2 アクセス制御文字列の上書き

論理ノード名のアクセス制御文字列を上書きするには、コマンド行に論理名とアクセス制御文字列の両方を指定します。

次の例では、アクセス制御文字列 "REVERE HTEBAZILE" が、BOS の等価文字列で与えられたアクセス制御文字列を上書きします。

```
$ DEFINE BOS "BOSTON" "ADAMS OLMEKIKI" "::"  
$ TYPE BOS "REVERE HTEBAZILE" "::RIDE.DAT
```

システムが論理ノード名を反復して変換すると、最初に変換された論理ノード名のアクセス制御情報が次のアクセス制御情報を上書きします。たとえば、論理名 TEST1 は、TORONTO "TEST NAMWENLUAP"::DBA1: に変換されます。

```
$ DEFINE TORONTO "TRNTO" "TEST EIZNEKCAM" "::"  
$ DEFINE TEST1 "TORONTO" "TEST NAMWENLUAP" "::DBA1:"  
$ TYPE TEST1:PROC.DAT
```

TORONTO は論理ノード名なので、反復変換が行われます。つまり、オペレーティング・システムは、定義内の全レベルの論理名が見つかる間で論理名テーブルを検索します。ただし、論理名割り当てコマンド DEFINE TEST1 のアクセス制御文字列は、論理ノード名割り当てコマンド DEFINE TORONTO のアクセス制御文字列を上書きします。したがって、TYPE コマンドにより、次のファイルが表示されます。

```
TRNTO"TEST NAMWENLUAP":DBA1:PROC.DAT
```

11.3.7 同一オブジェクトに複数論理名を作成

複数の DEFINE コマンドを使用すると、同一のオブジェクトを参照する複数の論理名を作成することができます。たとえば、次のコマンドは論理名 \$TERMINAL と CONSOLE をターミナルの物理名に等しいと定義するので、2 つの論理名は同じ装置 (LTA69) に変換されます。

```
$ DEFINE $TERMINAL LTA69  
$ DEFINE CONSOLE LTA69
```

11.4 論理名の削除

論理名を削除するには、DEASSIGN コマンドを使用します。プロセスとジョブ論理名テーブルで論理名を定義すると、プロセスが終了するかまたはユーザの操作で明示的に削除しない限り、論理名は削除されません。ただし、DEFINE コマンドに /USER_MODE 修飾子を指定すると、論理名はプロセス論理名テーブルに定義されて、次のコマンド・イメージを実行した後に自動的に削除されます。

コロンの終わっている論理名を削除するには、2 つのコロンを指定します。DEASSIGN コマンドは、ASSIGN コマンドと同様、コロンを 1 つ削除してから一致する論理名を捜して論理名テーブルを検索します。

11.5 論理名の変換

システムは、DCL コマンド行のファイル指定または装置名を読み込むと、左端のコンポーネントが論理名であるかどうかファイル指定または装置名を調べます。左端のコンポーネントがコロン、スペース、カンマ、行終了文字 (Enter など) のいずれかで終わっている場合には、システムはこれを論理名として変換しようとします。左端のコンポーネントが他の文字で終わっている場合には、システムはこれを論理名としては変換しようとしません。

次の例に示すコマンドを入力すると、PUP がファイル指定の左端のコンポーネントなので、システムは PUP が論理名であるかどうかチェックします。左端のコンポーネントが Enter で終わっているため、システムは PUP を変換しようとします。

```
$ TYPE PUP 
```

次の例に示すコマンドを入力すると、システムは DISK が論理名であるかどうかチェックします。DISK が左端のコンポーネントであり、コロンで終わっているため、システムはこれを変換しようとします。システムは、PUP のチェックを行いません。

```
$ TYPE DISK:PUP 
```

第 3 の例では、左端のコンポーネントが右角括弧(])で終わっているの
で、[DRYSDALE]PUP を変換しようとはしません。

```
$ TYPE [DRYSDALE]PUP 
```

11.5.1 反復変換

論理名変換は、反復されることがあります。システムは論理名を変換すると、見つけた論理名に対して最初の論理名に含まれていた変換プロセスを繰り返します。

システムは、論理名の変換を実行するレベル数を制限します。レベルの数はシステムの機能によって異なりますが、最低 9 レベルです。システムによって決まるレベル数を超える数を定義した場合、または循環する定義を作成した場合は、論理名を使用するとエラーが発生します。

次の例では、最初の DEFINE コマンドが論理名 DISK を装置名 DUA1 に等しいと定義します。第 2 の DEFINE コマンドは、論理名 MEMO をファイル指定 DISK:[JEFF.MEMOS]COMPLAINT.TXT に等しいと定義します。

```
$ DEFINE DISK DUA1:  
$ DEFINE MEMO DISK:[JEFF.MEMOS]COMPLAINT.TXT
```

システムは、論理名 MEMO を変換すると、等価文字列
DISK:[JEFF.MEMOS]COMPLAINT.TXT を捜します。次にシステムは、このファイル指定の左端のコンポーネントがコロン、スペース、カンマ、行末終了文字のいずれで終わっているかチェックをします。DISK の後のコロンが見つかったら、システムは、その論理名も変換します。ファイル指定の最終的な変換は、次のようになります。

```
DUA1:[JEFF.MEMOS]COMPLAINT.TXT
```

11.5.2 システムの省略時の設定で補完されるフィールド

システムは、論理名を変換する際、ファイル指定の中で欠けているフィールドを、現在の省略時設定の装置、ディレクトリ名、バージョン番号で補完します。論理名を使用してコマンドに入力ファイルを指定すると、コマンドは論理名を使用してファイル指定を出力ファイルにも割り当てます。

等価文字列にファイル名とファイル・タイプが含まれる場合、出力ファイルは同じファイル名とファイル・タイプを与えられます。等価文字列にファイル・タイプが含まれない場合には、省略時設定のファイル・タイプが補充されます。補充されるファイル・タイプは、使用しているコマンドによって異なります。

入力ファイルのリストを指定する際に論理名を使用すると、各論理名の等価文字列は一時的な省略時の設定値を使用します。

次の例では、装置名が論理名 HIG に対して指定されていないので、MAL に対する装置名が一時的省略時設定装置として DBA1 を定義します。

```
$ SET DEFAULT DBA2:[CASEY]
$ DEFINE MAL DBA1:[MALCOLM]
$ DEFINE HIG [HIGGINS]
$ PRINT ALPHA,MAL:BETA,HIG:GAMMA
```

PRINT コマンドは、次のファイルをさがします。

```
DBA2:[CASEY]ALPHA.LIS
DBA1:[MALCOLM]BETA.LIS
DBA1:[HIGGINS]GAMMA.LIS
```

11.5.3 論理名変換に対する省略時の検索順序

同一の論理名が 1 つ以上の論理名テーブルに存在することがあります。システムは、ファイル指定にある論理名を変換する場合、一致するものが見つかるまで論理名テーブルのリストを検索します。システムは、見つけた最初の一致する論理名を使用します。

検索が行われる論理名テーブルのリストは、論理名 LNMS\$FILE_DEV の定義で指定されます。省略時設定のリストには、プロセス、ジョブ、グループ、システム、クラスタ単位のシステム論理名テーブルから構成されます。検索は、この順序(プロセス、ジョブ、グループ、システム、クラスタ単位のシステム)で行われます。

第 11.11 節で説明するように、検索順序を変更することもできます。

11.6 論理名の表示

SHOW LOGICAL コマンドを使用して、論理名とその等価文字列を表示します。

場合によっては、論理名の定義に別の論理名が含まれることがあります。SHOW LOGICAL コマンドは、反復変換を実行します。次に等価文字列と変換のレベルを表示します。レベル数は、0 が第 1 レベル、1 が第 2 レベル、というようにゼロが基準になっています。指定された論理名に対して検索した最初の変換だけを表示するには、SHOW TRANSLATION コマンドを使用します(詳細については、『OpenVMS DCL デictionary』を参照してください)。

SHOW LOGICAL コマンドを使用して、プロセスパーマネント・ファイル (第 11.13 節を参照) に等価文字列を定義する場合、コマンドにより表示されるのは、文字列の装置部分だけです。例を次に示します。

```
$ SHOW LOGICAL SYS$INPUT
"SYS$INPUT" = "_TTB4:" (LNM$PROCESS_TABLE)
```

次の例では、論理名 MYDISK が表示されます。2 つの変換がおこなわれ、数値 1 は変換の第 2 レベルを示しています。

```
$ SHOW LOGICAL MYDISK
"MYDISK" = "WORK4" (LNM$PROCESS_TABLE)
1 "WORK4" = "$255$DUA17:" (LNM$SYSTEM_TABLE)
```

次の例では、論理名 WORKFILE の等価文字列が表示されます。

```
$ SHOW LOGICAL WORKFILE
"WORKFILE" = "DISK2:[WALSH.REPORTS]WORK_SUMMARY.DAT" (LNM$PROCESS_TABLE)
```

システムは、論理名、その変換、および論理名を含む論理名テーブルの名前を表示します。

11.6.1 論理名テーブル検索の指定

省略時設定により、SHOW LOGICAL コマンドは、プロセス、ジョブ、グループ、システム、クラスタ単位のシステム・テーブルを検索して、一致するものすべてを表示します。ただし、/TABLE 修飾子を使用すると特定の論理名テーブルを検索するように指定することができます。また、修飾子 /GROUP、/SYSTEM、/JOB、/PROCESS を使用して、グループ、システム、ジョブ、プロセス論理名テーブルの論理名をそれぞれ表示することもできます。

次の例では、/TABLE 修飾子を使用して SHOW LOGICAL コマンドがプロセス論理名テーブル (LNM\$PROCESS) にある論理名を表示します。

```
$ 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:"
```

11.6.2 変換属性とアクセス・モードの表示

論理名の変換属性とアクセスモードを表示するには、次のように SHOW LOGICAL /FULL コマンドを使用します。

```
$ SHOW LOGICAL/FULL SYS$ERROR  
"SYS$ERROR" [exec] = "_PADRAIC$TDA824:" [terminal] (LNM$PROCESS_TABLE)
```

この例では、エグゼクティブ・モードの論理名 SYS\$ERROR を表示し、変換属性、ターミナルを示します。

11.7 検索リストの作成と使用

論理名が、単一の DEFINE (または ASSIGN) コマンド内の複数の等価文字列に等しいと定義されると、検索リストが作成されます。

ファイル指定にある検索リストを使用すると、検索リストは次のように変換されます。

- 検索リストに含まれる装置が 1 つだけの場合、元の省略時設定のディレクトリが検索されます。
- 検索リストに 1 つの装置と 1 つのディレクトリが含まれる場合、完全なファイル指定を構成するためにその両方が使用されます。

システムは、一致するものを見つけるまで、等価文字列を指定した順序で論理名を変換します。

コマンドの効果が及ぶのは、見つけた最初のファイルだけです。その時点で検索は終了します。一致するものが見つからない場合には、システムは見つけようとした最後のファイルに関してエラーを報告するだけです。

検索リストはワイルドカードではありませんので注意してください。これは調べるべき場所のリストです。

次の例では、論理名 GETTYSBURG が検索リストです。

```
$ DEFINE GETTYSBURG [JONES.HISTORY],[JONES.WORKFILES]  
$ SHOW LOGICAL GETTYSBURG  
  
"GETTYSBURG" = "[JONES.HISTORY]" (LNM$PROCESS_TABLE)  
              = "[JONES.WORKFILES]"
```

次の例では、TYPE コマンドは等価文字列[JONES.HISTORY]を検索してから、[JONES.WORKFILES]を検索します (GETTYSBURG に指定した論理名の優先定義に指定した順序)。

```
$ 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 コマンドは、SPEECH.TXT という名前のファイルを見つけると、検索を終了してそのファイルを表示します。

11.7.1 ワイルドカードを使用できるコマンドでの検索リストの使用

ワイルドカードを使用できるコマンドでは検索リストを使用することができます。ワイルドカードを使用すると、システムは検索リストの各等価文字列を使用してファイル指定を形成します。コマンドは、既存のファイルを示す各ファイル指定コマンドで動作します。

次の例で、DIRECTORY コマンドは、バージョン・フィールドにワイルドカード文字を指定しています。このコマンドは、GETTYSBURG で定義された検索リスト内の SPEECH.TXT の全バージョンを探し出します。

```
$ 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.
```

(たとえば DIRECTORY コマンドを使用して) 検索リストを入力すると、オペレーティング・システムはリストの一部にある要素を使用して、リストの他の部分から省略されているファイル指定の部分を補充します。ファイル指定が完全ではない場合 (次の例の SYS\$LOGIN で示すように)、コマンド行が multiple files and file-not-found 条件を生成します。

```
$ DIRECTORY SYS$MANAGER:LOGIN.COM,SYS$LOGIN
```

次に示すように、ファイル指定の後にセミコロンを入れると multiple files and file-not-found 条件を回避することができます。

```
$ DIRECTORY SYS$MANAGER:LOGIN.COM; ,SYS$LOGIN
```

11.7.2 SET DEFAULT コマンドによる検索リストの使用

SET DEFAULT コマンドのパラメータの最初の部分として検索リストを指定すると、システムは検索リスト名を変換せずに SYS\$DISK に割り当てます (SYS\$DISK は省略時設定のディスクに変換する論理名です)。検索リストを SET DEFAULT コマンドのパラメータの最初の部分として指定すると、検索リスト内の各等価文字列には装置名が含まれている必要があります。

次の例では、装置とディレクトリの両方が指定されています。したがって、両方がファイル指定の構成に使用されます。

```
$ DEFINE FIFI DISK1:[FRED],DISK2:[GLADYS],DISK3:[MEATBALL.SUB]
$ DIRECTORY FIFI:MEMO.LIS
```

このコマンドにより次のファイルのリストが表示されます。

```
DISK1:[FRED]MEMO.LIS
DISK2:[GLADYS]MEMO.LIS
DISK3:[MEATBALL.SUB]MEMO.LIS
```

次の例では、SHOW DEFAULT コマンドは省略時設定のディスクとディレクトリを DISK2:[MEATBALL.SUB]として示します。次に、検索リスト FIFI が定義されます。SET DEFAULT コマンドは、そのパラメータとして検索リストを使用します。SHOW DEFAULT コマンドを2度めに実行すると、省略時のディレクトリが変更されていないことが示されます。ただし、検索リスト FIFI がその等価文字列とともに省略時設定の装置として表示されます。SHOW DEFAULT コマンドは、検索リストがシステムに定義された順序で、検索リストを表示します。

```
$ 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]
```

11.7.3 RUN コマンドによる検索リストの使用

RUN コマンドの後に検索リストが続く場合、システムは前述のようにファイル指定を形成します。ただし、システムはリスト内のファイルがインストールされたイメージであるかどうかをチェックします。システムは、検索リスト内のインストールされたイメージの中の最初に指定されたファイルを実行します。その後 RUN コマンドが終了します。

ファイル指定のいずれもインストールされたイメージではない場合には、システムはファイル指定を形成するプロセスを繰り返します。今度は、システムはディスク上の各ファイル指定を調べます。そこで最初に見つけたファイルを実行します。指定されたファイルのいずれも既知のファイル・リストまたはディスク上に見つからない場合は、エラー・メッセージが表示されます。

11.7.4 複数の検索リストの検索順序

ファイル指定には、複数の検索リストが含まれることがあります。その場合、ファイル名検索リスト内の各項目が使用されますが、最初の装置名は一定のまま保持されます。ファイル名検索リスト内のすべての項目が最初の装置名と組み合わせられると、今度は第2の装置名と組み合わせられます。このプロセスは、それぞれの装置が検索されるまで継続します。

検索リスト内の1つの名前が別の検索リストに変換される場合には、反復(ネストした)検索リストを持つこともあります。この場合には、システムはサブリスト内のそれぞれの名前を使用してから、次の上位レベル名へと継続していきます。

次の例では、ファイル名と装置名に検索リストを持つファイル指定を示します。

```
$ 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.
```

各ファイル名を一覧するディレクトリ・コマンドは、最初に WORK1:[ROSE]おを表示し、次に WORK2:[THORN]を表示しています。

次の例では、反復検索リストを示しています。

```
$ DEFINE NESTED FRED.DAT, NEW_LIST, RICKY.DAT  
$ DEFINE NEW_LIST ETHEL.DAT, LUCY.DAT
```

検索リスト NESTED の検索順序は次のとおりです。

```
FRED.DAT  
ETHEL.DAT  
LUCY.DAT  
RICKY.DAT
```

11.8 論理名テーブルの特徴

論理名テーブルには、次の属性があります。

- 有効範囲 (共有可能かプロセス固有かどうか)
- アクセス・モード
- 名前
- 親論理名テーブル
- アクセス制御 (共用可能論理名テーブルのみ)
- 制限 (論理名により専有されるプールの量を制限する)

システムの初期化中に、共用可能論理名テーブルがいくつか作成されます。新しいプロセスが作成されると、システムはそのプロセスに対して共用可能でプロセス固有のテーブルを他にいくつか作成します。表 11-1 にこれらのテーブルのすべてを示します。

論理名テーブルのアクセス・モードは、その作成時に指定できます。これが指定されないモードは、テーブル作成が要求されたアクセス・モードに省略時設定されますが、通常これはスーパーバイザ・モードまたはユーザ・モードです。論理名テーブルには、それ自身のアクセス・モードとより低い特権アクセス・モードの論理名が含まれることがあります。論理名テーブルは、同じまたはそれ次の特権アクセス・モードの別のテーブルに対する親テーブルである場合もあります。

論理名テーブルは、その名前で識別されますが、名前自体が論理名です。論理名として各名前テーブル名が論理名テーブル内に含まれている必要があります。

11.8.1 論理名テーブルのディレクトリ

ディレクトリという 2 つの特殊な論理名テーブルが、論理名テーブル名のコンテナとして存在します。

- プロセス・ディレクトリ LNM\$PROCESS_DIRECTORY

プロセス・ディレクトリには、そのプロセスに対するすべてのプロセス固有テーブルの名前と自身のテーブル名が含まれます。各プロセスには、それ自身のプロセス固有ディレクトリがあります。

- システム・ディレクトリ LNM\$SYSTEM_DIRECTORY

システム・ディレクトリには、すべての共用可能テーブルの名前と自身のテーブル名が含まれます。LNM\$SYSTEM_DIRECTORY は、システムごとに 1 つしかありません。

これらのディレクトリには、テーブル名に反復的に変換する名前が含まれます。すべての論理名テーブル名と、テーブルに変換する論理名は、これらのディレクトリに保持されます。

論理名テーブルの親テーブルは、必ずしもディレクトリ・テーブルである必要はありません。つまり、この階層構造は論理名テーブル名の位置から明白です。

11.8.2 ディレクトリ・テーブルの構造の表示

論理名テーブルに対する論理名ディレクトリ・テーブルの関係を表示するには、次の例に示すように、SHOW LOGICAL/STRUCTURE コマンドを入力します。

```
$ 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)
  .
  .
  .
```

この例では、各論理名テーブル・ディレクトリ内に常駐する論理名テーブル名を示します。またこれは、LNM\$CLUSTER_TABLE と LNM\$SYSCLUSTER_TABLE との関係も示します。

11.9 省略時設定の論理名テーブル

システム・ディレクトリとプロセス・ディレクトリ・テーブルを含む、エグゼクティブによって作成された省略時設定のテーブルを表 11-1 に示します。

表 11-1 省略時設定の論理名テーブル

テーブル名	完全なテーブル名	論理名	説明
プロセス論理名テーブル			
プロセス・ディレクトリ	LNMSPROCESS_DIRECTORY	(他の論理名なし)	プロセス固有論理名テーブル名とテーブル名に反復的に変換する名前の定義を含む。
プロセス・テーブル	LNMSPROCESS_TABLE	LNMSPROCESS	SYSDISK および SY\$INPUT などのプロセス固有論理名を含む。
共用論理名テーブル			
システム・ディレクトリ	LNMS\$SYSTEM_DIRECTORY	(他の論理名なし)	共用可能論理名テーブルとテーブル名に反復的に変換する名前の定義を含む。
システム・テーブル	LNMS\$SYSTEM_TABLE	LNMS\$SYSTEM	システム内のすべてのプロセスに共用される名前を含む。SY\$LIBRARY および SY\$SYSTEM など。
クラスタ単位システム・テーブル	LNMS\$SYSCLUSTER_TABLE	LNMS\$SYSCLUSTER	OpenVMS Cluster システムのすべてのプロセスが共用する名前を含む。
クラスタ単位親テーブル	LNMS\$CLUSTER_TABLE	LNMS\$CLUSTER	LNMS\$SYSCLUSTER_TABLE を含む、すべてのクラスタ単位の論理名テーブルの親テーブル。
グループ・テーブル	LNMS\$GROUP_gggggg ¹	LNMS\$GROUP	UIC グループのすべてのプロセスが共用する名前を含む。
ジョブ・テーブル	LNMS\$JOB_xxxxxxx ²	LNMS\$JOB	ジョブ・ツリーのすべてのプロセスが共用する名前を含む。SY\$LOGIN および SY\$SCRATCH など。

¹文字列ggggggは、プロセスの UIC グループ番号を含む 6 桁の 8 進数を表す。

²文字列xxxxxxxは、ジョブ情報ブロックのアドレスである 8 桁の 16 進数を表す。

11.9.1 プロセス論理名ディレクトリ

ログイン時にプロセス論理名ディレクトリ・テーブル LNM\$PROCESS_DIRECTORY で作成されるプロセス固有論理名を表 11-2 に示します。

表 11-2 プロセス論理名ディレクトリの省略時設定の論理名

名前	説明
LNMSGROUP	LNMSGROUP_ggggggとして定義されるグループ論理名。ここでggggggはグループ番号を表す。LNMSGROUP_gggggg ¹ は UIC グループが使用する論理名テーブル。テーブル LNMSGROUP_ggggggは、システム・ディレクトリ・テーブル内でカタログ化される。したがって、LNMSGROUP はグループの論理名テーブルの名前に反復的に変換する論理名である。
LNMSJOB	LNMSJOB_xxxxxxxxとして定義されるジョブ論理名。ここでxxxxxxx ² はそれぞれのジョブ・ツリーに固有の数値を表す。LNMSJOB_xxxxxxxは、ジョブで使用する論理名テーブル。テーブル LNMSJOB_xxxxxxxは、システム・ディレクトリ・テーブル内でカタログ化される。したがって、LNMSJOB はジョブ論理名テーブルの名前に反復的に変換する論理名である。
LNMSPROCESS	プロセス論理名テーブルの名前である LNM\$PROCESS_TABLE に反復的に変換するプロセス論理名。
LNMSPROCESS_DIRECTORY	プロセス・ディレクトリ論理名テーブルの名前。

¹文字列ggggggは、プロセスの UIC グループ番号を含む 6 桁の 8 進数を表す。

²文字列xxxxxxxは、ジョブ情報ブロックのアドレスである 8 桁の 16 進数を表す。

11.9.2 プロセス論理名テーブル

システム上のすべてのプロセスには、LNMSPROCESS_TABLE という名前のプロセス論理名テーブルがあります。プロセス・テーブルにある名前を使用できるのは、そのプロセスとそれに従属するサブプロセスだけです。ログイン時に、システムはプロセスに対する論理名を作成し、それをそのプロセス・テーブルに配置します。

名前 LNM\$PROCESS を通じて間接的に LNM\$PROCESS_TABLE を参照することができます。この間接参照により、LNMSPROCESS を複数の等価名として再定義して、次の例で示すようにその中に各自のテーブルを 1 つ以上含めることができますようになります。

```
$CREATE/NAME_TABLE APPLICATION_NAMES
$DEFINE/TAB=LNMSPROCESS_DIRECTORY LNM$PROCESS APPLICATION_NAMES,
LNMSPROCESS_TABLE
```

省略時設定により、プロセス・テーブルには表 11-3 に示す論理名が含まれます。これらの論理名 SYS\$INPUT, SYS\$OUTPUT, SYS\$ERROR, SYS\$COMMAND は、プロセスパーマネント・ファイル(プロセスの処理中にオープンしたままのファイル)を参照します。プロセスパーマネント・ファイルについての詳細は、第 11.13 節を参照してください。

表 11-3 プロセス論理名テーブルの省略時設定の論理名

名前	説明
SYSS\$COMMAND	DCL が入力を読み込む初期ファイル (通常は使用するターミナル)。DCL が入力を読み込むファイルは、入力ストリームと呼ぶ。コマンド・インタプリタは SYSS\$COMMAND を使用して元の入力ストリームを“記憶”しておく。
SYSS\$DISK	ログイン時に設定された、または SET DEFAULT コマンドによって変更された省略時設定の装置。
SYSS\$ERROR	DCL が、警告、エラー、および重大なエラーによって生成されたエラー・メッセージを書き込む省略時設定の装置またはファイル。
SYSS\$INPUT	DCL が入力を読み込む省略時設定のファイル。
SYSS\$NET	DECnet for OpenVMS タスク間通信のターゲット・プロセスを起動するソース・プロセス。ターゲット・プロセスによってオープンされると、SYSS\$NET はプロセスがその相手とデータを交換できる論理リンクを表す。SYSS\$NET が定義されるのは、タスク間通信の期間中に限られる。
SYSS\$OUTPUT	DCL が出力を書き込む省略時設定のファイル (通常は使用するターミナル)。DCL が出力を書き込むファイルは、出力ストリームと呼ぶ。
TT	ターミナルに対する省略時設定の装置名。

11.9.3 システム論理名ディレクトリ

システム・ディレクトリ・テーブル LNM\$SYSTEM_DIRECTORY に含まれる省略時設定のシステム論理名を表 11-4 に示します。

表 11-4 システム論理名ディレクトリの省略時設定の論理名

名前	説明
LNMS\$CLUSTER	LNMS\$CLUSTER_TABLE に反復的に変換するクラスタ単位の親テーブルの論理名。
LNMS\$DCL_LOGICAL	LNMS\$FILE_DEV として定義される DCL 論理名。LNMS\$DCL_LOGICAL は、SHOW LOGICAL コマンド、SHOW TRANSLATION コマンド、FSTRNLNM レキシカル関数によって検索され表示される論理名テーブルのリストに反復的に変換する。省略時設定により、これらのコマンドは、プロセス、ジョブ、グループ、システム、およびクラスタ単位のシステム論理名テーブルをこの順に検索して表示する。
LNMS\$DIRECTORIES	LNMS\$PROCESS_DIRECTORY および LNM\$SYSTEM_DIRECTORY として定義されるディレクトリ論理名。
LNMS\$FILE_DEV	ファイル指定を処理する場合にシステムが検索する論理名テーブルのリストとして定義される検索リストの論理名。LNMS\$PROCESS、LNMS\$JOB、LNMS\$GROUP、LNM\$SYSTEM として定義され、システムは、プロセス、ジョブ、グループ、システム、およびクラスタ単位のシステム論理名テーブルをこの順に検索する。
LNMS\$GROUP	グループ・テーブル LNMS\$GROUP_gggggg ¹ に定義されるグループ論理名。
LNMS\$JOB	LNMS\$JOB_xxxxxxx ² として定義されるジョブ論理名。

¹文字列ggggggは、プロセスの UIC グループ番号を含む 6 桁の 8 進数を表す。

²文字列xxxxxxxは、ジョブ情報ブロックのアドレスである 8 桁の 16 進数を表す。

(次ページに続く)

表 11-4 (続き) システム論理名ディレクトリの省略時設定の論理名

名前	説明
LNMS\$PERMANENT_MAILBOX	LNMS\$SYSTEM として定義されるパーマネント・メールボックス論理名。パーマネント・メールボックスに関連付けられた論理名は、論理名 LNMS\$PERMANENT_MAILBOX が反復して変換する論理名テーブルに入力される。
LNMS\$SYSCLUSTER	LNMS\$SYSCLUSTER_TABLE に反復的に変換するクラスタ単位のシステム論理名テーブルの論理名。
LNMS\$SYSTEM	LNMS\$SYSTEM_TABLE, LNMS\$SYSCLUSTER に反復して変換するシステム論理名テーブル名。
LNMS\$TEMPORARY_MAILBOX	LNMS\$JOB として定義される一時メールボックス論理名。一時メールボックスに関連付けられた論理名は、論理名 LNMS\$TEMPORARY_MAILBOX が反復して変換する論理名テーブルに入力される。

11.9.4 共用論理名テーブル

ここでは、次の省略時設定の共用論理名テーブルについて説明します。

- クラスタ単位のシステム・テーブル
- クラスタ単位の親テーブル
- グループ・テーブル
- ジョブ・テーブル
- システム・テーブル

クラスタ単位のシステム・テーブル LNMS\$SYSCLUSTER_TABLE

LNMS\$SYSCLUSTER_TABLE は、クラスタ単位のシステム論理名テーブルの名前です。このテーブルには、クラスタのすべてのユーザが使用できる論理名が含まれています。

名前 LNMS\$SYSCLUSTER を通じて間接的に LNMS\$SYSCLUSTER_TABLE を参照することができます。間接参照により、LNMS\$SYSCLUSTER を複数の等価名として再定義して、その中に各自のテーブルを含めることができます。

クラスタ単位の親テーブル LNMS\$CLUSTER_TABLE

LNMS\$CLUSTER_TABLE は、LNMS\$SYSCLUSTER_TABLE を含む、すべてのクラスタ単位の論理名テーブルの親テーブルです。論理名 LNMS\$CLUSTER を使用してこれを参照します。

グループ・テーブル LNMS\$GROUP_gggggg

それぞれのグループ・テーブルの名前は LNMS\$GROUP_gggggg です (gggggg はユーザ ID コード[UIC]グループ番号を表します)。このテーブルにある名前は、同じ UIC グループ番号を持つすべてのユーザが使用できます。システム上のすべてのグループには、対応するグループ論理名テーブルがあります。

名前 LNMS\$GROUP を通じて間接的に LNMS\$GROUP_gggggg を参照することができます。間接参照により、LNMS\$GROUP_gggggg を複数の等価名として再定義して、その中に各自のテーブルを含めることができます。またこれは、自分の UIC グループ

ブ番号を記憶しておく必要を省き、最も新しく定義されたテーブルを必ず使用できるようにします。

ジョブ・テーブル LNM\$JOB_XXXXXXXX

それぞれのジョブ・テーブルの名前は LNM\$JOB_XXXXXXXX です (XXXXXXXX はジョブ・ツリーに対してシステムが定義したジョブ情報ブロック [JIB] アドレスを表します)。

ジョブ・テーブルには、ジョブ・ツリー、つまりプロセスとそのサブプロセスにあるすべてのプロセスで使用できる論理名が含まれます。システムには各ジョブ・ツリーに対して 1 つのジョブ・テーブルがあります。ジョブ・テーブルは共用可能なので、ジョブ・ツリー内のすべてのプロセスがアクセスできます。

名前 LNM\$JOB を通じて間接的に LNM\$JOB_XXXXXXXX を参照することができます。この間接参照により、LNM\$JOB を複数の等価名として再定義して、その中に各自のテーブルを含めることができます。さらに、LNM\$JOB を使用すると、JIB アドレスを見つける必要がなく、最も新しく定義されたテーブルを必ず使用できるようになります。

システムは、マウントされたディスク、マウントされたテープ、一時メールボックスに対して作成された論理名をジョブ論理名テーブルに配置します。さらに、システムは次の論理名を作成します。

- SYS\$LOGIN

ログイン時の省略時設定の装置およびディレクトリ。

- SYS\$LOGIN_DEVICE

ログイン時の省略時設定の装置。

- SYS\$REM_ID

DECnet for OpenVMS ネットワーク接続を通じて開始されたジョブに対して、ジョブが発信されたリモート・ノード上のプロセスの ID。OpenVMS オペレーティング・システムでは、代理ログインが使用可能な場合、この ID がプロセスのユーザ名です。代理ログインが使用できない場合には、これがプロセス ID (PID) 番号です。

(代理アカウントへの代理ログインにより、ユーザはアクセス制御文字列を指定せずにネットワーク上でファイルにアクセスできます。)

- SYS\$REM_NODE

DECnet for OpenVMS ネットワーク接続を通じて開始されたジョブに対して、ジョブが発信されたリモート・ノードの名前。

- SYS\$SCRATCH

一時ファイルが書き込まれる省略時設定の装置およびディレクトリ。

システム・テーブル LNM\$SYSTEM_TABLE

システム・テーブルの名前は LNM\$SYSTEM_TABLE です。システム・テーブルには、システム・レベルでシステムのすべてのユーザが使用できる論理名が含まれます。

システム・テーブルは通常、LNM\$SYSTEM を通じて間接的に参照することができ、これは検索リスト LNM\$SYSTEM_TABLE、LNM\$SYSCLUSTER として定義されます。LNM\$SYSTEM を使用して、このノードにローカルなシステム名と、クラスタ上のすべてのノードに共通なシステム名を含めることができます。

システムの起動時にシステム・テーブルで自動的に定義される論理名を表 11-5 に示します。

表 11-5 システム論理名テーブルの省略時設定の論理名

名前	説明	省略時アドレス
DBG\$INPUT	プロセス・レベルで SYS\$INPUT に等しいと定義される、OpenVMS Debugger の省略時入力ストリーム。	適用せず
DBG\$OUTPUT	プロセス・レベルで SYS\$OUTPUT に等しいと定義される、OpenVMS Debugger の省略時出力ストリーム。	適用せず
SYS\$COMMON	SYS\$SYSDEVICE:[SYSn.SYSCOMMON]、ここで n はプロセッサのルート・ディレクトリ番号。 共通部分に対する装置およびディレクトリ名。	
SYS\$ERRORLOG	エラー・ログ・データ・ファイルの装置およびディレクトリ名。	SYS\$SYSROOT:[SYSERR]
SYS\$EXAMPLES	システム例の装置およびディレクトリ名。	SYS\$SYSROOT:[SYSHLP.EXAMPLES]
SYS\$HELP	システム・ヘルプ・ファイルの装置およびディレクトリ名。	SYS\$SYSROOT:[SYSHLP]
SYS\$INSTRUCTION	システム命令データ・ファイルの装置およびディレクトリ名。	SYS\$SYSROOT:[SYSCBI]
SYS\$LIBRARY	システム・ライブラリの装置およびディレクトリ名。	SYS\$SYSROOT:[SYSLIB]
SYS\$LOADABLE_IMAGES	オペレーティング・システムのエグゼクティブでロード可能なイメージ、デバイス・ドライバ、その他エグゼクティブでロードされたコードの装置およびディレクトリ。	SYS\$SYSROOT:[SYS\$LDR]
SYS\$MAINTENANCE	システム保守ファイルの装置およびディレクトリ名。	SYS\$SYSROOT:[SYSMAINT]
SYS\$MANAGER	システム・マネージャ・ファイルの装置およびディレクトリ名。	SYS\$SYSROOT:[SYSMGR]
SYS\$MESSAGE	システム・メッセージ・ファイルの装置およびディレクトリ名。	SYS\$SYSROOT:[SYSMSG]

(次ページに続く)

表 11-5 (続き) システム論理名テーブルの省略時設定の論理名

名前	説明	省略時アドレス
SYSSNODE	DECnet for OpenVMS がシステム上でアクティブであり、ネットワークに接続中の場合、ローカル・システムに対するネットワーク・ノード名。	適用せず
SYSSPROCUMP	イメージ・ダンプが書き込まれるディレクトリ (ユーザが設定)。	省略時の設定なし
SYSSSHARE	システム共用イメージのの装置およびディレクトリ名。	SYSSSYSROOT:[SYSLIB]
SYSSSPECIFIC	SYSSSYSDEVICE のノード固有部分に対する装置およびディレクトリ名。	SYSSSYSDEVICE:[SYSn.]、ここでnはプロセッサのルート・ディレクトリ番号。
SYSSSTARTUP	システム起動ファイルの装置およびディレクトリ名。	まず SYSSSYSROOT:[SYSSSTARTUP]、次に SYSSMANAGER を指す検索リスト。
SYSSSYSDEVICE	システム・ディレクトリを含むシステム・ディスク。	通常は SYSSDISK
SYSSSYSROOT	システム・ディレクトリに対する装置およびディレクトリ名。	まず SYSSSYSDEVICE:[SYSn.] (nはプロセッサのルート・ディレクトリ番号)、次に SYSSCOMMON を指す検索リスト。
SYSSSYSTEM	オペレーティング・システムのプログラムとプロシージャの装置およびディレクトリ。	SYSSSYSROOT:[SYSEXE]
SYSSTEST	環境設定テスト・パッケージ (UETP) ファイルの装置およびディレクトリ名。	SYSSSYSROOT:[SYSTEST]
SYSSUPDATE	システム更新ファイルの装置およびディレクトリ名。	SYSSSYSROOT:[SYSUPD]

11.9.5 共用論理名テーブルの省略時保護

オペレーティング・システムから提供される共用論理名テーブルは、省略時の保護設定付きで作成されます。それぞれの共用論理名テーブルに対する省略時保護を表 11-6 に示します。

表 11-6 共用論理名テーブルの省略時保護

テーブルの種類	テーブル名	省略時の保護
ジョブ・テーブル	LNMSJOB_xxxxxxxx ¹	SYSTEM=RWCD, OWNER=RWCD, GROUP=NO ACCESS, WORLD=NO ACCESS
グループ・テーブル	LNMSGROUP_gggggg ²	SYSTEM=RWCD, OWNER=R, GROUP=R, WORLD=NO ACCESS

¹文字列xxxxxxxxは、ジョブ情報ブロックのアドレスである 8 桁の 16 進数を表す。

²文字列ggggggは、プロセスの UIC グループ番号を含む 6 桁の 8 進数を表す。

(次ページに続く)

表 11-6 (続き) 共用論理名テーブルの省略時保護

テーブルの種類	テーブル名	省略時の保護
システム・テーブル	LNMS\$SYSTEM_TABLE	SYSTEM=RWC, OWNER=RWC, GROUP=R, WORLD=R
クラスタ単位システム・テーブル	LNMS\$SYSCUSTER_TABLE	SYSTEM=RWC, OWNER=RWC, GROUP=R, WORLD=R
クラスタ単位親テーブル	LNMS\$CLUSTER_TABLE	SYSTEM=RWC, OWNER=RWC, GROUP=R, WORLD=R
ユーザが作成したテーブル	ユーザが指定	SYSTEM=RWCD, OWNER=RWCD, GROUP=NO ACCESS, WORLD=NO ACCESS

11.9.6 共用論理名管理のための特権とアクセス必要条件

表 11-7 には、それぞれの共用論理名テーブルにおける論理名の作成，削除，読み込み (変換) に必要な特権とアクセス権を示します。特権，アクセス・タイプ，アクセス制御の詳細については，このマニュアルの第 10 章を参照してください。

表 11-7 共用論理名の作業に必要な特権またはアクセス・タイプ

論理名が存在するテーブル	テーブル名	作業	必要な特権またはアクセス・タイプ
ジョブ・テーブル	LNMS\$JOB_xxxxxxx ¹	論理名の作成または削除	論理名を作成する，または削除するテーブルに対して，WRITE (W) アクセス
		論理名の読み込み (変換)	論理名が存在するテーブルに対して，READ (R) アクセス
グループ・テーブル	LNMS\$GROUP_gggggg ²	論理名の作成または削除	論理名を作成する，または削除するテーブルに対して，WRITE (W) アクセス，または，GRPNAM 特権
		論理名の読み込み (変換)	論理名が存在するテーブルに対して，READ (R) アクセス
システム・テーブル	LNMS\$SYSTEM_TABLE	論理名の作成または削除	システムの UIC グループ番号 (0 からシステム・パラメータ MAXSYSGROUP までの間)，または，SYSNAM 特権
		論理名の読み込み (変換)	論理名が存在するテーブルに対して，READ (R) アクセス
クラスタ単位システム・テーブル	LNMS\$SYSCUSTER_TABLE	論理名の作成または削除	システムの UIC グループ番号 (0 からシステム・パラメータ MAXSYSGROUP までの間)，または，SYSNAM 特権
		論理名の読み込み (変換)	論理名が存在するテーブルに対して，READ (R) アクセス
クラスタ単位親テーブル	LNMS\$CLUSTER_TABLE	論理名の作成または削除	システムの UIC グループ番号 (0 からシステム・パラメータ MAXSYSGROUP までの間)

¹文字列xxxxxxxは，ジョブ情報ブロックのアドレスである 8 桁の 16 進数を表す。

²文字列ggggggは，プロセスの UIC グループ番号を含む 6 桁の 8 進数を表す。

(次ページに続く)

表 11-7 (続き) 共用論理名の作業に必要な特権またはアクセス・タイプ

論理名が存在するテーブル	テーブル名	作業	必要な特権またはアクセス・タイプ
ユーザ作成テーブル	ユーザ指定	論理名の読み込み (変換)	論理名が存在するテーブルに対して、 READ (R) アクセス
		論理名の作成または削除	論理名を作成する、または削除するテーブルに対して、 WRITE (W) アクセス
		論理名の読み込み (変換)	論理名が存在するテーブルに対して、 READ (R) アクセス

11.10 論理名テーブルの作成

CREATE/NAME_TABLE コマンドは、論理名テーブルを作成し、それをディレクトリ論理名テーブルの 1 つにカタログ化します。論理名テーブルを識別する、または論理名テーブルに反復的に変換する論理名は、常にディレクトリ論理名テーブルの 1 つに入力する必要があります。

11.10.1 プロセス固有の論理名テーブルの作成

プロセスに固有の論理名テーブルを作成するには、テーブルを LNMSPROCESS_DIRECTORY (省略時設定) に作成します。

ディレクトリ・テーブルの名前には 1 ~ 31 の文字を含むことができます。大文字英数字、ドル記号(\$), アンダスコア(_)だけが有効です。小文字のテーブル名を指定すると、自動的に大文字に変換されます。

次の例では、TAX という名前のプロセス固有論理名テーブルを作成し、論理名 CREDIT の定義をテーブル内に配置し、テーブルの作成をチェックします。SHOW LOGICAL/TABLE コマンドにより、表示する論理名テーブルを指定することができます。

```
$ CREATE/NAME_TABLE TAX
$ DEFINE/TABLE=TAX CREDIT [ACCOUNTS.CURRENT]CREDIT.DAT
$ SHOW LOGICAL/TABLE=TAX CREDIT

"CREDIT" = "[ACCOUNTS.CURRENT]CREDIT.DAT" (TAX)
```

ファイルのフックアップ中に新しいテーブルをシステムに自動的に検索させるようにするため、次の例に示すように LNMSPROCESS を再定義することができます。

```
$ DEFINE/TABLE=LNMSPROCESS_DIRECTORY LNMSPROCESS LNMSPROCESS_TABLE, TAX
```

11.10.2 共用可能論理名テーブルの作成

共用可能論理名テーブルを作成するには、/PARENT_TABLE 修飾子を使用し、共用可能テーブルを指定します。例を次に示します。

```
$ CREATE/NAME_TABLE/PARENT_TABLE=LNMS$SYSTEM_DIRECTORY NEWTAB
```

11.10.3 クラスタ単位の論理名テーブルの作成

他の共用可能論理名テーブルを作成する場合と同様の方法で、クラスタ単位の論理名テーブルを作成することができます。クラスタ単位の論理名テーブルは、共用論理名テーブルの特殊なタイプで、すべての共用可能論理名テーブルに適用する特権とアクセス必要条件に従います (第 11.10.4 項を参照してください)。

次の例では、クラスタ単位の論理名テーブルを作成する方法を示します。

```
$ CREATE/NAME_TABLE/PARENT_TABLE=LNMS$CLUSTER_TABLE -  
_$ new_clusterwide_logical_name_table
```

新しいクラスタ単位の論理名テーブルに常駐するクラスタ単位の論理名を作成するには、次の例に示すように、DEFINE コマンドで新しいクラスタ単位の論理名を定義して、新しいテーブルの名前を/TABLE 修飾子で指定します。

```
$ DEFINE/TABLE=new_clusterwide_logical_name_table logical_name -  
_$ equivalence_string
```

11.10.4 特権とアクセスの必要条件

特権を持つユーザは、特殊目的のために共用論理名テーブルを作成できます。例えば、アプリケーションで 1 つ以上の共用論理名テーブルを作成すると、ファイルの記憶位置などの情報をアプリケーション・ユーザとやり取りすることができます。

```
$ CREATE/NAME_TABLE APPX_FILE_LOCATOR /PARENT=LNMS$SYSTEM_DIRECTORY -  
_$ /PROTECTION = (S:RWD,O:RWD,G:R,W:R)
```

共用論理名テーブルを作成するには、次のものを備えている必要があります。

- 親テーブルへの CREATE (C) アクセス
- SYSPRV 特権または LNMS\$SYSTEM_DIRECTORY への WRITE (W) アクセス

共用論理名テーブルを削除するには、次のものを備えている必要があります。

- テーブルへの DELETE (D) アクセス
- SYSPRV 特権または LNMS\$SYSTEM_DIRECTORY への WRITE (W) アクセス

11.10.5 省略時保護の変更

オペレーティング・システムは、システムまたはユーザが作成する共用論理名テーブルに対して、省略時の保護を提供します。この省略時保護はシステム管理者またはテーブル所有者が変更できるセキュリティ・プロファイルに格納されます。詳細については、『OpenVMS Guide to System Security』を参照してください。

自分が作成したテーブルの省略時保護は、次の方法で変更できます。

- DCL の CREATE/NAME_TABLE コマンドで/PROTECTION 修飾子を使用する方法。このコマンドでは UIC ベースの保護を設定できる。
- すでに作成されているテーブルに対して、ACL エディタまたは SET SECURITY /ACL/OBJECT_TYPE=LOGICAL_NAME_TABLE コマンドで ACL 保護を適用する方法。

共用論理名テーブルの ACL は、システムのブート間には保存されません。システムがブートされるごとにこれらの論理名テーブルで ACL を再設定する必要があります。

共用論理名テーブルへの ACL 保護の適用についての詳細は、『OpenVMS DCL ディクショナリ』の SET SECURITY/ACL コマンドを参照してください。

11.10.6 論理名テーブルへの制限値の設定

制限は、所定の論理名テーブルが消費することのできるシステム資源の量を制限するために使用します。プロセス、グループ、およびシステム論理名テーブルには、無限の制限があります。省略時設定により、論理名テーブルを作成すると、これも無限の制限を持つことになります。

制限値を指定して、作成する論理名テーブルのサイズをバイト単位で制限することができます。論理名が作成される前に、そのデータ構造のサイズがテーブルの残りの制限値に対してチェックされます。新しいエントリに使用できる制限値が十分でないと、システムはエラー・メッセージを表示します。

いったんテーブルに制限値を設定してしまうと、変更することはできません。テーブルに余地がなくなった場合は、DEASSIGN コマンドを使用して古い論理名を削除します。これで新しい論理名のためにスペースを解放できます。

次の例では、論理名テーブル ABC が作成され、500 バイトの制限値を与えられます。

```
$ CREATE/NAME_TABLE/QUOTA=500 ABC
```

11.10.6.1 ジョブ・テーブルの制限値の設定

ジョブ論理名テーブルは、共用可能テーブルです。ジョブ論理名テーブルの制限値は、テーブルの作成時に設定されます。制限値は、次の基準の1つまたはそれ以上で決まります。

- システム利用者登録ファイル SYSUAF.DAT でユーザに設定されている JTQUOTA 値 (プロセスによって起動された最初のイメージがシステム・イメージ LOGINOUT であった場合)
- Create Process (\$CREPRC) システム・サービスへの呼び出しで指定される PQL\$JTQUOTA 制限リスト値
- 独立プロセスを作成するために使用する RUN コマンド上の /JOB_table_QUOTA 修飾子の値
- SYSGEN パラメータ PQL_DJTQUOTA (先行する条件のいずれも適用しない場合)。このパラメータの標準の省略時設定値は、1024 バイトですが、システム・マネージャはこれを変更することができます。System Generation ユーティリティ (SYSGEN) は、パラメータ PQL_DJTQUOTA (省略時設定のジョブ論理名テーブルの制限) および PQL_MJTQUOTA (最小ジョブ論理名テーブルの制限) の値を表示して設定するために使用されます。

ジョブ論理名テーブルに対する制限値 0 は、制限がないことを示します。あらゆる実用的な目的からも、制限は無限にされます。

11.11 論理名変換の順序の変更

LNMSFILE_DEV は、検索される論理名テーブル、および論理名変換の検索順序を定義します。一般には、省略時設定の検索順序を変更する必要はありません。ただし、新しい、プロセス固有の論理名テーブルの名前を追加して、LNMSFILE_DEV で指定されたテーブルより先に検索されるようにしたい場合もあります。同様に、システム管理者が、1つ以上の共用可能論理名テーブルの名前を追加して、LNMSFILE_DEV で指定されたテーブルより先に検索されるようにしたい場合もあります。

システムが最初に検索する論理名の新しいテーブルで LNMSFILE_DEV のプロセス固有定義を作成するには、次の手順にしたがいます。

1. 新しい論理名を含むファイルを作成します。
2. この新しいファイルを新しい論理名テーブルに変換します。
3. プロセス論理名ディレクトリ・テーブルを親テーブルとして指定して、LNMSFILE_DEV の固有定義を作成します。
4. LNMSFILE_DEV の固有定義のテーブル名リストの始めに新しい論理名テーブル名を追加します。

次の例では、新しい論理名テーブル、NEWTAB が作成され、最初に検索されるテーブルに NEWTAB をリストして LNM\$FILE_DEV のプロセス固有定義が作成されます。

```
$ CREATE/NAME_TABLE NEWTAB
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$FILE_DEV -
_$ NEWTAB, LNM$PROCESS, LNM$JOB, LNM$GROUP, LNM$SYSTEM
```

上記の例では、次の理由からシステムは NEWTAB テーブルを最初に検索します。

- LNM\$FILE_DEV のプロセス固有定義が、省略時設定のシステム・バージョンの代わりに使用されている。
- LNM\$FILE_DEV 内では、NEWTAB が他の論理名テーブルより前にリストされている。

LNM\$FILE_DEV のシステム定義に新しい論理名テーブルを追加する場合には、SYSNAM または SYSPRV 特権を備えている必要があります。

次の例は、NEWTAB がプロセス固有テーブルではなく共用可能テーブルとして作成される点を除いては、以前のものと同様です。

```
$ 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
```

LNM\$FILE_DEV によって定義された検索リストから論理名テーブルを削除することもできます。次の例では、LNM\$FILE_DEV のプロセス固有定義が、プロセスおよびシステム論理名テーブルだけを含むように作成されます。プロセス固有定義には LNM\$JOB および LNM\$GROUP が含まれないので、論理名を変換する必要のある後続のコマンドはそのジョブまたはグループ・テーブルを検索しません。

```
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY -
_$ LNM$FILE_DEV LNM$PROCESS, LNM$SYSTEM
```

11.12 論理名テーブルの削除

論理名テーブルを削除するには、それを含むテーブル(システムまたはプロセス・ディレクトリ論理名テーブル)とテーブルの名前を指定します。親論理名テーブルを削除すると、子孫のテーブルにあるすべての論理名(と子孫のテーブル自身)が削除されます。

共用可能論理名テーブルを削除するには、テーブルへの DELETE アクセスまたは SYSPRV 特権を備えている必要があります。

次の例では、コマンドが論理名 WORKFILE を削除します。

```
$ DEASSIGN WORKFILE
```

次の例では、コマンドがプロセス・ディレクトリ・テーブルから論理名テーブル TAX を削除します。

```
$ DEASSIGN/TABLE=LNM$PROCESS_DIRECTORY TAX
```

11.13 プロセスパーマネント論理名

DCL は、ログイン時にプロセスパーマネント論理名を作成します。これらの名前の定義は、プロセスの期間中にわたり有効です。これらの論理名を再割り当てすることはできません (DEFINE コマンドに別の等価文字列を指定して) これらを再定義することはできますが、再定義された名前が後に再割り当てされると、プロセスパーマネント名は再設定されます。

次のプロセスパーマネント論理名が使用できます。

- SYSS\$INPUT
省略時設定の入力装置またはファイルを参照する論理名。
- SYSS\$OUTPUT
省略時設定の出力装置またはファイルを参照する論理名。
- SYSS\$ERROR
システムがメッセージを書き込む省略時設定の装置またはファイルを参照する論理名。
- SYSS\$COMMAND
ログイン時に SYSS\$INPUT の値を参照する論理名。

11.13.1 会話型とバッチ処理の間の等価名の相違点

システムを会話型で使用する場合には、DCL が SYSS\$INPUT, SYSS\$OUTPUT, SYSS\$ERROR, SYSS\$COMMAND を使用するターミナルに等しいと定義します。ただし、コマンド・プロシージャを実行してバッチ・ジョブを発行すると、DCL はこれらの論理名に対して新しい等価文字列を作成します。

コマンド・プロシージャを会話型で実行すると、次の状況が生じます。

- SYSS\$INPUT は、コマンド・プロシージャに等しいと定義されます。したがって、DCL はコマンド・プロシージャからデータを取得します。この割り当ては一時的なものです。コマンド・プロシージャが終了すると、SYSS\$INPUT はその元の値を取り戻します。
- SYSS\$OUTPUT, SYSS\$COMMAND, SYSS\$ERROR は、ターミナルに等しいと定義されたままです。

バッチ・ジョブを発行すると、次の状況が生じます。

- SYSS\$INPUT および SYSS\$COMMAND は、バッチ・ジョブのコマンド・プロシージャに等しいと定義されます。
- SYSS\$OUTPUT および SYSS\$ERROR は、バッチ・ジョブのログ・ファイルに等しいと定義されます。

コマンド・プロシージャをネストすると(つまり、他のコマンド・プロシージャを実行するコマンド・プロシージャを記述する場合)、SYSS\$INPUT の等価文字列が現在実行中のコマンド・プロシージャを指すように変更されます。ただし、SYSS\$OUTPUT、SYSS\$ERROR、SYSS\$COMMAND の等価文字列は、明示的に変更を加えない限り変化はありません。

さらに、ファイルをオープンするコマンドを入力すると、DCL はそのファイルをプロセスパーマメント・ファイルとしてオープンします。たとえば、OPEN コマンドでファイルをオープンすると、このファイルはプロセスパーマメント・ファイルとしてオープンされます。ファイルを明示的にクローズするか、またはログアウトするまで、このファイルはオープンされた状態です。

プロセスパーマメント・ファイルは、メモリの特殊な領域に格納されます。同時に多数のファイルをオープンした状態にしておくと、この領域を消費してしまう可能性があるので注意してください。そのような状況が発生したら、ファイルをいくつかクローズ(するかまたはログアウト)してください。

11.13.2 プロセスパーマメント論理名を使用するファイル入出力の切り換え

プロセスパーマメント論理名を使用して、ファイル入出力を切り換えることができます。コマンド・プロシージャで、これらの名前を使用してターミナルからのデータを読み込み、データを表示することができます(第 13 章および第 14 章を参照してください)。DCL は、SYSS\$INPUT および SYSS\$COMMAND に対する新しい定義を無視するので注意してください。

OpenVMS バージョン 7.1 では、DCL PIPE コマンドが導入されました。PIPE コマンドは、ファイル入出力を切り換える代替方法です。PIPE コマンドについては、『OpenVMS DCL デクショナリ: N-Z』を参照してください。

11.13.2.1 SYSS\$INPUT の再定義

SYSS\$INPUT を再定義して、コマンド・プロシージャによって起動されたイメージが端末や他のファイルからの入力を読み込むようにすることができます。DCL は常に省略時設定の入力ストリームから入力を取得するので、DCL は SYSS\$INPUT の再定義を無視します。

次の例で、コマンドは新しいコマンド・プロシージャ・ファイルの一部です。DEFINE コマンドは、SYSS\$INPUT を SYSS\$COMMAND に再定義します。SYSS\$COMMAND は、ログイン時の最初の入力ストリームであるターミナルを

参照します。この新しい定義で、コマンド・プロシージャによって起動されたイメージは、コマンド・プロシージャ・ファイルから (省略時設定) ではなくターミナルから入力を取得しますが、これは特定の期間に限られます。

/USER_MODE 修飾子は、SYSSINPUT が次のイメージの期間だけ再定義されることをコマンド・プロシージャに知らせます。この例では、次のイメージはエディタです。エディタが終了すると、SYSSINPUT はその省略時設定値を再開します。この場合、省略時設定値は、コマンド・プロシージャ・ファイルです。

```
$ DEFINE/USER_MODE SYSSINPUT SYSSCOMMAND  
$ EDIT/TPU MYFILE.DAT  
.  
.  
.
```

11.13.2.2 SYSSOUTPUT の再定義

SYSSOUTPUT を再定義して、省略時設定の装置から別のファイルに出力を切り換えることができます。SYSSOUTPUT を再定義すると、システムは論理名割り当てで指定した名前でファイルをオープンします。SYSSOUTPUT を定義すると、すべての後続の出力は新しいファイルに出力されます。

忘れずに SYSSOUTPUT を割り当て解除してください。解除しないと、出力は指定したファイルに引き続き書き込まれます。ユーザ・モードで SYSSOUTPUT を再定義して (DEFINE/USER_MODE で)、出力をイメージから切り換えることもできます。この定義は、次のコマンド・イメージが実行されるまでしか有効ではありません。いったんコマンド・イメージが実行されると (つまり、出力がファイルに取り込まれると)、論理名 SYSSOUTPUT はその省略時設定値を再開します。

ログイン時に、システムは SYSSOUTPUT という 2 つの論理名を作成します。1 つの名前はエグゼクティブ・モードで作成され、もう 1 つの名前はスーパーバイザ・モードで作成されます。SYSSOUTPUT を再定義してスーパーバイザ・モード論理名を優先することができます。スーパーバイザ・ノード名を再割り当てすると、システムは、エグゼクティブ・モードの等価文字列を使用して SYSSOUTPUT をスーパーバイザ・モードで再定義します。エグゼクティブ・モード名は、割り当て解除することはできません。

SYSSOUTPUT をファイルに再定義すると、出力が指定したファイルに向けられていても、論理名にはファイル指定の装置部分しか含まれません。

SYSSOUTPUT を再定義したとき、システムが指定したファイルをオープンできない場合には、エラー・メッセージが表示されます。

SYSSOUTPUT を再定義すると、ほとんどのコマンドは出力を既存のバージョンのファイルに向けます。ただし、新しいバージョンのファイルを作成してから出力を書き込む特定のコマンドもあります。

次の例では、SYSS\$OUTPUT が MYFILE.LIS として定義されてから、SHOW DEVICES コマンドが入力されます。SHOW DEVICES によって生成される表示は、ターミナルではなく現在のディレクトリの MYFILE.LIS に向けられます。他のテキスト・ファイルの場合と同様に、このデータを操作することができます。

```
$ DEFINE SYSS$OUTPUT MYFILE.LIS
$ SHOW DEVICES
```

以下の例では、SYSS\$OUTPUT がファイル TEMP.DAT に再定義されています。SYSS\$OUTPUT が再定義されると、DCL からの出力とイメージからの出力はファイル TEMP.DAT に向けられます。SHOW LOGICAL コマンドからの出力と SHOW TIME コマンドからの出力も、TEMP.DAT に向けられます。SYSS\$OUTPUT が再割り当てされると、システムはファイル TEMP.DAT をクローズして、SYSS\$OUTPUT をターミナルに再定義します。TYPE コマンドが入力されると、TEMP.DAT に集められた出力がターミナルに表示されます。

```
$ DEFINE SYSS$OUTPUT TEMP.DAT
$ SHOW LOGICAL SYSS$OUTPUT
$ SHOW TIME
$ DEASSIGN SYSS$OUTPUT
$ TYPE TEMP.DAT
"SYSS$OUTPUT" = "DISK1:" (LNM$PROCESS_TABLE)
06-MAY-1998 13:26:53
```

SYSS\$OUTPUT が再定義されると、等価文字列には装置名 DISK1 が含まれますが、ファイル指定全体は含まれません。

11.13.2.3 SYSS\$ERROR の再定義

SYSS\$ERROR を再定義して、エラー・メッセージを指定するファイルに指向することができます。ただし、SYSS\$ERROR を再定義したためこれが SYSS\$OUTPUT と異なる場合 (または SYSS\$ERROR を同時に再定義せずに SYSS\$OUTPUT を再定義した場合) には、DCL コマンドは SYSS\$ERROR と SYSS\$OUTPUT の両方に、情報、警告、エラー、重大エラーメッセージを送ります。このため、SYSS\$ERROR の定義によって指示されたファイルで 1 回、そして SYSS\$OUTPUT によって指示されたファイルで 1 回と、これらのメッセージを合計 2 回受け取ることになります。正常終了のメッセージが送られるのは、SYSS\$OUTPUT に指示されたファイルだけです。

標準のエラー表示メカニズムを使用する DCL コマンドとイメージは、SYSS\$ERROR が SYSS\$OUTPUT と異なっても、SYSS\$ERROR と SYSS\$OUTPUT の両方にエラー・メッセージを送ります。ただし、SYSS\$ERROR を再定義してから SYSS\$ERROR を参照するイメージを実行する場合、イメージがエラー・メッセージを送るのは、SYSS\$ERROR で指示されたファイルだけです。これは、SYSS\$ERROR が STSS\$OUTPUT と異なっている場合でも該当します。

11.13.2.4 SYS\$COMMAND の再定義

SYS\$COMMAND を再定義することはできますが、DCL はその定義を無視します。DCL は常に、最初の入力ストリームに対して省略時の定義を使用します。ただし、SYS\$COMMAND を参照するイメージを実行する場合には、イメージは新しい定義を使用することができます。

シンボル，コマンド，式の定義

シンボルは，数値，文字，論理値 (真または偽など) を表す名前です。DCL コマンド行でシンボルを使用すると，DCL は，シンボルに対応する値に置き換えてからコマンドを実行します。

パラメータ，修飾子，または値をいくつも指定しなければならない DCL コマンド行を入力するのは面倒で時間がかかります。DCL との会話を簡単にし，時間を節約するためには，頻繁に使用するコマンドの代わりに使用するシンボルを設定します。

また，コマンド・プロシージャ内でシンボルを使用して，特定のデータ型を収集したり，格納したり，操作することもできます。コマンド・プロシージャについての詳細は，第 13 章と第 14 章を参照してください。

本章では，次のことについて説明します。

- シンボルの使用
- シンボルの表示
- 他のシンボルに対するシンボルの使用
- シンボルによるデータの格納と操作
- 文字列
- 数値と式の使用方法
- 論理値と式の使用方法
- 式での値タイプの変換
- シンボル・テーブル
- シンボルの値のマスク
- シンボル置換
- コマンド処理の 3 つのフェーズ
- シンボルを使用するための別の方法: 自動的なフォーリン・コマンド

その他の情報については，次のマニュアルを参照してください。

- シンボルとその使い方の追加情報については，『OpenVMS DCL ディクショナリ』
- 新しいコマンドの定義方法の追加情報については，『OpenVMS Command Definition, Librarian, and Message Utilities Manual』

12.1 シンボルについて

シンボルは，次のように使用できます。

- コマンド，パラメータ，コマンド行の同意語として使用する。長いコマンド行を入力する代わりにシンボルを使用できる。
- シンボル名を入力するだけでイメージを実行するフォーリン・コマンドを定義する。このコマンドは，DCL に未知のものなので「フォーリン」と形容される。
- コマンド・プロシージャ・ファイルで，変数の条件付き実行や置換などのプログラミング・タスクを実行する。

シンボルは，式の中の変数として使用したり，コマンド・プロシージャとの間でパラメータを渡すのに使用したりできる。また，READ，WRITE，INQUIRE などの DCL コマンドは，シンボルを使用してデータ・レコードを表す。

たとえば，頻繁にアクセスするディレクトリに省略時の値を設定するシンボルを作成することができます。次のコマンドは，省略時の値を WORK1:[JONES.WORK] ディレクトリに設定する WORK シンボルを定義して使用方法を示しています。

```
$ WORK ::= SET DEFAULT DISK1:[JONES.WORK]
$ WORK
$ SHOW DEFAULT
  DISK1:[JONES.WORK]
```

12.1.1 論理名とシンボルの違い

論理名とシンボルは似ていますが，使用目的は異なります。論理名とシンボルの機能，用途，その他の性質の違いを次の表に示します。

性質	論理名	シンボル
機能	デバイス，ディレクトリ，ファイル，キュー，その他のシステム・オブジェクトの指定を表す。	コマンドまたはコマンド文字列の一部を表す。
用途	デバイス，ディレクトリ，ファイル，キュー，その他のシステム・オブジェクトの完全な指定の代わりに使用する。論理名は，変換のためにファイル・システムに渡すコマンド文字列パラメータの一部として使用しなければならない。	コマンド文字列の代わりに使用する。シンボルは，コマンド言語インタプリタによって変換されるコマンド文字列の最初の語として使用しなければならない。
格納	プロセス，ジョブ，グループ，またはシステム論理名テーブルに格納される。第 11.10 節を参照。	グローバル・シンボル・テーブルとローカル・シンボル・テーブルに格納される。第 12.10 節を参照。
作成	論理名を作成するには ASSIGN コマンドまたは DEFINE コマンドを使用する。第 11.3 節を参照。	シンボルを作成するには割り当て文 (= または ==) を使用する。第 12.2 節を参照。

性質	論理名	シンボル
表示	論理名を表示するには SHOW LOGICAL コマンドまたは SHOW TRANSLATION コマンドを使用する。第 11.6 節を参照。	シンボルを表示するには SHOW SYMBOL コマンドを使用する。第 12.3 節を参照。
削除	論理名を削除するには DEASSIGN コマンドを使用する。第 11.4 節を参照。	シンボルを削除するには DELETE /SYMBOL コマンドを使用する。第 12.2.5 項を参照。

12.2 シンボルの使用

ローカルとグローバルの 2 つのタイプのシンボルを作成できます。ローカル・シンボルには、現在のコマンド・レベルから、および現在のコマンド・レベルで実行されるコマンド・プロシージャからアクセスできます。グローバル・シンボルには、すべてのコマンド・レベルからアクセスできます。

シンボルは、文字列、数値、レキシカル関数、論理値、別のシンボルを使用して定義できます。シンボル名の長さは 1 ~ 255 文字とし、先頭の文字が英字、アンダスコア(_)，ドル記号(\$)でなければなりません。シンボル名の中では、小文字も大文字もすべて大文字として扱われます。

シンボルを作成するには、割り当て文 (=または==)または文字列割り当て (:=または:=) を使用します。文字列割り当てを使用すると、すべての英字が大文字に変換され、複数のスペースやタブが 1 つのスペースに圧縮されます。文字列割り当ては、DCL コマンドを表すシンボルを作成したり、フォーリン・コマンドを定義したりするのに使用できます (いずれの場合も 255 文字が上限です)。文字列割り当ての中で文字列が 2 行にわたる場合には、ハイフンを使用します。

READ と INQUIRE コマンドを使用してもシンボルを作成することができます (第 13 章と第 14 章を参照)。

ローカル・シンボルの作成

次の例では、ローカル・シンボル SS が DCL コマンド SHOW SYMBOL に割り当てられます。

```
$ SS = "SHOW SYMBOL"
```

次の例では、ローカル・シンボル DB が DCL コマンド DIRECTORY ACCOUNTS:[BOLIVAR]に割り当てられます。

```
$ DB := DIRECTORY ACCOUNTS:[BOLIVAR]
```

グローバル・シンボルの作成

次の例では，グローバル・シンボル DC は，DCL コマンド行を表すために使用されています。DCL コマンド DIRECTORY は，シンボル名の入力時に指定の修飾子付きで実行されます。

```
$ DC == "DIRECTORY/SIZE=ALL DISK1:[JONES.TAX]MONEY.LIS"
```

次の例では，グローバル・シンボル READY は DCL コマンド行を表すために使用されています。このシンボル名を入力すると，DCL の PRINT コマンドが指定された修飾子と一緒に実行されます。

