日本語 HP OpenVMS

DEC XTPU
リファレンス・マニュアル

AA-PY6UE-TE

2005 年 4 月

本書は， DEC XTPU (DEC eXtended Text Processing Utility) 言語についての解説書です。

改訂／更新情報: 日本語 OpenVMS V7.3 『DEC XTPU リファレンス・マニュアル』の改訂版です。

オペレーティング・システム: 日本語 OpenVMS I64 V8.2
日本語 OpenVMS Alpha V8.2
日本語 OpenVMS VAX V7.3

ソフトウェア・バージョン: DEC XTPU V3.8

日本ヒューレット・パッカード株式会社

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

目次

図

表

まえがき

本書の目的

本書は，DEC XTPU (DEC eXtended Text Processing Utility) 言語について説明します。英語版の DECTPU と DEC XTPU の相違点についてのみ記述されていますので，本書に記述されていない機能の解説は，『DEC Text Processing Utility Reference Manual』を参照してください。

本書は主に参照資料として使用するように構成されています。

対象読者

本書は DEC XTPU で拡張された機能に関する解説書であり，読者が一般的な DECTPU の機能を理解していることを前提としています。一般的な DECTPU の機能については『DEC Text Processing Utility Reference Manual』 を参照してください。 DEC XTPU の一部の高度な機能，たとえば呼び出し可能なインタフェースや FILE_PARSE 組込みプロシージャなどは， OpenVMS システムの概念を十分理解した経験の豊富なシステム・プログラマを対象としています。 OpenVMS オペレーティング・システムに関するマニュアルについては，この後の "関連資料 "の節を参照してください。

本書の構成

本書は， 6 つの章と 2 つの付録から構成されています。

第 1 章	DEC XTPU の概要を示します。
第 2 章	DEC XTPU のデータ・タイプのうち，文字セットに関連するものについて詳しく説明します。
第 3 章	DEC XTPU の字句要素について説明します。この中には文字セット，識別子，変数，定数，および DEC XTPU 言語ステートメントなどの予約語が含まれています。
第 4 章	DEC XTPU 組込みプロシージャのうち，機能拡張のために追加，または変更された組込みプロシージャについて説明します。
第 5 章	DEC XTPU を起動する方法を示します。
第 6 章	DEC XTPU の呼び出し可能なインタフェースについて説明します。
付録 A	DEC XTPU でサポートされる端末装置について説明します。

付録 B　DEC XTPU で英語版に追加されたメッセージとその省略形，および重大度レベルのリストを示します。

関連資料

- 『Guide to the DEC Text Processing Utility』

  英語版 DECTPU の概説書です。

- 『DEC Text Processing Utility Reference Manual』

  本書に記述されていない，英語版の DECTPU と同一の機能について記述しています。

- 『OpenVMS Utility Routines Manual』

  DECTPU の呼び出し可能なインタフェースについて記述しています。

- 『OpenVMS System Messages and Recovery Procedures Reference Manual』

  DECTPU のメッセージ一覧とその説明，およびユーザの処置が記述されています。

- 『OpenVMS DCL ディクショナリ』
- 『OpenVMS System Services Reference Manual』
- 『OpenVMS Programming Interfaces: Calling a System Routine』
- 『OpenVMS Calling Standard』
- 『OpenVMS RTL Library (LIB$) Manual』
- 『OpenVMS Record Management Services Reference Manual』
- 『日本語 EVE ユーザーズ・ガイド』
  『日本語 EVE リファレンス・マニュアル』

  簡単に学習でき，かつ効率よく使用できるように設計されたエディタである日本語 EVE について記述しています。

- 『日本語ライブラリ 利用者の手引き』
- 『日本語ユーティリティ 利用者の手引き』
- 『フォント管理ユーティリティ 利用者の手引き』
- 『漢字コード表』

表記法

DEC XTPU プログラムでは，字下げやスペーシングなどの特殊なフォーマッティングは必要とされません。本書のプログラミングの例では，DEC XTPU プログラムを作成するためのいろいろな方法を示すために，複数の異なるフォーマッティング・スタイルが使用されています。プロシージャに含まれる長いステートメントは複数行に分割することにより，読みやすくなっています。本書で使用されているフォーマット

に必ずしも従う必要はなく，自分の使いやすいフォーマットを使用することができます。

本書では以下の表記法を使用します。

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。 • 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 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。
italic text	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*number*)，コマンド・ライン (たとえば/PRODUCER=*name*)，コマンド・パラメータ (たとえば*device-name*) などの変数を示す場合にも使用されます。
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。

表記法	意味
–	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，要求に対する引数が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。

1

DEC XTPU の概要

この章では，DEC XTPU(DEC eXtended Text Processing Utility) に関して，以下の項目について解説します。

- DEC XTPU とは何か，第 1.1 節
- 日本語 EVE とは何か，第 1.2 節
- DEC XTPU 言語とは何か，第 1.3 節
- DEC XTPU はどんなハードウェアをサポートするのか，第 1.4 節
- DEC XTPU を使用する方法，第 1.5 節

1.1 DEC XTPU とは何か

DEC XTPU とは英語版の DEC Text Processing Utility (DECTPU) を拡張し，複数のコードセットを処理する機能を追加した，プログラミング可能なテキスト処理ユーティリティです。DEC XTPU は，テキスト処理インタフェースを開発するアプリケーション・プログラマ，およびシステム・プログラマを援助するためのツールとして設計されています。たとえば，プログラマは DEC XTPU を用いて，特別な環境のエディタを作ることができます。この DEC XTPU ユーティリティにはコンパイラ，インタプリタ，高級手続き言語，および DEC XTPU で書かれたインタフェースが含まれています。

DEC XTPU はスクリーン向きエディタのための一般的な機能の他に，以下の特殊機能も備えています。

- 複数のバッファ
- 複数のウィンドウ
- DEC XTPU の内部および DCL レベルでの複数のサブプロセス
- バッチ・モードでのテキスト処理
- テキストの挿入モードまたは置換モード
- フリー・カーソル移動またはバウンド・カーソル移動
- 学習シーケンス
- パターン照合
- キー定義
- 手続き言語

- 呼び出し可能なインタフェース

DEC XTPU のレイヤ上に構築されたエディタや他のアプリケーション・プログラムは，ユーザと DEC XTPU 間のインタフェースになります。ユーザは日本語 EVE を使うか，または独自のアプリケーション・プログラムを作ることによって， DEC XTPU を利用できます。

すなわち， DEC XTPU は日本語のテキスト処理アプリケーションを構築するための基礎となるものであると考えることができます。日本語 EVE という編集インターフェイスは， DEC XTPU の上にレイヤ構造として作成されたインタフェースの例であり， DEC XTPU 言語で書かれています。

1.2 日本語 EVE とは何か

日本語 EVE は，簡単に学習し，使用することができる日本語エディタです。一般的な編集機能は日本語 EVE キーパッドの 1 つのキーを押すことにより利用できるので，これまでエディタを使ったことがない人でも，基本的な編集操作なら，すみやかに実行することができます。

また，テキスト・エディタの使用経験の豊富なユーザにとっても，日本語 EVE は強力で効率のよいエディタです。高度な編集機能は，日本語 EVE コマンド・ラインにコマンドを入力することにより実行できます。このコマンドを使用すれば， DEC XTPU の多くの特殊な機能を使用することができます。日本語 EVE インタフェースについての詳しい説明は，『日本語 EVE ユーザーズ・ガイド』および『日本語 EVE リファレンス・マニュアル』を参照してください。

1.3 DEC XTPU 言語

DEC XTPU は，強力なテキスト処理作業を行う高級手続き型プログラミング言語です。 DEC XTPU 言語は DEC XTPU のもっとも基本的な構成要素であると考えることができます。 DEC XTPU の機能をアクセスするには， DEC XTPU 言語でプログラムを作成し， DEC XTPU ユーティリティを使ってそのプログラムをコンパイルし，実行しなければなりません。 1 つの DEC XTPU ステートメントだけしか含まない単純なプログラムから，日本語 EVE インタフェースのセクション・ファイルのように複雑なプログラムまで，いろいろなプログラムを DEC XTPU 言語で作成することができます。

DEC XTPU 言語はブロック構造になっているため，簡単に学習でき，また使用できます。 DEC XTPU 言語には多くのデータ・タイプ，関係演算子，エラー処理機能，繰り返しや CASE 文，重要な機能を実行するための広範囲にわたる組込みプロシージャなどがあります。コメントはコメント文字 (!) によって示すことができるので，プロシージャに関する内部的な記述を，プロシージャに含めることができます。また，

ユーザ作成デバッグ・プログラムを用いてプロシージャをデバッグすることもできます。

1.3.1 DEC XTPU のデータ・タイプ

DEC XTPU 言語には多くのデータ・タイプが含まれています。データ・タイプは変数の内容の意味を解釈するために使用されます。他の多くの言語と異なって，DEC XTPU 言語にはデータ・タイプを変数に割り当てるための宣言ステートメントはありません。DEC XTPU の変数のデータ・タイプは，代入ステートメントで代入されたときのデータ・タイプであると解釈されます。たとえば，次のステートメントは，文字列データ・タイプを this_var という変数に代入します。

```
this_var := 'This can be a string of your choice';
```

次のステートメントは，ウィンドウ・データ・タイプを変数 lower_case に代入します。ウィンドウはスクリーンの 1 行目から始まる 15 行を使用し，ステータス・ラインは OFF(表示されない) です。

```
x := CREATE_WINDOW(1, 15, OFF)
```

多くの DEC XTPU データ・タイプ (たとえば学習タイプやパターン) は，通常のプログラミング言語で使用されているデータ・タイプと異なっています。DEC XTPU のデータ・タイプは表 1–1 に示すとおりです。

表 1–1 DEC XTPU データ・タイプ

データ・タイプ	説明
ARRAY	要素の集まり
BUFFER	テキスト・レコードの集まり－ BUFFER は編集作業ができる範囲を示します。
INTEGER	整数－有効な値は，-2,147,483,648 から 2,147,483,647 です。
KEYWORD	DEC XTPU コンパイラにとって特別の意味がある予約語
LEARN	DEC XTPU キーストロークの集まり
MARKER	バッファ内の文字位置
PATTERN	一連の文字－パターン演算子とパターン組込みプロシージャは結果としてこのデータ・タイプを通知します。PATTERN はバッファ内の特定のテキストを見つけるために，SEARCH 組込みプロシージャで使用されます。
PROCESS	VMS サブプロセス
PROGRAM	実行可能な DEC XTPU ステートメントのコンパイルされた形式
RANGE	2 つのマーカの間に存在するすべてのテキスト (2 つのマーカも含む)
STRING	文字列
UNSPECIFIED	変数宣言を持つコードがコンパイルされた後のグローバル変数の初期状態

(次ページに続く)

表 1–1 (続き) DEC XTPU データ・タイプ

データ・タイプ	説明
WINDOW	スクリーンを分割した領域－ウィンドウは，テキスト・バッファのうちスクリーン上に現れている領域です。

DEC XTPU データ・タイプについての詳しい説明は，『Guide to the DEC Text Processing Utility』および本書の第 2 章 "DEC XTPU のデータ・タイプ "を参照してください。

1.3.2 DEC XTPU 言語の宣言文

DEC XTPU 言語の宣言文は表 1–2 に示すとおりです。

表 1–2 DEC XTPU 言語の宣言文

モジュール宣言	MODULE － IDENT － ENDMODULE
プロシージャ宣言	PROCEDURE － ENDPROCEDURE
定数宣言	CONSTANT
グローバル変数宣言	VARIABLE
ローカル変数宣言	LOCAL

DEC XTPU 言語の宣言文についての詳しい説明は，『Guide to the DEC Text Processing Utility』および本書の第 3 章を参照してください。

1.3.3 DEC XTPU 言語ステートメント

DEC XTPU 言語ステートメントは表 1–3 に示すとおりです。

表 1–3 DEC XTPU 言語ステートメント

代入ステートメント	:=
繰返しステートメント	LOOP － EXITIF － ENDLOOP
条件ステートメント	IF － THEN － ELSE － ENDIF
ケース・ステートメント	CASE － ENDCASE
エラー・ステートメント	ON_ERROR － ENDON_ERROR

DEC XTPU 言語ステートメントについての詳しい説明は，『Guide to the DEC Text Processing Utility』および本書の第 3 章を参照してください。

1.3.4 DEC XTPU 組込みプロシージャ

DEC XTPU 言語には多くの組込みプロシージャが含まれており，スクリーン管理やキー定義，テキスト操作，およびプログラム実行などの機能を提供します。

独自のプロシージャを作成するときには，組込みプロシージャを使うことができます。また日本語 EVE から組込みプロシージャを使うこともできます。 DEC XTPU 組込みプロシージャについての詳しい説明は，『DEC Text Processing Utility Reference Manual』および本書の第 4 章 を参照してください。

1.3.5 ユーザ作成プロシージャ

DEC XTPU 組込みプロシージャの呼び出しと DEC XTPU 言語ステートメントを用いて，独自のプロシージャを作成することができます。 DEC XTPU のプロシージャは，値を返すことができます。また再帰的なプロシージャを作成することもできます。プロシージャを作成し，コンパイルしておけば，プロシージャ名を使ってそのプロシージャを呼び出すことができます。

プロシージャを作成するときには，以下のガイドラインに従ってください。

- プロシージャは PROCEDURE ではじめてください。その後にはプロシージャ名がきます。
- プロシージャは ENDPROCEDURE で終了してください。
- 各ステートメントやプロシージャ呼び出しの後に，ステートメントや呼び出しが続くときには，セミコロン (;) を置いてください。

例 1–1 は，現在の文字位置を現在のバッファの先頭に移動するプロシージャの例です。 DEC XTPU 言語ステートメント (PROCEDURE-ENDPROCEDURE) と組込みプロシージャ (POSITION, BEGINNING_OF, CURRENT_BUFFER, MESSAGE, GET_INFO) が使用されています。

例 1–1 ユーザ作成プロシージャの例

```
! This procedure moves the editing
! position to the top of the buffer

PROCEDURE user_top

  POSITION (BEGINNING_OF(CURRENT_BUFFER));
  MESSAGE ("現在のバッファは " + GET_INFO (CURRENT_BUFFER, "name") + " です");
ENDPROCEDURE
```

このプロシージャをコンパイルしておけば， user_top という名前を使ってこのプロシージャを呼び出すことができます。

1.4 DEC XTPU がサポートするハードウェア

DEC XTPU は，日本語 OpenVMS がサポートするすべてのハードウェアで実行することができます。

DEC XTPU は日本語端末 VT280 シリーズ (DEC 漢字 1983 年版) および VT382 でスクリーン編集機能をサポートします。日本語端末 VT280 シリーズ (DEC 漢字 1978 年版) は， DEC 漢字 1983 年版で変更になった文字が表示されない，という制限のもとでサポートされます。

DEC XTPU のサポートする日本語端末に関する制限事項については，付録 A を参照してください。

1.5 DEC XTPU の起動

DCL レベルで DEC XTPU を起動するには， EDIT/XTPU コマンドを入力し，ファイルの名前を指定します。

```
$ EDIT/XTPU text_file.lis
```

このコマンドは *text_file.lis* というファイルを編集のためにオープンします。追加ファイルは，編集セッションで DEC XTPU の内部から READ_FILE 組込みプロシージャを使って指定することができます。

上記のコマンドで DEC XTPU を起動する場合には，省略時の編集インタフェースが使用されます (/NOSECTION を指定しなかった場合)。 DEC XTPU の省略時の編集インタフェースは日本語 EVE です。使用されるインタフェース名 (ここでは日本語 EVE (JEVE)) をコマンドとして， DEC XTPU を起動することができるように，次に示すようなシンボルを定義することをお勧めします。

```
$ JEVE  :== EDIT/XTPU
```

1.5.1 DEC XTPU コマンドの修飾子

DEC XTPU コマンドには修飾子を指定することができます。 DEC XTPU 修飾子は中断されたセッションからの回復や， DEC XTPU をアクセスするために使用されるインターフェイスを与える初期化ファイルなどのアイテムを制御します。 DEC XTPU に対する修飾子は表 1–4 に示すとおりです。

表 1–4 DEC XTPU コマンドの修飾子

修飾子	省略時の値
/CODESET=*codeset_keyword*	/CODESET=ISO_LATIN1
/[NO]COMMAND[=*filespec*]	/COMMAND=XTPU$COMMAND
/[NO]CREATE	/CREATE
/[NO]DEBUG[=*filespec*]	/NODEBUG
/[NO]DISPLAY[=*keyword*]	/DISPLAY=CHARACTER_CELL
/[NO]INITIALIZATION[=*filespec*]	/NOINITIALIZATION
/INTERFACE[=*Keyword*]	/INTERFACE=CHARACTER_CELL
/[NO]JOURNAL[=*filespec*]	/[NO]JOURNAL
/[NO]KANJI_DICTIONARY[=*filespec*]	/KANJI_DICTIONARY
/[NO]MODIFY	/MODIFY
/[NO]OUTPUT[=*filespec*]	/OUTPUT=*input_file.type*
/[NO]READ_ONLY	/NOREAD_ONLY
/[NO]RECOVER	/NORECOVER
/[NO]SECTION[=*filespec*]	/SECTION=XTPU$SECTION
/START_POSITION=(*line,column*)	/START_POSITION=(1,1)
/[NO]WORK=[*filespec*]	/NOWORK
/[NO]WRITE	/WRITE

DEC XTPU コマンドの修飾子についての詳しい説明は，第 5 章を参照してください。

1.5.2 初期化ファイル

初期化ファイルには，**コマンド・ファイル**，**セクション・ファイル**そして**イニシャライゼーション・ファイル**の 3 つのファイルがあります。コマンド・ファイルとセクション・ファイルを用いると，日本語 EVE エディタあるいはアプリケーションをカスタマイズできます。イニシャライゼーション・ファイルは，日本語 EVE やその他のアプリケーションを，それら特有のコマンド，設定，およびキー定義を使ってカスタマイズするのに使用されます。

コマンド・ファイル

ファイル・タイプ　.TPU
DEC XTPU 修飾子　/COMMAND=*filespec*
省略時のファイル　現在のディレクトリの XTPU$COMMAND.TPU

コマンド・ファイルには， DEC XTPU 言語で書かれたソース・コードが入っています。

省略時設定は，/COMMAND=XTPU$COMMAND です。論理名 XTPU$COMMAND に任意のファイルを定義して使うこともできます。/NOCOMMAND 修飾子が指定されない限り， DEC XTPU はコマンド・ファイルを読み込もうとします。

/COMMAND 修飾子についての詳しい説明は『Guide to the DEC Text Processing Utility』および本書の第 5.5.2 項を参照してください。

セクション・ファイル

ファイル・タイプ	.XTPU$SECTION
DEC XTPU 修飾子	/SECTION=*filespec*
省略時のファイル	SYS$LIBRARY ディレクトリの JEVE$SECTION_V3.XTPU$SECTION

セクション・ファイルは DEC XTPU ソース・コードをコンパイルした形式のバイナリ・ファイルです。

省略時設定は，/SECTION=XTPU$SECTION です。日本語 OpenVMS では，起動時に XTPU$SECTION という論理名が JEVE$SECTION_V3 に定義されます。このため， DCL レベルで EDIT/XTPU コマンドを入力すると，日本語 EVE エディタが起動されます。日本語 EVE を使用しないときには，別のセクション・ファイルを指定するか，/NOSECTION 修飾子を指定しなければなりません。

注意

/NOSECTION 修飾子を指定して DEC XTPU を起動した場合には，DEC XTPU に対するインタフェースを与えるためのバイナリ・ファイルの読み取りは実行されません。この場合には，Return や<X キーさえも定義されていない状態になります。新しいセクション・ファイルを作成しているときに，既存のセクション・ファイルからプロシージャや変数，あるいは定義を含みたくない場合には，/NOSECTION 修飾子を使用します。

/SECTION 修飾子についての詳しい説明は『Guide to the DEC Text Processing Utility』および本書の第 5.5.14 項を参照してください。

イニシャライゼーション・ファイル

ファイル・タイプ	.EVE
DEC XTPU 修飾子	/INITIALIZATION=*filespec*
省略時のファイル	現在のディレクトリまたは SYS$LOGIN ディレクトリの JEVE$INIT_V3.EVE

イニシャライゼーション・ファイルには， DEC XTPU を使用して書かれたアプリケーションのためのコマンドが入っています。たとえば，日本語 EVE のイニシャライゼーション・ファイルには，キー定義やマージンの設定を含めることができます。

省略時の設定は，/INITIALIZATION です。論理名 JEVE$INIT_V3 に任意のファイルを定義して使うこともできます。イニシャライゼーション・ファイルはとても簡単に作成できますが，セクション・ファイルやコマンド・ファイルを使用したときよりも，起動に時間がかかります。/NOINITIALIZATION 修飾子が指定されない限り，DEC XTPU はイニシャラゼーション・ファイルを読み込もうとします。

/INITIALIZATION 修飾子についての詳しい説明は『Guide to the DEC Text Processing Utility』 および本書の第 5.5.6 項を，また，イニシャライゼーション・ファイルの作成については日本語 EVE ユーザーズ・ガイド の "イニシャライゼーション・ファイルの作成 "を参照してください。

初期化ファイルの使用法

コマンド・ファイルとセクション・ファイルのどちらか一方，または両方を使用することにより，既存のインタフェースを変更または拡張することができます。

- コマンド・ファイルは一般に，インタフェースの一部だけを変更するときに使用します。
- セクション・ファイルを使用すれば始動時間が短くなるので，インタフェースを大きく変更する場合や，変更内容が複雑な場合には，セクション・ファイルを使用します。また既存のインタフェースの上にレイヤ構造として構成されていないインタフェースを作成するときにも，セクション・ファイルを使用します。
- イニシャライゼーション・ファイルは，アプリケーションがサポートしているときのみ使用できます。

日本語 EVE のソース・ファイルは XTPU$EXAMPLES にあります。これらのリストを見るには，以下のコマンドを入力してください。

```
$ DIRECTORY XTPU$EXAMPLES:JEVE$*.TPU
```

システムにこのファイルが含まれていないときには，システム管理者に連絡してください。

2

DEC XTPU のデータ・タイプ

データ・タイプとは，同じ方法で作成され，同じ方法で処理される要素グループです。変数のデータ・タイプは，その変数に対してどのような操作を実行できるかを決定します。 DEC XTPU のデータ・タイプは以下に示すとおりです。 DEC XTPU で拡張されていて，この章で解説されているものにはアスタリスク (*) がついています。その他のデータ・タイプは，英語版 DECTPU とまったく同じに扱われます。