```
$ READY ::= PRINT/CONFIRM/QUEUE=AKI$LN03/NOTIFY/RESTART  
$ READY FILE.DAT
```

12.2.1 DCL コマンドを表すシンボルの使用

DCL コマンドを表すシンボルは，ログイン・コマンド・ファイル (LOGIN.COM) で定義することも，DCL レベルで会話形式で定義することもできます。ログイン・コマンド・ファイルでシンボルを定義した場合は，ログインするたびにそのシンボルを使用できますが，シンボルを会話形式で定義した場合は，現在のプロセス中でのみシンボルを使用できます。

DCL コマンドと同じ名前を持つシンボルを定義すると，その定義は DCL コマンド名を無効にします。たとえば，TYPE HELP.LST コマンドとして HELP シンボルを定義すると，HELP を入力しても，システムのヘルプ・ユーティリティを起動できなくなります。

12.2.2 シンボルの短縮

アスタリスク(*)を使用してシンボルを作成すると，シンボルの短縮形を使用できます。一般に，シンボル定義の短縮形は，シンボルを使用できる状況であればいつでも使用できます。ただし，部分文字列の置換を含むシンボルは例外です。詳細は，第 12.6.5 項を参照してください。

シンボルを定義する際には，既存のシンボルが新しいシンボルと置き換えられてしまうことがあります。既存のシンボルがアスタリスクの位置，またはその後まで新しいシンボルと完全に一致する場合には，既存のシンボルは新しいシンボルに置き換えられます。また，アスタリスクの位置まで，またはその後まで既存のシンボルと部分的に一致する名前を持つシンボルを新たに定義することはできません。

次の例は，PR，PRI，または PRIN に短縮できるローカル・シンボル PRINT を作成します。

```
$ PR*INT = "PRINT/CONFIRM/QUEUE=AKI$LN03/NOTIFY/RESTART"
```

指定された修飾子を付けて DCL の PRINT コマンドを実行するには，シンボルのフルネームかその短縮形を使用します。

12.2.3 フォーリン・コマンドの定義

DCL 以外のイメージのファイル指定をシンボルに定義すると，そのシンボル名でイメージを実行できます。イメージを実行するシンボルをフォーリン・コマンドといいます。フォーリン・コマンドは，コマンド・インタプリタが DCL コマンドとして認識しないイメージです。(DCL コマンドのどの要素とも同様に，フォーリン・コマンドの文字数の上限は 255 文字です。)

シンボルをフォーリン・コマンドとして定義するときの形式は，次のとおりです。

シンボル名:[=] \$イメージ・ファイル指定
シンボル名:[=] "\$イメージ・ファイル指定"

シンボル定義冒頭のファイル指定の前のドル記号(\$)は(ドル記号とファイル指定の間にはスペースはない)，イメージの実行要求を意味しています。

イメージ・ファイル指定の場合，省略時のデバイスとディレクトリ名は SYSSYSTEM に，省略時のファイル・タイプは EXE になります。省略時のファイル・バージョン番号は最も大きいバージョンになります。

フォーリン・コマンドを使用すると，Command Definition ユーティリティによって新しいコマンドを定義することもできます。詳細は，『OpenVMS Command Definition, Librarian, and Message Utilities Manual』を参照してください。

シンボルを指定せずに，フォーリン・コマンドを自動的に実行する方法もあります。詳細については，第 12.14 節を参照してください。

次の例では，グローバル・シンボル PRINTALL は，DISK1:[ACCOUNTS]PRINTALL.EXE イメージを実行するように定義されています。

```
$ PRINTALL ::= $[ACCOUNTS]PRINTALL
```

コマンド行で，PRINTALL の後にパラメータが付くことがあります。

次の例では，ファイル指定 RAT.DAT は，PRINTALL で定義されたイメージに渡されるパラメータです。

```
$ PRINTALL RAT.DAT
```

12.2.4 シンボルの置換

コマンド・インタプリタは、一重引用符(')で囲まれたシンボルを検索して、それを変換します。したがって、前に一重引用符が付いたシンボルまたはレキシカル関数を使用してパラメータを指定した場合は、シンボル置換が行われます (第 12.12 節を参照)。それ以外の場合には、コマンド・インタプリタはコマンド行を解析しません。このため、パラメータを受け取るイメージ側で、コマンド行の解析または評価を行わなければなりません。

12.2.5 シンボルの削除

DELETE/SYMBOL コマンドは、シンボルを削除します。グローバル・シンボルを削除するには、/GLOBAL 修飾子を指定します。たとえば、グローバル・シンボル TEMP を削除するには、次のコマンドを入力します。

```
$ DELETE/SYMBOL/GLOBAL TEMP
```

12.3 シンボルの表示

SHOW SYMBOL コマンドは、シンボルの値を表示します。個々のシンボルの値を表示するには、SHOW SYMBOL コマンドの後にシンボルの名前を入力します。個々のグローバル・シンボルの値を表示するには、/GLOBAL 修飾子を指定します。SHOW SYMBOL/ALL コマンドはすべてのローカル・シンボルを、SHOW SYMBOL/ALL/GLOBAL コマンドはすべてのグローバル・シンボルを表示します。

シンボルが整数値を持つ場合、SHOW SYMBOL コマンドは値を 10 進数、16 進数、8 進数で表示することに注意してください。

次の例では、シンボル PR が表示されます。

```
$ SHOW SYMBOL PR
PR*INT = "PRINT/CONFIRM/COPIES=2/QUEUE=DOC$LN03/NOTIFY/RESTART"
```

次の例では、シンボル TOTAL に対して整数値が表示されます。

```
$ SHOW SYMBOL TOTAL
TOTAL = 4   Hex = 00000004   Octal = 00000000004
```

12.4 別のシンボルの値としてのシンボルの使用

シンボルを定義しておくと、それを別のシンボルの値として使用することができます。DCL は、シンボルを使用するコンテキストに応じて、シンボルを文字列または数値として解釈します。

次の例では，整数値 3 をシンボル COUNT に割り当てています。

```
$ COUNT = 3
```

この場合，COUNT の値を別の割り当て文で使用できます。たとえば，COUNT の値が 1 に加算されます。

```
$ TOTAL = COUNT + 1
```

結果の (4) が TOTAL シンボルに定義されます。

12.4.1 シンボルの連結

いくつかのシンボルを連結して長い文字列を作成するには，プラス記号(+)を使用できます。それぞれのシンボル名の前後に一重引用符(')を置けば，2 つ以上のシンボルを連結することもできます。

シンボル置換の要求については，第 12.12.2 項を参照してください。

次の例では，シンボル "Saturday" と "Sunday" を使用して，シンボル "WEEKEND" が作成されます。

```
$ DAY1 = "Saturday, "  
$ DAY2 = "Sunday"  
$ WEEKEND = DAY1 + DAY2  
$ SHOW SYMBOL WEEKEND  
    WEEKEND = "Saturday, Sunday"
```

次の例では，一重引用符を使用して，シンボル NAME と TYPE が連結されます。

```
$ NAME = "MYFILE"  
$ TYPE = ".DAT"  
$ PRINT 'NAME' 'TYPE'
```

PRINT コマンドは，MYFILE.DAT の内容を印刷します。

12.4.2 文字列割り当ての中に含まれるシンボル

文字列割り当ての中にローカル・シンボルを指定するには，コロンと等号 (:=) を使用します。文字列割り当ての中にグローバル・シンボルを指定するには，コロンと 2 つの等号 (:=) を使用します。どちらのタイプのシンボル (ローカルまたはグローバル) の場合も，シンボルを一重引用符(')で囲みます。そうしないと，DCL はそれをシンボルとは認識しません。

シンボルに空の文字列を定義すると，次の例に示すように，そのシンボルは 0 の値を持つようになります。

次の例は，文字列割り当て文の中に COUNT シンボルを指定しています。

```
$ BARK := P'COUNT'
```

前の例では，COUNT に整数値 3 を割り当てています。この例では，COUNT は文字列値に変換され，文字 P に追加されます。このとき，ローカル・シンボル BARK は P3 と同じ値を持っています。

次の例では，シンボル A はヌルです。

```
$ A = ""
$ B = 2
$ C = A + B
$ SHOW SYMBOL C
    C = 2  Hex = 00000002  Octal = 00000000002
```

12.5 シンボルによるデータの格納と操作

シンボルは，コマンド・プロシージャの中の変数として使用できます。変数には，リテラル値以外のものとして計算したり割り当てたりする値が入ります。たとえば，レキシカル関数の値を変数に割り当てたり，ファイル・レコードの値を変数に読み込んだりすることができます。

式は，値が組み合わされたものです。コマンド・プロシージャの中では，式は，シンボル割り当て文(等号の右辺)，IF 文，WRITE コマンドの中で，あるいはレキシカル関数の引数として使用されます。

シンボル定義では，割り当て文の左辺がシンボル名，右辺が式になります。式の中のそれぞれの値(オペランドとも呼びます)は，演算子によって別の値に接続できます。DCL は式を評価して，その結果をシンボルに割り当てます。式が文字列として評価された場合は，シンボルは文字列値を持ちます。

次の例では，ローカル・シンボル BARK が，3 つの数値を加算する式として定義されています。

```
$ BARK = 1 + 2 + 3
```

オペランドは 1，2，および 3 です。演算子はプラス記号(+)です。前の例では，評価された式が整数なので，シンボルは整数値を持ちます。

12.6 文字列

文字列には，印刷可能な文字が入ります。付録 A に，文字列に指定できる ASCII 文字セットと DEC Multinational 文字セットの表があります。これらの表は文字列に含むことのできる文字を示しています。

文字は，次の 3 つに分類されます。

- 英数字

A ~ Z の大文字，a ~ z の小文字，1 ~ 9 の数字，ドル記号(\$)，アンダスコア(_)，ハイフン(-)。

- 特殊文字

表示や印刷が可能な上記以外の文字。感嘆符(!)，引用符(")，番号記号(#)など。

- 印刷不可能な文字

表示や印刷ができないすべての文字。一般に，印刷不可能な文字は表示や印刷のときには無視されますが，いくつかの印刷不可能な文字は，次のような制御機能を持ちます。

文字	機能
HT	次の水平タブで印刷や入力を開始する
LF	次の行で印刷や入力を開始する
FF	次のページの冒頭で印刷や入力を開始する
CR	同一行の最初のスペースで印刷や入力を開始する
ESC	ターミナル・エスケープ・シーケンスの開始を示す
SP	スペースを 1 つ挿入する

12.6.1 文字列の定義

文字列を引用符(" ")で囲めば，文字列を定義できます。シンボル割り当てを行ったときの英字の大文字/小文字とスペースは，この方法で残します。次の点に注意してください。

- 文字列の中に引用符(")を指定するには，引用符を 2 つ連続して入力する。
- 文字列が 2 行にわたる場合には，プラス記号 (文字列の連結) とハイフン (継続) を使用する。

引用符で囲まれた文字列の中では，ハイフン継続文字は使用できない。

次の例では，文字列 "YES" が引用符で囲まれます。したがって，引用符の内部で定義しなければなりません。

```
$ PROMPT = "Type "YES" or "NO" "  
$ SHOW SYMBOL PROMPT  
PROMPT = "Type YES or NO "
```

次の例では，文字列が 2 行に継続されます。

```
$ HEAD = "MONTHLY REPORT FOR" + -  
_ $ " DECEMBER 1999"  
$ SHOW SYMBOL HEAD  
HEAD = "MONTHLY REPORT FOR DECEMBER 1999"
```

12.6.2 文字列式

文字列式には，文字列，文字列として評価されるレキシカル関数，文字列値を持つシンボルが入ります。式の中で文字列を使用する場合には，文字列を引用符(" ")で囲まなければなりません。引用符を使用しないと，文字列はシンボルとして処理されます。

文字列式は，次の値（「文字列オペランド」と呼びます）を組み合わせます。

- 引用符で囲まれた文字列
- 文字列を表すシンボル
- 文字列として評価されるレキシカル関数

文字列と数値との間で演算や比較を実行すると，DCL はその文字列を数値に変換します。

文字列オペランドは，後述するように，加算（文字列連結），減算（文字列削減），比較，他の文字列との置換を行うことができます。

次の例では，文字列 "CAT" は引用符で囲まなければなりません。

```
$ TEMP = "CAT"
```

次の例では，TEMP シンボルは文字列 "CAT" を表しています。TOPIC シンボルは，文字列 "THE" と TEMP シンボルが表す文字列 ("CAT") を連結します。結果は "THE CAT" となります。

```
$ TOPIC = "THE" + TEMP
```

次の例では，シンボル COUNT はレキシカル関数 F\$STRING(65) を表します。

```
$ COUNT = F$STRING(65)
```

12.6.3 文字列の操作

次のような文字列操作を指定できます。

- 連結

プラス記号は，2 つの文字列を連結します。

- 削減

マイナス記号は，最初の文字列から 2 番目の文字列を削除します。

2 番目の文字列が最初の文字列の中で 2 回以上生じている場合には，最初の文字列だけが削除されます。

次の例では，プラス記号 (+) を使用して，2 つの文字列を連結します。

```
$ COLOR = "light brown"
$ WEIGHT = "30 lbs."
$ DOG2 = "No tag, " + COLOR + ", " + WEIGHT
$ SHOW SYMBOL DOG2
DOG2 = "No tag, light brown, 30 lbs."
```

次の例では，マイナス記号 (−) を使用して，文字列を削除します。

```
$ SHOW SYMBOL DOG2
DOG2 = "No tag, light brown, 30 lbs."
$ DOG2 = DOG2 - ", 30 lbs."
$ SHOW SYMBOL DOG2
DOG2 = "No tag, light brown"
```

12.6.4 文字列の比較

2 つの文字列を比較する場合，文字列は文字ごとに比較されます。長さの異なる文字列は等しくありません (たとえば，"dogs" は "dog" より大きいとみなされます)。

比較の際の基準となるのは，文字の ASCII 値です。この基準のもとでは，0 ~ 9 の数字は A ~ Z の英字より小さく，A ~ Z の大文字は a ~ z の小文字より小さくなります。次のいずれかの条件が真になると，文字列比較は終了します。

- すべての文字の比較が終了した。この場合，文字はすべて等しいとみなされる。
- 一致しない最初の文字を検出した。

表 12-1 に様々な文字列比較のタイプを示します。

表 12-1 文字列比較

比較	演算子	説明
等しい	.EQS.	文字列と文字列を比較して，等しいかどうかを判定する。
より大きい， または等しい	.GES.	文字列と文字列を比較して，最初に指定された文字列の値が 2 番目の文字列の値より大きいかまたは等しいかを判定する。
より大きい	.GTS.	文字列と文字列を比較して，最初に指定された文字列の値が 2 番目の文字列の値より大きいかどうかを判定する。
より小さい， または等しい	.LES.	文字列と文字列を比較して，最初に指定された文字列の値が 2 番目の文字列の値より小さいかまたは等しいかを判定する。
より小さい	.LTS.	文字列と文字列を比較して，最初に指定された文字列の値が 2 番目の文字列の値より小さいかどうかを判定する。
等しくない	.NES.	1 つの文字列を別の文字列と比較して，等しくないかどうかを判定する。

これらの例の中では，LAST_NAME シンボルが "WHITFIELD" の値を持つものとします。

- 次の例では，LAST_NAME シンボルの値は "HILL" というリテラルに等しくないため，0 (偽) が戻されます。

```
$ TEST_NAME = LAST_NAME .EQS. "Hill"  
$ SHOW SYMBOL TEST_NAME  
TEST_NAME = 0    . . .
```

- 次の例では，LAST_NAME シンボルの値が "HILL" というリテラルより大きい
か，または等しいため，1 (真) が戻されます。

```
$ TEST_NAME = LAST_NAME .GES. "HILL"  
$ SHOW SYMBOL TEST_NAME  
TEST_NAME = 1    . . .
```

- 次の例では，LAST_NAME シンボルの値が "HILL" というリテラルより大きい
ため，1 (真) が戻されます。

```
$ TEST_NAME = LAST_NAME .GTS. "HILL"  
$ SHOW SYMBOL TEST_NAME  
TEST_NAME = 1    . . .
```

- 次の例では，LAST_NAME シンボルの値が "HILL" というリテラルより大きい
ため，0 (偽) が戻される。

```
$ TEST_NAME = LAST_NAME .LES. "HILL"  
$ SHOW SYMBOL TEST_NAME  
TEST_NAME = 0    . . .
```

- 次の例では，LAST_NAME シンボルの値が "HILL" というリテラルより小さくな
いため，0 (偽) が戻される。

```
$ TEST_NAME = LAST_NAME .LTS. "HILL"  
$ SHOW SYMBOL TEST_NAME  
TEST_NAME = 0    . . .
```

- 次の例では，LAST_NAME シンボルの値が "HILL" というリテラルに等しくない
ため，1 (真) が戻される。

```
$ TEST_NAME = LAST_NAME .NES. "HILL"  
$ SHOW SYMBOL TEST_NAME  
TEST_NAME = 1    . . .
```

12.6.5 部分文字列の置換

置換文字列の位置とサイズを指定すると，文字列の一部だけを別の文字列に置き換える
ことができます。この場合の形式は次のとおりです。

ローカル・シンボルの場合は，
シンボル名[オフセット, サイズ] := 置換文字列

グローバル・シンボルの場合は，
シンボル名[オフセット, サイズ] := 置換文字列

各要素は次のとおりです。

オフセット	元の文字列の最初の文字を基準として置換文字列の位置を示す整数である。オフセット 0 はシンボルの最初の文字を意味し，オフセット 1 は 2 番目の文字を意味する。
サイズ	置換文字列の長さを示す整数である。

部分文字列を置換する場合には，次の規則に従います。

- 大括弧[]は必ず指定しなければならない。シンボル名と左大括弧の間にはスペースはいれない。
- サイズとオフセットには，0 ~ 768 の範囲の整数値を指定する。
- 置換文字列は文字列でなければならない。
- 指定するシンボル名は，最初は未定義でもかまわない。割り当て文でシンボル名を作成し，必要であれば，シンボル値の前または後にスペースを挿入する。
- オフセットとサイズを指定すれば，ブランク行を表すシンボルを作成することもできる。

レコードを一行に並べれば，リストが読みやすくなり，ソートしやすくなります。この形式を使用して，データの格納方法を指定できます。

次の例では，最初の割り当て文は A というシンボルに PACKRAT の値を割り当てています。2 番目の割り当て文は，MUSK を A の値の中の最初の 4 文字に置き換えることを指定しています。この結果，A の値は MUSKRAT になります。

```
$ A := PACKRAT
$ A[0,4] := MUSK
$ SHOW SYMBOL A
  A = "MUSKRAT"
```

次の例では，シンボル B が以前に値を持っていない場合には，4 つのスペースの後に RAT を付けた値が与えられます。

```
$ B[4,3] := RAT
```

次の例では，80 個のブランクからなる値を LINE シンボルに割り当てています。

```
$ LINE[0,80] := " "
```

次の例では，最初の文は，DATA の最初の 15 桁に NAME の値を埋め込みます。2 番目の文は，18 桁目に GRADE の値を埋め込みます。16 桁目と 17 桁目はブランクになります。

```
$ DATA[0,15] := 'NAME'
$ DATA[17,1] := 'GRADE'
```

12.7 数値と式の使用方法

数値は次のいずれかの値になります。

- 10 進数 - 0 ~ 9 の ASCII 文字
- 16 進数 - 0 ~ 9 と A ~ F の ASCII 文字
- 8 進数 - 0 ~ 7 の ASCII 文字

シンボルに割り当てる数値は，-2147483648 ~ 2147483647 (10 進数) の範囲でなければなりません。この範囲にない数値を指定した場合や，この範囲にない数値が計算で得られた場合には，エラーは報告されませんが，結果は間違った値になります。

12.7.1 数値の指定

DCL コマンド・レベルとコマンド・プロシージャの中では，数値は次のように指定します。

- 正の整数

正の整数は，該当する数字を入力することによって指定します。

- 負の整数

負の整数には前にマイナス記号(-)を付けます。

- 基数

10 進数以外の基数を指定するには，16 進数の場合には%X を，8 進数の場合には%0 を数値の前 (ただし，マイナス記号の後) に付けます。

- 小数

数値には小数点を含めることができません。計算のときには，小数点は切り捨てられます。たとえば，8 を 3 で除算すると 2 になります。

次の例では，13 という数値を DOG_COUNT というシンボルに割り当てています。

```
$ DOG_COUNT = 13
$ SHOW SYMBOL DOG_COUNT
DOG_COUNT = 13   Hex = 0000000D   Octal = 00000000015
```

次の例では，負の値 (-15237) がマイナス記号(-)によって表現されます。

```
$ BALANCE = -15237
$ SHOW SYMBOL BALANCE
BALANCE = -15237   Hex = FFFFC47B   Octal = 37777742173
```

次の例では，16 進数 D が%X という接頭語によって表現されます。


```
$ 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
```

12.7.2 数値の内部記憶

数値は，内部的には符号付きの 4 バイト整数，すなわちロングワードとして格納されます。正の整数は 0 ~ 2147483647 の値に，負の整数は 4294967296 から数値の絶対値を引いた値になります。たとえば，-15237 という数値は 4294952059 として格納されます。負の整数は，ASCII や 10 進数で表示する場合には，マイナス記号形式に変換されますが，16 進数や 8 進数で表示する場合には，マイナス記号形式には変換されません。たとえば，-15237 という数値は，16 進数の -00003B85 ではなく，16 進数の FFFFC47B (10 進数 4294952059) として表示されます。

数値は，ASCII 表記規則を使用して一連の数字としてテキスト・ファイルに格納されます (たとえば，数値の 1 は 49 として格納されます)。

数値式では，値はリテラル数 (3 など) または数値を持つシンボルでなければなりません。また，数値を表す文字列 (たとえば，"23"または"-51") を使用できます。数値と文字列の間で演算を行ったり比較したりする場合，文字列は数値に変換されます。

数値式は，次の値 (整数オペランドと呼ばれます) を組み合わせます。

- 整数。たとえば，

```
$ COUNT = 1
```

- 整数として評価されるレキシカル関数。たとえば，

```
$ B = F$INTEGER("-9" + 23)
```

- 整数値を持つシンボル。たとえば，

```
$ A = B - 6
```

前の例では，B というシンボルは F\$INTEGER 関数 (-923) によって戻された整数値を表します。

このような整数オペランドは，後述するように，算術演算子，論理演算子，比較演算子によって接続できます。

12.7.3 算術演算

次のような算術演算を指定できます。

- 乗算

アスタリスク(*)は，2つの数値を乗算します。

- 除算

スラッシュ(/)は，最初に指定された数値を2番目に指定された数値で除算します。数値が割り切れない場合には，余りは切り捨てられ，四捨五入は行われません。

- 加算

プラス記号(+)は，2つの数値を加算します。

- 減算

マイナス記号(-)は，最初に指定された数値から2番目に指定された数値を減算します。

- 単項プラスと単項マイナス

プラス記号とマイナス記号は，後に続く数値の符号を変更します。

例

- 次の例では，シンボルを割り当てるときに，乗算を使用する方法を示しています。

```
$ BALANCE = 142 * 14
$ SHOW SYMBOL BALANCE
BALANCE = 1988   Hex = 000007C4   Octal = 00000003704
```

- 次の例では，シンボルを割り当てるときに，除算を使用する方法を示しています。

```
$ BALANCE = BALANCE / 14
$ SHOW SYMBOL BALANCE
BALANCE = 142   Hex = 0000008E   Octal = 00000000216
```

- 次の例では，シンボルを割り当てるときに，加算を使用する方法を示しています。

```
$ BALANCE = BALANCE + 37
$ SHOW SYMBOL BALANCE
BALANCE = 179   Hex = 000000B3   Octal = 00000000263
```

- 次の例では，シンボルを割り当てるときに，減算を使用する方法を示しています。

```
$ BALANCE = BALANCE - 15416
$ SHOW SYMBOL BALANCE
BALANCE = -15237   Hex = FFFFC47B   Octal = 00000142173
```

- 次の例では，単項マイナスを使用して，数値-142の符号を変更する方法を示しています。

```
$ BALANCE = -(- a142)
$ SHOW SYMBOL BALANCE
BALANCE = 142    Hex = 0000008E    Octal = 00000000216
```

12.7.4 数値の比較

表 12-2 は，数値比較のタイプの一覧です。

表 12-2 数値比較

比較	演算子	説明
等しい	.EQ.	数値と数値を比較して，等しいかどうかを判定する。
より大きい，または等しい	.GE.	数値と数値を比較して，最初の数値が 2 番目の数値より大きい，または等しいかどうかを判定する。
より大きい	.GT.	数値と数値を比較して，最初の数値が 2 番目の数値より大きいかどうかを判定する。
より小さい，または等しい	.LE.	数値と数値を比較して，最初の数値が 2 番目の数値より小さい，または等しいかを判定する。
より小さい	.LT.	数値と数値を比較して，最初の数値が 2 番目の数値より小さいかどうかを判定する。
等しくない	.NE.	数値と数値を比較して，等しくないかどうかを判定する。

ここに示す例では，BALANCE シンボルが -15237 の値を持つと想定しています。

- 次の例では，BALANCE は -15237 に等しいため，1 (真) が戻されます。

```
$ TEST_BALANCE = BALANCE .EQ. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 1    . . .
```

- 次の例では，BALANCE は -15237 より大きい，または等しいため，1 (真) が戻されます。

```
$ TEST_BALANCE = BALANCE .GE. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 1    . . .
```

- 次の例では，BALANCE は -15237 より大きくないので，0 (偽) が戻されます。

```
$ TEST_BALANCE = BALANCE .GT. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 0    . . .
```

- 次の例では，BALANCE は -15237 より小さい，または等しいので，1 (真) が戻されます。

```
$ TEST_BALANCE = BALANCE .LE. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 1    . . .
```

- 次の例では，BALANCE は -15237 より小さくないので，0 (偽) が戻されます。

```
$ TEST_BALANCE = BALANCE .LT. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 0    . . .
```

- 次の例では，BALANCE は -15237 に等しいので，0 (偽) が戻されます。

```
$ TEST_BALANCE = BALANCE .NE. -15237
$ SHOW SYMBOL TEST_BALANCE
TEST_BALANCE = 0    . . .
```

12.7.5 数値オーバーレイの実行

特殊な形式の割り当て文を使用すれば，現在のシンボル値のバイナリ・(ビット・レベル) オーバーレイを実行できます。

ローカル・シンボルの場合は，
シンボル名[ビット位置, サイズ]=置換式

グローバル・シンボルの場合は，
シンボル名[ビット位置, サイズ]==置換式

各要素は次のとおりです。

ビット位置	オーバーレイを実行するビット 0 からの相対位置を示す整数である。
サイズ	オーバーレイするビット数を示す整数である。

数値オーバーレイを使用するには，次の規則に従います。

- 大括弧([])は必ず指定しなければならない。シンボル名と左大括弧の間にはスペースをいれない。
- リテラル値は 10 進数とみなされる。
- サイズの最大長は 32 ビット。
- 置換式は数値式でなければならない。
- シンボル名が未定義であるか，文字列として定義されている場合には，オーバーレイの結果は文字列になる。それ以外の場合には，オーバーレイの結果は整数になる。

次の例は，BELL シンボルを 7 の値に定義しています。BELL の下位バイトは 00000111 というバイナリ値です。ここで，このバイナリ値のオフセット 5 の位置の 0 を 1 に変更すれば(ビットは，右から左に 0 から順にカウントされます)，00100111 (10 進数 39) というバイナリ値が得られます。

```
$ BELL = 7  
$ BELL[5,1] = 1  
$ SHOW SYMBOL BELL  
BELL = 39   Hex = 00000027   Octal = 00000000047
```

12.8 論理値と式の使用

これ以降の節では，論理値と式の使用について説明します。

12.8.1 論理演算

一部の演算では，数値と文字列を次のような値を持つ論理データとして解釈します。

- 真

数値が奇数の場合 (すなわち，下位バイトが 1 の場合) には，その数値の論理値は真となります。文字列の場合は，最初の文字が大文字または小文字の T または Y のときに，その文字列の論理値は真となります。

- 偽

数値が偶数の場合 (すなわち，下位バイトが 0 の場合) には，その数値の論理値は偽となります。文字列の場合は，最初の文字が大文字または小文字の T または Y でないときに，その文字列の論理値は偽となります。

次の例では，DOG_COUNT に 13 という値が割り当てられています。IF STATUS は，論理値 STATUS が真であることを意味します。

```
$ STATUS = 1  
$ IF STATUS THEN DOG_COUNT = 13  
  
$ STATUS = "TRUE"  
$ IF STATUS THEN DOG_COUNT = 13
```

12.8.2 論理式

論理演算は，演算の対象となる数値のすべてのビットに影響します。論理式の値は整数であり，式の結果も整数になります。論理式で文字列値を指定すると，文字列が整数に変換されてから，式の評価が行われます。

通常は，論理式を使用して，論理値の下位ビットを評価します。すなわち，値が真か偽かを判断します。次の論理演算を指定できます。

- 論理否定 (.NOT.)

.NOT. 演算子は，論理値のビット構成を反転します。真の値は偽となり，偽の値は真となります。

- 論理積 (.AND.)

.AND. 演算子は, 次を示すように, 2 つの論理値の論理積を求めます。

ビット・レベル	要素レベル
1 .AND. 1 = 1	真.AND. 真 = 真
1 .AND. 0 = 0	真.AND. 偽 = 偽
0 .AND. 1 = 0	偽.AND. 真 = 偽
0 .AND. 0 = 0	偽.AND. 偽 = 偽

- 論理和 (.OR.)

.OR. 演算子は, 次を示すように, 2 つの論理値の論理和を求めます。

ビット・レベル	要素レベル
1 .OR. 1 = 1	真.OR. 真 = 真
1 .OR. 0 = 1	真.OR. 偽 = 真
0 .OR. 1 = 1	偽.OR. 真 = 真
0 .OR. 0 = 0	偽.OR. 偽 = 偽

次の例は, 真の値を偽に反転させます。式の評価は -2 で, この値は偶数であるため, 結果は偽となります。

```
$ SHOW SYMBOL STATUS
STATUS = 1   Hex = 00000001   Octal = 00000000001
$ STATUS = .NOT. STATUS
$ SHOW SYMBOL STATUS
STATUS = -2   Hex = FFFFFFFE   Octal = 37777777776
```

次の例では, 真の値と偽の値の論理積を求めた結果が偽の値になっています。

```
$ STAT1 = "TRUE"
$ STAT2 = "FALSE"
$ STATUS = STAT1 .AND. STAT2
$ SHOW SYMBOL STATUS
STATUS = 0   Hex = 00000000   Octal = 00000000000
```

次の例は, 真の値と偽の値の論理和を求めることによって, 結果は真の値になっています。

```
$ STAT1 = "TRUE"
$ STAT2 = "FALSE"
$ STATUS = STAT1 .OR. STAT2
$ SHOW SYMBOL STATUS
STATUS = 1   Hex = 00000001   Octal = 00000000001
```

12.8.3 論理演算の結果

次の表は，ビット単位と数値単位の論理演算の結果を示しています。論理演算では，大文字または小文字の T または Y で始まる文字列は数値 1 として処理され，これ以外の文字で始まる文字列は，数値 0 として処理されます。また，奇数は真になり，偶数とゼロは偽になります。

指定された:		結果:		
ビット A	ビット 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

指定された:		結果:		
数値 A	数値 B	.NOT. A	A .AND. B	A .OR. B
奇数	奇数	偶数	奇数	奇数
奇数	偶数	偶数	偶数	奇数
偶数	奇数	奇数	偶数	奇数
偶数	偶数	奇数	偶数	偶数

12.8.4 レキシカル関数が戻す値の使用

レキシカル関数は，通常コマンド・プロシージャの中で使用され，システム・プロセス，バッチ・キューと印刷キュー，ユーザ・プロセスなどについての情報をシステムから検索します。レキシカル関数を使用して，文字列を処理したり論理名を変換することもできます。レキシカル関数をシンボルに割り当てた場合，実際にはレキシカル関数によって戻される情報（たとえば，数値や文字列）がシンボルに割り当てられます。このとき，DCL レベルでは，DCL の SHOW SYMBOL コマンドでその情報を表示できます。コマンド・プロシージャの中では，シンボルに格納された情報を後でプロシージャで使用できます。それぞれのレキシカル関数については，『OpenVMS DCL ディクショナリ』を参照してください。

レキシカル関数を使用するには，レキシカル関数の名前（先頭に F\$が付く）とその引数リストを指定します。次の形式を使用します。

F\$関数名 (引数[,...])

引数リストは関数名の後に指定し，その間にスペースやタブをいくつでも入れることができます。

レキシカル関数を使用する場合は，次の規則に従います。

- 引数リストは括弧で囲む。

- 引数リストの中では，引数を正確な順序で指定し，それぞれの引数はコンマで区切る。オプションの引数を省略しても，コンマは省略しない。
- 引数が必要ない場合は，括弧だけを入力する。
- 式を作成するときの規則に従う。すなわち，文字列は引用符で囲み，整数，シンボル，レキシカル関数は引用符で囲まない。

レキシカル関数は，文字列，整数，シンボルと同じように使用します。式の中でレキシカル関数を使用すると，DCL は自動的に関数を評価して，関数を戻り値と置き換えます。

次の例では，F\$LENGTH 関数は，引数 BUMBLEBEE として指定された値の長さを戻します。DCL は，自動的に戻り値 (9) を判別し，この値を使用して式を評価します。したがって，式 (9 + 1) の結果は 10 となり，この値が SUM シンボルに割り当てられます。

```
$ SUM = F$LENGTH("BUMBLEBEE") + 1
$ SHOW SYMBOL SUM
SUM = 10   Hex = 0000000A   Octal = 00000000012
```

各レキシカル関数は，情報を整数または文字列のいずれかとして戻すことに注意してください。また，レキシカル関数の引数は，整数または文字列式として指定しなければなりません。

たとえば，F\$LENGTH 関数は，文字列式である引数を必要とし，整数の値を戻します。前の例で，"BUMBLEBEE" 引数は文字列式であり，戻り値 (9) は整数です。

レキシカル関数は，シンボルを使用できる場所であればどこでも使用できます。シンボルを一重引用符で囲んで (第 12.12 節を参照)，シンボル置換を強制しなければならない場所では，レキシカル関数の前後に一重引用符を置いて，レキシカル関数の評価を強制しなければなりません。レキシカル関数は，他のレキシカル関数の引数値としても使用できます。

F\$LENGTH 関数に対して引数を指定する，他の方法を以下の例に示します。それぞれの例では，引数は文字列式です。

- 次の例は，シンボルと文字列を両方含む引数を示しています。

```
$ BUG = "BUMBLEBEE"
$ LEN = F$LENGTH(BUG)
$ SHOW SYMBOL LEN
LEN = 9   Hex = 00000009   Octal = 00000000011
```

BUG シンボルを引数として使用する場合，前後に引用符は付けません。レキシカル関数は "BUMBLEBEE" 値を自動的に BUG に置換して，長さを判別し，9 の値を戻します。

- 次の例は，シンボルと文字列を両方含む引数を示しています。

```
$ 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
```

BUG シンボルは引用符で囲まれていませんが，文字列の "S" は引用符で囲まれています。引数を評価してからでないと，F\$LENGTH 関数は長さを判別できません。BUG シンボル ("BUMBLEBEE") によって表現される値は "S" 文字列と連結され，結果は "BUMBLEBEES" になっています。F\$LENGTH 関数は，"BUMBLEBEES" 文字列の長さを判別して，10 の値を戻します。

- 次の例は，F\$DIRECTORY 関数の引数として別のレキシカル関数を使用しています。F\$DIRECTORY 関数は，大括弧を含む，現在の省略時のディレクトリの名前を戻します。この例では，現在の省略時のディレクトリは[SALMON]です。

```
$ LEN = F$LENGTH(F$DIRECTORY())
$ SHOW SYMBOL LEN
  LEN = 8   Hex = 00000008   Octal = 00000000010
```

F\$DIRECTORY を引数として使用する場合は，前後に引用符を付けません。関数は自動的に評価されます。F\$DIRECTORY 関数の結果が戻ってからでないと，F\$LENGTH 関数は長さを判別できません。結果が戻ると，F\$LENGTH 関数は省略時のディレクトリの長さ (大括弧を含む) を判別します。

12.8.5 演算の順序

式には演算と比較操作をいくつでも指定できます。次の表は，式の中に 2 つ以上の演算子がある場合の演算子を評価順にリストしたものです。演算子は，優先順位の高い順にリストされています。すなわち，上にある演算子ほど先に実行されます。

優先順位	演算
7	単項プラス(+)と単項マイナス(-)
6	乗算(*)と除算(/)
5	加算(連結)と減算(削減)
4	すべての数値および文字比較
3	論理 NOT 演算
2	論理 AND 演算
1	論理 OR 演算

1 つの式の中に同じ優先順位を持つ複数の演算子がある場合には，左から右に演算が行われます。通常の優先順位 (演算と比較を評価する順) を変更するには，最初に行う演算を括弧で囲みます。括弧をネストすることもできます。

次の例では，括弧があるために，乗算の前に加算が実行されます。括弧がなければ，乗算を先に実行するため，結果は 26 になります。括弧はネスト可能です。

```
$ RESULT = 4 * (6 + 2)
$ SHOW SYMBOL RESULT
RESULT = 32   Hex = 00000020   Octal = 00000000040
```

12.8.6 データ・タイプの評価

DCL のシンボルの評価結果は，文字列または整数値のいずれかになります。シンボルのデータ・タイプ (文字または整数) は，現在割り当てられている値のデータ・タイプによって決まります。データ・タイプは永久的なものではないので，値のタイプが変更されると，シンボルのタイプも変更されます。

式は，値のタイプと使用する演算子に応じて，整数値になったり文字列値になったりします。

次の例では，ローカル・シンボル NUM には，最初に文字値が割り当てられ，次に整数式が割り当てられるときに整数値に変換されます。

```
$ NUM = "ABC"
$ NUM = 2 + 5
```

次の表は，DCL がどのように式を評価するかをまとめたものです。最初の欄には，式に指定できる値と演算子を示します。2 番目の欄は，それぞれの場合に，式全体の評価結果のタイプを示しています。表の中で，任意の値は文字列または整数を表しています。

式	結果の値タイプ
整数値	整数
文字列値	文字列
整数レキシカル関数	整数
文字列レキシカル関数	文字列
整数シンボル	整数
文字列シンボル	文字列
+, -, または .NOT. 任意の値	整数
任意の値.AND. または .OR. 任意の値	整数
文字列 + または - 文字列	文字列
整数 + または - 任意の値	整数
任意の値 + または - 整数	整数
任意の値*または/任意の値	整数
任意の値 (文字列比較) 任意の値	整数
任意の値 (数値比較) 任意の値	整数

12.9 式の値タイプの変換

式の中のオペランドがすべて同じ値のデータ・タイプになっていないと，DCL は式を評価できません。値のタイプには，文字列データ・タイプと整数データ・タイプがあります。文字列データには，文字列，文字列値を持つシンボル，文字列値を戻すレキシカル関数があります。整数データには，整数，整数値を持つシンボル，整数値を戻すレキシカル関数があります。1つの式に数値オペランドと文字列オペランドの両方が入っている場合には，DCL はすべての文字列を整数に変換するか，すべての整数を文字列に変換します。

一般に，文字列値と整数値の両方を使用すると，文字列値が整数に変換されます。ただし，DCL が文字列比較を行うときは例外で，この場合には整数が文字列に変換されます。

また，次のレキシカル関数を使用すると，式の値を決定したり変更したりできます。

- F\$TYPE は，シンボルの現在の値タイプを決定する。
- F\$INTEGER は，文字列式を整数値に変換する。
- F\$STRING は，整数式を文字列値に変換する。

12.9.1 文字列を整数に変換する

文字列は，次のようにして整数に変換されます。

- 数値を含む文字列は，その整数値に変換される。たとえば，文字列 "45" は整数 45 に変換される。

- 文字列が T, t, Y, または y で始まる場合には，整数 1 に変換される。
- 文字列がこれ以外の文字で始まる場合には，整数 0 に変換される。

次の表は，文字列を整数値に変換する場合を示しています。

文字列	結果として生じる整数
"123"	123
"12XY"	0 (偽)
"Test"	1 (真)
"hello"	0 (偽)

12.9.2 整数を文字列に変換

整数を文字列に変換すると，結果として生じる文字列には整数値に対応する数値が入ります。次の表は，整数を文字列値に変換する場合を示しています。

整数	結果として生じる文字列
123	"123"
1	"1"
0	"0"

12.10 シンボル・テーブル

シンボルは，オペレーティング・システムによって管理されるローカル・シンボル・テーブルまたはグローバル・シンボル・テーブルに格納されます。

12.10.1 ローカル・シンボル・テーブル

DCL では，コマンド・プロシージャの実行時，CALL コマンドの使用時，およびバッチ・ジョブのキュー登録時にユーザが作成するすべてのコマンド・レベルとメイン・プロセスに対して 1 つずつのローカル・シンボル・テーブルが保持されます。ローカル・シンボルを作成すると，DCL は，そのシンボルを現在のコマンド・レベルのローカル・シンボル・テーブルに収めます。コマンド・レベルのローカル・シンボル・テーブルは，そのレベルが続くかぎり存在しますが，該当コマンド・レベルが終了すると，そのローカル・シンボル・テーブル（およびその中のすべてのシンボル）も削除されます。プロセス，コマンド・プロシージャ，バッチ・ジョブについての詳細は，第 16 章を参照してください。

ローカル・シンボル・テーブルには，ユーザが定義したローカル・シンボルに加えて，DCL の 8 つのシンボルが収められます。P1 ~ P8 までのこれらのシンボルは，パラメータをコマンド・プロシージャに渡すときに使用されます。コマンド・プロシ

ージャに渡されるパラメータは文字列とみなされます。それ以外の場合，P1 ~ P8 は空の文字列 ("") として定義され，ローカル・シンボル・テーブルに格納されます。

12.10.2 グローバル・シンボル・テーブル

DCL では，1 つのプロセスが存在する間，1 つのグローバル・シンボル・テーブルだけが保持され，すべてのグローバル・シンボルがそのテーブルに収められます。グローバル・シンボル・テーブルには，ユーザがアクセスするグローバル・シンボルに加えて，後述する予備グローバル・シンボルが収められています。このようなグローバル・シンボルは，使用しているプログラムやコマンド・プロシージャ，ならびにシステム・コマンドやユーティリティについての状態情報を提供します。

\$STATUS 予備グローバル・シンボル

\$STATUS は，最も最近実行されたコマンドが戻す条件コードです。シンボル \$STATUS は OpenVMS オペレーティング・システムのメッセージ・コードの形式に準拠します。EXIT コマンドにパラメータ値を指定すれば，アプリケーション・プログラムでグローバル・シンボル \$STATUS の値を設定できます。システムは，\$STATUS の値を使用して，どのようなメッセージを表示するか，次に高いコマンド・レベルで実行を継続するかどうかなどを決定します。\$STATUS の下位 3 ビットの値は，グローバル・シンボル \$SEVERITY に収められます。

\$SEVERITY 予備グローバル・シンボル

\$SEVERITY は，最も最近実行されたコマンドが戻す条件コードの重大度レベルです。\$STATUS の下位 3 ビットに等しいシンボル \$SEVERITY は，次のような値を持ちます。

0	警告
1	成功
2	エラー
3	情報
4	重大 (回復不能な) エラー

\$RESTART 予備グローバル・シンボル

\$RESTART は，システム・クラッシュによる割り込みが生じた後でバッチ・ジョブが再開された場合に TRUE の値になります。それ以外の場合は，FALSE の値になります。

12.10.3 シンボル・テーブル検索順序

コマンド・インタプリタはシンボルの値を判別するときに，シンボル・テーブルを次のような順序で検索します。

1. 現在のコマンド・レベルのローカル・シンボル・テーブル。
2. 前のコマンド・レベルのローカル・シンボル・テーブル。現在のレベルから後方に検索する。

3. グローバル・シンボル・テーブル。

12.11 シンボルの値のマスク

これ以降の節では，シンボルの値をマスクする方法について説明します。

12.11.1 SET SYMBOL コマンド

省略時の設定では，外側のコマンド・プロシージャ・レベルで定義されるすべてのシンボル(グローバルとローカル)は，内側のプロシージャ・レベルにアクセスできます。ただし，SET SYMBOL コマンドを使用すれば，コマンド・プロシージャの中のローカル・シンボルまたはグローバル・シンボルを他のコマンド・プロシージャで定義されたシンボルと区別できます。SET SYMBOL コマンドは，ローカル・シンボルとグローバル・シンボルの値をマスクします。したがって，コマンド・プロシージャが別のコマンド・プロシージャを実行する場合，2 番目のプロシージャで SET SYMBOL コマンドを指定すれば，両方のプロシージャで同じシンボル名を使用できます。

SET SYMBOL コマンドは，DCL がコマンド行を処理する前に，動詞文字列(コマンド行の最初のトークン)をシンボルとして変換しようとするかどうかを制御します。省略時の設定では，変換を行おうとします。この設定を変更して変換を行わないようにすると，コマンドを起動するときに，コマンド・プロシージャが外側のプロシージャ・レベル環境によって影響されなくなるので便利です。

12.11.2 シンボルの有効範囲

シンボルの有効範囲は，ローカル・シンボルとグローバル・シンボルとは異なります。プロシージャ・レベルを終了して，前のプロシージャに戻ると，ローカル・シンボルとグローバル・シンボルの両方で，前のレベルのシンボル有効範囲コンテキストが復元されます。

現在の汎用シンボルの有効範囲状態を表示するには，レキシカル関数 F\$ENVIRONMENT("SYMBOL_SCOPE") を使用します。現在の動詞の有効範囲状態を表示するには，レキシカル関数 F\$ENVIRONMENT("VERB_SCOPE") を使用します。

ローカル・シンボルの有効範囲

ローカル・シンボルはプロシージャ・レベルによって左右されます。ローカル・シンボルを外側のプロシージャ・レベルで定義すると，内側のどのプロシージャ・レベルでもシンボルを読み込むことができます(ただし，シンボルへの書き込みはできません)。外側のプロシージャ・レベルにローカルなシンボルに値を割り当てると，現在のプロシージャ・レベルで新しいシンボルが作成されます。ただし，外側のプロシージャ・レベルのシンボルは変更されません。

SET SYMBOL/SCOPE=NOLOCAL コマンドを使用すると，外側のプロシージャ・レベルで定義されたすべてのローカル・シンボルが現在のプロシージャ・レベルとその内側のプロシージャ・レベルではアクセスできなくなります。たとえば，プロシージャ・レベル 2 と 4 で SET SYMBOL/SCOPE=NOLOCAL を指定すると，次のようになります。

- プロシージャ・レベル 2 では，レベル 2 のローカル・シンボルのみの読み書きが可能。
- プロシージャ・レベル 3 では，レベル 2 のローカル・シンボルを読み込める（書き込みは不可）。また，レベル 3 のローカル・シンボルの読み書きが可能。
- プロシージャ・レベル 4 では，レベル 4 のローカル・シンボルのみの読み書きが可能。

グローバル・シンボルの有効範囲

グローバル・シンボルはプロシージャ・レベルに左右されません。現在のグローバル・シンボルの有効範囲コンテキストがそれ以降のすべてのプロシージャ・レベルに適用されます。

/SCOPE=NOGLOBAL 修飾子を使用すると，/SCOPE=GLOBAL 修飾子を指定するか，プロシージャを終了して，グローバル・シンボルがアクセス可能であった前のレベルに戻るまで，すべてのグローバル・シンボルはそれ以降のすべてのコマンドにアクセスできません。また，/SCOPE=NOGLOBAL 修飾子を指定すると，/SCOPE=GLOBAL 修飾子指定するまで，新しいグローバル・シンボルを作成できなくなります。

12.12 シンボルの置換

コンテキストによっては，英字で始まる文字列がシンボル名またはレキシカル関数として使用されることがあります。このような場合，DCL はシンボルまたはレキシカル関数をその値と置き換えようとします。シンボルをその現在の値で置き換えることをシンボル置換といいます。これ以外のコンテキストでシンボルまたはレキシカル関数を使用する場合には，置換演算子を使用してシンボル置換を要求しなければなりません。

シンボルの自動評価

シンボルやレキシカル関数が次のように使用された場合，DCL はシンボルやレキシカル関数を自動的に評価します。

- 割り当て (=) 文の右辺
- レキシカル関数の引数の中
- DEPOSIT，EXAMINE，IF，または WRITE コマンドの中
- 文字列の後に等号またはコロンが付かない場合，コマンド行の先頭

- 部分文字列置換または数値オーバーレイを実行している場合の，割り当て文の左辺にある大括弧の中 (第 12.6.5 項を参照)

次の例では，コマンド・インタプリタは，英字で始まる文字列をシンボル名として，数値または基数演算子(%)で始まる文字列をリテラル数値として使用します。

- 次の例では，COUNT が自動的に認識され，シンボルとして評価されます。

```
$ TOTAL = COUNT + 1
```

- この例では，2 行目で，QUERY シンボルが F\$LENGTH 関数と一緒に使用されているため，QUERY は自動的に評価されます。また，F\$LENGTH 関数は割り当て文の右辺にあるため，自動的に評価されます。

```
$ QUERY = "Have we met before?"
$ LEN = F$LENGTH(QUERY) + 5
$ SHOW SYMBOL LEN
LEN = 27   Hex = 0000001B   Octal = 000033
```

- 次の例では，IF コマンドは A と B の両方をシンボル名として使用し，その現在の値を使用します。

```
$ IF A .EQ. B THEN WRITE SYS$OUTPUT "DONE"
```

- この例では，2 行目で，コマンド・インタプリタは，PDEL をその現在の値と自動的に置き換え，その結果得られたコマンドを実行します。

```
$ PDEL = "DELETE SYS$PRINT/ENTRY="
$ PDEL 181
```

- 次の例では，DCL はシンボル BELL を 7 という値として自動的に定義し，代入文の左辺の大括弧で囲まれた値をもとに，新しい値を割り当てます。

```
$ BELL = 7
$ BELL[5,1] = 1
$ SHOW SYMBOL BELL
BELL = 39   Hex = 00000027   Octal = 00000000047
```

12.12.1 シンボル置換の強制

上記の場所がないシンボルの置換を強制するには，次のようにシンボルを一重引用符(')で囲みます。

```
$ TYPE 'B'
```

引用符で囲まれた文字列の中のシンボルの置換を強制するには，そのシンボルの前に 2 つの一重引用符(')を付け，シンボルの後に 1 つの一重引用符(')を付けます。

```
$ T = "TYPE ''B'"
```


コマンド行を処理する場合，DCL は次の順序でシンボルをその値に置換します。

- 強制置換

一重引用符 (二重引用符で囲まれた文字列の場合には，2 つの一重引用符) で区切られたすべての文字列を左から右に置換します。前に 1 つの一重引用符が付いたシンボルは反復変換されますが，前に 2 つの一重引用符が付いたシンボルは反復変換されません。

- 自動置換

コマンド行の中のそれぞれの値を左から右に評価して，値がコマンドの場合は実行し，式の場合には評価します。式の中のシンボルは割り当てられた値と置き換えられますが，この置換は反復されません。

次の例は，DCL がシンボルを置換する順序がどのように影響するかを示しています。シンボル PN，FILE1，および NUM は，次のように定義します。

```
$ PN = "PRINT/NOTIFY"
$ FILE1 = "[BOLIVAR]TEST_CASE.TXT"
$ NUM = 1
```

このシンボル定義の場合，次のコマンドは，[BOLIVAR]TEST_CASE.TXT というファイルを印刷します。

```
$ FILE = "'FILE' 'NUM' '"
$ PN 'FILE'
```

最初のコマンドでは，強制置換によって NUM が 1 になり，FILE' 'NUM' は FILE1 になります。SHOW SYMBOL FILE コマンドを入力すると，FILE="FILE1'" になります。

2 番目のコマンドは 2 つの置換を行います。最初に，'FILE' は 'FILE1' に置換されます。一重引用符 (') で囲まれているため，'FILE1' も置換が必要です。自動置換によって FILE1 は [BOLIVAR]TEST_CASE.TXT になります。このファイル名は次に，PN の値である PRINT/NOTIFY に渡されます。結果は，次のような文字列になります。

```
$ PRINT/NOTIFY [BOLIVAR]TEST_CASE.TXT
```

12.12.2 シンボル置換の演算子

置換演算子を使用すると，DCL が通常はシンボル置換を行わない場所でのシンボル置換を要求できます。DCL は，次の 2 つの置換演算子を受け入れます。

- 一重引用符 (')
- アンパサンド (&)

この 2 つの演算子の相違点は，置換が行われる時間です。前に一重引用符が付くシンボルは DCL コマンド処理の第 1 フェーズで置換され，前にアンパサンドが付くシンボルは第 2 フェーズで置換されます。コマンド処理のフェーズについての詳細は，第 12.13 節を参照してください。

一重引用符(')

一重引用符(')は，最もよく使用する置換演算子です。コマンド・パラメータや修飾子の代わりにシンボルを使用する場合に，一重引用符を使用してシンボル置換を要求します。一重引用符は，文字列割り当て(:=)文の右辺でシンボル置換を要求するのに使用します。

引用符で囲まれた文字列の中でのシンボル置換を要求するには，シンボル名の前に 2 つの一重引用符を置き，シンボル名の後に 1 つの一重引用符を置きます。

一重引用符を使用してシンボル置換を要求した場合は，置換する値の途中では (ハイフン継続文字を使用して) 改行できません。

次の例では，TYPE コマンドは，次にくる文字列がファイル指定を想定するコマンドです。一重引用符は，LIT が評価しなければならないシンボルであることを示しています。一重引用符を省略すると，DCL は LIT.LIS (LIS は，TYPE コマンドの省略時のファイル・タイプ) というファイルを探します。

```
$ LIT = "LIGHT.BILLS"  
$ TYPE 'LIT'
```

次の例では，NAME の値が置換されて，FILE は REPORT.DAT になります。

```
$ NAME := REPORT  
$ FILE := 'NAME'.DAT  
$ SHOW SYMBOL FILE  
  FILE = "REPORT.DAT"
```

次の例では，NAME シンボルの現在の値は FRED です。

```
$ MESSAGE = "Creating file '"NAME'.DAT"
```

この場合，MESSAGE は次の値を持ちます。

```
Creating file FRED.DAT
```

アンパサンド(&)

アンパサンド(&)もコマンド・インタプリタが認識する置換演算子です。多くの場合，一重引用符とアンパサンドは同じ機能を果たします。アンパサンドは，一重引用符と一緒に使用して置換の実行順序に影響を及ぼすときの置換演算子として最も効果的です。

シンボルが未定義の場合のコマンド・インタプリタのアクションは，コマンドのコンテキストによって決まります。詳細は，第 12.13.5 項を参照してください。

最初のコマンドでは，コマンド・インタプリタは，コマンド処理の第1フェーズ(検索)でNAMEシンボルを現在の値と置き換えます。2番目のコマンドは，コマンド処理の第2フェーズ(解析)でNAMEシンボルを現在の値と置き換えます。方法は異なっても，結果は同じになります。

```
$ TYPE 'NAME'  
$ TYPE &NAME
```

次の例では，アンパサンドは，一重引用符と一緒に使用されて換の実行順序に影響を及ぼしています。

```
$ P1 = "FRED.DAT"  
$ COUNT = 1  
$ TYPE &P'COUNT'
```

最初に，コマンド・インタプリタは一重引用符で囲まれたシンボル('COUNT')を評価します。結果は次のようになります。

```
TYPE &P1
```

次に，前にアンパサンドが付いたシンボル(P1)を評価します。結果は次のようになります。

```
TYPE FRED.DAT
```

次の例のように，PとCOUNTの両方と一緒に一重引用符を使用したとします。

```
$ TYPE 'P' 'COUNT'
```

コマンド・インタプリタは，左から右に処理を進めてPを評価しようとします。Pは定義済みのシンボルではないため，DCLはPに空の値を指定します。次に，コマンド・インタプリタはCOUNTシンボルを評価します。結果は次のようになります。

```
TYPE 1
```

次の例では，AはBの現在の値に等しく定義されています。

```
$ B = "MYFILE.DAT"  
$ A = "&B"  
$ TYPE 'A'
```

アンパサンド(&)を引用符(" ")の中で使用してもシンボル置換は行われません。したがって，割り当てを行っても，Bの値は置換されませんが，TYPEコマンドはMYFILE.DATを表示します。これは，コマンド・インタプリタが最初に&Bの値をAに置換するからです。コマンド・インタプリタは，次にMYFILE.DATを&Bシンボルに置換します。Bを再定義すれば，TYPEコマンドの結果はそれに応じて変化します。

アンパサンドを使用するときには，次の規則に従ってください。

- アンパサンドはシンボル名の前に置き，シンボル名の後には置かない。
- アンパサンドは区切り文字 (ブランクまたは特殊文字) の後に置かなければならない。
- アンパサンドを使用して，引用符(" ")で囲まれた文字列の中で置換を要求することはできない。
- アンパサンドを使用して2つ以上のシンボル名を連結することはできない。
- 一般に，シンボルを正しく変換するために必要な場合を除いて，シンボル置換にアンパサンドは使用してはならない。

12.13 コマンド処理の3つのフェーズ

コマンド・インタプリタは，シンボル置換を次の3つのフェーズで実行します。

12.13.1 第1フェーズ: コマンド入力検索

コマンド入力検索では，前に一重引用符が付いたシンボルを左から右に順に評価します。前に1つの一重引用符が付いたシンボルは，第1フェーズ置換で説明するように，反復変換されますが，前に2つの一重引用符が付いたシンボルは反復変換されません。

12.13.2 第2フェーズ: コマンドの解析

コマンド解析フェーズでは，次のことを行います。

- コマンド行を解析する。当該行の最初の項目がシンボルかどうかをチェックして，シンボルの場合には評価する。
- 前にアンパサンドが付いたシンボルを左から右に順に評価する。

このフェーズでのシンボル置換は反復されません。

12.13.3 第3フェーズ: 式の評価

式の評価フェーズでは，次のことが行なわれます。

- 前に DEPOSIT，EXAMINE，IF，および WRITE コマンドが置かれたシンボルを評価する。
- レキシカル関数の中のシンボルを評価する。

このフェーズでのシンボル置換は反復されません。

コマンド・インタプリタは，コマンド・プロシージャの中で実行されるコマンドやプログラムが入力データとして読み込んだ行は解析しないことに注意してください。したがって，このようなデータ行ではシンボル置換を行いません。

AVERAGE プログラムは，SYSS\$INPUT (コマンド入力ストリーム) から 55，57，9999 を読み込みます。これらのデータ行はコマンド・インタプリタによっては読み込まれません。シンボル名を入力しても評価されません。

```
$ RUN AVERAGE
55
57
9999
```

12.13.4 繰り返し置換と反復置換

シンボル置換には，繰り返し置換と反復置換があります。

- 繰り返し置換は，1つのコマンド行で2つ以上のタイプの置換を行います。
- 反復置換は，コマンド・インタプリタが置換された値を調べて，値自体がシンボルかどうかをチェックします。反復置換が行われるのは，前に一重引用符が付いたシンボルがコマンド処理の第1フェーズで変換される場合のみです。

第1フェーズ置換

一重引用符(')を使用してシンボル置換を要求すると，コマンド・インタプリタは，コマンド処理の第1フェーズで反復置換を実行します。

ただし，引用符で囲まれた文字列の中にシンボルを指定した場合には，一重引用符を使用する置換は反復されません。

次の例では，置換は反復されます。

```
$ MAC = "5"
$ A = "'MAC'"
$ B = 'A'
$ SHOW SYMBOL B
B = 5 Hex = 00000005 Octal = 00000000005
```

次の理由から，B = 'A'文の後，Bシンボルの値は5になります。

- シンボル名 A は一重引用符で囲まれているため，その現在の値('MAC')で置き換えられる。
- この値('MAC')も一重引用符で囲まれているため，コマンド・インタプリタはMACを現在の値(5)と置き換える。
- この値(5)には一重引用符が付いていないので，コマンド処理の第1フェーズは終了である。第2フェーズまたは第3フェーズでのこれ以上の置換は必要ないため，5がシンボル名Bに割り当てられる最終値になる。

ただし，A を引用符で囲まれた文字列の中で指定した場合にはどうなるでしょう。

```
$ B = "'A'"
$ SHOW SYMBOL B
B = "'MAC'"
```

この場合は，B は'MAC'の値を持ちます。引用符で囲まれた文字列の中では置換は反復されないため，シンボル名 A は一回だけ置換されます。

第2フェーズ置換

コマンド・インタプリタが反復置換を自動的に実行するのは，コマンド行に一重引用符がある場合だけです。場合によっては，コマンド同意語の定義をネストするとよいでしょう。

次の例では，EXEC が処理されると，コマンド・インタプリタは置換を一回だけ実行します。

```
$ MAC = "TYPE A.B"
$ EXEC = "'MAC'"
$ EXEC
```

結果は文字列'MAC'になります。MAC はコマンドとして認識されないため，エラー・メッセージが出されます。このエラーは，コマンド処理の第1フェーズに置換が実行されない (EXEC 文字列が一重引用符によって区切られていない) ために生じています。第2フェーズでは，EXEC がコマンド行の最初の値であるため，文字列'MAC'がEXEC に置き換えられます。この置換は反復されません。したがって，'MAC'が一重引用符で区切られていても，これ以上置換は行われません。

コマンド同意語 EXEC を正しく使用するには，次に示すように，EXEC を一重引用符で囲みます。

```
$ 'EXEC'
```

この場合，コマンド処理の第1フェーズで EXEC シンボルが評価されます。この置換は反復置換であるため，('MAC') も評価され，TYPE A.B 文字列が置換されます。

第3フェーズ置換

コマンド・インタプリタがコマンドの中の式を解析すると，式の中に指定されたシンボルは一回だけ置換されます。ただし，式の中に一重引用符またはアンパサンドを使用すれば，反復置換を強制できます。このようにして反復置換を強制した場合には，次の点に注意してください。

- コマンド・インタプリタは，コマンド文字列を実行する前に，一重引用符またはアンパサンドによって要求されたすべての置換を実行する。
- シンボル置換を自動的に実行するコマンドは，コマンド処理の第1フェーズと第2フェーズの後でこれを行う。

置換の結果，有効なシンボル名が得られない場合には，コマンドは正しく実行されないことに注意してください。

次の例は，IF コマンドでの反復置換を示しています。

```
$ P1 = "FRED.DAT"  
$ COUNT = 1  
$ IF P'COUNT' .EQS. "" THEN GOTO END
```

コマンド・インタプリタはこの行を検索して，COUNT シンボルをその現在の値と置き換えます。結果は次のようになります。

```
IF P1 .EQS. "" THEN GOTO END
```

この文字列には一重引用符がないため，コマンド・インタプリタはこれ以上置換を実行しません。ただし，IF コマンドを実行すると，シンボル名 P1 を自動的に評価して，その現在の値と置き換えます。

次の例では，シンボル名 FILENAME は無効です。

```
$ FILENAME = "A.B"  
$ IF 'FILENAME' .NES. "" THEN TYPE 'FILENAME'
```

コマンド・インタプリタは，FILENAME シンボルをその現在の値 (A.B) に置き換えます。結果は次のようになります。

```
IF A.B .NES. "" THEN TYPE A.B
```

IF コマンドがコマンド行を実行するとき，A.B は有効なシンボルでないため，エラーになります。この IF コマンドを正しく処理するためには，次のように，一重引用符を省略します。

```
$ IF FILENAME .NES. "" THEN TYPE 'FILENAME'
```

12.13.5 未定義シンボル

コマンド行でシンボルを使用するときにそのシンボルが未定義の場合には，コマンド・インタプリタは，コンテキストに応じて，エラー・メッセージを表示するか，シンボルを空の文字列と置き換えます。未定義シンボルについては，次の規則が適用されます。

- コマンド処理の第1フェーズと第2フェーズでは，前に一重引用符またはアンパサンドが付いたすべての未定義シンボルを空の文字列と置換する。
- コマンド処理の第3フェーズでは，未定義シンボルを見つけると，警告メッセージを表示して処理を最後まで実行しない。

次の例は，コマンド・インタプリタが前に一重引用符が付いた未定義シンボルをどのように処理するかを示しています。

```
$ FILE := MYFILE'FILE_TYPE'  
$ SHOW SYMBOL FILE  
FILE = "MYFILE"  
$ PRINT 'FILE'
```

FILE シンボルが作成されると，FILE_TYPE シンボルがその現在の値と置き換えられます。FILE_TYPE が定義されていない場合は，FILE_TYPE は空の文字列と置き換えられます。ファイル指定の中にファイル・タイプがないと，PRINT コマンドは省略時のファイル・タイプである LIS を使用します。したがって，ファイル指定は MYFILE.LIS と解釈されます。

次の例では，コマンド処理の第3フェーズで式が評価されます。

```
$ A = 1  
$ C = A + B  
%DCL-W-UNDSYM, undefined symbol - check validity and spelling
```

B シンボルは未定義であるため，コマンド・インタプリタは式を評価できません。

12.14 シンボルを使用するための別の方法: 自動的なフォーリン・コマンド

プロシージャに対してシンボルを定義せずに，DCL レベルからコマンド・プロシージャ (ファイル・タイプ.COM) または実行可能イメージ (ファイル・タイプ.EXE) を起動することもできます。自動的なフォーリン・コマンドを使用すると，DCL は特定のディレクトリからコマンド・プロシージャまたは実行可能イメージを検索して，それを自動的に実行できます。

DCL シンボルでなく，DCL コマンド・テーブルに登録されていないコマンド動詞を入力すると，一般に次のメッセージが表示されます。

```
DCL-W-IVVERB, unrecognized command verb - check validity and spelling
```

しかし，論理名 DCLSPATH が定義されている場合には (およびブランクでない場合には)，DCL はファイル名に無効動詞が含まれており，省略時のファイル指定として DCLSPATH:.* が指定されているファイルに対して，RMS \$SEARCH を実行します。

DCL が.COM または.EXE ファイルを検索すると，コマンド行の残りの部分をパラメータとして使用して，そのファイルを自動的に実行します (この動作は，DOS，UNIX，その他のオペレーティング・システムの PATH オプションによく似ています)。

次の例では，DCL シンボル SYSGEN は必要ありません。DCL は SYS\$SYSTEM ディレクトリから SYSGEN.EXE を検索します。DCL は，シンボル "SYSGEN" が

"\$SYSS\$SYSTEM:SYSGEN"として定義されているかのように動作し，SYSGEN イメージをフォーリン・コマンドとして起動します。

```
$ 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
```

次の例では，SS は "@SS.COM"として定義しておく必要がありません。これは，DCL が SYSS\$SYSTEM ディレクトリから SS.COM または SS.EXE を自動的に検索するからです。その処理を正しく実行できない場合には，DCL はカレント・ディレクトリから SS.COM または SS.EXE を検索します。

```
$ 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 = ""
```

この例では，DCL は SS.COM を検索し，"SS"が "@SS.COM"として定義されているシンボルであるかのように動作します。コマンド行の残りの部分をパラメータとして解析して，コマンド・プロシージャが起動されます。"SS.EXE"がイメージ SS.EXE を起動するのではなく，2つのパラメータを使用して SS.COM が起動され，最初のパラメータが ".EXE"というテキスト文字列であることに注意してください。これは，OpenVMS オペレーティング・システムで，コマンドの解析とシンボルの置換が実行される方法と一貫しています。

12.14.1 自動的なフォーリン・コマンドの使用

次のことに注意してください。

- 論理名 DCL\$PATH は検索リスト・タイプの論理名でもかまわない。
- 論理名の各変換のノード, デバイス, ディレクトリの部分だけが使用される。
- 通常の論理名の優先順位が有効である。ユーザは独自の論理名を定義することにより, DCL\$PATH のシステム定義を変更できる。システム定義が存在するときに, ユーザがその機能を使用したくない場合には, 論理名の定義を " " に変更することにより, システム定義を無効にできる。
- DCL の動詞およびシンボル名に対して使用できる文字は, ファイル名に対して使用できる文字と異なる。たとえば, DCL シンボルではハイフン (-) を使用できず, 1 文字目にドル記号 (\$) を使用することもできない。実行するイメージまたはプロシージャが DCL シンボル名として正しくない場合には, この新しい機能を使用して直接起動することはできない。
- DCL はコマンドを解析していない。起動されるイメージが独自のコマンド解析を実行しなければならない。C プログラムの場合には, main() ルーチンに対して "argc" と "argv" パラメータを使用する。他の言語で作成されたプログラムの場合には, LIB\$GET_FOREIGN を呼び出して, コマンド行全体を入手する。その後, コマンド行をプログラムで解析しなければならない。
- ディレクトリにコマンド・プロシージャと実行可能イメージの両方が登録されている場合には, 最初に検索されたファイルが起動される。OpenVMS システムでは, ディレクトリはアルファベット順に並べられているため, ".COM" ファイルの方が ".EXE" ファイルより先に検索される。他のオペレーティング・システムを稼動しているノードを示す DCL\$PATH 論理名に, ネットワーク・ファイル指定を指定した場合には, ".COM" ファイルより前に ".EXE" ファイルが検索される可能性がある。

DCL は無効動詞をファイル指定として使用し, "DCL\$PATH:.*" を省略時のファイル指定として使用して検索を実行するため, 特定のファイルが検索されるような方法で論理名を定義することができる。たとえば, 論理名 FOO を "FOO.EXE" として定義し, DCL プロンプトに対して "FOO" と入力すると, FOO.COM は絶対に起動されず, FOO.EXE だけが起動される。

注意

特権付きユーザが省略時のデバイスおよびディレクトリを他のユーザ・アカウントに設定した場合には, "SYS\$DISK:[]" を DCL\$PATH 論理名の定義に指定しないでください。この操作を実行すると, DCL はカレント・ディレクトリを検索します。入力エラーがある場合や, 検索リストで変換の配置方法が不適切な場合には, カレント・ディレクトリのユーザ・イメージが検索され, 特権が与えられた状態で誤って起動される可能性があります。

12.14.2 自動的なフォーリン・コマンドの制限事項

次の制限事項に注意してください。

- OpenVMS オペレーティング・システム Version 6.2 より以前のバージョンで，自動的なフォーリン・コマンドを使用することはできない。
- 新しい動詞をいつでも DCL コマンド・テーブルに追加することができるため，ある時点で自動的なフォーリン・コマンドによって動作したコマンドが，将来動作しなくなる可能性がある。
- 自動的なフォーリン・コマンド機能は，どのような場合にも動作するわけではない。次の例では，DCL (動詞の最初の 4 文字だけを調べる) は，SHOW という動詞 (SHOWME の最初の 4 文字) と一致するものを検索し，SHOWME.COM プロシージャではなく，SHOW USERS コマンドを実行する。SHOWME を DCL シンボルとして定義した場合には，SHOWME コマンドは SHOWME.COM を起動する。

```
$ 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
```


コマンド・プロシージャの概要

コマンド・プロシージャとは、DCL コマンドと、DCL コマンドで使用するデータ行が格納されたファイルです。1 つか 2 つの DCL コマンドしか入っていない簡単なコマンド・プロシージャもありますが、複雑なコマンド・プロシージャになると、洗練されたコンピュータ・プログラムのような働きをします。コマンド・プロシージャを実行すると、DCL インタプリタはファイルを読み込んで、その中に入っているコマンドを実行します。

システム管理者がシステム・ログイン・コマンド・プロシージャを設定した場合には、ログインするたびにそれが実行されます。システム・ログイン・コマンド・プロシージャを使用すれば、システム管理者は、自分を含むシステム上のすべてのユーザがログインしたときに、必ず特定のコマンドが実行されるように設定できます。

システム・ログイン・コマンド・プロシージャを実行した後、システムはパーソナル・ログイン・コマンド・プロシージャを実行します (存在する場合)。パーソナル・ログイン・コマンド・プロシージャは、システム環境をカスタマイズするためのものです。パーソナル・ログイン・コマンド・プロシージャに登録されているコマンドは、ログインするたびに実行されます。つまり、ログイン時には、最大 2 つのログイン・コマンド・プロシージャが自動的に実行されます (システム全体のログイン・コマンド・プロシージャの他にユーザ独自のログイン・コマンド・プロシージャがある場合はその両方)。

ユーザのアカウントを設定した人が、ユーザの最上位ディレクトリにログイン・コマンド・プロシージャを格納することもあります。ログイン・コマンド・プロシージャが最上位ディレクトリに格納されていない場合には、自分でログイン・コマンド・プロシージャを作成し、LOGIN.COM という名前を付け、最上位ディレクトリに格納することもできます。システム管理者が特に指定しない限り、ユーザがログインしたとき、そのユーザが作成した LOGIN.COM ファイルが実行されます。

本章では、次のことについて説明します。

- コマンド・プロシージャを作成するための基礎的な説明
- コマンド・プロシージャの作成手順
- コマンド・プロシージャの実行
- コマンド・プロシージャの終了、解釈、エラー処理
- ログイン・コマンド・プロシージャ

DCL コマンド・プロシージャには、次の 2 種類があります。

- 単純なプロシージャ
作成された順に、一連の DCL コマンドを実行する。
- 複雑なプロシージャ
プログラムのような機能を実行する。

13.1 コマンド・プロシージャを作成するための基礎的な説明

コマンド・プロシージャを作成するには、次の 2 種類の方法があります。

- EVE のようなテキスト・エディタを使用して、新しいファイルを作成する方法
- DCL コマンド CREATE を使用して、新しいファイルを作成する方法

作成するファイルには、コマンド行、ラベル、コメント、条件文、変数を格納できます。

13.1.1 省略時のファイル・タイプ

コマンド・プロシージャの省略時のファイル・タイプは.COM です。コマンド・プロシージャの名前を指定するときに、ファイル・タイプとして.COM を指定した場合には、ファイル名を指定するだけで、コマンド・プロシージャを実行できます。SUBMIT コマンドとプロシージャ実行(@)コマンドでは、特に指定した場合を除き、ファイル・タイプは.COM であるものと解釈されます。

13.1.2 コマンドの作成

コマンド・プロシージャにコマンドを登録する場合には、次のことに注意してください。

- コマンドと修飾子の名前は完全に指定する。このようにすると、OpenVMS の将来のリリースに対して、コマンド・プロシージャの互換性を維持できる。
- プロシージャを読みやすくするために、継続行を使用する。継続行の先頭には、ドル記号がない。次の例を参照。

```
$ PRINT LAB.DAT -  
    /AFTER=17:00 -  
    /COPIES=20 -  
    /NAME="COMGUIDE"
```

13.1.3 コマンド行の作成

コマンド行を作成する場合には、次のことに注意してください。

- コマンド、コメント、ラベルを指定する各行の先頭には、ドル記号(\$)を指定しなければならない。
- データを含む行が必要な場合には、その行にはドル記号(\$)を指定しない。
- ドル記号(\$)から始まるデータ行を指定しなければならない場合には、DCL コマンド DECK と EOD を使用する。次の例を参照。

```
$ ! 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
```

先頭にドル記号が指定されていないコマンド行も、DCL で正しく解釈されると思われますが、なるべく DCL コマンド行の先頭にはドル記号を指定してください。

13.2 コマンド行でのラベルの使用

DCL コマンド・プロシージャでは、ループ、コード・セクション、サブルーチンの先頭をマークするためにラベルを使用します。ラベルを使用する場合には、次の規則に注意してください。

- ループ、サブルーチン、条件コードをわかりやすく区別するために、ラベルは単独の行に指定する。
- 255 文字以内のラベル名を使用し、名前の内部ではスペースを使用しない。
- ラベルとコマンドの区別は、ラベルの場合は、ドル記号(\$)のすぐ後に指定し、コマンドの場合は、その前にスペースを指定することにより行う。
- 各ラベルの最後にはコロンを指定する。
- ラベルを削除することはできない。

13.2.1 ローカル・シンボル・テーブル内のラベル

コマンド・インタプリタがラベルを検出すると、ローカル・シンボル・テーブルの特殊なセクションにそのラベルが登録されます。ラベルのために使用できる容量は制限されています。コマンド・プロシージャで多くのシンボルを使用し、多くのラベルが含まれている場合には、コマンド・インタプリタはシンボル・テーブル空間をすべて使用してしまう可能性があり、エラー・メッセージが表示されます。この場合には、プロシージャに DELETE/SYMBOL コマンドを指定して、不要になったシンボルを削除します (ただし、ラベルを削除することはできません)。

13.2.2 重複するラベル

コマンド・プロシージャで同じラベルを 2 回以上使用する場合には、ローカル・シンボル・テーブルの既存の定義は新しい定義に置き換えられます。

重複するラベルが存在する場合には、GOTO コマンドは、DCL が最後に処理したラベルに制御を渡します。また、GOTO コマンドは重複するラベルを処理するときに、次の規則も使用します。

- 重複するラベルがすべて GOTO コマンドの前に指定されている場合には、制御は、GOTO コマンドにもっとも近いラベルに渡される。
- 重複するラベルが GOTO コマンドの前と後に指定されている場合には、制御は、前に指定されているラベルのうち、GOTO コマンドにもっとも近いラベルに渡される。
- 重複するラベルがすべて GOTO コマンドの後に指定されている場合には、制御は、GOTO コマンドにもっとも近いラベルに渡される。

13.3 コマンド・プロシージャでのコメントの使用

コマンド・プロシージャを作成する場合には、なるべくコメントを指定してください。コマンド・プロシージャを更新したり、問題を解決するときに、コメントが指定されていると役立ちます。コメントは次の方法で使用できます。

- プロシージャの先頭で、プロシージャについて説明し、プロシージャに渡されるパラメータについても説明する。
- 各コマンド・ブロックの先頭で、プロシージャのそのセクションについて説明する。
- ドル記号(\$)と感嘆符(!)をどちらも含む行で、コマンド・シーケンスを区切る。このようにすると、コマンド・プロシージャの概略がわかりやすくなる。ブランク行を挿入すると、コマンド・インタプリタはそれらの行をデータ行として解釈し、データ行が無視されたことを警告するメッセージが生成される。

コマンド・プロシージャにコメントを指定する場合には、次の規則が適用されます。

- コメントの先頭を示すには、感嘆符(!)を使用する。コマンド・プロシージャを実行する場合、コマンド・インタプリタは感嘆符の右側のすべてのテキストを無視する。
- コマンド行にリテラルとして感嘆符を指定するときは、感嘆符を引用符で囲む(" ")。

13.4 コマンド・プロシージャの作成方法

コマンド・プロシージャの作成を開始する前に、コマンド・プロシージャが実行するタスクを会話形式で実行します。必要なコマンドを入力し、使用される変数と条件、および発生する会話を記録します。

これ以降の節では、簡単なコマンド・プロシージャの作成方法を説明します。これ以降で使用する例は、CLEANUP.COM というコマンド・プロシージャです。このコマンド・プロシージャは、ディレクトリを整理するために使用します。

定義

- 変数
タスクを実行するたびに変化するデータ
- 条件
変化する可能性のあるコマンドまたはコマンドの集合であり、タスクを実行するたびにテストしなければならない。
- 繰り返し
条件が満足されるまで繰り返し実行されるコマンドまたはコマンドの集合。

13.5 コマンド・プロシージャの作成手順

コマンド・プロシージャを作成するには、次の手順を実行します。

手順	操作
1	コマンド・プロシージャを設計する。
2	変数を割り当て、条件をテストする。
3	ループを追加する。
4	コマンド・プロシージャを終了する。
5	プログラム・ロジックをテストおよびデバッグする。
6	クリーアップ・タスクを追加する。
7	プロシージャを完成する。

13.5.1 手順 1: コマンド・プロシージャを設計する

コマンド・プロシージャを設計するには、次の操作を実行します。

手順	操作
1	プロシージャが実行するタスクを決定する。
2	コマンド・プロシージャが使用する変数と、その変数のロード方法を判断する。
3	コマンド・プロシージャが必要とする条件と、条件をテストする方法を判断する。
4	コマンド・プロシージャを終了する方法を決定する。

クリーンアップ操作で一般に実行される特定のコマンドがあります。次の表は、これらのコマンドと、そのコマンドが実行するタスクを示しています。

コマンド	実行するタスク
DIRECTORY	カレント・ディレクトリの内容を表示する。
TYPE filespec	ファイルを表示する。
PURGE filespec	ファイルをパージする。
DELETE filespec	ファイルを削除する。
COPY filespec new-filespec	ファイルをコピーする。

変数

タスクを実行するときに変化するデータは変数です。ディレクトリにファイルを作成したり、ファイルを削除する場合には、ディレクトリを整理するたびに、ファイル名が異なります。したがって、CLEANUP.COM でファイル名は変数です。

条件

コマンド・プロシージャを実行するたびにテストしなければならないコマンドは、条件であると考えられます。CLEANUP.COM のコマンドは、実行しなければならない操作に応じて、一部またはすべてが実行されるため、各コマンドは条件です。

設計の決定

CLEANUP.COM コマンド・プロシージャで使用する変数と条件を判断した後、変数のロード方法、条件のテスト方法、コマンド・プロシージャの終了方法を判断しなければなりません。CLEANUP.COM コマンド・プロシージャの場合には、次の判断を下しました。

タスク	実現方法
変数のロード	コマンド・プロシージャはターミナルからファイル名を入手する。
条件のテスト	コマンド・プロシージャ: <ul style="list-style-type: none">ターミナルからコマンド名を入手し、コマンド名をもとにして適切な文を実行する。DELETE コマンドと DIRECTORY コマンドを区別するために、各コマンド名の最初の 2 文字を確実に読み込む。
ループの終了	ループを終了するには、EXIT コマンドを入力しなければならない。

コマンド・プロシージャを理解しやすく、また管理しやすくするには、プロシージャが最初のコマンドから最後のコマンドへと順に実行されるように、文を作成しなければなりません。

13.5.2 手順 2: 変数を割り当て、条件をテストする

値を変数に割り当てるには、多くの方法があります。この節では、INQUIRE コマンドの使い方について説明します。他の方法については、第 14 章を参照してください。

値を変数に割り当て、条件をテストするには、次の操作を実行します。

手順	操作
1	INQUIRE コマンドを使用して、値を変数に割り当てる。
2	実行しなければならない処理を判断する。
3	IF 文と THEN 文を使用して条件をテストする。
4	プログラム・スタブを作成し、コマンドのプレースホルダとしてコマンド・プロシージャに挿入する。
5	必要に応じてエラー・メッセージを作成する。

13.5.2.1 INQUIRE コマンドの使用

INQUIRE コマンドは、値を要求するプロンプトを表示し、ターミナルから値を読み込み、その値をシンボルに割り当てます。

省略時の設定では、INQUIRE コマンドは次の操作を実行します。

- 応答を大文字に変換する。
- 複数のブランクとタブを 1 つのスペースに置換する。
- 先頭と後続のスペースを削除する。
- 応答にシンボルやレキシカル関数が含まれている場合には、引用符置換を実行する。

次の例は、コマンド・プロシージャ CLEANUP.COM の中でコマンド名の入力を求めるコマンドです。INQUIRE コマンドは入力された値をシンボル COMMAND に割り当てます。

```
$ INQUIRE COMMAND-  
  "Enter command (DELETE, DIRECTORY, PRINT, PURGE, TYPE)"
```

13.5.2.2 リテラル文字の保存

INQUIRE コマンドを使用するときに、小文字や複数のスペースとタブを保存するには、応答を引用符 (" ") で囲みます。応答の内部で引用符を使用するときは、引用符で囲んだテキストを引用符で囲みます ("text")。

13.5.2.3 IF と THEN を使用した条件のテスト

INQUIRE コマンドで変数を要求するプロンプトを表示した後、どのような処理を実行するかを判断する文をコマンド・プロシージャに指定しなければなりません。たとえば、どのコマンドを実行するかを判断するには、ユーザが入力したコマンドを可能な各コマンドに対してチェックする文を、コマンド・プロシージャに指定しなければなりません。

条件が真であるかどうかをテストするには、IF コマンドと THEN コマンドを使用します。次の表は、CLEANUP.COM でチェックしなければならない可能性を示しています。

場合	結果
一致するものが見つかった場合	コマンドを実行する
一致するものが見つからない場合	次のコマンドに進む
有効なすべてのコマンドをチェックした後、一致するものが見つからない場合	エラー・メッセージを出力する

13.5.2.4 プログラム・スタブの作成

プログラム・スタブとは、設計をテストするときに、プロシージャで使用する一時的なコード・セクションです。通常、プログラム・スタブは、そのスタブのかわりに実際に使用される機能を示すメッセージを出力します。全体の設計が正しく動作することを確認した後は、各スタブを正しいコーディングに置き換えます。

例: 変数の割り当てと条件のテスト

次の例では、変数の割り当てと条件のテスト方法を示しています。

```
$ 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
```

例を確認するときは、次のことに注意してください。

- 1 この IF 文は、ユーザが入力したコマンド (COMMAND) が "DELETE" に等しいかどうかをテストする。COMMAND が DELETE に等しい場合には、コマンド・プロシーダは次のコマンドを実行する。
- 2 この文には GOTO コマンドも指定されている。GOTO コマンドは、実行の流れをプロシーダ内のラベルに変更するために使用する。この場合、COMMAND が DELETE に等しくない場合には、プロシーダは DIRECTORY ラベルに進む。
- 3 この文はプログラム・スタブである。コマンド・プロシーダのロジックのテストが終了した後、この行は DELETE 操作にとって必要な実際のコマンドに置き換えられる。
- 4 これは DIRECTORY サブルーチンのラベルである。各コマンド・ブロックを識別するラベルは、オプション・リストのコマンドと同じである。このため、GOTO 文でシンボル COMMAND を使用できる (これはユーザの要求に等しく定義されている)。
- 5 この IF 文は、"TYPE" コマンドが入力されたかどうかをテストする。"TYPE" が入力された場合には、プロシーダは "This is the TYPE section." を出力する。しかし、これはテストする最後のコマンドであるため、入力されたコマンドが "TYPE" でない場合には、プログラムはエラー・メッセージを表示する。
- 6 すべてのコマンドをテストした後、正しいコマンド名が見つからない場合には、プログラムは "You have entered an invalid command." を出力する。

13.5.3 手順 3: ループを追加する

ループとは、条件が満足されるまで繰り返し実行される文の集まりです。ループは次のように機能します。

1. ユーザ入力から値を入手します。
2. コマンドを処理します。
3. ユーザがコマンド・プロシージャを終了するまで、処理を繰り返します。

ループを作成するには、次の手順を実行します。

手順	操作
1	ループの先頭にラベルを付ける。
2	変数を調べて、ループの中のコマンドを実行する必要があるかどうかを判別する。
3	ループを実行する必要がある場合には、ループの終わりに進む。
4	ループを実行する必要がある場合には、ループ本体の中のコマンドを実行してから、ループの始めに戻る。
5	ループを終了する。

次の例では、CLEANUP.COM コマンド・プロシージャでのループの使い方を示しています。

```
$ 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
```

コマンドを実行した後、ユーザが EXIT コマンドを入力するまで、制御は GET_COM_LOOP ラベルに戻されます。EXIT コマンドが入力されると、プロシージャはディレクトリが整理されたことを示すメッセージを出力します。

13.5.4 手順 4: コマンド・プロシージャを終了する

コマンド・プロシージャを終了するには、次の手順を実行します。

手順	操作
1	どこでコマンド・プロシージャを終了するかを判断する。
2	必要に応じて EXIT または STOP コマンドを指定する。

13.5.4.1 EXIT コマンドの使用

コマンド・プロシージャで EXIT コマンドを使用すると、次のことが可能になります。

- プロシージャが特定の行を実行しないようにすることができる。
- 複数の実行パスを持つプロシージャを終了できる。
- コマンド・プロシージャを終了できる。

次の例は、プロシージャの最後に指定されているエラー処理ルーチンを実行しないようにするために、EXIT コマンドを使用する方法を示しています。

```
.  
. .  
. .  
$ EXIT ! End of normal execution path  
$ ERROR_ROUTINE  
. .  
. .
```

次の例は、複数の実行パスを持つプロシージャを終了するために、EXIT コマンドを使用する方法を示しています。

```
$ 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
```

各ラベル (TAPE と DISK) の後のコマンドは、プロシージャで異なるパスを与えます。DISK ラベルの前の EXIT コマンドは、プロシージャがそのラベルに明示的に分岐した場合を除き、DISK ラベルの後のコマンドが実行されないようにします。

プロシージャの最後に EXIT コマンドを指定する必要はありません。これは、プロシージャのエンド・オブ・ファイルにより、暗黙に EXIT コマンドが実行されるからです。しかし、なるべく EXIT コマンドを指定するようにしてください。

13.5.4.2 STOP コマンドの使用

コマンド・プロシージャで STOP コマンドを使用すると、重大なエラーが発生したときに、プロシージャを終了することができます。会話形式で実行されているコマンド・プロシージャで STOP コマンドを使用した場合には、制御は DCL レベルに戻されます。バッチ・モードで実行されているコマンド・プロシージャの場合には、バッチ・ジョブが終了します。

このコマンド行は、重大なエラーが発生したときに、プロシージャを停止するように要求します。

```
$ ON SEVERE_ERROR THEN STOP
```

13.5.5 手順 5: プログラム・ロジックをテストおよびデバッグする

プログラム・スタブを使用してコードを作成した場合には、コマンド・プロシージャの全体的なロジックをテストしなければなりません。可能性のあるすべての実行パスをテストしなければなりません。

コマンド・プロシージャをテストおよびデバッグするには、次の手順を実行します。

手順	操作
1	コマンド・プロシージャに正しい各コマンドを入力して、プログラム・ロジックをテストする。
2	誤ったコマンドを入力して、プログラム・ロジックのテストを継続する。
3	EXIT コマンドを使用して、コマンド・プロシージャを終了することにより、プログラム・ロジックのテストを終了する。
4	必要な場合には、SET VERIFY, SET PREFIX, SHOW SYMBOL コマンドを使用して、プログラムをデバッグする。

次の例では、可能な各コマンドと無効なコマンドを入力および実行して、コマンド・プロシージャをテストし、最後にプロシージャを終了する方法を示しています。

```
$ @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
$
```


13.5.5.1 コマンド・プロシージャのデバッグ

次のコマンドは、コマンド・プロシージャをデバッグするのに役立ちます。

- SET VERIFY

実行する前に各行を表示する。チェック設定でエラーが発生した場合には、そのエラーと、エラーを生成した行が表示される。SET VERIFY コマンドでは、コマンド行だけをチェックするのか、データ行もチェックするのかを指定するために、キーワードを使用できる。

SET VERIFY コマンドは、ログアウトするまで、SET NOVERIFY コマンドを入力するまで、または F\$VERIFY レキシカル関数を使用してチェック設定を変更するまで有効である (チェック設定の変更についての詳しい説明は、第 15 章を参照)。

- SET PREFIX

チェック機能が有効な場合には、DCL コマンド SET PREFIX を使用し、各コマンド行の前に、そのコマンドが実行された時刻を挿入することにより、プロシージャ・ログ・ファイルにタイム・スタンプを出力できる。

- SHOW SYMBOL

SHOW SYMBOL コマンドを使用すれば、プロシージャ内のシンボルがどのように定義されているかを判断できる。

例: SET VERIFY コマンドの使用によるデバッグ

次の例では、END_LOP というラベルの綴りが誤っています。チェック機能が有効に設定されているため、エラーがどこで発生したのかを正確に判断できます。

```
$ 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
```

エラーを修正するには、ラベルを END_LOOP に変更します。

例: SET PREFIX コマンドの使用によるデバッグ

次の例ではタイム・スタンプの使い方を示しています。

```
$ 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]
```

コマンド・プロシーダの概要

13.5 コマンド・プロシーダの作成手順

例: SHOW SYMBOL コマンドの使用によるデバッグ

次の例では、SHOW SYMBOL コマンドを使用して、シンボル COMMAND がどのように定義されているか判断する方法を示しています。

```
$ 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 コマンドを実行すると、シンボル COMMAND の値が "EXIT" であることがわかります。INQUIRE コマンドは入力を大文字に自動的に変換しますが、コマンドをテストする IF 文では、文字列 "exit" で小文字を使用しているため、DCL は文字列が等しくないと判断します。このエラーを修正するには、IF 文に指定した、引用符で囲んだ文字列を大文字にします。文字列の残りの部分は、大文字でも小文字でもかまいません。

13.5.5.2 実行中にチェック機能をオンにする

コマンド・プロシーダの実行中に割り込みをかけて、チェック機能を有効にすることもできます。コマンド・プロシーダに SET VERIFY コマンドや Ctrl/Y キー・シーケンスが含まれていない限り、次の操作を実行して、チェック機能を有効に設定できます。

手順	操作
1	Ctrl/Y を押して実行に割り込みをかける。
2	SET VERIFY コマンドを入力する。
3	CONTINUE コマンドを入力して、コマンド・プロシーダの実行を続行する (チェック機能を有効にした状態で)。

13.5.6 手順 6: クリーンアップ・タスクを追加する

一般に、コマンド・プロシーダを実行した結果、ユーザのプロセス状態が変化しないようにしなければなりません。したがって、コマンド・プロシーダには、プロセスをもとの状態に戻す一連のコマンドを指定しなければなりません。このコマンドは通常、"CLEAN_UP" というラベルの付いたサブルーチンの一部です。一般的なクリーンアップ操作としては、ファイルを閉じる操作と、省略時のデバイスおよびディレクトリをリセットする操作があります。

コマンド・プロシージャにクリーンアップ・タスクを追加するには、次の操作を実行します。

手順	操作
1	CLEAN_UP などのラベルを使用して、クリーンアップ・サブルーチンを開始する。
2	F\$GETJPI レキシカル関数を使用して、開かれているファイルがあるかどうかテストする。
3	DELETE コマンドまたは PURGE コマンドを使用して、一時ファイルまたは余分なファイルを削除する。
4	(デバイスやディレクトリなど) 省略時の設定を変更した場合には、SET DEFAULT コマンドを使用して、元の状態に復元する。
5	クリーンアップ操作が確実に実行されるように、ON CONTROL_Y 文を指定する。

13.5.6.1 ファイルを閉じる

ファイルが開かれている場合には、プロシージャを終了する前に確実にファイルを閉じてください。レキシカル関数 F\$GETJPI を使用すれば、プロセスに対して残されているオープン・ファイル・クォータ (FILCNT) を調べることができます。FILCNT が、コマンド・プロシージャの開始時と終了時に同じである場合には、開かれたままになっているファイルはありません。

この例では、ファイルが開かれたままになっていることをユーザに警告するために使用するコマンドを示しています。

```
$ FIL_COUNT = F$GETJPI ("", "FILCNT")
.
.
.
$ IF FILCNT .NE. F$GETJPI ("", "FILCNT") THEN-
  WRITE SYS$OUTPUT "WARNING -- file left open)
```

13.5.6.2 一時ファイルまたは余分なファイルの削除

一時ファイルを作成した場合には、それを削除します。一般に、ファイルを更新した場合には、ファイルをパージして以前のコピーを削除しなければなりません。自分で作成していないファイルを削除する場合には、そのファイルを削除してもかまわないかどうか確認してください。たとえば、重要なデータが格納されているファイルを更新した場合には、パージ操作は必要に応じて省略できます。

省略時のデバイスとディレクトリのどちらか一方または両方を変更した場合には、コマンド・プロシージャを終了する前に、元の省略時の設定に戻さなければなりません。元の省略時のディレクトリの名前をセーブするには、F\$ENVIRONMENT レキシカル関数の DEFAULT キーワードを使用します。コマンド・プロシージャの最後に、保存したデバイスとディレクトリを復元する SET DEFAULT コマンドを指定します。

この例に示したコマンド行は、デバイスとディレクトリの省略時の設定を保存し、復元します。

コマンド・プロシージャの概要

13.5 コマンド・プロシージャの作成手順

```
$ SAV_DEFAULT = F$ENVIRONMENT ("DEFAULT")  
.  
.  
.  
$ SET DEFAULT 'SAV_DEFAULT'
```

13.5.6.3 一般に変更されるプロセス属性

次の表は、一般に変更されるその他のプロセス属性、これらの属性をセーブするために使用されるレキシカル関数、復元するために使用されるレキシカル関数またはコマンドを示しています。

属性	セーブのための レキシカル関数	復元のための レキシカル関数
DCL プロンプト	F\$ENVIRONMENT	SET PROMPT
省略時の保護	F\$ENVIRONMENT	SET PROTECTION/DEFAULT
特権	F\$SETPRV	F\$SETPRV または SET PROCESS/PRIVILEGES
制御文字	F\$ENVIRONMENT	SET CONTROL
検査	F\$VERIFY	F\$VERIFY
メッセージ・フォーマット	F\$ENVIRONMENT	SET MESSAGE
キーの状態	F\$ENVIRONMENT	SET KEY

これらのレキシカル関数についての詳細は、『OpenVMS DCL デクショナリ』を参照してください。

13.5.6.4 クリーンアップ操作を確実に実行するには

コマンド・プロシージャが途中で打ち切られる場合でも、クリーンアップ操作が確実に実行されるようにするには、コマンド・プロシージャの各コマンド・レベルを次の文で開始します。

```
$ ON CONTROL_Y THEN GOTO CLEANUP
```

ON CONTROL_Y コマンドの使用方法についての詳細は、第 14 章を参照してください。

13.5.7 手順 7: コマンド・プロシージャを完成する

全般的な設計が正しく動作している場合には、次の操作を実行してコマンド・プロシージャを完成します。

手順	操作
1	コマンド・プロシージャ内の最初のプログラム・スタブをコマンドに置き換える。
2	コマンド・プロシージャをテストして、新しいコマンドが正しく機能するかどうか確認する。
3	必要に応じて、コマンド・プロシージャをデバッグする。

手順	操作
----	----

- | | |
|---|--|
| 4 | 最初のプログラム・スタブが正常に機能する場合には、次のスタブに移動する。すべてのプログラム・スタブが置換されるまで、この操作を続行する。 |
|---|--|
-

例: プログラム・スタブとコマンドの置換

次の例は、CLEANUP.COM のTYPE セクションのコードを示しています。

```
$! Execute if user entered TYPE
$! TYPE:
$   IF COMMAND .NES. "TYPE THEN GOTO ERROR
$   INQUIRE FILE "File to type"
$   TYPE 'FILE'
$   GOTO GET_COM_LOOP
```

既存のコードは次のように置き換えられます。

```
$ WRITE SYS$OUTPUT "This is the TYPE section."
```

例: CLEANUP.COM コマンド・プロシーダ

次の例は、完成した CLEANUP.COM コマンド・プロシーダを示しています。

```
$ 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
$!
```

コマンド・プロシージャの概要

13.5 コマンド・プロシージャの作成手順

```
$!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
```

13.6 コマンド・プロシージャの実行

コマンド・プロシージャを動作させるには、実行しなければなりません。コマンド・プロシージャは次の方法で実行できます。

- 別のコマンド・プロシージャの内部から実行する方法
- リモート・ノードで実行する方法
- DCL コマンドにパラメータまたは修飾子として指定する方法
- 会話形式で実行する方法
- バッチ・ジョブとして実行する方法
- ディスクおよびテープ・ボリューム上で実行する方法

この節では、これらの各方法について説明します。

13.6.1 他のコマンド・プロシージャの内部からのコマンド・プロシージャの実行

プロシージャ実行コマンド(@)を使用すれば、別のコマンド・プロシージャの内部からコマンド・プロシージャを実行できます。

次のコマンド・プロシージャ WRITEDATE.COM は、コマンド・プロシージャ GETDATE.COM を起動します。

```
$! WRITEDATE.COM
$!
$ INQUIRE TIME "What is the current time in hh:mm format?"
$ @GETDATE [JONES.COM]GETDATE.COM
```

13.6.2 リモート・ノードでのコマンド・プロシージャの実行

TYPE コマンドを使用すると、リモート・ノードの別のアカウントのトップ・レベル・ディレクトリでコマンド・プロシージャを実行できます。次の操作を実行するコマンド・プロシージャを実行できます。

- ローカル OpenVMS Cluster のサービスのうち、クラスタ全体に提供されないサービスの状態を表示する。
- リモート・ノードにログインしているユーザの一覧を表示する。

TYPE コマンドの後にアクセス制御文字列を入力します。次の形式を使用してください。

```
$ TYPE nodename"username password"::"TASK=command_procedure"
```

username および password という変数は、リモート・ノードのアカウントのユーザ名とパスワードです。

このコマンド・プロシージャは、コマンド・プロシージャが存在するリモート・ノードにログインしているユーザを表示します。

```
$!SHOWUSERS.COM
$!
$ IF F$MODE() .EQS. "NETWORK" THEN DEFINE/USER SYS$OUTPUT SYS$NET
$ SHOW USERS
```

次の例では、SHOWUSERS.COM はノード ORIOLE の BIRD のアカウントのトップ・レベル・ディレクトリに格納されており、パスワードは BOULDER です。SHOWUSERS.COM は、リモート・ノード ORIOLE で DCL コマンド SHOW USERS を実行します。TYPE コマンドは、SHOWUSERS.COM からの出力をローカル・ノード、つまり type コマンドを入力したターミナルに表示します。

```
$ 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
```

13.6.2.1 セキュリティに関する注意

TYPE コマンドとアクセス制御文字列を入力すると、ターミナルにパスワードが表示されます。第 16 章の説明に従って、セキュリティに関する適切な予防措置をとってください。

13.6.3 DCL 修飾子またはパラメータを指定したコマンド・プロシージャの実行

DCL コマンドのパラメータまたは修飾子を指定するコマンド・プロシージャを作成できます。この種のコマンド・プロシージャは、1 つ以上のコマンドで特定のパラメータまたは修飾子の組み合わせを頻繁に使用するとき便利です。

コマンド行で修飾子やパラメータを指定する場所に、プロシージャ実行コマンド(@)を入力することができます。

このコマンド・プロシージャを使用すると、LINK コマンドに対する修飾子を入力できます。

```
$! This command procedure contains command  
$! qualifiers for the LINK command.  
$!  
/DEBUG/SYMBOL_TABLE/MAP/FULL/CROSS_REFERENCE
```

このコマンド行は、DEFLINK.COM に指定された修飾子を使用して、SYNAPSE.OBJ という名前のオブジェクトをリンクします。

```
$ LINK SYNAPSE@DEFLINK
```

このコマンド・プロシージャは、DCL コマンドに対するパラメータ CHAP1.TXT、CHAP2.TXT、CHAP3.TXT を入力するために使用できます。

```
$! PARAM.COM  
$! This command procedure contains a list of  
$! parameters that can be used with commands.  
$!  
CHAP1, CHAP2, CHAP3
```

このコマンド行は、パラメータ・リストのかわりにコマンド・プロシージャ PARAM を指定します。次の例では、パラメータは PARAM.COM に指定されているファイル名です。

```
$ DIRECTORY/SIZE @PARAM
```

注意

実行プロシージャ・コマンド(@)を実行すると、指定したファイル全体が DCL のコマンド入力先とされます。

13.6.3.1 制限事項

コマンド・プロシージャを実行する場合には、次の制限事項があります。

- コマンド・プロシージャの先頭に修飾子名が指定されている場合には、プロシージャ実行コマンド(@)の前にスペースを指定できない。

- コマンド・プロシージャの先頭にパラメータが指定されている場合には、プロシージャ実行コマンド(@)の前にスペースを指定できない。

13.6.4 会話形式でのコマンド・プロシージャの実行

会話形式でコマンド・プロシージャを実行するには、コマンド・プロシージャ名の前にプロシージャ実行コマンド(@)タイプして入力します。

このコマンドは、WORKDISK: ディスクの[MAINT.PROCEDURES]ディレクトリのプロシージャ SETD.COM を実行します。

```
$ @WORKDISK:[MAINT.PROCEDURES]SETD Return
```

長いコマンド行を表すためにシンボル名を定義できます。定義したシンボルを使用して、コマンド・プロシージャを実行できます。

シンボルを使用して上記例のコマンド・プロシージャを実行するには、ログイン・コマンド・プロシージャに次のコマンド行を指定しておきます。

```
$ SETD == "@WORKDISK:[MAINT.PROCEDURES]SETD"
```

ログイン後、次のようにシンボル名を入力すれば、SETD.COM プロシージャを実行することができます。

```
$ SETD Return
```

省略時の設定では、コマンド・プロシージャを会話形式で実行する場合、オペレーティング・システムは出力をターミナルに表示します。しかし、実行コマンドに対して/OUTPUT 修飾子を指定すれば、出力をファイルにリダイレクトできます。

コマンド・プロシージャの出力をファイルにリダイレクトする場合には、プロシージャはエラー・メッセージをターミナルに送信し、さらに、出力を受信しているファイルにもエラー・メッセージを送信します。

このコマンド・プロシージャは、SETD.COM からの出力を、ターミナルではなく、ファイル RESULTS.TXT に書き込みます。

```
$ @SETD/OUTPUT=RESULTS.TXT
```

/OUTPUT 修飾子は、コマンド・プロシージャ名のすぐ後に指定しなければならず、修飾子とプロシージャ名の間にスペースを指定することはできません。この制限事項が守られないと、DCL は修飾子を、プロシージャに渡すパラメータとして解釈します。

13.6.5 バッチ・ジョブとしてのコマンド・プロシージャの実行

長い処理時間を必要とするコマンド・プロシージャを使用する場合 (たとえば、大きいプログラムをコンパイルしたりアセンブルする場合) には、これらのプロシージャをバッチ・ジョブとして実行すれば、ターミナルを引き続き会話形式で使うことができます。

コマンド・プロシージャをバッチ・モードで実行するには、DCL コマンド SUBMIT を入力して、コマンド・プロシージャをバッチ・キュー (実行を待っているバッチ・ジョブのリスト) に登録します。ジョブをキューに登録すると、ジョブは省略時のバッチ・ジョブ SYS\$BATCH に登録され、実行を待っているジョブのキューの最後に追加されます。前に登録されていたジョブが終了すると、新たに登録したジョブが実行されます。OpenVMS システムでは、同時に実行できるバッチ・ジョブの数は、システム管理者がバッチ・ジョブを作成するときに指定します。

次の例では、JOB1.COM という名前のコマンド・プロシージャの実行方法を示しています。SUBMIT コマンドは省略時のファイル・タイプ.COM を使用します。したがって、コマンド・プロシージャのファイル・タイプが.COM である場合には、ファイル・タイプを入力する必要はありません。

```
$ SUBMIT JOB1
Job JOB1 (queue SYS$BATCH, entry 651, started on SYS$BATCH))
```

13.6.5.1 リモート・バッチ・ジョブ

システムがネットワークの一部として接続されている場合には、コマンド・プロシージャをリモート・ノードにバッチ・ジョブとして登録できます。このようにして使用するコマンド・プロシージャでは、ローカル・ファイルの場合と同じコマンドおよび修飾子を使用して、リモート・ノードのファイルを開いたり、閉じたりする DCL コマンドや、これらのファイルのレコードを読み込んだり、書き込むコマンドを使用できます。

13.6.5.2 バッチ・ジョブの再起動

省略時の設定では、ジョブが終了するためにシステムに障害が発生した場合には、バッチ・ジョブは最初の行から再実行されます。しかし、コマンド・プロシージャで次のシンボルを使用すれば、別の場所から再起動することを指定できます。

- \$RESTART

これはグローバル・シンボルであり、今回の実行の前に少なくとも 1 回以上、バッチ・ジョブが起動されていた場合には、このシンボルの値は真になる。\$RESTART の値を指定してはならない。システムが適切な値を割り当てる。

- BATCH\$RESTART

これはグローバル・シンボルであり、シンボルの値は、SET RESTART_VALUE コマンドを使用して指定できる。

\$RESTART と BATCH\$RESTART の使用

次の操作手順は、\$RESTART シンボルと BATCH\$RESTART シンボルの使用方法を示しています。

手順	操作
1	プロシージャの可能な各開始点の先頭にラベルを指定する。
2	各セクションの最初の処理として、SET RESTART_VALUE コマンドを使用して、BATCH\$RESTART の値をラベルに等しく設定する。
3	プロシージャを開始するときに\$RESTART をテストする。
4	\$RESTART が真の場合には、BATCH\$RESTART を分岐先ラベルとして使用して、GOTO 文を実行する。

このコマンド・プロシージャは、ライブラリから多くのモジュールを取り出し、それらのモジュールを連結し、作成されたファイルをソートします。

```
$! 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
```

このコマンド・プロシージャが途中で打ち切られた場合には、BATCH\$RESTART の値に応じて、ファイルの先頭から再実行されるか、CONCATENATE_LIBRARIES というラベルの付いた文から再実行されるか、SORT_FILE というラベルの付いた文から再実行されます。多くの独立したモジュールを取り出しているため、各抽出操作は別々のセクションにすることができます。

13.6.6 ディスクおよびテープ・ボリュームでのコマンド・プロシージャの実行

これ以降の節では、ディスクおよびテープ・ボリューム上でコマンド・プロシージャを実行する方法を説明します。

13.6.6.1 個人用ディスクでの実行

SUBMIT コマンドを使用してコマンド・プロシージャをキューに登録する場合には、割り当てられているデバイス上のファイルにアクセスすることができません。しかし、/SHARE 修飾子を使用してマウントされている個人用ディスクに存在するコマンド・プロシージャは実行できます。

13.6.6.2 テープ・ボリューム上での実行

次の場合には、テープ・ボリュームに存在するコマンド・プロシージャを実行できます。

- プロシージャで他のプロシージャを起動しない場合。
- プロシージャで、GOTO コマンドの前に指定されているラベルを参照する GOTO コマンドを実行しない場合。

上記のいずれかの条件が満足される場合には、次の操作により、コマンド・プロシージャを実行できます。

手順	操作
1.	コマンド・プロシージャを共用ディスク・ボリュームにコピーする。
2.	コマンド・プロシージャを共用ディスク・ボリュームで実行する。

13.7 コマンド・プロシージャの終了と割り込み

この節で説明する方法を使用してコマンド・プロシージャを終了する場合には、コマンド・レベルについて理解しておかなければなりません。

コマンド・レベルとは、DCL レベル・インタプリタの入力ストリームです。ターミナルからコマンドを入力する場合には、コマンドはコマンド・レベル 0 で入力されます。単純な会話形式のコマンド・プロシージャ (CLEANUP.COM など) は、コマンド・レベル 1 で実行されます。プロシージャが終了し、DCL プロンプトが画面に再表示されると、コマンド・レベル 0 に戻ります。

13.7.1 終了方法

コマンド・プロシージャの実行中に、そのプロシージャを終了するには、次の 3 種類の方法を使用できます。

- コマンド・プロシージャに EXIT コマンドを指定する方法
- コマンド・プロシージャに STOP コマンドを指定する方法

- プログラムの実行中に Ctrl/Y を入力する方法

EXIT コマンドによる終了

コマンド・プロシージャは、プロシージャの終わりに到達するか、EXIT コマンドを検出すると終了します。いずれの場合も、次に高いコマンド・レベルに制御が戻ります。EXIT コマンドによって終了する場合、状態値をEXIT コマンドのパラメータとして指定すれば、その状態値を次に高いコマンド・レベルに戻すことができます。

たとえば、DCL コマンド・レベルでSUB を起動し、SUB がSUB1 を呼び出すと、次のようなアクションが生じます。

1. SUB1 を終了すると、SUB1 を呼び出した後のコマンド行でSUBに戻る。
2. SUB を終了すると、DCL コマンド・レベルに戻る。

STOP コマンドによる終了

STOP コマンドによって終了した場合には、STOP コマンドを実行するコマンド・レベルにかかわらず、制御はいつでもDCL コマンド・レベルに戻ります。

バッチ・ジョブの中でSTOP コマンドを実行すると、バッチ・ジョブが終了します。

Ctrl/Y による終了

Ctrl/Y を押して、コマンド・プロシージャを中断させてから、EXIT コマンドまたはSTOP コマンドを使用してプロシージャを終了することができます。この場合、EXIT またはSTOP どちらのコマンドを使用した場合でもDCL レベルに戻ります。

次の例では、Ctrl/Y を押すことにより、TESTALL プロシージャに割り込みがかかります。EXIT コマンドはプロシージャの処理を終了し、DCL レベルに戻ります (プロシージャに割り込みをかけた後、STOP コマンドを入力することもできます)。

```
$ @TESTALL Return
Ctrl/Y
$ EXIT Return
$
```

13.7.2 終了処理ルーチン

コマンド・プロシージャを中断させる場合、コマンド (イメージ) が終了処理ルーチンを宣言するものであると、EXIT コマンドは終了処理ルーチンに制御を渡します。ただし、STOP コマンドはこれらのルーチンを実行しません。

13.8 エラー処理

省略時の設定では、コマンド・インタプリタは、コマンドを実行してエラーまたは重大エラーが生じると EXIT コマンドを実行します。このとき、プロシージャは終了して前のコマンド・レベルに戻ります。他の重大度 (成功, 警告, 通知エラー) の場合には、コマンド・プロシージャの処理が継続されます。

コマンド・インタプリタは以上のようにしてエラー処理を行いますが、ユーザがコマンド・プロシージャの中のラベルを参照したときに、そのラベルが存在しない場合 (たとえば、GOTO ERR1 コマンドをコマンド・プロシージャ内に指定したが、ERR1 がプロシージャ内でラベルとして使用されていない場合) には、警告メッセージを出してコマンド・プロシージャは終了します。

システムがエラー処理ルーチンの一部として EXIT コマンドを出した場合は、\$STATUS の値は直前のコマンド・レベルに戻されます。このとき、コマンド・インタプリタは\$STATUS の上位桁を 1 に設定するため、状態値に関連するメッセージを再表示しません。

たとえば、次のコマンド・プロシージャ TEST.COM には、出力ファイル指定にエラーがあります。

```
$ CREATE DUMMY.DAT\  
THIS IS A TEST FILE  
$ SHOW TIME
```

このプロシージャを実行すると、CREATE コマンドが\$STATUS にエラーを戻し、対応するメッセージを表示します。このとき、コマンド・インタプリタは\$STATUS の値を調べて、エラーが生じたかどうかを判別し、EXIT コマンドを出して\$STATUS の値を戻します。CREATE コマンドはすでに 1 回メッセージを表示したため、プロシージャが終了してもエラー・メッセージは再表示されません。DCL コマンド・レベルでは、\$STATUS にエラー・メッセージが保持されていますが、上位桁は 1 に設定されています。

```
$ @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
```

13.8.1 省略時のエラー処理

次の表は、コマンド・プロシージャの実行中に、エラーが発生したり、Ctrl/Y による割り込みが発生したときに、実行される省略時の処理を示しています。これらの省略時の処理は、ON, SET [NO]ON, SET [NO]CONTROL=Y コマンドを使用して変更できます。

割り込み	省略時の処理
エラーまたは重大なエラー	プロシージャを終了し、次のコマンド・レベルに戻る。
DCL コマンド・レベルまたはコマンド・レベル 1 での Ctrl/Y	プロシージャに割り込みがかかる。他のイメージによって強制的に終了されない場合には、プロシージャを続行できる。
レベル 1 より下のコマンド・レベルでの Ctrl/Y	プロシージャは終了し、次の上位コマンド・レベルに移動する。

13.9 エラー処理のその他の方法

これ以降の節では、その他のエラー処理方法を説明します。

13.9.1 ON コマンド

ON コマンドは、特定の重大度またはそれ以上の重大度のエラーが生じたときに実行されるアクションを指定します。このようなエラーが生じた場合には、システムは次のようなアクションをとります。

- ON コマンドによって指定されたアクションを実行する。
- 指定された ON アクションの結果を示すように\$STATUS と\$SEVERITY を設定する。通常は、成功状態に設定される。
- 省略時のエラー・アクション (エラーまたは重大エラーが生じた場合には終了する) を再設定する。

ON コマンドの処理は 1 回だけ実行されます。したがって、コマンド・プロシージャが ON コマンドに指定された処理を実行した後、省略時のエラー処理はリセットされます。

ON コマンドによって指定された処理は、コマンドが実行されているコマンド・レベルの内部だけで適用されます。したがって、別のプロシージャを起動するプロシージャの内部で ON コマンドを実行した場合には、ON コマンドの処理は、ネスティングしたプロシージャに適用されません。

ON コマンドの形式は次のとおりです。

ON condition THEN [\$] command

ただし、condition は次のいずれかのキーワードです。

ON キーワード	アクション
WARNING	コマンド・プロシージャは、警告、エラー、重大エラーが生じたときに指定されたアクションを実行する。
ERROR	コマンド・プロシージャは、エラーまたは重大エラーが生じたときに指定されたアクションを実行し、警告エラーの場合には処理を継続する。
SEVERE_ERROR	コマンド・プロシージャは、重大 (回復不可能な) エラーが生じたときに指定されたアクションを実行し、警告またはエラーが生じた場合には処理を継続する。

特定の重大度レベルに対して ON コマンドの処理が設定された場合には、同じ重大度またはそれより重大度の高いエラーが発生したときに、コマンド・インタプリタは指定された処理を実行します。重大度の低いエラーが発生した場合には、コマンド・インタプリタはファイルの処理を続行します。

例: ON コマンドの使用

このコマンドを使用すれば、省略時のエラー処理を変更して、警告、エラー、または重大なエラーが発生したときに、プロシージャを終了できます。

```
$ ON WARNING THEN EXIT
```

例: エラー発生後の再開

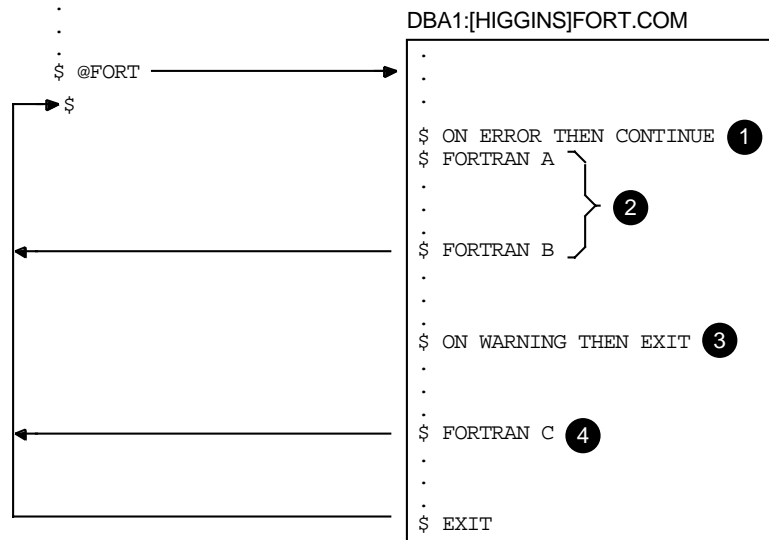
コマンド・プロシージャにこのコマンドが指定されている場合には、エラーまたは重大なエラーが発生するまで、コマンド・プロシージャは通常どおりに実行されます。

```
$ ON ERROR THEN GOTO ERR1
```

このようなエラーが発生すると、プロシージャは ERR1 から実行を再開します。\$STATUS と \$SEVERITY は正常終了に設定され、省略時のエラー処理はリセットされます。別の ON コマンドまたは SET NOON コマンドが実行される前に、2 番目のエラーが発生した場合には、プロシージャは終了し、前のコマンド・レベルに戻ります。ON コマンドで指定されたアクションは、コマンドが実行されているコマンド・レベルでのみ適用されます。したがって、あるプロシージャ内で他のプロシージャを起動するように ON コマンドを使用した場合、ON コマンドのアクションはネストしたプロシージャには及びません。

図 13-1 は、ON コマンド・アクションを図式化しています。

図 13-1 ON コマンドのアクション



ZK-0826-GE

- 1 この ON コマンドは、省略時のコマンド・アクション (警告の場合は処理を継続し、エラーまたは重大エラーの場合は終了する) を無効にする。A.FOR をコンパイル中にエラーまたは重大エラーが生じると、次のコマンドの処理に進む。
- 2 前の ON コマンドが有効な場合は、省略時のコマンド・アクションに戻るため、A.FOR と B.FOR の両方をコンパイル中にエラーまたは重大エラーが生じると、コマンド・プロシージャは終了する。
- 3 C.FOR をコンパイル中に警告、エラーまたは重大エラーが生じると、コマンド・プロシージャは終了する。
- 4 コマンドの実行前にコマンド・プロシージャが終了していなければ、そのコマンドが実行される。

付録 B のサンプル・コマンド・プロシージャ FORTUSER.COM と CALC.COM は、ON コマンドを使用してエラー処理を設定する方法を示しています。

13.10 SET NOON コマンドの使用方法

コマンド・プロシージャの中で SET NOON コマンドを使用すれば、コマンド・インタプリタがコマンドによって戻される状態をチェックしないようにできます。このコマンドは、ON コマンドを NO 状態に設定します。SET NOON コマンドを使用しても、コマンド・インタプリタは引き続き値を \$STATUS と \$SEVERITY に収めますが、エラー・チェックは行いません。エラー・チェック機能を再開させるには、SET ON コマンドまたは ON コマンドを使用します。

プロシーダでエラー・チェックが無効に設定されている場合には、コマンドまたはプログラムを実行した後、\$STATUS の値を明示的にチェックできます。

次の例では、RUN コマンドの前に SET NOON コマンドが指定されているため、プログラム TESTA のまたは TESTB がエラーを戻した場合には、コマンド・プロシーダは実行を続行します。SET ON コマンドは、コマンド・インタプリタによる省略時のエラー・チェックを復元します。

```
$ SET NOON
$ RUN TESTA
$ RUN TESTB
$ SET ON
```

次の例では、最初の IF コマンドは、\$STATUS の値が真であるかどうか(つまり、値が奇数の数値であるかどうか)を確認します。その場合には、FORTRAN コマンドは正常終了しており、LINK コマンドが実行されます。LINK コマンドを実行した後、\$STATUS が再度テストされます。\$STATUS の値が奇数である場合には、RUN コマンドが実行されます。奇数でない場合には、RUN コマンドは実行されません。SET ON コマンドは現在の ON 条件処理を復元します。つまり、SET NOON コマンドを実行する前に有効だった条件を復元します。

```
$ SET NOON
$ FORTRAN MYFILE
$ IF $STATUS THEN LINK MYFILE
$ IF $STATUS THEN RUN MYFILE
$ SET ON
```

SET ON または SET NOON コマンドは、現在のコマンド・レベル、すなわちコマンドを実行するコマンド・レベルにだけ適用されます。別のコマンド・プロシーダを呼び出すコマンド・プロシーダの中で SET NOON コマンドを使用すると、ネストされたプロシーダの中では省略時のエラー・チェック方式が適用されます。SET NOON は、DCL レベルで会話形式で入力しても意味を持ちません。

13.11 Ctrl/Y による割り込み処理

省略時の設定では、コマンド・プロシーダの実行中に Ctrl/Y を押すと、コマンド・インタプリタは、Ctrl/Y コマンド・レベルと呼ばれる特別なコマンド・レベルでコマンド入力を求めるプロンプトを出します。Ctrl/Y コマンド・レベルでは、コマンド・インタプリタ内で実行される DCL コマンドを入力した後、CONTINUE コマンドを入力すれば、コマンド・プロシーダの実行を再開できます。また、コマンド・プロシーダの実行を強制的に終了する DCL コマンドを入力すれば、プロシーダを終了できます。

この節では、ON コマンドを使用することにより、コマンド・プロシーダが Ctrl/Y による割り込みを処理する方法の変更について説明します。

13.11.1 コマンド・プロシージャの中断

会話形式で実行中のコマンド・プロシージャを中断するには、Ctrl/Y を押します。Ctrl/Y を押すと、コマンド・インタプリタは Ctrl/Y レベルと呼ばれる新しいコマンド・レベルを設定して、コマンド入力を求めるプロンプトを出します。いつ中断が生じるかは、実行中のコマンドまたはプログラムによって異なります。

- コマンドがコマンド・インタプリタ自体によって実行されている場合 (たとえば、IF、GOTO、割り当て文) には、コマンド・インタプリタが Ctrl/Y レベルでコマンドを求めるプロンプトを出す前にコマンドの実行が終了する。
- コマンドまたはプログラムが別のイメージ (すなわち、コマンド・インタプリタ以外のイメージ) の場合には、コマンドが中断されてから、コマンド・インタプリタは Ctrl/Y レベルで別のコマンドの入力を求めるプロンプトを出す。

Ctrl/Y レベルでは、コマンド・インタプリタは以前に設定されたすべてのコマンド・レベルの状態を格納するため、Ctrl/Y による中断後に正しい状態を復元できます。

プロシージャの中断後は、次のようにします。

- コマンド・インタプリタの中で実行される DCL コマンドを入力する。

DCL コマンドの中では、SET VERIFY、SHOW TIME、SHOW TRANSLATION、ASSIGN、EXAMINE、DEPOSIT、SPAWN、ATTACH です。これらのコマンドのうちの 1 つ以上を入力すると、CONTINUE コマンドでプロシージャの実行を再開できる。コマンド・インタプリタの中で実行されるコマンドのリストについては、第 14.7.2 項を参照すること。

ユーザが CONTINUE コマンドを入力すると、中断されたコマンドやプログラムから、または最も最近に完了したコマンドの後の行からコマンド・プロシージャが再開される。

- 別のイメージを実行する DCL コマンドを入力する。

ユーザが新しいイメージを起動するコマンドを入力すると、コマンド・インタプリタはコマンド・レベル 0 に戻って該当コマンドを実行する。これでコマンド・プロシージャの実行は終了する。新しいイメージを開始する前であれば、中断されたイメージによって宣言されたどの終了ハンドラでも実行できる。

- EXIT または STOP コマンドを入力してコマンド・プロシージャの実行を終了する。

EXIT コマンドを使用した場合は、中断されたイメージによって宣言された終了ハンドラを実行できるが、STOP コマンドはこれらのルーチンを実行しない。

注意

Ctrl/Y を押した後に (コマンド・レベルから明示的に、または ON ルーチンの一部として) コマンド・プロシージャを終了しない場合、次に入力されるコマンドはこのコマンド・プロシージャのコンテキストの中で解釈されます。たとえば、会話レベルで次のシンボルを定義したとします。

```
MAIL = "mail/edit=(send,reply,forward)"
```

Ctrl/Y を押して、この定義を含まないコマンド・プロシーダを中断した後、MAIL コマンドを入力してメッセージを送信しても、エディタは自動的に起動されません。

13.11.2 特権イメージの中断

特権イメージの実行を中断した場合、このイメージのコンテキストをセーブしたければ、CONTINUE、SPAWN、ATTACH のいずれかのコマンドを入力します。他のコマンドを入力すると（ユーザが生成または接続したサブプロセスの中から入力する場合を除く）、特権イメージは強制的に終了されます。

13.12 Ctrl/Y アクション・ルーチンの設定

これ以降の節では、Ctrl/Y アクション・ルーチンの設定方法を説明します。

13.12.1 ON コマンドの使用

エラー状態のときのアクションを定義する ON コマンドによって、コマンド・プロシーダの実行中に生じた Ctrl/Y による中断のアクション・ルーチンを定義する方法を指定することもできます。ユーザが指定するアクションは、省略時の Ctrl/Y アクション（Ctrl/Y コマンド・レベルでコマンド入力を求めるプロンプトを出す）を無効にします。

```
$ ON CONTROL_Y THEN EXIT
```

プロシーダがこの ON コマンドを実行した場合、この後のプロシーダの実行中に Ctrl/Y による中断が生じると、プロシーダは終了します。このとき、前のコマンド・レベルに制御が渡されます。

Ctrl/Y を押して ON CONTROL_Y を使用するプロシーダを中断した場合は、次のアクションを実行します。

- 現在実行中のコマンドがコマンド・インタプリタの中で実行されるコマンドの場合には、コマンドを完了してから Ctrl/Y アクションが実行される。
- 現在のコマンドまたはプログラムがコマンド・インタプリタ以外のイメージによって実行される場合には、イメージが強制終了されてから Ctrl/Y アクションが実行される。ただし、イメージが終了ハンドラを宣言している場合には、終了ハンドラを実行してから Ctrl/Y アクションが実行される。Ctrl/Y アクションの後はイメージ処理は継続できない。

13.12.2 Ctrl/Y の入力結果

Ctrl/Y アクションを実行しても、省略時の Ctrl/Y アクションには自動的に戻りません。Ctrl/Y アクションは、次のいずれかの条件が生じるまで有効です。

- プロシージャが終了する (Ctrl/Y を押す、EXIT または STOP コマンドを実行する、または省略時のエラー状態処理アクションの結果として)。
- 別の ON CONTROL_Y コマンドを実行する。
- プロシージャが SET NOCONTROL=Y コマンドを実行する (第 13.13 節を参照)。

Ctrl/Y アクションは各アクティブ・コマンド・レベル内で指定でき、指定されたコマンド・レベルに対してのみ影響します。

次の例に見られるように、このプロシージャが実行されると、Ctrl/Y 割り込みが生じるたびに SHOW TIME コマンドが実行されます。SHOW TIME コマンドの実行後、プロシージャは、中断されたコマンドの後のコマンドから実行を再開します。

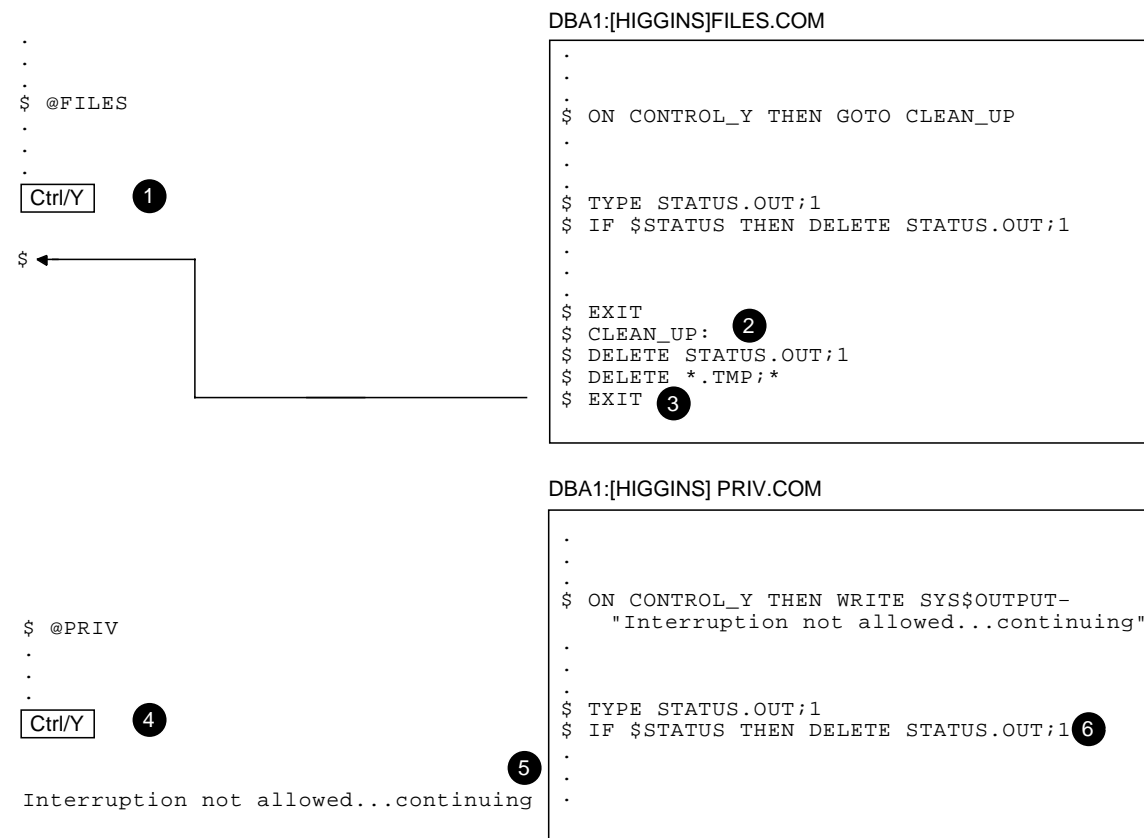
```
$ ON CONTROL_Y THEN SHOW TIME
```

コマンド・プロシージャの概要

13.12 Ctrl/Y アクション・ルーチンの設定

次の図は、Ctrl/Y による中断後の実行フローを示しています。

図 13-2 Ctrl/Y アクション後の実行フロー



ZK-0827-GE

- 1 TYPE コマンドの実行中に Ctrl/Y による割り込みが発生。
- 2 ラベル CLEAN_UP に制御が移る。
- 3 ルーチンの実行完了後にコマンド・プロシージャは終了し、会話コマンド・レベルに戻る。
- 4 TYPE コマンドの実行中に Ctrl/Y による割り込みが発生。
- 5 ON コマンドで指定されている WRITE コマンドが実行される。
- 6 中断されたコマンドの次のコマンドからコマンド・プロシージャが再開される。

次の図は、ネストされたコマンド・プロシージャの実行中に Ctrl/Y を押すとどうなるかを示しています。

図 13-3 ネストされたコマンド・プロシージャ実行中の Ctrl/Y



ZK-0828-GE

- 1 SEARCH.COM の実行中に Ctrl/Y による割り込みが生じると、`CLEAN_UP` ラベルに制御が渡される。
- 2 SUBSEARCH.COM の実行中に Ctrl/Y による割り込みが生じると、SEARCH.COM の中の `NEXT_STEP` ラベルに制御が渡される。
- 3 SUBSEARCH.COM では Ctrl/Y アクションが指定されていないため、Ctrl/Y 割り込みが生じると、プロシージャは終了して前のコマンド・レベルに戻る。
- 4 SUBSUB.COM の実行中に Ctrl/Y による割り込みが生じると、`SHOW TIME` が実行される。

13.13 Ctrl/Y による割り込みの無効化/有効化

これ以降の節では、Ctrl/Y による割り込みを有効にする方法、および無効にする方法を説明します。

13.13.1 SET NOCONTROL=Y の使用

SET NOCONTROL=Y コマンドは、Ctrl/Y による割り込みを無効にします。すなわち、コマンド・プロシージャが SET NOCONTROL=Y コマンドを実行した場合は、Ctrl/Y を押しても割り込みは生じません。

SET NOCONTROL=Y コマンドは、ON CONTROL_Y コマンドによって設定された現在の Ctrl/Y アクションも取り消します。省略時の Ctrl/Y アクションに戻すには、次の 2 つのコマンドを使用します。

```
$ SET NOCONTROL=Y  
$ SET CONTROL=Y
```

SET NOCONTROL=Y コマンドは、Ctrl/Y の処理を無効にし、現在の ON CONTROL_Y の処理を取り消します。SET CONTROL=Y コマンドは Ctrl/Y の処理を有効にします。この時点で、省略時の処理が再度有効になります。つまり、プロシージャの実行中に Ctrl/Y が押された場合には、コマンド・インタプリタは、Ctrl/Y コマンド・レベルでコマンドを要求するプロンプトを表示します。

SET NOCONTROL=Y コマンドはどのコマンド・レベルでも使用できます。SET CONROL=Y コマンドによって Ctrl/Y による割り込みを使用可能にするまで、すべてのコマンド・レベルで Ctrl/Y は無効です。

13.13.2 SET CONTROL=Y コマンドの使用

ON CONTROL_Y コマンドは、別の ON CONTROL_Y または SET NOCONTROL=Y コマンドが実行されるか、コマンド・プロシージャが終了するまで有効です。

Ctrl/Y が無効に設定されているときに、無限に終了しないループを終了するには、DCL コマンド STOP を使用して、別のターミナルからプロセスを削除しなければなりません。省略時の Ctrl/Y の処理を無効に設定した場合には、できるだけだちにリセットしてください。省略時の Ctrl/Y の処理をリセットするには、SET NOCONTROL=Y コマンドの後に SET CONTROL=Y コマンドを実行します。