ARRAY	アレイ (*)
BUFFER	バッファ
LEARN	学習
INTEGER	整数
KEYWORD	キーワード
MARKER	マーカ (*)
PATTERN	パターン (*)
PROGRAM	プログラム
PROCESS	プロセス
RANGE	レンジ (*)
STRING	文字列 (*)
UNSPECIFIED	不定
WINDOW	ウィンドウ

以降の各節では DEC XTPU のデータ・タイプのうち，英語版 DECTPU と異なった意味を持つものについて説明します。それ以外のデータ・タイプについては『Guide to the DEC Text Processing Utility』を参照してください。

2.1 アレイ

アレイ・データ・タイプは，データをグループ化して扱うためのデータ・タイプです。アレイ・データ・タイプの要素には，任意のデータ・タイプを入れることができます。アレイ・データ・タイプの要素となるデータ・タイプが文字列データ・タイプのときには，アレイ・データ・タイプに日本語データを入れることができます。

例

```
mark1 := MARK (none);
mix_array {mark1} := "日本語文字列";
```

またアレイ・データ・タイプは，その INDEX に文字列を使うことができます。このとき，その INDEX の中には日本語文字を含めることができます。

例

```
mix_array {"英語"} := "hello";
mix_array {"日本語"} := "こんにちは";
```

2.2 マーカ

マーカ・データ・タイプはバッファ内の文字に関連する参照点です。マーカ・データ・タイプは MARK 組込みプロシージャを使って作成されます。マーカは文字に対してセットされるため，複数カラム文字の 2 カラム目以降に編集点があるときでもマーカはその文字，すなわち 1 カラム目に対してセットされます。ただし，フリー・マーカは複数カラム文字の 2 カラム目以降にセットすることができます。フリー・マーカを作成するには， MARK (FREE_CURSOR) を使用してください。

2.3 パターン

パターンは，バッファ内の特定の文字列を探索するために使用される構造です。パターンは SEARCH および SEARCH_QUIETLY 組込みプロシージャの第 1 パラメータとして使用されます。パターンを作成するには， DEC XTPU パターン演算子 (+, &, | , @) を用いて以下の要素を結合します。

- 文字列定数
- 文字列変数
- パターン変数
- パターン組込みプロシージャの呼び出し
- 以下のキーワード

  ANCHOR
  LINE_BEGIN
  LINE_END
  PAGE_BREAK
  REMAIN
  UNANCHOR

- カッコ (式を囲むために使用します)

上記の要素のうち，文字列定数，文字列変数，パターン組込みプロシージャの文字列パラメータには DEC XTPU で使用できるすべての文字を含むことができます。

パターン・データ・タイプを通知する組込みプロシージャは表 2–1 に示すとおりです。

表 2–1 パターン・データ・タイプを通知する組込みプロシージャ

プロシージャ	説明
ANY	パラメータとして使用した文字列のどの文字 (1 文字) とも一致する
ARB	指定した長さの一連の文字と一致する
MATCH	現在の文字位置から始まり，MATCH に対するパラメータとして指定した文字列まで続く (その文字列も含む) 一連の文字と一致する
NOTANY	パラメータとして指定した文字列に含まれていないどの 1 文字とも一致する
SCAN	パラメータとして指定した文字列に含まれているどの文字も含まない最長文字列と一致する
SCANL	SCAN と同じであるが，SCANL はレコード境界をこえることができる
SPAN	パラメータとして指定した文字列の文字だけを含む最長文字列と一致する
SPANL	SPAN と同じであるが，SPANL はレコード境界をこえることができる

これらのプロシージャの文字列パラメータには，2 バイト文字を含むことができます。

2.4 レンジ

レンジ・データ・タイプは，指定した 2 つのマーカの間のすべてのテキスト (2 つのマーカも含む) を表現します。レンジは CREATE_RANGE 組込みプロシージャを使用して作成することができます。レンジ・データ・タイプは文字単位でつけられるので，複数カラム文字の一部分だけを含んだレンジを作成することはできません。

2.5 文字列

DEC XTPU は文字データを表現するために文字列データ・タイプを使用します。文字列データ・タイプの値には，ASCII 文字セット，DEC 補助文字セット，ISO Latin1 補助文字セット，JIS ローマ字セット，JIS カタカナ・セットおよび DEC 漢字セットで定義されるすべての文字を含むことができます。文字定数を指定するには，値を引用符で囲まなければなりません。DEC XTPU では，文字列の区切り文字として ASCII 文字の二重引用符 (") または単一引用符 (') を使用することができますが，DEC 漢字セットの二重引用符 (“ , ”) や単一引用符 (’) を文字列の区切り文字として使用することはできません。

3

DEC XTPU 言語のレキシカル要素

この章では，次の DEC XTPU レキシカル要素のうち，日本語化されたものについて説明します (アスタリスク (＊) で示されています)。それ以外のレキシカル要素については，『Guide to the DEC Text Processing Utility』を参照してください。

- 文字セット (＊)
- 識別子 (＊)
- 変数
- 定数
- 演算子
- 式
- 予約語

3.1 概要

DEC XTPU プログラムはレキシカル要素から構成されます。レキシカル要素は算術演算子などの個々の文字や，識別子などの文字グループです。レキシカル要素の基本単位は ASCII， DEC MCS， ISO Latin1， JIS Roman， JIS カタカナ， DEC 漢字セットの文字です。 DEC 漢字セットの詳細は，『漢字コード表』を参照してください。

3.2 文字セット

DEC XTPU は ASCII 文字セット， DEC 補助文字セット， ISO Latin1 補助文字セット， JIS ローマ字セット， JIS カタカナ・セットおよび DEC 漢字セットに含まれる文字を扱うことができます。

ASCII 文字セットは， 128 文字の 7 ビット文字セットです。各文字は，等価な 10 進数で表わされます。 ASCII 文字セットは，次のように分類されます。

0 ～ 31	プリントされない文字 (タブ，ライン・フィード，キャリッジ・リターン，ベルなど)
32	スペース
33 ～ 64	特殊文字 (アンパサンド "&"，疑問符 "?"，等号 "="，および "0"から "9"までの数字)

65 ～ 122	大文字の "A"から "Z"，および小文字の "a"から "z"
123 ～ 126	特殊文字 (左中カッコ "{"，オーバーライン "~"など)
127	Delete

漢字コードとして， 1983 年版 DEC 漢字文字セット (JIS X 0208-1983 に準拠) と拡張漢字文字セットが用いられます (本書では，これらを総称して DEC 漢字セットと呼ぶ場合があります)。漢字コードの詳しい体系については，『漢字コード表』を参照してください。

注意

128 ～ 159 は，拡張制御文字として扱われます。

DEC XTPU コンパイラは引用符で囲まれた文字列を除いて， ASCII 文字の大文字と小文字は区別しません。たとえば，以下のように表記した "EDITOR"という単語は，すべて同じ意味になります。

```
EDITOR
EDitOR
editor
```

しかし，次のように引用符で囲まれた 2 つの文字列は，異なる値を表わします。

```
'xyz'
'XYZ'
```

DEC XTPU コンパイラは ASCII 文字および ISO Latin1 補助文字のみを解釈しますので， DEC 漢字セットに含まれる文字などは，引用符で囲まれた文字列の内部を除いて使用できません。たとえば以下の記述は認められます。

```
'文字'
```

3.2.1 制御文字の入力

DEC XTPU では， 2 通りの方法で制御文字を入力することができます。

1. 入力したい制御文字に対応する 10 進数を指定した ASCII 組込みプロシージャを使用します。たとえば， COPY_TEXT (ASCII (27)) を実行すると，エスケープ文字が現在のバッファに入力されます。
2. 日本語 EVE で準備されている特殊機能を使用して制御文字を入力します。

   日本語 EVE の場合，Ctrl/V に対して定義されている quote コマンドを使用して，バッファに制御文字を挿入することができます。Ctrl/V を押し，次に入力したいキーを押します。たとえば，ESC キー (エスケープ・キー) を押すと，エスケープ文

字が現在のバッファに入力されます。別の方法として，以下のようなコード入力方法があります。

a. Doキーを押す

b. CODE *n* と入力する
(*n* は，たとえばエスケープ文字のコード番号である 27 など)

c. Returnキーを押す

あるいは

a. GOLDキーを押す

b. Xキーを押す

c. コード番号 (たとえば，エスケープ文字を示す 27) を入力する

d. Returnキーを押す

3.2.2 DEC XTPU **のシンボル**

DEC XTPU では，一部のシンボルは特殊な意味を持っています。これらのシンボルはステートメント区切り文字 (;)，演算子 (+) やその他の構文要素として使用されます。 DEC XTPU シンボルは英語版の DECTPU とまったく同じです。『Guide to the DEC Text Processing Utility』を参照してください。複数の文字から構成されるシンボルの場合，各文字をスペースで区切ることはできません。また，シンボルの記述はすべて ASCII 文字に限られ， ASCII 文字セットに属さない文字はシンボルとして解釈されません。

3.3 識別子

DEC XTPU では，識別子はプログラム，プロシージャ，キーワード，および変数の名前を指定するために使用されます。識別子には ASCII 文字セットに含まれる，アルファベット，数字，ドル記号，およびアンダースコアを含むことができますが，以下の制約条件に従わなければなりません。

- 識別子にスペースやシンボル (ドル記号とアンダースコアを除く) を含むことはできない
- 識別子に ASCII 文字セットに含まれない文字を含むことはできない
- 識別子の長さは最大 132 文字である

DEC XTPU の組込みプロシージャ，キーワード，およびグローバル変数の識別子は予約語です。予約語は本書では大文字で示されています。

プログラム，プロシージャ，および変数に名前をつけるために，ユーザ定義識別子を作成することができます。

4

DEC XTPU 組込みプロシージャ

この章では DEC XTPU 組込みプロシージャについて，2 つの節に分けて説明しています。

第 4.1 節 ― 組込みプロシージャが以下の機能に応じて分類されています。

- スクリーン・レイアウト
- カーソルの移動
- 編集位置の移動
- テキスト操作
- パターン照合
- 編集コンテキストのステータス
- キーの定義
- マルチ処理
- プログラムおよびプロシージャの実行
- DECwindows
- その他

第 4.2 節 ― 組込みプロシージャのうち日本語特有な機能変更や，DEC XTPU で追加のあったものについて個別に説明されています。また，DEC XTPU では制限付きで使用できるプロシージャも記述されています。

4.1 機能別に分類した組込みプロシージャ

以下のリストは，編集作業を実行するときに，どの作業に対してどの組込みプロシージャを使用すればよいかを判断するのに役立ちます。組込みプロシージャについての詳しい説明は，第 4.2 節の各プロシージャに関する説明を参照してください。英語版の DECTPU と機能的に変更のないプロシージャは，第 4.2 節には記述されていませんので『Guide to the DEC Text Processing Utility』を参照してください。第 4.2 節に記述されているプロシージャは，アスタリスク (*) で示されています。

4.1.1 スクリーン・レイアウト

ADJUST_WINDOW
CREATE_WINDOW
MAP
REFRESH
SET (DISPLAY_VALUE)
SET (HEIGHT)
SET (PAD)
SET (PROMPT_AREA)
SET (SCREEN_UPDATE)
SET (SCROLLING)
SET (STATUS_LINE)
SET (TEXT)
SET (VIDEO)
SET (WIDTH)
SHIFT
UNMAP
UPDATE

4.1.2 カーソルの移動

ALIGN_CURSOR (*)
CURSOR_HORIZONTAL
CURSOR_VERTICAL
SCROLL
SET (COLUMN_MOVE_VERTICAL)
SET (CROSS_WINDOW_BOUNDS)
SET (DETACHED_ACTION)
SET (MOVE_VERTICAL_CONTEXT)

4.1.3 編集位置の移動

MOVE_HORIZONTAL
MOVE_VERTICAL
POSITION

4.1.4 テキスト操作

APPEND_LINE
BEGINNING_OF
CHANGE_CASE (*)
CHANGE_CODE (*)
COPY_TEXT
CREATE_BUFFER
CREATE_RANGE
EDIT
END_OF
ERASE
ERASE_CHARACTER
ERASE_LINE
FILE_PARSE
FILE_SEARCH
FILL (*)
MARK (*)
MESSAGE_TEXT
MODIFY_RANGE
MOVE_TEXT
PCS_CLASS (*)
READ_FILE
SEARCH
SEARCH_QUIETLY
SELECT (*)
SELECT_RANGE (*)
SET (ERASE_UNMODIFIABLE)
SET (MODIFIABLE)
SET (MODIFIED)
SPLIT_LINE (*)
TRANSLATE
VERIFY_BUFFER (*)
WRITE_FILE

4.1.5 パターン照合

ANCHOR
ANY
ARB
LINE_BEGIN
LINE_END

MATCH
NOTANY
PAGE_BREAK
REMAIN
SCAN
SCANL
SPAN
SPANL
UNANCHOR

4.1.6 編集コンテキストのステータス

CURRENT_BUFFER
CURRENT_CHARACTER
CURRENT_COLUMN
CURRENT_DIRECTION
CURRENT_LINE
CURRENT_OFFSET
CURRENT_ROW
CURRENT_WINDOW
DEBUG_LINE
ERROR
ERROR_LINE
ERROR_TEXT
GET_INFO (*)
LOCATE_MOUSE
RECOVER_BUFFER
SET (AUTO_REPEAT)
SET (BELL)
SET (CODESET) (*)
SET (DEBUG)
SET (DEFAULT_DIRECTORY)
SET (FACILITY_NAME)
SET (FILL_NOT_BEGIN) (*)
SET (FILL_NOT_END) (*)
SET (FILL_TRIM_SPACE) (*)
SET (FORWARD)
SET (INFORMATIONAL)
SET (INSERT)
SET (JOURNALING)
SET (KEYBOARD_CODESET) (*)
SET (KEYSTROKE_RECOVERY)
SET (LEFT_MARGIN)

SET (LEFT_MARGIN_ACTION)
SET (LINE_NUMBER)
SET (MARGINS)
SET (MARGIN_ALLOWANCE) (*)
SET (MAX_LINES)
SET (MESSAGE_ACTION_LEVEL)
SET (MESSAGE_ACTION_TYPE)
SET (MESSAGE_CODESET) (*)
SET (MESSAGE_FLAGS)
SET (MOUSE)
SET (NO_WRITE)
SET (OUTPUT_CODESET) (*)
SET (OUTPUT_FILE)
SET (OVERSTRIKE)
SET (PAD_OVERSTRUCK_TABS)
SET (PERMANENT)
SET (RECORD_ATTRIBUTE)
SET (REVERSE)
SET (RIGHT_MARGIN)
SET (RIGHT_MARGIN_ACTION)
SET (SPECIAL_ERROR_SYMBOL)
SET (SUCCESS)
SET (SYSTEM)
SET (TAB_STOPS)
SET (TIMER)
SET (TRACEBACK)
SET (VIDEO_CHARACTER_SET) (*)
SHOW

4.1.7 キーの定義

ADD_KEY_MAP
CREATE_KEY_MAP
CREATE_KEY_MAP_LIST
DEFINE_KEY
KEY_NAME (*)
LAST_KEY
LOOKUP_KEY
REMOVE_KEY_MAP
SET (KEY_MAP_LIST)
SET (POST_KEY_PROCEDURE)
SET (PRE_KEY_PROCEDURE)
SET (SELF_INSERT)

SET (SHIFT_KEY)
SET (UNDEFINED_KEY)
UNDEFINE_KEY

4.1.8 マルチ処理

ATTACH
CREATE_PROCESS
SEND
SEND_EOF
SPAWN

4.1.9 プログラムおよびプロシージャの実行

ABORT
BREAK
COMPILE
EXECUTE
RETURN
SAVE

4.1.10 DECwindows

CREATE_WIDGET
DEFINE_WIDGET_CLASS
GET_CLIPBOARD
GET_DEFAULT
GET_GLOBAL_SELECT
LOWER_WIDGET
MANAGE_WIDGET
RAISE_WIDGET
READ_CLIPBOARD
READ_GLOBAL_SELECT
REALISE_WIDGET
SEND_CLIENT_MESSAGE
SET (ACTIVE_AREA)
SET (CLIENT_MESSAGE)
SET (DEFAULT_FILE)
SET (DRM_HIERARCHY)
SET (ENABLE_RESIZE)
SET (FIRST_INPUT_ACTION)

SET (GLOBAL_SELECT)
SET (GLOBAL_SELECT_GRAB)
SET (GLOBAL_SELECT_READ)
SET (GLOBAL_SELECT_TIME)
SET (GLOBAL_SELECT_UNGRAB)
SET (ICON_NAME)
SET (ICON_PIXMAP)
SET (ICONIFY_PIXMAP)
SET (INPUT_FOCUS)
SET (INPUT_FOCUS_GRAB)
SET (INPUT_FOCUS_UNGRAB)
SET (MAPPED_WHEN_MANAGED)
SET (MENU_POSITION)
SET (RESIZE_ACTION)
SET (SCREEN_LIMITS)
SET (SCROLL_BAR)
SET (SCROLL_BAR_AUTO_THUMB)
SET (UID)
SET (WIDGET)
SET (WIDGET_CALL_DATA)
SET (WIDGET_CALLBACK)
SET (WIDGET_CONTEXT_HELP)
SET (WIDGET_RESOURCE_TYPES)
UNMANAGE_WIDGET
WRITE_CLIPBOARD
WRITE_GLOBAL_SELECT

4.1.11 その他

ASCII (*)
CALL_USER (*)
CODE (*)
COLUMN_LENGTH (*)
CONVERT
CONVERT_KANA (*)
CREATE_ARRAY
DEC_KANJI (*)
DELETE
DELETE_TANGO (*)
ENTER_TANGO (*)
EXIT
EXPAND_NAME
FAO (*)

HELP_TEXT
INDEX
INT
JOURNAL_CLOSE
JOURNAL_OPEN
LEARN_ABORT
LEARN_BEGIN
LEARN_END
LENGTH
MESSAGE
QUIT
READ_CHAR (*)
READ_KEY (*)
READ_LINE (*)
SET (EOB_TEXT)
SLEEP
STR
SUBSTR
SYMBOL (*)

4.2 組込みプロシージャの説明

各組込みプロシージャの説明には下記の項目が含まれています。

- 簡単な機能定義
- 形式
- 説明
- 警告およびエラーを通知するシグナル・エラー
- 例

各組込みプロシージャはアルファベット順に説明されています。

注意

パラメータで使われる文字列のほとんどのものには ASCII 以外の文字を使用できますが，ファイル名，プロセス名などを指定するパラメータ文字列には，ASCII 文字のみが使用できます。

ALIGN_CURSOR

この組込みプロシージャは，カーソルが文字境界上に存在しないときに行の先頭方向へカーソルを移動して，文字境界にカーソルを合わせます。

形式

[integer :=] ALIGN_CURSOR

引数

なし

戻り値

ALIGN_CURSOR によって移動したカラム数。

シグナル・エラー

XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

ASCII

この組込みプロシージャは，整数を ASCII 文字 (半角文字) 列に変換するか，あるいは ASCII 文字を整数に変換します。

形式

{integer2 | string2} := ASCII *({integer1 | keyword | string1})*

引数

integer1
ISO Latin1 文字セットに含まれる文字を示す 10 進数。

keyword
キーワードはキー名でなければなりません。キー名が印字可能文字を生成するキーの名前のときは， ASCII はその文字を返します。そうでなければ， ASCII の値が 0 である文字を返します。

string1
ASCII 値を得たい文字。文字列の長さが 1 文字よりも長いときには，最初の文字の ASCII 値が得られます。

戻り値

指定された ASCII 値を持つ文字 (integer または keyword パラメータを指定したとき)。または，指定された文字の ASCII 値 (string パラメータを指定したとき)。

説明

ASCII 組込みプロシージャの基本的な説明は，『DEC Text Processing Utility Reference Manual』を参照してください。

ASCII 組込みプロシージャは ISO Latin1 以外の文字を扱いません。このため以下のような制限事項があります。

- パラメータ integer1 の値は 0 から 255 までのみが有効です。それ以外の値を指定したときには， ASCII の値が 0 である文字が戻されます。

- string パラメータとして ISO Latin1 文字セットに属さない文字を指定したときは，整数値 0 が戻されます。

ISO Latin1 文字セットに属さない文字に関して，文字と数字の相互変換を行いたいときには， DEC_KANJI 組込みプロシージャまたは CODE 組込みプロシージャを使用してください。

シグナル・エラー

XTPU$_ARGMISMATCH	ERROR	パラメータのデータ・タイプが正しくない
XTPU$_NEEDTOASSIGN	ERROR	ASCII は代入文の右辺でのみ使用できる
XTPU$_NULLSTRING	WARNING	長さ 0 の文字列が渡された
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

例

```
1. my_character := ASCII (182)
```

この代入ステートメントは my_character という変数に ISO Latin1 文字の"¶ "を代入します。

```
2. character := ASCII ("B")
```

この代入ステートメントは， "B"という文字の ASCII 値 (66) を character という変数に代入します。

CALL_USER

この組込みプロシージャは，DEC XTPU の内部から他の言語で書かれたプログラムを呼び出します。CALL_USER のパラメータはそのまま外部ルーチンに渡されます。DEC XTPU はパラメータに対して何の処理も行いません。パラメータ integer は参照によって渡され，string1 はディスクリプタによって渡されます。string2 は外部プログラムから返される値です。

形式

string2 := CALL_USER *(integer, string1)*

引数

integer
ユーザ作成プログラムに参照によって渡される整数値。

string1
ユーザ作成プログラムにディスクリプタによって渡される文字列。

戻り値

呼び出されたプログラムから返される値。

説明

CALL_USER に返される値 string2 に加えて，外部プログラムは正しく実行されたかどうかを示すステータス・コードを返します。このステータス・コードは，ON_ERROR 文によって処理することができます。偶数のステータス・コードが返されると ON_ERROR 文が実行されます。ERROR 文はプログラムからのステータス値をキーワードとして戻します。

CALL_USER は次のように使用します。

1. 任意の言語でプログラムを書きます。そのプログラムは XTPU$CALLUSER という名前のグローバル・ルーチンでなければなりません。
2. プログラムをコンパイルします。