このコマンド・プロシージャでは、ファイルの内容を表示しているときに Ctrl/Y を押すと、制御はラベル END_TYPE に渡されます。


```
.  
. .  
.  
$! 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  
.  
. .  
.
```

注意

ON CONTROL_Y コマンドと SET NOCONTROL=Y コマンドは特別な用途で使います。一般に、Ctrl/Y による割り込みを無効にすることはおすすめしません。Ctrl/Y が無効なときに非終了ループを終了するには、ループ・プロシージャを実行しているプロセスを別のターミナルから削除しなければなりません。

13.14 条件コードを使用しているコマンド・プロシージャでのエラーの検出

コマンド・プロシージャの中のそれぞれの DCL コマンドの実行が終了すると、コマンド・インタプリタは、コマンドの終了理由を説明する条件コードをセーブします。このコードを見ると、処理が完了したのか、情報またはエラー・メッセージが出されたのかがわかります。

コマンド・インタプリタは、コマンド・プロシージャの中のそれぞれのコマンドを実行し終わると、条件コードを調べます。特別なアクションが必要なエラーが生じた場合は、システムはそのアクションを実行します。それ以外の場合には、プロシージャの中の次のコマンドが実行されます。

13.14.1 条件コードの表示 (\$STATUS)

コマンド・インタプリタは、条件コードを 32 ビットのロングワードとして予約グローバル・シンボル \$STATUS にセーブします。\$STATUS は、次のようなシステム・メッセージ・コードの形式に従います。

- ビット 0 ~ 2 には、メッセージの重大度が指定される。
- ビット 3 ~ 15 には、メッセージ番号が指定される。
- ビット 16 ~ 27 には、メッセージを生成する機能に関連する番号が指定される。

- ビット 28 ~ 31 には、内部制御フラグが指定される。

コマンドの実行が完了した場合、\$STATUS は奇数値になります (ビット 0 ~ 2 には 1 または 3 が指定されます)。何らかの警告またはエラーが生じた場合は、\$STATUS は偶数値になります (ビット 0 ~ 2 に 0, 2, または 4 が指定されます)。コマンド・インタプリタは、\$STATUS の現在の 16 進値を保持して表示します。SHOW SYMBOL \$STATUS コマンドを入力すると、\$STATUS の ASCII 変換値が表示されます。

次の例では、ファイル名が誤って入力されています。

```
$ 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
```

13.14.2 EXIT コマンドを含む条件コード

コマンド・プロシージャが終了すると、コマンド・インタプリタは前のコマンドの条件コードを\$STATUSに戻します。条件コードは、最後に実行されたコマンドが正常終了したかどうかに関する情報を提供します。

コマンド・プロシージャで EXIT コマンドを使用する場合には、DCL が\$STATUSに割り当てる値を変更する値を指定できます。この値は状態コードと呼び、整数式として指定しなければなりません。

コマンド・プロシージャに、ネスティングしたコマンド・プロシージャが含まれており、複数のコマンド・レベルを作成する場合には、EXIT コマンドを使用して、省略時の条件コードを明示的に無効にする値を戻すことができます。

次の 2 つのコマンド・プロシージャについて調べてみましょう。

```
$! This is file A.COM
$!
$ @B
.
.
.
```

```
$! This is file B.COM
$!
$ ON WARNING THEN GOTO ERROR
.
.
.
$ ERROR:
$ EXIT 1
```

B.COM の ON コマンドは、B.COM の実行中に警告、エラー、重大なエラーのいずれかが発生した場合には、プロシージャの制御がラベル ERROR に渡されることを示しています。ここでは、条件コードは明示的に 1 に設定されており、正常終了を示します。したがって、B.COM が終了すると、エラーが発生したかどうかとは無関係に、正常終了コードが A.COM に戻されます。

13.14.3 重大度レベルの決定

\$STATUS の下位 3 ビットは、コマンドが終了する原因となった条件の重大度を示します。条件コードのこの部分は、予約されているグローバル・シンボル \$SEVERITY に格納されます。\$SEVERITY シンボルの値は 0 ~ 4 の範囲であり、各値はそれぞれ次の重大度レベルを表現します。

値	重大度
0	警告
1	正常終了
2	エラー
3	情報
4	致命的な (重大な) エラー

正常終了コードと情報コードは奇数の数値であり、警告コードとエラー・コードは偶数の数値です。

13.14.4 正常終了のテスト

次に示すように、\$SEVERITY または \$STATUS に対して論理テストを実行する IF コマンドを使用すれば、コマンドの正常終了をテストできます。

```
$ IF $SEVERITY THEN GOTO OKAY
$ IF $STATUS THEN GOTO OKAY
```

これらの IF コマンドは、\$SEVERITY および \$STATUS の値が真 (奇数) の場合には、ラベル OKAY に分岐します。\$SEVERITY と \$STATUS の現在の値が奇数の場合には、コマンドまたはプログラムは正常終了しています。コマンドまたはプログラムが正常終了しなかった場合には、\$SEVERITY と \$STATUS の値は偶数です。したがって、IF 式は偽になります。

条件が真であるかどうかをテストするかわりに、偽であるかどうかをテストすることもできます。次の例を参照してください。

```
$ IF .NOT. $STATUS THEN ...
```

コマンド・インタプリタは条件コードの重大度レベルを使用して、ON コマンドによって定義される処理を実行するかどうかを判断します。第 13.9 節の説明を参照してください。

13.15 \$STATUS を設定しないコマンドの使用

大部分の DCL コマンドは、正常終了したときに状態値とエラー・メッセージを生成するシステム・ユーティリティを起動します。しかし、正常終了したときに、\$STATUS と \$SEVERITY の値を変更しないコマンドがいくつかあります。これらのコマンドは次のとおりです。

CONTINUE	DECK	DEPOSIT
EOD	EXAMINE	GOTO
IF	RECALL	SET SYMBOL/SCOPE
SHOW STATUS	SHOW SYMBOL	STOP
WAIT		

これらのコマンドを正しく実行できなかった場合には、条件コードは \$STATUS に格納され、重大度レベルは \$SEVERITY に格納されます。

13.16 ログイン・コマンド・プロシージャ

ログイン・コマンド・プロシージャとは、ユーザがログインするたびに、オペレーティング・システムが自動的に実行するコマンド・プロシージャです。また、ユーザがキューに登録した各バッチ・ジョブの先頭で、システムはこのプロシージャを実行します。

ログイン・コマンド・プロシージャには、次の 2 種類があります。

- システム全体で有効な (またはグループで定義された) プロシージャ
- 個人用プロシージャ

13.16.1 システム全体のログイン・コマンド・プロシージャ

システム全体で有効なログイン・コマンド・プロシージャには、次の特徴があります。

- 個人用ログイン・コマンド・プロシージャより前に実行される。

- ・ システム全体で有効なログイン・コマンド・プロシージャが終了すると、制御は個人用ログイン・コマンド・プロシージャに渡される。
- ・ このようなコマンド・プロシージャを使用すると、システム管理者は、ユーザがログインするときに特定のコマンドを必ず実行するように設定できる。

システムまたはグループ・ログイン・コマンド・プロシージャを設定する場合、システム管理者は、論理名 SYSSYLOGIN を該当するログイン・コマンド・プロシージャに定義します。このログイン・コマンド・プロシージャは、すべてのシステム・ユーザに対して適用するように指定することも、特定のユーザ・グループに対してのみ適用されるように指定することもできます。

13.16.2 パーソナル・ログイン・コマンド・プロシージャ

ユーザは、ログインするたびに同じコマンドを実行するパーソナル・ログイン・コマンド・プロシージャを作成できます。

ログイン・コマンド・プロシージャのファイル指定は、システム管理者が割り当てます。ほとんどの場合、ログイン・コマンド・プロシージャは LOGIN.COM という名前にします。システム管理者が特に指定しない限り、ログイン・コマンド・プロシージャを LOGIN.COM という名前にします。

以下は LOGIN.COM プロシージャの例です。

```
$IF F$MODE() .NES. "INTERACTIVE" THEN EXIT
$SET TERMINAL/INSERT
$DIR ::= DIR/DATE/SIZE
$EDIT ::= EDIT/EDT
$EXIT
```

13.16.3 専用アカウントのログイン・コマンド・プロシージャ

システム管理者は、特別なコマンド・プロシージャの名前をアカウントの LGICMD フィールドに置くことによって、専用のアカウントを設定できます。専用のアカウントにログインした場合に実行できるのは、コマンド・プロシージャでそのアカウントに指定された機能だけで、完全な DCL コマンド・セットは実行できません。専用アカウントについての詳細は、『OpenVMS システム管理者マニュアル』を参照してください。

13.17 拡張ファイル指定と (Extended File Specifications) 解析スタイル

特定のファイル名解析スタイルを必要とするコマンド・プロシージャでは、そのコマンド・プロシージャ内にコマンドを入れて解析スタイルを切り替えることができます。次のコマンド・プロシージャは、現在の解析スタイルを保存し、解析スタイルを

TRADITIONAL に設定し、コマンド (未指定) を実行して、保存された解析スタイルを復元します。

```
$ original_style= f$getjpi("", "parse_style_perm")
$ SET PROCESS/PARSE_STYLE=TRADITIONAL
.
.
.
$ SET PROCESS/PARSE_STYLE='original_style'
```

最初のコマンドにより、'original_style' が現在の解析スタイルになります。2 番目のコマンドにより、解析スタイルが TRADITIONAL に設定されます。最後のコマンドにより、解析スタイルが元のスタイルに再設定されます。

13.18 DCL コマンド・パラメータでの拡張ファイル名の使用

ファイル名をパラメータとして使用するコマンド・プロシージャは、ODS-5 環境では異なった結果を生成する場合があります。

解析スタイルは TRADITIONAL から EXTENDED に変更することができます。この項では、EXTENDED に変更した場合に影響を受ける可能性のある、次の分野について説明します。

- コマンド・プロシージャのファイル指定
- 大文字と小文字の区別および \$FILE
- アンパサンドと一重引用符の置換

解析スタイルの切り替えについての詳細は、第 5.3 節を参照してください。

13.18.1 コマンド・プロシージャのファイル指定

インダイレクト・コマンド・プロシージャが使用されている場合には、一部のプロシージャの引数を引用符で囲む必要がある場合があります。

次の例は、同じコマンド・ファイル SS.COM を使用した場合の、TRADITIONAL 解析スタイルと EXTENDED 解析スタイルの出力の違いを示しています。

```
$ 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
```

- 解析スタイルを TRADITIONAL に設定し、コマンド・ファイル SS.COM を実行すると、出力は次のようになる。

```
$ set process/parse_style=traditional
$ @ss ^ parg2 parg3
p1 = ^
p2 = PARG2
p3 = PARG3
```

サーカンフレックス (^) は、最初の引数であり (エスケープ文字ではない)、プロシージャ引数 p2 および p3 の大文字と小文字の区別が保存されていないことに注意してください。

- 解析スタイルを EXTENDED に設定し、同じコマンド・プロシージャを実行すると、出力は次のようになる。

```
$ set process/parse_style=extended
$ @ss ^ parg2 parg3
p1 = ^ PARG2
p2 = PARG3
```

コマンド・プロシージャでサーカンフレックス (^) が、スペースを引数の区切り文字としてではなくリテラル文字として識別するエスケープ文字として認識されていることと、"^ PARG2"が最初の引数であることに注意してください。大文字と小文字の区別は保存されません。

- サーカンフレックス (^) に引用符を追加すると、結果は次のようになる。

```
$ @ss "^" parg2 parg3
p1 = ^
p2 = PARG2
p3 = PARG3
```

サーカンフレックス (^) は引用文字列の中にあるため、エスケープ文字としては処理されません。

- 引数 p3 に引用符を追加すると、結果は次のようになる。

```
$ @ss "^" parg2 "parg3"
p1 = ^
p2 = PARG2
p3 = parg3
```

プロシージャ引数 p3 の大文字と小文字の区別が保存されていることに注意してください。

- 解析スタイルを TRADITIONAL に設定すると、次のコマンドではサーカンフレックス (^) と、parg2 および parg3 の文字列とが、プロシージャ引数として処理され、コマンド・プロシージャを実行すると、結果は次のようになる。

```
$ set process/parse_style=traditional
$ @ss^ parg2 parg3
p1 = ^
p2 = PARG2
p3 = PARG3
```

- 解析スタイルを EXTENDED に設定すると、サーカンフレックス (^) は、スペースをリテラル文字として識別するエスケープ文字として処理される。DCL は、ファイル "SS^_PARG2.COM" を探し、次の例で示されているエラーが生成される。

```
$ set process/parse_style=extended
$ @ss^ parg2 parg3
-RMS-E-FNF, file not found
```

13.18.2 大文字と小文字の区別の保存と\$FILE

DCL は、ファイル指定の大文字と小文字の区別を保存しようとします。実際にファイル指定の大文字と小文字の区別が保存されるのは、Command Definition Utility (CDU) を使用して定義されたコマンドに限られます。DCL は、\$FILE 解析タイプを持つコマンド定義ファイル (.CLD) の中で定義されている項目の大文字と小文字の区別を保存します。

詳細は、『OpenVMS Command Definition, Librarian, and Message Utilities Manual』を参照してください。

13.18.3 アンパサンドと一重引用符の置換

一重引用符ではなくアンパサンド (&) による置換を使用して、従来型の解析の際に大文字と小文字の区別を保存することができます。

次の従来型の解析の例は、文字列の大文字と小文字の区別を変更する一連のコマンドを示しています。

```
$ 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)
```

アンパサンド (&) を使用することにより、変数 x に割り当てられた文字列の大文字と小文字の区別が保存されていることに注意してください。

一重引用符による置換は、コマンド行が大文字に設定される前に実行され、アンパサンドによる置換は、コマンド行が大文字に設定された後で実行されます。

次の拡張解析機能の例は、同じ一連のコマンドを示しています。


```
$ 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)
```

変数 `y` に割り当てられた文字列はどちらも小文字で返されていることに注意してください。これは、`DEFINE` コマンドが、大文字と小文字の区別を保存する `$FILE` を使用しているために発生します。

このような特徴を持つことから、アンパサンドによる置換は、解析スタイルが `TRADITIONAL` に設定されている場合でも `EXTENDED` ファイル名を指定するために使用することができます。次に例を示します。

```
$ 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
```

注意

アンパサンドによる置換は、フォーリン・コマンドには使用できません。

DCL での拡張プログラミング

拡張 DCL プログラミングとは、複雑なコマンド・プロシージャと DCL コマンドの PIPE コマンドを使用することです。この章は、第 13 章を読み、DCL でのプログラミングについての基本的な知識があり、高度なプログラミング方式を学習したい方を対象にしています。

複雑なコマンド・プロシージャは、プログラムと同様の機能を実行できます。コマンド・プロシージャでは、変数入力を使用でき、特定の条件が真のときにだけプロシージャのセクションを実行でき、サブルーチンを実行したり、他のコマンド・プロシージャを起動することができます。

DCL の PIPE コマンドも、プログラムと同様の機能を実行できます。たとえば PIPE コマンドを使用すると、同一 DCL コマンド行から、次に示す 1 つまたは複数の操作を実行できます。

- パイプライン処理 (コマンドのシーケンス)
- 入出力のリダイレクト
- 複数の条件コマンドの実行
- バックグラウンド処理

本章では、次のことについて説明します。

- コマンド・プロシージャ入力の実行
- ネスティングしたコマンド・プロシージャにデータを渡すためのパラメータの使用
- コマンド・プロシージャ出力の実行
- ファイルの読み込みと書き込み (ファイル入出力)
- ファイル入出力エラーの処理
- 実行の流れを制御する方法
- 新しいコマンド・レベルの作成
- CASE 文の書き方
- PIPE コマンドの使用

14.1 コマンド・プロシージャ入力の実行

コマンド・プロシージャでは、ユーザが提供するデータを必要とすることがよくあります。このデータ、つまり入力は会話形式で入手でき（第 13 章を参照）、非会話形式で入手することもできます。本章では、非会話型入力方式について説明し、第 13 章で説明していない会話型入力方式についても説明します。

コマンド・プロシージャを実行するたびに、同じデータを使用できます。同じデータを使用するには、コマンド・プロシージャでデータを必要とするコマンドの後のデータ行にデータを指定します。

次のコマンド・プロシージャは、CENSUS.EXE コマンド・プロシージャを実行します。CENSUS.EXE はプロシージャが実行されるたびに 1993、1994、1995 のデータを読み込みます。

```
$ ! CENSUS.COM
$ !
$ RUN CENSUS
1993
1994
1995
$ EXIT
```

14.1.1 コマンド・プロシージャにデータを登録する場合の制限事項

DCL は、データ行に指定されたテキストをコマンド・プロシージャに直接渡します。したがって、次に示すように、変換しなければならないデータは処理されません。

- シンボル
- 論理名
- 数式

14.1.2 その他のデータ入力方式

この後の節では、次に示すように、コマンド・プロシージャに対する入力データを入手するための別の方法について説明します。

- データを渡すためのパラメータの使用
- バッチ・ジョブにデータを渡すためのパラメータの使用
- ネスティングしたコマンド・プロシージャにデータを渡すためのパラメータの使用
- データを要求するプロンプトを表示するための READ、INQUIRE コマンドの使用
- データを入手するための SYS\$INPUT 論理名の使用

14.2 データを渡すためのパラメータの使用

コマンド・プロシージャにデータとしてパラメータを渡す場合には、次のガイドラインに従ってください。

- パラメータはコマンド・プロシージャのファイル指定の後に指定する。
- 1つのコマンド・プロシージャに対して最大8つのパラメータを渡すことができる。
- 渡すパラメータ値の数が8未満の場合には、値が渡されなかったシンボルにはヌル値が割り当てられる。ヌル値とは空文字列であり、引用符(" ")で表現される。
- パラメータは1つ以上のスペースまたはタブで区切る。

DCL はローカル・シンボル P1 ~ P8 を使用して、パラメータをコマンド・プロシージャに渡します。P1 には最初のパラメータ値が割り当てられ、P2 には2番目のパラメータ値、P3 には3番目のパラメータ値が割り当てられます。以下も同様です。たとえば、次のコマンドは、コマンド・プロシージャ SUM.COM を起動し、8つのパラメータをプロシージャに渡します。

```
$ @SUM 34 52 664 89 2 72 87 3
```

14.2.1 整数としてのパラメータの指定

パラメータとして整数を指定した場合には、それは文字列に変換されます。次の例では、P1 は文字列値 24 であり、P2 は文字列値 25 です。

```
$ @ADDER 24 25
```

シンボル P1 ~ P8 は、整数式と文字列式の両方で使用できます。DCL は必要な変換を自動的に実行します。

14.2.2 文字列としてのパラメータの指定

文字列内のスペース、タブ、小文字を保存するには、文字列の前後を引用符(" ")で囲みます。次の例を参照してください。

```
$ @DATA "Paul Cramer"
```

次の例では、P1 は Paul Cramer であり、P2 はヌルです。引用符を省略した場合には、各文字列は独立したパラメータとして渡されます。次の例を参照してください。

```
$ @DATA Paul Cramer
```

この例では、文字列 Paul と Cramer が大文字に変換され、P1 は PAUL に、P2 は CRAMER になります。

もう 1 つの例として、次のコマンドで DATA.COM を起動した場合を考えます。

```
$ @DATA "Paul Cramer" 24 "(555) 111-1111")
```

P1 ~ P8 は DATA.COM の中では次のように定義されています。

```
P1 = Paul Cramer  
P2 = 24  
P3 = (555) 111-1111  
P4-P8 = null
```

14.2.3 シンボルとしてのパラメータの指定

シンボル値を渡すには、シンボルの前後を一重引用符で囲みます。シンボル値の内部でスペース、タブ、小文字を保存するには、値を 3 組の引用符で囲みます。文字列の一部として引用符を使用する場合には、3 組の引用符を使用しなければなりません。

別の方法として、テキストを引用符で囲み、シンボルが表示されるときに、シンボルの前に 2 つの一重引用符を指定し、シンボルの後に 1 つの一重引用符を指定する方法があります。

次の例では、P1 は Paul であり、P2 は Cramer です。シンボルをコマンド・プロシージャに渡すときに、DCL は引用符を削除します。

```
$ NAME = "Paul Cramer"  
$ @DATA 'NAME'
```

次の例では、P1 は“Paul Cramer”であり、P2 はヌルです。

```
$ NEW_NAME = ""Paul Cramer""  
$ @DATA 'NEW_NAME'
```

次の例では、P1 は Paul Cramer に変換されます。

```
$ ! DATA.COM  
$ @NAME "'P1'"
```

14.2.4 ヌル値としてのパラメータの指定

空のパラメータを渡すには、コマンド文字列の位置を表す一組の二重引用符を使用します。次の例では、DATA.COM に渡される最初のパラメータは空のパラメータです。

```
$ @DATA "" "Paul Cramer"
```

この例では、P1 が空値で、P2 が Paul Cramer です。

14.3 バッチ・ジョブにデータを渡すためのパラメータの使用

バッチ・モードで実行されるコマンド・プロシージャにパラメータを渡すには、SUBMIT コマンドの修飾子/PARAMETERS を使用します。

1 つの SUBMIT コマンドを使用して、複数のコマンド・プロシージャを実行する場合には、指定したパラメータは、バッチ・ジョブ内の各コマンド・プロシージャに対して使用されます。

次の例では、コマンドは、3 つのパラメータをコマンド・プロシージャ ASK.COM と GO.COM に渡します。これらのコマンド・プロシージャはバッチ・ジョブとして実行されます。

```
$ SUBMIT/PARAMETERS=(TODAY,TOMORROW,YESTERDAY) ASK.COM, GO.COM)
```

次の例では、SUBMIT コマンドは、2 つのパラメータをコマンド・プロシージャ LIBRARY.COM と SORT.COM に渡します。

```
$ SUBMIT-  
_$/PARAMETERS=(DISK:[ACCOUNT.BILLS]DATA.DAT,DISK:[ACCOUNT]NAME.DAT) -  
_ $ LIBRARY.COM, SORT.COM
```

ユーザがログインして、各コマンド・プロシージャを実行したかのように、バッチ・ジョブが実行されます。この SUBMIT コマンドが実行するバッチ・ジョブは、ユーザのアカウントにログインし、ユーザのログイン・コマンド・プロシージャを実行し、その後、次のコマンドを実行します。

```
$ @LIBRARY DISK:[ACCOUNT.BILLS]DATA.DAT DISK:[ACCOUNT]NAME.DAT)  
$ @SORT DISK:[ACCOUNT.BILLS]DATA.DAT DISK:[ACCOUNT]NAME.DAT)
```

データをコマンド・プロシージャの中に指定する方法、SYSSINPUT をファイルとして定義する方法でも、データをバッチ・ジョブに渡すことができます。指定されたパラメータは、バッチ・ジョブのそれぞれのコマンド・プロシージャで使用されます。

14.4 ネスティングしたコマンド・プロシージャにデータを渡すためのパラメータの使用

ネスティングしたコマンド・プロシージャに最大 8 つのパラメータを渡すことができます。ネスティングしたプロシージャでのローカル・シンボル P1 ~ P8 は、起動するプロシージャのローカル・シンボル P1 ~ P8 と関連づけられません。

次の例では、DATA.COM は、ネスティングしたコマンド・プロシージャ NAME.COM を起動します。

```
$ ! DATA.COM  
$ @NAME 'P1' Joe Cooper
```

DATA.COM の P1 が Paul Cramer という文字列である場合には、この文字列引用符が含まれていないため、2 つのパラメータとして NAME.COM に渡されます。NAME.COM では、P1 ~ P8 は次のように定義されます。

```
P1 = PAUL
P2 = CRAMER
P3 = JOE
P4 = COOPER
P5-P8 = null
```

DATA.COM の P1 が "Paul Cramer" (引用符が含まれている) の場合には、次に示すように、3 組の引用符で P1 を囲むことにより、1 つのパラメータとして値を NAME.COM に渡すことができます。

```
$ ! DATA.COM
$ QUOTE = ""
$ P1 = QUOTE + P1 + QUOTE
$ @NAME 'P1' "Joe Cooper"
```

この例では、コマンド・プロシージャ NAME.COM で P1 は Paul Cramer であり、P2 は Joe Cooper です。

14.5 データを要求するプロンプトの表示

コマンド・プロシージャのデータを会話形式で入手するには、INQUIRE コマンド (第 13 章を参照) または READ コマンドを使用できます。どちらのコマンドも、入力を要求するプロンプトを表示し、応答をシンボルに割り当てます。

READ コマンドと INQUIRE コマンドの相違点は次のとおりです。

INQUIRE コマンド	READ コマンド
値を要求するプロンプト	値を要求するプロンプト
ターミナルから値を読み込む	最初のパラメータによって指定されたソースから値を読み込む
値をシンボルに割り当てる	2 番目のパラメータとして指定されたシンボルに値を割り当てる。

READ コマンドは、プロンプトに対する応答としてターミナルに入力されたすべての文字を、正確な文字列値として受け付けます (大文字と小文字の区別、スペース、タブは保存されます)。/PROMPT 修飾子を省略した場合には、READ コマンドは省略時のプロンプトとして Data: を表示します。

入力されたパラメータを受け付けることができ、また、必須パラメータが指定されなかったときに、ユーザ入力を要求するプロンプトを表示するコマンド・プロシージャを作成することもできます。

次の例では、コマンドは `Filename:` というプロンプトをターミナルに表示し、論理名 `SYSS$COMMAND` (省略時の設定ではターミナル) によって指定されるソースから応答を読み込み、応答をシンボル `FILE` に割り当てます。

```
$ READ/PROMPT="Filename: " SYSS$COMMAND FILE
```

次の例では、プロシージャを起動したときに、ファイル名が指定されなかった場合には、ファイル名を要求するプロンプトが表示されます。

```
$ ! 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
```

注意

コマンド・プロシージャをキューに登録して、バッチ・ジョブとして実行する場合、DCL は、`INQUIRE` コマンドに続くデータ行から `INQUIRE` コマンドに指定されたシンボルの値を読み込みます。データ行を指定しない場合には、シンボルには空値が割り当てられます。

14.6 データを入手するための `SYSS$INPUT` 論理名の使用

コマンド、ユーティリティ、その他のシステム・イメージは、省略時の入力ストリームである論理名 `SYSS$INPUT` に指定されるソースから入力を取得します。コマンド・プロシージャの中では、`SYSS$INPUT` はコマンド・プロシージャ・ファイルとして定義され、データを必要とするコマンドやイメージはこのファイルの中でデータ行を検索します。ただし、`SYSS$INPUT` を再定義すれば、ターミナルや別の入力ファイルからデータを得ることができます。

14.6.1 ターミナルとしての `SYSS$INPUT` の再定義

`SYSS$INPUT` をターミナルとして再定義できます。このようにすると、コマンド・プロシージャから呼び出されたイメージは、コマンド・プロシージャ内のデータ行からではなく、会話形式で入力入手できます。

コマンド・プロシージャで会話型入力を必要とする DCL コマンドやユーティリティを使用する場合には、`SYSS$INPUT` をターミナルとして再定義しなければなりません。

たとえば、次のコマンド・プロシージャは、入力を会話形式で `CENSUS.EXE` イメージに提供できるようにします。

```
$ ! Execute CENSUS getting data from the terminal
$ DEFINE/USER_MODE SYSS$INPUT SYSS$COMMAND
$ RUN CENSUS
$ EXIT
```

DEFINE/USER_MODE コマンドは、CENSUS.EXE を実行しながら、SYSS\$INPUT を一時的に再定義するため、CENSUS.EXE はターミナルから入力を受け入れます。CENSUS.EXE が終了すると、SYSS\$INPUT は元の定義 (コマンド・プロシージャ・ファイル) に戻されます。

たとえば、次のコマンド・プロシージャは EVE エディタを使用します。

```
$ ! 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 SYSS$INPUT SYSS$COMMAND
$     EDIT/TPU 'FILE'
$     GOTO EDIT_LOOP
```

このコマンド・プロシージャは、ユーザが Return キーを押してループを終了するまで、ファイル名を求めるプロンプトを出します。ファイル名を入力すると、プロシージャは自動的に EVE エディタを起動してファイルを編集します。エディタの実行中、SYSS\$INPUT はターミナルとして定義されるので、編集内容を会話形式で入力できます。

14.6.2 SYSS\$INPUT を個別ファイルとして定義する場合

SYSS\$INPUT をファイルとして定義すれば、コマンド・プロシージャはファイルから入力を得ることができます。コマンド・プロシージャは、データ行のテキストをコマンドまたはイメージに直接渡すことに注意してください。DCL はデータ行は処理しません。データ行に DCL シンボルや式を指定しても、DCL は値をシンボルに置換しませんし、式を評価しません。データ行で感嘆符を使用すると、データを渡す先のイメージが感嘆符(!)を処理します。

データ・ファイルの名前を SYSS\$INPUT として指定すれば、コマンド・プロシージャ・ファイルにプログラムを登録できます。この操作を実行すると、コンパイラは別のファイルではなく、コマンド・プロシージャからプログラムを読み込みます。

次の例は、FORTRAN コマンドの後にプログラムの文が続くコマンド・プロシージャを示しています。

```
$ FORTRAN/OBJECT=TESTER/LIST=TESTER SYSS$INPUT
C THIS IS A TEST PROGRAM
  A = 1
  B = 2
  STOP
  END
$ PRINT TESTER.LIS
$ EXIT
```

FORTTRAN コマンドは、論理名 SYSS\$INPUT を使用してコンパイルするファイルを識別します。SYSS\$INPUT はコマンド・プロシージャとして定義されているため、FORTTRAN コンパイラは、FORTTRAN コマンドに続く文を (始めにドル記号が付く次の行まで) コンパイルします。コンパイルが終了すると、2つの出力ファイル TESTER.OBJ と TESTER.LIS が作成されます。このとき、PRINT コマンドは、リスト・ファイルを印刷します。

14.7 コマンド・プロシージャ出力の実行

データやエラー・メッセージ、コマンド行のチェックなど、コマンド・プロシージャからの出力は、ターミナルまたは他のファイルに送信できます。この節では、次の出力方法について説明します。

- データの表示
- コマンドとイメージの出力先を切り換える
- コマンド・プロシージャからデータを戻す
- エラー・メッセージの出力先を切り換える

14.7.1 データの表示

シンボルやレキシカル関数を含むデータを書き込むには、WRITE コマンドを使用します。データを二重引用符(" ")で囲む場合を除いて、WRITE コマンドは自動的にシンボル置換を実行します。

次の例では、SYSS\$INPUT はデータ・ファイルとして指定されています。TYPE コマンドは、その後続くデータ行からデータを読み込み、データ行をターミナルに表示します。

```
$ ! Using TYPE to display lines
$ TYPE SYSS$INPUT
REPORT BY MARY JONES
PREPARED APRIL 15, 2002
SUBJECT: Analysis of Tax Deductions for 2002
.
.
.
$ EXIT
```

シンボルやレキシカル関数を含むデータを書き込むには、WRITE コマンドを使用します。データを二重引用符(" ")で囲む場合を除いて、WRITE コマンドは自動的にシンボル置換を実行します。

WRITE コマンドを使用して、文字列をリテラル・テキストとして表示するには、二重引用符(" ")で囲みます。たとえば、次のようになります。

```
$ WRITE SYS$OUTPUT "Two files are written."  
Two files are written.
```

文字列の中に二重引用符を指定するには、2組の二重引用符("" "")を使用します。たとえば、次のようになります。

```
$ WRITE SYS$OUTPUT "Summary of "Q & A" Session"  
Summary of "Q & A" Session
```

テキスト行を2行以上に継続するには、2つの文字列をプラス記号(+)とハイフン(-)で連結します。たとえば、次のようになります。

```
$ WRITE SYS$OUTPUT "Report by Mary Jones" + -  
" Prepared April 15, 2002"  
Report by Mary Jones Prepared April 15, 2002
```

WRITE コマンドは、シンボル置換を自動的に実行し、シンボルの値を表示します。文字列内でシンボル置換を強制的に実行するには、シンボルを一重引用符で囲みます。たとえば、次のようになります。

```
$ AFILE = "STAT1.DAT"  
$ BFILE = "STAT2.DAT"  
$ WRITE SYS$OUTPUT "'AFILE' and 'BFILE' ready."  
STAT1.DAT and STAT2.DAT ready.
```

この例では、STAT1.DAT が AFILE シンボルの変換結果で、STAT2.DAT が BFILE シンボルの変換結果になっています。

14.7.2 コマンドとイメージからの出力先の切り換え

コマンド、ユーティリティ、その他のシステム・イメージは、論理名 SYS\$OUTPUT によって指定されるソースに出力を書き込みます。省略時の設定では、SYS\$OUTPUT はターミナルに定義されます。ただし、次のいずれかの方法を使用すれば、出力先を切り換えることができます。

- コマンドを起動するときに /OUTPUT 修飾子を使用する。/OUTPUT 修飾子を受け入れる DCL コマンドには、ACCOUNTING, CALL, DIRECTORY, HELP, LIBRARY, RUN (プロセス), SPAWN, TYPE コマンドがある。
- DEFINE/USER_MODE コマンドを使用して、SYS\$OUTPUT をファイルとして一時的に再定義する。

- コマンドからの出力を停止するために、SYS\$OUTPUT をヌル・デバイスとして一時的に定義する (DEFINE/USER_MODE コマンドを使用する)。

次の例では、コマンド・プロシージャは、SHOW USERS コマンドからの出力をファイルに切り換えます。SYS\$OUTPUT の新しい定義は、SHOW USERS コマンドの実行に対してだけ有効になります。

```
$ 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
```

次の例では、SYS\$OUTPUT はヌル・デバイス (NL:) として定義されています。

```
$ DEFINE/USER_MODE SYS$OUTPUT NL:
$ APPEND NEW_DATA.DAT STATS.DAT
.
.
.
```

/USER_MODE 修飾子は、次のイメージが完了するまでの間だけ有効な一時的な論理名割り当てを作成します。このコマンドを実行すると、SYS\$OUTPUT は省略時の定義 (ターミナル) に戻ります。

DEFINE/USER_MODE コマンドを使用しても、コマンド・インタプリタの中で実行される DCL コマンドの出力先を切り換えることはできません。この場合には、DEFINE コマンドを使用して SYS\$OUTPUT を再定義し、処理が終了したら、DEASSIGN コマンドを使用して定義を削除します。

次の表は、コマンド・インタプリタの内部で実行される DCL コマンドを示しています。

=	ALLOCATE	ASSIGN
ATTACH	CALL	CANCEL
CLOSE	CONNECT	CONTINUE
CREATE/LOGICAL_NAME_TABLE	DEALLOCATE	DEASSIGN
DEBUG	DECK	DEFINE
DEFINE/KEY	DELETE/SYMBOL	DISCONNECT
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

次の例では、SHOW TIME コマンドからの出力をファイル TIME.DAT に切り換えるために使用されるコマンドを示しています。SYS\$OUTPUT の割り当てを解除した後、省略時の定義 (ターミナル) に戻ります。

```
$ DEFINE SYS$OUTPUT TIME.DAT
$ SHOW TIME
$ DEASSIGN SYS$OUTPUT
```

14.7.3 コマンド・プロシージャからのデータを戻す

グローバル・シンボルと論理名は、コマンド・プロシージャのデータを呼び出し側プロシージャまたは DCL コマンド・レベルに戻します。グローバル・シンボルや論理名は、どのコマンド・レベルでも読み込めます。論理名は、ネストされたコマンド・プロシージャから呼び出し側のプロシージャにデータを戻すことができます。

次の例では、グローバル割り当て文で作成されたグローバル・シンボルを使用して、コマンド・プロシージャが値を渡す方法を示しています。

```
$ @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.COM は、コマンド・プロシージャ NAME.COM を起動し、NAME.COM に姓名を渡します。NAME.COM はグローバル・シンボル LAST_NAME に姓を格納します。NAME.COM が終了すると、DCL は DATA.COM の実行を続行し、このコマンド・プロシージャは、グローバル・シンボル LAST_NAME を指定することにより、姓を読み込みます。コマンド・プロシージャ NAME.COM は別のファイルに格納されています。この例では、わかりやすくするためにインデントを使用して示しています。

このコマンド・プロシージャでは、REPORT.COM はレポートのファイル名を入手し、ファイル名を論理名 REPORT_FILE に等しく定義し、レポートを REPORT_FILE に書き込むプログラムを実行します。

```
$! 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
```

次の例では、コマンド・プロシージャ REPORT.COM は別のプロシージャから起動されます。呼び出し側のプロシージャは、論理名 REPORT_FILE を使用して、レポート・ファイルを参照します。

```
$! 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
```

14.7.4 エラー・メッセージの出力先の切り換え

これ以降の節では、エラー・メッセージの出力先を切り換える方法を説明します。

14.7.4.1 SYS\$ERROR の再定義

省略時の設定では、コマンド・プロシージャは、SYS\$ERROR に示されるファイルにシステム・エラー・メッセージを送信します。SYS\$ERROR を再定義すれば、エラー・メッセージを指定するファイルに書き出すこともできます。ただし、SYS\$ERROR を SYS\$OUTPUT と異なるように再定義すると (SYS\$ERROR を再定義せずに SYS\$OUTPUT を再定義すると)、標準 VMS システム・エラー表示方式を使用する DCL コマンドやイメージは、システム・エラー・メッセージとシステム重大エラー・メッセージを SYS\$ERROR と SYS\$OUTPUT の両方に送信します。したがって、ユーザはこれらのメッセージを、SYS\$ERROR の定義に示されるファイルで 1 回と、SYS\$OUTPUT に示されるファイルで 1 回の合計 2 回受信することになります。成功メッセージ、通知メッセージ、警告メッセージは、SYS\$OUTPUT に示されるファイルにだけ送信されます。

DCL コマンドからエラー・メッセージが出されないようにしたい場合は、SYS\$ERROR も SYS\$OUTPUT もターミナルに定義しないようにします。

コマンド・プロシージャから自分のイメージを実行するときに、そのイメージで SYS\$ERROR を参照する場合には、イメージはシステム・エラー・メッセージを、SYS\$ERROR によって示されるファイルにだけ送信します。SYS\$ERROR が SYS\$OUTPUT と異なる場合でも、この規則が適用されます。SYS\$ERROR と SYS\$OUTPUT が異なる場合には、標準的なシステム・エラー表示メカニズムを使用する DCL コマンドとイメージだけが、この 2 つのファイルにメッセージを送信します。

このコマンド・プロシージャはパラメータとしてディレクトリ名を受け付け、省略時の値をそのディレクトリに設定し、ディレクトリ内のファイルをパージします。システム・エラー・メッセージが出力されないようにするために、プロシージャは、SYS\$ERROR と SYS\$OUTPUT をヌル・デバイスとして一時的に定義します。

```
$ ! 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
```

14.7.4.2 システム・エラー・メッセージの禁止

システム・メッセージが出力されないようにするために、SET MESSAGE コマンドを使用することもできます。/NOFACILITY、/NOIDENTIFICATION、/NOSEVERITY、/NOTEXT のいずれかの修飾子を使用すれば、機能名、メッセージ識別、重大度レベル、メッセージ・テキストを出力しないように設定できます。

次の例では、2 番目の SET MESSAGE コマンドが実行されるまで、機能、識別、重大度、テキスト・メッセージの出力が一時的に禁止されます。

```
$ ! 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
```

14.8 ファイルの読み込みと書き込み(ファイル入出力)

コマンド・プロシージャからファイルを読み込んだり、コマンド・プロシージャにファイルを書き込んだりするには、次のようにします。

手順	操作
1	OPEN コマンドを使用してファイルをオープンする。 OPEN コマンドは、論理名をファイルに割り当て、ファイルを読み込むのか、書き込むのか、または読み込みと書き込みを両方行うのかを指定する。これ以降の READ、WRITE、CLOSE コマンドは、この論理名を使用してファイルを参照する。
2	READ または WRITE コマンドを使用して、ファイルからレコードを読み込んだり、ファイルにレコードを書き込んだりする。 ファイルへの入出力を行う 1 つの方法に、レコードを読み込んだり、レコードを処理したり、変更済みのレコードを同じファイルまたは別のファイルに書き込んだりするループを設計する方法がある。
3	CLOSE コマンドを使用してファイルをクローズする。 CLOSE コマンドを取り込まない場合、ログアウトするまでファイルがオープンされたままになる。

注意

ユーザがログインすると、システムが SYS\$INPUT、SYS\$OUTPUT、SYS\$COMMAND、SYS\$ERROR の各ファイルを自動的にオープンするため、これらのファイルを読み書きのために明示的にオープンする必要はありません。

この後の節では、次のことについて説明します。

- OPEN コマンドの使用
- ファイルへの書き込み

- WRITE コマンドの使用
- READ コマンドの使用
- CLOSE コマンドの使用
- ファイルの変更
 - レコードの更新
 - 新しい出力ファイルの作成
 - ファイルへのレコードの追加

14.9 OPEN コマンドの使用

OPEN コマンドは、順編成ファイル、相対編成ファイル、索引順編成ファイルをオープンします。ファイルはプロセス永久ファイルとしてオープンされ、CLOSE コマンドでファイルを明示的にクローズしない限り、プロセスが存在する間はオープンされたままになります。ファイルがオープンされているときは、プロセス永久ファイルの使用に関する OpenVMS RMS の制限事項に従います。

OPEN コマンドは、ファイルをオープンすると、論理名 (最初のパラメータとして指定される) をファイル (2 番目のパラメータとして指定される) に割り当て、その名前をプロセス論理名テーブルに収めます。これ以降の READ、WRITE、CLOSE コマンドはこの論理名を使用してファイルを参照します。

次の例では、OPEN コマンドは、論理名 INFILE を DISK4:[MURPHY]STATS.DAT ファイルに割り当てます。

```
$ OPEN/READ INFILE DISK4:[MURPHY]STATS.DAT
```

注意

OPEN コマンドの中の論理名は固有の名前でなければなりません。入力したコマンドは正しいのに、OPEN コマンドが動作しない場合には、OPEN コマンドの中の論理名を変更します。論理名定義のリストを表示するには、SHOW LOGICAL コマンドを使用します。

コマンド・プロシージャが正しいファイルにアクセスできるようにするには、完全なファイル指定 (たとえば、DISK4:[MURPHY]STATS.DAT) を使用するか、ファイルをオープンする前に SET DEFAULT コマンドを使用して正しいデバイスとディレクトリを指定します。

共用可能ファイルも指定できます。/SHARE 修飾子を使用すると、他の開かれているファイルをアクセス可能に設定できます。さらに、ユーザは DCL コマンド TYPE と SEARCH を使用して、共用可能ファイルにアクセスできます。

OPEN/READ コマンドはファイルを開き、論理名をファイルに割り当て、レコード・ポインタをファイルの先頭に設定します。読み込みの対象としてファイルを開く場合には、レコードを読み込むことはできますが、書き込むことはできません。レコードを読み込むたびに、ポインタは次のレコードに移動します。

このコマンド・プロシージャの OPEN/READ コマンドは、ファイル STATS.DAT を開き、論理名 INFILE をファイルに割り当てます。

```
$ OPEN/READ INFILE DISK4:[MURPHY]STATS.DAT
$ READ_FILE:
$ READ/END_OF_FILE=DONE INFILE DATA
$ GOTO READ_FILE
$ DONE:
$ CLOSE INFILE
$ EXIT
```

新しいファイルに書き込む場合には、OPEN/WRITE コマンドを使用します。OPEN/WRITE コマンドは印刷ファイル形式で順編成ファイルを作成します。ファイルのレコード形式は固定長制御部付き可変長 (VFC) であり、2 バイトのレコード・ヘッダが付いています。/WRITE 修飾子を /APPEND 修飾子と組み合わせて使用することはできません。

既存のファイルを指定した場合には、OPEN/WRITE コマンドは、既存のファイルより 1 つだけ大きいバージョン番号の新しいファイルを開きます。

次の例のコマンド・プロシージャは、書き込みのために使用できる新しいファイル (NAMES.DAT) を作成します。

```
$ 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 コマンドは、既存のファイルの最後にレコードを追加します。存在しないファイルを開こうとした場合には、エラーが発生し、ファイルは開けません。/APPEND 修飾子を /WRITE 修飾子と組み合わせて使用することはできません。

次の例では、レコードが既存のファイル、NAMES.DAT の最後に追加されます。

```
$ OPEN/APPEND OUTFILE DISK4:[MURPHY]NAMES.DAT
$ INQUIRE NEW_RECORD "Enter name"
$ WRITE OUTFILE NEW_RECORD
.
.
.
$ CLOSE OUTFILE
```

OPEN/READ/WRITE コマンドは、レコード・ポインタをファイルの先頭に設定し、最初のレコードを読み込むことができますようにします。この方法を使用してファイルを開く場合には、最後に読み込んだレコードだけを置換できます。新しいレコードをファイルの最後には書き込むことはできません。さらに、変更されたレコードは、置換されるレコードと正確に同じサイズでなければなりません。

次の例では、レコード・ポインタがファイル STATS.DAT の先頭に設定され、最初のレコードを読み込むことができますようにします。

```
$ OPEN/READ/WRITE FILE DISK4:[MURPHY]STATS.DAT
```

14.10 ファイルへの書き込み

ファイルにデータを書き込むには、次のようにします。

手順	操作
1	書き込み用のファイルをオープンする。
2	書き込みループの先頭にラベルを付ける。 単独のレコードを書き込んだり読み込んだりする場合を除いて、ファイル入出力は常にループの中で実行される。
3	書き込もうとするデータを読み込む。 INQUIRE コマンドまたは READ コマンドを使用して、データをシンボルに読み込む。
4	データをテストする。 データが入ってるシンボルをチェックする。シンボルが空の場合 (たとえば、Return を押して当該行にデータを入力しない場合) には、ファイルに書き込もうとするデータの終わりに達したら、ループの終わりに進む。それ以外の場合には、処理を続ける。
5	データをファイルへ書き込む。 WRITE コマンドを使用して、シンボルの値 (1 つのレコード) をファイルに書き込む。
6	ループの先頭に戻る。 以下、ファイルに書き込むデータがなくなるまで、ループが繰り返される。
7	ループを終了して、ファイルをクローズする。

次のコマンド・プロシージャは、データを新しいファイル STATS.DAT に書き込みます。この名前のファイルが存在する場合には、新しいバージョンが作成されます。

```

$ ! 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

```

14.10.1 固有の名前を持つファイルの作成

固有の名前を持つファイルを作成するには、FS\$SEARCH レキシカル関数を使用して、その名前がすでにディレクトリに存在するかどうかを調べます。FS\$SEARCH についての詳細は、『OpenVMS DCL ディクショナリ』のレキシカル関数の説明を参照してください。

次のコマンド・プロシージャは、ファイル名を求めるプロンプトを出してから、FS\$SEARCH レキシカル関数を使用して、省略時のディレクトリで指定された名前を検索します。指定された名前を持つファイルがすでに存在する場合には、ERROR_1 に制御が渡され、プロシージャは、"The file already exists"のメッセージを表示してから、GET_NAME ラベルに制御を戻します。このとき、プロシージャは以下の例に見られるような他のファイル名を求めるプロンプトを出します。

```

$ ! FILES.COM
$ !
$GET_NAME:
$ INQUIRE FILE "File"              ! Prompt the user for a file name
$ IF FS$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

```

14.11 WRITE コマンドの使用

これ以降の節では、WRITE コマンドの使用について説明します。

14.11.1 データの指定

WRITE コマンドのデータを指定する場合は、第 12 章に説明する文字列式の規則に従います。次に、いくつかのデータの指定方法を示します。

- 文字列式として書き込むデータを指定する。WRITE コマンドは、シンボルとレキシカル関数を自動的に置換する。
- リテラル文字列として文字列を出力ファイルに書き込む。WRITE コマンドは、引用符で囲まれた文字列に対してシンボル置換を実行しない。
- リテラル文字列とシンボル名を組み合わせる。シンボル置換を強制的に実行するには、文字列全体を引用符で囲み、シンボルの前に 2 つの一重引用符、シンボルの後に 1 つの一重引用符を指定する。

リテラル文字列とシンボル名を組み合わせる別の方法として、シンボルの前後にカンマを挿入し、カンマで区切られたシンボルの前後を引用符で囲み、文字列全体を二重引用符で囲む方法がある。次の例を参照。

```
$ WRITE OUTFILE "Count is ",COUNT,"."
```

- 強制的なシンボル置換を実行するには、WRITE コマンド行に一重引用符を指定する。
- 一重引用符を使用して文字列内で強制的なシンボル置換を実行することにより、リテラル文字列とレキシカル関数を組み合わせる。

例

```
$! 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
$
```

例を確認する際には、次のことに注意してください。

- 1 書き込むデータを文字列式として指定する。
- 2 *CREATED*文字列をリテラル文字列として出力ファイルに書き込む。
- 3 リテラル文字列とシンボル名を組み合わせる。
- 4 WRITE コマンドの中で一重引用符を使用してシンボル置換を強制する。この例では、WRITE コマンドは、値を COUNT シンボルに置換し、その結果のコマンド文字列 (P4) でシンボル置換を行う。
- 5 一重引用符を使用してリテラル文字列とレキシカル関数を組み合わせる。
- 6 前の WRITE コマンドによって出力ファイル DATA.OUT に書き込まれたデータを表示する。

14.11.2 /SYMBOL 修飾子の使用

WRITE コマンドは、レコードを書き込むとき、書き込まれたレコードの後にレコード・ポインタを置きます。WRITE コマンドは、長さが最大 2,048 バイトまでのレコードを書き込みます。

次のいずれかの条件が存在するときには、/SYMBOL 修飾子を使用してレコードを書き込みます。

- レコードが 1,024 バイトより長い。
- WRITE コマンドの中の式が 255 バイトより長い。

長いレコードの書き込みについての詳細は、『OpenVMS DCL デクショナリ』の WRITE コマンドの説明を参照してください。

14.11.3 /UPDATE 修飾子の使用

WRITE コマンドと一緒に/UPDATE 修飾子を使用すれば、新しいレコードを挿入せずに、レコードを変更できます。/UPDATE 修飾子を使用するには、読み込みと書き込み両方でファイルをオープンしなければなりません。

14.12 READ コマンドの使用

READ コマンドは、レコードを読み込み、その内容にシンボルを割り当てます。READ コマンドを使用すると、1,024 文字以内の長さのレコードを読み込むことができます。ファイルからデータを読み込むには、次のようにします。

手順	操作
1	読み込み用のファイルをオープンする。
2	読み込みループの先頭にラベルを付ける。 単独のレコードを読み込んだり書き込んだりする場合を除いて、ファイル入出力は常にループの中で行われる。
3	ファイルからデータを読み込む。 READ コマンドと一緒に/END_OF_FILE 修飾子を使用して、レコードを読み込み、その内容をシンボルに割り当てる。/END_OF_FILE 修飾子を使用すると、ファイル終わりへの到達時点で、/END_OF_FILE 修飾子によって指定されたラベルに制御が渡される。普通は、読み込みループの終わりを示すラベルを指定する。
4	データを処理する。 ファイルを順に読み込む場合には、現在のレコードを処理してから次のレコードを読み込む。
5	ループの始めに戻る。 ファイル終わりに到達するまで、ループが繰り返される。
6	ループを終了して、ファイルをクローズする。

次のコマンド・プロシージャは、STATS.DAT ファイルの中のそれぞれのレコードを読み込んで処理します。ファイルの終端状態が戻るまで、READ コマンドを繰り返し実行します。ファイルの終端状態が戻ると、END_READ のラベルが付いた行に分岐します。

```
$ 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 コマンドのシンボル名を指定すると、コマンド・インタプリタは、現在のコマンド・レベルのローカル・シンボル・テーブルにシンボル名を置きます。2 つ以上の READ コマンドに同じシンボル名を使用すると、それぞれの READ コマンドがシンボル名の値を再定義します。たとえば、上例のプロシージャでは、READ コマンドは、ループを通るたびに入力ファイル (STATS.DAT) から新しいレコードを読み込み、このレコードを使用して RECORD シンボルの値を再定義します。

14.12.1 /END_OF_FILE 修飾子の使用

ファイルから読み込みを行う場合、普通は、ファイルの終端に到達するまで1つ1つのレコードを順に読み込んで処理していきます。READ コマンドと一緒に/END_OF_FILE 修飾子を使用すれば、ループが作成され、ファイルからレコードを読み込んで処理し、すべてのレコードを読み込み終わった時点でループを終了することができます。

/END_OF_FILE 修飾子に指定するラベルは、GOTO コマンドに指定するラベルと同じ規則に従うことに注意してください (GOTO コマンドについての詳細は、第 13 章を参照してください)。

ループの中で READ コマンドを使用するときには、必ず/END_OF_FILE 修飾子を使用するようにします。そうしないと、OpenVMS レコード管理サービス (OpenVMS RMS) によってファイルの終端を示すエラー状態が戻されたときに、コマンド・インタプリタは、現在の ON コマンドに指定されるエラー・アクションを実行します。たとえば、OpenVMS RMS がエラー状態%RMS-E-EOF を戻すと、コマンド・プロシージャが固有のエラー処理方法を設定している場合を除いて、コマンド・プロシージャは終了します。

14.12.2 /INDEX と/KEY 修飾子の使用

索引順編成ファイルからレコードをランダムに読み込むには、READ コマンド修飾子の/INDEX と/KEY を使用します。/INDEX と/KEY 修飾子は、索引の中で指定されたキーを探してそのキーに関連するレコードを戻すことによって、レコードをファイルから読み込むように指定します。索引を指定しないと、主インデックス (0) が使用されます。

レコードをランダムに読み込む場合、/KEY または/INDEX 修飾子を指定せずに READ コマンドを使用すると、ファイルの残りの部分を順に読み込むことができます。

14.12.3 /DELETE 修飾子の使用

READ コマンドと一緒に/DELETE 修飾子を使用すると、索引順編成ファイルからレコードを削除できます。/DELETE 修飾子を指定した場合は、レコードの読み込みが終わってからレコードがファイルから削除されます。/DELETE 修飾子と一緒に/INDEX と/KEY 修飾子を使用すれば、特定のキーによって指定されたレコードを削除できます。

/DELETE、/INDEX、/KEY 修飾子についての詳細は、『OpenVMS DCL ディクショナリ』の READ コマンドの説明を参照してください。

14.13 CLOSE コマンドの使用

CLOSE コマンドは、ファイルを閉じ、OPEN コマンドで作成された論理名の割り当てを解除します。コマンド・プロシージャを終了する場合には、その前にコマンド・プロシージャで開いたすべてのファイルを閉じてください。開いたファイルを閉じなかった場合には、コマンド・プロシージャが終了するときに、ファイルは開かれたままになり、開かれているファイルに割り当てられている論理名は、プロセス論理名テーブルから削除されません。

次の例では、CLOSE コマンドはファイル STATS.DAT を閉じ、論理名 INFILE の割り当てを解除します。

```
$ OPEN INFILE DISK4:[MURPHY]STATS.DAT
.
.
.
$ CLOSE INFILE
```

14.14 ファイルの変更

この節では、次の 3 種類のファイル変更方法について説明します。

- レコードの更新
- 新しい出力ファイルの作成
- レコードの追加

14.14.1 レコードの更新

更新方式を使用してレコードを変更する場合には、ファイル内の一部のレコードだけを変更できます。この方式では、レコードのサイズを変更したり、ファイル内のレコード数を変更することができないため、書式化されたレコード (たとえばデータ・ファイル内のレコード) に対してだけ使用できます。

ファイルの一部だけを変更するには、次の操作を実行します。

手順	操作
1	読み込みアクセスと書き込みアクセスの両用ファイルをオープンする。
2	READ コマンドを使用して、変更したいレコードに到達するまでファイルを読み込む。

手順	操作
3	レコードを変更する。 順編成ファイルでは、このレコードのテキストは元のレコードと同じサイズでなければならない。変更済みのレコードのテキストの方が短い場合は、レコードにスペースを埋め込んで、元のレコードと同じ長さになるまで変更済みのレコードの終わりにスペースを追加する。変更済みレコードのテキストの方が長い場合は、新しいファイルを作成する必要がある。
4	WRITE/UPDATE コマンドを使用して変更したレコードをファイルに書き込む。
5	変更しようとするすべてのレコードの変更が終わるまで、手順 2 ~ 4 を繰り返す。
6	CLOSE コマンドを使用してファイルをクローズする。 ファイルをクローズした後、個々のレコードが変更済みであっても、ファイルのバージョン番号は最初と同じである。

次のコマンド・プロシージャは、それぞれのレコードを読み込んで更新することによって、順編成ファイルに変更を加える方法を示しています。

```
$! 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
```

レコードはターミナルに表示され、レコードを変更する必要があるかどうか尋ねられます。レコードの変更を選択すると、ターミナルから新しいレコードが読み込まれ、新しいレコードの長さと元のレコードの長さとが比較されます。元のレコードの方が長い場合には、新しいレコードにスペースを追加して同じ長さにします。元のレコードの方が短い場合には、エラー・メッセージが表示され、新しいレコードを求めるプロンプトがもう一度出されます。

14.14.2 新しい出力ファイルの作成

ファイルに大きな変更を加えるには、そのファイルを読み込みアクセス用にオープンし、新しいファイルを書き込みアクセス用にオープンします。新しい出力ファイルを作成するので、レコードのサイズを変更したり、レコードを追加したり削除したりできます。

OPEN/WRITE コマンドは、書き込みアクセス用の新しいファイルをオープンします。新しいファイルは元のファイルと同じ名前にすることができ、バージョン番号は、古いファイルのバージョン番号より 1 だけ大きくなります。

注意

読み込み用に正しいファイルがオープンされるように、既存のファイルを読み込みアクセス用にオープンしてから、新しいバージョンを書き込みアクセス用にオープンするにしなければなりません。

ファイルに変更を加えるには、次の手順に従ってください。

手順	操作
1	該当ファイルを読み込みアクセス用にオープンする。 これは入力ファイルであり、このファイルを変更する。
2	書き込みアクセス用に新しいファイルをオープンする。 これは出力ファイルであり、このファイルを作成する。出力ファイルに入力ファイルと同じ名前を指定すると、出力ファイルは、入力ファイルより 1 だけ大きいバージョン番号を持つ。
3	READ コマンドを使用して、変更中のファイルからレコードを 1 つずつ読み込む。 元のファイルからそれぞれのレコードを読み込むときに、レコードをどのように処理するかを決定する。
4	すべてのレコードが終了するまで、レコードの読み込みと処理を続ける。
5	CLOSE コマンドを使用して、入力ファイルと出力ファイルの両方をクローズする。

次の表では、RECORD シンボルに元のファイルから読み込んだレコードが収められます。

レコードが次の ような場合	結果
変更なし	新しいファイルに同じシンボルを書き込む。
変更あり	INQUIRE コマンドを使用して別のレコードをシンボルに読み込んでから、変更済みのシンボルを新しいファイルに書き込む。
削除された	シンボルを新しいファイルに書き込まない。
挿入された	ループを使用して、シンボルにレコードを読み込んだり、シンボルを新しいファイルに書き込んだりする。

例：レコードの変更

- 次の例では、NEW_FILE シンボルが新しいファイルに書き込まれています。

```
$ ! No change
$ WRITE NEW_FILE RECORD
```

- 次の例では、INQUIRE コマンドが変更されたシンボルを新しいファイルに書き込むために使用されています。

```
$ ! Change
$ INQUIRE NEW_RECORD "New record"
$ WRITE NEW_FILE NEW_RECORD
```

- 次の例では、ループが新しいファイルにシンボルを書き込むために使用されています。

```
$ ! 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:
```

例：出力ファイルの作成

次の例は、入力ファイルからレコードを読み込み、レコードを処理してから出力ファイルにコピーするコマンド・プロシージャを示しています。

```
$! 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
```

14.14.3 ファイルへのレコードの追加

OPEN/APPEND コマンドは、既存のファイルの終端にレコードを追加します。レコードをファイルに追加するには、次のようにします。

手順	操作
1	OPEN コマンドと一緒に/APPEND 修飾子を使用して、レコード・ポインタをファイルの終端に置く。 /APPEND 修飾子は、ファイルの新しいバージョンは作成しない。
2	WRITE コマンドを使用して新しいデータ・レコードを書き込む。
3	すべてのレコードが終了するまで、レコードを追加し続ける。
4	CLOSE コマンドを使用してファイルをクローズする。

以下のコマンド・プロシージャは STATS.DAT という名前のファイルの末尾にレコードを追加しています。

```
$! 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
```

14.15 ファイル入出力エラーの処理方法

OPEN, READ, WRITE コマンドのうちのいずれかと一緒に/ERROR 修飾子を使用すると、システム・エラー・メッセージを表示せずに指定されたラベルに制御を渡すことができます。入出力操作中にエラーが生じた場合には、/ERROR 修飾子は他のすべてのエラー制御方式 (READ コマンドの/EDN_OF_FILE 修飾子を除く) に優先します。

次の例は、OPEN コマンドと一緒に/ERROR 修飾子を使用しています。

```
$ OPEN/READ/ERROR=CHECK FILE CONTINGEN.DOC
.
.
.
$ CHECK:
$ WRITE SYS$OUTPUT "Error opening file"
```

OPEN コマンドは、読み込みモードで CONTINGEN.DOC ファイルをオープンするように要求しています。ファイルをオープンできない場合 (たとえば、ファイルが存在しない場合) には、OPEN コマンドはエラー状態を戻して、CHECK ラベルに制御を渡します。

/ERROR 修飾子によって指定されたエラー・パスは、コマンド・レベルに設定されている現在の ON 状態に優先します。エラーが生じて、ターゲット・ラベルに制御が渡されると、システムに予約されているグローバル・シンボル \$STATUS にエラー・コ

ードが保持されます。エラー処理ルーチンで F\$MESSAGE レキシカル関数を使用すれば、\$STATUS のメッセージを表示できます。

以下の例では、F\$STATUS レキシカルの内容を表示するためにレキシカル関数 F\$MESSAGE が使用されています。

```
$ OPEN/READ/ERROR=CHECK FILE 'P1'
.
.
.
$ CHECK:
$ ERR_MESSAGE = F$MESSAGE($STATUS)
$ WRITE SYS$OUTPUT "Error opening file: ",P1
$ WRITE SYS$OUTPUT ERR_MESSAGE
.
.
.
```

14.15.1 省略時のエラー・アクション

OPEN, READ, WRITE, または CLOSE コマンドを使用しているときにエラーが生じ、エラー・アクションを指定しないと、現在の ON コマンド・アクションが有効になります。

READ コマンドがファイルの終端メッセージを受け取った場合のエラー・アクションは、次のようにして決定されます。

- /END_OF_FILE 修飾子を使用している場合は、指定されたラベルに制御が渡される。
- /END_OF_FILE 修飾子を使用していない場合は、/ERROR 修飾子に指定されたラベルに制御が渡される。
- どちらの修飾子 (/END_OF_FILE または /ERROR) も指定していない場合には、現在の ON コマンド・アクションが有効になる。

14.16 実行フローの制御方法

コマンド・プロシージャでの通常の実行フローは順次です。つまり、ファイルの最後に到達するまで、プロシージャ内のコマンドが順に実行されます。しかし、特定の文を実行するかどうかなど、プロシージャの実行を続行するときの条件を制御することもできます。

この後の節では、次のことについて説明します。

- 実行フローを制御または変更するために使用できる DCL コマンド
 - IF, THEN, ELSE

- GOTO
- GOSUB
- CALL
- コマンド・ブロックの使用
- 場合分けの文の作成
- ループの作成

14.16.1 IF コマンドの使用

IF コマンドは、式の値をテストして、式の結果が真の場合にコマンドまたはコマンドのブロックを実行します。式の結果が偽の場合には、次のいずれかになります。

- THEN コマンドの後に 1 つのコマンドが続く場合には、THEN コマンドは実行されず、その後のコマンドが実行される。
- THEN コマンドの後にコマンドのブロックが続き、ELSE コマンドが指定されていない場合には、ENDIF コマンドのすぐ後にコマンドが実行される。
- ELSE コマンドが指定されている場合には、ELSE コマンドの後のコマンドまたはコマンドのブロックが実行される。

DCL の IF コマンドには、2 つの形式があります。1 つは、第 13 章で説明されているように IF コマンドに指定された式が真の場合に単独のコマンドを実行する形式です。

DCL にはブロック構造の IF 形式もあります。ブロック構造の IF コマンドは、指定された式が真の場合に複数のコマンドを実行します。ELSE 文を指定すれば、式が偽の場合に 1 つ以上のコマンドを実行することができます。

14.16.2 THEN コマンドの使用

式が真の場合に複数のコマンドを実行するには、動詞 (前にドル記号が付いた DCL コマンド) として THEN 文を指定し、残りのブロック構造文の終わりに ENDIF 文を指定します。

次の例では、THEN 文は動詞として使用されています。

```
$ IF expression
$   THEN
$       command
$       command
.
.
.
$ ENDIF
```

14.16.3 ELSE コマンドの使用

式が偽の場合に 1 つ以上のコマンドを実行するには、動詞として ELSE 文を指定し、残りのブロック構造文の終わりに ENDIF 文を指定します。

次の例では、ELSE コマンドは動詞として使用されています。

```
$ IF expression
$   THEN
$       command
$       command
.
.
.
$   ELSE
$       command
$       command
.
.
.
$ ENDIF
```

14.16.4 コマンド・ブロックの使用

同じコマンド・プロシージャでコマンドを実行するのか、別のコマンド・プロシージャにコマンドを置いて実行するのかに応じて、コマンド・ブロックにはいくつかの実行方法があります。ガイドラインは次に示します。

- 同じコマンド・プロシージャでコマンドを実行する場合には、THEN 文の後にコマンドを指定する。次の例を参照。

```
$ IF condition
$ THEN  command
$       command
.
.
.
$ ENDIF
```

- 別のプロシージャでコマンドを実行する場合には、THEN 文の一部としてそのコマンド・プロシージャを呼び出す。次の例を参照。

```
$ IF condition
$ THEN @command_procedure
$ ELSE command
$       command
$ ENDIF
```

- 非ブロック構造の IF 形式を指定して、指定された条件が満たされれば、ラベルが付いたリージョンに制御を渡すことができる。次の例を参照。

```
$ IF not condition THEN GOTO END_LABEL
.
.
.
$END_LABEL:
```

IF 式の結果が真の場合には、THEN コマンドの後のコマンド・ブロックを実行できます。コマンド・ブロックを使用する場合には、IF コマンドの後の行に最初のコマンドとして THEN コマンドを指定します。

次の例では、2 つの SET TERMINAL コマンドが実行され、F\$MODE が“INTERACTIVE”に等しい場合には、プロシージャは制御をラベル PROCEED に渡します。F\$MODE が“INTERACTIVE”に等しくない場合には、プロシージャは終了します。

```
$ IF F$MODE ( ) .EQS. "INTERACTIVE"
$ THEN
$   SET TERMINAL/DEVICE=VT320
$   SET TERMINAL/WIDTH=132
$   GOTO PROCEED
$ ENDIF
$ EXIT
$PROCEED:
```

次の例では、IF コマンドと ELSE コマンド、およびコマンド・ブロックを組み合わせ使用する方法を示しています。

```
$ 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
```

条件が真の場合には、プロシージャはメッセージを SYS\$OUTPUT に書き込み、SHOW DEVICE コマンドと SET DEVICE コマンドを実行します。条件が真でない場合には、プロシージャは 2 つのメッセージを SYS\$OUTPUT に書き込みます。

IF-THEN-ELSE を使用する場合には、次のような制限事項があります。

- ネストされた IF 文は 15 レベル以下にする。

- THEN 文で始まるコマンド・ブロックは ELSE または ENDIF 文によって終了する。
- ELSE 文で始まるコマンド・ブロックは ENDIF 文で終了する。
- THEN 文は、IF 文に続く最初の実行可能文として指定する。
- THEN または ELSE 文を含む行にはラベルを指定しない。ただし、ENDIF 文を含む行にはラベルを指定できる。現在のコマンド・ブロックの中ではプログラムを分岐できるが、別のコマンド・ブロックの途中に分岐しないこと。

真の文

IF コマンドに続く文は、1 つ以上の数値定数、文字列リテラル、シンボリック名、レキシカル関数で構成し、論理演算子、算術演算子、または文字列演算子によって区切ります。次のいずれかの値を持つ場合に式は真となります。

- 奇数の整数値
- Y, y, T, t のいずれかの英字で始まる文字列値
- 奇数の整数になる数字を含む文字列値 (たとえば、文字列 "27")

偽の文

次のいずれかの値を持つ場合に式は偽となります。

- 偶数の整数値
- Y, y, T, t 以外の英字で始まる文字列値
- 偶数の整数になる数字を含む文字列値 (たとえば、文字列 "28")

式の作成

IF コマンドの式を作成する場合には、特に、次の点に注意してください。

- IF 文の中でシンボルを使用すると、その値は自動的に置換される。反復置換を強制する必要がある場合を除いて、置換演算子として一重引用符(')を使用しない。
- 文字列比較演算子の終わりには "S" が付く。たとえば、文字列を比較するには、.EQS.、.LTS.、.GTS. などの演算子を使用する。また、整数を比較するには、.EQ.、.LT.、.GT. の演算子を使用する。
- 2 つの文字列が等しいかどうかを調べる場合は、両方の文字列で同じように大文字と小文字を区別しなければならない。すなわち、文字列 "COPY" は文字列 "copy" または "CoPy" とは等しくない。

この後に示す例は、IF コマンドと一緒に使用できる式を示しています。さらに詳しい例が必要であれば、『OpenVMS DCL デュクショナリ』の IF コマンドの説明を参照してください。

次の例では、論理演算子を使用し、THEN 文の後の 1 つのコマンドだけを実行します。シンボル CONT が真でない場合には、プロシージャは終了します。

```
$ INQUIRE CONT "Do you want to continue [Y/N]"
$ IF .NOT. CONT THEN EXIT
.
.
.
```

次の例は、IF 式の中でシンボルとラベルを使用しています。

```
$ INQUIRE CHANGE "Do you want to change the record [Y/N]"
$ IF CHANGE THEN GOTO GET_CHANGE
.
.
.
$ GET_CHANGE:
.
.
.
```

CHANGE シンボルが真の場合は、プロシージャは GET_CHANGE ラベルに制御を渡します。それ以外の場合には、IF コマンドの後のコマンドを実行します。

次の例は、2 つの異なる IF コマンドを示しています。

```
$ COUNT = 0
$ LOOP:
$   COUNT = COUNT + 1
$   IF COUNT .EQ. 9 THEN EXIT
$   IF P'COUNT' .EQS. "" THEN EXIT
.
.
.
$ GOTO LOOP
```

最初の IF コマンドは 2 つの整数を比較し、2 番目の IF コマンドは 2 つの文字列を比較しています。整数比較には.EQ. 演算子が、文字列比較には.EQS. 演算子が使用されています。

最初に、COUNT の値が整数の 9 と比較されます。値が等しいとプロシージャは終了し、値が等しくないとプロシージャは継続されます。8 つのパラメータ (許容最大数) を処理した後、ループが終了します。

2 番目の IF コマンドでは、P'COUNT' シンボルの文字列値を空の文字列と比較して、シンボルが未定義かどうかを調べます。COUNT シンボルの反復置換を強制するには、一重引用符を使用しなければなりません。たとえば、COUNT が 2 の場合は、最初の変換結果は P2 になります。次に、P2 の値を文字列比較に使用します。

IF 式の結果が真の時に別のコマンド・プロシージャを実行することもできます。次の例では、IF 式の結果が真のときに、コマンド・プロシージャ EXIT_ROUTINE.COM を実行します。

```
$ GET_COMMAND_LOOP:
$   INQUIRE COMMAND -
    "Enter command (DELETE, DIRECTORY, EXIT, PRINT, PURGE, TYPE)"
$   IF COMMAND .EQS. "EXIT" THEN @EXIT_ROUTINE
```

14.16.5 GOTO コマンドの使用

GOTO コマンドは、コマンド・プロシージャ内のラベルの付いた行に制御を渡します (ラベルの使い方についての詳細は、第 13 章を参照してください)。GOTO コマンドは、THEN 句で使用するると特に便利であり、プロシージャを前後に分岐させることができます。たとえば、コマンド・プロシージャでパラメータを使用する場合には、プロシージャの先頭でパラメータをテストし、適切なラベルに分岐できます。

GOTO コマンドまたは GOSUB コマンドの分岐先ラベルは、独立した IF-THEN-ELSE 構造または独立したサブルーチンの内部に存在できません。

次の例では、IF コマンドは、P1 が空文字列でないことをチェックします。

```
$ IF P1 .NES. "" THEN GOTO OKAY
$ INQUIRE P1 "Enter file spec"
$ OKAY:
$ PRINT/COPIES=10 'P1'
.
.
.
```

P1 が空文字列の場合には、GOTO コマンドは実行されず、INQUIRE コマンドはパラメータ値を要求するプロンプトを表示します。それ以外の場合には、GOTO コマンドは INQUIRE コマンドの後に分岐します。どちらの場合も、プロシージャは、OKAY というラベルの付いた行の後の PRINT コマンドを実行します。

次の例では、GOTO コマンドはエラー・メッセージを戻します。これは、分岐先 (TEST_1) が IF-THEN 構造の内部に存在するからです。

```
$ 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
```

14.16.5.1 再実行の回避

正常に実行されたジョブの一部が再実行されないようにするときにも GOTO コマンドを使用できます。この場合は、次のようにします。

手順	操作
1	プロシージャの中の開始ポイントにラベルを付ける。
2	ラベルの後、SET RESTART_VALUE =ラベル名 コマンドを使用して、再開ポイントをそのラベルに設定する。 SET RESTART_VALUE =ラベル名 コマンドの実行後にバッチ・ジョブが中断された場合、システムはバッチ・ジョブを再開するときにグローバル・シンボル BATCH\$RESTART に該当するラベル名を割り当てる。
3	プロシージャの先頭で、\$RESTART シンボルの値をテストする。 \$RESTART が真の場合には、BATCH\$RESTART シンボルを転送ラベルとして使用して GOTO 文を実行する。

\$RESTART グローバル・シンボル

\$RESTART は、システムがユーザ用に残してある予備グローバル・シンボルです。いずれか 1 つのバッチ・ジョブが中断後に再開された場合は、\$RESTART は真であり、そうでない場合は、\$RESTART は偽です。予備グローバル・シンボル \$RESTART は削除できません。

コマンド・プロシージャの中に SET RESTART_VALUE コマンドがあり、この要素の中でジョブを再実行したい場合には、SET ENTRY/NOCHECKPOINT コマンドを入力して、グローバル・シンボル BATCH\$RESTART を削除します。中断されたジョブを再起動した場合には、ジョブは中断されたセクション内で実行されます。

次のコマンド・プロシージャは、バッチ・ジョブの中で値を再開する方法を示しています。

```
$ ! 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
```

このコマンド・プロシージャを再開可能なバッチ・ジョブとしてキューに登録するには、ジョブに登録するときに SUBMIT コマンドと一緒に /RESTART 修飾子を指定します。割り込まれたジョブを再開する場合は、割り込みのあった場所からジョブの実行が開始されます。たとえば、SORT_FILE ルーチンの中でジョブに割り込みがあった場合は、ジョブを再開すると、SORT_FILE ラベルから実行が開始されます。

システムで障害が発生した場合には、プロセス環境の大部分は維持管理されません。システム障害が発生した後も管理されるシンボルは、\$RESTART と BATCH\$RESTART だけです。したがって、コマンド・プロシージャで使用したシンボルやプロセス論理名は、それぞれの SET RESTART_VALUE コマンドの後で再定義するか、または \$RESTART が真のときに実行される THEN ブロックの内部で再定義しなければなりません。シンボルと論理名を THEN ブロックの内部で定義した場合には、コマンド GOTO 'BATCH\$RESTART' を THEN ブロックの最後のコマンドとして指定しなければなりません。

14.16.6 GOSUB と RETURN コマンドの使用

GOSUB コマンドは、コマンド・プロシージャの中のラベルの付いたサブルーチンに制御を渡します。コマンド・プロシージャの中にラベルがない場合には、実行は継続されず、強制終了されます (ラベルについての詳細は、第 13 章を参照してください)。GOSUB コマンドは、プロシージャ・レベルあたり最大 16 までネストすることができます。

GOSUB コマンドは、ローカル・サブルーチン・コールの 1 つで、新しいプロシージャ・レベルは作成しません。このため、現在のコマンド・レベルで定義されているすべてのラベルとローカル・シンボルを GOSUB によって起動されたサブルーチンで使用できます。

RETURN コマンドは、サブルーチンを終了して、GOSUB コマンドの後のコマンドに制御を戻します。RETURN コマンドで \$STATUS の値を指定すると、DCL がサブルーチンの終了時に \$STATUS に割り当てる値を無効にできます。この値は、0 ~ 4 の整数か同値式でなければなりません。\$STATUS の値を指定すると、DCL はこの値を条件コードとして解釈します。\$STATUS の値を指定しないと、\$STATUS の現在の値がセーブされます。

次の例は、GOSUB コマンドを使用して制御をサブルーチンに渡す方法を示しています。


```

$!
$! 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

```

例を確かめる場合は、次の点に注意してください。

- 1 最初の GOSUB コマンドは TEST1 のラベルが付いたサブルーチンに制御を渡す。
- 2 プロシージャは、TEST1 サブルーチンの中のコマンドを実行して、TEST2 のラベルが付いたサブルーチンに分岐する。
- 3 TEST1 サブルーチンの中の RETURN コマンドは、メイン・コマンド・プロシージャに制御を戻し、正しく実行されたことを示す 1 の値を\$STATUS に渡す。
- 4 TEST2 サブルーチンの中の RETURN コマンドは、TEST1 サブルーチンに制御を戻す。このコマンドは 3 番目のコマンドの前に実行されることに注意。

14.17 新しいコマンド・レベルの作成

新しいコマンド・レベルを作成するには、次の 2 種類の方法があります。

- コマンド・プロシージャの内部でプロシージャ実行(@) コマンドを使用して、別のコマンド・プロシージャを起動することにより、コマンド・プロシージャをネスティングする方法 (第 13 章を参照)
- CALL コマンドを使用して、コマンド・プロシージャの内部に存在するサブルーチンを呼び出す方法

14.17.1 CALL コマンドの使用

CALL コマンドは、コマンド・プロシージャの中のラベルが付いたサブルーチンに制御を渡して、新しいコマンド・レベルを作成します。CALL コマンドを使用すると、1つのファイルの中に2つ以上の関連コマンド・プロシージャを保持できるため、プロシージャを管理しやすくなります。サブルーチン・ラベルは一意でなければならず、コマンド・プロシージャの中のCALL コマンドの前後に置きます。サブルーチン・ラベルの入力規則については、第13章を参照してください。

ラベルに加えて、サブルーチンには最大8個のオプション・パラメータを渡すことができます。パラメータについての詳細は、第14.2節を参照してください。

CALL コマンドを使用する場合には、次の規則に従ってください。

- 出力はSYSS\$OUTPUTに送信する。
- サブルーチンからファイルに出力を送信できるように、オプションの/OUTPUT修飾子を指定する。
- 出力ファイルに対しては、省略時のファイル・タイプ.LISを使用する。
- 出力ファイル指定でワイルドカード文字を使用することはできない。

14.17.1.1 CALL コマンドの省力時の設定

CALL コマンドを使用する場合には、次の省略時の設定が使用されます。

- CALL コマンドで呼び出したサブルーチンとプロシージャ実行(@)コマンドで呼び出したプロシージャは、最大32コマンド・レベルまでネストすることができる。
- SET SYMBOL コマンドによってマスクされている場合を除いて、外側のレベルで定義されたローカル・シンボルは内側のどのプロシージャまたはサブルーチン・レベルでも使用でき、グローバル・シンボルはどのコマンド・レベルでも使用できる。
- ラベルは、定義されたときのレベルでのみ有効。

14.17.1.2 サブルーチンの始まりと終わり

SUBROUTINE コマンドとENDSUBROUTINE コマンドは、CALL サブルーチンの始まりと終わりを定義します。SUBROUTINE コマンドのすぐ前にサブルーチンへのエントリ・ポイントを定義するラベルを置きます。ENDSUBROUTINE コマンドのすぐ前にEXIT コマンドを置くことができますが、サブルーチンを終了する必要はありません。ENDSUBROUTINE コマンドは、サブルーチンを終了して、CALL コマンドのすぐ後のコマンド行に制御を渡します。

サブルーチンの中のコマンド行が実行されるのは、サブルーチンをCALL コマンドで呼び出す場合だけです。コマンド・プロシージャを行ごとに実行する場合、コマンド言語インタプリタは、SUBROUTINE コマンドとENDSUBROUTINE コマンドの間のすべてのコマンドをスキップします。

サブルーチン・エントリ・ポイントの有効範囲を定義したり，ラベル参照を使用するときには，次のような制限事項が適用されます。

- 別のサブルーチンの中で定義されるサブルーチン・エントリ・ポイントは，該当サブルーチンに対してローカルとなる。サブルーチン・エントリ・ポイントが別のサブルーチン・ブロックの中にある場合には，そのサブルーチンは呼び出せない。
- サブルーチン・エントリ・ポイントが IF-THEN-ELSE ブロックの中にある場合には，IF-THEN-ELSE ブロックの外からはこのサブルーチンを呼び出せない。
- すべての SUBROUTINE コマンドには，サブルーチンの終わりを示すための ENDSUBROUTINE コマンドがなければならない。

次の例では，呼び出しは正しくありません。これは，CALL BAR コマンドが MAIN サブルーチンの外部に存在するからです。

```
$ CALL BAR
$
$ MAIN: SUBROUTINE
$
$     BAR: SUBROUTINE
$     ENDSUBROUTINE
$
$ ENDSUBROUTINE
```

この CALL コマンドが正しく機能するようにするには，このコマンドを，SUBROUTINE と ENDSUBROUTINE で囲まれる部分に指定しなければなりません。

この例に示した呼び出しは，IF-THEN-ELSE ブロックの内部に存在するため，実行できません。

```
$ IF 1
$ THEN
$     BOB: SUBROUTINE
$     ENDSUBROUTINE
$ ENDIF
$ CALL BOB
```

次の例には，SUB 1 と SUB2 という 2 つのサブルーチンがあります。これらのサブルーチンは，CALL コマンドで呼び出されるまでは実行されません。

DCL での拡張プログラミング

14.17 新しいコマンド・レベルの作成

```
$
$! 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=NAME$ LOG SUB1 "THIS IS P1"
    .
    .
    .
$     CALL SUB2 "THIS IS P1" "THIS IS P2"
    .
    .
    .
$ EXIT          !Exit this command procedure file.
```

CALL コマンドが SUB1 サブルーチンを起動して、出力を NAMES.LOG ファイルに切り換えます。SUB1 サブルーチンは SUB2 サブルーチンを呼び出しています。プロシージャは SUB2 を実行して、プロシージャ実行(@)コマンドによってコマンド・プロシージャ FILE.COM を起動します。SUB1 のすべてのコマンドが実行されると、メイン・プロシージャの中の CALL コマンドが SUB2 をもう一度呼び出します。SUB2 の実行が終わると、プロシージャは終了します。

14.18 場合分け文の使用

場合分け文は、特殊な形式の条件コードであり、変数または式の値に応じて、一連のコマンド・ブロックの中の 1 つのブロックだけを実行します。普通、場合分け文で有効な値は、それぞれのコマンド・ブロックの先頭にあるラベルです。場合分け文は、指定された値を GOTO 文のターゲット・ラベルとして使用することによって、該当するコード・ブロックに制御を渡します。

場合分け文を作成するには、次のようにします。

- 1. ラベルをリストする。
- 2. 場合分け文を作成する。
- 3. コマンド・ブロックを作成する。

14.18.1 ラベルのリスト

ラベルをリストするには、スラッシュ (または、区切り文字として選択した文字) で区切られたラベルのリストが入っている文字列をシンボルに定義します。このシンボル定義はコマンド・ブロックの前に置かななくてはなりません。

次の例は、COMMAND_LIST シンボルをラベル PURGE、DELETE、EXIT と同じにしています。

```
$ COMMAND_LIST = "/PURGE/DELETE/EXIT/"
```

14.18.2 場合分け文の作成

場合分け文を作成するには、次の手順に従ってください。

手順	操作
1	INQUIRE コマンドを使用して場合分け変数の値を得る。
2	IF コマンドと一緒に F\$LOCATE と F\$LENGTH を使用して、場合分け変数の値が有効かどうかを判別する。
3	場合分け変数が有効な場合には、場合分け文 (GOTO コマンド) を実行して、該当するコード・ブロックに制御を渡す。 場合分け変数が有効でない場合は、メッセージを表示して処理を終了するか、別の場合分け値を要求する。

次の例では、ラベルが完全なコマンド名に定義されています。したがって、F\$LOCATE は、コマンド名の検索文字列の中に区切り文字を含めて、コマンドが短縮されないようにしています。

```
$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
```

それぞれのコマンド・ブロックには、1 つ以上のコマンドを指定できます。コマンド・ブロックの先頭には固有のラベルを付けます。それぞれのコマンド・ブロックを終わるときには、コマンド・ブロックのリストにないラベルに制御を渡します。

コマンド・ブロックの作成

次の例では、それぞれのコマンド・ブロックに、1 つ以上のコマンドを指定しています。コマンド・ブロックの先頭には固有のラベルを付けます (PURGE:, DELETE:)。それぞれのコマンド・ブロックを終わるときには、コマンド・ブロックのリストにないラベル (GOTO GET_COMMAND) に制御を渡します。

```
$GET_COMMAND:
.
.
.
$PURGE:
$ INQUIRE FILE
$ PURGE 'FILE'
$ GOTO GET_COMMAND
$ !
$DELETE:
$ INQUIRE FILE
$ DELETE 'FILE'
$ GOTO GET_COMMAND
$ !
$EXIT:
```

14.19 ループの作成

コマンド・ブロックの先頭に、変数をテストするループを作成できます (第 13 章を参照)。しかし、次の操作を実行すれば、ループの最後に終了変数をテストするループを作成することもできます。

手順	操作
1	ループを開始する。
2	ループ本体の中のコマンドを実行する。

手順	操作
3	終了変数を変更する。
4	終了変数をテストする。 条件が満足されない場合は、ループの始めに戻る。
5	ループを終了する。

ループの終わりにある終了変数をテストする場合は、終了変数の中の値にかかわらず、ループ本体のコマンドが少なくとも 1 回は実行されます。

次の 2 つの例は、COMMAND が "EX" (EXIT) に定義されたときに終了するループを実行します。F\$EXTRACT は COMMAND を最初の 2 文字に切り捨てます。最初の例では、終了変数の COMMAND がループの始めにテストされ、2 番目の例では、ループの終わりでテストされます。

```
$ ! 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
```

ループを一定回数実行するには、終了変数としてカウンタを使用します。次の例では、ユーザによって入力された 10 個のファイル名がローカル・シンボル FIL1, FIL2, ..., FIL10 に収められます。

DCL での拡張プログラミング

14.19 ループの作成

```
$ 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
.
.
.
```

次の例は、カウンタを使用してループの実行回数を制御します。この例では、ループは 10 回実行されます。終了変数はループの終わりにテストされます。

```
$! 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 シンボルを使用してループの中のコマンドの実行回数を記録します。また、シンボル名 FILE_1, FILE_2 から FILE_10 までを作成するのに COUNT を使用しています。COUNT の値が増分されるのはループの始めですが、テストされるのはループの終わりであることに注意してください。したがって、COUNT が 9 から 10 に増分されると、IF 文が偽の結果を検出する前にループがもう 1 回だけ実行されます (FILE_10 の値を得ます)。

一連の値に対してループを実行するには、FSELEMENT レキシカル関数を使用します。FSELEMENT 関数は、区切り文字によって区切られた項目リストから項目を取り出します。FSELEMENT の引数として項目番号、項目区切り文字、リストを指定しなければなりません。

FSELEMENT レキシカル関数の使用方法についての詳細は、『OpenVMS DCL ディクショナリ』を参照してください。

次の例では、ファイル CHAP1, CHAP2, CHAP3, CHAPA, CHAPB, CHAPC が順に処理されます。


```
$ 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
```

次のコマンド・プロシージャは、ループを使用して、FILE_LIST シンボルにリストされたファイルを別のノードのディレクトリにコピーします。

```
$ 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 レキシカル関数から戻される最初のファイルは CHAP1 であり、次のファイルは CHAP2 です。以下も同様です。ループを実行するたびに、NUM の値が増分されるため、次のファイル名が入手されます。F\$ELEMENT がスラッシュを戻した場合には、FILE_LIST のすべての項目が処理されたため、ループは終了します。

14.20 PIPE コマンドの使用

PIPE コマンドを使用すると、同一コマンド行から 1 つまたは複数の DCL コマンドを実行できます。これにより、コマンドのパイプライン処理、入出力のリダイレクト、条件実行、バックグラウンド処理など、UNIX 形式のコマンドを処理できるようになります。

このようなコマンド処理の形式は、インターネット・ソフトウェアの開発や使用をサポートします。インターネット・ソフトウェアでは、ホスト・システムとターゲッ

ト・システムの両方で存在を解析する，パイプライン・コマンドの使用を前提にしています。

これ以降の節では，PIPE コマンドへの割り込み方法，サブプロセスの性能を向上させる方法など，DCL コマンドの実行以外の目的で PIPE コマンドを使用する方法を説明します。

PIPE コマンドについての詳細は，『OpenVMS DCL デictionary: N-Z』を参照してください。

1 つの PIPE コマンドに，複数の DCL コマンドを指定できます。指定した DCL コマンドは，順番に実行されます。次の形式を使用します。

```
PIPE command-sequence ; command-sequence [; command-sequences]...
```

14.20.1 条件コマンド実行のための PIPE コマンドの使用

先行するコマンド・シーケンスの実行結果により，次のコマンド・シーケンスが実行されます。

```
PIPE command-sequence1 && command-sequence2
```

command-sequence1 が成功した時のみ，command-sequence2 が実行される点に注意してください。次の形式を使用すると，command-sequence1 が失敗した時のみ command-sequence2 が実行されます。

```
PIPE command-sequence1 || command-sequence2
```

14.20.2 パイプライン実行のための PIPE コマンドの使用

パイプラインは，縦線(|)区切り文字で表されるパイプで接続されたパイプライン・セグメント・コマンドのシーケンスです。パイプライン・セグメント・コマンドとは，パイプライン内の DCL コマンドです。パイプはパイプライン・セグメント・コマンドの SYSS\$OUTPUT を，次のコマンドの SYSS\$INPUT に接続します。パイプラインの形式を次に示します。

```
PIPE pipeline-segment-command | pipeline-segment-command [| ... ]
```

各パイプライン・セグメント・コマンドは，SYSS\$OUTPUT を次のパイプライン・セグメント・コマンドの SYSS\$INPUT に接続し，別々のサブプロセスで実行されます。これらのサブプロセスはパラレルに実行されます。ただし，最初のパイプライン・セグメント・コマンドを除く各パイプライン・セグメント・コマンドが，その先行パイプライン・セグメント・コマンドの標準出力をその標準入力として読み込む程度に同期化されます。最後のパイプライン・セグメント・コマンドが終了すると，パイプラインは実行を終了します。

通常、パイプラインでは“フィルタ・アプリケーション”を使用します。フィルタ・アプリケーションとは、SYSS\$INPUT からのデータを取り、指定した方法でデータを変換し、それを SYSS\$OUTPUT に書き込むプログラムのことです。

パイプラインのコンテキストでは、DCL の機能は一部異なります。次に例を示します。

- SYSS\$COMMAND の使用

サブプロセスの SYSS\$COMMAND は、通常、その SYSS\$INPUT(コマンド・プロシージャが関係しない場合) と同じです。ただしパイプラインでは、サブプロセスの SYSS\$COMMAND は、先行するパイプ (パイプライン・セグメント・コマンドの SYSS\$INPUT) ではなく、親プロセスの SYSS\$COMMAND に設定されます。

- ファイル・アクセス方法

パイプライン・セグメント・コマンドは、パイプとの読み込みと書き込みに、RMS 順編成ファイル・アクセス方法しか使用できません。一部の OpenVMS ユーティリティは、順次アクセス以外の方法を使用して、入力ファイルや出力ファイルにアクセスすることがあります。これらの操作は、パイプラインではサポートされていないので失敗します。次の例を参照してください。

```
$ PIPE CC/NOOBJ/NOLIS TEST.C | SEARCH SYSS$INPUT/WIND=(1,1) "%cc-w-"
```

```
%SEARCH-F-RFAERR, RMS error using RFA access  
-RMS-F-RAC, invalid record access mode
```

この例では、SEARCH コマンドに/WINDOW 修飾子を指定して実行するためには、相対編成ファイル・アクセス方法が必要です。

- シンボルの置換

DCL がシンボルを置換する順序に注意してください。シンボルの置換は、コマンド処理の第 1 フェーズで行われます。PIPE コマンドの一部としてシンボルを定義した場合、DCL は、シンボルを置換してから、そのシンボルが実際に定義されているコマンドを実行しようとします。シンボルの置換を遅延させるには、アンパサンド (&) を使用します。詳細は、第 12.12.2 項を参照してください。

- SYSS\$PIPE の使用

ほとんどの場合、パイプからの入力は、SYSS\$INPUT からデータを読み込むことにより取得できます。ただし、コマンド・プロシージャがパイプライン・セグメント・コマンドとして起動される場合は、SYSS\$INPUT はコマンド・プロシージャ・ファイルにリダイレクトされます。コマンド・プロシージャの中でパイプからデータを取得するには、論理名 SYSS\$PIPE を使用できます。

次に、パイプライン DCL アプリケーション例 TEE.COM を示します。

```
$ ! 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.COM を使用して、次の PIPE コマンドを入力します。

```
$ PIPE SHOW SYSTEM | @TEE showsys.log | SEARCH SYS$INPUT LEF
```

コマンド・プロシージャ TEE.COM は、パイプラインを流れるデータを記録します。データは、SYS\$INPUT ではなく SYS\$PIPE から読み込まれます。

- イメージのチェック

パイプラインでは、PIPE コマンドを入力する前に SET VERIFY=IMAGE コマンドを実行した場合でも、省略時の設定でイメージのチェックはオフになっています。これにより、データ・レコードが重複してパイプラインを通過しないようになります。

パイプラインでのイメージのチェックをオンにするには、パイプライン・セグメント・コマンドの前に明示的に SET VERIFY=IMAGE コマンドを使用しなければなりません。この場合、次のようにサブシェルを使用できます。

```
$ PIPE ... | (SET VERIFY=IMAGE ; ...) | ...
```

14.20.3 サブシェル実行のための PIPE コマンドの使用

サブシェルは、区切り文字で区切られ、括弧で囲まれた 1 つまたは複数のコマンド・シーケンスです。サブシェルの形式を次に示します。

```
PIPE ( command-sequence [separator command-sequence]... )
```

サブシェル内のコマンド・シーケンスは、サブプロセス環境で実行されます。DCL は、サブシェルが終了してから、次のコマンド・シーケンスを実行します。() 区切り文字は、SPAWN/WAIT コマンドに似ています。

この形式の PIPE コマンドを使用するときには、シンボルの置換の扱いに注意してください。シンボルを定義した後、そのシンボルを使用するときには、シンボルの前にアンパサンド (&) を付けてシンボルの置換を遅延させます。アンパサンドを付けなかった場合、シンボルの置換はコマンド処理の第 1 フェーズで行われ、シンボルの定義が信頼できないものになります。

14.20.4 バックグラウンド実行のための PIPE コマンドの使用

コマンド・シーケンスは、次の形式を使用してサブプロセス環境で実行されます。

PIPE command-sequence [separator command-sequence]... &

DCL は、コマンド・シーケンスの終了を待ちません。バックグラウンド・サブプロセスが作成されると、制御は DCL に戻ります。

14.20.5 入出力リダイレクトのための PIPE コマンドの使用

コマンド・シーケンスは、次のようにコマンドの実行中に、SYSS\$INPUT、SYSS\$OUTPUT、または SYSS\$ERROR をファイルにリダイレクトできます。

- SYSS\$INPUT をリダイレクトするには、次の形式を使用します。

PIPE command-sequence < redirected-input-file

- SYSS\$OUTPUT をリダイレクトするには、次の形式を使用します。

PIPE command-sequence > redirected-output-file

- SYSS\$ERROR をリダイレクトするには、次の形式を使用します。

PIPE command-sequence 2> redirected-error-file

パイプライン・セグメント・コマンドも、SYSS\$INPUT、SYSS\$OUTPUT、または SYSS\$ERROR をリダイレクトできます。ただし、SYSS\$OUTPUT のリダイレクトは、最後のパイプライン・セグメント・コマンドだけに使用でき、SYSS\$INPUT のリダイレクトは、最初のパイプライン・セグメント・コマンドだけに使用できます。

PIPE コマンドのリダイレクトは、DEFINE コマンドや ASSIGN コマンドを使用して行うリダイレクトとは異なります。相違点を以下に示します。

- リダイレクトは、スーパーバイザ・モードで作成される。つまり、ユーザ・モード・アプリケーションと DCL コマンドの両方が、リダイレクトに影響される。
- リダイレクトされた環境は、リダイレクト構文を指定するコマンド・シーケンス、またはパイプライン・セグメント・コマンドだけに適用される。コマンド・シーケンスまたはパイプライン・セグメント・コマンドの実行後、コマンドの実行が続行される前に、元のプロセスの入出力環境 (たとえば SYSS\$INPUT、SYSS\$OUTPUT、および SYSS\$ERROR) が復元される。

SYSS\$OUTPUT をリダイレクトすると、コマンド・シーケンスが実際に SYSS\$OUTPUT に書き込むかどうかにかかわらず、常にリダイレクトされた出力ファイルが作成されます。リダイレクトされた出力ファイルと同じ名前を持つファイルのバージョンがすでに存在する場合、そのファイルの新しいバージョンが作成されます。この動作は、スーパーバイザ・モードで DEFINE または ASSIGN コマンドを使用して、SYSS\$OUTPUT を再定義する場合と同じです。リダイレクトされたファイルは、コマンド・シーケンスが実行される前に作成されることに注意してください。次

の例のように、リダイレクトされたファイルがコマンド・シーケンスでも使用される場合は、操作が失敗することがあります。

```
$ 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
```

この例では、新しいバージョンの TRANS.LOG が作成され、書き込みアクセス用にオープンされます。次に SEARCH コマンドが、前のバージョンではなく最新バージョンの TRANS.LOG に読み込みアクセスしようとします。

SYSS\$ERROR をリダイレクトすると、コマンド・シーケンスが実行中に実際に SYSS\$ERROR に書き込む場合のみ、リダイレクトされたファイルが作成されます。リダイレクトされたエラー・ファイルと同じ名前を持つ既存のファイルは存在しません。リダイレクトされたエラー・ファイルと同じ名前を持つファイルがすでに存在する場合は、そのファイルがリダイレクトされたエラー・ファイルとしてオープンされます。次に、このコマンド・シーケンスで作成されたエラー出力は、リダイレクトされたエラー・ファイルの最後に追加されます。この動作は、スーパーバイザ・モードで DEFINE または ASSIGN コマンドを使用して、SYSS\$ERROR を再定義する場合と同じです。

14.20.6 PIPE コマンドへの割り込み

Ctrl/Y を押すと、PIPE コマンドに割り込みをかけることができます。PIPE コマンドをパイプラインまたはサブシェル・コマンド・シーケンスで実行している場合は、コマンド・シーケンスと PIPE コマンドが削除されます。この場合、割り込み直後に CONTINUE コマンドを入力しても、PIPE コマンドの実行は再開されません。

PIPE コマンドが、サブシェル・コマンド・シーケンスまたはパイプライン・コマンド・シーケンス以外の、コマンド・シーケンスを実行している場合は、DCL は、コマンド・シーケンスが PIPE コマンド動詞なしで入力され、Ctrl/Y で割り込みをかけられたように動作します。Ctrl/Y 割り込みについての詳細は、第 13.11 節を参照してください。

14.20.7 サブプロセスの性能向上

PIPE コマンドは、実行中に多数のサブプロセスを作成できます。通常、コマンド・シーケンスにより起動されるアプリケーションは、プロセスの論理名とシンボル名に依存しません。この場合、/NOLOGICAL_NAME および/NOSYMBOLS 修飾子を使用すると、サブプロセスを素早く作成できます。これらの修飾子により、プロセスの論理名とシンボル名は、PIPE コマンドにより作成されたサブプロセスに渡されなくなります。

PIPE コマンドの使用例を次に示します。

- 次の例は、シンボル定義を持つ複数のコマンドを使用して、コマンド・プロシージャに有用なツールを作成する 2 つの簡単な方法を示しています。

```
$ 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:
$      :
```

- 次の例は、コンパイルおよびリンク操作を示しています。コンパイル中にエラーが発生しなければ、オブジェクト・ファイルは実行可能イメージを生成するためにリンクされます。コンパイル・エラーが発生すると、リンクはスキップされます。

```
$ PIPE cc foo.c && link foo, sys$library:vaxcrtl.olb/lib
```

- 次の例は、条件付きコマンドを実行して、コマンド・プロシージャ中にエラー処理制御フローを簡単に作成する方法を示しています。イメージ COLLECT_DATA が失敗すると、制御は CLEAN_UP に移ります。

```
$ PIPE RUN COLLECT_DATA.EXE || GOTO CLEAN_UP
$      :
$      :
$ EXIT
$
$ CLEAN_UP:
$      :
$      :
```

- この例の PIPE コマンドは、大きなファイルをコピーするためのバックグラウンド・プロセスを作成します。

```
$ PIPE COPY LARGE_FILE.DAT REMOTE"user password"::[DESTINATION]*.* &
```

- 次の例は、サブシェル・コマンド・シーケンスを、サブプロセスで実行しています。その結果、プロセス固有の属性 (たとえば省略時のディレクトリ) を変更しても、サブシェルの終了後、現在のプロセスに影響しません。この例では、プログラム FOO を実行するために必要なデータを提供するため、セーブ・セットがサブディレクトリに復元されます。

```
$ PIPE (SET DEF [.DATA_DIR] ; BACKUP DATA.SAV/SAV [...]) ; RUN FOO
```

- 次の例は、パイプライン機能を使用して、システム上のすべてのバイバネート・プロセスを 1 つのコマンドで特定します。

```
$ PIPE SHOW SYSTEM | SEARCH SYS$INPUT HIB
```

- 次の例は、パイプライン機能を使用して、1 つのコマンドでテストを実行し、その結果をソートし、結果をベンチマーク・ファイルと比較します。この時、不要な中間ファイルは生成します。

```
$ PIPE RUN TEST | SORT/SPECIFICATION=TEST.SRT SYS$INPUT SYS$OUTPUT -  
| DIFF SYS$INPUT TEST.BENCHMARK
```

- 次の例は、パイプラインで、パイプ・セグメント・コマンドとしてサブシェルを指定する 1 つの方法を示しています。

```
$ PIPE ( SET DEF WRK$:[WORK] ; RUN REPORT ) | MAIL SYS$INPUT SMITH
```

- 次の例は、パイプラインで/PAGE 修飾子を使用する例です。/PAGE は、他の多数のコマンドにも存在し、同じように PIPE コマンドと組み合わせて使用して他の有用なツールを作成することができます。

```
$ more := TYPE/PAGE=SAVE SYS$INPUT  
$ PIPE ANA/RMS PAGE.TXT | more
```

```
Check RMS File Integrity                               26-JAN-2002 16:12:00.06 Page 1  
SYS$SYSDEVICE:[TEST]PAGE.TXT;2
```

FILE HEADER

```
File Spec: SYS$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

レキシカル関数を使用しての情報の取得と処理

レキシカル関数は、コマンド行またはコマンド・プロシージャに情報を戻します。戻される情報は、プロセス、システム、ファイルとデバイス、論理名、文字列、データ型に関するものです。レキシカル関数には、接頭辞 F\$ が付いています。

レキシカル関数は、通常、シンボルや式を使用するコンテキストの中で使用します。コマンド・プロシージャの中では、論理名を変換したり、文字列処理を行ったり、プロシージャの現在の処理モードを判別したりするのに使用します。レキシカル関数が戻す情報の多くは、DCL コマンドで取得することもできます。

本章では、次のことを説明します。

- レキシカル関数の機能方法
- プロセスに関する情報の入手
- システムに関する情報の入手
- ファイルとデバイスに関する情報の入手
- 論理名の変換
- 文字列の処理
- データ・タイプの処理

レキシカル関数についての詳しい説明は、オンライン・ヘルプを参照してください。本章で説明するコマンドについての詳細は、『OpenVMS DCL デictionary』を参照してください。

15.1 なぜレキシカル関数を使うのか

コマンドから情報を得るよりもレキシカル関数から情報を得た方が、コマンド・プロシージャの中で情報を処理しやすくなります。たとえば、F\$ENVIRONMENT 関数または SHOW DEFAULT コマンドのどちらを使用しても、現在の省略時のディレクトリの名前を得ることができます。ただし、次の点が異なります。

- F\$ENVIRONMENT レキシカル関数を使用した場合は、結果をシンボルに割り当てておき、後でコマンド・プロシージャの中でこのシンボルを使用できる。

レキシカル関数を使用しての情報の取得と処理

15.1 なぜレキシカル関数を使うのか

```
$ DIR_NAME = F$ENVIRONMENT("DEFAULT")
$ SET DEFAULT DISK4:[TEST]
.
.
.
$ SET DEFAULT 'DIR_NAME'
```

上記の例で、F\$ENVIRONMENT 関数は、現在の省略時のディスクとディレクトリを戻し、この値を DIR_NAME シンボルに格納する。プロシージャの終わりに、この DIR_NAME シンボルを使用して SET DEFAULT コマンドによって省略時の値に戻している。

- 現在の省略時のディレクトリの値を F\$ENVIRONMENT レキシカル関数によってではなく、SHOW DEFAULT コマンドによって得た場合には、この出力をシンボルに直接に割り当てることはできない。プロシージャは次のようになる。

```
$! 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.
```

15.2 プロセスについての情報

コマンド・プロシージャを実行している間、プロセス属性を頻繁に変更し、その後で属性を復元します。次のレキシカル関数を使用すれば、プロセスに関する情報を入手できます。

F\$DIRECTORY	現在の省略時のディレクトリ文字列を戻す。
F\$ENVIRONMENT	ユーザ・プロセスのコマンド環境についての情報を戻す。
F\$GETJPI	ユーザ・プロセスまたはシステム上の他のプロセスについての会計情報、状態、識別情報を戻す。
F\$MODE	プロセスを実行中のモードを示す。
F\$PRIVILEGE	ユーザ・プロセスに特権が指定されているかどうかを示す。
F\$PROCESS	プロセスの名前を戻す。
F\$SETPRV	指定された特権を設定する。指定された特権が、F\$SETPRV 関数を使用する前に使用可能であったかどうかを示す。
F\$USER	UIC (ユーザ識別コード) を戻す。
F\$VERIFY	チェック機能のオンまたはオフを示す。

次の表は、コマンド・プロシージャで一般に変更されるプロセス属性を示しています。また、これらの属性をセーブするレキシカル関数と、元の設定を復元する DCL コマンドも示しています。

属性	操作	コマンドまたはレキシカル関数
制御文字	セーブ	F\$ENVIRONMENT("CONTROL")
	復元	SET CONTROL
DCL プロンプト	セーブ	F\$ENVIRONMENT("PROMPT")
	復元	SET PROMPT
省略時の保護	セーブ	F\$ENVIRONMENT("PROTECTION")
	復元	SET PROTECTION/DEFAULT
キー状態	セーブ	F\$ENVIRONMENT("KEY_STATE")
	復元	SET KEY
メッセージ形式	セーブ	F\$ENVIRONMENT("MESSAGE")
	復元	SET MESSAGE
特権	セーブ	F\$PRIVILEGE または F\$SETPRV
	復元	F\$SETPRV または SET PROCESS /PRIVILEGES
チェック機能	セーブ	F\$VERIFY または F\$ENVIRONMENT
	復元	F\$VERIFY または SET VERIFY

プロセス属性をセーブした場合、エラーまたは Ctrl/Y による割り込みによって、元の属性を復元する前にプロシージャが終了することのないようにしなければなりません。エラーと Ctrl/Y による割り込みの処理については、第 13 章を参照してください。

15.2.1 チェック設定値の変更

F\$VERIFY レキシカル関数を使用すると、コマンド・プロシージャが実行されている間だけチェック機能を無効にすることができます。このとき、ユーザは、プロシージャを実行しながらそのプロシージャの内容を表示することはできません。

チェック機能には、次の 2 つのタイプがあります。

- プロシージャ・チェック
各コマンド行を実行しながら表示する。
- イメージ・チェック
各データ行を処理しながら表示する。

省略時の設定では、SET [NO]VERIFY コマンドと FSVERIFY レキシカル関数は、両方のタイプのチェック機能をオン、オフします。通常は、プロシージャの中のプロシージャおよびイメージ・チェック設定値は同じになっています (両方がオンか、両方がオフになっています)。ただし、設定値を変更する場合には、それぞれのチェック設定値を別々にセーブします。

次の例では、シンボル TEMP を使用して、チェック機能を有効および無効に設定します。

```
$ ! Enable verification
$ !
$ TEMP = FSVERIFY(1)
$ LOOP:
$   INQUIRE FILE "File name"
$   IF FILE .EQS." " THEN EXIT
$   PRINT 'FILE'
$   GOTO LOOP
$ ! Disable verification
$ !
$ TEMP = FSVERIFY(0)
$ EXIT
```

次の例では、チェック機能の設定がセーブされます。

```
$ ! Save each verification state
$ ! Turn both states off
$ SAVE_VERIFY_IMAGE = F$ENVIRONMENT("VERIFY_IMAGE")
$ SAVE_VERIFY_PROCEDURE = FSVERIFY(0)
.
.
.
$ ! Restore original verification states
$ SAVE_VERIFY_IMAGE = FSVERIFY(SAVE_VERIFY_PROCEDURE,-
    SAVE_VERIFY_IMAGE)
```

上記の例で、F\$ENVIRONMENT 関数は、現在のイメージ・チェック設定値を戻し、この値を SAVE_VERIFY_IMAGE シンボルに割り当てています。次に、FSVERIFY 関数が現在のプロシージャ・チェック設定値を戻し、この値を SAVE_VERIFY_PROCEDURE シンボルに割り当てています。FSVERIFY 関数は、イメージ・チェックとプロシージャ・チェックの両方を無効にします。F\$ENVIRONMENT 関数を使用すれば、FSVERIFY によってチェック機能を無効にする前にプロシージャ・チェック設定値を得ることができます。ただし、上記の例に示すように、FSVERIFY を使用して 1 つのコマンド行の中で両方のタスクを行った方がプロシージャの長さが短くてすみます。

プロシージャの終わりに、FSVERIFY 関数で元の設定値 (シンボル SAVE_VERIFY_PROCEDURE と SAVE_VERIFY_IMAGE によって指定される) を復元しています。

注意

時刻印字を使用する場合、チェック機能が使用可能な場合にだけ適用されることに注意してください。時刻印字と SET PREFIX コマンドについての詳細は、『OpenVMS DCL ディクショナリ』または DCL ヘルプを参照してください。

15.2.2 省略時のファイル保護の変更

コマンド・プロシージャの中で省略時のファイル保護を変更したい場合があります。次のコマンド・プロシージャは、プロシージャの実行中に作成されたファイルに関連する省略時の保護を変更します。この場合、終了する前に元の省略時のファイル保護を復元しています。

```
$ SAVE_PROT = F$ENVIRONMENT("PROTECTION")
$ SET PROTECTION = (SYSTEM:RWED, OWNER:RWED, GROUP, WORLD)/DEFAULT
.
.
.
$ SET PROTECTION=('SAVE_PROT')/DEFAULT
$ EXIT
```

この例で、F\$ENVIRONMENT 関数は、SET PROTECTION コマンドの構文を使用して省略時の保護コードを戻しています。このため、SET PROTECTION コマンドと一緒に SAVE_PORT シンボルを使用して元の省略時のファイル保護を復元できます。

15.3 システムについての情報

システムについての情報を得るには、次のレキシカル関数を使用します。

F\$CONTEXT	F\$PID 関数と一緒に使用する場合は選択基準を指定する。 F\$CONTENT 関数と一緒に F\$PID 関数を使用すると、OpenVMS Cluster の任意のノードのプロセスについての情報を得ることができる。
F\$CSID	VMS クラスタ識別番号を戻して、システムの VMS クラスタ・ノード・リストの中の現在の位置を指すようにコンテキスト・シンボルを更新する。
F\$GETQUI	システム・ジョブ・キュー・ファイルに収められたキュー、現在キューの中にあるバッチ・ジョブと印刷ジョブ、フォーム定義、属性定義についての情報を戻す。
F\$GETSYI	ローカル・システムまたは VMS クラスタ内のノード (システムが VMS クラスタの一部である場合) についての情報を戻す。
F\$IDENTIFIER	識別子を指定された形式から数値形式に、または数値形式から指定された形式に変換する。
F\$MESSAGE	状態コードに関連するメッセージ・テキストを戻す。

レキシカル関数を使用しての情報の取得と処理

15.3 システムについての情報

FSPID	ユーザがチェックすることが許容されているプロセスのプロセス識別番号 (PID) を戻す。
F\$TIME	現在の日時を戻す。

15.3.1 OpenVMS Cluster ノード名の決定

使用中のシステムがネットワークまたは OpenVMS Cluster の一部で、多数のノードにログインできる場合には、現在使用中のノードを示すように DCL プロンプトを設定できます。このためには、ログイン・コマンド・プロシージャの中に F\$GETSYI 関数を指定して、ノード名を判別します。次に、SET PROMPT コマンドを使用して、ノード固有のプロンプトを設定します。

プロンプト文字列の中でノード名の一部分だけを使用したい場合には、F\$EXTRACT 関数を使用して、該当する文字を取り出します。文字の取り出しについては、第 15.6.2 項を参照してください。

次の例では、シンボル NODE が F\$GETSYI("NODENAME") として定義され、その後、ノード名がプロンプトとして使用されます。

```
$ NODE = F$GETSYI("NODENAME")
$ SET PROMPT = "'NODE'$ "
.
.
.
```

15.3.2 キューについての情報

F\$GETQUI 関数を使用すると、バッチ・キューと印刷キューについての各種の情報を得ることができます。キューの中のジョブやファイルについての情報を得るには、該当ジョブに対する読み込みアクセス権か SYSPRV または OPER 特権を持っていないわけではありません。

次の例は、バッチ・キュー VAX1_BATCH が終了状態にあるかどうかを判別する方法を示しています。戻される値は真か偽のいずれかです。キューが終了状態にない場合には、コマンド・プロシージャはジョブをキューに登録します。

```
$ QSTOPPED = F$GETQUI("DISPLAY_QUEUE", "QUEUE_STOPPED", "VAX1_BATCH")
$ IF QSTOPPED THEN GOTO NOBATCH
$ SUBMIT/QUEUE=VAX1_BATCH TEST.COM
$ NOBATCH:
.
.
.
```

15.3.3 プロセスについての情報

F\$PID 関数を使用すると、チェックすることが許容されているすべてのプロセスのプロセス識別番号 (PID) を得ることができます。

- WORLD 特権を持つ場合はシステムのすべてのプロセス
- GROUP 特権を持つ場合はグループ内のすべてのプロセス
- GROUP 特権も WORLD 特権も持たない場合は自分のプロセスだけ

PID 番号を入手した後、F\$GETJPI 関数を使用すれば、プロセスに関する特定の情報を入手できます。

次の例は、チェックすることが許容されているプロセスの PID を得て表示する方法を示しています。

```
$ ! 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
```

システムは CONTEXT シンボルを使用して、ポインタを PID のシステム・リストに保持しています。ループを通過するたびに、ポインタを変更して、リストの中の次の PID を指すようにします。すべての PID が表示されると、プロシージャは終了します。

次のプロシージャは、PID とそれぞれのプロセスの UIC を表示します。

```
$ 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
```

WRITE コマンドの中に F\$GETJPI 関数を指定すれば、このコマンド・プロシージャを短くできます。

レキシカル関数を使用しての情報の取得と処理

15.3 システムについての情報

```
$ CONTEXT = ""
$ START:
$ PID = F$PID(CONTEXT)
$ IF PID .EQS. "" THEN EXIT
$ WRITE SYS$OUTPUT "Pid --- 'PID'   Uic --- 'F$GETUPI(PID,"UIC")'"
$ GOTO START
```

15.3.4 F\$CONTEXT レキシカル関数

OpenVMS Cluster 内の任意のノードからプロセスに関する情報を入手するには、F\$CONTEXT 関数を使用します。

次の例では、選択基準を設定するために F\$CONTEXT が 3 回呼び出されています。

```
$!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$CONTEXT ("PROCESS", ctx, "CURPRIV", "SYS$PRV,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$GETUPI(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
```


例を調べる場合には、次のことに注意してください。

- 1 最初の呼び出しでは、検索が OpenVMS Cluster のすべてのノードで実行されることを要求する。
- 2 2 番目の呼び出しでは、ユーザ名が M から始まるプロセス、またはユーザ名が SYSTEM であるプロセスだけを処理することを要求する。
- 3 3 番目の呼び出しでは、現在の特権に SYSPRV (システム特権) と OPER (オペレータ特権) の両方が含まれており、他の特権も持つことができるプロセスだけを選択する。
- 4 “loop”と“endloop”というラベルの間のコマンド行は、F\$PID を継続的に呼び出して、F\$CONTEXT 呼び出しで設定された条件を満足するプロセスを入手する。
- 5 各 PID 番号を検索した後、F\$GETJPI が呼び出され、プロセスで実行中のイメージの名前が戻される。
- 6 最後に、プロシージャはイメージの名前を表示する。
- 7 エラーが発生したり、Ctrl/Y が押された場合には、制御はerrorに渡され、必要に応じてコンテキストは閉じられる。
- 8 シンボル・タイプ PROCESS_CONTEXT のチェックに注意しなければならない。シンボルがこのタイプである場合には、F\$CONTEXT を呼び出すことにより、選択条件を取り消さなければならない。シンボルが PROCESS_CONTEXT タイプでない場合には、選択条件が F\$CONTEXT でまだ設定されていないか、またはエラーが発生するかプロセス・リストの最後に到達するまで、F\$PID に対してシンボルが使用されていた。

15.4 ファイルとデバイスについての情報

ファイルとデバイスについての情報を得るには、次のレキシカル関数を使用します。

F\$DEVICE	指定された選択基準を満たすシステム上のすべてのデバイスのデバイス名を戻す。
F\$FILE_ATTRIBUTES	ファイル属性についての情報を戻す。
F\$GETDVI	指定されたデバイスについての情報を戻す。
F\$PARSE	ファイル指定を解析して、要求された 1 つまたは複数のフィールドを戻す。
F\$SEARCH	ディレクトリでファイルを検索する。

15.4.1 デバイスの検索

システム・サービス\$GETDVIを使用することによって特定のデバイスについての情報を得るには、デバイス名をサービスに提供しなければなりません。デバイス名がわからない場合には、F\$DEVICE レキシカル関数を使用して検索できます。

F\$DEVICE 関数を使用すると、デバイス名、デバイス・クラス、またはデバイス・タイプに基づくワイルドカード検索ができます。デバイス・タイプに基づいて検索を行うには、デバイス・クラスも指定しなければなりません。

コマンド・プロシージャのループの中で F\$DEVICE 関数を使用すると、指定された選択基準に一致するデバイス名が戻されます。F\$DEVICE 関数を実行するたびに、選択基準に一致する次のデバイスが戻されます。デバイス名が戻される順序は一定でないことに注意してください。最後のデバイス名が戻されてから F\$DEVICE 関数を呼び出すと、空の文字列が戻されます。

次の例のコマンド・プロシージャは、ユニット番号が 0 のユニット上のすべての RA60 のデバイス名を表示します。

```
$ START:
$   DEVICE_NAME = F$DEVICE("*0:", "DISK", "RA60")
$   IF DEVICE_NAME .EQS. "" THEN EXIT
$   SHOW SYMBOL DEVICE_NAME
$   GOTO START
```

15.4.2 ディレクトリの中でのファイルの検索

ファイルを処理する際には、あらかじめ F\$SEARCH 関数を使用してそのファイルが存在するかどうかを調べなくてはなりません。たとえば、次のコマンド・プロシージャは、F\$PARSE を使用して、デバイスとディレクトリ文字列を STATS.DAT ファイルに適用しています。次に、F\$SEARCH 関数を使用して、DISK3:[JONES.WORK]に STATS.DAT が存在するかどうか判別します。存在すれば、ファイルを処理し、存在しない場合には、別の入力ファイルを求めるプロンプトを出します。

```
$ 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
```

ファイルの存在を判別した後、F\$PARSE または F\$FILE_ATTRIBUTES 関数を使用すれば、ファイルについての補足情報を得ることができます。

```
$ 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
```

15.4.3 ファイルの古いバージョンの削除

プロシージャの終了後にユーザが必要としないファイルが作成された場合には、プロシージャを終了する前にこれらのファイルを削除またはパージします。最も新しいバージョンを除くすべてバージョンを削除するには、PURGE コマンドを使用します。ファイルの特定のバージョンを削除するには、DELETE コマンドと一緒にバージョン番号を指定します。ファイルのすべてのバージョンを削除するには、DELETE コマンドのバージョン・フィールドにワイルドカード文字を使用します。

コマンド・プロシージャの中で DELETE コマンドを使用するとき、エラー・メッセージが出されないようにするには、F\$SEARCH 関数を使用して、削除する前にそのファイルが存在するかどうかをチェックします。たとえば、特定のモジュールが存在する場合にのみ TEMP.DAT というファイルを作成するコマンド・プロシージャを作成することができます。次のコマンド行は、TEMP.DAT が存在する場合にだけ DELETE コマンドを出します。

```
$ IF F$SEARCH("TEMP.DAT") .NES. "" THEN DELETE TEMP.DAT;*
```

15.5 論理名の変換

論理名を変換するには、次のレキシカル関数を使用します。

F\$LOGICAL	論理名の同値文字列を戻す。
F\$TRNLNM	論理名の同値文字列または要求された属性を戻す。

注意

F\$TRNLNM 関数は、OpenVMS オペレーティング・システムの以前のバージョンで使用されていた F\$LOGICAL に代わるものです。コマンド・プロシージャが現在のシステム手法を使用して論理名を処理するようにするには、F\$LOGICAL の代わりに F\$TRNLNM を使用してください。

場合によっては、コマンド・プロシージャの変数としてシンボルではなく、論理名を使用するとよいことがあります。プログラムは、DCL シンボルにアクセスするより論理名にアクセスする方が簡単だからです。したがって、コマンド・プロシージャから実行するプログラムに情報を渡すには、シンボルを使用して情報を得てから、

DEFINE または ASSIGN コマンドを使用して、シンボルの値を論理名に定義します。

また、F\$TRNLNM 関数を使用して論理名の値を判断してから、その値をシンボルに割り当てることができます。

次の例は、論理名 NAMES が定義されているかどうかをテストします。定義されている場合には、プロシージャは PAYROLL.EXE を実行し、定義されていない場合には、FILE シンボルの値を得て、この値を使用して論理名 NAMES を作成します。PAYROLL.EXE は、論理名 NAMES を使用して従業員名のファイルを表しています。

```
$ ! 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
.
.
.
```

このコマンド・プロシージャは、PAYROLL プログラムで使用される論理名を定義します。

```
$ DEFINE NAMES DISK4:[JONES]EMPLOYEE_NAMES.DAT
$ RUN PAYROLL
.
.
.
$ WRITE SYS$OUTPUT "Finished processing ",F$TRNLNM("NAMES")
```

プロシージャの終わりに、ファイルが処理されたことを示すメッセージが WRITE コマンドによって表示されます。

15.6 文字列の処理

文字列を処理するには、次のレキシカル関数を使用します。

F\$CVTIME	時刻文字列についての情報を戻す。
F\$EDIT	文字列を編集する。
F\$ELEMENT	各要素が区切り文字によって分けられている文字列から要素を取り出す。
F\$EXTRACT	文字列の一部を取り出す。
F\$FAO	出力文字列を書式化する。
F\$LENGTH	文字列の長さを決定する。

FSLOCATE 文字列の中の文字または部分文字列を探して、オフセットを戻す。

15.6.1 文字列または文字の有無の判別

文字列を調べる理由の 1 つに、文字列の中に文字 (または部分文字列) が存在するかどうかの判別があります。このためには、F\$LENGTH と F\$LOCATE 関数を使用します。F\$LOCATE 関数が戻す値が F\$LENGTH 関数が戻す値と等しければ、探している文字は存在しません。

次のプロシージャは、バージョン番号を含むファイル名を要求しています。バージョン番号が存在するかどうかを判別するために、ユーザが入力したファイル名の中に、バージョン番号の前に置くセミコロン(;)があるかどうかをテストします。

```
$ INQUIRE FILE "Enter file (include version number)"
$ IF F$LOCATE(";", FILE) .EQ. F$LENGTH(FILE) THEN -
    GOTO NO_VERSION
.
.
.
```

F\$LOCATE 関数は、セミコロンのオフセットを戻します。オフセットは 0 で始まるため、セミコロンが文字列の最初の文字であれば、F\$LOCATE 関数は整数 0 を戻します。文字列の中にセミコロンがなければ、F\$LOCATE 関数は、文字列の最後の文字のオフセットより 1 だけ大きいオフセットを戻します。この値は、番号 1 から始まる文字列の長さに対応する F\$LENGTH が戻す長さと同じです。

15.6.2 文字列の一部の取り出し

文字列の一部を取り出すには、F\$EXTRACT 関数または F\$ELEMENT 関数のどちらかを使用します。定義されたオフセットから始まる部分文字列を取り出すには F\$EXTRACT 関数を使用し、2 つの区切り文字の間の文字列の一部を取り出すには、F\$ELEMENT 関数を使用します。どちらの関数を使用するかを選択するためには、解析する文字列の一般形式がわかっていなければなりません。

ファイル指定または時刻文字列を解析するには、F\$EXTRACT または F\$ELEMENT を使用する必要はありません。代わりに、F\$PARSE または F\$CVTIME を使用して、ファイル指定または時刻文字列の一部を取り出します。

グループ名を取り出すのと同時に、グループ名の長さを判断することもできます。

文字列にその文字列の異なる部分を区切る区切り文字が含まれている場合には、F\$ELEMENT 関数を使用して、必要な部分を取り出すことができます。F\$ELEMENT を使用すると、文字列の中でカンマで区切られた部分を取り出すことにより、異なるアクセス・タイプを入手できます。システム・アクセスを判断するには、最初の要素を入手します。オーナー・アクセスを判断するには、2 番目の要素を入手します。以下も同様です。F\$ELEMENT 関数を使用する場合には、要素番号は 0

から始まります。この理由から、4番目の要素を指定するときは、整数3を使用します。

次のコマンド・プロシージャは、F\$EXTRACT関数を使用してUICのグループ部分を取り出します。このとき、ユーザのUICグループに応じて、異なるコマンド・セットを実行できます。

```
$ UIC = F$USER()  
$ GROUP_LEN = F$LOCATE(", ", UIC) - 1  
$ GROUP = F$EXTRACT(1, GROUP_LEN, UIC)  
$ GOTO 'GROUP'_SECTION  
  
.  
.  
.  
$ WRITERS_SECTION:  
.  
.  
.  
$ MANAGERS_SECTION:  
.  
.  
.
```

最初に、F\$USER関数によってUICを判別します。次に、F\$LOCATEを使用してコンマのオフセットを検出して、グループ名の長さを判別します。コンマによってUICのグループとユーザ部分が区切られているからです。左括弧とコンマの間にあるものがグループ名の部分になります。たとえば、UIC [WRITERS, SMITH]のグループ名はWRITERSです。

長さを判別した後、プロシージャはF\$EXTRACT関数によってグループの名前を取り出します。名前はオフセット1で始まり、コンマの前の文字で終わります。最後に、該当するラベルに処理を移します。

次の例では、保護コードの各アクセス・タイプがカンマでどのように区切られるかを示しています。

```
$ UIC = F$USER()  
$ GROUP = F$EXTRACT(1, F$LOCATE(", ", UIC) - 1, UIC)  
$ GOTO 'GROUP'_SECTION
```

次の例では、保護コードの各アクセス・タイプがカンマでどのように区切られるかを示しています。

```
$ PROT = F$ENVIRONMENT("PROTECTION")  
$ SHOW SYMBOL PROT  
PROT = "SYSTEM=RWED, OWNER=RWED, GROUP=RE, WORLD"
```

この例のコマンドは、省略時の保護コードからワールド・アクセスの部分(4番目の要素)を取り出します。

```
$ PROT = F$ENVIRONMENT("PROTECTION")
$ WORLD_PROT = F$ELEMENT(3,"",PROT)
.
.
.
```

この例では、F\$ELEMENT 関数は、3 番目のコンマと文字列の終わりの間にあるすべてを戻すため、省略時の保護がワールド・ユーザの読み込みアクセスを許容している場合には、文字列 " WORLD=R" が戻されます。

ワールド・アクセス文字列を得たら、さらにそれを調べる必要があります。たとえば、次のようにします。

```
$ 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
.
.
.
```

15.6.3 出力文字列の書式化

WRITE コマンドを使用すると、文字列をレコードに書き込むことができます。レコード内のカラムをそろえるには、FSFAO 関数を使用してレコード・フィールドを定義し、これらのフィールドにプロセス名とユーザ名を格納します。FSFAO 関数を使用する場合には、制御文字列を使用してレコード内のフィールドを定義します。その後、これらのフィールドに格納する値を指定します。

レコード内のフィールドを書式化する別の方法として、文字列オーバーレイを使用する方法があります。しかし、FSFAO 関数の方が文字列オーバーレイより強力です。FSFAO 関数を使用すると、広範囲にわたる出力操作を実行できます。

この例に示したコマンド・プロシージャでは、WRITE コマンドを使用して、システムのプロセスのプロセス名と PID 番号を表示します。

```
$ ! 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
```

次に示すように、WRITE コマンドからの出力では、プロセス名とユーザ名の間に 5 つのスペースが挿入されますが、桁そろえは行われません。

レキシカル関数を使用しての情報の取得と処理

15.6 文字列の処理

Process Name	PID
MARCHESAND	2CA0049C
TRACTMEN	2CA0043A
FALLON	2CA0043C
ODONNELL	2CA00453
PERRIN	2CA004DE
CHAMPIONS	2CA004E3

この例の中では、コマンド・プロシージャは、F\$FAO 関数を使用して、16 文字フィールドと 12 文字フィールドを定義しています。F\$FAO 関数は、最初のフィールドにプロセス名を収めてから、スペースはスキップして、2 番目のフィールドに PID を収めます。

```
$ ! 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
```

ここで、プロシージャを実行すると、桁がそろえられます。

Process Name	PID
MARCHESAND	2CA0049C
TRACTMEN	2CA0043A
FALLON	2CA0043C
ODONNELL	2CA00453
PERRIN	2CA004DE
CHAMPIONS	2CA004E3

次の例は、オーバーレイを使用して、RECORD シンボルの最初の 16 文字 (オフセット 0 から始まる) にプロセス名を収め、次の 12 文字 (オフセット 17 から始まる) に PID を収めます。

```
$ ! 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
```


このプロシージャでは、F\$FAO 関数で作成したのと同じ書式が生成されます。

Process Name	PID
MARCHESAND	2CA0049C
TRACTMEN	2CA0043A
FALLON	2CA0043C
ODONNELL	2CA00453
PERRIN	2CA004DE
CHAMPIONS	2CA004E3

15.7 データ・タイプの処理

データを文字列から整数に、整数から文字列に変換するには、次のレキシカル関数を使用します。

F\$CVSI	文字列からビット・フィールドを取り出して、符号付きの値である結果を整数に変換する。
F\$CVUI	文字列からビット・フィールドを取り出して、符号なしの値である結果を整数に変換する。
F\$INTEGER	文字列式を整数に変換する。
F\$STRING	整数式を文字列に変換する。
F\$TYPE	シンボルのデータ・タイプを判別する。

15.7.1 データ・タイプの変換

整数と文字列の間で変換を行うには、F\$INTEGER 関数と F\$STRING 関数を使用します。たとえば、次のコマンド・プロシージャはデータ・タイプを変換します。ユーザが文字列を入力すると整数の同値が表示され、整数を入力すると文字列の同値が表示されます。GOTO 文の中でラベル名を使用するときには、F\$TYPE 関数をどのように使用したらよいかに注意してください。F\$TYPE は、シンボルのデータ・タイプに応じて、"STRING"または"INTEGER"を戻します。

```
$ 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
```

15.7.2 式の評価

INQUIRE や READ などのコマンドは、文字列データしか受け付けません。これらのコマンドを使用して、整数式として評価したいデータを得るには、F\$INTEGER 関数を使用してこのデータを変換してから評価します。

EXP シンボルを F\$INTEGER 関数の引数として使用する場合には、EXP シンボルの前後に一重引用符(')を置かなければなりません。こうすると、シンボル置換の最初のフェーズで EXP の値が置換されます。

次の例では、F\$INTEGER 関数を使用して、整数式を評価します。

```
$ INQUIRE EXP "Enter integer expression"
$ RES = F$INTEGER('EXP')
$ WRITE SYS$OUTPUT "Result is",RES
```

このコマンド・プロシージャからの出力は次のようになります。

```
Enter integer expression: 9 + 7
Result is 16
```

"9 + 7"の値が置換されます。F\$INTEGER 関数が引数の "9 + 7"を処理する場合、式を評価して正しい結果を戻します。

15.7.3 シンボルの有無の判別

シンボルが存在するかどうかを判別するには、F\$TYPE 関数を使用します。F\$TYPE 関数は、シンボルが未定義であると、空の文字列を戻します。

```
.
.
.
$ IF F$TYPE(TEMP) .EQS. "" THEN TEMP = "YES"
$ IF TEMP .EQS. "YES" THEN GOTO TEMP_SEC
.
.
.
```

このプロシージャは、TEMP シンボルが前に定義されているかどうかをテストします。定義済みの場合には、TEMP の正しい値が保持され、TEMP が定義されていない場合には、IF 文により "YES"の値が TEMP に割り当てられます。

プロセスとバッチ・ジョブ

プロセスとは、システムとの会話を可能にするために、OpenVMS オペレーティング・システムで作成される環境です。プロセスには、独立プロセス (他のプロセスから独立しているプロセス) とサブプロセス (別のプロセスに従属して存在し、資源を得ているプロセス) があります。メイン・プロセスは親プロセスとも呼ばれ、独立プロセスの 1 つです。本章では、次のことについて説明します。

- プロセス・コンテキストの解釈
- サブプロセスの使用
- 仮想ターミナルで切断されたプロセスへの接続
- バッチ・ジョブの操作

プロセスの作成方法

ユーザが次のいずれかのタスクを実行すると、システムはプロセスを作成します。

- ログイン
システムはユーザがログインするごとにそのユーザ用にプロセスを作成する。
- バッチ・ジョブの登録
システムはバッチ・ジョブを受け取るとそのジョブ用にプロセスを作成する。バッチ・ジョブが完了すると、システムはプロセスを削除する。
- サブプロセスの生成
ユーザが SPAWN コマンドを使用すると、システムはプロセスを作成する。
- プログラムの実行
/DETACHED 修飾子または/UIC=uic 修飾子を使用してプログラムを実行すると、システムはプロセスを作成する。

16.1 プロセス・コンテキストへの割り込み

特権、シンボル、論理名などプロセスが使用する属性を、プロセス・コンテキストといいます。システムは、プロセス固有の特性をユーザ登録ファイル(UAF)から取得します。UAF は、システムへのアクセスを許可されたユーザをリストして、それぞれのユーザのプロセスの特性を定義します。通常、UAF の管理はシステム管理者が行います。システムは一度に 1 つずつプログラム (イメージまたは実行可能イメージと

プロセスとバッチ・ジョブ

16.1 プロセス・コンテキストへの割り込み

も呼ばれる)を実行しますが、これらのプログラムはプロセスと呼ばれる環境内で実行されます。

現在のプロセスのプロセス・コンテキストを表示するには、SHOW PROCESS/ALL コマンドを使用します。

次の例は、プロセス・コンテキストのサンプルです。