3. オプション・ファイルを使ってプログラムをリンクし，共有イメージを作ります。
4. 論理名 XTPU$CALLUSER に作ったルーチンが入ったファイルを定義します。
5. DEC XTPU を起動します。
6. 組込みプロシージャ CALL_USER に必要なパラメータを指定して， DEC XTPU セッションから外部プログラムを実行します。プログラムが正しくリンクされて論理名 XTPU$CALLUSER が定義されていると，組込みプロシージャ CALL_USER はパラメータを外部ルーチンに渡します。

CALL_USER のパラメータは呼び出したプログラムへの入力パラメータになります。DEC XTPU は，パラメータに対して何の処理も行わずに外部プロシージャに渡します。呼び出されるルーチンが必要としていない場合でも，パラメータは必ず 2 つ渡さなければなりません。値を渡す必要のないときには，以下のような NULL パラメータを渡してください。

```
CALL_USER (0, "")
```

シグナル・エラー

XTPU$_CALLUSERFAIL	WARNING	CALL_USER ルーチンの実行が失敗した
XTPU$_ARGMISMATCH	ERROR	パラメータのデータ・タイプが正しくない
XTPU$_BADUSERDESC	ERROR	ユーザ・ルーチンがリターン・ディスクリプタに正しくない値を入れた
XTPU$_INVPARAM	ERROR	パラメータの型が間違っている
XTPU$_NEEDTOASSIGN	ERROR	CALL_USER は代入文の右辺でのみ使用できる
XTPU$_NOCALLUSER	ERROR	実行するルーチンが光からない
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

例

```
ret_value := CALL_USER (6, "ABC")
```

この文はユーザが書いたプログラムを呼び出します。 DEC XTPU を起動する前に論理名 XTPU$CALLUSER で CALL_USER によって呼び出したいプログラムが入っているファイルを指定します。 DEC XTPU は第 1 パラメータ (6) を参照によって，第 2 パラメータ ("ABC") をディスクリプタによって渡します。もし数字と文字列を入力値として使っているならば，プログラムは 6 と "ABC"を処理しま

す。プログラムが結果を返すように作られていれば，結果は ret_value に返されます。

以下の例は，組込みプロシージャ CALL_USER の使い方を具体的に示したものです。浮動小数点演算を行うために呼び出されるルーチンは， BASIC で書かれています。

1. 渡された値の浮動小数点演算を行う BASIC のプログラムを書きます。

```
! Filename:FLOATARITH.BAS
1     sub XTPU$CALLUSER ( some_integer% , input_string$ , return_string$)
10    ! don't check some_integer% because this function only does
      ! floating point arithmetric
20    ! parse the input string
      ! find and extract the operation
      comma_location = pos ( input_string$, ", ", 1% )
      if comma_location = 0 then go to all_done end if
      operation$ = seg$( input_string$, 1%, comma_location - 1% )
      ! find and extract the 1st operand
      operand1_location = pos ( input_string$, ", ", comma_location +1)
      if operand1_location = 0 then go to all_done end if
      operand1$ = seg$( input_string$, comma_location + 1%, &
                          operand1_location -1 )
      ! find and extract the 2nd operand
      operand2_location = pos ( input_string$, ", ", operand1_location + 1)
      if operand2_location = 0 then
              operand2_location = len( input_string$) + 1
      end if
      operand2$ = seg$( input_string$, operand1_location + 1%, &
                        operand2_location -1 )
      select operation$ ! do the operation
      case "+"
              result$ = sum$( operand1$, operand2$ ) !
      case "-"
              result$ = dif$( operand1$, operand2$ ) !
      case "*"
              result$ = num1$( Val( operand1$ ) * Val( operand2$ ))
      case "/"
              result$ = num1$( Val( operand1$ ) / Val( operand2$ ))
      case else
              result$ = "unkown operation."
      end select
      return_string$ = result$
999   all_done: end sub
```

2. プログラムをコンパイルします。

```
$ BASIC/LIST floatarigh
```

3. BASIC のプログラムをリンクするときに使うオプション・ファイルを作ります。

```
!+
!   File: FLOATARITH.OPT
!
!   Options file to link floatarith BASIC program with VAXTPU
!-
FLOATARITH.OBJ
UNIVERSAL=XTPU$CALLUSER
```

4. プログラムをリンクし，共有イメージを作ります。

```
$ LINK floatarith/SHARE/OPT/MAP/FULL
```

5. 論理名 XTPU$CALLUSER に BASIC プログラムの実行イメージを定義します。

```
$ DEFINE XTPU$CALLUSER device:[directory]floatarith.EXE
```

6. DEC XTPU を起動します。

7. 以下の DEC XTPU プロシージャを書いてコンパイルします。

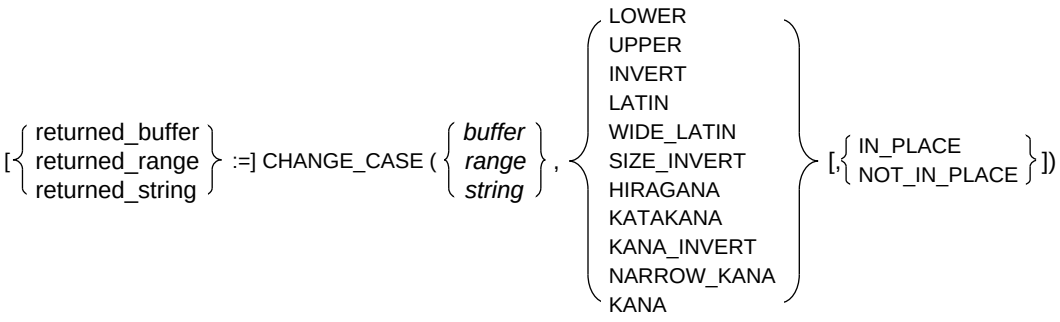
```
PROCEDURE my_call_user

! test the built-in procedure call_user

  LOCAL output,
        input;

  input := READ_LINE ("Call user >");  ! パラメータを入力
  output := CALL_USER (0, input);      ! プログラムの呼び出し
  MESSAGE (output);
ENDPROCEDURE;
```

8. プロシージャ my_call_user を呼び出すと， BASIC ルーチンに渡すパラメータを聞いてきます。パラメータは，オペレータ，値，値の順番で渡されます。例えばプロンプトに， +, 3.33, 4.44 と入力すると結果の 7.77 がメッセージ領域に表示されます。

CHANGE_CASE

この組込みプロシージャは，指定されたキーワードに従って，指定されたバッファ，レンジ，または文字列中のテキストの，大文字/小文字変換，ひらがな/カタカナ/半角カナ変換，全角 Latin 文字/半角 Latin 文字変換，および ASCII からひらがなへの変換を実行します。 CHANGE_CASE は変換された文字列を含むバッファ，レンジ，または文字列を戻すこともできます。

形式



引数

buffer
変更したい文字列を含むバッファ。第 1 パラメータにバッファを指定したときには，キーワード NOT_IN_PLACE は使えません。

range
変更したい文字列を含むレンジ。第 1 パラメータにレンジを指定したときには，キーワード NOT_IN_PLACE は使えません。

string
変換したい文字列。第 3 パラメータに IN_PLACE を指定したときには，第 1 パラメータに指定された文字列を変更します。文字列定数に対しては何も行いません。

LOWER
指定した文字列のアルファベット (全角，半角) は小文字に変換されます。

UPPER
指定した文字列のアルファベット (全角，半角) は大文字に変換されます。

INVERT
指定した文字列のアルファベット (全角，半角) が大文字の場合には小文字に変換され，小文字の場合には大文字に変換されます。

LATIN
指定した文字列の全角の Latin 文字は半角の Latin 文字に変換されます。

WIDE_LATIN
指定した文字列の半角の Latin 文字は全角の Latin 文字に変換されます。

SIZE_INVERT
指定した文字列の半角の Latin 文字は全角に，全角の Latin 文字は半角に変換されます。

HIRAGANA
指定した文字列のカタカナ (全角) および半角カナはひらがなに変換されます。

KATAKANA
指定した文字列のひらがなおよび半角カナはカタカナ (全角) に変換されます。

KANA_INVERT
指定した文字列のひらがなはカタカナ (全角) に，カタカナ (全角) はひらがなに変換されます。

NARROW_KANA
指定した文字列のひらがなおよびカタカナ (全角) は半角カナに変換されます。

KANA
指定した文字列のアルファベット (半角，全角) はローマ字として扱われ，ひらがなに変換されます。また，文字列中のカタカナおよび半角カナはひらがなに変換されます。

戻り値

returned_buffer	第 1 パラメータにバッファを指定したときに，変更されたテキストが含んだバッファが戻されるバッファ型の変数。変数 "returned_buffer"は第 1 パラメータで指定されたバッファと同じバッファを示します。
returned_range	第 1 パラメータにレンジを指定したときに，変更されたテキストを含んだレンジが戻されるレンジ型の変数。変数 "returned_range"は第 1 パラメータで指定されたレンジと同じレンジを示します。
returned_string	第 1 パラメータに文字列を指定したときに，変更されたテキストを含む文字列が戻される文字列型の変数。IN_PLACE を指定したときにも，文字列は戻されます。

表 4–1 は CHANGE_CASE の各キーワードとその変換項目です。

表 4–1 CHANGE_CASE のキーワードと変換項目

	キーワード							
文字列	LATIN	WIDE_ LATIN	KANA	HIRA GANA	KATA KANA	KANA_ INVERT	NARROW_ KANA	SIZE_ INVERT
半角	none	全角	ひら	none	none	none	none	全角
全角	半角	none	ひら	none	none	none	none	半角
ひらがな	none	none	none	none	カタ	カタ	半カ	none
カタカナ	none	none	ひら	ひら	none	ひら	半カ	none
半角カナ	none	none	ひら	ひら	カタ	none	none	none

none = 変換されない
ひら = ひらがな
カタ = カタカナ
半カ = 半角カナ

シグナル・エラー

XTPU$_BADKEY	WARNING	間違ったキーワードが指定された
XTPU$_NOTMODIFIABLE	WARNING	変更できないバッファの中のテキストを変更することはできない
XTPU$_ARGMISMATCH	ERROR	引数の型が正しくない
XTPU$_CONTROLC	ERROR	CHANGE_CASE の実行中に Ctrl/C が押された
XTPU$_INVPARAM	ERROR	パラメータのデータ・タイプが間違っている
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

CHANGE_CODE

この組込みプロシージャは，指定されたキーワードに従って，指定されたバッファまたはレンジ中のテキストのコードセットを変換します。

形式

[{returned_buffer |
returned_range} :=] CHANGE_CODE *({buffer | range }, keyword1, keyword2)*

引数

buffer
変更したい文字列を含むバッファ。

range
変更したい文字列を含むレンジ。

keyword1
変換前のコードセットを示すキーワード。有効なキーワードは，DEC_MCS， ISO_LATIN1， ASCII_JISKANA， DECKANJI，DECKANJI2000， SDECKANJI，SJIS， ISO2022JP， UCS2，および UTF8 です。

keyword2
変換後のコードセットを示すキーワード。有効なキーワードは，DEC_MCS， ISO_LATIN1， ASCII_JISKANA， DECKANJI，DECKANJI2000， SDECKANJI，SJIS， ISO2022JP， UCS2，および UTF8 です。

戻り値

returned_buffer	第 1 パラメータにバッファを指定したときに，変更されたテキストが含んだバッファが戻されるバッファ型の変数。変数 "returned_buffer"は第 1 パラメータで指定されたバッファと同じバッファを示します。
returned_range	第 1 パラメータにレンジを指定したときに，変更されたテキストを含んだレンジが戻されるレンジ型の変数。変数 "returned_range"は第 1 パラメータで指定されたレンジと同じレンジを示します。

説明

この組込みプロシージャは，あるコードセットを指定してファイルから読み込んだバッファの内容のすべて，または一部を別のコードセットと解釈し直す場合に使用されます。たとえば，DEC 漢字コードセットと思って読み込んだファイルが，実際は ISO Latin1 コードセットで書かれていた場合，この組込みプロシージャを使用すれば，もう一度 ISO Latin1 コードセットを指定してファイルから読み直すことなしに，バッファ中のコードセットの解釈のみを ISO Latin1 コードセットに変えることができます。これによって正しい文字が画面に表示されることになります。

シグナル・エラー

XTPU$_BADKEY	WARNING	間違ったキーワードが指定された
XTPU$_NOTMODIFIABLE	WARNING	変更できないバッファの中のテキストを変更することはできない
XTPU$_CONTROLC	ERROR	CHANGE_CODE の実行中に Ctrl/C が押された
XTPU$_INVPARAM	ERROR	パラメータのデータ・タイプが間違っている
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

CODE

この組込みプロシージャは， UCS-2 (Universal Character Set (ISO 10646) の 2 オクテット表現，いわゆる Unicode) と文字との相互変換を行います。またキーに対応する文字を得るためにも使用されます。

形式

{integer3|string2} := CODE *({integer1|keyword|string1} [， integer2])*

引数

integer1
UCS-2 での文字の 10 進数表現。 0 から 65535 までの値が有効です。それ以外の数字を渡したときの動作は定義されていません。

keyword
対応する文字が必要なキーの名前。プリント可能な文字に対応しないキーのキー名を指定した場合には， CODE は UCS-2 の値が 0 である文字を戻します。

string1
対応する UCS-2 の値が必要な文字。 1 文字より長い文字列を指定した場合には，CODE は最初の文字の値を戻します。

integer2
文字のカラム幅を示す整数。このパラメータは第 1 パラメータに整数を指定したときのみ指定できます。省略時の値は 1 です。漢字など 2 カラム文字の UCS-2 コードから文字列への変換を行うときは，このパラメータに 2 を指定しなければなりません。

説明

第 1 パラメータに整数が与えられたときには，このプロシージャは第 2 パラメータに指定された数字に従って， 1 カラムあるいは 2 カラムの文字を通知します。第 1 パラメータの数字と文字の対応は， UCS-2 の規定によって決められています。変換されたコードに対応する文字が表示できるかどうかはチェックされません。

文字列パラメータが与えられたときには，このプロシージャは最初の文字に対応する UCS-2 コードを通知します。

シグナル・エラー

XTPU$_NULISTRING	WARNING	長さ 0 の文字列が渡された
XTPU$_ARGMISMATCH	ERROR	引数のデータ・タイプが正しくない
XTPU$_NEEDTOASSIGN	ERROR	CODE は代入文の右辺でのみ使用できる
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる

COLUMN_LENGTH

この組込みプロシージャは，文字列またはレンジに含まれる文字の，画面上のカラム数を示す整数を通知します。

形式

integer := COLUMN_LENGTH *({buffer | range | string})*

引数

buffer
カラム長を知りたいバッファの名前。バッファを指定した場合，行の終端はカラム数としてカウントされないので注意してください。

range
カラム長を知りたいレンジの名前。レンジを指定した場合，行の終端はカラム数としてカウントされないので注意してください。

string
カラム長を知りたい文字列。

説明

バッファ，レンジ，あるいは文字列の中に TAB 文字が含まれていたときは， TAB 文字のカラム数は表示されているカラム数に関わらず 1 カラムとして計算されます。

シグナル・エラー

XTPU$_ARGMISMATCH	ERROR	COLUMN_LENGTH のパラメータは，バッファ，レンジ，文字列のいずれかでなければならない
XTPU$_CONTROLC	ERROR	COLUMN_LENGTH の実行中に Ctrl/C が押された
XTPU$_NEEDTOASSIGN	ERROR	COLUMN_LENGTH は代入文の右辺でのみ使用できる
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる

XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

CONVERT_KANA

この組込みプロシージャは，日本語 OpenVMS かな漢字変換ライブラリを使用して，ひらがなまたはカタカナの文字列を漢字に，また，ローマ字をひらがな/カタカナの表示に変換します。

形式

[{strint2|integer2} :=] CONVERT_KANA *(keyword [，{string1 | integer1}])*

引数

keyword
変換に関する各指示を指定するキーワードで，下記の中から 1 つを指定します。指定するキーワードによって，その他のパラメータや戻り値のデータ・タイプが異なります。

START_CONVERSION	新しい変換を開始し，変換文字列を返します
FORWARD	現在の文節の次候補を含む変換文字列を返します
REVERSE	現在の文節の前候補を含む変換文字列を返します
NONE	現時点での変換文字列を返します
END_CONVERSION	漢字辞書を更新します
HIRAGANA	現在の文節をひらがなに変換し，変換文字列を返します
KATAKANA	現在の文節をカタカナ (全角) に変換し，変換文字列を返します
NARROW_KANA	現在の文節を半角カナに変換し，変換文字列を返します
ROMAN	現在の文節を全角英数字に変換し，変換文字列を返します
SHRINK	現在の文節の長さを縮小し，変換文字列を返します
EXPAND	現在の文節の長さを拡大し，変換文字列を返します
CLAUSE_OFFSET	現在の文節の漢字列中の文字オフセットを返します
CLAUSE_LENGTH	現在の文節の文字長を返します
PHONETIC_OFFSET	現在の文節の読み文字列中の文字オフセットを返します
PHONETIC_LENGTH	現在の文節の読みの長さを返します
CLAUSE_NEXT	次の文節に移動します
CLAUSE_PREVIOUS	前の文節に移動します
CLAUSE_NUMBER	現在の文節番号を設定します
MAX_CLAUSE_NUMBER	現在の変換文字列の文節数を返します

string1
新しい変換を開始するときの読み文字列。かな漢字変換は全角かなの部分を対象とします。キーワードが START_CONVERSION の場合にのみ必要で，それ以外の場合に指定してはいけません。

integer1
現在の文節を指定した番号の文節に設定します。キーワードが CLAUSE_NUMBER の場合のみに有効で，それ以外の場合に指定してはいけません。

キーワードが CLAUSE_OFFSET の場合，現在の文節の変換文字列中での文字オフセットの値が整数値で返されます。 CLAUSE_LENGTH の場合，現在の文節の変換後文字列に占める文字長が整数値で返されます。

キーワードが CLAUSE_OFFSET， CLAUSE_LENGTH の場合，変換された文字列を返します。この文字列は，現在の文節に対応する文字列ではなく，string1 で指定した読み文字列全体に対する変換文字列です。ただし， FORWARD， REVERSE， HIRAGANA などの変換操作の対象となるのは，文字列全体ではなく，現在の文節に対応する文字列です。

説明

学習機能

かなを漢字に変換する場合，いくつかの候補文字 (同音異字語) が存在することがあります。このプロシージャは， START_CONVERSION のキーワードを指定すると辞書の第 1 候補を返し， FORWARD， REVERSE を指定するとそれぞれ次候補，前候補を返します。また， END_CONVERSION を指定するとその時点での候補の，次に同じ単語の変換 (START_CONVERSION) を開始する時の優先順位が上がるので，よく使われる単語は再変換の回数が少なくてすむようになります。

変換機能

ASCII(半角) / ROMAN(全角) からひらがな，カタカナ，ローマ字へ，あるいはひらがな，カタカナから漢字への変換が可能です。

START_CONVERSION	string1 で指定された読みの変換を開始します。このとき，string1 に対する文法解析を行い，変換文字列は 1 またはそれ以上の文節に分けられます。このとき，現在の文節は 1 番目の文節となります。
FORWARD	現在の文節の自立語の次候補を求め，新しい変換文字列を返します。
REVERSE	現在の文節の自立語の前候補を求め，新しい変換文字列を返します。
NONE	現時点での変換文字列をそのまま返します。

END_CONVERSION	変換文脈を終了し，変換文字列を確定するとともに，個人辞書の学習を行います。
HIRAGANA	現在の文節をひらがなに変換し，新しい変換文字列を返します。 ASCII， ROMAN，ひらがなおよびカタカナからなる部分が変換対象となります。
KATAKANA	現在の文節をカタカナ (全角) に変換し，新しい変換文字列を返します。 1 回目に呼ばれた時は文節の自立語のみをカタカナに変換し， 2 回目に呼ばれた時は文節全体をカタカナに変換します。 START の時 string1 にはひらがなで渡されていなければなりません。
NARROW_KANA	現在の文節を半角カナに変換し，新しい変換文字列を返します。 1 回目に呼ばれた時は文節の自立語のみを半角カナに変換し， 2 回目に呼ばれた時は文節全体を半角カナに変換します。 START の時 string1 にはひらがなで渡されていなければなりません。
ROMAN	現在の文節を全角英数字に変換し，新しい変換文字列を返します。 ASCII からなる部分のみが変換対象となります。
SHRINK	現在の文節の長さを 1 文字分縮小し，文法解析を再び行って新しい変換文字列を返します。文法解析の結果，文節の長さが 1 以上縮小されることもあります。
EXPAND	現在の文節の長さを拡大し，文法解析をやりなおして新しい変換文字列を返します。

一般的に，文節の縮小/拡大を実行すると，現在の文節以降の文字列が変化し，それに伴って文節数も変化する可能性が高いので注意してください。

CLAUSE_OFFSET	現在の文節が，変換文字列の中のどの位置から始まるかを文字オフセットとして返します。文節移動を行った結果として値が変化します。
CLAUSE_LENGTH	現在の文節の長さを返します。変換操作を行った後は値が変化する可能性があります。
PHONETIC_OFFSET	現在の文節が，読み文字列の中のどの位置から始まるかを文字オフセットとして返します。文節移動を行った結果として値が変化します。
PHONETIC_LENGTH	現在の文節の読み文字列の長さを返します。変換操作を行った後は値が変化する可能性があります。
CLAUSE_NEXT	変換対象を次の文節に移動します。この結果，FORWARD， REVERSE などの対象となる部分が次の文節に移動します。
CLAUSE_PREVIOUS	変換対象を前の文節に移動します。現在の文節が先頭の文節である場合，最後の文節に移動します。この結果，FORWARD， REVERSE などの対象となる部分が次の文節に移動します。
CLAUSE_NUMBER	パラメータ integer1 で指定した文節を現在の文節とします。integer1 を指定しなかったときは，現在の文節の文節番号を返します。
MAX_CLAUSE_NUMBER	現在の変換文字列の文節数を返します。