```
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] 6
Base priority: 4 7
Default file spec: DISK1:[CLEAVER] 8
Number of Kthreads: 1

Devices allocated: ALPHAI$VTA2195:

Process Quotas: 9
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: 10
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: 11
GROUP may affect other processes in same group
TMPMBX may create temporary mailbox
OPER operator privilege
NETMBX may create network device

Process rights: 12
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      13
  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: 14
  CLEAVER (*)
```

例を確認するときは、次のことに注意してください。

1 現在の日時

SHOW PROCESS/ALL コマンドを実行した日時。

2 ユーザ名

プロセスに関連するアカウントに割り当てられたユーザ名。

3 プロセス識別番号 (PID)

システムによってプロセスに割り当てられた固有の番号。SHOW PROCESS コマンドは、PID を 16 進数で表示する。

4 プロセス名

プロセスに割り当てられた名前。プロセス名は一意であるため、あるアカウントでログインした最初のプロセスにはユーザ名が割り当てられ、同じアカウントでログインしたそれ以降のプロセスにはターミナル名が割り当てられる。プロセス名は、DCL の SET PROCESS/NAME コマンドで変更できる。

5 ユーザ識別コード (UIC)

プロセスに関連するアカウントに割り当てられたグループとメンバの番号または文字 (たとえば、[PERSONNEL,RODGERS])。UIC により、ユーザが属するグループが分かる。同じグループに属するユーザ同士は、他のグループのユーザとの間より自由にファイルやシステム資源を共用できる。

6 優先順位

プロセスの現在の優先順位。

7 省略時のファイル指定

現在のデバイスとディレクトリ。現在の省略時の値は、DCL の SET DEFAULT コマンドによって変更することができる。

8 プロセス・クォータ

プロセスに関連するクォータ (上限)。クォータは、SHOW PROCESS コマンドに /QUOTAS または /ALL 修飾子を付けて実行することにより確認することができる。

9 会計情報

プロセスのメモリの使用状況と CPU 時間に関する情報。この情報は、時々刻々と更新されており、SHOW PROCESS コマンドに/ACCOUNTING または/ALL 修飾子を付けて実行することにより、最新の情報を得ることができる。

10 プロセス特権

プロセスに付与された特権。特権は、特定のシステム・アクティビティを特定のユーザしか実行できないように制限する。特権は、SHOW PROCESS コマンドに/PRIVILEGES または/ALL 修飾子を付けて実行することにより確認することができる。

11 プロセス・ライト

アクセス制御リスト (ACL) 保護と一緒に使用されるシステムが定義する識別子。識別子は、ACL の中でユーザを指定する手段である。ACL は、ファイル、デバイス、メールボックスなどのオブジェクトのユーザに許容または拒否されるアクセスの種類を定義するセキュリティ・ツール。

12 プロセスの動的メモリ空間

プロセスの現在の動的メモリの使用状況。動的メモリは、イメージが実行されるときにシステムによってそのイメージに割り当てられる。プロセスがそのメモリを必要としなくなった場合には、システムはそれを別のプロセスに割り当てる。この情報は、SHOW PROCESS コマンドに/MEMORY または/ALL 修飾子を付けて実行することにより得られる。

13 階層構造の中のプロセス

親プロセスに属するサブプロセスのリスト。現在のプロセスはその末尾にアスタリスク(*)が付けられる。このリストは、SHOW PROCESS コマンドに/SUBPROCESSES または/ALL 修飾子を付けて実行することにより得られる。

16.2 独立プロセスの使用

独立プロセスは、親プロセスの種類にしたがって、会話型または非会話型のいずれかになります。DCL の RUN コマンドまたは Create Process システム・サービス (\$CREPRC) にユーザが指定する引数に応じて、ユーザまたは OpenVMS オペレーティング・システムがログインを行います。RUN も \$CREPRC も SYS\$SYSTEM で LOGINOUT.EXE イメージを実行します。

16.3 サブプロセスの使用

SPAWN コマンドを使用すると、現在のプロセスのサブプロセスを作成できます。このサブプロセスの中で、ユーザはシステムと会話して、サブプロセスからログアウトして親プロセスに戻ったり、親プロセスとサブプロセスを切り替えたりできます。一度に 1 つのプロセスしか実行できません。

システム上のそれぞれのユーザは、ジョブ階層構造によって表現されます。ジョブ階層構造とは、メイン・プロセスを頂点とし、それに属するすべてのプロセスとサブプロセスからなる階層をいいます。サブプロセスは親プロセスに従属し、親プロセスが終了すると削除されます。省略時の設定では、サブプロセスは、親プロセスの名の後にアンダスコアと固有の番号を付けた名前を持ちます。たとえば、親プロセス名が DOUGLASS の場合、サブプロセスには DOUGLASS_1、DOUGLASS_2 というような名前が付けられます。

16.3.1 Spawn タスクによるサブプロセスの使用

タスクに割り込み、2 番目のタスクを実行してから、元のタスクに戻るには、Ctrl/Y を使用してタスクに割り込んだ後、2 番目のタスクを実行するサブプロセスを生成し、そのサブプロセスを終了してから、CONTINUE コマンドを入力して元のタスクに戻ります。省略時の設定では、ユーザがサブプロセスを作成すると、親プロセスはハイバネート状態になり、サブプロセスの中で DCL レベルでユーザに制御が渡されます。省略時のディレクトリは親プロセスの現在のディレクトリである。たとえば、Ctrl/Y を押して EVE 編集セッションに割り込んだ場合には、CONTINUE コマンドを入力してから Ctrl/W を押して画面を再表示します。

16.3.2 サブプロセスの使用による複数のタスクの実行

最初のタスクの実行を続けながら、2 番目のタスクを実行するには、SPAWN/NOWAIT コマンドでサブプロセスを作成します。SPAWN/NOWAIT は、非会話型のバッチに似たサブプロセスを生成し、入力を必要としないコマンドを実行するときだけに使用します。

親プロセスとサブプロセスの両方が同時に実行されるため、双方がターミナルを制御しようとし、混乱が生じるのを避けるため、次の修飾子やパラメータも指定してください。

- /OUTPUT 修飾子

サブプロセスが、ターミナルではなく、指定されたファイルに出力を書き込むように指定する。

- SPAWN コマンド・パラメータまたは/INPUT 修飾子

サブプロセスが、ターミナルから入力を読み込むのではなく、指定されたコマンドを実行するように指定する。

SPAWN コマンドの/INPUT 修飾子を指定すると、サブプロセスは非会話型のプロセスとして生成され、重大エラーまたはファイルの終端を検出すると終了します。DCL レベルでは、Ctrl/Z がファイルの終端標識として扱われます。

16.3.3 サブプロセスの作成

ユーザが作成するそれぞれのプロセスは一意であるため、通常は、あるプロセスで実行されたコマンドが他のプロセスに影響することはありません。ただし、ターミナルの制御はプロセス間で受け渡されるため、ターミナル属性に影響するコマンド（たとえば、SET TERMINAL）は、そのターミナルを制御するすべてのプロセスに影響を及ぼします。たとえば、1つのプロセスがエコー表示を禁止し、それを復元せずに終了した場合には、次にターミナルを制御するプロセスでもエコー表示は禁止されたままになります。変更したターミナル属性は、SET TERMINAL コマンドによって再設定します。

次の例では、ユーザは、Ctrl/Y を押してコマンド・イメージ (TYPE コマンド) に割り込んでサブプロセスを生成し、そのサブプロセスを終了してから元のプロセスに戻ります。

```
$ 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.
```

16.3.4 サブプロセスの終了

SPAWN コマンドによって作成されたサブプロセスを終了するには、次のいずれかのコマンドを使用します。

- LOGOUT

LOGOUT コマンドでサブプロセスを終了すると、そのサブプロセスは（それが作成した他のサブプロセスとともに）削除され、親プロセスに戻る。

- ATTACH

ATTACH コマンドでサブプロセスを終了すると、そのサブプロセスはハイバネート状態になり、ターミナルの制御は指定されたプロセスに渡される。ATTACH コマンドのパラメータとしてプロセス名を指定するか、ATTACH コマンドの/IDENTIFIER 修飾子の値としてプロセス識別番号 (PID) を指定しなければならない。

次の例は、DOUGLASS_1 サブプロセスを終了して、DOUGLASS プロセスに接続する方法を示しています。

```
$ 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
```

16.3.5 サブプロセス・コンテキスト

サブプロセス・コンテキストは、サブプロセスが親プロセスから受け継ぐ環境です。省略時の設定では、サブプロセスは、省略時の値、特権、シンボル、論理名、制御文字、メッセージ形式、チェック状態、キー定義を受け継ぎます。これらの項目が集まってサブプロセスの環境が形成されます。

次の項目は、親プロセスからは受け継がれません。

- プロセス識別番号 (PID)

システムは、作成したそれぞれのプロセスに固有の PID を割り当てる。

- プロセス名

省略時の設定では、サブプロセス名は、親プロセスの名前の後にアンダスコアと整数を付けたもの。別のプロセス名を指定するには、SPAWN コマンドに/PROCESS 修飾子を使用する。プロセス名は一意でなければならない。

- 作成したコマンド

SET COMMAND コマンドを使用して親プロセスで定義したコマンドは、サブプロセスにコピーされない。作成されたコマンドをサブプロセスで使用するには、SET COMMAND を使用してサブプロセスに対してそのコマンドを作成しなければならない。

- 特権を付与する特権

サブプロセスを生成する場合、プロセス・コンテキストには、親プロセスの特権は含まれるが、特権を付与するための特権は含まれない。たとえば、Mail でサブプロセスを生成して、特権付き操作を実行したい場合には、Mail を起動する前に親プロセスで正しい特権を設定しておかなければならない。

サブプロセスが親プロセスのコンテキスト項目の一部を受け継がないようにするには、次の SPAWN 修飾子を使用します。

SPAWN コマンド修飾子	禁止または変更される項目
/CARRIAGE_CONTROL, /PROMPT	DCL プロンプト
/NOCLI	CLI (コマンド言語インタプリタ。省略時の設定では DCL。)
/NOKEYPAD	キーボード定義
/NOLOGICAL_NAMES	論理名
/NOSYMBOL	シンボル

/SYMBOL 修飾子と/LOGICAL_NAMES 修飾子は、システムが定義するシンボル (\$SEVERITY や \$STATUS など) やシステムが定義する論理名 (SYS\$COMMAND や SYS\$OUTPUT など) には影響を及ぼしません。

論理名やシンボルをサブプロセスにコピーすると、時間 (数秒) がかかるため、サブプロセスの中で論理名やシンボルを使用する予定がなければ、SPAWN コマンドに/NOLOGICAL_NAMES と/NOSYMBOL 修飾子を使用するとよいでしょう。サブプロセスを頻繁に使用する場合には、ATTACH コマンドを使用すると、最も効率良くサブプロセスで作業を開始したり終了したりできます。この方法を使用すると、システムが新しいサブプロセスを作成するのを待つことなく、親プロセスとサブプロセスとの間で制御を迅速に切り換えることができます。

16.4 仮想ターミナルでの切断されたプロセスの接続

仮想ターミナルが使用可能な状態にあるときに、モデム回線の接続が断たれた場合、プロセスは切断された仮想ターミナル・プロセスとしてシステム上にアクティブ状態のまま存在し続けます。ユーザは、システム管理者によって指定された時間 (省略時の値は 900 秒、すなわち 15 分) 内にプロセスに再接続しなければなりません。この時間内にプロセスに再接続しないと、プロセスは削除されます。

注意

UIC (利用者識別コード) に対応する仮想ターミナル・プロセスにしか接続できません。

16.4.1 ターミナルの切断

ターミナルが切断されるのは次のような状況の場合です。

- ホストとターミナル間でモデム・シグナルを消失した。
- TT2\$M_SECURE 属性を設定した状態でターミナル上で BREAK キーを押した。
- DCL の DISCONNECT コマンドを入力した。
- DCL の CONNECT/CONTINUE コマンドを入力した。

プロセスが切断されている場合、古いプロセスに再接続して、切断される前の状態に戻することもできます。ログインすると、次のようなプロンプトが出されます。

```
You have the following disconnected process:
Terminal  Process name  Image name
VTA52:    RWOODS        (none)
Connect to above listed process [YES]:
```

プロンプトに対して Return キーを押すか、Yes と入力すると、DCL の CONNECT /CONTINUE コマンドが自動実行されたかのように現在のプロセスからログアウトできます。No と入力するか、応答が遅れると (応答時間が時間切れになると)、新しいプロセスにログインしたままになります。以後、古いプロセスに接続することはできません。

複数の切断されたセッションがある場合には、再接続したい仮想ターミナルの名前を求めるプロンプトが出されます。表示されたセッションのどれにも接続したくない場合には、No と入力します。

16.4.2 切断されたプロセスの削除

一定の間隔が経過した後、システムは切断されたプロセスを自動的に削除します。しかし、次に示すように、切断されたプロセスから直接ログアウトすれば、システム資源を節約できます。

手順	操作
1	他に切断されたジョブがあるかどうかを判断するには、DCL の SHOW USERS コマンドを入力する。
2	DCL の CONNECT/LOGOUT コマンドを入力して、現在のプロセスからログアウトする。存在する最後のプロセスに到達するまで、関連する各仮想ターミナル (先頭に VTA がついたターミナル) に接続する。
3	DCL の LOGOUT コマンドを入力する。

16.4.3 切断されたプロセスの管理

仮想ターミナルを使用すると、一度に複数の切断されたプロセスを管理することができます。ただし、仮想ターミナルにログインしているときは、物理ターミナルが切断

されていることに注意してください。現在の仮想ターミナル・プロセスに関連する物理ターミナル以外のデバイスに入出力要求を向けると、待ち状態になります。待ち状態のプロセスは、時間切れになると終了します。ただし、入出力要求を受けるはずの物理ターミナルに再接続すれば、待ち状態に入ったポイントからプロセスの実行が継続されます。それぞれのプロセスにそのコンテキストに関連する名前を付けておくと、プロセスの再接続が簡単になります。

たとえば、SMITH というユーザがファイルを編集するプロセスを実行する場合、SET PROCESS/NAME コマンドを使用して、プロセスを SMITH_EDIT と命名できます。こうすれば、SMITH は、編集を続けるためにどのプロセスに接続すればよいのかが簡単に分かります。

システム管理者は、システム全般でまたはターミナル単位で仮想ターミナルの使用を制限できます。

16.5 バッチ・ジョブ

バッチ・ジョブは、非会話型のプロセスです。バッチ・ジョブは固有のプロセスで実行されるため、同時に異なる処理を行う 2 つ以上のプロセスを指定することができます。たとえば、以下のようなことができます。

- ユーザがタスクを会話形式で実行しているときに、システムはプログラムやコマンド・プロシージャをバッチ・モードで実行することができる。
- 実行時間の長いコマンド・プロシージャを実行することができる。
- コマンド・プロシージャやプログラムを数時間後に実行することができる。
- 特定のプログラムを優先順位を下げて実行したりするときに、バッチ・ジョブを使用することができる。

16.5.1 バッチ・ジョブの登録

ユーザがバッチ・ジョブをキューに登録すると、使用中のアカウント属性とプロセスの属性を持つ独立プロセスが作成されます。システムはそのプロセスからジョブを実行して、ジョブが完了するとプロセスを削除します。また、システム・ログイン・コマンド・プロシージャ (SYLOGIN.COM) とパーソナル・ログイン・コマンド・プロシージャ (LOGIN.COM) を実行してから、バッチ・ジョブの中のコマンド・プロシージャを実行します。これらのプロシージャを実行すると、出力がログ・ファイルに書き込まれます。バッチ・ジョブが完了すれば、ログ・ファイルを印刷したり、ディレクトリの 1 つにセーブしたりできます。

バッチ・モードでジョブを実行するには、DCL の SUBMIT コマンドを入力してジョブをバッチ・キュー (実行を待っているバッチ・ジョブのリスト) に登録します。キューに登録されたジョブは、省略時のバッチ・キューである SYSS\$BATCH に渡され、実行を待っているジョブのキューの終わりに追加されます。先に登録されていたジョ

ブが完了してから、新たに登録されたジョブが実行されます。OpenVMS システムでは、同時に実行できるバッチ・ジョブ数は、システム管理者がバッチ・キューを作成するときに指定します。省略時の設定では、SUBMIT コマンドはファイル・タイプに.COM を使用します。

たとえば、次のコマンドは、JOB1.COM を SYS\$BATCH に登録します。

```
$ SUBMIT JOB1
Job JOB1 (queue SYS$BATCH, entry 651, started on SYS$BATCH)
```

システムは、ジョブの名前、ジョブが登録されているキュー、ジョブに割り当てられたエントリ番号を表示します。ジョブがバッチ・キューに登録されると、DCL プロンプトが出されます。DCL コマンド (たとえば、DELETE/ENTRY) の中でバッチ・ジョブを参照しなければならない場合には、ジョブ・エントリ番号を使用して参照します (ジョブ・エントリ番号は、SHOW ENTRY コマンドによって得られます)。複数のプロシージャを 1 つのバッチ・ジョブに登録した場合、エラーまたは回復不可能な (重大) エラー状態が生じていずれかのプロシージャが終了するとバッチ・ジョブも終了します。

バッチ・ジョブは、ユーザがバッチ・キューに登録したときにすぐ実行を開始する必要はありません。バッチ・ジョブの実行を開始する時刻を指定するには、SUBMIT /AFTER コマンドを入力します。次の例では、ジョブは午後 11 時 30 分以降にキューに登録されます。

```
$ SUBMIT/AFTER=23:30 JOB1.COM
```

コマンド・プロシージャをキューに登録してバッチ実行を行う場合には、システムは、バージョン番号を含む、コマンド・プロシージャの完全なファイル指定をセーブします。コマンド・プロシージャをキューに登録した後でコマンド・プロシージャを更新すると、バッチ・ジョブは、新しいバージョンではなく、キューに登録したコマンド・プロシージャのバージョンを実行します。

ログイン時の省略時の値は、通常、コマンド・プロシージャに示されたファイルにアクセスするのに必要な省略時の値ではないため、次のいずれかの方法を使用して、正しいファイルにアクセスするようにします。

- 完全ファイル指定を使用する。

コマンド・プロシージャの中のファイルを参照したり、ファイルをコマンド・プロシージャに渡す場合には、ファイル指定の一部としてデバイス名とディレクトリ名を指定する。

- SET DEFAULT コマンドを使用する。

コマンド・プロシージャの中のファイルを参照する前に、コマンド・プロシージャの始めに SET DEFAULT コマンドを使用して、正しいデバイスとディレクトリを指定する。

バッチ・ジョブを実行すると、出力をログ・ファイルに書き込みます。省略時の設定では、ログ・ファイルはキューに登録するコマンド・プロシージャと同じ名前を持ち、ファイル・タイプは.LOG になります。ジョブが完了すると、システムはログ・ファイルを印刷し、ディレクトリから削除します。ログ・ファイルのセーブについては、第 16.5.3 項を参照してください。

ログイン・コマンド・プロシージャ内部のバッチ・ジョブのチェック

バッチ・ジョブをキューに登録するたびに、システムは、パーソナル・ログイン・コマンド・プロシージャを実行します。F\$MODE()レキシカル関数を使用してバッチ・ジョブをテストすれば、バッチ・ジョブを実行するときにログイン・コマンド・プロシージャの一部を指定したり省略したりできます。

たとえば、バッチ・ジョブ専用のコマンド、論理名、シンボルを含むログイン・コマンド・プロシージャの中のセクションがあるとします。この場合、このセクションに BATCH_COMMANDS のラベルを付けてから、ログイン・コマンド・プロシージャの始めに次のコマンドを指定します。

```
IF F$MODE() .EQS. "BATCH" THEN GOTO BATCH_COMMANDS
.
.
.
```

ユーザがバッチ・ジョブをキューに登録したときに、システムがログイン・コマンド・プロシージャの中のコマンドを実行しないようにするには、プロシージャの始めに次のコマンドを指定します。

```
IF F$MODE() .NES. "INTERACTIVE" THEN EXIT
```

このコマンドは、ログイン・コマンド・プロシージャのどこにでも指定できます。ユーザがバッチ・ジョブをキューに登録すると、システムは、前のコマンドが置かれたポイントまでログイン・コマンド・プロシージャを実行します。

複数のコマンド・プロシージャの登録

SUBMIT コマンドを入力する場合、1 つのジョブの中で複数のコマンド・プロシージャを実行するように指定できます。/NAME 修飾子で名前を指定する場合を除いて、SUBMIT コマンドは最初のコマンド・プロシージャの名前をジョブ名として使用します。エラーによってジョブの中のいずれかのコマンド・プロシージャが終了すると、ジョブ全体が終了します。

バッチ・ジョブを実行する場合、最初のプロシージャ (UPDATE.COM) の操作コンテキストは 2 番目のプロシージャ (SORT.COM) には残されません。すなわち、システムは、UPDATE.COM によって作成されたローカル・シンボルを削除してから、SORT.COM を実行します。ただし、グローバル・シンボルは残されます。

1つのジョブの中では、それぞれのコマンド・プロシージャに別々のパラメータを指定することはできません。

次の例では、SUBMIT コマンドはバッチ・ジョブを作成します。このバッチ・ジョブは、UPDATE.COM を実行した後、SORT.COM を実行します。

```
$ SUBMIT UPDATE,SORT
Job UPDATE (queue SYS$BATCH, entry 207) started on SYS$BATCH
```

次の例は、同じ2つのパラメータをUPDATE.COMとSORT.COMに渡します。

```
$ SUBMIT UPDATE, SORT/PARAMETERS = -
_$ (DISK1:[ACCOUNT.BILLS]DATA.DAT, DISK2:[ACCOUNT]NAME.DAT)
$ Job UPDATE (queue SYS$BATCH, ENTRY 208) started on SYS$BATCH
```

16.5.2 バッチ・ジョブへのデータの受け渡し

バッチ・ジョブの省略時の入力ストリーム(SYS\$INPUT)は、実行中のコマンド・プロシージャです。独立プロセスがバッチ・ジョブを実行しているため、(会話形式で実行するコマンド・プロシージャの場合のように)SYS\$INPUTをターミナルに再定義することはできません。バッチ・ジョブに入力を渡すには、次のいずれかの方法を使用します。

- コマンド・プロシージャ自体にデータを指定する。

コマンド・プロシージャにデータを指定するには、コマンドまたはイメージの後の行にデータを置きます。

- SYS\$INPUTを一時的にファイルとして定義する。

SYS\$INPUTを一時的にファイルとして定義するには、DEFINE/USER_MODE コマンドを使用します。

- コマンド・プロシージャをキューに登録して実行するときに、コマンド・プロシージャにパラメータを渡す。

コマンド・プロシージャにパラメータを渡すには、バッチ・ジョブをキューに登録するときに/PARAMETERS 修飾子を使用します。

1つのジョブの中ではそれぞれのコマンド・プロシージャに別々のパラメータは指定できません。別々のパラメータを渡す必要がある場合には、それぞれのコマンド・プロシージャに対してSUBMIT コマンドを使用します。

次の例では、データ行がイメージAVERAGE.EXEに渡されます。

プロセスとバッチ・ジョブ

16.5 バッチ・ジョブ

```
$! Execute AVERAGE.EXE
$ RUN AVERAGE
647
899
532
401
$ EXIT
```

次の例では、SYS\$INPUT はファイルとして一時的に定義されます。

```
$ DEFINE/USER_MODE SYS$INPUT STATS.DAT
$ RUN AVERAGE
$ EXIT
```

次の例では、ファイル EMPLOYEES.DAT のパラメータがコマンド・プロシージャ CHECKS.DAT に渡されます。

```
$ SUBMIT/PARAMETERS=(DISK1:[PAYROLL]EMPLOYEES.DAT) CHECKS
Job CHECKS (queue SYS$BATCH, entry 209) started on SYS$BATCH
```

注意

SHOW QUEUE/FULL コマンドは、バッチ・キューの中のジョブについての完全な情報を表示します。この中には、プロシージャに渡すパラメータも含まれるため、バッチ・ジョブにはパスワードなどの機密情報は渡さないでください。

16.5.3 バッチ・ジョブ出力の制御

省略時の設定では、ログ・ファイルはバッチ・ジョブの中の最初のコマンド・プロシージャと同じ名前を持ち、ファイル・タイプは.LOG になります。システムは、バッチ・ジョブからの出力を 1 分おきにログ・ファイルに書き込みます。これとは異なる時間間隔を指定するには、コマンド・プロシージャに SET OUTPUT_RATE コマンドを指定します。

システムがログ・ファイルに書き込みを行っているときに、EDT エディタを使用してログ・ファイルを読み込もうとすると、ファイルが別のユーザによってロックされていることを示すメッセージが出されます。この場合は、数秒待ってから、もう一度読み込んでください。ただし、EVE エディタを使用すると、同じ状況でもバッチ・ジョブのログ・ファイルを読み込めます。EDIT/TPU/READ_ONLY とログ・ファイルの名前を指定すれば、EVE コマンドを使用してログ・ファイルの中を参照でき、ファイルを変更しても、その内容はセーブされません。/READ_ONLY 修飾子を指定せずに、何らかの方法でログ・ファイルを変更すると、バッチ・ジョブは終了します。

バッチ・ジョブはユーザのユーザ名でログインし、そのユーザのパーソナル・ログイン・コマンド・プロシージャを実行するプロセスなので、バッチ・ジョブからの出力にはパーソナル・ログイン・コマンド・プロシージャの内容も含まれます。バッチ・ジョブからの出力には、バッチ・ジョブ・ログ・ファイルに書き込まれたすべての情報(コマンド・プロシージャ出力、エラー・メッセージなど)と完全なログアウト・メッセージも含まれています。パーソナル・ログイン・コマンド・プロシージャがバッチ・ログ・ファイルに書き込まれないようにするには、パーソナル・ログイン・コマンド・プロシージャの始めに次のコマンドを追加します。

```
$ IF F$MODE() .EQS. "BATCH" THEN SET NOVERIFY
```

省略時の設定では、ログ・ファイル名は、ジョブをキューに登録したときの名前になります。また、省略時の設定では、ログ・ファイルは、ファイル・タイプ.LOGを持ち、ログイン時の省略時の値によって指定されたデバイスとディレクトリを想定します。ジョブをキューに登録するときに別のログ・ファイル名を指定するには、SUBMIT コマンドに/LOG_NAME 修飾子を使用します。

バッチ・ジョブ・ログ・ファイルには、SYSS\$OUTPUT と SYSS\$ERROR へのすべての出力が収められます。また、省略時の設定では、コマンド・プロシージャで実行されるすべてのコマンド行も収められます。コマンド行が印刷されないようにするには、コマンド・プロシージャの中で SET NOVERIFY コマンドまたは F\$VERIFY レキシカル関数のいずれかを使用します。ジョブが完了すると、システムは(長いシステム・ログアウト・メッセージ形式を使用して)ジョブ終了情報をログ・ファイルに書き込みます。

SET VERIFY コマンドが有効な場合には、SET PREFIX コマンドを使用してそれぞれのコマンド行を時刻印字すれば、それぞれのコマンド行が実行される正確な時間を知ることができます。

バッチ・ジョブを正しく実行できない場合には、ログ・ファイルを調べて、コマンド・プロシージャに障害が生じたポイントと障害を引き起こしたエラー状態を判別します。

ログ・ファイルのセーブ

ログ・ファイルをセーブするには、/KEEP または/NOPRINTER 修飾子を使用します。/KEEP 修飾子は印刷後にログ・ファイルをセーブし、/NOPRINTER 修飾子はログ・ファイルを印刷せずにセーブします。いずれの修飾子も指定しないと、省略時のアクションがとられて、ログ・ファイルは省略時の印刷キューである SYSS\$PRINT に登録され、印刷後は削除されます。/KEEP 修飾子と/NORPINTER 修飾子は、ログ・ファイルを省略時のログイン・ディレクトリにセーブします。ログ・ファイルはバッチ・ジョブの中の最初のコマンド・プロシージャと同じ名前を持ち、ファイル・タイプは.LOG です。別のファイル名やディレクトリ名、またはその両方を指定するには、/LOG_FILE 修飾子を使用します。ログ・ファイルのファイル指定を変更してセ

ーブするには、/LOG_FILE に加えて、/KEEP または/NOPRINTER のいずれかを指定しなければなりません。

次の例では、ログ・ファイルが DISK2:[JONES.RESULTS]UPDATE.LOG という名前のファイルにセーブされます。

```
$ SUBMIT/LOG_FILE=DISK2:[JONES.RESULTS]/NOPRINTER -  
_ $ DISK2:[JONES.RESULTS]UPDATE
```

ログ・ファイルの読み込み

TYPE コマンドを使用すると、ログ・ファイルを読み込んでバッチ・ジョブがどの程度完了したかを判別できます。ただし、システムがログ・ファイルに書き込みを行っているときにログ・ファイルを表示しようとすると、ファイルが別のユーザによってロックされていることを示すメッセージが出されます。この場合には、数秒待ってから、もう一度試してください。

バッチ・ジョブ・ログにすべてのコマンド出力を指定する

通常、プログラムをコンパイルしたりリンクしたり実行したりするバッチ・ジョブ・コマンド・プロシージャは、コンパイラ・リストやリンカ・マップなどの印刷済みの出力を生成します。このようなファイルの印刷済みコピーを生成するには、バッチ・ジョブ・コマンド・プロシージャにファイルを印刷するのに必要な PRINT コマンドを指定します。

バッチ・ジョブ・ログに、コンパイラまたはリンカ出力ファイルの印刷済みリストを含む、コマンド・プロシージャからのすべての出力を収めたい場合には、次のいずれかを実行します。

- コマンド・プロシージャの中で PRINT コマンドの代わりに TYPE コマンドを使用する。TYPE コマンドは SYSS\$OUTPUT に書き込みを行う。バッチ・ジョブでは、SYSS\$OUTPUT はバッチ・ジョブ・ログ・ファイルに等しく定義される。
- 修飾子または該当するコマンドを使用して、出力を SYSS\$OUTPUT に向ける。
この方法を使用する場合、ログ・ファイルをセーブしない限り、出力ファイルはディスクにセーブされないことに注意してください。

このコマンド・プロシージャの処理が終了すると、バッチ・ジョブ・ログ、コンパイラ・リスト、リンカ・マップの3つの出力リストが生成されます。

```
$ FORTRAN/LIST BIGCOMP  
$ PRINT BIGCOMP.LIS  
$ LINK/MAP/FULL BIGCOMP  
$ PRINT BIGCOMP.MAP
```

次の例では、修飾子を使用して、出力を SYSSOUTPUT に送信する方法を示しています。

```
$ FORTRAN/LIST=SYS$OUTPUT BIGCOMP  
$ LINK/MAP=SYS$OUTPUT/FULL BIGCOMP
```

バッチ・ジョブの中でこれらのコマンドが実行されると、コンパイラとリンカからの出力ファイルは直接ログ・ファイルに書き込まれます。

16.5.4 バッチ・ジョブ属性の変更

ジョブをキューに登録しても、ジョブの実行開始前であれば、SET ENTRY または SETQUEUE/ENTRY コマンドと一緒に適切な修飾子を使用すれば、ジョブに関連する属性を変更できます。

次の例では、バッチ・ジョブをバッチ・キューに保留した状態で、バッチ・ジョブの名前を変更するために使用できる 2 種類の方法を示しています。

```
$ SET QUEUE/ENTRY=209/NAME=NEW_NAME SYS$BATCH  
  
$ SET ENTRY 209 /NAME=NEW_NAME
```

どちらのコマンドも、ジョブ番号 209 の名前を NEW_NAME に変更します。

SET ENTRY または SET QUEUE/ENTRY コマンドで変更できる内容の一部を次に示します。修飾子の完全なリストは、『OpenVMS DCL デクシオナリ』を参照してください。SUBMIT コマンドで指定できるほとんどの修飾子が SET ENTRY と SET QUEUE/ENTRY コマンドでも使用できます。

次のような変更を加えることができます。

- ジョブの実行開始を遅らせる。
ジョブを実行できるようになる時間を指定するには/AFTER 修飾子を、明示的に解放するまでジョブを保留するには/HOLD 修飾子を使用する。
- ジョブを解放する。
/HOLD または/AFTER 修飾子でキューに登録されたジョブを解放するには、/NOHOLD または/RELEASE 修飾子を使用する。
- ジョブを別のキューへ移動する。
ジョブを実行するキューを変更するには、/REQUEUE 修飾子を使用する。
- 実行属性を変更する。
ワーキング・セットの省略時の値、ワーキング・セットの超過値、ワーキング・セットのサイズ、ジョブ・スケジューリング優先順位、CPU 時間制限などの実行属性を変更する。
- ジョブに渡すパラメータを変更する。

パラメータを変更するには、/PARAMETERS 修飾子を使用する。

16.5.5 SUBMIT コマンド修飾子

バッチ・ジョブの特性を制御するために SUBMIT コマンドに対して指定できる修飾子は次のとおりです。省略時のワーキング・セット、ワーキング・セットの超過値、ワーキング・セット・サイズ、ジョブ・スケジューリング優先順位、CPU 時間制限など、実行属性も指定できます。

/AFTER

バッチ・ジョブを実行できるようになる時間を指定する。ジョブは、指定された時間になるまでバッチ・キューに留まる。ユーザがジョブを明示的に解放するまでキューの中に保留するには、/HOLD 修飾子を使用する。保留されているジョブを解放するには、SET ENTRY/RELEASE コマンドを使用する。

/NAME

バッチ・ジョブの名前を指定する。これを指定しないと、ジョブ名の省略時の値は、ジョブの中の最初の (または唯一の) コマンド・プロシージャのファイル名になる。

/NOTE

SHOW QUEUE/FULL コマンドを実行した結果に表示されるメッセージ文字列を指定する。ジョブについての情報をオペレータやシステム管理者に伝達できるようになる。

/NOTIFY

ジョブの完了を通知するように要求する。バッチ・ジョブの実行が終了すると、システムはメッセージをターミナルに送信する。

/PARAMETERS

パラメータをバッチ・ジョブに渡す。

/NOPRINTER または/KEEP

バッチ・ジョブ・ログ・ファイルをセーブする。

/QUEUE

バッチ・ジョブを SYSS\$BATCH 以外のキューに送信する。リモート・ノードにあるコマンド・プロシージャを実行するには、/REMOTE 修飾子を使用する。このとき、ジョブはリモート・ノードの SYSS\$BATCH に送信される。

/RESTART

ジョブの実行中にシステムに障害が生じた場合に、ユーザがジョブを再開できるようにする。

/RETAIN

バッチ・ジョブが完了してもバッチ・ジョブをキューの中に残す。ジョブの完了状態を調べるには、SHOW QUEUE または SHOW ENTRY コマンドを使用する。

16.5.6 バッチ・キューの中のジョブの表示

ジョブがバッチ・ジョブ・キューに登録されていれば、SHOW ENTRY コマンドまたはSHOW QUEUE コマンドを使用してジョブの状態を監視できます。キューの中にジョブが存在しない場合には、次のメッセージが表示されます。

```
$ SHOW QUEUE BOSTON_BATCH
Batch queue BOSTON_BATCH, on BOSTON::
```

ジョブに関する完全な情報を表示するには、SHOW ENTRY または SHOW QUEUE コマンドに/FULL 修飾子を指定します。キューの中の他のジョブの状態を表示するには、SHOW QUEUE/ALL コマンドを使用します。

次の例では、エントリ番号 999 が表示されます。

```
$ 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

次の例では、/FULL 修飾子が指定されているため、BOSTON_BATCH に関する統計情報と、ジョブ番号 999 に関連する属性が表示されます。

```
$ 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)

次の例では、SHOW QUEUE/ALL コマンドを使用して、BOSTON_BATCH キュー内のすべてのジョブを表示します。

```
$ 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            O'DONNELL     Executing
```

特権ユーザの場合を除いて、自分のアカウントでキューに登録されたジョブの情報しか表示されないことに注意してください。

16.5.7 バッチ・ジョブの削除と終了

バッチ・ジョブは、実行前でも実行中でも削除できます。保留中またはすでにバッチ・キューで実行されている項目を削除するには、DELETE/ENTRY コマンドを使用します。自分がキューに登録しなかったジョブを削除するには、特別な特権が必要です。DELETE/ENTRY コマンドの結果としてジョブが終了した場合は、ログ・ファイルは印刷されず、ディレクトリからも削除されません。

DELETE/ENTRY コマンドを使用してジョブを終了した場合は、オペレーティング・システムの通常のジョブ終了アクティビティが先取りされるため、異常終了として扱われます。この結果、バッチ・ジョブ・ログには、ジョブ時間と会計情報を要約した標準ログアウト・メッセージは収められません。ただし、明示的に EXIT コマンドまたは STOP コマンドを出して終了した場合やこのいずれかのコマンドを (現在の ON 状態の結果として) 暗黙に実行して終了した場合には、正常終了とみなされます。正常終了後、オペレーティング・システムは、正しいランダウンおよび会計情報プロシージャを実行します。

次のコマンドは SYS\$BATCH のなかでジョブ・エントリ 210 を削除しています。

```
$ DELETE/ENTRY=210 SYS$BATCH
```

16.5.8 バッチ・ジョブの再開

バッチ・ジョブの実行中にシステムが異常終了した場合には、ジョブは完了しません。システムが回復してキューが再開されると、実行中のジョブは強制終了され、キューの中の次のジョブが実行されます。ただし、バッチ・ジョブをキューに登録するときに /RESTART 修飾子を指定しておけば、ジョブが終了する前にシステムがクラッシュしても、システムが復旧した後、そのジョブは再実行されます。

省略時の設定では、バッチ・ジョブは最初の行から再実行されます。コマンド・プロシージャにシンボルを追加すれば、別の場所から実行を再開できますが、これについては第 13 章と第 14 章を参照してください。

システム・クラッシュ後のジョブの再開に加えて、ジョブを明示的に終了した後もジョブを再開できます。ジョブを終了した後、同じキューまたは別のキューでジョブを再開するには、STOP/QUEUE/REQUEUE/ENTRY コマンドを使用します。

この例に示したコマンドは、SYSS\$BATCH のジョブ 212 を停止し、SYSS\$BATCH に再登録します。

```
$ STOP/QUEUE/REQUEUE/ENTRY=212 SYS$BATCH
```

このコマンドを入力するには、ジョブ 212 が SUBMIT コマンドに/RESTART 修飾子を指定して登録されていなければなりません。バッチ・ジョブを 2 度目に実行する場合、システムは、グローバル・シンボル BATCH\$RESTART を使用してどこからジョブの実行を開始するかを判別します。

16.5.9 バッチ・ジョブ実行の同期化

コマンド・プロシージャの中で SYNCHRONIZE コマンドと WAIT コマンドを使用すれば、プロシージャを待ち状態に置くことができます。SYNNCHRONIZE コマンドを指定すると、プロシージャは指定されたジョブが完了するのを待ちますが、WAIT コマンドを指定すると、指定された時間が経過するのを待ちます。

SYNCHRONIZE コマンドでジョブ名を指定する場合は、同期化するジョブがユーザ名と対応している必要があります。通常ジョブは、そのジョブをキューに登録するプロセスのユーザ名と対応しています。異なるユーザ用のジョブを同期化するには、SYNCHRONIZE コマンドに/ENTRY 修飾子を付けて、ジョブ・エントリ番号を指定しなければなりません。

たとえば、2 つのジョブを同時にキューに登録して共同作業を行う場合、一方のジョブには次のコマンドを指定できます。

```
$ SYNCHRONIZE BATCH25
```

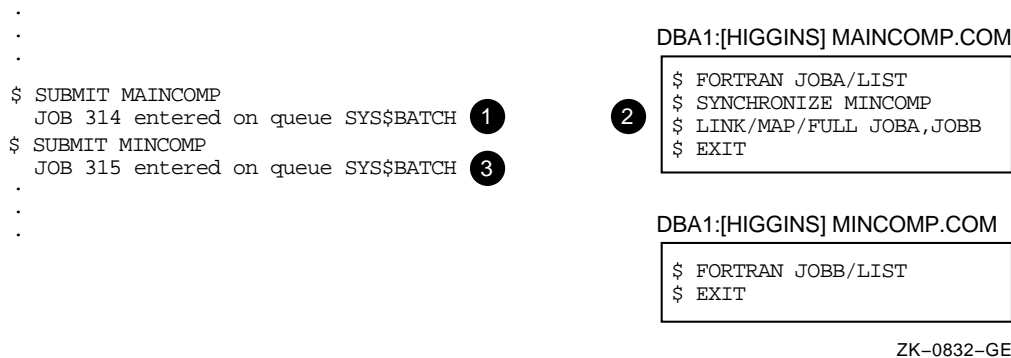
このコマンドを実行すると、ジョブ名 BATCH25 を持つジョブの実行が完了するまで、コマンド・プロシージャの実行を継続できません。

この SYNCHRONIZE コマンドは、ジョブ 454 が完了するまで、現在のコマンド・プロシージャを待ち状態に置きます。

```
$ SYNCHRONIZE/ENTRY=454
```

図 16-1 は、並行して実行するためにキューに登録されたが、正しく実行するためには同期化しなければならないコマンド・プロシージャの例です。それぞれのプロシージャが大きなソース・プログラムをコンパイルします。

図 16-1 バッチ・ジョブ実行の同期化



例を確認するときは、次のことに注意してください。

- 1 それぞれの SUBMIT コマンドで 2 つのジョブをキューに登録することが要求されている。最初のプロセスが作成される。
- 2 FORTRAN コマンドの実行後、SYNCHRONIZE コマンドが実行される。ジョブ 315 が現在のジョブまたは待ち状態のジョブの場合には、ジョブ 314 は次のコマンドを実行しない。
- 3 ジョブ 315 の実行が完了すると、ジョブ 314 は次のコマンドを実行する。

16.5.10 WAIT コマンドの使用法

WAIT コマンドは、コマンド・プロシージャがディスクやテープ・ドライブなどの共有システム資源へアクセスしなければならない場合に便利です。

次の例は、テープ・ドライブの割り当てを要求するプロシージャです

```

$ 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 コマンドが正常に終了しないと、プロシージャは待ち状態になります。5 分後に、要求を再試行します。

ALLOCATE 要求の後の IF コマンドは、\$STATUS の値をチェックします。\$STATUS の値が正常終了を示している場合には、コマンド・プロシージャは継続されますが、そうでない場合には、WAIT コマンドが実行されます。WAIT コ

マンドは、5 分の時間間隔を指定しています。5 分待機した後、次のコマンド GOTO が実行され、要求が繰り返されます。このプロシージャは、正常終了するまで、またはバッチ・ジョブが削除されるか終了されるまで、ループを繰り返し、デバイスを割り当てようとしています。

文字セット

DEC 各国語文字セット (MCS) は、Digital Equipment Corporation で作成され使用された 00 ~ FF の 16 進数で表現される文字の定義から成ります。DEC MCS は、7 ビットの (00 ~ 7F の 16 進数で表現される) ASCII 文字セットと、80 ~ FF の 16 進数で表現される 8 ビット文字のセットの 2 つに分類されます。DEC MCS は、Digital Equipment Corporation で作成され販売されたソフトウェアのほとんどのユーザにとって使い慣れている文字セットです。

Unicode Standard Character Set (UCS-2) は、Unicode Consortium によって定義された、0000 ~ FFFF の値で表現される 16 ビット文字のセットです。

ISO Latin-1 文字セットは、00 ~ FF の 16 進数で表現される 8 ビット文字の UCS-2 の定義です。ISO Latin-1 文字セットの定義は、80 ~ FF の 16 進数から成る DEC MCS の定義とは少し異なります。

表 A-1 には、DEC 各国語文字セット (MCS) が示されています。表 A-1 は、2 種類の文字セットでの文字の違いを示し、図 A-1 は、異なっている文字を示しています。

表 A-2 には、DCL 文字セットが示されています。

Unicode (UCS-2) 文字セットについての詳細は、Unicode Consortium から発行されている『The Unicode Standard』を参照してください。

表 A-1 DEC 各国語文字セット

16 進数のコード	MCS 文字 または短縮形	DEC 各国語文字名
ASCII 制御文字 ¹		
00	NUL	空文字
01	SOH	ヘッダの始点 (Ctrl/A)
02	STX	テキストの始点 (Ctrl/B)
03	ETX	テキストの終端 (Ctrl/C)
04	EOT	転送の終端 (Ctrl/D)
05	ENQ	問い合わせ (Ctrl/E)
06	ACK	肯定応答 (Ctrl/F)

¹ ALTMODE 文字および DELETE 文字 (10 進数の 125, 126, および 127) も、制御文字です。

(次ページに続く)

表 A-1 (続き) DEC 各国語文字セット

16 進数のコード	MCS 文字 または短縮形	DEC 各国語文字名
ASCII 制御文字 ¹		
07	BEL	ベル (Ctrl/G)
08	BS	バックスペース (Ctrl/H)
09	HT	水平タブ (Ctrl/I)
0A	LF	ライン・フィード (Ctrl/J)
0B	VT	垂直タブ (Ctrl/K)
0C	FF	フォーム・フィード (Ctrl/L)
0D	CR	キャリッジ・リターン (Ctrl/M)
0E	SO	シフト・アウト (Ctrl/N)
0F	SI	シフト・イン (Ctrl/O)
10	DLE	データ・リンク・エスケープ (Ctrl/P)
11	DC1	デバイス制御 1 (Ctrl/Q)
12	DC2	デバイス制御 2 (Ctrl/R)
13	DC3	デバイス制御 3 (Ctrl/S)
14	DC4	デバイス制御 4 (Ctrl/T)
15	NAK	否定応答 (Ctrl/U)
16	SYN	同期アイドル (Ctrl/V)
17	ETB	転送ブロックの終端 (Ctrl/W)
18	CAN	取り消し (Ctrl/X)
19	EM	媒体の終端 (Ctrl/Y)
1A	SUB	置換 (Ctrl/Z)
1B	ESC	エスケープ
1C	FS	ファイル区切り文字
1D	GS	グループ区切り文字
1E	RS	レコード区切り文字
1F	US	ユニット区切り文字
ASCII 特殊文字および数値文字		
20	SP	スペース
21	!	感嘆符
22	"	引用符 (二重引用符)
23	#	ポンド記号
24	\$	ドル記号
25	%	パーセント記号
26	&	アンパサンド
27	'	アポストロフィ (一重引用符)

¹ALTMODE 文字および DELETE 文字 (10 進数の 125, 126, および 127) も、制御文字です。

(次ページに続く)

表 A-1 (続き) DEC 各国語文字セット

16 進数のコード	MCS 文字 または短縮形	DEC 各国語文字名
ASCII 特殊文字および数値文字		
28	(左括弧
29)	右括弧
2A	*	アスタリスク
2B	+	正符号
2C	,	コンマ
2D	–	ハイフンまたは負符号
2E	.	ピリオドまたは小数点
2F	/	スラッシュ
30	0	ゼロ
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	:	コロン
3B	;	セミコロン
3C	<	不等号 (左辺は右辺より小さい)
3D	=	等号
3E	>	不等号 (左辺は右辺より大きい)
3F	?	疑問符
ASCII 英文字		
40	@	アットマーク
41	A	大文字の A
42	B	大文字の B
43	C	大文字の C
44	D	大文字の D
45	E	大文字の E
46	F	大文字の F
47	G	大文字の G
48	H	大文字の H

(次ページに続く)

表 A-1 (続き) DEC 各国語文字セット

16 進数のコード	MCS 文字 または短縮形	DEC 各国語文字名
ASCII 英文字		
49	I	大文字の I
4A	J	大文字の J
4B	K	大文字の K
4C	L	大文字の L
4D	M	大文字の M
4E	N	大文字の N
4F	O	大文字の O
50	P	大文字の P
51	Q	大文字の Q
52	R	大文字の R
53	S	大文字の S
54	T	大文字の T
55	U	大文字の U
56	V	大文字の V
57	W	大文字の W
58	X	大文字の X
59	Y	大文字の Y
5A	Z	大文字の Z
5B	[左大括弧
5C	\	バックスラッシュ
5D]	右大括弧
5E	^	サーカンフレックス
5F	_	アンダースコア
60	`	低アクセント
61	a	小文字の a
62	b	小文字の b
63	c	小文字の c
64	d	小文字の d
65	e	小文字の e
66	f	小文字の f
67	g	小文字の g
68	h	小文字の h
69	i	小文字の i
6A	j	小文字の j
6B	k	小文字の k

(次ページに続く)

表 A-1 (続き) DEC 各国語文字セット

16 進数のコード	MCS 文字 または短縮形	DEC 各国語文字名
ASCII 英文字		
6C	l	小文字の l
6D	m	小文字の m
6E	n	小文字の n
6F	o	小文字の o
70	p	小文字の p
71	q	小文字の q
72	r	小文字の r
73	s	小文字の s
74	t	小文字の t
75	u	小文字の u
76	v	小文字の v
77	w	小文字の w
78	x	小文字の x
79	y	小文字の y
7A	z	小文字の z
7B	{	左中括弧
7C		縦線
7D	}	右中括弧 (ALTMODE)
7E	~	チルダ (ALTMODE)
7F	DEL	削除 (DELETE)
制御文字		
80		[予約領域]
81		[予約領域]
82		[予約領域]
83		[予約領域]
84	IND	索引
85	NEL	次の行
86	SSA	選択領域の始点
87	ESA	選択領域の終端
88	HTS	水平タブの設定
89	HTJ	行揃えを使用した水平タブの設定
8A	VTS	垂直タブの設定
8B	PLD	行単位での部分的な下方向への移動
8C	PLU	行単位での部分的な上方向への移動

(次ページに続く)

表 A-1 (続き) DEC 各国語文字セット

16 進数のコード	MCS 文字 または短縮形	DEC 各国語文字名
制御文字		
8D	RI	逆方向の索引
8E	SS2	シングル・シフト 2
8F	SS3	シングル・シフト 3
90	DCS	デバイス制御文字列
91	PU1	プライベート使用 1
92	PU2	プライベート使用 2
93	STS	転送状態の設定
94	CCH	文字の取り消し
95	MW	メッセージの待機
96	SPA	保護領域の始点
97	EPA	保護領域の終端
98		[予約領域]
99		[予約領域]
9A		[予約領域]
9B	CSI	制御シーケンス・イントロデューサ
9C	ST	文字列終了文字列
9D	OSC	オペレーティング・システム・コマンド
9E	PM	プライバシー・メッセージ
9F	APC	アプリケーション
その他の文字		
A0		[予約領域] ²
A1	ı	逆感嘆符
A2	¢	セント記号
A3	£	貨幣のポンド記号
A4		[予約領域] ²
A5	¥	円記号
A6		[予約領域] ²
A7	§	セクション記号
A8	¤	一般貨幣記号 ²
A9	©	著作権記号
AA	ª	女性形序数指示子
AB	«	左角引用符
AC		[予約領域] ²
AD		[予約領域] ²

²ISO Latin-1 では別の文字です。図 A-1 を参照してください。

(次ページに続く)

表 A-1 (続き) DEC 各国語文字セット

16 進数のコード	MCS 文字 または短縮形	DEC 各国語文字名
その他の文字		
AE		[予約領域] ²
AF		[予約領域] ²
B0	°	温度記号
B1	±	正負符号
B2	²	スーパースクリプト 2
B3	³	スーパースクリプト 3
B4		[予約領域] ²
B5	μ	マイクロ記号
B6	¶	段落記号, 段落標
B7	·	中黒
B8		[予約領域] ²
B9	¹	スーパースクリプト 1
BA	♂	男性形序数指示子
BB	»	右角引用符
BC	¼	小数部四分の一
BD	½	小数部二分の一
BE		[予約領域] ²
BF	¿	逆疑問符
C0	À	低アクセント付きの大文字の A
C1	Á	鋭アクセント付きの大文字の A
C2	Â	サーカンフレックス付きの大文字の A
C3	Ã	チルダ付きの大文字の A
C4	Ä	ウムラウト付きの大文字の A (分音符号)
C5	Å	リング付きの大文字の A
C6	Æ	二重母音化した大文字の AE
C7	Ç	セディーユ付きの大文字の C
C8	È	低アクセント付きの大文字の E
C9	É	鋭アクセント付きの大文字の E
CA	Ê	サーカンフレックス付きの大文字の E
CB	Ë	ウムラウト付きの大文字の E (分音符号)
CC	Ì	低アクセント付きの大文字の I
CD	Í	鋭アクセント付きの大文字の I
CE	Î	サーカンフレックス付きの大文字の I
CF	Ï	ウムラウト付きの大文字の I (分音符号)
D0		[予約領域] ²

²ISO Latin-1 では別の文字です。図 A-1 を参照してください。

(次ページに続く)

表 A-1 (続き) DEC 各国語文字セット

16 進数のコード	MCS 文字 または短縮形	DEC 各国語文字名
		その他の文字
D1	Ñ	チルダ付きの大文字の N
D2	Ò	低アクセント付きの大文字の O
D3	Ó	鋭アクセント付きの大文字の O
D4	Ô	サーカンフレックス付きの大文字の O
D5	Õ	チルダ付きの大文字の O
D6	Ö	ウムラウト付きの大文字の O (分音符号)
D7	Œ	合字の大文字の OE ²
D8	Ø	スラッシュ付きの大文字の O
D9	Û	低アクセント付きの大文字の U
DA	Ú	鋭アクセント付きの大文字の U
DB	Û	サーカンフレックス付きの大文字の U
DC	Ü	ウムラウト付きの大文字の U (分音符号)
DD	Ÿ	ウムラウト付きの大文字の Y (分音符号)
DE		[予約領域] ²
DF	ß	ドイツ語固有のエスツェット
E0	à	低アクセント付きの小文字の a
E1	á	鋭アクセント付きの小文字の a
E2	â	サーカンフレックス付きの小文字の a
E3	ã	チルダ付きの小文字の a
E4	ä	ウムラウト付きの小文字の a (分音符号)
E5	å	リング付きの小文字の a
E6	æ	二重母音の小文字の ae
E7	ç	セディーユ付きの小文字の c
E8	è	低アクセント付きの小文字の e
E9	é	鋭アクセント付きの小文字の e
EA	ê	サーカンフレックス付きの小文字の e
EB	ë	ウムラウト付きの小文字の e (分音符号)
EC	ì	低アクセント付きの小文字の i
ED	í	鋭アクセント付きの小文字の i
EE	î	サーカンフレックス付きの小文字の i
EF	ï	ウムラウト付きの小文字の i (分音符号)
F0		[予約領域] ²
F1	ñ	チルダ付きの小文字の n
F2	ò	低アクセント付きの小文字の o
F3	ó	鋭アクセント付きの小文字の o

²ISO Latin-1 では別の文字です。図 A-1 を参照してください。

(次ページに続く)

表 A-1 (続き) DEC 各国語文字セット

16 進数のコード	MCS 文字 または短縮形	DEC 各国語文字名
		その他の文字
F4	ô	サーカンフレックス付きの小文字の o
F5	õ	チルダ付きの小文字の o
F6	ö	ウムラウト付きの小文字の o (分音符号)
F7	œ	合字の小文字の oe ²
F8	ø	スラッシュ付きの小文字の o
F9	ù	低アクセント付きの小文字の u
FA	ú	鋭アクセント付きの小文字の u
FB	û	サーカンフレックス付きの小文字の u
FC	ü	ウムラウト付きの小文字の u (分音符号)
FD	ÿ	ウムラウト付きの小文字の y (分音符号) ²
FE		[予約領域] ²
FF		[予約領域] ²

²ISO Latin-1 では別の文字です。図 A-1 を参照してください。

図 A-1 Differences Between DEC Multinational Character Set and ISO Latin-1 Character Set

Hex Code	MCS Char or Abbrev.	DEC Multinational Character Name	Isolatin-1 Char	Isolatin-1 Character Name
A0		[reserved]		nonbreaking space
A4		[reserved]	¤	currency sign
A6		[reserved]		broken vertical bar
A8	¤	currency sign	¨	spacing diaeresis
AC		[reserved]	¬	not sign
AD		[reserved]	—	soft hyphen
AE		[reserved]	®	registered trademark sign
AF		[reserved]	—	spacing macron
B4		[reserved]	´	spacing acute
B8		[reserved]	¸	spacing cedilla
BE		[reserved]	¾	fraction three quarters
D0		[reserved]	Ð	Latin capital letter eth
D7	Œ	uppercase OE ligature	×	multiplication sign
DE		[reserved]	Þ	Latin capital letter thorn
F0		[reserved]	ø	Latin small letter eth
F7	œ	lowercase oe ligature	÷	division sign
FD	ÿ	lowercase y with umlaut, (diaeresis)	ý	Latin small letter y acute
FE		[reserved]	þ	Latin small letter thorn
FF		[reserved]	ÿ	Latin small letter y diaeresis

VM-0128A-AI

表 A-2 DCL 文字セット

シンボル	名前	意味
@	アットマーク	コマンド・プロシージャ・ファイルの内容をコマンド入力ストリームに収める。
:	コロソ	ファイル指定の中のデバイス名の区切り文字。ダブルコロソ(::)はノード名区切り文字。コロソは修飾子区切り文字としても使用でき、修飾子名とその値とを分割する。
/	スラッシュ	修飾子の接頭辞。
+	プラス記号	パラメータのセパレータ。一部のコマンドでは、パラメータの連結子として働く。文字列連結演算子、単項プラス記号、数値式の中の補足演算子としても認識される。
,	コンマ	パラメータまたは引数リストのリスト要素のセパレータ。
-	ハイフン	継続文字。文字列分割演算子、単項マイナス記号、数値式の中の減算演算子、およびディレクトリ検索ワイルドカード文字としても認識される。
()	括弧	引数リストのリスト区切り文字。数値式の中で演算順序を示すときにも使用される。
[]	大括弧	ファイル指定の中のディレクトリ名の区切り文字。山括弧に等しい。
<>	山括弧	ファイル指定の中のディレクトリ名の区切り文字。大括弧に等しい。
?	疑問符	ヘルプ文字。
&	アンパサンド	実行時置換演算子。これ以外の場合には、予備の特殊文字。
\	バックスラッシュ	予備の特殊文字。
=	等号	修飾子値の区切り文字。修飾子名と引数とを分割する。等号(=)は、シンボル定義での割り当て文としても利用される。
^	サーカンフックス	予備の特殊文字。
#	番号記号	予備の特殊文字。
*	アスタリスク	ファイル指定の中のワイルドカード文字。数値式の中の乗算演算子ならびにシンボル定義の中の短縮形区切り文字としても使用される。
'	一重引用符	置換演算子。
.	ピリオド	ファイル指定の中のファイル・タイプとバージョン番号区切り文字。サブディレクトリ区切り文字としても使用される。
;	セミコロソ	ファイル指定の中のバージョン番号区切り文字。
%	パーセント記号	ファイル指定の中のワイルドカード文字。基数演算子としても使用される。
!	感嘆符	コメントを示す。
"	二重引用符	リテラル文字列区切り文字。

コマンド・プロシージャの例

ここでは、第 13 章、第 14 章、および第 15 章で説明した概念と手法について実際のコマンド・プロシージャを示しながら説明します。1 つの節で 1 つのコマンド・プロシージャを解説します。コマンド・プロシージャごとに次の内容を示します。

- プロシージャの概要
- プロシージャの例
- プロシージャで使用される概念と手法に関する説明
- プロシージャの実行例の結果

B.1 CONVERT.COM コマンド・プロシージャ

このコマンド・プロシージャは、(これ以降の時間の) 絶対時刻をデルタ時間に変換し、現在の時刻とユーザが指定した時刻との差を示します。このプロシージャは、FSTIME と FSCVTIME レキシカル関数の使用方法と、割り当て文を使用して算術計算を行ったりシンボル値を連結する方法を示します。

例: 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)
```

コマンド・プロシージャの例

B.1 CONVERT.COM コマンド・プロシージャ

```
$ !
$ ! Restore default error handling and message format
$ ON ERROR THEN EXIT
$ SET MESSAGE'SAVE_MESSAGE'
$ !
$ IF F$LENGTH(P1) .NE. 5 .OR. -
    F$LOCATE(":",P1) .NE. 2 -
    THEN GOTO BADTIME
3
$ !
$ ! 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 -
    MINUTES_TO_WAIT = 24*60 + FUTURE_TIME - CURRENT_TIME
8
$ !
$ ! 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 コマンド・プロシージャの説明

- 1 パラメータが省略されたのか、またはパラメータとして入力された値が疑問符(?)なのかをチェックする。いずれの場合も、プロシージャは TELL ラベルに分歧する。
- 2 F\$CVTIME 関数を使用して、時間値が有効な 24 時間形式の時刻を示しているかどうかをチェックする。F\$CVTIME は、入力時間が有効でないと警告メッセージを戻す。F\$CVTIME 関数がエラーを戻す場合には、省略時の ON アクションを変更して、BADTIME ラベルに制御を渡す。

プロシージャは、F\$ENVIRONMENT 関数を使用して現在のメッセージ設定値をセーブしてから、警告メッセージやエラー・メッセージが表示されないようにメッセージ形式を設定する。時間値をチェックしたら、省略時の ON 状態とメッセージ形式を復元する。

- 3 パラメータの形式をチェックする。次の形式の時間値でなければならない。

hh:mm

IF コマンドは、(1) 入力された値の長さが 5 文字であるかどうか、(2) 3 番目の文字 (オフセット 2) がコロンであるかどうかをチェックする。IF コマンドには論理 OR 演算子が含まれているため、いずれかの式が真の場合 (すなわち、値の長さが 5 文字でないか、3 番目の文字の位置にコロンがない場合) には、プロシージャは BADTIME ラベルに分岐する。

- 4 F\$TIME レキシカル関数が現在の時間値を TIME シンボルに収める。
- 5 F\$CVTIME 関数が現在の時間 (TIME シンボルにセーブされている) から "minute" と "hour" フィールドを取り出してから、変換したい時間から "minute" と "hour" フィールドを取り出す。
- 6 この割り当て文は、現在の時間と将来の時間を分に変換する。割り当て文の中でシンボル MINUTES, HOURS, FUTURE_HOURS, および FUTURE_MINUTES を使用すると、これらの値は自動的に整数に変換される。
- 7 ここで、将来の時間 (分) から現在の時間 (分) を減算する。
- 8 この結果が 0 未満の場合には、将来の時間は翌日になると解釈される。この場合、プロシージャは、これ以降の時間に 24 時間を加算してから現在の時間を減算する。
- 9 プロシージャはループに入り、その中で MINUTES_TO_WAIT の値から時間数を計算する。ループを通過するたびに、MINUTES_TO_WAIT が 60 より大きいかどうかをチェックする。60 より大きい場合には、MINUTES_TO_WAIT から 60 を減算してから、時間数のアキュムレータ (HOURS_TO_WAIT) に 1 を加算する。
- 10 ループを終了すると、プロシージャは時間値と分値を時間文字列に連結する。シンボル HOURS_TO_WAIT と MINUTES_TO_WAIT はそれと同値の文字列に置き換えられ、間にコロンが置かれる。これによって得られた文字列は、これ以降の時間のデルタ時間が収められる WAIT_TIME シンボルに割り当てられる。WAIT_TIME はグローバル・シンボルとして定義されているため、CONVERT.COM プロシージャが終了しても削除されない。
- 11 2 番目のパラメータである SHOW が入力された場合には、結果の時間値を表示し、入力されなかった場合には、プロシージャが終了する。
- 12 BADTIME ラベルでは、プロシージャは、間違って入力された値と正しい形式を示すエラー・メッセージを表示する。エラー・メッセージを出した後、CONVERT.COM が終了する。EXIT コマンドは、上位桁が 1 に設定されたエラー状態を戻すため、エラー・メッセージは表示されない。

プロシージャは、EXIT コマンドでエラー状態を明示的に指定するため、ユーザは別のプロシージャの中から CONVERT.COM を実行できる。CONVERT.COM が完了すると、呼び出し側プロシージャは時間が正しく変換されたかどうかを判別できる。

- 13 TELL ラベルでは、プロシージャの実行内容を表示する。TYPE コマンドは、入力データ・ストリームの SYSSINPUT にリストされた行を表示する。

CONVERT.COM コマンド・プロシージャの実行結果例

```
$ SHOW TIME
10-JUN-1999 10:38:26
$ @CONVERT 12:00 SHOW
WAIT_TIME = "1:22:00.00"
```

SHOW TIME コマンドは、現在の日時を表示します。CONVERT.COM はパラメータ 12:00 と SHOW を指定して実行されます。プロシージャは、絶対時刻 12:00 をデルタ値に変換して、それをターミナルに表示します。

B.2 REMINDER.COM コマンド・プロシージャ

所定の時間にターミナルに催促メッセージを表示します。このプロシージャは、メッセージを表示したい時間とメッセージのテキストを求めるプロンプトを出し、CONVERT.COM を使用して時間をデルタ時間に変換します。次に、指定された時間まで待ってから催促メッセージを表示するサブプロセスを生成します。F\$ENVIRONMENT、F\$VERIFY、および F\$GETDVI 関数の使用方法を示します。

例: 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
```

コマンド・プロシージャの例

B.2 REMINDER.COM コマンド・プロシージャ