シグナル・エラー

XTPU$_ROUND	INFORMATION	FORWARD による次候補が一巡して最初の単語に戻った
XTPU$_NEEDTOASSIGN	ERROR	返される値を代入する変数が必要である
XTPU$_NODIC	WARNING	この編集セッションでは辞書を使用していない
XTPU$_BADVALUE	ERROR	指定した整数値が有効な範囲内にない
XTPU$_CNVERR	ERROR	かな漢字変換ルーチンの内部エラーが発生した
XTPU$_DICUPDERR	ERROR	個人辞書の更新時にエラーが発生した
XTPU$_INVPARAM	ERROR	引数の型が正しくない
XTPU$_NOCLA	ERROR	変換対象となる文節が指定されていない，変換を開始していない
XTPU$_NODICENT	WARNING	文節の縮小あるいは拡大ができない。単語が辞書にない (変換不可能)
XTPU$_STRTOOLONG	ERROR	入力文字数が 253 文字を越えた
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる
XTPU$_TRUNCATE	WARNING	変換文字列が長すぎるため，切り捨てが行われた

DEC_KANJI

この組込みプロシージャは，文字と文字コード (ASCII コードあるいは DEC 漢字コード) の相互変換をします。

形式

{integer2 | string2} := DEC_KANJI *({integer1 | string1})*

引数

integer1
ASCII 文字あるいは漢字に変換したい整数。 ASCII あるいは DEC 漢字セットで決められた整数 (10 進数)。

string1
ASCII コードあるいは DEC 漢字コードを得たい文字。

説明

整数パラメータが与えられたときには，このプロシージャは指定された数字に従って，文字を通知します。数字と文字の対応は， ASCII および DEC 漢字セット規定によって決められています。文字が定義されていない値が指定されたときには，値が 0 の文字を戻します。

文字列パラメータが与えられたときには，このプロシージャは最初の文字に対応する ASCII コードあるいは DEC 漢字コードを戻します。

シグナル・エラー

XTPU$_NULLSTRING	WARNING	長さ 0 の文字列が渡された
XTPU$_ARGMISMATCH	ERROR	引数のデータ・タイプが正しくない
XTPU$_NEEDTOASSIGN	ERROR	DEC_KANJI は代入文の右辺でのみ使用できる
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる

例

次のサンプル・プロシージャは， JIS 漢字セットの任意の 1 区のテーブルをバッファに挿入します。たとえば， user_kanji_list (20) を実行すると，第 20 区のテーブルを現在のカーソルの位置に挿入します。

```
PROCEDURE user_kanji_list (jis_ku)
LOCAL cnt, col, low_byte;

cnt := jis_ku * 256 + 41120;
max := cut + 94;

COPY_TEXT (FAO('第 !ZL 区', jis_ku));
SPLIT_LINE;
COPY_TEXT ("0 1 2 3 4 5 6 7 8 9 A B C D E F");
SPLIT_LINE;

COPY_TEXT (user_hex(cnt) + ' ');
col := 1;
LOOP
   EXITIF cnt > max;
   IF col > 16 THEN
      SPLIT_LINE;
      UPDATE (current_window);
      col := 1;
      COPY_TEXT (user_hex(cnt) + ' ');
   ENDIF;
   low_byte := cnt-((cnt / 256)*256);
   IF low_byte <> 160 THEN
      COPY_TEXT (' '+ DEC_KANJI(cnt));
   ELSE
      COPY_TEXT ('  ');
   ENDIF;
   cnt := cnt + 1;
   col := col + 1;
ENDLOOP;
SPLIT_LINE;

ENDPROCEDURE;

PROCEDURE user_hex (dec_num)
LOCAL res, rmn, temp;
```

```
temp := dec_num;
IF temp =0 THEN
   res := '0'
ELSE
   res := ''
ENDIF;
LOOP
   EXITIF temp  <= 0;
   rmn := temp-((temp / 16) * 16);
   temp := temp / 16;
   IF (0 < rmn) AND (rmn < 16) THEN
      res := SUBSTR ('123456789ABCDEF', rmn, 1) + res;
   ELSE
      res := '0' + res;
   ENDIF;
ENDLOOP;
user_hex := res;

ENDPROCEDURE;
```

DELETE_TANGO

この組込みプロシージャは，単語とその読みを個人辞書から削除します。

形式

DELETE_TANGO *(string1, string2)*

引数

string1
辞書から削除される単語の読みを表わすひらがな文字列。

string2
辞書から読みを削除される単語。

説明

この組込みプロシージャは，個人辞書から単語の読みを削除します。これは日本語 OpenVMS かな漢字変換ライブラリを使用しています。単語とその読みは，個人辞書に登録されているものでなければなりません。

『日本語ライブラリ 利用者の手引き』 "かな漢字変換ライブラリ "の JLB$DEL_TANGO を参照してください。

シグナル・エラー

XTPU$_NODIC	WARNING	この編集セッションでは個人辞書は使用していない
XTPU$_NODICENT	WARNING	個人辞書に指定された単語が見つからない
XTPU$_DICUPDERR	ERROR	個人辞書の変更時にエラーが発生した
XTPU$_INVPARAM	ERROR	引数のデータ・タイプが正しくない
XTPU$_NORETURNVALUE	ERROR	DELETE_TANGO は値を返さない
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる

例

```
1. DELETE_TANGO ("じゅうしょ", "東京都あきる野市")
```

このステートメントは，個人辞書に登録されている単語 "東京都あきる野市 "と，その読み "じゅうしょ "を個人辞書から削除します。

次のサンプル・プロシージャは，選択された単語を個人辞書から削除します。読みがなを引数としてとり，引数が空文字であると，その読みをローマ字で入力するように要求します。単語が選択されていない場合や削除に失敗した (単語とその読みが正しくない) ときにはエラー・メッセージが表示されます。このプロシージャで用いている user_select_position は，選択開始位置を示すグローバル変数です。

```
! This procedure specifies start position of selection
PROCEDURE user_select

ON_ERROR
    IF error = XTPU$_ONESELECT THEN
        MESSAGE ("すでに選択されています。");
        RETURN;
    ENDIF;
ENDON_ERROR;

user_select_position := SELECT (reverse);

ENDPOROCEDURE

! This procedure removes selected TANGO and its YOMIGANA
! from personal dictionary.
PROCEDURE user_delete_tango (yomi)

LOCAL yomigana, tango_range, tango;

ON_ERROR
   IF (error = XTPU$_NOSELECT) OR (error = XTPU$_SELRANGEZERO) THEN
       MESSAGE ("単語が選択されていません。");
   ELSE
       MESSAGE ("単語を削除できません。");
   ENDIF;
   user_select_position := 0;
   RETURN;
ENDON_ERROR;

tango_range := select_range;
user_select_position := 0;  ! This global variable is used to
                            ! specify start position of selection
POSITION (BEGINNING_OF(tango_range));
tango := SUBSTR (tango_range, 1, LENGTH(tango_range));
tango_range := 0;
yomigana := yomi;
IF yomigana = ''THEN
   yomigana := READ_LINE ("読みがなをローマ字で入力してください:");
ENDIF;
CHANGE_CASE (yomigana, KANA);
```

```
    DELETE_TANGO (yomigana, tango);
    ENDPROCEDURE;
```

ENTER_TANGO

この組込みプロシージャは，単語とその読みを個人辞書に登録します。

形式

ENTER_TANGO *(string1, string2)*

引数

string1
個人辞書に登録する単語の読みを表わすかな文字列。

string2
個人辞書に登録する単語。

説明

このプロシージャは，個人辞書に新しい単語とその読みを登録します。これは日本語 OpenVMS かな漢字変換ライブラリを使用しています。単語とその読みは，個人辞書に登録されているものであってはなりません。

『日本語ライブラリ 利用者の手引き』の "かな漢字変換ライブラリ "の JLB$ENT_TANGO を参照してください。

シグナル・エラー

XTPU$_NODIC	WARNING	この編集セッションでは個人辞書は使用していない
XTPU$_DICUPDERR	ERROR	個人辞書の変更時にエラーが発生した
XTPU$_INVPARAM	ERROR	引数のデータ・タイプが正しくない
XTPU$_NORETURNVALUE	ERROR	ENTER_TANGO は値を返さない
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる

例

1. ENTER_TANGO ("じゅうしょ", "東京都あきる野市")

このステートメントは，個人辞書に単語 "東京都あきる野市 "と，その読み "じゅうしょ "を新しく登録します。

次のサンプル・プロシージャは，選択された単語を個人辞書に登録します。読みがなを引数としてとり，引数が空文字であると，その読みをローマ字で入力するように要求します。単語が選択されていないときや登録に失敗した (単語とその読みが正しくない) ときにはエラー・メッセージを表示します。このプロシージャで用いている user_select_position は，選択開始位置を示すグローバル変数です。

```
PROCEDURE user_select

! This procedure specifies start position of selection

ON_ERROR
    IF error = XTPU$_ONESELECT THEN
        MESSAGE ("すでに選択されています。");
        RETURN;
    ENDIF;
ENDON_ERROR;

user_select_position := SELECT (REVERSE);

ENDPROCEDURE;

! This procedure reserves selected TANGO and its YOMIGANA
! in personal dictionary.
PROCEDURE user_enter_tango (yomi)

LOCAL yomigana, tango_range, tango;

ON_ERROR
    IF (error = XTPU$_NOSELECT) OR (error = XTPU$_SELRANGEZERO) THEN
        MESSAGE ("単語が選択されていません。");
    ELSE
        MESSAGE ("単語を登録できません。");
    ENDIF;
    user_select_position := 0;
    RETURN;
ENDON_ERROR;

tango_range := select_range;
user_select_position := 0;  ! This global variable is used to
                            ! specify start position of selection
POSITION (BEGINNING_OF(tango_range));
tango := SUBSTR (tango_range, 1, LENGTH(tango_range));
tango_range := 0;
yomigana := yomi;
IF yomigana = '' THEN
   yomigana := READ_LINE ("読みがなをローマ字で入力してください：");
ENDIF;
CHANGE_CASE (yomigana, KANA);
```

```
ENTER_TANGO (yomigana, tango);
ENDPROCEDURE;
```

FAO

この組込みプロシージャは，制御文字列を書式整形された ASCII 出力文字列に変換するために， Formatted ASCII Output ($FAO) システム・サービスを呼び出します。 FAO ディレクティブに対する引数を制御文字列として指定することにより，$FAO システム・サービスによって実行される処理を制御することができます。 FAO 組込みプロシージャは書式整形された ASCII 出力を含む文字列を通知します。

FAO には最大 127 個までのパラメータを指定することができます。

FAO は，指定された文字列をメッセージ・コードセットに指定されたコードセットに変換して，$FAO システム・サービスを呼び出します。このため，そのときのメッセージ・コードセットで表現できない文字は， FAO を呼び出した結果，失われます。メッセージ・コードセットについて詳しくは， SET (MESSAGE_CODESET) 組込みプロシージャを参照してください。

$FAO システム・サービスについての詳しい説明は，『OpenVMS System Services Reference Manual』を参照してください。

形式

string2 := FAO *(string1 [， FAO parameters])*

引数

string1
引用符で囲まれた文字列，文字列定数を含む変数名，または文字列を表わす式であり，固定長テキストの出力文字列と FAO ディレクティブから構成されます。

文字列の一部として指定することができる FAO ディレクティブは下記のとおりです。

!AS	文字列を入力されたとおりに挿入する
!OL	倍長語 (ロングワード) を 8 進数に変換する
!XL	倍長語 (ロングワード) を 16 進数に変換する
!ZL	倍長語 (ロングワード) を 10 進数に変換する
!UL	倍長語 (ロングワード) を 10 進数に変換するが，負の数値に対する調整は実行しない
!SL	倍長語 (ロングワード) を 10 進数に変換し，負の数値も正しく変換する
!/	新しい行を挿入する (キャリッジ・リターン/ライン・フィード)
!_	タブを挿入する

!^	フォーム・フィードを挿入する
!!	感嘆符を挿入する
!%S	最後に変換された数値が 1 でない場合には S を挿入する
!%T	パラメータとして 0 を入力した場合には，現在の時刻を挿入する (DEC XTPU は 4 倍長語を使用しないため，特定の時刻を渡すことはできない)
!%D	パラメータとして 0 を入力した場合には，現在の日付と時刻を挿入する (DEC XTPU は 4 倍長語を使用しないため特定の日付を渡すことはできない)

FAO パラメータについては『OpenVMS System Services Reference Manual』を参照してください。一般的には， FAO パラメータは string1 の FAO ディレクティブに対応しています。

説明

FAO 組込みプロシージャについての説明は，『Guide to the DEC Text Processing Utility』を参照してください。

FAO 組込みプロシージャの引数に指定される文字列は，メッセージ・コードセットに指定される文字列でなければなりません。メッセージ・コードセットについての詳しい説明は， SET (MESSAGE_CODESET) を参照してください。

シグナル・エラー

XTPU$_INVFAOPARAM	WARNING	引数が整数または文字列でない
XTPU$_INVPARAM	ERROR	FAO の第 1 引数は文字列でなければならない
XTPU$_NEEDTOASSIGN	ERROR	FAO 組込みプロシージャは代入文の右辺でのみ使用できる
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる

FILL

この組込みプロシージャは，指定されたバッファまたはレンジに含まれるテキストのフォーマットを変更することにより，テキスト・ラインをほぼ同じ長さにそろえます。

形式

FILL *({buffer|range}, [string [， integer1 [， integer2 [， integer3]]]])*

引数

buffer
形式を変更したいテキストを含むバッファ。

range
形式を変更したいテキストを含むレンジ。

string
テキストをバッファに挿入するときにワード区切り文字として使用したい文字の集合。半角の空白文字は常にワード区切り文字です。

integer1
左マージンの値。左マージンの値は 1 以上で，右マージンの値より小さくなければなりません。省略時にはバッファの左マージンが使われます。

integer2
右マージンの値。右マージンの値は左マージンより大きく，バッファの最大レコード・サイズ以下でなければなりません。省略時にはバッファの右マージンが使われます。

integer3
最初の行の段付けの値。この値は最初の行の左マージンを変更します。値は正負ともに有効ですが，この値を左マージンに足した結果が 1 以上で，かつ右マージンより小さくなければなりません。省略時には 0 が使われます。

説明

FILL 組込みプロシージャに関しての基本的な説明は，『DEC Text Processing Utility Reference Manual』を参照してください。

DEC XTPU では FILL 組込みプロシージャが 2 カラム以上の文字 (複数カラム文字) を扱うことができます。複数カラム文字が含まれているときには，1 カラム文字のワード区切り文字に加えて，2 つの複数カラム文字の間もワード区切りとして扱われます。すなわち，2 つの複数カラム文字の間で行が分割されることがあります。

DEC XTPU の FILL 組込みプロシージャでは，日本語の禁則処理をすることができます。行頭禁則文字の指定には，SET (FILL_NOT_BEGIN) を，行末禁則文字の指定には，SET (FILL_NOT_END) を使います。また，SET (MARGIN_ALLOWANCE) で，右マージンの右側に置くことのできる行頭禁則文字の文字数を指定することができます。

Latin 文字からなる文章に FILL を実行したときに行末に空白文字がくると，その空白文字は削除されます。また，FILL の結果複数行がつながると，その間に空白文字が置かれます。しかし複数カラム文字が含まれているときには，この機能が不要になることがあります。そのようなときには，SET (FILL_TRIM_SPACE) で FILL を実行するときに，空白文字の削除および追加を行うかどうかを制御できます。

シグナル・エラー

XTPU$_BADMARGINS	WARNING	正しくない FILL のマージンが指定された
XTPU$_INVRANGE	WARNING	正しくないレンジ領域が指定された
XTPU$_NOTMODIFIABLE	WARNING	変更が禁止されているバッファで FILL を実行することはできない
XTPU$_ARGMISMATCH	ERROR	パラメータのデータ・タイプが正しくない
XTPU$_CONTROLC	ERROR	FILL の実行中に Ctrl/C が押された
XTPU$_INVPARAM	ERROR	パラメータのデータ・タイプが正しくない
XTPU$_NOCACHE	ERROR	割り当てられたメモリの不足で新しい行が作れない
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

GET_INFO

この組込みプロシージャは，編集コンテキストの現在のステータスに関する情報を通知します。 GET_INFO が通知する情報のリストのうち， DEC XTPU で拡張されたものが表 4–2 と表 4–3 にまとめられているので参照してください。

形式

return_value := GET_INFO *(parameter1, parameter2)*

形式

return_value := GET_INFO *(parameter1, parameter2, parameter3)*

引数

parameter1
DEC XTPU のデータ・タイプまたはキーワード。 GET_INFO は，parameter1 として指定されたアイテムに関する情報を通知します。parameter1 として指定できるデータ・タイプのうち DEC XTPU で拡張されたものについては，表 4–2 を参照してください。また，parameter1 として指定できるキーワードのうち DEC XTPU で拡張されてものについては，表 4–3 を参照してください。

parameter2
引用符で囲まれた文字列，変数名，または表 4–2 または表 4–3 の文字列定数を表わす式。parameter2 として使用される文字列は，parameter1 で指定されたアイテムに関して要求される情報の種類を指定します。文字列は大文字で入力しても小文字で入力してもかまいません。

parameter3
引用符で囲まれた文字列，変数名。

parameter1 が SCREEN で paramerter2 が "video_character_set"という文字列のときは，parameter3 には文字セットを示すキーワードを与えます。有効なキーワードは DEC_SUPPLEMENTAL， LATIN1_SUPPLEMENTAL， JIS_ROMAN， JIS_KATAKANA， KANJI_1，または KANJI_UDC です。

説明

要求される情報の種類に応じて，GET_INFO は以下のいずれかを通知します。

- アレイ
- バッファ
- 整数 (真という値は 1 として，また偽という値は 0 として通知される)
- キーワード
- マーカ
- プロセス
- レンジ
- 文字列
- ウィンドウ

表 4–2 GET_INFO - parameter1 として変数を使用する場合

parameter1	parameter2	戻り値	戻り値の説明
バッファ変数	"character_index"	整数	現在の編集位置の文字境界からのカラム・オフセット
	"character_length"	整数	現在の編集位置の文字のカラム長
	"codeset"	キーワード	バッファのコードセットを示すキーワード
	"output_codeset"	キーワード	バッファの出力コードセットを示すキーワード
マーカ変数	"character_index"	整数	マーカの文字境界からのカラム・オフセット
	"character_length"	整数	マーカのある文字のカラム長

表 4–3 GET_INFO - parameter1 としてキーワードを使用する場合

parameter1	parameter2	戻り値	戻り値の説明
COMMAND_LINE	"codeset"	キーワード	/CODESET=に指定したコードセット名
	"kanji_dictionary"	[1]整数 (1 か 0)	/KANJI_DICTIONARY がアクティブであるかどうかを示す値 (省略時の値によって，または DEC XTPU を起動するときに/KANJI_DICTIONARY を指定することによって)

[1] 1 という整数値は条件が真であることを示し，0 という整数値は偽であることを示す。

(次ページに続く)

表 4–3 (続き) GET_INFO - parameter1 としてキーワードを使用する場合

parameter1	parameter2	戻り値	戻り値の説明
	"kanji_dictionary_file"	文字列	/KANJI_DICTIONARY=に指定した個人辞書ファイル名
SCREEN	"video_character_set"	[1]整数 (1 か 0)	第 3 パラメータで指定した文字セットを画面に表示できるかどうかを示す値
SYSTEM	"codeset"	キーワード	システム・コードセットを示すキーワード
	"fill_not_begin"	文字列	行頭禁則文字からなる文字列
	"fill_not_end"	文字列	行末禁則文字からなる文字列
	"fill_trim_space"	[1]整数 (1 か 0)	FILL 組込みプロシージャで分割された行の最後の空白文字を取り除くかどうかを示す値
	"jis_roman"	[1]整数 (1 か 0)	JIS ローマ字文字セットが ASCII 文字セットに代わって使われるかどうかを示す値
	"kanji_dictionary_file"	文字列	DEC XTPU の起動時に使用された個人辞書ファイル名
	"keyboard_codeset"	キーワード	キーボード・コードセットを示すキーワード
	"margin_allowance"	整数	右マージンを越えて置くことができる行頭禁則文字の数
	"message_codeset"	キーワード	メッセージ・コードセットを示すキーワード

[1]1 という整数値は条件が真であることを示し，0 という整数値は偽であることを示す。

シグナル・エラー

XTPU$_BADREQUEST	WARNING	2 番目の引数によって指定される要求が最初の引数のデータ・タイプに対して正しくないものである
XTPU$_BADKEY	WARNING	最初の引数として誤ったキーワード値，または認識されないデータ・タイプが渡されていた
XTPU$_NOBREAKPOINT	WARNING	文字列定数が使用できるのはブレーク・ポイントの後のみである
XTPU$_NOCRRENTBUF	WARNING	現在のバッファが定義されていない
XTPU$_NOKEYMAP	WARNING	キー・マップが定義されていない
XTPU$_NOKEYMAPLIST	WARNING	キー・マップ・リストが定義されていない
XTPU$_NONAMES	WARNING	要求された名前にマッチするものがない
XTPU$_INVPARAM	ERROR	引数のデータ・タイプに誤りがある

XTPU$_NEEDTOASSIGN	ERROR	GET_INFO 組込みプロシージャは代入文の右辺でのみ使用できる
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる
XTPU$_UNKEYWORD	ERROR	引数に誤ったキーワードが使われた

例

```
1. my_dictionary := GET_INFO (SYSTEM, "kanji_dictionary_file");
```

この代入ステートメントは，現在使用中の個人辞書の名前を my_dictionary という変数に代入します。

```
2. current_codeset := GET_INFO (CURRENT_BUFFER, "codeset");
```

この代入ステートメントは，現在のバッファのコードセットを current_codeset という変数に代入します。

KEY_NAME

この組込みプロシージャは，1つのキーまたは複数のキーの組合わせを示すDEC XTPUキーワードを通知します。

形式

keyword2 := KEY_NAME *({integer|key-name|string} [，SHIFT_KEY] [，{FUNCTION|KEYPAD}])*

引数

integer
キーを示すキーワードの整数表現，DEC XTPUがISO Latin1文字セットに解釈する0から255までの整数。

key-name
キーに対するDEC XTPUキー名を示すキーワード。

string
メイン・キーボードのキーの値を示す文字列。

SHIFT_KEY
戻されるキーワードの前に1つまたは複数のシフト・キーが存在することを示すパラメータ。SHIFT_KEYはDEC XTPUが定義するSHIFTキーで，(省略時の値はPF1)，キーボードのShiftキーではありません。