```
$ !
$ @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_DECCRT") .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 6
$ END: 7
$ ! 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
```

REMINDER.COM コマンド・プロシージャの説明

- 1 F\$ENVIRONMENT 関数を使用してイメージ・チェック設定値を SAVE_VERIFY_IMAGE シンボルにセーブする。次に、F\$VERIFY 関数を使用して、プロシージャ・チェック設定値を SAVE_VERIFY_PROC シンボルにセーブする。F\$VERIFY 関数は両方のタイプのチェック設定値をオフにする。
- 2 INQUIRE コマンドを使用して、催促メッセージを送信する時間を求めるプロンプトを出す。この値は CONVERT.COM プロシージャへの入力として使用される。メッセージのテキストを求めるプロンプトも出す。
- 3 ネストされたプロシージャ CONVERT.COM を実行する。ファイル指定の一部としてディスクとディレクトリを指定ようにする。こうすると、システムは、REMINDER.COM を実行するディレクトリにかかわらず、CONVERT.COM を探すことができる。

CONVERT.COM は、催促をデルタ時間に変換して、この時間をグローバル・シンボル WAIT_TIME に戻す。このデルタ時間は、現在の時間からメッセージを送信する時間までの時間間隔を示している。CONVERT.COM がエラーを戻す場合は、プロシージャは BADTIME ラベルに分岐する。

- 4 CREATE コマンドを使用して新しいプロシージャ WAKEUP.COM を作成する。このプロシージャはサブプロセスの中から実行される。CREATE コマンドでドル記号で始まる行を読み込めるようにするには、DECK と EOD コマンドを CREATE コマンドの入力の前後に置く。このため、DECK と EOD コマンドの間の行はすべて WAKEUP.COM に書き込まれる。
- 5 WAKEUP.COM は、次のタスクを実行する。

- WAIT_TIME シンボルによって示される時間まで待つ。
- ターミナル・ベルを鳴らす BELL シンボルを作成する。
- ターミナルが DEC_CRT ターミナルかどうかと、ダブルサイズの点滅文字を表示するのにエスケープ・シーケンスを受け入れられるかどうかを判別する。DEC_CRT ターミナルがあるかどうかを調べるには、SHOW TERMINAL コマンドを入力して、この属性がリストされるかどうかを見る。
- ターミナルが DEC_CRT ターミナルの場合には、プロシージャはシンボル SET_FLASH, TOP, BOT を定義する。これらのシンボルがあると、ターミナルは点滅するダブルサイズ文字を使用するようになる。プロシージャは、ターミナルを元の状態に戻す SET_NOFLASH シンボルも定義している。これらの定義を EDT エディタを使用して作成するときに、エスケープ文字 (<ESC>) を入力するには、ESC キーを 2 回押す。

これらのシンボルを定義した後、ターミナルに 3 行を書き込む。1 行目は、ベルを鳴らし、点滅文字をオンにしてから、メッセージの上半分を (ダブルサイズ文字を使用して) 表示する。2 行目は、ベルをもう一度鳴らし、メッセージの下半分を表示する。3 行目は、現在の時間を書き込んで、点滅属性をオフにしてターミナルを通常の状態に戻す。

コマンド・プロシージャの例

B.2 REMINDER.COM コマンド・プロシージャ

DEC_CRT ターミナルがない場合には、ターミナル・ベルを鳴らし、ユーザからのメッセージと時間を表示する。

- DELETE コマンドは、WAKEUP.COM プロシージャが実行後に削除されるようにする。

- 6 WAKEUP.COM を作成した後、サブプロセスを生成して、入力コマンド・ファイルとして WAKEUP.COM を使用するようにそのサブプロセスに指示する。
/NOWAIT 修飾子があるので、サブプロセスが WAKEUP.COM からコマンドを実行しているときに、ユーザはターミナルで作業を続けることができる。指定された時間に、WAKEUP.COM はターミナルにユーザのメッセージを表示する。

省略時の設定では、SPAWN コマンドは、グローバル・シンボルとローカル・シンボルを 1 つのサブプロセスに渡す。したがって、ユーザが REMINDER にシンボル WAIT_TIME と MESSAGE_TEXT の値を与えるが、WAKEUP.COM もこの 2 つのシンボルにアクセスできる。

- 7 終了する前に元のチェック設定値を復元する。

REMINDER.COM コマンド・プロシージャの実行結果例

```
$ @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
```

時間値とメッセージを求めるプロンプトを出します。次に、メッセージを表示するサブプロセスを生成します。ユーザがターミナルで作業を継続していても、サブプロセスは、指定された時間になると、ターミナル・ベルを鳴らしたり、ユーザのメッセージを表示したり、時間を表示したりします。

B.3 DIR.COM コマンド・プロシージャ

DCL コマンド・プロシージャ DIRECTORY/SIZE=ALL/DATE を模倣して、指定されたファイルの (使用済みと割り当てられた) ブロック・サイズと作成日を表示します。F\$PARSE, F\$SEARCH, F\$FILE_ATTRIBUTES, および F\$FAO レキシカル関数の使用方法を示します。

例: 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
```

コマンド・プロシージャの例

B.3 DIR.COM コマンド・プロシージャ

```
$ !
$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<!UL!> !17AS",FILENAME,-
$             USED, ALLOC, REVISED)
$     WRITE SYS$OUTPUT LINE
$     GOTO LOOP
$
$ !
$ ! Output summary information, reset verification, and exit
$ !
$ DONE:           5
$     WRITE SYS$OUTPUT ""
$     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 コマンド・プロシージャの説明

- 1 F\$PARSE 関数を使用して、ユーザが提供するファイル指定である P1 のブランク・フィールドにアスタリスクを置く。DIR.COM を実行するときにユーザがパラメータを指定しないと、F\$PARSE 関数は "*.;*;" の値を P1 に割り当てる。この結果、DIR.COM は現在の省略時のディレクトリにあるすべてのファイルを表示する。
- 2 F\$SEARCH レキシカル関数がこのディレクトリで P1 によって示される 1 つまたは複数のファイルを検索する。P1 にワイルドカード文字 (アスタリスク) が指定されている場合、F\$SEARCH 関数は一致するすべてのファイル指定を戻す。最後のファイル指定が戻されると、DONE ラベルに分岐する。
- 3 最初のループ実行時に、ディレクトリ表示用のヘッダを書き込む。このヘッダには、デバイスとディレクトリの名前が含まれている。これらの名前を得るために、F\$PARSE 関数を使用している。
- 4 F\$PARSE レキシカル関数を使用して、ディレクトリの中のそれぞれのファイル指定からファイル名を取り出す。このとき、F\$FILE_ATTRIBUTES レキシカル関数は、使用済みのブロック数、割り当てられたブロック数、それぞれのファイルについての作成日情報を得る。最後に、F\$FAO 関数がディレクトリの中のそれぞれのファイルごとに 1 行ずつフォーマットする。F\$FAO 関数はファイル名とファイル属性情報を引数として使用する。

- 5 FSSEARCH が空の文字列を戻す場合は、DONE ラベルに分岐して、ファイルの合計数、使用されたブロックの合計数、ディレクトリで割り当てられたブロックの合計数を示す要約情報を表示する。

DIR.COM コマンド・プロシージャの実行結果例

```
$ @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.
```

このプロシージャは、[VERN]ディレクトリのすべての COM ファイルに関する情報を戻します。

B.4 SYS.COM コマンド・プロシージャ

現在のプロセス、グループ内のすべてのプロセス (現在のプロセスが group 特権を持つ場合)、およびシステム上のすべてのプロセス (現在のプロセスが world 特権を持つ場合) についての統計情報を戻します。FSPID、FSEXTRACT、および FSGETJPI レキシカル関数の使用方法を示します。

例: 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"
```

コマンド・プロシージャの例

B.4 SYS.COM コマンド・プロシージャ

```
$ !
$ ! 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"),IMAGENAME)
$     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 コマンド・プロシージャの説明

- 1 CONTEXT シンボルが空の値で初期化される。このシンボルを F\$PID 関数と一緒に使用すると、プロセス識別番号のリストを得ることができる。
- 2 ヘッダを書き込む。
- 3 最初のプロセス識別 (PID) 番号を得る。現在のプロセスに group または world 特権がない場合には、現在のプロセスの PID 番号が戻される。現在のプロセスが group 特権を持っている場合には、グループ・リストの中の最初の PID 番号が戻され、現在のプロセスが world 特権を持っている場合には、システム・リストの中の最初の PID 番号が戻される。最後の PID 番号が戻されるまで、順に PID 番

号を戻し続ける。この時点で、空の文字列が戻され、プロシージャの終わりに分岐する。

- 4 F\$GETJPI レキシカル関数を使用して、それぞれの PID 番号のイメージ・ファイル指定を得る。F\$PARSE 関数は、F\$GETJPI 関数が戻す指定からファイル名を取り出す。
- 5 F\$GETJPI 関数を使用して、それぞれの PID 番号のターミナル名を得る。F\$EXTRACT 関数は、F\$GETJPI(PID,"MODE") が戻す MODE 指定の最初の 3 文字を取り出して、プロセスのタイプを判別する。F\$GETJPI 関数をもう一度使用して、プロセスがサブプロセスかどうかを判別する。
- 6 F\$GETJPI レキシカル関数を使用して、ユーザ名、プロセス名、プロセス状態、プロセス優先順位、戻されたそれぞれの PID 番号のプロセスの基本優先順位を得る。F\$FAO レキシカル関数は、この情報を画面に合わせてフォーマットする。

SYS.COM コマンド・プロシージャの実行結果例

\$ @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

このプロシージャは、システム上のすべてのプロセスに関する情報を戻します。現在のプロセスは world 特権を持っています。

B.5 GETPARMS.COM コマンド・プロシージャ

このコマンド・プロシージャは、プロシージャに渡すパラメータの数を戻します。別のプロシージャから GETPARMS.COM を呼び出せば、呼び出し側プロシージャに渡されたパラメータの数を判別できます。

例: 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)
```

コマンド・プロシージャの例

B.5 GETPARMS.COM コマンド・プロシージャ

```
$ !
$ 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 コマンド・プロシージャの説明

- 1 現在のイメージとプロシージャのチェック設定値をセーブしてから、チェックをオフに設定する。
- 2 疑問符がパラメータとしてプロシージャに渡された場合には、TELL ラベル (注 6) に分岐する。
- 3 プロシージャに渡されたパラメータ数をカウントするループが設定される。カウンタ COUNT と LASTNONNULL は、ループに入る前に 0 に初期化される。ループの中で、COUNT は増分され、8 の値になっているかどうかテストされる。COUNT が 8 に等しい場合には、最大数のパラメータが入力されている。空以外のパラメータが渡されるたびに LASTNONNULL はそのパラメータの番号に定義される。

IF コマンドを実行するたびに、COUNT シンボルは異なる値を持つ。1 回目は、COUNT の値は 1 で、IF コマンドは P1 をチェックし、2 回目は、P2 をチェックし、以下も同様である。

- 4 パラメータ・カウントが 8 になると、END_COUNT に分岐する。
LASTNONNULL シンボルには、最後に渡された空以外のパラメータのカウントが入っている。この値はグローバル・シンボル PARMCOUNT に収められる。PARMCOUNT は、呼び出し側のコマンド・レベルで値をテストできるようにグローバル・シンボルとして定義しなくてはならない。
- 5 元のチェック設定値が復元される。
- 6 TELL ラベルでは、TYPE コマンドが入力ストリームに収められたデータを表示する。コマンド・プロシージャの中では、入力ストリームはコマンド・プロシージャ・ファイルである。TYPE コマンドは、GETPARMS.COM の使用方法に関する命令を表示する。

GETPARMS.COM コマンド・プロシージャの実行結果例

SORTFILES.COM プロシージャは、3 つの空以外のパラメータを渡すようにユーザに要求します。SORTFILES.COM プロシージャには、次の行が入っています。

```
$ @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
```

SORTFILES.COM プロシージャは、次のようにして起動できます。

```
$ @SORTFILES DEF 4
Three non-null parameters required. Type SORTFILE HELP for info.
```

このプロシージャを正しく起動するには、すなわち、SORTFILES に渡されたパラメータをそのまま GETPARMS に渡して処理するには、シンボル P1 ~ P8 が 1 組の二重引用符で囲まれていなくてはなりません。

GETPARMS からの戻り値が 3 でない場合には、SORTFILES はエラー・メッセージを出力して終了します。

B.6 EDITALL.COM コマンド・プロシージャ

EDT エディタを繰り返し起動して、同じファイル・タイプを持つファイルのグループを編集します。このプロシージャは、レキシカル関数を使用して出力からファイル名を取り出す方法を示しています。また、コマンド・プロシージャの中で起動されたプログラムの入力ストリームを再定義する方法も示しています。

コマンド・プロシージャの例

B.6 EDITALL.COM コマンド・プロシージャ

例: 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 コマンド・プロシージャの説明

- 1 ON コマンドは、このプロシージャの条件処理を設定する。このプロシージャの実行時に Ctrl/Y が押されると、DONE ラベルに分岐する。同様に、エラーまたは重大エラーが生じてても、DONE ラベルに分岐する。
- 2 パラメータが入力されたかどうかをチェックする。入力されていない場合には、ファイル・タイプを求めるプロンプトを出す。

- 3 DIRECTORY コマンドは、P1 として指定されたファイル・タイプを持つすべてのファイルをリストする。コマンド出力は DIRECT.COM ファイルに書き込まれる。/VERSIONS=1 修飾子は、それぞれのファイルの最も大きいバージョン番号だけをリストするように要求する。NOHEADING 修飾子と/NOTRAILING 修飾子は、出力に見出し行やディレクトリ要約を含めないように要求する。/COLUMNS=1 修飾子は、レコードごとにファイル名が 1 つずつ指定されるようにする。
- 4 IF コマンドは、\$STATUS の値をテストすることによって、DIRECTORY コマンドからの戻り値をチェックする。DIRECTORY コマンドが正しく実行されないとき、\$STATUS は偶数の整数値になり、プロシージャは ERROR_SEC ラベルに分岐する。
- 5 OPEN コマンドは、ディレクトリ出力ファイルをオープンして、それに論理名 DIRFILE を指定する。
- 6 READ コマンドは、DIRECTORY コマンド出力の行をシンボル名 NAME に読み込む。すべての行を読み込むと、DEFINE コマンドを使用して、編集セッションの入力ストリーム (SYSS\$INPUT) がターミナルになるように再定義する。次に、エディタを起動して、NAME シンボルをファイル指定として指定する。編集セッションが完了すると、コマンド・インタプリタは、コマンド・プロシージャの中の次の行を読み込んで、NEWLINE ラベルに分岐する。ディレクトリの中にある指定されたファイル・タイプのすべてのファイルを編集すると、DONE ラベルに分岐する。
- 7 DONE ラベルは、READ コマンドの/END 修飾子のターゲット・ラベルであるとともに、プロシージャの始めに設定された ON CONTROL_Y 条件と ON ERROR 条件のターゲット・ラベルでもある。このラベルでは、クリーンアップ処理を行う。

CLOSE コマンドは、DIRECTORY コマンドの出力ファイルをクローズし、/ERROR 修飾子はファイルの中の次の行のラベルを指定する。このように /ERROR 修飾子を使用すると、ディレクトリ・ファイルがオープンされていないときに表示されるはずのエラー・メッセージが出されなくなる。このような状況は、ディレクトリ・ファイルをオープンする前に Ctrl/Y を押した場合などに生じる。

クリーンアップ処理の 2 番目のステップは、一時ディレクトリ・ファイルを削除することである。

EDITALL.COM コマンド・プロシージャの実行結果例

```
$ @EDITALL DAT
*
.
.
.
%DELETE-I-FILDEL, device:[directory]DIRECT.OUT:1 deleted (x blocks)
```

EDITALL プロシージャが DAT として指定された P1 によって起動されます。このプロシージャは、省略時のディレクトリにあるファイル・タイプ DAT を持つすべてのファイルのディレクトリ・リストを作成し、エディタを起動してそれぞれのファイルを編集します。ファイル・タイプ DAT を持つ最後のファイルを編集し終わると、一時ファイル DIRECT.OUT を削除して、ターミナルにメッセージを表示します。

B.7 MAILEDIT.COM コマンド・プロシージャ

このコマンド・プロシージャは、MAIL ユーティリティでテキスト・エディタを起動しています。

例: 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 コマンド・プロシージャの説明

- 1 DEFINE コマンドは、エディタ入力をコマンド・ファイルからではなく、ターミナルから得られるようにする。
- 2 IF 文は、異なる出力ファイル名を持つファイルを編集することと、同じファイル名を持つファイルを編集することを区別する。
- 3 この EDIT コマンドは、入力ファイル名と出力ファイル名でエディタを起動する。この行を、たとえば次のように編集すると、ユーザが選択したエディタを起動できる。


```
$ RUN XYZ_EDITOR.EXE /INPUT= 'P1' /OUTPUT='P2'
```

- 4 この EDIT コマンドは、1つのファイル名でエディタを起動する。この行を、たとえば次のように編集すると、ユーザが選択したエディタを起動できる。

```
$ RUN XYZ_EDITOR.EXE /INPUT= 'P2' /OUTPUT='P2'
```

MAILEDIT.COM コマンド・プロシージャの実行結果例

```
$DEFINE MAIL$EDIT MAILEDIT.COM
```

```
$MAIL
```

```
MAIL> SHOW EDITOR
```

```
Your editor is defined by the file MAILEDIT.COM.
```

B.8 FORTUSER.COM コマンド・プロシージャ

FORTRAN プログラムの作成、コンパイル、実行を行う会話型ユーザの、ターミナル環境を制御するシステム定義のログイン・コマンド・プロシージャのサンプルを示します。FORTUSER.COM が、ログイン・コマンド・プロシージャとしてリストされる専用アカウントにログインした場合は、FORTUSER.COM が受け入れるコマンドしか実行できません。このプロシージャは、レキシカル関数を使用してオプション・テーブルを参照し、ユーザが入力したコマンドと有効なコマンドのリストとを比較する方法も示しています。

例: 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.
```

コマンド・プロシージャの例

B.8 FORTUSER.COM コマンド・プロシージャ

```

$ !
$ 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:
$     WRITE SYS$OUTPUT " Invalid command"      7
$ !
$ HELP:
$     TYPE SYS$INPUT                          8
$     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:
$     DEFINE/USER_MODE SYS$INPUT SYS$COMMAND: 10
$     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:
$      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 コマンド・プロシージャの説明

- 1 SET NOCONTROL=Y コマンドは、このプロシージャの制御下でログインするユーザが、プロシージャまたはプロシージャの中のコマンドやプログラムに割り込めないようにする。
- 2 このオプション・テーブルには、ユーザが実行できるコマンドがリストされる。それぞれのコマンドはスラッシュによって区切られている。
- 3 プロシージャ自体を示している。
- 4 シンボル名 PROMPT にプロシージャの中のラベルの値が割り当てられる。プロシージャを初めて起動するとき、このシンボルの値は INIT になる。HELP コマンド・テキストは、PROMT ラベルを指定する GOTO コマンドで終了する。このテキストを初めて表示する場合には、GOTO コマンドの結果、HELP ラベルに分岐する。このとき、ユーザが入力可能なコマンドを示す HELP メッセージが表示される。次に、プロシージャは INIT ラベルに戻って、そこで PROMT の値を "GET_COMMAND" に変更し、最後に、FILE ラベルに分岐して、ファイル名を得る。これ以降は、ヘルプ・テキストが表示されると、GET_COMMAND ラベルに分岐して、次のコマンドを得る。
- 5 Ctrl/Y 条件アクションが、警告条件アクションと同じように、GET_COMMAND ラベルに戻るよう設定される。プロシージャは、コマンドを求めるプロンプトを出し、入力があるまでプロンプトを出し続ける。セッションを終了して会話型コマンド・レベルに戻るには、DONE コマンドを入力する。
- 6 F\$LOCATE レキシカル関数と F\$LENGTH レキシカル関数を使用して、オプションのリストにコマンドが指定されているかどうかを判別する。F\$LOCATE 関数は、ユーザが入力したコマンドで末尾にスラッシュが付くものを検索する。たとえば、EDIT と入力すると、EDIT/を検索する。オプション・リストにコマンドが指定されていない場合には、プロシージャは INVALID_COMMAN ラベルに分岐する。オプション・リストにコマンドが指定されている場合は、該当するラベルに分岐する。

- 7 INVALID_COMMAND ラベルで、プロシージャはエラー・メッセージを出し、有効なコマンドの一覧を示すヘルプ・テキストを表示する。
- 8 ヘルプ・テキストには、有効なコマンドがリストされる。このテキストは一番最初に表示される。また、ユーザが HELP コマンドや無効なコマンドを入力したときも表示される。
- 9 HELP テキストの末尾の GOTO コマンドはシンボル名 PROMPT を指定している。このプロシージャを初めて起動するとき、このシンボルは INIT の値になり、それ以降は GET_COMMAND の値になる。
- 10 リストの中の有効な各コマンドは、オプション・テーブルに対応するエントリを持ち、コマンド・プロシージャの中に対応するラベルを持つ。ターミナルから入力を読み込む EDIT などのコマンドの場合は、入力ストリームを SYSSCOMMAND として定義する DEFINE コマンドがプロシージャに含まれる。
- 11 BADFILE ラベルで、プロシージャはファイルが現在のディレクトリにないことを示すメッセージを表示する。次に、別のファイル名を求めるプロンプトを出す。
- 12 ファイル名を得ると、現在の省略時のディレクトリ以外のディレクトリをユーザが指定しなかったかどうかをチェックする。ここで FSPARSE 関数を使用して、ファイル名を取り出す (それぞれのコマンドには、該当する省略時のファイル・タイプがある)。次に、GET_COMMAND ラベルに戻って、ファイル进行处理するコマンドを得る。

FORTUSER.COM コマンド・プロシージャの実行結果例

この例は、このコマンド・プロシージャを専用コマンド・プロシージャとして使用する方法を示しています。

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
```

このサンプル実行は、CLASS30 というユーザが FORTUSER コマンド・プロシージャによって制御されるアカウントにログインするときのセッションを示しています。FORTUSER コマンド・プロシージャは、ユーザが実行できるコマンドと、セッションを再開するための命令を表示します。次に、ユーザが AVERAGE ファイルを指定して、それをコンパイル、リンク、実行します。この後、ユーザは FILE コマンドを入力して、別のファイルで作業を開始します。

B.9 LISTER.COM コマンド・プロシージャ

入力データを求めるプロンプトを出し、データを桁揃えしてからソートし、出力ファイルに書き込みます。このプロシージャは、READ と WRITE コマンドとともに、割り当て文の部分文字列オーバーレイ形式を示しています。

例: 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
```

コマンド・プロシージャの例

B.9 LISTER.COM コマンド・プロシージャ

```
$ !
$ 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 コマンド・プロシージャの説明

- 1 LISTER.COM は、現在のチェック設定値をセーブして、チェックをオフに設定する。
- 2 OPEN コマンドが書き込み用の一時ファイルをオープンする。
- 3 INQUIRE コマンドは、プログラマ名とファイル名を求めるプロンプトを出す。INQUIRE コマンドのプロンプトに対して Return と表示される空の行を入力すると、プロシージャは、これ以上入力されるデータがないとみなして、FINISHED ラベルに分岐する。
- 4 シンボル NAME と FILE に値を割り当てた後、割り当て文の文字列オーバーレイ形式を使用して、RECORD シンボルの値を構築する。RECORD の 1 ~ 21 桁には、NAME の現在の値が書き込まれる。コマンド・インタプリタは、NAME の値にスペースを埋め込んで、指定された 20 文字の長さにする。
同様に、RECORD の次の 20 桁に FILE の値が埋め込まれると、RECORD の値が出力ファイルに書き込まれる。
- 5 ファイルがクローズされると、プロシージャは出力ファイル DATA.TMP をソートする。DEFINE コマンドは、SORT コマンド出力を空のファイルである NL に向ける。これ以外の場合は、これらの統計情報はターミナルに表示される。
ソート操作は、最初の 20 桁、すなわちプログラマ名によって実行される。
ソート済み出力ファイルには、DOC.SRT の名前が付けられる。
- 6 OPEN コマンドで最終的な出力ファイルである DOCUMENT.DAT を作成する。ファイルに書き込まれる最初の数行はヘッダ行で、タイトル、日時、カラムの見出しが示される。

- 7 DOCUMENT.DAT ファイルをクローズして、ソート済み出力ファイル DOC.SRT をこのファイルに追加する。次に、出力ファイルはシステム・プリンタのキューに登録される。
- 8 最後に、中間ファイルを削除するかどうかを決定するようにプロンプトを出す。INQUIRE コマンド・プロンプトに対して真の応答 (T, t, Y, または y) を入力すると、ファイル DATA.TMP と DOC.SRT は削除される。偽の応答を入力すると、この 2 つのファイルは保管される。

LISTER.COM コマンド・プロシージャ実行結果例

```
$ @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
```

このプロシージャを実行して得られる出力ファイルは次のようになります。

```
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
```

B.10 CALC.COM コマンド・プロシージャ

算術計算を行い、その結果を 16 進値と 10 進値に変換します。

例: 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.
```

コマンド・プロシージャの例

B.10 CALC.COM コマンド・プロシージャ

```
$ !
$ 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 コマンド・プロシージャの説明

- 1 プロシージャを再開するエラー処理条件を設定する。警告またはそれ以上の重大度を持つエラーが生じると、プロシージャは始めに戻って、ON 条件を再設定する。

この方法を使用すると、ユーザが式を間違えて入力してもプロシージャは終了しない。

- 2 INQUIRE コマンドは、算術式を求めるプロンプトを出す。プロシージャは、次のいずれかの形式の式を受け入れる。

名前=式

式

ユーザが Return を押すと、プロシージャは、CALC セッションが終わったとみなして、終了する。

ユーザが "名前=式" 形式で入力した場合は、プロシージャは STATEMENT ラベルで実行を継続し、そうでない場合は、EXPRESSION ラベルに分岐する。

- 3 割り当て文を実行して、式の結果をシンボルに割り当てる。次に、シンボル名を取り出して、シンボルの値を Q に割り当てる。こうしておく、ユーザは、次にプロシージャを反復するときに Q を使用することもできるし、シンボルを使用することもできる。次に、プロシージャはこの結果を表示し、START ラベルに戻る。
- 4 式を評価して、評価結果を Q シンボルに割り当てる。ユーザは、次にプロシージャを反復するときに Q を使用できる。次に、この結果を表示して START ラベルに戻る。

CALC.COM コマンド・プロシージャの実行結果例

```
$ @CALC
Calc: 2 * 30
Decimal = 60      Hex = 0000003C  Octal = 00000000074
Calc: Q + 3
Decimal = 63      Hex = 0000003F  Octal = 00000000077
Calc: TOTAL = Q + 4
Decimal = 67      Hex = 00000043  Octal = 00000000103
Calc: 5 + 7
Decimal = 12      Hex = 0000000C  Octal = 00000000014
Calc: Return
$
```

プロシージャからのプロンプトの 1 つ 1 つに、ユーザは算術式を入力します。プロシージャはその結果を 10 進数、16 進数、8 進数で表示します。データの無い行で Return を押しただけの空の行で CALC セッションは終わりです。

B.11 BATCH.COM コマンド・プロシージャ

コマンド文字列、コマンド・プロシージャ、コマンドのリストのうちのいずれかを受け取って、バッチ・ジョブとして実行します。

例: 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
$!
```

コマンド・プロシージャの例

B.11 BATCH.COM コマンド・プロシージャ

```
$!
$! 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:
$ 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:
$ 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.
$!

```

コマンド・プロシージャの例

B.11 BATCH.COM コマンド・プロシージャ

```
$ 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 コマンド・プロシージャの説明

- 1 この IF 文は、BATCH.COM がバッチ・モードで実行されているかどうかをテストする。BATCH.COM を会話形式で起動した場合、バッチ・ジョブとして実行されるコマンド文字列またはコマンド・プロシージャをパラメータとして指定する。パラメータを指定しないと、BATCH.COM はコマンドを求めるプロンプトを出し、これらのコマンドをファイルに書き込んでから、このコマンド・プロシージャをバッチ・ジョブとして実行する。

バッチ・ジョブから 1 つまたは複数のコマンドを実行する準備が整うと、BATCH.COM は SUBMIT コマンドを使用して BATCH.COM 自体をバッチ・ジョブとしてキューに登録し、このジョブからコマンドを実行する (注 8 を参照)。BATCH.COM をバッチ・ジョブとして起動した場合には、プロシージャは EXECUTE_BATCH_JOB ラベルに分岐する。BATCH.COM をバッチ・モードで

実行するときには、実行するコマンドまたはコマンド・プロシージャを P1 として、省略時のディレクトリを P2 として指定しなければならない。

- 2 これらのコマンドは、バッチ・ジョブの実行準備を行う。最初に、バッチ・ジョブの名前を構築する。コマンド文字列が渡された場合、BATCH.COM は動詞名をジョブ名として使用し、コマンド・プロシージャが渡された場合は、ファイル名を使用する。入力が入力されなかった場合には、ジョブに BATCH という名前を付ける。
- 3 パラメータを連結して、実行するコマンド文字列にする。コマンド文字列は COMMAND シンボルに割り当てられる。
- 4 SUBMIT コマンドは、255 文字より長いパラメータは渡すことができない。したがって、プロシージャは、コマンド文字列とディレクトリ名の長さが 255 文字未満になっているかどうかをテストする。両方の文字列が 255 文字未満であり、ユーザが実際にはコマンド文字列を渡した場合には、SUBMIT ラベルに分岐する。
- 5 Ctrl/Y ハンドラが設定されるので、コマンド・プロシージャのこのセクションでユーザが Ctrl/Y を押すと、クリーンアップ処理が実行される。
- 6 実行するコマンドを収める一時ファイルを作成する。ユーザが長いコマンド文字列を指定した場合は、プロシージャは WRITE_LARGE_COMMAND に分岐して、このコマンドを一時ファイルに書き込む。そうでない場合には、実行するコマンドを求めるプロンプトを出す。それぞれのコマンドは一時ファイルに書き込まれる。
- 7 一時ファイルをクローズする前に、シンボル割り当て文をファイルに書き込むようにする。この文は、一時ファイルのファイル名を BATCH\$DELETE_FILESPEC シンボルに割り当てる。一時ファイルをクローズした後、プロシージャは省略時の Ctrl/Y ハンドラを再設定する。次に、COMMAND シンボルを定義して、実行時に COMMAND シンボルが一時コマンド・ファイルを起動するようにする。
- 8 定義されたジョブ名を使用して、プロシージャ自体をバッチ・ジョブとしてキューに登録する (注 2 を参照)。また、実行するコマンドまたはコマンド・プロシージャと、コマンドを実行するディレクトリの 2 つのパラメータも渡す。次に、EXIT ラベルに分岐する (注 11 を参照)。
- 9 一時ファイルの作成中にユーザが Ctrl/Y を押すと、ここのセクションでクリーンアップ処理を実行する。
- 10 このセクションは、一時ファイルを作成できないときにエラー・メッセージを作成する。
- 11 元のチェック設定値を再設定してから終了する。
- 12 これらのコマンドは、BATCH.COM がバッチ・モードで実行されるときに実行される。最初に、ON エラー処理機能が使用可能になり、ユーザの省略時のチェック設定値が設定される。次に、省略時の値が P2 によって示されるディレクトリに設定され、P1 によって示されるコマンドまたはコマンド・プロシージャが実行される。一時ファイルが作成された場合には、このファイルは削除される。

コマンド・プロシージャの例

B.11 BATCH.COM コマンド・プロシージャ

BATCH\$DELETE_FILESPEC を削除する前に、P1 の完了状態がセーブされる。この完了状態は EXIT コマンドによって戻される。

BATCH.COM コマンド・プロシージャの実行結果例

```
$ @BATCH RUN MYPROG
Job RUN (queue SYS$BATCH, entry 1715) started on SYS$BATCH
```

この例は、BATCH.COM を使用して、バッチ・ジョブの中からプログラムを実行しています。

B.12 COMPILE_FILE.COM コマンド・プロシージャ

PASCAL または FORTRAN で書かれたファイルのコンパイル、リンク、実行を行います。処理するファイルを求めるプロンプトを出し、ファイル・タイプが PAS か FOR かを判別します。ファイル・タイプが PAS でも FOR でもない場合、またはファイルが現在の省略時のディレクトリに存在しない場合には、該当するエラー・メッセージを出力します。このプロシージャは、IF-THEN-ELSE 言語構成の使用方を示しています。

例: 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 コマンド・プロシージャの説明

- 1 IF コマンドは、FSSEARCH レキシカル関数を使用して、ディレクトリを検索し、ファイルが存在するかどうかを判別する。
- 2 THEN コマンドの後のコマンド・ブロックは、次のことを行う。
 - F\$LENGTH レキシカル関数を使用して、ファイル・タイプの長さを判別する。
 - ファイル・タイプを判別する。
 - ファイル・タイプからピリオドを削除する。
 - ファイル指定からファイル・タイプを削除して、ファイル名を判別する。
 - ファイル名からピリオドを削除する。
 - エラーが生じたときのアクションを定義する。
 - FILE_TYPE シンボルによって定義されるラベルに分岐する。
- 3 "File to process:"プロンプトに対してユーザが入力したファイルがディレクトリの中にない場合には、ELSE コマンドの後のコマンドを実行する。
- 4 ENDIF コマンドは、IF-THEN-ELSE コマンド言語構成を終了する。

コマンド・プロシージャの例

B.12 COMPILE_FILE.COM コマンド・プロシージャ

- 5 FORTRAN プログラムをコンパイルして，LINK ラベルに分岐する。コンパイル中にエラーが生じると，PRINT ラベルに分岐する。
- 6 プログラムが正しくコンパイルされたことを表示して，プログラムをリンクしてから実行し，CLEANUP ラベルに分岐する。エラーが生じると，プログラムはEND ラベルに分岐する。
- 7 プログラムのリスト・ファイルを省略時の印刷キューに登録する。

例: COMPILE_FILE.COM コマンド・プロシージャの実行結果例

```
$ @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 
```


1 次パスワード (primary password)

ユーザ・パスワードのうち、ユーザが入力する最初のユーザ・パスワード。システムによっては、2 次パスワードも必要な場合がある。1 次パスワードは、そのパスワードとともに指定するユーザ名に対応づけたパスワードでなければならない。

2 次パスワード (secondary password)

1 次パスワードを正しく入力した直後に入力するログイン時のユーザ・パスワード。ただし、2 次パスワードは必要ない場合もある。1 次パスワードと2 次パスワードを使用することによって、ログイン時に複数のユーザがいることを各ユーザに認識させることができる。あまり一般的な方法ではないが、パスワード長を長くする目的で2 次パスワードを使用することもできる。この場合、2 つのパスワードの合計文字数が多くなるので、パスワードの推測には多くの時間がかかり、難しい。

ASCII (American Standard Code for Information Interchange)

情報交換用米国標準コードを参照。

CPU (central processing unit)

中央処理装置を参照。

DCL (DIGITAL Command Language)

DIGITAL コマンド言語を参照。

DECnet-Plus

Digital Network Architecture (DNA) フェーズ V を実現したコンパックのハードウェア/ソフトウェア製品のファミリであり、OSI および DNA プロトコルを統合する。OSI に準拠しており、DECnet フェーズ IV および TCP/IP と互換性がある。

DIGITAL コマンド言語 (DIGITAL Command Language (DCL))

OpenVMS システムでのコマンド・インタプリタ。ユーザとオペレーティング・システム間の通信を可能とする。

Extended File Specifications

OpenVMS にこれまで内在していたディレクトリやファイル命名に関する多くの制限を取り除くオプションの機能。深い階層のディレクトリや拡張したファイル名を使用できる。

MIME (Multipurpose Internet Mail Extension)

メール・メッセージに非テキスト・ファイルを添付する標準的な方法。グラフィック、サウンド・ファイル、等をエンコードし、プレーン・テキストとして送信する。このテキストは読むことができない。受信側は、MIME インタープリタ・ユーティリティを使用して元のファイル形式に戻すことができる。

RMS (Record Management Services)

ファイルおよびフィールド内のレコードを処理するために呼び出すオペレーティング・システムのプロシージャの集合。VMS RMS によりプログラムは、ブロックの読み込みおよび書き込み (ブロック入出力) 同様、レコード・レベルの GET および PUT 要求 (レコード入出力) を実行できる。VMS RMS はシステム・ソフトウェアの完全な部分であり、そのプロシージャはエグゼクティブ・モードで実行される。

UAF (user authorization file)

システム上の各アカウントの詳細を保持したファイル。各アカウントに割り当てられたユーザ名、パスワード、ユーザ識別コード (UIC)、クォータ、制限、特権を含む。

UFD (user file directory)

ディスクまたはテープに格納されたファイル群を一時的に登録するファイル。UFD には、格納された各ファイルの名前、タイプ、バージョン番号が保持される。また、ファイルの実際の記憶位置を識別し、そのファイル属性リストを指す一意な番号も入る。ディレクトリも参照のこと。

UIC (user identification code)

ユーザ、ファイル、グローバル・セクション、コマンド・イベント、フラグ・クラスタ、メールボックスに割り当てた番号の組み合わせ。UIC は、owner、group、world、system の各ユーザ・カテゴリが使用できるアクセス・タイプ (read、write、read/write、またファイルの場合は execute、delete、その両方) を指定する。

アカウント (account)

ユーザがシステムを使用するには、アカウントを保持していなければならない。アカウントは、ユーザのユーザ名によって識別される。アカウントが異なれば、異なるサービス・レベル (たとえば、ユーザが保持する特権、ログインできる期間など) を許可できる。

アクセス制御エントリ (access control entry (ACE))

アクセス制御リストの中のエントリ。アクセス制御エントリは、識別子、識別子の保持者に対するアクセス権の許可または禁止、ディレクトリに対する省略時の保護、機密保護アラームの詳細を指定できる。

アクセス制御リスト (access control list (ACL))

ユーザまたはグループが保護されたシステム・オブジェクトに対して持つアクセス権を定義したエントリの集合。

アクセス制御文字列 (access control string)

リモート・ノードに送信するログイン情報を表す 0 文字から 42 文字までの文字列。OpenVMS システムの場合、アクセス制御文字列は通常、ユーザ名、スペースまたはタブ、そしてパスワードから構成される。

安全ターミナル・サーバ (secure terminal server)

すでにログアウトされているターミナルだけにユーザがログインするように設計されている OpenVMS ソフトウェア。ユーザがターミナルで Break キーを押すと、安全ターミナル・サーバは (使用可能にされている場合)、ログインされているプロセスをすべて切断してから、ログインを開始することによって応答する。ターミナルでプロセスがログインされていない場合は、ただちにログインできる。

イメージ (image)

リンクによって結合された実行プログラムを構成するプロシージャとデータ。この実行可能なプログラムはプロセスで実行される。実行可能イメージ、共有イメージ、システム・イメージの 3 つのタイプがある。

印刷キュー (print queue)

印刷されるのを待機しているファイルのリスト。

エディタ (editor)

テキスト・ファイルの作成および修正のために使用するプログラム。

エラー・メッセージ (error message)

ユーザが要求した処理が失敗した場合にシステムが送信するメッセージ。各エラー・メッセージはエラーを検出したオペレーティング・システムの部分を特定する。ほとんどのエラー・メッセージは入力ミスまたは構文エラーが原因である。ほとんどの場合、コマンドの再入力によりエラーの訂正ができる。

演算子 (operator)

式の中でコンピュータにオペランドの処理方法を指定する部分。たとえば、プラス記号 (+) は加算を実行することを指定する演算子である。

オブジェクト

システムがアクセスを制御する情報の受動的な格納場所。オブジェクトへのアクセスは、そのオブジェクトに格納されている情報へのアクセスを意味する。

オープン・アカウント (open account)

パスワードが不要であるアカウント。

オペランド (operand)

式の中で値を持つ部分。オペランドは、式の評価の中で演算子によって処理され、結果が生成される。

オペレーティング・システム (operating system)

コンピュータ・プログラムの実行およびシステム関数の実行を制御する統合されたプログラムの集合。

カーソル (cursor)

モニタ画面上で使用され、画面上の位置を示す表示子。

開始位置修飾子 (start position qualifier)

EVE において、指定したバッファを開いたときに、最初にカーソルが表示される行および桁の位置を決定するための修飾子。

解析 (parsing)

次のいずれかです。

1. コマンド文字列を要素に分割し解釈すること。
2. OpenVMS RMS (レコード管理サービス) 同様に、ファイル指定を解釈すること。

階層ディレクトリ構造 (hierarchical directory structure)

1 つのディレクトリの下に複数のディレクトリが並び、それが何段にも並んだツリー構造のディレクトリ構造。

会話型モード (interactive mode)

ユーザがコマンドを入力するとシステムがそれを実行し反応する、オペレーティング・システムとの通信モード。1 つのコマンドを完了しないと次のコマンドを入力できない。

書き込み (write)

イメージがデータを送信することまたはその機能。たとえば、PRINT コマンドを実行すると、指定したファイルが格納されている記憶域から読み込まれ、プリンタに書き込まれる。読み込みも参照のこと。

空値 (null value)

コマンド・プロシージャの中で二重引用符 ("") で表示される文字を持たない文字列。

完全な名前 (full name)

DECdns ネームスペース内の名前の完全な指定であり、ルート・ディレクトリから、名前を指定しているオブジェクト、ディレクトリ、またはソフト・リンクまで、パス内のすべての親ディレクトリを含んでいる。ネームスペース名も含むことができるが、ネットワーク内にネームスペースが1つだけしか存在しない場合は不要である。

キー (key)

次のいずれかです。

1. 索引編成ファイルで、1 つの索引編成ファイルの各データ・レコードの文字列、バック 10 進数、2 バイトまたは 4 バイトの符号なしバイナリ数、2 バイトまたは 4 バイトの符号付き整数。ユーザがレコード内での長さや位置を定義し、OpenVMS RMS (レコード管理サービス) はキーを使用して索引を構成する。
2. 相対ファイルで、データ・ファイル内の各データ・レコードの相対レコード番号。OpenVMS RMS (レコード管理サービス) は相対レコード番号を使用して、ランダム・アクセス・モードで相対ファイル内のデータ・レコードの識別とアクセスを行う。
3. Sort/Merge ユーティリティで、レコードのソートに必要な情報を含むレコード内のデータ・フィールド。

キーパッド (keypad)

ターミナルのメイン・キーボードの隣にある小さなキー・セット。

キーボード (keyboard)

タイプライタに類似した操作を行うことができる入力デバイス。

キーワード (keyword)

通常コマンド文字列またはステートメントなど特定の構文形式で使用する予約語。

キュー (queue)

次のいずれかです。

1. 処理されるジョブの列。たとえば、バッチ・ジョブ・キューあるいはプリンタ・ジョブ・キュー。処理は基本的に FIFO (ファースト・イン/ファースト・アウト) であるが、ジョブの実行を要求したプロセスの優先順位を反映する。プリント・キューも参照。
2. リストまたは表の中にエントリを作成すること。INSQUE 命令を使用して行うことが多い。

区切り記号 (delimiter)

文字列、ステートメント、プログラムの要素の区切り、終了、まとめのために使用する文字。

クローズ (close)

ファイルに対するすべての操作を終了すること。

グローバル・シンボル (global symbol)

次のいずれかです。

1. プログラム内のモジュールで定義され、他のモジュールによる参照が可能なシンボル。リンカがグローバル・シンボルの定義と参照の一致を確認する。ローカル・シンボルも参照。
2. すべてのコマンド・レベルでアクセス可能なコマンド言語シンボル。

係留アカウント (captive account)

ユーザの処理を制限する OpenVMS アカウント。通常、ユーザが使用できるコマンド・プロシージャとコマンドは特定のものだけに制限される。たとえば、このアカウントのユーザは、Ctrl/Y キー・シーケンスを使用できない。このアカウントはターンキーや結合アカウントと同義である。

結合 (concatenate)

ファイルを一続きにリンクすること。

高性能 Sort/Merge ユーティリティ (high-performance Sort/Merge utility)

OpenVMS Alpha システムで使用可能な Sort/Merge ユーティリティのバージョン。

構文 (syntax)

スペル・チェック，修飾子およびパラメータの順序を含むコマンドの特別な書式。スペルミスの語が最も一般的な構文エラーである。

コマンド (command)

DCL (DIGITAL コマンド言語) で，通常英語表記による命令のこと。ユーザがターミナルから入力する。または，コマンド・プロシージャに書き込まれている。コマンドはソフトウェアにターミナルの監視あるいはコマンド・プロシージャの読み込みを要求し，いくつかの明確なアクティビティを実行する。たとえば COPY コマンドを入力した場合，1つのファイルの内容を別のファイルにコピーするようにシステムに要求する。

コマンド・イメージ (command image)

DCL コマンドと結合し DCL コマンドで起動されるプログラム。

コマンド・インタプリタ (command interpreter)

プロセスのコンテキストでスーパーバイザ・モードで実行されるプロシージャ・ベースのシステム・コード。ターミナルからのユーザ入力またはコマンド・ファイルで発行されたコマンドを受け取り，構文をチェックして解析する。

コマンド・パラメータ (command parameter)

ファイル指定，オプション，定数などの，スペースで区切られたオペランド。コマンドの中で，オペランドを指定する位置が異なれば，そのオペランドの意味も変わる。

コマンド・プロシージャ (command procedure)

ターミナルでユーザが個々のコマンドを入力するのではなく，コマンド・インタプリタが受け付けるコマンドとデータを含むファイル。したがって，コマンド・プロシージャは自動的にコマンドをオペレーティング・システムに渡す方法を提供する。さらに，ユーザはループ，カウンタ，ラベル，シンボル代入などのプログラミング手法を使用して，ユーザの会話型処理に代わる複雑なコマンド・シーケンスを設定できる。また，コマンド・プロシージャはバッチ・ジョブとしてシステムに処理される。

コマンド列 (command string)

コマンドまたはオプションでコマンドを修正する情報を含む行 (または連続した行の集合)。コマンド文字列は，コマンド，コマンド修飾子，コマンド・パラメータ (たとえば，ファイル指定)，コマンド・パラメータ修飾子から構成される。コマンド列は通常リターン・キーを押すことで終了する。

コマンド・レベル (command level)

コマンド・インタプリタ用の入力ストリーム。最初の入力ストリームは，必ずコマンド・レベル 0 である。会話型コマンド・プロシージャはコマンド・レベル 1 で実行を開始する。バッチ・ジョブ・コマンド・プロシージャは，コマンド・レベル 0 で実行を開始する。コマンド・プロシージャ内でプロシージャ実行コマンド (@) または CALL コマンドを使用すると，最高 32 個のネストしたコマンド・レベルを作成できる。

索引順編成ファイル (indexed sequential file)

各レコードに1つまたは複数のデータ・キーが埋め込まれているレコード・ファイル。ファイルの中のレコードは、そのレコードに関連づけられているキーを指定することで個々にアクセスできる。

サブディレクトリ (subdirectory)

上位のレベルのディレクトリにカタログされたディレクトリ。このディレクトリの所有者に所属する付加的なファイルをリストする。

サブプロセス (subprocess)

別のプロセスによって作成された補助プロセス。サブプロセスを作成したプロセスがその所有者である。プロセスおよびそのサブプロセスは、クォータ・プールおよび制限を共有する。所有者プロセスがシステムから削除された場合、すべてのそのサブプロセス (さらにそのサブプロセス) も削除される。

サブルーチン (subroutine)

別のプログラムで呼び出された場合、実行される補助ルーチン。ある条件が満たされるまで、サブルーチンはしばしば繰り返して呼び出される。

式 (expression)

変数、定数、その両方を演算子で組み合わせたもので、コンピュータが式を評価して結果を生成する。

識別子

ライト・データベースに登録され、アクセス要求の確認でシステムが使用する英数字文字列であり、ユーザまたはユーザ・グループを表現する。識別子には、環境、機能、汎用、ユーザ識別コード (UIC) の4種類がある。

磁気テープ (magnetic tape)

データの格納とアクセスが可能なメディア。

時刻印字 (timestamp)

日時を完全に指定する文字列。11-DEC-1998 17:13:21 はその一例である。

システム管理者 (system manager)

コンピュータ資源をユーザが使用できるようにしたり、資源の使用法を管理する制約を設定したりする人間。

システム・パスワード (system password)

ログインを開始する前にターミナルで必要なパスワード。

システム・ログイン・コマンド・プロシージャ (system login command procedure)

システム管理者が、ユーザがログインしたときに特定のコマンドが必ず実行されるように設定するプロシージャ。

実行可能イメージ (executable image)

プロセスの中で実行できるイメージ。実行すると、プロセスの実行用ファイルから実行可能イメージが読み込まれる。

指定ファイル (specification file)

Sort/Merge ユーティリティの中でソートに必要なコマンドと修飾子を指定するのに使用するコマンド・ファイル。

修飾子 (qualifier)

1 つあるいは複数のオプションを選択して、コマンド動詞またはコマンド引数を修正するコマンド文字列の部分。修飾子が存在する場合、それを適用するコマンド動詞またはパラメータの次に/qualifier[=option]の形式で指定する。たとえば、コマンド文字列“PRINT ファイル名/COPIES=3”では、COPIES 修飾子によりユーザが特定のファイルの出力を 3 部必要とすることを示す。

出力ファイル (output file)

処理操作の結果を含むファイル。たとえば、ソートあるいは編集されたファイル。

順次アクセス・モード (sequential access mode)

プログラムがレコードの出現する順番にレコードの検索あるいは書き込みを行うようなレコードの検索あるいは格納の方法。この場合、操作はファイル内の任意の位置から開始し、任意の位置で終了する。

順ファイル編成 (sequential file organization)

レコードが最初に書き込まれた順に構成されるファイル編成。レコードの長さは固定の場合もあれば、可変の場合もある。レコードは、レコード・アドレスの順にまたはランダムにアクセスできる。固定長レコードの場合は、相対レコード番号によってもランダムにアクセスできる。

照合シーケンス (collating sequence)

順序付けのために文字セット (たとえば、ASCII、DEC で定義している文字セット、EBCDIC など) の各文字に割り当てられた順序。

情報交換用米国標準コード (American Standard Code for Information Interchange(ASCII))

テキスト表現および通信プロトコルに使用されるアルファベット、句読点、数字、その他の特殊なシンボルを表現する 8 ビットの 2 進数の集合。

省略時の設定 (default)

ユーザが特に指定しない場合、自動的にコマンドで使用される数値または操作。ほとんどの場合、省略時の設定は通常のものあるいは使用される可能性の高いものである。

省略時のディスク (default disk)

省略時の設定で、ユーザが作成するすべてのファイルについて、システムが読み込みおよび書き込みを行うディスク。コマンドのファイル指定で明示的にデバイス名を指定しない場合は常に省略時のディスクが使用される。

省略時のディレクトリ (default directory)

ユーザがディレクトリ指定を行わない場合に、OpenVMS オペレーティング・システムが想定するディレクトリ。

ジョブ (job)

プロセスおよび存在するならばそのサブプロセス，およびこれらが作成するすべてのサブプロセスに等価な会計情報の単位。ジョブはバッチ型と会話型に分類される。たとえば，ユーザがシステムにログインした場合，ジョブ・コントローラはそのユーザの要求を処理する会話型ジョブを作成し，シンビオント・マネージャがコマンド入力ファイルをジョブ・コントローラに渡した場合，バッチ・ジョブを作成する。

ジョブ・ツリー (job tree)

すべてのプロセスとサブプロセスから構成され，メイン・プロセスが最上部にある階層構造。

侵入の試み (break-in attempt)

権限のない不正ソースが，システムにアクセスしようとする。最初にシステムにアクセスする場合はログインを介して行うので，侵入の試みとは非合法的なログインの試みを指す。この試みは，システム上でアカウントを保持していることが分かっているユーザのパスワードを推測や試行錯誤によって当てることで行う。

シンボル (symbol)

定義された場合，特定のコンテキストで特定の機能または実体 (たとえば，コマンド列，ディレクトリ名前，ファイル名など) を表す実体。

シンボル有効範囲 (symbol scope)

シンボルにアクセスできるコマンド・プロシージャ・レベルの集合。

数値式 (numeric expression)

算術演算子によって結合されたオペランドの集合である算術文。

スクローリング (scrolling)

1 行以上のテキストを縦方向に移動して表示するビデオ・ターミナルの機能。たとえば TYPE コマンドが入力された場合，最も古い表示は先頭行から削除され，新たな表示が画面の最下行に現れる。

制限付きアカウント (restricted account)

安全なログイン・プロシージャを持つ OpenVMS アカウント。このユーザは，システムまたはプロセスのログイン・コマンド・プロシージャの実行中，Ctrl/Y キー・シーケンスを使用できない。ログイン・コマンド・プロシージャの実行後，制御はユーザに戻される。

相対ファイル編成 (relative file organization)

各レコードがバケット内で同じ長さのセルを 1 つずつ占めているようなファイル編成。各セルに，ファイルの先頭からの相対位置を表す一連番号が割り当てられている。

ターミナル (terminal)

キーボードとビデオ画面あるいはプリンタを持つ周辺機器の一般名。プログラムの制御により，ユーザはターミナルのキーボードからコマンドおよびデータを入力でき，メッセージをビデオ画面あるいはプリンタに受信できる。端末とも呼ぶ。

ソフトウェア (software)

特定のコンピュータ・システムの操作に関連したイメージ，プロシージャ，規則，ドキュメンテーションの集合。たとえば，オペレーティング・システムはソフトウェアである。

タイムアウト (timeout)

デバイスが入出力転送を完了するまでの制限時間が満了すること。

大容量記憶デバイス (mass storage device)

データおよびその他のタイプのファイルを使用しない場合に格納するための入出力デバイス。典型的な大容量記憶デバイスは，ディスク，磁気テープ，フロッピー・ディスクを含む。

代理ログイン (proxy login)

ユーザがローカル・ノードにアカウントを保持している場合と同様に，リモート・ノードからローカル・ノードに実際にログインできるログイン。ただし，ユーザはアクセス制御文字列でパスワードを指定しない。リモート・ユーザは，アカウントを自分で保持することも，他のユーザとアカウントを共用することもできる。

単純文字 (simple character)

バージョン以外のすべてのファイル指定の構成要素に使用できる基本文字セット。

中央処理装置 (central processing unit (CPU))

プログラムの実行を含むすべての計算と入力および出力のルーチンを扱うハードウェア。すなわち，CPU はコンピュータ内で実際に計算を行う部分である。

ディスク (disk)

高速ランダム・アクセス・デバイス。数種類のディスクがある。フロッピー・ディスクは小型のフレキシブル・ディスクである。ハード・ディスクには固定式または可動式がある。可動式ディスクには，保護ケースに入った1つのハード・ディスクおよび保護ケースに入った複数の積み重ねられたディスクがある。

ディレクトリ (directory)

ディスクまたはテープに格納されたファイルの集合を簡単に一覧できるファイル。ディレクトリはファイルの名称，タイプ，バージョン番号を含む。また，ファイルの実際の位置の識別およびその属性の指定のための一意的な数値を含む。サブディレクトリを参照。

データ (data)

通信，分析，処理に適した形式を持ち，事実，概念，命令の任意の表現を示す一般的な用語。

デバイス (device)

データの受信，格納，送信が可能な，プロセッサに接続した任意の周辺装置の一般名称。カード・リーダー，ライン・プリンタ，ターミナルはデータ記録のためのデバイスの例である。磁気テープ・デバイス，ディスク・デバイスは大容量記憶デバイスの例である。ターミナル・ライン・インタフェース，プロセッサ間リンクは通信デバイスの例である。デバイスはハードウェアである必要はない。

デバイス名 (device name)

ファイルを格納するデバイス・ユニットを識別するファイル指定のフィールド。デバイス名はまた、データ転送要求の入出力周辺装置を識別するニーモニックを含む。デバイス名は、ニーモニック、次にコントローラ識別文字 (適用できる場合)、ユニット番号 (適用できる場合)、最後のコロンで構成される。

同値文字列 (equivalence string)

論理名テーブルの 1 つの論理名に結合した文字列。たとえば、同値文字列はデバイス名、別の論理名、ファイル指定の一部に結びついた論理名などである。

独立プロセス (detached process)

所有者のないプロセス。サブプロセスの木構造での親プロセス・ユーザがシステムにログインした場合、ジョブ・コントローラは独立プロセスを作成する。また、バッチ・ジョブを起動した場合、あるいは論理リンク接続要求に対してサービスを行う場合も独立プロセスを作成する。ジョブ・コントローラが作成したプロセスを所有しないため、このプロセスを独立プロセスと呼ぶ。DCL コマンド RUN/UIC およびプロセス作成システム・サービス (UIC を指定する) は、適当な特権プロセスの独立プロセスの作成を可能にする。

入力ストリーム (input stream)

コマンドとデータのソース。ユーザのターミナル、バッチ・ストリーム、コマンド・プロシージャなどがある。

入力ファイル (input file)

コンピュータに転送するためのデータを含むファイル。

しばしば、入力ファイルと出力ファイルは混同される。DCL は通常これらのファイルに対してプロンプトを表示する。しかし、ほとんどのシステム・ユーティリティはコマンド行での位置で入力ファイルと出力ファイルを識別する。使用するコマンドの構文に注意する必要がある。

ネットワーク (network)

相互接続された個々のコンピュータ・システムの集合。

ノード (node)

次のいずれかです。

1. ネットワーク内の他のコンピュータ・システムと通信可能な個々のコンピュータ・システム。
2. OpenVMS VAX システムの場合、中央処理装置 (CPU)、制御装置、メモリ・サブシステムなどの VAXBI インタフェース。それぞれ、VAXBI バス上の 16 個の論理記憶位置のうちの 1 つを占有する。
3. OpenVMS VAX システムの場合、システム通信サービス (SCS) ソフトウェアが認識する VAX プロセッサまたは HSC。

ノード指定 (node specification)

ファイル指定の最初のフィールド。このフィールドは、ネットワーク上のコンピュータの位置を識別する。

バージョン番号 (version number)

ファイル指定でファイル・タイプの次の次の数値フィールド。ファイルを編集した場合、バージョン番号が1つ上がる。

パーソナル・ログイン・コマンド・プロシージャ (personal login command procedure)

ユーザがシステム環境をカスタマイズするためのコマンド・プロシージャ。これに含まれているコマンドは、ユーザがログインするたびに実行される。

ハードウェア・デバイス (hardware device)

ライン・プリンタ、ターミナル、大容量記憶などの機械的なデバイスを含む物理的なコンピュータ装置。

ハードコピー・ターミナル (hardcopy terminal)

用紙に出力するターミナル。ターミナルを参照。

パスワード (password)

ユーザが自身の識別とそのアカウントへのアクセスの許可の証明のために提示する文字列。システム・パスワードおよびユーザ・パスワードの2種類がある。ユーザ・パスワードは、1次および2次パスワードを含む。

バッチ・ジョブ (batch job)

バッチ処理サブシステムの制御下でスケジューリングされ実行されるプログラム。バッチ・ジョブに対する入力制御は、ディスクに格納されたコマンド・プロシージャから指定され、出力先としてディスク・ファイルが指定される。

バッファ (buffer)

入力または出力操作中にデータ・レコードの一時的な記憶領域として使用される内部メモリの領域。

パラメータ (parameter)

次のいずれかです。

1. P1 ~ P8 のシンボルの範囲に対応して、コマンド・プロシージャに渡される値。コマンド・パラメータを参照。
2. ネットワーク管理コンポーネントのための運用時または永久データベースのエントリ。

反転ビデオ (reverse video)

省略時のビデオ・コントラストを反転させるビデオ・ターミナルの機能。省略時の表示が白地の背景に黒い文字の場合、反転ビデオは黒地の背景に白い文字となる。

反復変換 (iterative translation)

論理名の定義に別の論理名が含まれている場合に行われる、論理名の繰返し変換。

汎用デバイス名 (generic device name)

特別なユニットではなくデバイスのタイプを識別するデバイス名。特定のコントローラあるいはユニット番号が省略されたデバイス名。

ビデオ・ターミナル (video terminal)

オペレーティング・システムとの会話を表示するためのキーボードとビデオ画面 (またはモニタ)。ターミナルを参照。

ファイル (file)

ユーザにとって意味のある構造に配置されたデータ要素の集合。ファイルには任意の名前を付けることができる。また、プログラムあるいはデータを格納でき、システムがアクセスできる。アクセスには2つのタイプがある。ファイルの変更はできない読み込み専用と、ファイルの内容の変更も可能な読み込み/書き込みである。ボリュームを参照。

ファイル・パス (file path)

ファイル指定の中のディスクとディレクトリの部分。

ファイル指定 (file specification)

大容量記憶メディア上のファイルの一意的な名前。ノード、デバイス、ディレクトリ名、ファイル名、ファイル・タイプ、バージョン番号を識別する。

ファイル・タイプ (file type)

ピリオドに続く0～39文字の識別子から構成されるファイル指定のフィールド。通常このフィールドは、コンパイラあるいはアセンブラのリスト・ファイル、バイナリ・オブジェクト・ファイルなどの同一の使用法または属性を持つファイルの一般的なクラスを識別する。

ファイル名 (file name)

ファイル指定で、ファイル・タイプに先行する1～39文字のファイルの名称を含むフィールド。

ファンクション・キー (function key)

特別なシグナルをオペレーティング・システムに送信するキーボード・キー。ファンクション・キーはFxと記述され、xはキーに結合した番号である。たとえば、MailでF9キーと押した場合、メッセージの転送をシステムに通知する。

フィールド (field)

論理レコードの連続したバイトの集合。

フォーリン・コマンド (foreign command)

コマンド・インタプリタでDCLコマンドとして認識されない名称を持つイメージを実行するシンボル。

フォーリン・ファイル指定 (foreign file specification)

ファイル指定がOpenVMS構文または形式に従っていないファイル。

フォルダ (folder)

メール・メッセージを格納できるファイルの区分。

複合文字 (compound character)

単純文字と拡張文字セットの文字の組み合わせ。

物理デバイス名 (physical device name)

システムに対して物理デバイス (記憶ディスクあるいはターミナルなど) を一意的に識別する文字列。

プライベート・ボリューム (private volume)

プロセスが排他的に使用するために割り当てた大容量記憶媒体。

プリント・フォーム

印刷のためにページ設定とストックを定義する属性の集合。

プログラム (program)

特別の結果を目的にした一連の命令。プログラミング言語はプロシージャを作成するためのものであり、コンピュータで実行できる。イメージを参照。

プログラム・スタブ (program stub)

コマンド・プロシージャの作成中に、試験のために使用するコードの一時的な部分。プログラム・スタブは、通常、メッセージを出力して置換するプロシージャを知らせる。

プロセス (process)

システム・ソフトウェアでスケジューリングされる基本的実体。プロセスはイメージを実行するコンテキストを提供する。プロセスはアドレス空間、およびハードウェア・コンテキストとソフトウェア・コンテキストから構成される。

プロセス・デフォルト・ディレクトリ (process default directory)

システムは、自動的にユーザの最上位ディレクトリをログイン時のプロセス・デフォルト・ディレクトリにする。

プロンプト (prompt)

ユーザに入力を促すためにターミナル上に表示される文字列。

ベスト・エフォート・デリバリ (best-effort delivery)

回線障害などのエラーが発生した場合に、エラーの回復をせずにデータの受け渡しを試みるネットワーク・プロトコル。

ヘルプ (help)

HELP コマンドの使用に適合した形式のテキスト・ファイル。オンライン・ヘルプは最大 9 レベルの検索を提供できる。

保護コード (protection code)

システム・ユーザがファイルまたは他の保護オブジェクトに対してどのアクセス・カテゴリを持てるか、またアクセスした場合アクセス・カテゴリが何を実行できるかを指定する文字列。

保護されたオブジェクト

システムがアクセスを制御する共用可能情報を格納したオブジェクト。オブジェクトも参照。

ホスト (host)

ネットワークに接続されたシステム。ノードも参照。

ボリューム (volume)

ディスク・パックあるいは磁気テープなどの大容量記憶メディア。ボリュームはファイル構造の最大の論理ユニット。

ボリューム・セット (volume set)

1 つまたは複数の大容量記憶媒体にある、ファイル構造をしたデータの集まり。

マスター・ファイル・ディレクトリ (master file directory (MFD))

ディスクのメイン・ディレクトリが格納されているファイル。

メモリ (memory)

データまたは命令がバイナリ・ワード形式で置かれる一連の物理的な位置。メモリの各位置はアドレス可能であり、その内容は変更できる。メモリを大容量記憶デバイスと混同してはならない。

文字列 (character string)

印字可能な文字の連続した集まり。

文字列 (string)

連続して結合された文字。エディタがテキスト・ファイルで語または句を検索する場合、文字列を検索する。コマンドを形成する文字列はコマンド列と呼ばれる。

ユーザ登録ファイル (user authorization file (UAF))

UAFを参照。

ユーザ・パスワード (user password)

ユーザに対応したパスワード。ユーザはログイン時にこのパスワードを正しく指定しなければ、システムへのアクセスを許可されない。ユーザ・パスワードには1 次パスワードと2 次パスワードの2 種類がある。必要な場合は、1 次パスワードの後に2 次パスワードを入力する。

優先順位 (priority)

プロセスの実行中にシステム資源を取得する優先権を決定する、プロセスに割り当てられたランク。

ユーティリティ (utility)

プログラム開発ユーティリティ (エディタ、リンカ)、ファイル管理ユーティリティ (ファイル・コピー、ファイル・フォーマット変換ユーティリティ)、オペレーション管理ユーティリティ (ディスク・クォータ、自己診断プログラム) などの関連した汎用機能の集合を提供するプログラム。

読み込み (read)

イメージがデータを受け入れることまたはその機能。たとえば、TYPE コマンドを実行する場合、システムはディスクから指定のファイルを読み込み、それをターミナルへ書き込む。書き込みも参照のこと。

ライン・エディタ (line editor)

行単位でファイルへの追加，削除を行うプログラム。

ライン・プリンタ (line printer)

ファイルの 1 行分を同時にプリントする出力デバイス。低速のデバイスでは処理が終了しないような大量の出力をプリントするために使用する。ほとんどすべてのシステムは，ライン・プリンタ用のデバイスを持つ。ある場合，ライン・プリンタは実際に高速のターミナルである。

ランダム・アクセス (random access)

直前に検索あるいは書き込みを行ったデータの位置に依存しないデータの検索あるいは書き込みの手法。ランダム・アクセスは，すべての情報が等しくアクセス可能であるメモリあるいは大容量記憶デバイスを参照する。

リモート・ノード (remote node)

ネットワークの中の，現在ログインしているノード以外の任意のノード。

レキシカル関数 (lexical function)

DCL (DIGITAL コマンド言語) コマンド・インタプリタがコマンドを解釈する前に評価と置換を行うコマンド言語の構成。レキシカル関数は現在のプロセスに関する情報 (たとえば，ユーザ識別コード (UIC) または省略時のディレクトリ)，あるいは別の文字列に関する情報 (たとえば，文字列の長さまたは位置) を戻す。

レコード管理サービス (RMS)

RMS (Record Management Services) を参照。

レコード指向デバイス (record-oriented device)

ターミナル，ライン・プリンタ，カード・リーダーなどのデバイス。レコード指向デバイスの物理レコードは，プログラムが 1 回の入出力操作でアクセスできる最大のデータ単位である。

レコードのソート (record sorting)

レコードの並べ替えプロセス。レコード自体は変えずに，順序を変えてレコード全体の出力ファイルを生成する。

レコード・ファイル・アドレス (record file address (RFA))

ファイル内のレコードの一意的なアドレス。RFA を使用すると，すでにアクセスしたレコードに，後でランダムにアクセスできる。このアクセスは，ファイルの編成に関係なく実行できる。

ローカル・シンボル (local symbol)

次のいずれかです。

1. 定義されたモジュール内でのみ意味を持つシンボル。言語処理プログラムにグローバル・シンボルとして認識されないシンボルはローカル・シンボルと見なされる。言語処理プログラムはローカル・シンボルの参照と定義の一致を確認して結合する。これらはリンカに認識されず，別のオブジェクト・モジュールでは使用できない。しかし，リンカからデバッガには渡すことができる。グローバル・シンボルを参照。

2. 現在のコマンド・レベルおよびその後起動されるレベルでのみアクセス可能なコマンド言語シンボル。定義しているコマンド・レベルが終了すると削除される。

ローカル・ノード (local node)

ユーザが物理的に配置されているネットワーク・ノード。

ロギング・アウト (logging out)

DCL (DIGITAL コマンド言語) コマンド LOGOUT を入力する文字列。オペレーティング・システムにユーザが特定のターミナルを使用して終了することを通知する。

ロギング・イン (logging in)

システムに対するユーザの確認。ユーザがログ・インする場合、システムのプロンプトに対してユーザ名、パスワードを入力する。ユーザ名およびパスワードがシステムのアカウントに一致した場合、そのユーザはシステムへのアクセスを許可される。

ログイン・クラス (login class)

ユーザがシステムにログインする方法。システム管理者は、ログイン・クラス (ローカル、ダイヤルアップ、リモート、バッチ、ネットワーク) に基づいてシステム・アクセスを制御できる。

ログイン・コマンド・プロシージャ (login command procedure)

ログイン時およびバッチ・ジョブの開始時に自動的に実行されるコマンド・プロシージャ。

ログイン・ディレクトリ (login directory)

ユーザのログイン時に LOGINOUT が設定する省略時のディレクトリ。

論理式 (logical expression)

値が真または偽のどちらかである式。

論理デバイス名 (logical device name)

隠されたデバイス名を短い意味のある名称と等価にする文字列。

論理名 (logical name)

システム・オブジェクトを示すため別の文字列の代わりとするためにユーザが指定した名前。システム・オブジェクトは、例えばファイル、ディレクトリ、デバイス、キュー等。論理名は論理名テーブルで管理される。