FUNCTION
パラメータでファンクション・キーによって発生するコードを指定していることを示す。

KEYPAD
パラメータでキーパッドによって発生するコードを指定していることを示す。

説明

KEY_NAME組込みプロシージャについての基本的な説明は，『DEC Text Processing Utility Reference Manual』を参照してください。

stringパラメータに，日本語の全角文字を指定することができます。また，日本語の全角文字もキー定義に使用することができます。

シグナル・エラー

XTPU$_BADKEY	WARNING	指定できるキーワードは SHIFT_KEY, FUNCTION, KEYPAD のみである
XTPU$_INCKWDCOM	WARNING	キーワードの組み合せが正しくない
XTPU$_MUSTBEONE	WARNING	文字列は 1 文字でなければならない
XTPU$_NOTDEFINABLE	WARNING	キーに定義されない値が指定された
XTPU$_NEEDTOASSIGN	ERROR	KEY_NAME は代入文の右辺でのみ使用できる
XTPU$_ARGMISMATCH	ERROR	パラメータのデータ・タイプが正しくない
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

例

```
1. key1 := KEY_NAME ('タ')
```

この代入ステートメントはキーボードの "タ "というキーに対して key1 というキー名を作成します。

```
2. key2 := KEY_NAME ('タ', SHIFY_KEY)
```

この例では，キーの組合せに対してキー名を作成するために KEY_NAME が使用されています。

MARK

この組込みプロシージャは，現在のバッファの現在の文字位置にマーカを設定して，そのマーカを戻します。マーカがスクリーンにどのように表示されるかを (NONE, REVERSE, BOLD, または UNDERLINE) 指定しなければなりません。

形式

marker := MARK *(keyword)*

引数

keyword
以下のいずれかのキーワードを使用しなければなりません。

BLINK	マーカが点滅することを指定します。
BOLD	マーカがボールド体で表示されることを指定します。
FREE_CURSOR	マーカがフリー・マーカ (文字に対応しないマーカ) であることを指定します。 MARK (FREE_CURSOR) が実行されたときのカーソル位置が，行頭より前，行末より後ろ，タブの中， DEC 漢字文字の第 2 カラム上，バッファの終端より下でないときにはフリー・マーカにはなりません。フリー・マーカはビデオ属性を持ちません。
NONE	ビデオ属性がマーカに適用されないことを指定します。
REVERSE	マーカが反転表示されることを指定します。
UNDERLINE	マーカにアンダーラインが引かれることを指定します。

説明

MARK 組込みプロシージャに関する基本的な説明は，『DEC Text Processing Utility Reference Manual』を参照してください。

マーカは文字の境界にセットされます。したがって，たとえば漢字上にカーソルがあるときに，マーカをセットしようとすると，現在の文字位置が漢字の 2 カラム目にあるときでも，マーカはその文字の 1 カラム目にセットされます。ただしフリー・マーカは漢字の 2 カラム目にセットすることができます。

シグナル・エラー

XTPU$_NOCURRENTBUF	WARNING	マーカはバッファ中にのみ設定できる
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる
XTPU$_NEEDTOASSIGN	ERROR	MARK は代入文の右辺でのみ使用できる
XTPU$_INVPARAM	ERROR	パラメータのデータ・タイプが正しくない
XTPU$_BADKEY	ERROR	指定できるキーワードは NONE, BOLD, BLINK, REVERSE, UNDERLINE, FREE_CURSOR のいずれかである
XTPU$_UNKKEYWORD	ERROR	正しくないキーワードが指定された
XTPU$_INSVIRMEM	FATAL	マーカを作るのに十分なメモリがない

PCS_CLASS

この組込みプロシージャは，指定した文字が属する PCS (Primitive Character Set) をビット・フィールド形式で整数値として戻します。

形式

integer2 := PCS_CLASS *(string [， integer1])*

引数

string
文字が属する PCS を知りたい文字を指定します。 2 文字以上の文字列を指定したときには最初の文字のみが有効です。

inteter1
PCS のグループ番号を指定します。このパラメータは将来 32 以上の PCS を扱うために用意されていますが，現在はグループ番号 0 のみが使われますので，省略可能です。省略時の値は 0 です。

説明

次の表は PCS を示すビットの割り当てを示しています。プログラム中では予め定義された定数と， PCS_CLASS から戻される値を論理演算で比較することになります。

クラス	ビット	定数
ASCII_CHAR	0	XTPU$K_ASCII_CHAR
DEC_SUPPLEMENTAL	1	XTPU$K_DEC_SUPP
LATIN1_SUPPLEMENTAL	2	XTPU$K_LATIN1_SUPP
JIS_ROMAN	3	XTPU$K_JIS_ROMAN
JIS_KATAKANA	4	XTPU$K_JIS_KATAKANA
KANJI_1 (JIS X0208)	8	XTPU$K_KANJI_1
KANJI_UDC	16	XTPU$K_KANJI_UDC

シグナル・エラー

XTPU$_INVPARAM	ERROR	引数の型が正しくない
XTPU$_NEEDTOASSIGN	ERROR	PCS_CLASS は代入文の右辺でのみ使用できる
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

READ_CHAR

この組込みプロシージャは，次にキーボードから入力される文字を文字列で通知します。

形式

string := READ_CHAR

引数

なし

説明

READ_CHAR 組込みプロシージャに関する基本的な説明は，『DEC Text Processing Utility Reference Manual』を参照してください。

キーボードから入力される文字コードの解釈は，キーボード・コードセットの設定に従います。詳しくは GET_INFO (SYSTEM, "keyboard_codeset") および SET (KEYBOARD_CODESET) を参照してください。

シグナル・エラー

XTPU$_NEEDTOASSIGN	ERROR	READ_CHAR は代入文の右辺でのみ使用できる
XTPU$_NOCHARREAD	WARNING	READ_CHAR が文字を読まなかった
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

READ_KEY

この組込みプロシージャは，次のキーボードからの入力を待ち，そのキーに対するキーワードを通知します。このプロシージャは，エスケープ・シーケンスを送るキーやコントロール・キーも受け付けます。

形式

keyword := READ_KEY

引数

なし

説明

READ_KEY 組込みプロシージャに関する基本的な説明は，『DEC Text Processing Utility Reference Manual』を参照してください。

キーボードから入力される文字コードの解釈は，キーボード・コードセットの設定に従います。詳しくは GET_INFO (SYSTEM, "keyboard_codeset") および SET (KEYBOARD_CODESET) を参照してください。

シグナル・エラー

XTPU$_CONTROLC	ERROR	READ_KEY の実行中に Ctrl/C が押された
XTPU$_NEEDTOASSIGN	ERROR	READ_KEY は代入文の右辺でのみ使用できる
XTPU$_REQUIRESTERM	ERROR	NODISPLAY モードでは READ_KEY を使用できない
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

READ_LINE

この組込みプロシージャは，入力を要求するプロンプトとして指定されたテキストを表示し，プロンプトに対する応答として入力された情報を読み込みます。読み込みたい文字の最大のバイト数を指定することもできます。 READ_LINE は，プロンプトに対する応答として入力されたデータを含む文字列を通知します。

形式

string2 := READ_LINE *[(string1 [， integer])]*

引数

string1
入力を要求するプロンプトとして使用されるテキストを指定します。このパラメータは省略することができます。

integer
プロンプトに対する応答として入力されたデータから何バイトを読み込むかを指定します。指定できる最大値は 132 文字です。このパラメータは省略することができます。このパラメータが指定されなかったときには，Return キーまたは Ctrl/Z が入力されるまで，あるいは文字列の長さが 132 文字になるまで入力できます。

説明

READ_LINE 組込みプロシージャに関する基本的な説明は，『DEC Text Processing Utility Reference Manual』を参照してください。

キーボードから入力される文字コードの解釈は，キーボード・コードセットの設定に従います。詳しくは GET_INFO (SYSTEM， "keyboard_codeset") および SET (KEYBOARD_CODESET) を参照してください。

シグナル・エラー

XTPU$_INVPARAM	ERROR	パラメータのデータ・タイプが正しくない

XTPU$_NEEDTOASSIGN	ERROR	READ_LINE は代入文の右辺でのみ使用できる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

SELECT

この組込みプロシージャは現在のバッファの現在の文字位置にマーカをセットして，そのマーカを返します。マーカがスクリーンにどのように表示されるかを指定しなければなりません (NONE， REVERSE， BOLD， BLINK，または UNDERLINE)。

SELECT から通知されるマーカは選択・レンジの最初の文字位置を示しています。マーカに対して指定したビデオ属性は選択・レンジに含まれるすべての文字に適用されます。選択・レンジの作成方法については， SELECT_RANGE を参照してください。

形式

marker := SELECT *(keyword)*

引数

keyword
戻されるマーカがスクリーン上でどのように表示されるかを指定します。以下のいずれかのキーワードを使用しなければなりません。

BLINK	マーカが点滅することを指定します
BOLD	マーカがボールド体で表示されることを指定します
NONE	ビデオ属性がマーカに適用されないことを指定します
REVERSE	マーカが反転表示されることを指定します
UNDERLINE	マーカにアンダーラインが引かれることを指定します

説明

SELECT 組込みプロシージャに関する基本的な説明は，『DEC Text Processing Utility Reference Manual』を参照してください。

SELECT が実行されたときに，現在の編集位置が複数カラム文字の 2 カラム目以降にあった場合には，マーカは 1 カラム目にセットされます (現在の編集位置は動きません)。

シグナル・エラー

XTPU$_BADKEY	WARNING	正しくないキーワードが指定された
XTPU$_NOCURRENTBUF	WARNING	現在のバッファが設定されていない
XTPU$_ONESELECT	WARNING	SELECT は現在のバッファですでに実行中である
XTPU$_TOOFEW	ERROR	パラメータの数が少なすぎる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる
XTPU$_NEEDTOASSIGN	ERROR	SELECT は代入文の右辺でのみ使用できる
XTPU$_INVPARM	ERROR	SELECT のパラメータがキーワードでない

SELECT_RANGE

この組込みプロシージャは SELECT 組込みプロシージャを使ってセットされたマーカと現在の文字位置の間のすべての文字を含むレンジを通知します。 SELECT_RANGE の場合，選択されたレンジの中の最後の位置には文字は含まれません。

形式

range := SELECT_RANGE

引数

なし

説明

SELECT_RANGE 組込みプロシージャに関する基本的な説明は，『DEC Text Processing Utility Reference Manual』を参照してください。

SELECT_RANGE を実行したときに現在の編集位置が複数カラム文字の 2 カラム目以降にあった場合には，編集位置が 1 カラム目にあるときと同様に動作します。

シグナル・エラー

XTPU$_NOCURRENTBUF	WARNING	現在のバッファが設定されていない
XTPU$_NOSELECT	WARNING	現在のバッファで SELECT が実行されていない
XTPU$_SELRANGEZERO	WARNING	選択されたレンジに文字が入っていない
XTPU$_NEEDTOASSIGN	ERROR	SELECT_RANGE は代入文の右辺でのみ使用できる
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

SET

この組込みプロシージャは，DEC XTPU セッションの特定の機能を設定または変更するために使用されます。SET 組込みプロシージャでは最初のパラメータとしてキーワードを指定しなければなりません。キーワードはエディタのどの機能をセットするのかを指定します。セットできる機能はテキスト入力モード，スクリーンの特定の部分に表示されるテキスト，バッファの方向，バッファのステータスなどです。

形式

SET *(keyword, parameter [])*

引数

keyword
最初のパラメータとして使用されるキーワードは，設定または変更される機能を指定します。SET 組込みプロシージャに対して以下のキーワードを指定することができます。アスタリスク (*) がついたものは DEC XTPU で追加されたもので，後の節で説明されています。

ACTIVE_AREA
AUTO_REPEAT
BELL
CLIENT_MESSAGE
CODESET (*)
COLUMN_MOVE_VERTICAL
CROSS_WINDOW_BOUNDS
DEBUG
DEFAULT_DIRECTORY
DEFAULT_FILE
DETACHED_ACTION
DISPLAY_VALUE
DRM_HIERARCHY
ENABLE_RESIZE
EOB_TEXT
ERASE_UNMODIFIABLE
FACILITY_NAME
FILL_NOT_BEGIN (*)
FILL_NOT_END (*)

FILL_TRIM_SPACE (*)
FIRST_INPUT_ACTION
FORWARD
GLOBAL_SELECT
GLOBAL_SELECT_GRAB
GLOBAL_SELECT_READ
GLOBAL_SELECT_TIME
GLOBAL_SELECT_UNGRAB
HEIGHT
ICON_NAME
ICON_PIXMAP
INFORMATIONAL
INPUT_FOCUS
INPUT_FOCUS_GRAB
INPUT_FOCUS_UNGRAB
INSERT
JOURNALING
KEY_MAP_LIST
KEYSTROKE_RECOVERY
KEYBOARD_CODESET (*)
LEFT_MARGIN
LEFT_MARGIN_ACTION
LINE_NUMBER
MAPPED_WHEN_MANAGED
MARGINS
MARGIN_ALLOWANCE (*)
MAX_LINES
MENU_POSITION
MESSAGE_ACTION_LEVEL
MESSAGE_ACTION_TYPE
MESSAGE_CODESET (*)
MESSAGE_FLAGS
MODIFIABLE
MODIFIED
MOUSE
MOVE_VERTICAL_CONTEXT
NO_WRITE
OUTPUT_CODESET (*)
OUTPUT_FILE
OVERSTRIKE
PAD
PAD_OVERSTRUCK_TABS
PERMANENT
POST_KEY_PROCEDURE

PRE_KEY_PROCEDURE
PROMPT_AREA
RECORD_ATTRIBUTE
RECORD_MODE
RESIZE_ACTION
REVERSE
RIGHT_MARGIN
RIGHT_MARGIN_ACTION
SCREEN_LIMITS
SCREEN_UPDATE
SCROLL_BAR
SCROLL_BAR_AUTO_THUMB
SCROLLING
SELF_INSERT
SHIFT_KEY
SPECIAL_ERROR_SYMBOL
STATUS_LINE
SUCCESS
SYSTEM
TAB_STOPS
TEXT
TIMER
TRACEBACK
UID
UNDEFINED_KEY
VIDEO
VIDEO_CHARACTER_SET (*)
WIDGET
WIDGET_CALL_DATA
WIDGET_CALL_BACK
WIDGET_CONTEXT_HELP
WIDGET_RESOURCE_TYPES
WIDTH

これらの各キーワードとその後に指定されるパラメータについては，以降のページで詳しく説明します。キーワードはアルファベット順に説明されています。

parameter []
最初のパラメータの後に指定されるパラメータの数は，どのキーワードを使用したかによって異なります。パラメータはキーワードの説明の形式の部分に示されています。

説明

SET 組込みプロシージャは編集インタフェースを作成するプログラマ，およびその編集インタフェースを利用する人が使用できるように設計されています。プログラマは編集インタフェースの特定の省略時の動作や，スクリーン表示を設定することができます。また，ユーザは省略時の動作を変更し， SET 組込みプロシージャを使って既存の DEC XTPU セッションを簡単にカスタマイズすることができます。

SET (CODESET)

形式

SET *(CODESET, {SYSTEM | buffer}, keyword)*

引数

CODESET
システムまたはバッファのコードセットを指定するためのキーワード。

SYSTEM
システム・コードセットを設定することを示します。

buffer
コードセットを設定するバッファを指定します。

keyword
設定するコードセットを示すキーワード。指定できるキーワードは， DEC_MCS， ISO_LATIN1， ASCII_JISKANA， DECKANJI， DECKANJI2000， SDECKANJI， SJIS， ISO2022JP， UCS2，または UTF8 です。

説明

CREATE_BUFFER 組込みプロシージャで新しくバッファを作るときには，新しいバッファのコードセットおよび出力コードセットはシステム・コードセットと同じになります。

バッファのコードセットには 2 種類あり， SET (CODESET) で指定するコードセットは， READ_FILE 組込みプロシージャでファイルを読み込むときに使われます。ファイルに書き出すときに使用させれるコードセット (出力コードセット) は， SET (OUTPUT_CODESET) 組込みプロシージャで設定します。

SET (KEYBOARD_CODESET)， SET (MESSAGE_CODESET)， SET (OUTPUT_CODESET) も参照してください。

シグナル・エラー

XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる
XTPU$_INVPARAM	ERROR	引数の型が正しくない
XTPU$_BADKEY	ERROR	正しくないキーワードが指定された
XTPU$_ARGMISMATCH	ERROR	引数の型が正しくない

SET (FILL_NOT_BEGIN)

形式

SET *(FILL_NOT_BEGIN, string)*

引数

FILL_NOT_BEGIN
行頭禁則文字を指定するためのキーワード。

string
行頭禁則文字からなる文字列。

説明

FILL 組込みプロシージャを実行する時に使用する "行頭禁則文字 "を指定します。FILL は行頭禁則文字に指定された文字が行頭にこないように，行を分割する位置を決定します。

FILL， SET (FILL_NOT_END)， SET (MARGIN_ALLOWANCE) も参照してください。

シグナル・エラー

XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる
XTPU$_INVPARAM	ERROR	引数の型が正しくない

SET (FILL_NOT_END)

形式

SET *(FILL_NOT_END, string)*

引数

FILL_NOT_END
行末禁則文字を指定するためのキーワード。

string
行末禁則文字からなる文字列。

説明

FILL 組込みプロシージャを実行する時に使用する "行末禁則文字 "を指定します。FILL は行末禁則文字に指定された文字が行末にこないように，行を分割する位置を決定します。

FILL， SET (FILL_NOT_BEGIN)， SET (MARGIN_ALLOWANCE) も参照してください。

シグナル・エラー

XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる
XTPU$_INVPARAM	ERROR	引数の型が正しくない

SET (FILL_TRIM_SPACE)

形式

SET *(FILL_TRIM_SPACE, {ON | OFF | 1 | 0})*

引数

FILL_TRIM_SPACE
FILL 組込みプロシージャの実行によって行末にきた空白文字を，取り除くかどうかを制御することを示すキーワード。

ON **または** 1
FILL 組込みプロシージャの実行によって行末にきた空白文字を，取り除くことを指示します。

OFF **または** 0
FILL 組込みプロシージャの実行によって行末にきた空白文字を，取り除かないことを指示します。

説明

FILL， SET (FILL_NOT_BEGIN)， SET (FILL_NOT_END)， SET (MARGIN_ALLOWANCE) も参照してください。

シグナル・エラー

XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる
XTPU$_INVPARAM	ERROR	引数の型が正しくない
XTPU$_BADKEY	WARNING	正しくないキーワードが指定された
XTPU$_ONOROFF	ERROR	第 2 パラメータは ON， OFF， 1，0 のいずれかでなければならない

SET (KEYBOARD_CODESET)

形式

SET *(KEYBOARD_CODESET, keyword)*

引数

KEYBOARD_CODESET
キーボードのコードセットを指定するためのキーワード。

keyword
設定するキーボード・コードセットを示すキーワード。指定できるキーワードは，DEC_MCS， ISO_LATIN1， ASCII_JISKANA， DECKANJI， DECKANJI2000，SDECKANJI， SJIS， ISO2022JP， UCS2，または UTF8 です。

説明

SET (KEYBOARD_CODESET) はキーボード (端末) から送られるバイト列の解釈の方法を規定します。たとえば，端末から 177 というコードが送られたときに，キーボード・コードセットが ISO_LATIN1 に設定されていると半角の "± "という文字に解釈されますが，キーボード・コードセットが ASCII_JISKANA に設定されていると， JIS カタカナ文字セット (半角) の "ア "という文字に解釈されます。

UTF8 に出力した場合， UTF-8 端末を使用していると想定され，画面への出力コードセットも UTF-8 となります。この場合，エスケープシーケンスは 7 bit のコードが出力されます。

SET (CODESET)， SET (MESSAGE_CODESET)， SET (OUTPUT_CODESET) も参照してください。

シグナル・エラー

XTPU$_BADKEY	WARNING	正しくないキーワードが指定された
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる

XTPU$_INVPARAM	ERROR	引数の型が正しくない

SET (MARGIN_ALLOWANCE)

形式

SET *(MARGIN_ALLOWANCE, integer)*

引数

MARGIN_ALLOWANCE
右マージンを越えて置くことができる行頭禁則文字のカラム数を指定するためのキーワード。

integer
右マージンを越えて置くことができる行頭禁則文字のカラム数を指定する整数。

説明

組込みプロシージャ FILL は行頭禁則文字と行末禁則文字を考慮に入れて文字列を詰め込みます。 SET (MARGIN_ALLOWANCE) を使用して，行頭禁則文字が何カラムまで右マージンを越えた位置に存在できるかを指定できます。

初期値は 0 で，行頭禁則文字は右マージンを越えて存在できません。この場合は，行頭禁則文字が行の先頭にこないように，行頭禁則文字でない文字を伴って次の行に移されるので，右マージンまで埋まらない行ができることがあります。

FILL， SET (FILL_NOT_BEGIN)， SET (FILL_NOT_END) も参照してください。

シグナル・エラー

XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる
XTPU$_INVPARAM	ERROR	引数の型が正しくない
XTPU$_BADVALUE	ERROR	負の値が指定された

SET (MESSAGE_CODESET)

形式

SET *(MESSAGE_CODESET, keyword)*

引数

MESSAGE_CODESET
メッセージのコードセットを指定するためのキーワード。

keyword
設定するメッセージ・コードセットを示すキーワード。指定できるキーワードは, DEC_MCS, ISO_LATIN1, ASCII_JISKANA, DECKANJI, DECKANJI2000, SDECKANJI, SJIS, ISO2022JP, UCS2, または UTF8 です。

説明

SET (MESSAGE_CODESET) はメッセージ・ファイルに書かれた内容の解釈の方法を規定します。たとえばメッセージファイルに 177 というコードが書かれていたときに, メッセージ・コードセットが ISO_LATIN1 に設定されていると半角の "± "という文字に解釈されますが, メッセージ・コードセットが ASCII_JISKANA に設定されていると, JIS カタカナ文字セット (半角) の "ア "という文字に解釈されます。

SET (CODESET), SET (KEYBOARD_CODESET), SET (OUTPUT_CODESET) も参照してください。

シグナル・エラー

XTPU$_BADKEY	WARNING	正しくないキーワードが指定された
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる
XTPU$_INVPARAM	ERROR	引数の型が正しくない

SET (OUTPUT_CODESET)