論理名テーブル (logical name table)

論理名および対応する等価名を含むテーブル。論理名はプライベートにも共用にもすることができる。省略時の共用テーブルは、ジョブ、グループ、システム、クラスタワイド・システム、クラスタワイド・パレント・テーブル。

ワイルドカード文字 (wildcard character)

アスタリスク(*)やパーセント(%)記号などの非英数字文字で、ファイル指定のファイル名、ファイル・タイプ、バージョン番号などの一部またはすべてと置き換えて使用され、そのフィールドのすべてを指定する。

割り当てステートメント (assignment statement)

DCL (DIGITAL コマンド言語) で、シンボル名を文字列または数値と結合して使用すること。シンボルはシステム・コマンドの同義語を定義でき、コマンド・プロセスの変数としても使用できる。

@
 コマンド・プロシージャ実行コマンド . . . 13-18

A

ACE(アクセス制御エントリ) 10-13
 ACL(アクセス制御リスト)
 省略時の保護 10-6
 ファイルの保護 3-20
 ALLOCATE コマンド 6-3
 形式 6-4
 ASCII(American Standard Code for Information
 Interchange) 3-14
 ASCII character set 5-2
 ASCII 文字セット A-1
 ASSIGN コマンド 11-3
 アクセス・モード 11-6
 ATTACH コマンド 8-52

B

Backspace キー 2-18, 2-20
 BATCH\$RESTART シンボル
 コマンド・プロシージャ内での使用 13-22
 BATCH.COM コマンド・プロシージャ B-27
 実行例 B-32
 BUFFER コマンド 8-48

C

CALC.COM コマンド・プロシージャ B-25
 実行例 B-27
 CDU (コマンド定義ユーティリティ) 13-44
 Character sets 5-6
 ISO Latin-1 Multinational 5-2
 Circumflex character 5-6
 CLEANUP.COM コマンド・プロシージ
 ヤ 13-17
 CLOSE コマンド
 コマンド・プロシージャ内の
 論理名の削除 11-4
 Code compilers 5-9
 COMPILE_FILE.COM コマンド・プロシージ
 ヤ B-32
 実行例 B-34
 CONNECT コマンド
 /CONTINUE 修飾子 16-9
 CONTINUE コマンド 16-5

CONTROL_Y コマンド上
 Ctrl/Y の有効化 13-36
 CONVERT.COM コマンド・プロシージャ . . . B-1
 実行例 B-5
 COPY コマンド 3-15
 Mail の 7-17
 /SINCE 修飾子 3-15
 CREATE/DIRECTORY コマンド 4-4, 4-9
 CREATE/NAME_TABLE コマンド
 /PROTECTION 修飾子 11-25
 CREATE/PROTECTION コマンド 3-20
 CREATE コマンド 3-14
 \$CREPRC システム・サービス 11-30
 Ctrl/A キー・シーケンス 2-20
 Ctrl/B キー・シーケンス 2-15, 10-8
 コマンドの再呼び出し 2-14, 2-19
 Ctrl/C キー・シーケンス 2-19
 メール・メッセージの取り消し 7-5, 7-12
 Ctrl/E キー・シーケンス 2-20
 Ctrl/T キー・シーケンス
 DCL コマンドへの割り込み 2-19
 プロセスの状態をチェックする 1-21
 Ctrl/U キー・シーケンス 2-18
 Ctrl/W キー・シーケンス
 画面再表示 16-5
 Ctrl/Y キー・シーケンス
 DCL コマンドの取り消し 2-19
 DCL コマンドの割り込み 2-19
 PIPE コマンドへの割り込み 14-52
 アクション・ルーチンの設定 13-32
 イメージの割り込み 16-5
 コマンド・プロシージャのキャンセル . . 13-30
 コマンド・プロシージャへの割り込み . . 13-30
 終了 13-31
 使用不能にする 13-36
 入力結果 13-33
 有効化 13-36
 リモート・セッションの強制終了 1-25
 Ctrl/Z キー・シーケンス
 Mail でのファイルの送信 7-12
 ファイル終了文字の終端としての 2-18
 メール・メッセージの送信 7-5
 Ctrl キー
 一般的な 2-18

D

DBG\$INPUT 論理名	11-24
DBG\$OUTPUT 論理名	11-24
DCL\$PATH 論理名	12-38, 12-40
DCL(DIGITAL コマンド言語)	
定義	2-1
DCL コマンド	
PIPE コマンド	2-7
大文字	2-7
大文字と小文字の区別	2-2
空文字の使用	2-7
句読点の必要性	2-7
構成	2-3
構文	2-3, 2-5
コマンド・インタプリタ内で実行されるコマンド	14-11
コマンド行の編集	2-16
コマンド修飾子	2-9
コマンド・プロシージャ内での順序	13-7
コマンド・プロシージャへの取り込み	13-2
小文字	2-7
再呼び出し	
下向き矢印キーによる	2-19
自動的なフォーリン・コマンド	
制限	12-41
自動フォーリン・コマンド	12-40
修飾子の短縮	2-9
出力先の切り換え	14-10
省略時の値	2-6
省略時の修飾子	2-9
短縮	2-8
コマンド・プロシージャでの	2-8
定位置修飾子	2-10
動詞	2-4
取り消し	2-19
入力	2-2
入力規則	2-7
パラメータ	2-5
パラメータ修飾子	2-10
パラメータの指定	2-8
複数行のコマンド	2-6
複数の	2-7
ブランクの挿入	2-7
プロンプト	2-4
矛盾する修飾子	2-10
要素の最大数	2-7
割り込み	2-19
.DDIF ファイル	
Mail 内での	7-11
DEALLOCATE コマンド	6-4
DEASSIGN コマンド	11-6, 11-9, 11-29, 11-31
Debugger	
省略時出力ストリーム	11-24
省略時入力ストリーム	11-24
DECnet	
完全なノード名の明記	3-7

DECnet (続き)

トランスポートの指定	7-6
複数のファイル	3-16
リモート・システムへの接続の消失	1-25
DEC Text Processing Utility(DECTPU)	8-1
DEC 各国語文字セット	A-1
DEFINE/KEY コマンド	2-18
DEFINE コマンド	11-3, 11-9
Mail での/KEY 修飾子	7-25, 7-30
初期化ファイルでの	7-26
/USER_MODE 修飾子	11-34
アクセス・モード	11-6
論理名	11-3
DELETE BUFFER コマンド	8-46
DELETE/SYMBOL コマンド	
/GLOBAL 修飾子	12-6
DELETE コマンド	3-18
/ENTRY 修飾子	16-20
FSSEARCH レキシカル関数とともに使用	15-11
Mail の	7-20
DIR.COM コマンド・プロシージャ	B-8
実行例	B-11
DIRECTORY コマンド	4-5
Mail の	7-4, 7-18
Disk defragmenters	5-9
DISMOUNT コマンド	6-8
/NOUNLOAD 修飾子	6-9
.DIS ファイル・タイプ	
を持つ配布リスト	7-10

E

EDITALL.COM コマンド・プロシージャ	B-15
実行例	B-17
EDIT コマンド	
/EDT 修飾子	
/READ_ONLY 修飾子	3-17
EVE の起動	8-5
/RECOVER 修飾子	8-39
/TPU 修飾子	
/READ_ONLY 修飾子	3-17
Enter キー	2-18
Escape character	5-2
EVE	
EDIT コマンド行修飾子	8-32
/READ_ONLY または/NOWRITE の上書き	8-34
開始位置	8-33
作業ファイルのための	8-34
バッファの変更	8-34
EVE と DCL との切り替え	8-52
Mail エディタとしての	7-23
mail での使用方法	7-22
SPAWN コマンドによるサブプロセスの作成	8-52 ~ 8-53
ウィンドウ	8-49
キー	8-49

EVE

ウィンドウ (続き)	
コマンド	8-50
上書モード	8-14
ボックス編集	8-26
カーソルの移動	8-9
キー	8-9
操作手順	8-12
キー	
定義	
VT100 シリーズ・ターミナル	8-7
キー定義	
VT200 シリーズ・ターミナル	8-7
VT300 シリーズ・ターミナル	8-7
VT400 シリーズ・ターミナル	8-7
起動	8-5
検索リストの使用	8-35
複数の入力ファイルの使用	8-35, 8-36
ワイルドカード・ディレクトリ名の使	
用	8-35, 8-36
ワイルドカードの使用	8-35 ~ 8-36
機能の要約	8-2
キー名の表記法	8-2
コマンド	
BUFFER	8-48
DELETE BUFFER	8-46
EXIT	8-9
FIND	8-27
GET FILE	8-48
HELP	8-4
INCLUDE FILE	8-48
OPEN SELECTED	8-48
RECOVER BUFFER	8-39
REPLACE	8-31, 8-32
RESTORE BOX SELECTION	8-26
RESTORE SELECTION	8-26
SET BUFFER	8-46
SET FIND CASE EXACT	8-28, 8-31
SET PENDING DELETE	8-25
SHOW BUFFERS	8-47
SPAWN	8-52 ~ 8-53
SPLIT WINDOW	8-51
WRITE FILE	8-8, 8-49
書式化	8-41
テキストの書式化	8-41
コマンドの入力	8-6
あらかじめ定義済みキー	8-7
定義済みキー	8-7
サブプロセスの作成	8-52
終了	8-8, 8-9
省略時のエディタとしての	8-2
セッションの終了	8-9
挿入モード	8-14
ボックス編集	8-26
定義	8-1
テキストの移動	8-18
コマンド	8-19
操作手順	8-21

EVE

テキストの移動 (続き)	
編集キー	8-19
テキストの検索	8-26
FIND コマンド	8-27
大文字と小文字の区別	8-28
検索方向	8-27
コマンド	8-26
マーク付け	8-30
テキストのコピー	8-21
操作手順	8-21
テキストの削除	8-15
コマンド	8-16
作成手順	8-17
編集キー	8-16
保留削除	8-25
テキストの挿入	8-13
テキストの置換	8-15, 8-31
編集キー	8-16
テキストの入力	
上書モード	8-14
コマンド	8-14
操作手順	8-15
挿入モード	8-14
特殊文字	8-13
ファイルの挿入	8-13
編集キー	8-14
テキストの復元	
コマンド	8-16
作成手順	8-17
ボックス編集	8-26
保留削除操作の後	8-26
テキストの検索	
ワイルドカードの使用	8-29
バッチ・ジョブ・ログ・ファイルの読み込	
み	16-14
バッファ	
BUFFER コマンド	8-48
GET FILE コマンド	8-48
OPEN SELECTED コマンド	8-48
SET BUFFER コマンド	8-46 ~ 8-47
ウィンドウの分割	8-51
削除	8-43, 8-46
作成	8-43
状態の変更	8-46 ~ 8-47
情報の表示	8-45
操作	8-43
定義	8-43
複数	8-47
2 つのセクションの表示	8-51
2 つのバッファの編集	8-51
メッセージの表示	8-47
バッファ・ジャーナリング	
RECOVER BUFFER コマンド	8-39
/RECOVER 修飾子	8-39
使用可能	8-41
使用不可能	8-40
編集の回復	8-39, 8-40

EVE (続き)

バッファ・ジャーナル	
ファイル	8-38
ファイル名	8-38
バッファ変更ジャーナリング	8-37
コマンド	8-37
使用不可能化	8-37
定義	8-37
ファイルの書き込み	8-49
ファイルの読み込み	8-48
ファイル・バックアップ	8-37
ヘルプ	8-3
HELP コマンド	8-4
キーパッド	8-3
編集キー	8-41
編集のセーブ	8-8
EXIT コマンドによる	8-9
WRITE FILE コマンドによる	8-8
ボックス編集	8-22
上書モード	8-26
コマンド	8-23, 8-25
操作手順	8-24
挿入モード	8-26
テキストのカット	8-23
テキストの選択	8-22
テキストの復元	8-26
テキストのペースト	8-23
保留削除の効果	8-26
モードの変更	8-15
EVE コマンド	
ATTACH	8-52
GOTO	8-30
MARK	8-30
QUIT	8-9
EXIT コマンド	
Mail の	7-3
コマンド・プロシージャ内の	13-11
コマンド・プロシージャを使用	13-25
条件コード	13-38
使用する場合	13-11
Extended File Specifications	
ファイル名	
DCL コマンド・パラメータ内での使	
用	13-42
EXTRACT コマンド	
Mail での	7-12

F

FSCONTEXT レキシカル関数	15-8
FSCVTIME レキシカル関数	15-13
FSDIRECTORY レキシカル関数	12-23
FSELEMENT レキシカル関数	15-13
F\$ENVIRONMENT レキシカル関数	
省略時のファイル保護の変更	15-5
FSEXTRACT レキシカル関数	15-13
FSFAO レキシカル関数	15-15

F\$GETQUI レキシカル関数	
キュー情報の取得	15-6
F\$GETSYI レキシカル関数	
情報の取得	
システム	15-6
F\$INTEGER レキシカル関数	
データ・タイプの変換	15-17
データタイプの変換	12-25
データの評価	15-18
F\$MODE レキシカル関数	
ログイン・プロシージャの中の	16-12
F\$PARSE レキシカル関数	15-13
F\$PID レキシカル関数	15-7
プロセス情報の取得	15-7
F\$SEARCH レキシカル関数	
DELETE コマンドでの使用	15-11
ファイルの検索	15-10
F\$STRING レキシカル関数	
データ・タイプの変換	15-17
データタイプの変換	12-25
F\$VERIFY レキシカル関数	15-3
F\$LENGTH レキシカル関数	
F\$LOCATE とともに使用	15-13
F\$LOCATE レキシカル関数	
F\$LENGTH とともに使用	15-13
F\$LOGICAL レキシカル関数	15-11
Files-11 ディスク構造	6-6
FILE コマンド	
Mail の	7-17
FORTUSER.COM コマンド・プロシージャ	
ヤ	B-19
実行例	B-22
FORWARD コマンド	
Mail の	7-14
F\$TRNLNM レキシカル関数	15-11
LNMSDCL_LOGICAL 論理名	11-21
論理名の変更	15-11
F\$TYPE レキシカル関数	
シンボルの確認	15-18
シンボルの識別	12-25

G

\$GETDVI レキシカル関数	15-9
GET FILE コマンド	8-48
GETPARMS.COM コマンド・プロシージャ	
ヤ	B-13
実行例	B-15

H

HELP/MESSAGE コマンド	1-24
HELP コマンド	1-22
EVE の	8-3

I

IF コマンド	
コマンド・プロシージャ内での使用方	
法	13-8
INCLUDE FILE コマンド	8-48
INITIALIZE コマンド	6-5
Files-11 ディスク構造上の	6-6
形式	6-5
INQUIRE コマンド	
READ コマンドとの比較	14-6
コマンド・プロシージャ内での使用方	
法	13-7
シンボルとバッチ・ジョブとの使用	14-7
ISO Latin-1 文字セット	A-1

J

JTQUOTA 値	11-30
-----------	-------

K

Known images	5-9
--------------	-----

L

LINK command	5-9
LISTER.COM コマンド・プロシージャ	B-23
実行例	B-25
LNMSCLUSTER_TABLE 論理名	11-22
LNMSCLUSTER 論理名	11-21
LNMSDCL_LOGICAL 論理名	
LNMSFILE_DEV	11-21
SHOW LOGICAL コマンド	11-21
LNMSFILE_DEV 論理名	
検索リスト	11-11, 11-21
プロセス固有定義	11-30
LNMSGROUP 論理名	11-21, 11-22
LNMSJOB 論理名	11-21, 11-23
LNMSPERMANENT_MAILBOX 論理名	11-21
LNMSYSCLUSTER_TABLE 論理名	11-22
LNMSYSCLUSTER 論理名	11-22
LNMSYSTEM 論理名	11-22, 11-24
LNMSTEMPORARY_MAILBOX 論理名	11-22
LOCKPWD フラグ	1-7
LOGIN.COM ファイル	
論理名の使用	11-3
LOGOUT コマンド	1-24
/FULL 修飾子	1-25

M

MAIL\$INTERNET_MODE 論理	7-6
MAIL\$INTERNET_TRANSPORT 論理	7-6
MAIL\$KEYDEF.INI ファイル	7-26
サンプル	7-26

MAIL.MAI ファイル	7-16
MAILEDIT.COM コマンド・プロシージャ	
ヤ	7-24, B-18
実行例	B-19
Mail コマンド	7-2
/EDIT 修飾子	7-22
MAIL コマンド	
/SUBJECT 修飾子	7-11
Mail サブディレクトリ	
作成	7-16
MAIL フォルダ	
MAIL	7-4
NEWMAIL	7-3
WASTEBASKET	7-20
削除	7-18
作成	7-17
選択	7-17
リストの表示	7-17
Mail ユーティリティ (MAIL)	
EVE の使用方法	7-22, 7-23
エディタの設定	7-23
起動	7-2
キーパッド・コマンド	7-24
コマンドの要約	7-26
初期化ファイル	7-26
セキュリティに関する考慮事項	7-21
テキスト・エディタの使用方法	7-22
トランスポート	7-6
ネットワークを介したメッセージの送信	7-6
配布リストへのメッセージの送信	7-8
ファイルの送信	7-10
DCL レベルから	7-11
変更点と拡張機能	
SET FORWARD コマンド	7-15
メッセージの交換	7-28
メッセージの削除	7-20, 7-28
メッセージの表示	7-3
メッセージの読み込み	7-27
メッセージへのファイルの追加	7-13
メッセージをファイルに落とす	7-12
問題と制約	
入れ子になった引用符を含むアドレスへの返	
信	7-14
MARK コマンド	7-30
Merge コマンド	
/NODUPLICATES 修飾子	9-18
/STABLE 修飾子	9-18
MERGE コマンド	9-16
Sort/Merge ユーティリティ参照	
/KEY 修飾子	9-17, 9-18
MFD(マスタ・ファイル・ディレクトリ)	4-2
MIME\$FILETYPES.DAT	7-34
MIME\$MAILCAP.DAT	7-34
MIME ユーティリティ	7-33
コマンド	7-38
ファイルのエンコード	7-38
ファリルを抽出	7-37

MOUNT コマンド	6-3, 6-6
/FOREIGN 修飾子	6-7
形式	6-6
MOVE コマンド	
Mail の	7-17

O

ON コマンド	
Ctrl/Y アクション・ルーチンの設定	13-32
OPEN SELECTED コマンド	8-48
OpenVMS Cluster システム	
デバイス名	6-3
クラス・フィールドの割り当て	6-3
形式	6-3
デュアル・パス	6-3
OpenVMS 画面管理ソフトウェア	
コマンドの再呼び出し	2-16
OPEN コマンド	
コマンド・プロシージャ内の	
論理名の作成	11-4
プロセスパーマネント・ファイル	11-33
プロセスパーマネント論理名	11-33

P

Page and swap files	5-10
PID 番号	B-12
FSPID レキシカル関数による取得	15-7
とプロセス・コンテキスト	16-3
PIPE コマンド	2-7, 11-33, 14-47 ~ 14-54
割り込み	14-52
PQLS_JTQUOTA 制限リスト値	11-30
PQL_DJTQUOTA システム・パラメータ	11-30
PQL_MJTQUOTA システム・パラメータ	11-30
PRINT コマンド	3-20 ~ 3-23
Mail の	7-21
修飾子	3-24
PURGE コマンド	3-19
Mail の	7-20, 7-28

Q

QUIT コマンド	7-3
EVE の	8-9

R

READ コマンド	
INQUIRE コマンドとの比較	14-6
Mail の	7-3, 7-18
Mail の/NEW 修飾子	7-3, 7-20
コマンド・プロシージャ内の	
論理名の使用	11-4
コマンド・プロシージャへのデータの読み込み	14-6
RECALL コマンド	2-15
/ERASE 修飾子	2-15, 10-8

RECALL コマンド (続き)	
矢印キーの使用	2-15
REMINDER.COM コマンド・プロシージャ	
ヤ	B-5
実行例	B-8
RENAME コマンド	3-17
REPLY コマンド	
Mail の	7-14
\$RESTART シンボル	
コマンド・プロシージャ内での使用	13-22
RMS	
Mail でのファイル・タグ	7-11
RSX システム	
UIC 形式によるディレクトリ名の指定	4-11
RUN コマンド	
/JOB_table_QUOTA 修飾子	11-30
検索リストによる	11-16
プロセスによる	16-4

S

SEARCH コマンド	
Mail の	7-5
SELECT コマンド	
Mail の	7-4, 7-18
SEND コマンド	
Mail の	7-5, 7-10
/EDIT 修飾子	7-23
SET [NO]AUTO_PURGE コマンド	
Mail の	7-28
SET [NO]CC_PROMPT コマンド	7-30
SET [NO]FORM コマンド	
Mail の	7-29
SET [NO]MAIL_DIRECTORY コマンド	
Mail の	7-31
SET [NO]PERSONAL_NAME コマンド	
Mail の	7-31
SET [NO]QUEUE コマンド	
Mail の	7-29
SET CONTROL=T コマンド	1-21
SET CONTROL=Y コマンド	13-36
SET COPY_SELF コマンド	7-30
SET DEFAULT コマンド	
省略時のディレクトリの設定	4-6
省略時のデバイスの設定	4-7
SET EDITOR コマンド	
Mail の	7-23, 7-31
SET ENTRY コマンド	16-17
修飾子	16-17
SET FILE コマンド	7-29
Mail の	7-18
SET FOLDER コマンド	7-29
Mail の	7-18
SET FORWARD コマンド	7-31
Mail の	7-15
SET HOST コマンド	1-10
SET NOCONTROL=Y コマンド	13-36

SET NOON コマンド	
コマンド・プロシージャ内での	13-29
コマンド・レベルによる	13-30
SET ON コマンド	
コマンド・レベルによる	13-30
SET OUTPUT_RATE コマンド	16-14
SET PASSWORD コマンド	1-14
/GENERATE 修飾子	1-14
/SECONDARY 修飾子	1-16
パスワード自動生成	1-14
SET PREFIX コマンド	16-15
SET PROCESS コマンド	
/NAME 修飾子	16-3
SET PROTECTION コマンド	
/DEFAULT 修飾子	10-6
SET QUEUE コマンド	
/ENTRY 修飾子	16-17
Mail の	7-21
修飾子	16-17
SET SECURITY コマンド	
/ACL 修飾子	10-6
/PROTECTION 修飾子	3-20, 4-9, 10-7
/PROTECTI 修飾子	10-4
オブジェクトのセキュリティ・プロファイルの変 更	10-4
新保護コードの確認	10-4
SET SIGNATURE_FILE コマンド	
メール内での	7-13
SET SYMBOL コマンド	12-28
動詞文字列の変換	12-28
SET TERMINAL コマンド	2-17, 2-18
/BROADCAST 修飾子	1-21
/INSERT 修飾子	2-17
/LINE_EDIT 修飾子	2-17
/OVERSTRIKE 修飾子	2-17
/WRAP 修飾子	2-17
SET TERMINAL コマンド	
/OVERSTRIKE 修飾子	2-17
SET VERIFY コマンド	16-15
SHOW ALL コマンド	
Mail の	7-31
SHOW AUTO_PURGE コマンド	
Mail の	7-28
SHOW BUFFERS コマンド	8-47
SHOW COPY_SELF コマンド	7-31
SHOW DEFAULT コマンド	4-7
SHOW DEVICES コマンド	6-2
SHOW EDITOR コマンド	
Mail の	7-31
SHOW ENTRY コマンド	3-21, 16-11, 16-19
SHOW FILE コマンド	7-29
Mail の	7-18
SHOW KEY コマンド	7-30
SHOW LOGICAL コマンド	11-11
LNMSDCL_LOGICAL 論理名	11-21
アクセス・モード	
表示	11-13
ディレクトリ・テーブルの構造	

SHOW LOGICAL コマンド	
ディレクトリ・テーブルの構造 (続き)	
表示	11-18
プロセスパーマネント・ファイル名	
表示	11-11
論理名テーブル	
表示	11-12
SHOW TRANSLATION コマンド	11-21
SHOW PROCESS コマンド	10-2
/ALL 修飾子	16-2
SHOW QUEUE コマンド	3-21, 16-19
/FULL 修飾子	16-14
SHOW SECURITY コマンド	
オブジェクトのセキュリティ・プロファイルの表 示	10-3
SHOW SYMBOL コマンド	12-6, 12-21
SHOW TERMINAL コマンド	2-16
SHOW TRANSLATION コマンド	11-11
SMTTP (Simple Mail Transfer Protocol)	
トランスポートの指定	7-6
Sort/Merge ユーティリティ (SORT/MERGE)	
Sort/Merge ユーティリティ (高性能) も参照	
/KEY 修飾子	9-7, 9-8
/NODUPLICATES 修飾子	9-10
SORT コマンド	9-3, 9-15
/STABLE 修飾子	9-9
指定ファイル	9-19
形式	9-19
コマンドの無効化	9-20
コメントの挿入	9-20
修飾子の順序	9-20
修飾子	9-28 ~ 9-45
SORT および MERGE コマンドでの使 用	9-28
指定ファイル	9-37
出力ファイル指定	9-33
入力ファイルの指定	9-32
出力ファイル	9-11
照合順序	9-13
ASCII	9-13
/COLLATING_SEQUENCE 修飾 子	9-13
EBCDIC	9-13
NCS	9-13
各国語対応	9-13
省略時の設定	9-13
ユーザ定義	9-13
省略時のキーの使用	9-4
性能の向上	9-24
/STATISTICS 修飾子	9-24
作業ファイルの管理	9-27
ソート中の	9-25, 9-27
ワーキング・セット拡張領域の変 更	9-28
ソート	
キー・フィールドによる	9-10
同一キー・フィールドを使用	9-9
非文字データ・ファイル	9-10

Sort/Merge ユーティリティ (SORT/MERGE)	
ソート (続き)	
複数のキー・フィールドの使用	9-8
レコード	9-12
ターミナルからのレコードの入力	9-18
バッチ・ジョブとして実行	9-15
コマンド・プロシージャ	9-15
入力レコードの包含	9-15
ファイルのソート	9-3
ファイルのマージ	9-16
キー・フィールドによってソートされた場 合	9-17
レコードのマージ	
同一キー・フィールドによる	9-18
Sort/Merge ユーティリティ (高性能)	
ルーチン	9-2
SORT コマンド	
Sort/Merge ユーティリティ参照	
SPAWN コマンド	8-52, 16-4
/NOLOGICAL_NAMES 修飾子	16-8
/NOSYMBOL 修飾子	16-8
/NOWAIT 修飾子	16-5
SPLIT WINDOW コマンド	8-51
STOP コマンド	
コマンド・プロシージャ内での	13-12
SUBMIT/AFTER コマンド	16-11
SUBMIT コマンド	16-10, 16-11
/LOG 修飾子	16-15
/RESTART 修飾子	16-20, 16-21
コマンド・プロシージャを使用	13-22
コマンド・プロシージャへパラメータを渡 す	14-5
修飾子	16-18
複数のコマンド・プロシージャの指定	16-12, 16-13
SYLOGIN.COM コマンド・プロシージャ	16-10
SYNCHRONIZE コマンド	16-21, 16-22
SYSS\$COMMAND 論理名	11-21
コマンド・プロシージャで使用	11-34
再定義	11-36
プロセスパーマネント	11-32
SYSS\$COMMON 論理名	11-24
SYSS\$DISK 論理名	11-19, 11-21
SYSS\$ERRORLOG 論理名	11-24
SYSS\$ERROR 論理名	11-21
再定義	11-35
プロセスパーマネント	11-32
ログ・ファイル内の～への出力	16-15
SYSS\$EXAMPLES 論理名	11-24
SYSS\$HELP 論理名	11-24
SYSS\$INPUT 論理名	11-19, 11-21, 14-7
個別ファイルとして定義	14-8
再定義	11-33, 14-7
定義	14-5
プロセス永久ファイル	
バッチ・ジョブ・コマンド・プロシージャの 中の	16-13
プロセスパーマネント	11-32

SYSS\$INSTRUCTION 論理名	11-24
SYSS\$MAINTENANCE 論理名	11-24
SYSS\$MANAGER 論理名	11-24
SYSS\$MESSAGE 論理名	11-24
SYSS\$OUTPUT 論理名	11-21
再定義	11-34
プロセスパーマネント	11-32
ログ・ファイル内の～への出力	16-15
SYSS\$PRINT 論理名	3-21
SYSS\$PROCMP 論理名	11-25
SYSS\$REM_ID 論理名	11-23
SYSS\$REM_NODE 論理名	11-23
SYSS\$SCRATCH 論理名	11-19, 11-23
SYSS\$SHARE 論理名	11-25
SYSS\$SPECIFIC 論理名	11-25
SYSS\$STARTUP 論理名	11-25
SYSS\$SYSDEVICE 論理名	11-24, 11-25
SYSS\$SYSROOT 論理名	11-24, 11-25
SYSS\$SYSTEM 論理名	11-19, 11-25
SYSS\$UPDATE 論理名	11-25
SYS.COM コマンド・プロシージャ	B-11
実行例	B-13
SYSS\$LDR 論理名	11-24
SYSS\$NET 論理名	11-21
SYSS\$LIBRARY 論理名	11-19, 11-24
SYSS\$LOADABLE_IMAGES 論理名	11-24
SYSS\$NODE 論理名	11-25
SYSS\$LOGIN_DEVICE 論理名	11-23
SYSS\$LOGIN 論理名	11-19, 11-23
SYSS\$TEST 論理名	11-25
System Generation Utility (SYSGEN)	5-10
SYSUAF.DAT ファイル	
JTQUOTA 値	11-30

T

Tab キー	2-20
TCP/IP (Transmission Control Protocol/Internet Protocol)	
名前およびアドレスの指定	3-8
ファイルの印刷	3-22
ファイルのコピー	3-16
ファイルへのアクセス	3-9
リモート・システムへの接続の消失	1-25
THEN コマンド	
コマンド・プロシージャ内での使用方 法	13-8
TYPE コマンド	
ファイルの表示	3-17
リモート・ノード上にあるファイルの表 示	3-18
ワイルドカード文字と	3-18

U

UAF(利用者登録ファイル)	1-4
LOCKPWD フラグ	1-7
アカウントの満了	1-18
最終ログインのレコード	10-11
ログイン・クラスの制限	1-12
UIC(利用者識別コード)	
省略時の保護	10-6
UIC 識別子	
例	10-3
UIC ディレクトリ指定	4-11
名前形式に変換	4-12
UNMARK コマンド	7-30

V

VOLPRO(Volume Protection Override)	6-6
------------------------------------	-----

W

WAIT コマンド	
コマンド・プロシージャの同期化	16-21
複数のプロシージャの同期化	16-22
WASTEBASKET フォルダ	
Mail の ~ を空にする	7-20
World-Wide PostScript Printing Subsystem (WWPPS)	3-24
WRITE FILE コマンド	8-49
EVE の	8-8
WRITE コマンド	
コマンド・プロシージャ内の	
論理名の使用	11-4
データの書き込み	14-9
文字列のレコードへの書き込み	15-15
WWPPS ユーティリティ	3-24
コマンド	3-26

ア

アカウント	
アクセスの監査	10-11
オープン	1-7
最初のログイン	1-5
種類	1-7
初期パスワード	1-4
制限	1-7
2 次パスワード	1-6
満了	1-17
満了後の更新	1-18
アカウント情報	
ログアウト中の取得	1-25
アクセス制御	
オブジェクト・セキュリティ・プロファイ	
ル	10-3
論理名テーブル	
共用可能	11-17

アクセス制御文字列	3-7, 10-7
上書き	11-8
形式	10-7
コマンド・プロシージャ内の	10-8
情報の保護	10-8
入力規則	3-10
ノード間でのファイルのコピー	3-16
ノード名での形式	3-10
アクセス制御リスト	
論理名テーブル	11-29
アクセス・タイプ	
セキュリティ監査と	10-12
アクセス・モード	11-6
定義	11-34
論理名	
表示	11-13
論理名テーブル	11-17
アクセス・リスト	10-5
アスタリスク (*)	
シンボル定義として	12-4
ワイルドカード文字としての	
UIC 形式によるディレクトリ指定で	
の	4-12
使用規則	3-11
ディレクトリ指定	3-11
値	
定義	2-4
アットマーク (@)	
Mail の中での配布リストとの使用	7-9, 7-10
コマンド・プロシージャ実行コマンド	13-18
アラーム	
イベントの起動	10-11
セキュリティのため使用可能にする	10-11
アンバサンド (&)	
シンボル定義での	12-32 ~ 12-33

イ

一重引用符 (')	
シンボル定義での	12-32
イメージ	
自動フォーリン・コマンドで起動	12-38
出力先の切り換え	14-10
シンボルを使用しないで起動	12-38
印刷	
横置き	3-23
印刷ジョブ	3-20 ~ 3-23
キュー情報	3-21
実行	3-21
遅延	3-23
で使用する DCL コマンドのリスト	3-23
複数部コピーする	3-23
優先順位	3-21
インターネット	
トランスポートの指定	7-6

ウ

上向き矢印キー	
コマンドの再呼び出し	2-14, 2-19
上書きモード	2-17

エ

エスケープ文字	13-43
エラー	
コマンド・プロシージャでの処理	13-26
コマンド・プロシージャ内での削除	13-37
ラベル	13-26
ログイン中の	1-3
エラー処理	
SET NOON コマンド	13-29
ON コマンド	13-27
エラー・チェックの使用不能化	13-29
演算子	
評価の順序	12-23

オ

大文字と小文字の区別	
DCL コマンド行の中の	2-2
EVE の REPLACE コマンドでの	8-31
SET FIND CASE EXACT コマンドの使 用	8-31
オブジェクト	
セキュリティ・プロファイル	10-3
セキュリティ・プロファイルの変更	10-4
セキュリティ・プロファイル表示	10-3
オブジェクト所有権	
変更	10-4
オペランド	
整数	12-15
文字列式内の	12-10

カ

会話型モード	
ログイン	1-9
各国語文字セット	
DEC 各国語文字セットを参照	
仮想ターミナル	
禁止	1-8
制限事項	16-10
切断されたプロセス	16-9
切断されたプロセスの管理	16-9
切断されたプロセスへの再接続	16-8
プロセスの切断	16-9
空の値	
ファイル・タイプの	3-13
ファイル名の	3-13
空文字	
DCL コマンド内の	2-7

環境識別子	
例	10-3
監査機能イベント	10-11
完全な名前	3-7
感嘆符 (!)	
配布リスト内の	7-9

キ

キー	
矢印	
コマンドの再呼び出し	2-20
キー (キーボード)	
DCL に割り込む	2-19
DCL コマンドの入力	2-18
DCL コマンドを再呼び出しする	2-19
Mail での定義	7-25
カーソル位置を制御する	2-20
画面表示を制御する	2-20
定義	2-18
矢印	
コマンドの再呼び出し	2-14
キー・シーケンス	2-18
キー定義	
Mail での	7-26
割り当て	2-18
キーパッド	
Mail での省略時の設定	7-25
Mail での ~ の定義	7-26
キー・フィールド	9-1
キー名	
Mail での	7-25
行編集	2-16
キーワード	
DCL コマンド行	2-5
勤務時間に関する制限	1-12

ク

クラスタ単位のシステム・テーブル	
論理名テーブルを参照	
繰り返し	
定義	13-5
クリーンアップ操作	13-16
クリーンアップ・タスク	
コマンド・プロシージャ内での	13-14

ケ

検索リスト	11-2
RUN コマンドによる	11-16
複数の	
検索順序	11-16
変換	11-13
論理名	11-13
ワイルドカード	11-14

コ

構文

DCL コマンドの	2-3
UIC 形式によるディレクトリ名の指定	4-11
サブディレクトリ	4-4
テープ・ボリューム上でのファイル指定	3-14
ノード指定	3-7
コマンド	
Mail 中での入力	7-2
自動的なフォーリン	12-38
短縮	2-8
取り消し	2-6
コマンド・インタプリタ	
実行されるコマンド	14-11
コマンド行	
Mail 中の	7-3
コマンド・プロシージャへの取り込み	13-3
再呼び出し	2-14
自動改行	2-17
編集	2-16
コマンド行の	
文字の削除	2-18
コマンド行の編集	2-16
上書モード	2-17
行編集を使用可能にする	2-17
挿入モード	2-17
文字の削除	2-18
コマンド・シーケンス	
PIE コマンドで作成	14-50
PIPE コマンドで作成	14-48, 14-51
コマンド修飾子	2-9
コマンド処理	
フェーズ	12-34
コマンド値	
時刻形式	2-11
日付形式	2-11
コマンドの再呼び出し	2-14
RECALL コマンドの使用	2-15
コマンド・プロシージャ	
BATCH.COM	B-27
CALC.COM	B-25
CLEANUP.COM	13-17
COMPILE_FILE.COM	B-32
CONVERT.COM	B-1
Ctrl/Y によるキャンセル	13-30
Ctrl/Y による割り込み	13-30
DCL コマンドの順序	13-7
DCL コマンドの取り込み	13-2
DIR.COM	B-8
EDITALL.COM	B-15
EXIT コマンド	13-11, 13-25
FORTUSER.COM	B-19
GETPARMS.COM	B-13
IF コマンド	13-8
LISTER.COM	B-23

コマンド・プロシージャ (続き)

LOGIN.COM	16-12, 16-14
MAILEDIT.COM	B-18
REMINDER.COM	B-5
STOP コマンド	13-12
SYLOGIN.COM	16-10
SYSSINPUT の定義	14-5
SYS.COM	B-11
THEN コマンド	13-8
アクセス制御文字列の使用	10-8
エラー処理	13-26
SET NOON コマンド	13-29
エラーの削除	13-37
完成	13-16
クリーンアップ・タスク	13-14, 13-16
コマンド行の取り込み	13-3
コメント	13-4
作成	13-2, 13-5
作成手順	13-5
実行	13-18
会話形式	13-21
会話形式の出力のリダイレクト	13-21
個人用ディスクでの	13-24
自動フォーリン・コマンドでの	12-38
シンボルを使用しない場合	12-38
テープ・ボリューム上での	13-24
バッチ・ジョブ	13-22
バッチ・ジョブの再起動	13-22
他のコマンド・プロシージャ内部から	
の	13-18
リモート・ノード上での	13-19
リモート・バッチ・ジョブ	13-22
実行中にチェック機能オン	13-14
終了	13-11, 13-24
出力	14-9
種類	13-1
条件の決定	13-6
条件のテスト	13-7
省略時のエラー処理	13-27
省略時の設定の復元	13-15
省略時の設定の保存	13-15
省略時のタイプ	13-2
設計	13-6
チェック設定値の変更	15-3
チェック設定値レキシカル関数を無効にする	15-3
重複ラベル	13-4
定義	13-1
テスト	13-12
デバッグ	13-12
SET PREFIX コマンド	13-13
SET VERIFY コマンド	13-13
SHOW SYMBOL コマンド	13-13
ドル記号 (\$)	13-3
入力	14-2
ネスティング	
パラメータによってデータを渡す	14-5
パーソナル・ログイン	13-41

コマンド・プロシージャ

パーソナル・ログイン (続き)	
専用アカウント内	13-41
バッチ・ジョブ内での指定	16-12, 16-13
パラメータを使用してデータを渡す	14-3
ファイルの削除	13-15
ファイルを閉じる	13-15
プログラム・スタブ	13-8
プログラム・スタブの置換	13-17
変数の決定	13-6
変数の使用	15-11
変数の割り当て	13-7
ユーザ入力のためのプロンプト表示	14-6
ラベル	13-3
ラベル・エラー処理	13-26
リテラル文字の取り込み	13-8
ログイン	13-40
コマンド・プロシージャの実行	13-18
会話形式	13-21
会話形式の出力のリダイレクト	13-21
個人用ディスクでの	13-24
自動フォーリン・コマンドでの	12-38
シンボルを使用しない場合	12-38
テープ・ボリューム上での	13-24
バッチ・ジョブ	13-22
バッチ・ジョブの再起動	13-22
他のコマンド・プロシージャ内部から	
の	13-18
リモート・ノード上での	13-19
リモート・バッチ・ジョブ	13-22
コマンド・レベル	
SET [NO]ON コマンドによる	13-30
定義	13-24
コメント	
感嘆符 (!) の使用	7-9
コマンド・プロシージャないでの	13-4
配布リスト内の	7-9
コントローラ指示子フィールド	
省略時の値	6-3
コントローラ・デスティネーション・フィールド	
定義	6-4

サ

最上位ディレクトリ	4-2
再初期化ボリューム	6-9
再呼び出しバッファ	10-8
削除	2-16
サーカンフレックス	13-43, 13-44
削除	
コマンド行の文字の	2-18
サブシェル	
PIPE コマンド	14-50
サブディレクトリ	
一覧	4-5
構文	4-4
作成	4-4
省略時の ~ を設定する	4-6

サブプロセス

親プロセスから受け継ぐ環境	16-7
親プロセスからの環境の除外	16-8
コンテキスト	16-7
作成	16-5
終了	16-6, 16-7
ジョブ階層構造	16-4
制御の引き継ぎ	16-8
生成	16-6
複数	16-5
定義	16-4
サブルーチン, 制御を渡す	14-38

シ

時間	
絶対日付と時刻の指定	2-12
デルタ日付と時刻の指定	2-13
式	
DCL 評価	12-24
値データ・タイプの変換	12-25
演算子の優先順位	12-23
とシンボル	12-8
論理	12-19, 12-20
識別子	
UIC	10-3
環境	10-3
汎用	10-3
プロセスの表示	10-2
実行	
複数のコマンド文字列の	14-48
時刻印字	
SET PREFIX コマンドの使用	16-15
チェック機能の設定	15-4
システム	
制御使用	1-6
システム・セキュリティ	
監査ログ・ファイル	10-12
システム特権	
ファイル保護	3-20
システム・メッセージ	
メッセージも参照	
システム例	
論理名	11-24
下向き矢印キー	
によるコマンドの再呼び出し	2-14 ~ 2-15, 2-19
自動的なフォーリン・コマンド	12-38
修飾子	
値の指定	2-11
コマンド	2-9
時刻形式	2-11
定位置	2-10
定義	2-4
パラメータ	2-10
日付形式	2-11
矛盾する	2-10

終了	
コマンド・プロシージャからの	13-24
出力	
コマンドとイメージからの切り換え	14-10
コマンド・プロシージャでの指定	14-9
出力先の切り替え	6-8
出力ストリーム	
リダイレクト	14-51
出力文字列	
FSFAO レキシカル関数による書式化	15-15
条件	
コマンド・プロシージャ内での決定	13-6
コマンド・プロシージャ内でのテスト	13-7
定義	13-5
条件コード	
EXIT コマンドを含む	13-38
重大度レベル	13-39
定義	13-37
表示	13-37
省略時の	
システムが提供する値	2-6
定義	1-20
ファイル保護	10-5
省略時の値	
DCL コマンドの	2-6
省略時の設定	
コマンド・プロシージャ内からの復元	13-15
コマンド・プロシージャ内への保存	13-15
初期化	
ディスク・ボリュームの	6-5
ボリューム	
ディスク・ボリュームの	6-6
初期化ファイル	7-26
ジョブ	1-11
ジョブ階層構造	
定義	16-4
ジョブ強制終了	
勤務時間制限が適用される	1-12
ジョブ・コントローラ	
勤務時間制限が適用される	1-12
署名ファイル	
メール・メッセージの追加	7-13
シンタックス	
OpenVMS Cluster デバイス指定	6-3
侵入の試み	1-13
回避	1-13
監査	10-11
シンボル	
DCL コマンドとして	12-4
FSTYPE レキシカル関数による確認	15-18
アスタリスク (*)	12-4
値の表示	12-6
アンパサンド (&)	12-32 ~ 12-33
一重引用符 (')	12-32
グローバル	12-3, 12-4
削除	12-6
算術演算と	12-16
式と	12-8

シンボル (続き)

数値オーバーレイ	12-18
数値式と	12-15
数値の比較	12-17
数値の表示	12-14
数値を示す	12-24
短縮	12-4
置換	12-6, 12-29, 12-30
アンパサンド (&) の使用	12-32 ~ 12-33
一重引用符 (') の使用	12-32
演算子	12-31
強制された	12-30
繰り返し	12-35
コマンド入力検索	12-34
コマンドの解析	12-34
式の評価	12-34
自動	12-29, 12-30
順序	12-30, 12-31
データ行での	12-34, 12-35
反復	12-35
フェーズ	12-34
定義	12-1, 12-3
シンボルとして	12-6
文字列	12-9
レキシカル関数として	12-21, 12-22
評価	12-24
フォーリン・コマンドとして	12-5
変数としての使用	12-8
未定義	12-37
文字のタイプ	12-8
文字列	
式	12-10
操作	12-10
文字列と	12-8
文字列の定義	12-9
文字列割り当て内の	12-7
連結	12-7
ローカル	12-3
論理データ	12-19
論理名との違い	12-2
割り当て文を使用して作成	12-3
シンボル置換	
文字列内での強制的な	14-10
シンボル・テーブル	12-26
グローバル	12-26, 12-27
\$RESTART シンボル	12-27
\$SEVERITY シンボル	12-27
\$STATUS シンボル	12-27
検索順序	12-27
ローカル	12-26
パラメータ	12-26
シンボル有効範囲	
グローバル	12-29
定義	12-28
ローカル	12-28

ス

数値

DCL による整数値の認識	12-14
内部記憶	12-15
比較	12-17
評価	12-24
文字列値への変換	12-26
レキシカル関数を使用した変換	15-17
数値オーバーレイ	12-18
数値式	12-15
整数オペランドと	12-15

セ

制御文字一覧	A-1
制限	
論理名テーブル	11-17, 11-29
制限値	
論理名テーブル	
ジョブ	11-30
セキュリティ	
監査ログ・ファイル	10-12
管理者	1-6
高水準	10-11
データ保護機構	10-3
セキュリティ・アラーム	10-11
セキュリティ監査	10-11
アカウントとファイル・アクセス	10-11
使用時の判断	10-13
ファイルへの ACE の追加	10-13
メッセージ	10-12
セキュリティ監査ログ・ファイル	10-12
セキュリティ機能	
セキュリティ・アラーム	10-11
セキュリティの制限	
勤務時間	1-12
時刻	1-12
ログイン・クラス	1-12
セキュリティの特徴	
アカウントの有効期間	1-17
アカウントの有効期限	1-18
勤務時間に関する制限	1-12
侵入の回避	1-13
ダイアルアップの再試行	1-13
パスワードの満了	1-17
ログイン・クラスの制限	1-12
セキュリティ・プロファイル	
オブジェクト	10-3
オブジェクト表示	10-3
オブジェクト変更	10-4
プロセス	
表示	10-3
プロセス表示	10-2
ユーザ	
表示	10-3
ユーザ表示	10-2

絶対時刻

キーワードの指定	2-12
構文	2-12
省略時の値	2-12
デルタ時間との組み合わせ	2-13
入力規則	2-12
専用アカウント	13-41

ソ

挿入モード

定義	2-17
----	------

タ

ダイアルアップ

ログインの失敗	1-13
---------	------

ダイアルアップ回線

アクセス制御	1-5
タイムアウト	1-2
代理アカウント	3-8, 10-8
許容最大数	10-8
省略時の設定	10-10
単独ユーザの場合	10-9
汎用アクセス	10-10
複数の ~ からの選択	10-10
複数のユーザの場合	10-10
命名	10-10

代理ログイン

キー属性	1-11, 10-8
セキュリティ上の利点	10-9
セキュリティ上の利点	10-8

タスク指定文字列

ネットワーク上での	3-10
-----------	------

タブ・ストップ

Ctrl/K を使用して進める	2-20
-----------------	------

ターミナル

アクセス制御	1-5
仮想	1-8, 16-9
画面消去	10-8
システムパスワード	
要件	1-6
システム・パスワードの要求	1-12
切断	16-9
ダイアルアップ・ログイン	1-10
データの書き込み	14-9
表示の終了と開始	2-20
ターミナル制御文字	
数値	A-1

チ

チェック機能

コマンド・プロシージャ実行中に作動	13-14
-------------------	-------

デ

定位置修飾子	
定義	2-10
ディスク	
マウント	6-7
ディスマウント	
ボリューム	6-8
割り当てられたデバイス	6-9
ディレクトリ	
アクセス	4-9
削除	4-6
定義	4-1
内容の表示	4-5
論理名テーブル	
システム	11-18
プロセス	11-18
ディレクトリ構造	
サンプル	4-2
ディレクトリ指定	
形式	4-4
定義	4-4
ディレクトリの中にあるファイルの一覧表	
示	4-5
ディレクトリの中にあるファイルの表示	4-5
ディレクトリ・ファイル	
最上位	4-2
作成	4-4
省略時の	4-2, 4-6
省略時の ~ を設定する	4-6
名前形式	4-4
保護	4-8
ディレクトリ・フィールド	
アスタリスク (*)・ワイルドカード文字の使	
用	3-11
定義	3-3
パーセント記号 (%)・ワイルドカード文字の使	
用	3-12
ディレクトリ名	
UIC 形式から名前形式への変換	4-12
置換	
ハイフン (-)・ワイルドカード文字によ	
る	4-10
反復記号 (...)・ワイルドカード文字	4-9
ファイル指定の中の ~ の名前形式	4-4
テキスト・エディタ	
ファイル表示のための	3-17
テスト	
コマンド・プロシージャ	13-12
データ	
WRITE コマンドを使用しての書き込み	14-9
コマンド・プロシージャ内への取り込	
み	14-2
ターミナルへの書き込み	14-9
論理	12-19, 12-20
データ・タイプ	
レキシカル関数の使用	15-17

データ・ファイル	
プログラムの登録	14-8
デバイス	
FSDEVICE レキシカル関数による情報の取	
得	15-9
ファイルのアクセス	6-8
プライベート	6-8
ボリューム・セットへのアクセス	6-2
割り当て	6-4
割り当てのディスマウント	6-9
デバイス・フィールド	
定義	3-3
デバイス名	4-8
OpenVMS Cluster	6-3
汎用	6-3
論理	6-3
デバッグ	
コマンド・プロシージャ	13-12
SET PREFIX コマンド	13-13
SET VERIFY コマンド	13-13
SHOW SYMBOL コマンド	13-13
テープ	
ANSI ボリューム・ファイル指定形式	3-14
テープ・ボリューム	
ファイル指定	3-14
デルタ時間	
構文	2-13
省略時の値	2-13
絶対時刻との組み合わせ	2-13
入力規則	2-13

ト

等価名	
DEFINE コマンド	11-3
最大長	11-5
変換属性	11-5
有効な文字	11-5
等価文字列	11-2, 11-6
特権	
VOLPRO(Volume Protection Override)	6-6
トランスポート	
Mail での指定	7-6
ドル記号 (\$)	
DCL プロンプトとしての	1-3, 2-2
OpenVMS Cluster デバイス指定の中の	6-3
コマンド・プロシージャ内での	13-3
ファイル名としての	3-3

ニ

2 次パスワード	
長さ	1-7
二重引用符 (" ")	
文字列での	14-10
入力	
コマンド・プロシージャからのプロンプト表	
示	14-6

入力 (続き)	
パッチ・ジョブ	16-13
入力ストリーム	14-7
リダイレクト	14-51
入力ファイル	
パラメータ・リストの中の～の一時的な省略時の	
値	4-8
入力ファイル・リストの中の一時的な省略時の	
値	4-8

ネ

ネットワーク	
メールの送信	7-6
リモート・システムへの接続の消失	1-25
ログアウト	1-25
ネットワーク・アクセス制御文字列	10-7, 10-8
ネットワーク・ノード	
リモート	3-8
ローカル	3-9
ネットワーク・ファイル指定	
ULTRIX での制限事項列	3-10
従来の形式	3-9
タスク指定文字列	3-10
フォーリン・ファイル形式	3-10

ノ

ノード	
ローカル	3-9
ノード名	
完全な	3-7
定義	3-3
入力規則	3-7
ファイル指定での形式	3-7
論理名の作成	11-7

ハ

パイプライン	14-48
配布リスト	
DCL レベルからのメールの送信	7-10
Mail 内での使用	7-9
エディタを使用した作成	7-8
ハイフン (-)	
ディレクトリ名の中の	4-10
バージョン番号, Extended File Specifications で	
の	5-3
パスワード	
新しい	1-16
新しい設定	1-14
誤り	1-9
安全性	1-4
1 次	1-6, 1-7
受け付け	1-5
オープン・アカウント	1-7
危険性	1-4
再試行	1-13

パスワード (続き)

最初	1-4
最小長	1-5
再利用	1-4, 1-5
システム	1-5
入力	1-6
自動生成	1-14
初期	1-4
推測	1-4
制限事項	1-5
生成	1-14, 1-15
生成された	1-16
選択	1-4
選択のガイドライン	1-4

ダイアルアップ中に入力できる回

数	1-13
代理ログインでの省略	10-8
長さ	1-4, 1-5
2 次	1-6, 1-17
入力	1-7
二重	1-6
変更	1-14
/NEW_PASSWORD 修飾子の使用	1-16
回数に関するガイドライン	1-19
2 次	1-16
頻度についてのガイドライン	1-5
満了	1-17
ログイン時	1-5, 1-16
変更の確認	1-14
変更の失敗	1-18
変更理由	10-11
保護	1-18
満了	1-17
短かすぎる場合	1-14
ユーザ	1-5
ロック	1-7
パスワードの保護	1-18
ダイアルアップの再試行	1-13
パーセント記号 (%)	
ワイルドカード文字としての	3-12
パッチ・ジョブ	
SUBMIT コマンド	
修飾子	16-18
エントリ変更	16-17
開始時間の指定	16-11
許可	1-11
勤務時間制限が適用される	1-12
再開	16-20, 16-21
削除	16-20
終了	16-20
出力	16-14
状態の表示	16-19
使用方法	16-10
属性の変更	16-17
定義	9-15
登録	16-10, 16-11
コマンド・プロシージャ	13-22, 16-10

パッチ・ジョブ (続き)

入力	16-13
複数のコマンド・プロシージャの指定	16-12, 16-13
複数のプロシージャの同期化	16-21, 16-22
ログ・ファイル	16-14
パッチ処理	
プロセスパーマネント論理名	11-32
8 進数	
UIC 形式によるディレクトリ指定の中 の	4-12
バッファ	
DCL コマンドの再呼び出し	2-14
パラメータ	
コマンド・プロシージャヘデータを渡 す	14-3
指定	
シンボルとして	14-4
整数として	14-3
ヌル値として	14-4
文字列として	14-3
定義	2-4
ネスティングしたコマンド・プロシージャヘデー タを渡す	14-5
パッチ・ジョブにデータを渡す	14-5
パラメータ修飾子	2-10
パラメータ・リスト	
複数のファイル指定	4-8
複数のファイル指定の省略時の値	4-8
反復記号 (...)	
ディレクトリ名の中のワイルドカード文字として の	4-9, 4-10
汎用識別子	
例	10-3
汎用デバイス名	
定義	6-3

ヒ

左向き矢印キー

による DCL コマンド行内のカーソルの移 動	2-20
日付	
絶対日付と時刻の指定	2-12
デルタ日付と時刻の指定	2-13
ビデオ・ターミナル	1-3

フ

ファイル

FSSEARCH レキシカル関数を使用しての検 索	15-10
MAIL.MAI	7-16
/VERSION_LIMIT 修飾子の使用	3-6
アクセス	
プライベート・デバイス上の	6-8
ボリューム・セット上の	6-2
アクセスの監査	10-11, 10-13

ファイル (続き)

アラームの適用	10-13
印刷	3-20
書き換え	3-17
極秘ファイルの保護	10-13
コピー	3-15
ノード間での	3-15, 3-16
コマンド・プロシージャから閉じる	13-15
コマンド・プロシージャからの削除	13-15
削除	3-18
作成	3-14
COPY コマンドの使用	3-15
CREATE コマンドの使用	3-14
メール・メッセージからの	7-12
初期化	7-26
セキュリティ監査使用時の判断	10-13
セキュリティ監査のために ACE を追加す る	10-13
代理アカウントに必要な保護	10-10
定義	3-2
内容の表示	
ワイルドカードによる	3-18
ページ	3-19
バージョン番号	3-6
バージョン番号の制御	3-6
表示	
内容	3-17, 3-18
表示	
内容	3-18
プロセスパーマネント	11-33
変更	3-14
保護	3-20, 7-21
メール	
DCL レベルから	7-11
Mail ユーティリティの使用	7-10
メール・メッセージの追加	7-13
リモート・アカウントからのコピー	10-10
連結	3-15
ワイルドカード文字の指定	3-11
ファイル・アクセス	
制御文字列の使用	3-10
リモート・ノードからの	3-9
ファイル指定	3-3
アスタリスク (*)・ワイルドカード文字の使 用	3-11
インクルードされたファイルのリスト	3-3
空の値を持つ	3-13
磁気テープの代替形式	3-14
入力規則	3-3
ネットワーク	3-9
ノード名	3-7, 3-16
パーセント記号 (%)・ワイルドカード文字の使 用	3-12
有効な文字	3-3
リストの指定	4-8
ファイル・タイプ	3-3
省略時の値のリスト	3-4, 3-5
入力規則	3-3

ファイル・タイプ・フィールド	
アスタリスク (*)・ワイルドカード文字の使	
用	3-11
空の値を持つ	3-13
定義	3-3
パーセント記号 (%)・ワイルドカード文字の使	
用	3-12
ファイル入出力	
切り換え	11-33
ファイルの	
バージョン番号	3-3
ファイルの閲覧者	10-13
ファイルのコピー	3-15
ノード間での	3-15, 3-16
ファイルの送信	
Mail の	7-10
ファイルの保護	
FSENVIRONMENT レキシカル関数による省略時	
の設定の変更	15-5
ファイル・バージョン番号	
アスタリスク (*)・ワイルドカード文字の使	
用	3-11
ファイル保護	3-20
極秘ファイル	10-13
代理アカウントに必要な	10-10
フィールド	9-3
Sort/Merge ユーティリティ参照	
フォーリン・コマンド	12-5
自動的な	12-38
文字制限	12-3, 12-5
フォーリン・ファイル指定	
ネットワーク上での	3-10
複合時刻	
構文	2-13
入力規則	2-14
複数のファイル指定	
パラメータ・リストの中の	4-8
プリント・キュー	
制御	3-23
プログラム	
データ・ファイルへの取り込み	14-8
プログラム・スタブ	
コマンドの置換	13-17
コマンド・プロシージャ内での	13-8
プロシージャ実行 (@) コマンド	13-18
プロセス	
Ctrl/T で状態をチェックする	1-21
作成	16-1
ジョブ階層構造	16-4
属性のセーブ	15-3
接続	16-8
接続中	1-8
切断	1-8, 16-9
ログアウト	16-9
切断された	16-8
定義	16-1
独立した	16-4
プロセスのライト識別子の表示	10-2

プロセス (続き)	
レキシカル関数を使用しての属性の変	
更	15-2
プロセス・コンテキスト	
属性のリスト	16-1, 16-2, 16-3
プロセス属性	
一般的な変更	13-16
プロンプト	2-5
DCL	1-3

へ

ヘルプ	
コマンドのための	1-23
システム・メッセージについての	1-24
ヘルプ・メッセージ・ユーティリティ (MSGHLP)	
起動	1-24
変数	
INQUIRE コマンドを使用しての割り当	
て	13-7
コマンド・プロシージャ内での決定	13-6
コマンド・プロシージャ内での割り当	
て	13-7
定義	13-5

ホ

保護	
オブジェクト	10-3
省略時	10-5
ディレクトリ	4-8
ファイル	3-20
ファイルの	
Mail の	7-21
保護コード	10-4
アクセス・タイプ	10-5
カテゴリ	10-4
ボリューム	
オペレータの手助け	6-7
情報の表示	6-2
初期化	6-5
ディスクマウント	6-8, 6-9
割り当てられたデバイス	6-9
マウント	6-6
フォーリン	6-7
ボリューム・セット	
定義	6-2
マウント	6-6
ボリュームのマウント	
とセキュリティ監査	10-12
フォーリン	6-7

マ

マウント要求 6-7

ミ

右向き矢印キー
によるカーソルの移動 2-20

メ

メッセージ
Mail 中でのコピー 7-17
アナウンスメント 1-8
監査 10-12
コマンドへのシステム応答 1-20
コマンド・ライン・エラーの表示 1-20
システム・エラー 1-20
システム表示内での表記 1-21
情報 1-20
無表示 1-9
ログイン失敗 1-9
ログイン障害 10-11
ログイン中の 1-8
メッセージ・カウント
Mail での訂正 7-19
メッセージの読み込み
Mail の 7-27

モ

文字
印刷されない 12-8
英数字 12-8
特殊 12-8
文字セット
DEC 各国語文字セットを参照 A-1
文字列
強制的シンボル置換 14-10
式
オペランド 12-10
シンボル 12-10
シンボルと 12-8
整数値への変換 12-25, 12-26
操作 12-10
置換 12-12, 12-13
定義 12-9
二重引用符 (" ") の中の 14-10
比較 12-11
比較, 演算子の使用 14-34
評価 12-24
リテラル・テキストとして 14-10
レキシカル関数による取り出し 15-13
レキシカル関数を使用した変換 15-17
レキシカル関数を使用しての調査 15-13
レキシカル関数を持つ 15-12

文字列割り当て
シンボルを含む 12-7
モード
アクセス・モードを参照

ヤ

矢印キー
コマンドの再呼び出しによる 2-14 ~ 2-15
によるコマンドの再呼び出し 2-19

ユ

ユーザ
プロセスのライト識別子の表示 10-2
ユニット番号フィールド 6-4
省略時の値 6-3

ラ

ラベル
コマンド・プロシージャ 2-4
コマンド・プロシージャ内での 13-3
コマンド・プロシージャ内での重複 13-4
ローカル・シンボル・テーブル内での 13-4

リ

リストの検索
省略時設定
変更 11-30
リストを検索
省略時設定
変更 11-11
リテラル・テキスト
文字列 14-10
リテラル文字
コマンドプロシージャ内での取り込み 13-8
リモート・セッション
強制終了 1-25
リモート・ノード
FSCONTEXT による情報の取得 15-8
アクセス制御文字列 3-8
ログイン
リモート・システムへ 1-10

ル

ループ
作成方法 13-10
定義 13-10

レ

レキシカル関数

FSDIRECTORY	12-23
構文	12-21
コマンド・プロシージャ内での使用	12-21
出力文字列の書式化	15-15
情報の取得	15-2
OpenVMS Cluster ノードからの	15-8
キュー	15-6
システム	15-5, 15-6
プロセス	15-7
使用方法	12-22
シンボルの確認	15-18
定義	15-1
データ・タイプの処理	15-17
データ・タイプの変換	15-17
データの評価	15-18
デバイスの検索	15-9
ファイルとデバイスについての情報の取得	15-9
ファイルの検索	15-10
プロセス属性のセーブ	15-3
プロセス属性の変更	15-2
文字列の一部の取り出し	15-13
文字列の処理	15-12
文字列の調査	15-13
論理名の変換	15-11

レコード

Sort/Merge ユーティリティ参照	
ソート	9-12
フィールド	9-4
ターミナルからのレコードの入力	9-18

レコード単位取り扱いデバイス

ファイル指定用のアウトプットとして使用	6-8
---------------------	-----

レコードのソート

9-3, 9-26

連結, ファイル

3-15

ロ

ローカル・シンボル・テーブル

ラベル	13-4
-----	------

ログアウト

オペレーティング・システムからの	1-24
セキュリティ留意事項	1-26
切断されたプロセスからの	16-9
リモート・セッションより	1-25

ログイン

会話型	1-9
禁止	
侵入回避のため	1-13
最終～の監視	10-11
システムの操作	
エラー	1-3
制御	1-6
制限時間	1-12

ログイン (続き)

ダイアルアップ	1-10
パスワードを入力できる回数	1-13
代理	1-11
ネットワーク	1-10
パスワードの変更	1-16
バッチ	1-11
非会話型	1-9, 1-10
満了アカウント	1-18
リモート	1-10
ローカル	1-10

ログイン・クラス

会話型	1-10
制限	1-12
ダイアルアップ	1-10
ネットワーク	1-10
バッチ	1-11
非会話型	1-10
リモート	1-10
ローカル	1-10

ログイン・コマンド・プロシージャ

専用アカウント内	13-41
定義	13-40
パーソナル	13-41
バッチ・ジョブとして実行される	16-14
バッチ・ジョブ登録時の実行	16-12

ログイン障害

メッセージ	10-11
-------	-------

ログインの失敗

再試行	1-13
その原因	1-11 ~ 1-13
メッセージ	1-9
理由	1-3

ログイン・メッセージ

アナウンスメント	1-8
パスワードの期限満了	1-3
無表示	1-9

ログ・ファイル

コマンド出力の指定	16-16
セーブ	16-15, 16-16
内容	16-14
バッチ・ジョブ	16-14
バッチ・ジョブ実行時のチェック	16-16

論理式

12-19, 12-20

論理デバイス名

6-3

論理名

CONCEALED 属性	11-6
DCLSPATH	12-40
TERMINAL 属性	11-6
アクセス・モード	11-6
表示	11-13
検索リスト	
RUN コマンド	11-16
検索リストの作成	11-13
ワイルドカード	11-14
コマンド・プロシージャでの	11-2
コマンド・プロシージャ内で使用	11-4
最大長	11-5

論理名 (続き)

削除	11-9, 11-31
作成	11-3
ノード	11-7
作成規則	11-5
システム単位	11-21, 11-24
システムワイド	11-2
シンボルとの違い	12-2
1つのオブジェクトに複数名を定義	11-9
定義	11-1
ノード用の	
ファイル指定に使用	11-8
表示	11-11
ファイル出力	11-4
ファイル入力	11-4
プロセスパーマナント	11-32
表示	
コマンド・プロシージャで	11-33
再定義	11-32, 11-33, 11-36
SYSS\$OUTPUT	11-34
ファイル指定として	11-33
変換	11-9
会話型処理	11-32
検索リスト	11-13
コマンド・プロシージャに	11-32, 11-33
システムの省略時設定	11-10, 11-11
属性	11-5
バッチ・プロシージャに	11-32
反復	11-10
変更	11-30
リストの検索	11-30
メールボックス	11-22
文字列と同値	11-2
有効な文字	11-5
リストの検索	11-30
論理名ディレクトリへの追加	11-31
論理名 MAIL\$SYSTEM_FLAGS	7-14
論理名テーブル	
アクセス制御リスト	11-29
アクセス必要条件	11-28
共用可能	11-17, 11-19, 11-28
クラスタ単位	11-19, 11-22
作成	11-28
クラスタ単位の親	11-22
グループ	11-19, 11-22
検索順序	
変更	11-30
削除	11-31
作成	11-27
システム	11-19, 11-24
システム・ディレクトリ・テーブルの内	
容	11-21
省略時設定	11-19
省略時設定の検索順序	11-30
ジョブ	11-19, 11-23
制限値の指定	11-30
制限値の指定	11-29
ディレクトリ	11-18

論理名テーブル (続き)

特徴	11-17
特権必要条件	11-28
表示	11-12, 11-18
プロセス	11-17, 11-19, 11-20, 11-21, 11-27
プロセス・ディレクトリ	11-20
保護	11-29
保護の定義	11-25
論理名ディレクトリへの追加	11-30, 11-31

ワ

ワイルドカード文字

DCL コマンド行	2-5
アスタリスク (*)	3-11
ハイフン (-)	4-10
パーセント記号 (%)	3-12
反復記号 (...)	4-9, 4-10
ファイル名での使用	3-11

割り当て

クラス・フィールド	6-3
-----------	-----

割り当て文

シンボルの作成	12-3
---------	------

OpenVMS ユーザーズ・マニュアル

2002 年 10 月 発行

コンパックコンピュータ株式会社

〒140-8641 東京都品川区東品川 2-2-24 天王洲セントラルタワー

電話 (03)5463-6600 (大代表)

AA-PZWLG-TE