形式

SET *(OUTPUT_CODESET, buffer, keyword)*

引数

OUTPUT_CODESET
バッファの出力コードセットを指定するためのキーワード。

buffer
出力コードセットを設定するバッファを指定します。

keyword
設定するコードセットを示すキーワード。指定できるキーワードは， DEC_MCS， ISO_LATIN1， ASCII_JISKANA， DECKANJI， DECKANJI2000， SDECKANJI， SJIS， ISO2022JP， UCS2，または UTF8 です。

説明

バッファのコードセットには 2 種類あり， SET (OUTPUT_CODESET) で指定するコードセットは WRITE_FILE 組込みプロシージャでファイルを書き出すときに使われます。ファイルに読み込むときに使われるコードセットは SET (CODESET) 組込みプロシージャで設定します。

SET (CODESET)， SET (KEYBOARD_CODESET)， SET (MESSAGE_CODESET) も参照してください。

シグナル・エラー

XTPU$_BADKEY	WARNING	正しくないキーワードが指定された
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる
XTPU$_INVPARAM	ERROR	引数の型が正しくない

SET (VIDEO_CHARACTER_SET)

形式

SET *(VIDEO_CHARACTER_SET, keyword, {ON|OFF|1|0})*

引数

VIDEO_CHARACTER_SET
端末に表示する文字セットを設定するキーワード。

keyword
有効または無効にする文字セットを示すキーワード。指定できるキーワードは，DEC_SUPPLEMENTAL，LATIN1_SUPPLEMENTAL，JIS_ROMAN，JIS_KATAKANA，KANJI_1，KANJI_3，KANJI_4，または KANJI_UDC です。

ON **または** 1
指定した文字セットを有効にすることを指定します。

OFF **または** 0
指定した文字セットを無効にすることを指定します。

説明

SET (VIDEO_CHARACTER_SET) 組込みプロシージャは，端末が持っている文字セットの情報を DEC XTPU に通知するために使われます。 DEC XTPU は，端末が持っている文字セットの情報を使って，端末が表示できない文字をダイアモンド・シンボルで表示する機能を持っています。

しかし DEC XTPU が端末が持っている文字セットの正しい情報を持っていないと文字化けを起こしたり，表示できるはずの文字がダイアモンド・シンボルになってしまいます。このような場合は SET (VIDEO_CHARACTER_SET) 組込みプロシージャで正しい情報を DEC XTPU に知らせることができます。

シグナル・エラー

XTPU$_BADKEY	WARNING	間違ったキーワードが指定された
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる
XTPU$_INVPARAM	ERROR	引数の型が正しくない

SPLIT_LINE

この組込みプロシージャは現在の行を現在の文字位置で区切り，2 つの行を作成します。

形式

SPLIT_LINE

引数

なし

説明

SPLIT_LINE 組込みプロシージャに関する基本的な説明は，『DEC Text Processing Utility Reference Manual』を参照してください。

現在の文字位置が 2 カラム文字の 2 カラム目にあるときに SPLIT_LINE を実行すると，1 カラム目にあるときと同様に行が分割されます。

シグナル・エラー

XTPU$_NOCURRENTBUF	WARNING	現在のバッファが設定されていない
XTPU$_NOTMODIFIABLE	WARNING	変更が禁止されているバッファの内容は変更できない
XTPU$_NOCACHE	ERROR	十分なメモリがない
XTPU$_TOOMANY	ERROR	パラメータの数が多すぎる

SYMBOL

この組込みプロシージャは，1 ～ 3 個の文字の組合せを DEC 漢字コード中の特殊文字の 1 つに変換してそれを通知します。また，区点コード変換および 16 進コード変換を行います。

形式

string2 := SYMBOL *(string1)*

引数

string1
特殊文字を生成するもととなる文字列。

説明

この組込みプロシージャは，ASCII 文字から DEC 漢字セットの特殊記号文字および罫線文字の変換およびコード変換を行います。これは日本語 OpenVMS かな漢字変換ライブラリを使っています。変換対応表については，『日本語ライブラリ 利用者の手引き』の付録 B を参照してください。

シグナル・エラー

XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる
XTPU$_INVPARAM	ERROR	引数の型が正しくない
XTPU$_NEEDTOASSIGN	ERROR	SYMBOL は代入文の右辺でのみ使用できる

例

このプロシージャは，現在の編集位置に ' ≠ ' を挿入します。

```
PROCEDURE user_symbol
  COPY_TEXT (SYMBOL('=/'));
ENDPROCEDURE;
```

VERIFY_BUFFER

この組込みプロシージャは，バッファの中に存在する指定したコードセットで表現できない文字を検索し，表現できない文字が連続する領域をレンジで戻します。該当する文字がないときには整数値 0 を戻します。

形式

{range | 0} := VERIFY_BUFFER *(keyword, {FORWARD | REVERSE})*

引数

keyword
コードセットを表すキーワード。指定できるキーワードは， DEC_MCS， ISO_LATIN1， ASCII_JISKANA， DECKANJI， DECKANJI2000， SDECKANJI， SJIS， ISO2022JP， UCS2，または UTF8 です。

FORWARD
現在の変終点からバッファの終わりの方向に検索することを指定するキーワード。

REVERSE
現在の変終点からバッファの先頭の方向に検索することを指定するキーワード。

説明

この組込みプロシージャは WRITE_FILE でファイルに書き込む前に，バッファ中の文字がそのままファイルに書き込めるかどうかを調べるために使います。 VERIFY_BUFFER で検索された文字列は，指定したコードセットで正しくファイルに書き込むことができません。戻される range には，最初に見つかった該当する文字が連続する領域のレンジが入ります。

シグナル・エラー

XTPU$_BADKEY	WARNING	間違ったキーワードが渡された
XTPU$_TOOFEW	ERROR	引数の数が少なすぎる
XTPU$_TOOMANY	ERROR	引数の数が多すぎる

XTPU$_INVPARAM	ERROR	引数の型が正しくない
XTPU$_NEEDTOASSIGN	ERROR	VERIFY_BUFFER 組込みプロシージャは代入文の右辺でのみ使用できる

5

DEC XTPU の起動

この章では DEC XTPU の起動法について説明します。特に DEC XTPU を起動するコマンドである EDIT/XTPU コマンドの使い方を詳しく説明しています。

5.1 概要

DEC コマンド言語 (DCL) レベルで下記のコマンドを入力すると， DEC XTPU が起動され, 編集インタフェースとして日本語 EVE が使用されます。

```
$ EDIT/XTPU
```

この章は，以下のような節から構成されています。

- DEC XTPU が回復不可能なエラーを起こさないために，第 5.2 節
- DCL コマンド・プロシージャから DEC XTPU を起動する場合，第 5.3 節
- バッチ・ジョブから DEC XTPU を起動する場合，第 5.4 節
- DEC XTPU コマンド・ラインの修飾子，第 5.5 節
- DEC XTPU で解釈されない修飾子が日本語 EVE によってどのように扱われているか，第 5.5 節
- EDIT/XTPU コマンドのパラメータ指定，第 5.6 節

5.2 仮想アドレス空間に関するエラーを避ける方法

DEC XTPU はデータをプロセスの仮想メモリ空間で扱います。もし， DEC XTPU のイメージ，データ構造，およびメモリ中に展開されたファイルに必要な空間が仮想アドレス空間を越えてしまった時には， DEC XTPU はデータの一部をワーク・ファイルに書き出し，開放されたメモリを必要な処理のために使用します。

ワーク・ファイルがいっぱいになると XTPU$_GETMEM または XTPU$_NOCACHE というエラー・メッセージが出力されます。不要なバッファを消去することでメモリを開放することもできますが，このようなエラーが発生したときには，処理を中断することが望まれます。再起動した後，必要なバッファのみを使用して処理を続けてください。ワーク・ファイルを十分な容量のあるディスクに置くことも可能です。ワーク・ファイルを置くディレクトリは，論理名 XTPU$WORK で指定してください。

プロセスに割り当てられる仮想アドレス空間を拡げることで，この回復不可能な内部エラーを避けることもできます。仮想アドレス空間は，以下の二つの要素によって決まります。

- SYSGEN パラメータの VIRTUALPAGECNT
- 使用しているアカウントのページ・ファイル・クォータ

VIRTUALPAGECNT はプロセスにマップされる仮想ページの数を制御します。VIRTUALPAGECNT に関する詳しい説明は，『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。

ページ・ファイル・クォータは，プロセスに割り当てられるシステム・ページ・ファイルの数を制御します。ページ・ファイル・クォータに関する詳しい説明は，『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』の/PGFLQUOTA 修飾子を参照してください。

仮想アドレス空間を拡げるためには， VIRTUALPAGECNT とページ・ファイル・クォータの両方を変える必要があるかもしれません。

5.3 DCL コマンド・プロシージャから DEC XTPU を起動する場合

DEC XTPU をコマンド・プロシージャから起動しなければならないのは，以下の 2 つの場合です。

- 対話方式の編集のための特殊な環境を設定するため
- 対話方式でない， DEC XTPU を使ったアプリケーションを実行するため

5.3.1 特殊な編集環境の設定

最初に必要とする環境を設定し，次に DEC XTPU を起動する DCL コマンド・プロシージャを作成すれば，特殊な編集環境で DEC XTPU を実行することができます。この場合， SYS$INPUT に SYS$COMMAND と同じ値を代入しなければなりません。

```
$ DEFINE/USER SYS$INPUT SYS$COMMAND
```

例 5–1 は最後に編集したファイルを“記憶”し，そのファイルを DEC XTPU に対する入力ファイルとして使用する DCL コマンド・プロシージャを示したものです。ファイル名を DCL のシンボル last_file_editid に記憶した後，この DCL コマンド・プロシージャは省略時のセクション・ファイル (別のセクション・ファイルを指定していないため) とユーザ作成コマンド・ファイルを使用して DEC XTPU を起動します。

例 5–1 DCL コマンド・プロシージャ filename.COM

```
$ IF P1 .NES. "" THEN last_file_edited == P1
$ WRITE SYS$OUTPUT "*** ''last_file_edited' ***"
$ DEFINE/USER SYS$INPUT SYS$COMMAND
$ EDIT/XTPU/COMMAND=DISK$:[USER]XTPUINI.TPU 'last_file_edited
```

例 5–2 は FORTRAN プログラムのためのタブ・ストップを指定する環境を設定するコマンド・プロシージャです。

例 5–2 DCL コマンド・プロシージャ fortran_ts.COM

```
$ IF P1 .EQS. " THEN GOTO REGULAR_INVOKE
$ last_file_edited == P1
$ FTN_TEST = F$FILE_ATTRIBUTES (last_file_edited,"RAT")
$ IF FTN_TEST .NES. "FTN" THEN GOTO REGULAR_INVOKE
$ FTN_INVOKE:
$     DEFINE/USER SYS$INPUT SYS$COMMAND
$     EDIT/XTPU/COMMAND=FTNTABS.TPU 'last_file_edited
$ GOTO TPU_DONE
$ REGULAR_INVOKE:
$     DEFINE/USER SYS$INPUT SYS$COMMAND
$     EDIT/XTPU 'last_file_edited
$ TPU_DONE:
```

5.3.2 対話方式でない編集環境の設定

一部の編集操作では編集コマンドを対話方式で入力するより，すべての編集コマンドをファイルに記憶し，ファイルからそれらのコマンドを読み取る方が便利なことがあります。また編集結果をスクリーンに表示せずに編集を実行したいこともあります。このような編集操作はバッチ・ジョブから実行することができ，また編集セッションの結果をスクリーンに表示したい場合には，このような編集操作を DCL コマンド・プロシージャから実行することもできます。実行中の編集操作がスクリーンに表示されない場合でも，コマンド・プロシージャが実行されている間，端末装置を自由に使用することはできません。

例 5–3 は下記の方法で DEC XTPU を起動するためのコマンド・ラインを含む DCL コマンド・プロシージャを示したものです。

コマンド・プロシージャのファイル名は invisible_tpu.COM です。

例 5–3 DCL プロシージャ invisible_tpu.COM

```
$ EDIT/XTPU/NOSECTION/COMMAND=gsr.TPU/NODISPLAY/NOKANJI_DICTIONARY 'p1'
```

/NOSECTION	セクション・ファイルは使用しません (エディタを初期化するためにバイナリ・セクション・ファイルのプロシージャやキー定義は使用されません)。

/COMMAND=gsr.TPU	実行したい編集操作はコマンド・ファイルに含まれています。
/NODISPLAY	スクリーン管理機能 (ウィンドウ，カーソルなど) は認識されません。
/NOKANJI_DICTIONARY	かな漢字変換辞書を使用しません。

gsr.TPU というファイルは /COMMAND 修飾子に対するファイル仕様として使用されています。これは現在のバッファを探索し，文字列またはパターンを文字列と置き換えます。例 5–4 は gsr.TPU というファイルを示したものです。

例 5–4 DEC XTPU コマンド・ファイル gsr.TPU

```
PROCEDURE global_search_replace (str_or_pat, str2)

! This procedure performs a search through the current
! buffer and replaces a string or a pattern with a new string

LOCAL src_range, replacement_count;

! Return to caller if string not found
ON_ERROR
   msg_text := FAO ('Completed !UL  replacement!%S',
                                        replacement_count);
   MESSAGE (msg_text);
   RETURN;
ENDON_ERROR;

replacement_count := 0;

LOOP
   src_range := SEARCH (str_or_pat, FORWARD);
                              ! Search returns a range if found
   ERASE (src_range);         ! Remove first string
   POSITION (END_OF (src_range));! Move to right place
   COPY_TEXT(str2);           ! Replace with second string
   replacement_count := replacement_count + 1;
ENDLOOP;
ENDPROCEDURE                  ! global_search_replace

! Executable statements
input_file:=GET_INFO (COMMAND_LINE,'file_name');
main_buffer:=CREATE_BUFFER ('main', input_file);
POSITION (BEGINNING_OF (main_buffer));
global_search_replace ('xyz$_', 'user$_');
pat1:= '' & LINE_BEGIN & 't';
POSITION (BEGINNING_OF (main_buffer));
global_search_replace (pat1, 'T');
WRITE_FILE (main_buffer, 'newfile.dat');
QUIT;
```

注意

/NODISPLAY を EDIT/XTPU コマンド・ラインに指定した場合には，キーを定義することはできません。また編集操作を記憶しているファイルでウィンドウを作成したり操作することもできません。

/NOKANJI_DICTIONARY を EDIT/XTPU コマンド・ラインに指定しなかった時には，その編集操作に対して変換用個人辞書が割り当てられます。このため，バッチ・モードでかな漢字変換機能を使用しない時には，/NOKANJI_DICTIONARY を指定しておくことをお勧めします。

QUIT 組込みプロシージャまたは EXIT 組込みプロシージャを使用してエディタを終了する時には，変更したバッファを出力しなければなりません (この操作は例 5–4 に示されています)。バッファの内容を変更したのにそのバッファを出力しないと，その変更された内容は失われてしまいます。

5.4 バッチ・ジョブから DEC XTPU を起動する場合

端末装置からではなく，バッチ・ジョブで編集操作を実行したい場合には， DCL の SUBMIT コマンドを使用してジョブをバッチ・キューに送ることができます。

たとえば，前の節と同じ編集をバッチ・モードで実行したい場合には，以下のコマンドを入力することができます。

```
$ SUBMIT invisible_tpu.COM/LOG=invisible_tpu.LOG/parameter=my_file.txt
```

このジョブはシステムの省略時のバッチ・キューに入力され，結果はバッチ・ジョブが作成する LOG ファイルに出力されます。

バッチのようなコマンド・プロシージャに適用される制約条件は，バッチ・ジョブにも適用されます。/NODISPLAY をコマンド・ラインで使用するときと，編集操作を含むファイルで EXIT 組込みプロシージャまたは QUIT 組込みプロシージャを使用するときの注意事項について，上記の節を参照してください。

5.5 DEC XTPU コマンド・ラインの修飾子

EDIT/XTPU コマンドに修飾子を指定することによって， DEC XTPU の属性を設定したり， DEC XTPU 上に構築されたアプリケーションの属性を設定したりすることができます。修飾子は次の 2 種類に分類できます。

- DEC XTPU によって使われる修飾子。ここに分類される修飾子は， DEC XTPU によって省略時の値が定義されています。
- DEC XTPU で書かれたアプリケーションによって使われる修飾子。省略時の値が， DEC XTPU によって定義されているもの，アプリケーションによって定義されるもの，両方によって決まるものがあります。

表 5–1 は EDIT/XTPU コマンドの修飾子と，それらの省略時の値を定義する部分，それらの修飾子を解釈する部分を示しています。

表 5–1 EDIT/XTPU コマンドの修飾子

修飾子	省略時の値の定義	修飾子の扱い
/CODESET=*keyword*	DEC XTPU	DEC XTPU
/[NO]COMMAND[=*filespec*]	DEC XTPU	DEC XTPU
/[NO]CREATE	両方	アプリケーション
/[NO]DEBUG[=*filespec*]	DEC XTPU	DEC XTPU
/[NO]DISPLAY[=*keyword*]	DEC XTPU	DEC XTPU
/[NO]INITIALIZATION[=*filespec*]	両方	アプリケーション
/INTERFACE[=*keyword*]	DEC XTPU	DEC XTPU
/[NO]JOURNAL[=*filespec*]	両方	アプリケーション
/[NO]KANJI_DICTIONARY[=*filespec*]	DEC XTPU	DEC XTPU
/[NO]MODIFY	アプリケーション	アプリケーション
/[NO]OUTPUT[=*filespec*]	両方	アプリケーション
/[NO]READ_ONLY	両方	アプリケーション
/[NO]RECOVER	DEC XTPU	DEC XTPU
/[NO]SECTION[=*filespec*]	DEC XTPU	DEC XTPU
/START_POSITION[=(*line*,*column*)]	DEC XTPU	アプリケーション
/[NO]WORK[=*filespec*]	DEC XTPU	DEC XTPU
/[NO]WRITE	両方	アプリケーション

以降の節では，修飾子について詳しく説明します。以降の例の中では，修飾子は EDIT/XTPU コマンドのすぐ後，入力ファイル仕様の前に指定されていますが，修飾子は EDIT/XTPU コマンドの後なら，コマンド・ラインのどこに指定してもかまいません。以降の節は，インタフェースとして日本語 EVE 使用されていることを想定して書かれています。したがって，アプリケーションが扱いを指定できる修飾子も日本語 EVE をもとにして書かれています。日本語 EVE をベースにしないアプリケーションは，それらの修飾子を違った方法で使用できます。

5.5.1 /CODESET

/CODESET=*keyword*
/CODESET=ISO_LATIN1 (省略時設定)

DEC XTPU の起動時のシステム・コードセットを指定します。システム・コードセットは新しくバッファを作る時に使用されますので，組込みプロシージャでシステム・コードセットを変更しなければ，作成されるバッファのコードセットおよび出力コードセットの初期値は/CODESET 修飾子で指定した値になります。

/CODESET 修飾子のパラメータに指定できるキーワードは以下の通りです。

表 5–2 /CODESET 修飾子のパラメータ

キーワード	コードセット
DECKANJI	DEC 漢字コードセット
SDECKANJI	Super DEC 漢字コードセット
ISO_LATIN1	ISO Latin1 コードセット
DEC_MCS	DEC MCS コードセット
ASCII_JISKANA	ASCII (JIS ローマ字) + JIS カタカナコードセット
SJIS	シフト JIS コードセット
DECKANJI2000	DEC 漢字 2000 コードセット
ISO2022JP	ISO-2022-JP コードセット
UCS2	UCS-2 コードセット
UTF8	UTF-8 コードセット

/CODESET 修飾子を指定しなかった場合には，以下のような順序でシステム・コードセットが決められます。

1. 論理名 LANG の定義

   論理名 LANG に定義できるものは以下の通りです。大文字小文字の区別はありません。論理名 LANG はプロセスのコードセットの定義を行う場合に使用します。システム全体のコードセットの定義を行う時には論理名 XPG$DEFAULT_LANG を使用してください。

表 5–3 論理名 LANG の定義

値	コードセット
ja_JP.deckanji	DEC 漢字コードセット
ja_JP.sdeckanji	Super DEC 漢字コードセット
en_US.ISO8859-1	ISO Latin1 コードセット
ja_JP.sjis	シフト JIS コードセット
ja_JP.deckanji2000	DEC 漢字 2000 コードセット
ja_JP.UTF-8	UTF-8 コードセット

2. 論理名 XPG$DEFAULT_LANG の定義

   論理名 XPG$DEFAULT_LANG に定義できるものは以下の通りです。大文字小文字の区別はありません。論理名 XPG$DEFAULT_LANG はシステム全体のコードセットの定義を行う場合に使用します。プロセスごとにコードセットを定義するときには論理名 LANG を使用してください。

表 5–4 論理名 XPG$DEFAULT_LANG の定義

値	コードセット
ja_JP.deckanji	DEC 漢字コードセット
ja_JP.sdeckanji	Super DEC 漢字コードセット
en_US.ISO8859-1	ISO Latin1 コードセット
ja_JP.sjis	シフト JIS コードセット
ja_JP.deckanji2000	DEC 漢字 2000 コードセット
ja_JP.UTF-8	UTF-8 コードセット

3. 省略値の使用

   上記の論理名が定義されていない時にはコードセットの省略値が使用されます。省略値は論理名 XTPU$TEXT_LANGUAGE の値によってきまります。XTPU$TEXT_LANGUAGE が "JAPANESE"と定義されているときは DEC 漢字コードセットがシステム・コードセットになります。それ以外のときには ISO Latin1 コードセットがシステム・コードセットになります。

日本語 OpenVMS では以下のようにシステム論理名が定義されていますので, /CODESET 修飾子を指定しなかった場合には通常 DEC 漢字コードセットがシステム・コードセットになります。

- XTPU$TEXT_LANGUAGE = "JAPANESE"

コマンド・ラインで指定されたコードセットの名前を知るには以下のようにしてください。変数 X にはコードセットを示すキーワードが戻ります。

```
X := GET_INFO (COMMAND_LINE, "codeset");
```

次のコマンドを実行すると, DEC XTPU は起動時にシステム・コードセットを Super DEC 漢字コードセットに設定し, 編集セッションのための入力ファイルとして letter.txt を使用します。

```
$ EDIT/XTPU/CODESET=sdeckanji letter.txt
```

5.5.2 /COMMAND

```
/COMMAND[=filespec]
/NOCOMMAND
/COMMAND=XTPU$COMMAND ( 省略時設定 )
```

DEC XTPU が初期化のために, ユーザ作成コマンド・ファイルを読み取るかどうかを指定します。コマンド・ファイルは, DEC XTPU 上に作られたアプリケーションの拡張および変更や, 新しいアプリケーションを作るために使用されます。DEC XTPU コマンド・ファイルの省略時のファイル・タイプは.TPU です。ファイル指定にワイルド・カードを使うことはできません。特に指定しなかった場合, 省略時の値によって DEC XTPU は, 省略時のディレクトリの XTPU$COMMAND.TPU という

コマンド・ファイルを読み取ろうとします。省略時のコマンド・ファイル以外のコマンド・ファイルを使用する場合には，/COMMAND 修飾子の後に完全なファイル仕様を指定するか，または XTPU$COMMAND という論理名を定義してください。

ユーザが DCL コマンド・ラインで/COMMAND を指定したかどうかは，アプリケーションの中で以下のようにして知ることができます。

```
x := GET_INFO (COMMAND_LINE, "command");
```

/COMMAND が指定された時には 1，指定されなかった時には 0 が返されます。コマンド・ラインで指定されたコマンド・ファイルの名前を知るには以下のようにしてください。

```
x := GET_INFO (COMMAND_LINE, "command_file");
```

以下のコマンドを実行すると， DEC XTPU は sys$login:my_tpu$command.TPU という名前のコマンド・ファイルを読み取り，編集セッションのための入力ファイルとして， letter.rno を使用します。

```
$ EDIT/XTPU/COMMAND=sys$login:my_tpu$command.TPU letter.rno
```

DEC XTPU がコマンド・ファイルを処理しないようにするには，/NOCOMMAND 修飾子を使用します。コマンド・ファイルを使用せずに DEC XTPU を起動することが多いときには，以下のようなシンボルを定義しておくと良いでしょう。

```
$ JEVE == "EDIT/XTPU/NOCOMMAND"
```

/NOCOMMAND 修飾子を使用してコマンド・ファイルを使用しないと，コマンド・ファイルの探索が不要になるので，始動時間を短縮することができます。

指定したコマンド・ファイルが存在しないときには，編集を継続せずに DCL に戻ります。

5.5.3 /CREATE

```
/CREATE ( 省略時設定 )
/NOCREATE
```

指定された入力ファイルが存在しないときに， DEC XTPU が新しいファイルを作成するかどうかを指定します。/CREATE と/NOCREATE が両方とも指定されなかったときには， DEC XTPU は/CREATE と解釈しますが，ファイル名は指定されません。

DEC XTPU の上にレイヤ構造となっているインタフェースが，この修飾子を処理します。インタフェースは以下に示す GET_INFO 組込みプロシージャを使用することによって， DEC XTPU の起動時にこの修飾子が指定されていたかどうかを知ることができます。

```
x := GET_INFO (COMMAND_LINE, "create")
```

この値が 1 のときは，/CREATE が指定されていたことを， 0 のときは/NOCREATE が指定されていたことを示します。

日本語 EVE は，省略時には入力ファイルが存在しないときにファイルを作成します。/NOCREATE を指定して，入力ファイルに存在しないファイルを指定すると，編集を中断して， DCL コマンド・レベルに戻ります。

たとえば，現在のデバイスとディレクトリが DISK$:[USER]のときに NEWFILE.DAT という存在しない入力ファイルを指定したときには， DEC XTPU インタフェースはエラー・メッセージをプリントし，下記のように DCL コマンド・レベルに戻ります。

```
$ EDIT/XTPU/NOCREATE newfile.dat
入力ファイルがありません: NEWFILE.DAT
$
```

5.5.4 /DEBUG

```
/DEBUG[=filespec]
/NODEBUG ( 省略時設定 )
```

修飾子/DEBUG は， DEC XTPU デバッガを実装するファイルを実行するかどうかを指定します。/DEBUG が指定されると， XTPU は XTPU$INIT_PROCEDURE プロシージャやコマンド・ファイルを実行する前に，デバッガ・ファイルの内容を読み込んでコンパイルし，実行します。

省略時の設定では，デバッガ・ファイルは読み込まれません。デバッガ・ファイルを指定せずに/DEBUG を指定すると， SYS$SHARE:XTPU$DEBUG.TPU が読み込まれます。

省略時のデバッガ・ファイルの代わりに別のデバッガ・ファイルを使用するには，/DEBUG 修飾子にそんデバッガ・ファイルのファイル指定を与えます。装置名およびディレクトリを省略すると， SYS$SHARE が仮定されます。論理名 XTPU$DEBUG を定義した場合，ファイルを指定せずに/DEBUG を指定すれば，その論理名に指定されているファイルが読み込まれます。

5.5.5 /DISPLAY

```
/DISPLAY=CHARACTER_CELL ( 省略時設定 )
/DISPLAY=DECWINDOWS
/NODISPLAY
```

DECwindows 版と文字セル版の DEC XTPU を選択するには，DEC XTPU を起動するときに DCL コマンド・ラインの/DISPLAY 修飾子で行います。/DISPLAY コマンド修飾子はオプションです。省略時には，DEC XTPU をワークステーション上で使うか端末上で使うかに関わらず，/DISPLAY=CHARACTER_CELL が使われます。/DISPLAY=CHARACTER_CELL を指定したときには，DEC XTPU は文字セル端末用のスクリーン・マネージャを使用します。文字セル端末用のスクリーン・マネージャは日本語 DECterm 端末エミュレータあるいはハードウェア端末上で動作します。

/DISPLAY=DECWINDOWS を指定し，DECwindows の環境を使うことができるときには，DEC XTPU は DECwindows 用のスクリーン・マネージャを使用して，DEC XTPU のために DECwindows のウィンドウを作成します。/DISPLAY=DECWINDOWS を指定したときに，DECwindows の環境を使うことができない場合には，DEC XTPU は文字セル端末用のスクリーン・マネージャを使用します。

/NODISPLAY 修飾子を使用すると，DEC XTPU は端末装置のスクリーン表示機能やキーボード機能を使用せずに動作します。次の場合には，/NODISPLAY 修飾子を使用しなければなりません。

- バッチ・ジョブで DEC XTPU プロシージャを実行する場合
- サポートされない端末装置で DEC XTPU を使用する場合

/NODISPLAY を使用する場合には，DEC XTPU のウィンドウ操作コマンドやスクリーン操作コマンドを実行するとエラーが発生します。また/NODISPLAY が指定されているときでも，スクリーン操作コマンド (ADJUST_WINDOW，CREATE_WINDOW，MAP) およびキー定義を含む初期化ファイルは実行することができます。これらのコマンドは意味のないもので，バッチ・ログ・ファイルまたはスクリーンにエラー・メッセージが通知されることもあります。したがって，/NODISPLAY 修飾子を使用するセッションでは，特殊な初期化ファイル (セクション・ファイルまたはコマンド・ファイル) を使用してください。このファイルにはスクリーン操作コマンド (READ_LINE，MESSAGE，および LAST_KEY は除きます。これらはいくつかの制限はありますが正しく機能します) やキー定義を含んでいけません。さらにファイルは，完全な DEC XTPU セッションである必要があります。つまり，EXIT コマンドまたは QUIT コマンドで終了していなければなりません。

以下のコマンドを実行すると，DEC XTPU は端末装置を使用せずに my_batch_file.rno というファイルを編集します。

```
$ EDIT/XTPU/NODISPLAY my_batch_file.rno
```

5.5.6 /INITIALIZATION

```
/INITIALIZATION[=filespec] ( 省略時設定 )
/NOINITIALIZATION
```

DEC XTPU を使用して書かれたアプリケーションがイニシャライゼーション・ファイルを実行するかどうかを指定します。この修飾子の使い方は，アプリケーションによって異なります。

アプリケーションは，次のようにしてユーザが DCL コマンド・ラインで /INITIALIZATION を指定したかどうかを調べます。

```
x := GET_INFO (COMMAND_LINE, "initialization");
```

この値が 1 のときは，/INITIALIZATION が指定されていたことを， 0 のときには，指定されていなかったことを示します。コマンド・ラインで指定されたファイル名を知るためには次のようにします。

```
x := GET_INFO (COMMAND_LINE, "initialization_file");
```

詳しくは，『Guide to the DEC Text Processing Utility』を参照してください。

DCL コマンド・ラインで/[NO]INITIALIZATION を指定しなかったときには， DEC XTPU は/INITIALIZATION が指定されていたものとして処理を行います。ただし，省略時のファイル名はありません。省略時のファイル名は，アプリケーションによって指定されます。ユーザがアプリケーションを書くときには，次のようなフォーマットのファイルを省略時のファイルとすることをお勧めします。

```
facility$init.facility
```

日本語 EVE の省略時のイニシャライゼーション・ファイルは， JEVE$INIT_V3.EVE です。

日本語 EVE においてデバイスやディレクトリを指定しなかったときには，日本語 EVE はまず最初に現在のディレクトリを捜します。指定された (またはデフォルトの) イニシャライゼーション・ファイルがそこにないときには，日本語 EVE は SYS$LOGIN を捜します。イニシャライゼーション・ファイルが見つかれば，日本語 EVE はそのファイル中のコマンドを実行します。

日本語 EVE でのイニシャライゼーション・ファイルの使い方について，詳しくは，『日本語 EVE リファレンス・マニュアル』の "日本語 EVE のカスタマイズ "を参照してください。

5.5.7 /INTERFACE

```
/INTERFACE[ =keyword ]
/INTERFACE=CHARACTER_CELL ( 省略時設定 )
```

修飾子/INTERFACE は，インタフェースすなわち画面表示のタイプを指定します（修飾子/DISPLAY と同様）。省略時の設定は CHARACTER_CELL です。

たとえば，日本語 EVE を DECwindows インタフェースで起動するには，次のコマンドを使用します。

```
$ EDIT/XTPU/INTERFACE=DECWINDOWS
```

このとき， DECwindows が利用可能であれば， XTPU はワークステーション画面に別のウィンドウを作成して編集セッションを開始します。この場合，日本語 EVE の画面レイアウトに含まれているメニュー・バーやスクロール・バーなどの DECwindows 機能が利用できます。 DECwindows が利用できなければ， XTPU は文字セル端末インタフェースを使用します。

5.5.8 /JOURNAL

```
/JOURNAL=input_file.TJL
/NOJOURNAL ( 省略時設定 )
```

割り込まれたセッションから回復できるように， DEC XTPU が編集セッションのジャーナル・ファイルを保存するかどうかを指定します。 DEC XTPU は 2 種類のジャーナリングをサポートします。

- キー・ジャーナリング — どのバッファを使用しているかに関わらず， 1 つのジャーナル・ファイルにキー入力を保存します。
- バッファ・ジャーナリング — バッファの内容の変化をバッファごとのジャーナル・ファイルに保存します。

この修飾子の処理は DEC XTPU 上に作られたアプリケーションの責任です。

アプリケーションは，次のようにしてユーザが DCL コマンド・ラインで /JOURNAL を指定したかどうかを調べます。

```
x := GET_INFO (COMMAND_LINE, "journal");
```

この値が 1 のときは，/JOURNAL が指定されていたことを， 0 のときには，指定されていなかったことを示します。

あるバッファにたいしてバッファ・ジャーナリングが起動されているかどうかを知るためには次のようにします。

```
status := GET_INFO (buffer_name, "journaling");
```

コマンド・ラインで指定されたキー・ジャーナリングのファイル名を知るためには次のようにします。

```
x := GET_INFO (COMMAND_LINE, "journal_file");
```

詳しくは，『Guide to the DEC Text Processing Utility』を参照してください。

日本語 EVE では，/JOURNAL 修飾子を指定しないか，ジャーナル・ファイル名を付けずに/JOURNAL を指定したときには，バッファ・ジャーナリングが使用されます。バッファ・ジャーナリングの，ジャーナル・ファイルの省略時のファイル・タイプは.XTPU$JOURNAL です。

/JOURNAL=*filespec* というようにジャーナル・ファイル名を指定すると，キー・ジャーナリングも同時に使用されます。キー・ジャーナリングの，ジャーナル・ファイルの省略時のファイル・タイプは.TJL です。

日本語 EVE で，キー・ジャーナリング，バッファ・ジャーナリングの両方を使わないときには，修飾子に/NOJOURNAL を指定してください。たとえば，以下のコマンドは日本語 EVE に対してジャーナリングをしないで MEMO.TXT をいうファイルを編集することを指示します。

```
$ EDIT/XTPU/NOJOURNAL memo.txt
```

DEC XTPU 上にアプリケーションを作る時には， JOURNAL_OPEN 組込みプロシージャを使用して編集セッションのキー・ジャーナル・ファイルを作成します。JOURNAL_OPEN によって DEC XTPU はキーをジャーナリングするために 500 バイトのバッファを確保します。通常 DEC XTPU はそのバッファがいっぱいになったときにバッファの内容をジャーナル・ファイルに書き出します。

/NOJOURNAL を指定してバッファ・ジャーナリングを起動させていないときでも， SET (JOURNALING) 組込みプロシージャを使用して起動できます。SET (JOURNALING) はジャーナリングの頻度を設定する際にも使用されます。JOURNAL_OPEN および SET (JOURNALING) について詳しくは『Guide to the DEC Text Processing Utility』 を参照してください。

キー・ジャーナリングでジャーナル・ファイルを作成した後， DEC XTPU がジャーナル・ファイルに含まれるコマンドを処理するようにするには，/RECOVER 修飾子を使用します。下記のコマンドを実行すると， DEC XTPU は MEMO.TXT という名前の入力ファイルを編集した編集セッションをジャーナル・ファイル MEMO.TJL を使用して回復します。キー・ジャーナリングのジャーナル・ファイルを用いて回復を行う場合には，コマンド行に，/JOURNAL=*filespec* と/RECOVER 修飾子の両方を指定しなければなりません。

```
$ EDIT/XTPU/RECOVER/JOURNAL=memo.tjl memo.txt
```

バッファ・ジャーナリングでバッファの変更を回復するには RECOVER_BUFFER 組込みプロシージャを使用してください。 RECOVER_BUFFER について詳しくは『DEC Text Processing Utility Reference Manual』を参照してください。

中断された日本語 EVE セッションを回復する方法について詳しくは『日本語 EVE ユーザーズ・ガイド』の "システム割り込みからの回復 "を参照してください。

5.5.9 /KANJI_DICTIONARY

```
/KANJI_DICTIONARY[=kanji_dictionary_filename]
/NOKANJI_DICTIONARY
/KANJI_DICTIONARY=SYS$LOGIN:JSYKOJIN.JISHO ( 省略時設定 )
```

DEC XTPU がかな漢字変換用の個人辞書を使用するかどうかを指定します。指定されないときには， DEC XTPU の省略時の値によって JSY$KOJIN という論理名で示されるファイルを個人辞書として使用しようとします。このときに， JSY$KOJIN の論理名にファイルが割り当てられていないと， SYS$LOGIN:JSYKOJIN.JISHO が個人辞書として使用されます。個人辞書として使用されるファイルは，日本語 OpenVMS の個人辞書ファイルでなければなりません。

/KANJI_DICTIONARY=*filespec* 修飾子に対する値を指定すれば，個人辞書として使用されるファイルの名前を指定することができます。この結果，省略時の個人辞書である SYS$LOGIN:JSYKOJIN.JISHO は無効になります。

下記のコマンドを実行すると， DEC XTPU は personal.jisho というファイルを個人辞書として使用します。

```
$ EDIT/XTPU/KANJI_DICTIONARY=personal.jisho
```

個人辞書として指定されたファイルが存在しないときには， DEC XTPU は個人辞書を指定されたファイル名で作成して使用します。

/NOKANJI_DICTIONARY を指定したときには， DEC XTPU は個人辞書を使用しません。したがって CONVERT_KANA， ENTER_TANGO， DELETE_TANGO などの組込みプロシージャを実行することはできません。これらの組込みプロシージャを実行するには個人辞書が必要で，エラー・メッセージが表示されます。

DEC XTPU をバッチ・モードで使用するときのように，かな漢字変換を必要としないときには/NOKANJI_DICTIONARY 修飾子を指定すると良いでしょう。

```
$ EDIT/XTPU/NODISPLAY/NOKANJI_DICTIONARY/NOSECTION/COMMAND=format.tpu
```

個人辞書の利用方法は，論理名 JSY$KOJIN_MODE によって指定することができます。

1. JSY$KOJIN_MODE="0" (個人辞書共有モード)

個人辞書を用いて変換を行います。個人辞書への学習を行い，単語の登録および削除もできます。辞書を共有することができます。

2. JSY$KOJIN_MODE="1" (個人辞書学習モード)

   個人辞書を用いて変換を行います。個人辞書への学習を行い，単語の登録および削除もできます。辞書を共有することはできません。

3. JSY$KOJIN_MODE="2" (個人辞書参照モード)

   個人辞書を用いて変換を行います。ただし，個人辞書への学習は行われずに，単語の登録および削除もできません。個人辞書参照モードどうしなら，個人辞書を共有することができます。

4. JSY$KOJIN_MODE="3"

   個人辞書は使用されません。

JSY$KOJIN_MODE が論理名として定義されていないときの省略値は， "1."の個人辞書共有モードです。

5.5.10 /MODIFY

```
/MODIFY ( 省略時設定 )
/NOMODIFY
```

編集セッションの最初のユーザ・バッファが変更可能かどうかを指定します。 DEC XTPU を使用して書かれたアプリケーションが，/MODIFY 修飾子を処理します。

/MODIFY 修飾子がどのように使用されたかを知るためには次のようにします。

```
x := GET_INFO (COMMAND_LINE, "modify")
x := GET_INFO (COMMAND_LINE, "nomodify")
```

最初のステートメントの値が 1 のときには/MODIFY が明示的に指定されたことを示します。 2 番目のステートメントの値が 1 のときには/NOMODIFY が明示的に指定されたことを示します。両方のステートメントの値が 0 のときには，アプリケーションは省略時の動作をしなければなりません (省略時の動作はアプリケーションによって異なります)。

日本語 EVE を使うときに，/MODIFY，/NOMODIFY，/READ_ONLY，/NOWRITE のいづれも指定しなければ，日本語 EVE は編集セッションのすべてのバッファを変更可能にします。/NOMODIFY を指定すれば，すべてのユーザ・バッファは変更できなくなります。

日本語 EVE で，/MODIFY も/NOMODIFY も指定しなかったときには，/READ_ONLY あるいは，/WRITE 修飾子が指定されていたかどうかが調べられます。/READ_ONLY と/MODIFY，あるいは/NOWRITE と/MODIFY を同時に指定するとバッファは変更可能になります。同様に，/WRITE と/NOMODIFY，あるい

は/NOREAD_ONLY と/NOMODIFY を同時に指定するとバッファは変更不可能になります。

5.5.11 /OUTPUT

```
/OUTPUT=input_file.type ( 省略時設定 )
/NOOUTPUT
```

DEC XTPU セッションの出力がファイルに書き込まれるかどうかを指定します。DEC XTPU の上にレイヤ構造となっているインタフェースは，この修飾子を処理しなければなりません。インタフェースは以下に示す GET_INFO 組込みプロシージャを使用して， DEC XTPU の起動時にこの修飾子が指定されていたかどうかを知ることができます。

```
x := GET_INFO (COMMAND_LINE, "output")
```

日本語 EVE インタフェースでは，/OUTPUT を使用することにより， DEC XTPU を終了するときにメイン・バッファから作成されるファイルの名前を指定することができます。

インタフェースとして日本語 EVE を使用しているときに以下のコマンドを実行すると， DEC XTPU を終了するときに， DEC XTPU はメイン・バッファの内容を newlet.rno というファイルに出力します。

```
$ EDIT/XTPU/OUTPUT=newlet.rno letter.rno
```

出力ファイルの省略時の名前は入力ファイルの名前と同じで，バージョン番号は入力ファイルの既存のバージョンより 1 だけ大きな値になります。/OUTPUT 修飾子に対してファイル指定をすれば，出力ファイルに別の名前を付けることができます。

/NOOUTPUT 修飾子は， DEC XTPU 上に，ファイルに出力を書き込まないようなインタフェースを作成するときに使用することができます。たとえば，コマンド・ラインに/NOOUTPUT を指定すると， DEC XTPU がメイン・バッファに対して NO_WRITE 属性を設定し，そのバッファに対する出力ファイルは作成しないというようなアプリケーションを作ることができます。

5.5.12 /READ_ONLY

```
/READ_ONLY
/NOREAD_ONLY ( 省略時設定 )
```

DEC XTPU がジャーナル・ファイルを保存し，メイン・バッファを変更したときに，メイン・バッファの内容から出力ファイルを作成するかどうかを指定します。

/READ_ONLY 修飾子の処理は，/WRITE 修飾子と関係があります。/READ_ONLY は/NOWRITE を同じです。また/NOREAD_ONLY は，/WRITE と同じです。 DCL コマンドで次のように修飾子が組み合せられたときには， DEC XTPU は，エラーを出して終了します。

- /READ_ONLY と/WRITE
- /NOREAD_ONLY と/NOWRITE

DEC XTPU 上に作られたアプリケーションは，この修飾子を処理しなければなりません。

DCL コマンド・ライン上で修飾子/READ_ONLY ないし/NOWRITE が指定されたかどうかを判断するには，次の文を使用します。

```
x := GET_INFO (COMMAND_LINE, "read_only");
```

この文は，/READ_ONLY または/NOWRITE が明示的に指定されていれば値 1 を返します。

DCL コマンド・ライン上で修飾子/NOREAD_ONLY ないし/WRITE が指定されたかどうかを判断するには，次の文を使用します。

```
x := GET_INFO (COMMAND_LINE, "write");
```

この文は，/NOREAD_ONLY または/WRITE が明示的に指定されていれば値 1 を返します。

日本語 EVE では，/READ_ONLY 修飾子は，/NOMODIFY 修飾子，および /NOOUTPUT 修飾子のすべてを使用した場合と同じ結果になります。/READ_ONLY を指定すると， DEC XTPU はすべてのユーザ・バッファに対して NO_WRITE 属性と NO_MODIFY 属性をセットします。バッファが NO_WRITE にセットされている場合には， DEC XTPU を終了するときに，バッファの内容がファイルに書き込まれません。 EXIT 組込みプロシージャと QUIT 組込みプロシージャはどちらも，メイン・バッファの内容から新しいファイルを作成せずに編集セッションを終了します。

たとえば，次のコマンドを実行すると， DEC XTPU はエディタを終了するときに新しいファイルを作成しません。

```
$ EDIT/XTPU/READ_ONLY meeting.mem
```

/NOREAD_ONLY は， EXIT コマンドが実行されるときにメイン・バッファの内容が変更されていれば，メイン・バッファをファイルに書き込むように DEC XTPU に指示を与えます。これは省略時の処理です。

5.5.13 /RECOVER

```
/RECOVER
/NORECOVER ( 省略時設定 )
```

DEC XTPU が起動時にジャーナル・ファイルを読み込んで，前に中断された編集セッションを回復するかどうかを指定します。たとえば，次のコマンドを実行すると，DEC XTPU は notes.txt というファイルに対する前の編集セッションを回復します。

```
$ EDIT/XTPU/RECOVER notes.txt
```

DCL コマンド・ラインで/RECOVER が指定されたかどうかは，以下のようにして知ることができます。

```
x := GET_INFO (COMMAND_LINE, "recover")
```

1 という値は，/RECOVER が指定されたことを示します。

JOURNAL_OPEN 組込みプロシージャが実行されると，通常は， DEC XTPU は出力のためにジャーナル・ファイルをオープンします。/RECOVER を指定し，JOURNAL_OPEN 組込みプロシージャを実行すると，ジャーナル・ファイルは入力と出力のためにオープンされます。 DEC XTPU は，入力ファイルに含まれるすべてのコマンドを復元するために入力ファイルをオープンします。その後， DEC XTPU は編集セッションの残りの部分に関して，または JOURNAL_CLOSE が実行されるまで，キーストロークをジャーナル・ファイルに保存します。 DEC XTPU の上にレイヤ構造となっているインタフェースは， JOURNAL_OPEN 組込みプロシージャが実行されるかどうかを制御しなければなりません。

セッションを回復する場合，個人辞書の内容を含むファイルはすべて，回復する編集セッションを開始したときと同じ状態でなければなりません。また端末装置特性はすべて，回復する編集セッションを開始したときと同じ状態でなければなりません。端末装置の幅や 1 ページの長さを変更した場合には，回復する編集セッションを開始したときの値に戻しておかなければりません。特に次の値には注意してください。

- Device_Type
- Edit_mode
- Eightbit
- Page
- Width

ジャーナル・ファイルの名前が入力ファイルの名前と異なる場合には， DCL コマンド・ラインに/JOURNAL=*filespec* 修飾子と/RECOVER 修飾子の両方を指定しなければなりません。

```
$ EDIT/XTPU/RECOVER/JOURNAL=save.dat letter.dat
```

/NORECOVER は DEC XTPU の省略時の値です。

5.5.14 /SECTION

```
/SECTION[=filespec]
/NOSECTION
/SECTION=XTPU$SECTION ( 省略時設定 )
```

DEC XTPU が，セクション・ファイルを読み取るかどうかを指定します。セクション・ファイルとは，キー定義やコンパイルされたプロシージャをバイナリ形式で持っているファイルのことです。

省略時のセクション・ファイルは XTPU$SECTION です。セクション・ファイルを検索するときには省略時のディレクトリに SYS$SHARE，省略時のファイル・タイプに.XTPU$SECTION を使用します。 XTPU$SECTION という論理名には，デフォルトで JEVE$SECTION_V3 が定義されているので，省略時には SYS$SHARE:JEVE$SECTION_V3.XTPU$SECTION を読み込みます。これは日本語 EVE インタフェースです。初期化のために別のファイルを指定することもできます。このときには，省略時のファイル以外のセクション・ファイルを指定するために， XTPU$SECTION という論理名を再定義してください。また/SECTION 修飾子にセクション・ファイルの完全なファイル仕様を指定することもできます。

次のコマンドを実行すると， DEC XTPU は vt282ini.XTPU$SECTION というセクション・ファイルを読み取ります。

```
$ EDIT/XTPU/SECTION=disk$user:[smith]vt282ini
```

ファイル指定のときに，装置とディレクトリを指定しないと， DEC XTPU は SYS$SHARE にあるファイルをさがします。セクション・ファイルは， DEC XTPU を起動するノードと同じノードに存在しなければなりません。

DCL コマンド・ラインで/SECTION が指定されたかどうかを知るには，アプリケーションの中で以下のようにします。

```
x := GET_INFO (COMMAND_LINE, "section");
```

/SECTION が指定されたときには 1 が，指定されなかったときには 0 が戻されます。コマンド・ラインで指定されたファイル名を得るには以下のようにします。

```
x := GET_INFO (COMMAND_LINE, "section_file");
```

/SECTION=*filespec* に対する値として使用されるファイルは， DEC XTPU でファイルのソース・コードのバージョンを実行してコンパイルし， SAVE 組込みプロシージャを使用して保存しておかなければなりません。この処理を実行すると，ファイルは正しいバイナリ形式に変換されます。

/NOSECTION を指定した場合には，DEC XTPU はセクション・ファイルを読み取りません。/COMMAND 修飾子も使用しなかった場合には，DEC XTPU はユーザ・インタフェースを持たず，どのキーも定義されない状態となります。この状態では，Ctrl/Y を押さなければ DEC XTPU を終了することはできません。/NOSECTION は，日本語 EVE をベースにしないで独自の DEC XTPU アプリケーションを作るときに使用されます。

5.5.15 /START_POSITION

```
/START_POSITION=(line,column)
/START_POSITION=(1,1) ( 省略時設定 )
```

ユーザが DEC XTPU を使用して書かれたアプリケーションを起動したときの最初のカーソル位置を指定します。

DEC XTPU 上のアプリケーションが，この修飾子を使用してカーソル位置を設定する処理を行います。

次のようにすると DCL コマンド・ラインで/START_POSITION 修飾子で指定された値を知ることができます。

```
start_line := GET_INFO (COMMAND_LINE, "start_record");
start_line := GET_INFO (COMMAND_LINE, "start_character");
```

DCL コマンド・ラインで/START_POSITION 修飾子が指定されなかったときは，DEC XTPU は行，カラムともに 1 を設定します。

日本語 EVE ではこの修飾子を使用して，メイン・バッファのどの位置に最初にカーソルが置かれるかを指定します。省略時の最初の位置は，1 行目の 1 カラム目です。

5.5.16 /WORK

```
/WORK[=filespec]
/NOWORK
/WORK=SYS$SCRATCH:XTPU$WORK.XTPU$WORK ( 省略時設定 )
```

大きなファイルを編集する場合に XTPU がメモリの待避に使用する作業ファイルを指定します。作業ファイルは，終了時に自動的に削除されます。/NOWORK を指定して XTPU を起動すれば，作業ファイルは作られません。その場合，XTPU が使用できるメモリの量によって，編集できるファイルの大きさが制限されます。

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。編集セッションごとに 1 つの作業ファイルが用いられます。作業ファイルの省略時のファイル・タイプは.XTPU$WORK です。

省略時の設定では，XTPU$WORK.XTPU$WORK という作業ファイルが SYS$SCRATCH に作成されます。作業ファイルの指定には，2 通りの方法があります。

- 論理名 XTPU$WORK で作業ファイルを指定する。

  SYS$SCRATCH 以外の領域，たとえば，もっと大きなディスク上に作業ファイルを作成する場合に有用です。ファイル LOGIN.COM で定義できます。

- /WORK=修飾子を使用して，作業ファイルを指定する。

  論理名 XTPU$WORK の指定よりも優先されます。たとえば，次のコマンドは，作業用ファイルとして MYWORK.XTPU$WORK を使用します。

```
$ EDIT/XTPU /WORK=mywork
```

SYS$SCRATCH 以外の領域に作業ファイルを作るには，作業ファイル指定に装置名およびディレクトリを指定します。

作業ファイルを作成しない場合には，/NOWORK を使用します。これによって，システム資源の消費量が減り，XTPU の起動がより速くなります。一般に，システムがメモリを制限していない場合や，巨大なファイルを編集したり大量のファイルを編集したりすることがない場合には，/NOWORK を使用することができます。

5.5.17 /WRITE

```
/WRITE ( 省略時設定 )
/NOWRITE
```

メイン・バッファの内容が書き換えられたときに DEC XTPU 上のアプリケーションが内容を新しいファイルに書き込むかどうかを指定します。

/WRITE 修飾子がどのように処理されるかは，/READ_ONLY 修飾子にも関係しています。/WRITE は/NOREAD_ONLY と同じです。また/NOWRITE は/READ_ONLY と同じです。

次のような組み合せが DCL コマンド・ラインで使用されたときには，DEC XTPU はエラーを通知して DCL に戻ります。

- /READ_ONLY と/WRITE
- /NOREAD_ONLY と/NOWRITE

DEC XTPU で書かれたアプリケーションがこの修飾子を処理します。次のようにすれば，/WRITE または/NOREAD_ONLY 修飾子が DCL コマンド・ラインで使用されたかどうかを知ることができます。

```
x := GET_INFO (COMMAND_LINE, "write");
```

1 という値は，/NOREAD_ONLY あるいは/WRITE が明示的に指定されたことを示します。

次のようにすれば，/NOWRITE または/READ_ONLY 修飾子が DCL コマンド・ラインで使用されたかどうかを知ることができます。

```
x := GET_INFO (COMMAND_LINE, "read_only");
```

1 という値は，/READ_ONLY あるいは/NOWRITE が明示的に指定されたことを示します。

両方の GET_INFO ともに“偽”を返したときには，アプリケーションは省略時の動作をします (省略時の動作はアプリケーションによって異なります)。

日本語 EVE においては，/NOWRITE 修飾子は，/NOMODIFY，/NOOUTPUT を指定したときとまったく同じです。

5.6 DEC XTPU コマンドのパラメータ

DCL レベルで DEC XTPU コマンドの後にパラメータとして OpenVMS ファイル仕様を使用することができます。ファイル仕様は DEC XTPU で作成または編集したいファイルの名前をオペレーティング・システムに対して指定するために使用されます。以下のコマンド・ラインは日本語 EVE のセクション・ファイルを使用して DEC XTPU を起動し， history.txt というファイルを指定します。

```
$ EDIT/XTPU/SECTION=sys$library:jeve$section_v3 history.txt
```

セクション・ファイルを使用しないで DEC XTPU を起動するときには，パラメータを指定する必要はありません。しかし， DEC XTPU の上にレイヤ構造となっている編集インタフェースの多くは，処理したいファイルを指定するために DCL コマンド・ラインのパラメータが使用されています。たとえば，日本語 EVE は省略可能なパラメータとしてファイル仕様を受け付けます。ファイル名を指定しなくても編集セッションを開始することはできますが，バッファにデータを入力した後で DEC XTPU を終了しようとすると，ファイル名を要求するプロンプトが表示されます。

ファイル仕様はファイル名だけでも，また次の例に示されているように，完全なファイル仕様でも指定できます。

```
$ EDIT/XTPU disk$user:[smith]letter.dat
```

使用しているインタフェースによって， DEC XTPU は入力ファイル名の一部としてワイルドカード文字を認識できる場合とできない場合があります。日本語 EVE はワイルドカード文字に対して一致するものが 1 つのときに，ワイルドカード文字を処理します。それ以外の場合には，日本語 EVE は該当するファイル名の候補を表示して，指定を求めます。

ファイル仕様の一部としてバージョン番号を指定する必要はありません。バージョン番号を指定しなかったときには，DEC XTPU はディレクトリの中で最大のバージョン番号を読み込みます。古いバージョンのファイルを編集したい場合には，ファイル仕様にバージョン番号を指定してください。

ファイル仕様には複数のファイルを同時に指定することができます。複数のファイルを指定する場合にはファイル名をコンマ (,) で区切って指定します。以下の例では first.txt と second.txt の 2 つのファイルを指定しています。

```
$ EDIT/XTPU first.txt,second.txt
```

コマンド・ラインに/OUTPUT 修飾子を指定することにより出力ファイルの名前を指定しない限り，日本語 EVE は入力ファイルの名前を出力ファイル名として使用します。この場合，編集しているもとのバージョンが変更されることはありません。システム管理者がバージョン・リミットをセットしていない限り，もとのバージョンもディレクトリにそのまま保存されます。 DEC XTPU を終了すると，新しいファイルが入力ファイル・ディレクトリに作成されます (別のディレクトリを指定していない場合)。出力ファイルは入力ファイルと同じ名前になりますが，バージョン番号は入力ファイルより 1 つだけ大きい番号となります。

6

呼び出し可能な DEC XTPU

この章では，呼び出し可能な DEC XTPU ルーチンについて説明します。すなわち，呼び出し可能なルーチンの目的，ルーチン呼び出しのパラメータ，および主なステータス値について解説します。呼び出し構文のパラーメータは， DEC XTPU ルーチンに渡すオブジェクトを表します。それぞれのパラメータの説明では，そのオブジェクトのデータ型と受け渡し方法とを示します。データ型は， OpenVMS の標準データ型です。受け渡し方法は，パラメータ・リストがどう解釈されるかを示します。

この章は，対象読者として，以下の事柄を十分に理解しているシステム・プログラマを想定しています。

- OpenVMS 呼び出し規則
- OpenVMS ランタイム・ライブラリ (RTL)
- コンピュータ上でのデータ型の表現方法
- メイン・プログラムと異なる言語で書かれたルーチンの呼び出し方

6.1 概要

呼び出し可能な DEC XTPU は，他の言語やアプリケーションの中から DEC XTPU にアクセスできるようにするものです。 OpenVMS 呼び出し規則に従った呼び出しを生成する言語で書かれたプログラムであれば，その中から DEC XTPU を呼び出すことができます。また， MAIL ユーティリティなど， OpenVMS のユーティリティからも DEC XTPU を呼び出すことができます。呼び出し可能な DEC XTPU によって，ユーザ・プログラムの内部でテキスト処理機能を実行することが可能となります。

呼び出し可能な DEC XTPU は DEC XTPU の共有可能イメージである XTPUSHR.EXE に存在する呼び出し可能なルーチンから構成されます。呼び出し可能な DEC XTPU を使用するためには，この共有可能イメージとリンクします。共有可能イメージには，呼び出し可能なインタフェース・ルーチン名と定数が含まれています。 DCL レベルの DEC XTPU インタフェースの場合と同様に，呼び出し可能な DEC XTPU への入力として，また出力として，ファイルを使用することができます。さらに入力，出力，およびメッセージを処理するために独自のルーチンを作成することもできます。

呼び出し側のプログラムは，DEC XTPU プロシージャに渡すパラメータが，その DEC XTPU プロシージャの受け付けるデータ型および形式であることを確実にしなければなりません。

この章で説明している DEC XTPU ルーチンは，そのルーチンの終了ステータスを示す条件値を返します。返された条件値とテスト値を比較する場合には，ランタイム・ライブラリの LIB$MATCH_COND を使用しなければなりません。条件値を単純な整数としてテストしないようにしてください。

6.1.1 **呼び出し可能な** DEC XTPU **に対する** 2 **つのインタフェース**

呼び出し可能な DEC XTPU は，単純な呼び出し可能インタフェースと完全な呼び出し可能インタフェースの 2 通りの方法でアクセスすることができます。

単純な呼び出し可能インタフェース

呼び出し可能な DEC XTPU を使用するためのもっとも簡単な方法は，単純な呼び出し可能インタフェースを使用することです。 DEC XTPU には，単純な呼び出し可能インタフェースで使用できる 2 つのルーチンがあります。これらのルーチンはさらに以下の操作を実行する他のルーチンを呼び出します。

- エディタを初期化する
- エディタの操作に必要なパラメータをエディタに与える
- 編集セッションを制御する
- エラー処理を実行する

単純な呼び出し可能インタフェースを使用する場合， DEC XTPU を起動するためのコマンド・ラインを指定する XTPU$XTPU ルーチンか，または入力ファイルと出力ファイルとを指定する XTPU$EDIT ルーチンを使用します。 XTPU$EDIT はコマンド文字列を作成し，それを XTPU$XTPU へ渡します。これらの 2 つのルーチンについては，第 6.2 節で詳しく説明します。

完全な呼び出し可能インタフェース

完全な呼び出し可能インタフェースを使用する場合には，呼び出し可能な DEC XTPU のメイン・ルーチンを直接アクセスするプログラムを使用します。これらのルーチンは以下の操作を実行します。

エディタを初期化する	XTPU$INITIALIZE
DEC XTPU プロシージャを実行する	XTPU$EXECUTE_INIFILE XTPU$EXECUTE_COMMAND
エディタを制御する	XTPU$CONTROL
編集セッションを終了する	XTPU$CLEANUP

完全な呼び出し可能インタフェースを利用する場合，一定のパラメータに値を指定しなければなりません。ある場合には，値として追加ルーチンのアドレスを指定することになります。たとえば，XTPU$INITIALIZE を呼び出す場合には，初期化オプションを指定するルーチンのアドレスを指定しなければなりません。アプリケーションによっては，ユーザが追加ルーチンを作成しなければならないこともあります。たとえば，ファイルを操作したり，エラーを処理したり，あるいは編集セッションを制御したりするルーチンを作成する必要があるかもしれません。呼び出し可能な DEC XTPU は，そうした作業のいくつかを肩代わりするユーティリティ・ルーチンを用意しています。ユーティリティ・ルーチンによって以下のようなことができます。

- OpenVMS コマンド・ラインを解析して，エディタを初期化するときに使用するアイテム・リストを作成する
- ファイル操作を処理する
- エラー・メッセージを出力する
- 条件処理を行う

完全な呼び出し可能インタフェースについては，以下の各節に詳しく説明されています。

第 6.3 節	まずインタフェースの簡単な説明が示されていますが，この節の大部分は呼び出し可能な DEC XTPU のメイン・ルーチンの説明にあてられています (XTPU$INITIALIZE，XTPU$EXECUTE_INIFILE，XTPU$CONTROL，XTPU$EXECUTE_COMMAND，XTPU$CLEANUP)。
第 6.3.2 項	完全な呼び出し可能インタフェースで使用できる追加ルーチンが説明されています。
第 6.3.3 項	完全な呼び出し可能インタフェースで使用できるルーチンを作成するための必要条件が定義されています。

完全な呼び出し可能インタフェースは，呼び出し可能な DEC XTPU メイン・ルーチンと DEC XTPU ユーティリティ・ルーチンから構成されます。

6.1.2 共有可能イメージ

単純な呼び出し可能インタフェースを使用する場合も，完全な呼び出し可能インタフェースを使用する場合も，DEC XTPU の共有可能イメージにリンクすることにより，呼び出し可能な DEC XTPU にアクセスすることができます。このイメージにはアプリケーションで使用できるルーチン名と定数が含まれています。さらに，XTPUSHR.EXE には，以下のシンボルも含まれています。

XTPU$GL_VERSION	共有可能イメージのバージョン
XTPU$GL_UPDATE	共有可能イメージの更新番号
XTPU$_FACILITY	DEC XTPU のファシリティ・コード

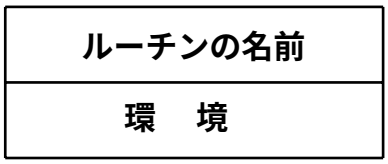
共有可能イメージ，XTPUSHR.EXE のリンク方法については，『日本語 OpenVMS 概説書』を参照してください。

6.1.3 **呼び出し可能な** DEC XTPU **ルーチンに対するパラメータの受渡し**

パラメータは参照またはディスクリプタによって呼び出し可能な DEC XTPU に渡されます。パラメータがルーチンの場合には，パラメータはバウンド・プロシージャ値 (BPV) データ・タイプとしてディスクリプタによって渡されます。

バウンド・プロシージャ値とは，2 つのロングワードから構成される値であり，最初のロングワードにはプロシージャのアドレスが含まれており，2 番目のロングワードには環境値が含まれています。図 6–1 を参照してください。環境値は最初のバウンド・プロシージャ値を作成するときに言語固有の方法で決定されます。バウンド・プロシージャが呼び出されると，呼び出しプログラムは 2 番目のロングワードを R1 にロードします。

図 6–1 バウンド・プロシージャ値



6.1.4 **エラー処理**

単純な呼び出し可能インタフェースを使用する場合，DEC XTPU はすべてのエラーを処理するために独自の条件ハンドラである XTPU$HANDLER を設定します。完全な呼び出し可能インタフェースを使用する場合には，以下の 2 通りの方法でエラーを処理することができます。

- DEC XTPU の省略時の条件ハンドラである XTPU$HANDLER を使用することができる
- 一部のエラーを処理するために独自の条件ハンドラを作成し，他のエラーは XTPU$HANDLER を呼び出すことにより処理することができる

省略時の条件ハンドラである XTPU$HANDLER については XTPU$FILEIO を参照してください。また，ユーザ独自の条件ハンドラを作成する方法については，『OpenVMS Programming Concepts Manual』を参照してください。

6.1.5 戻される値

DEC XTPU の条件コードはすべてユニバーサル・シンボルとして宣言されます。したがって，自分のプログラムを共有可能イメージにリンクすると，自動的にこれらのシンボルにアクセスできるようになります。 DEC XTPU は条件コード値を R0 にセットします。 DEC XTPU のリターン・コードについては，『DEC Text Processing Utility Reference Manual』を参照してください。 DEC XTPU が返すリターン・コードとそのメッセージについては， Help/Message ファシリティを参照してください。

この後の節では，呼び出し可能な DEC XTPU ルーチンの説明の中に条件コードに関する情報も示されています。この情報は " 戻される条件値 " という見出しの下に示されており，省略時の条件ハンドラが設定されているときに戻される値を示しています。

6.2 単純な呼び出し可能インタフェース

DEC XTPU の単純な呼び出し可能インタフェースは XTPU$XTPU と XTPU$EDIT という 2 つのルーチンから構成されています。 DEC XTPU に対するこれらのエントリ・ポイントは，以下のアプリケーションで使用することができます。

- 1 行のコマンド・ラインにすべての編集パラメータを指定できるアプリケーション
- 入力ファイルと出力ファイルだけを指定するだけでよいアプリケーション

6.2.1 単純なインタフェースの例

次の例は XTPU$EDIT を呼び出して， INFILE.DAT 中のテキストを編集し，その結果を OUTFILE.DAT に書き込みます。 XTPU$EDIT に渡すパラメータはディスクリプタ渡しでなければなりません。

例

```
/*
  Sample C program that calls DEC XTPU.  This program uses XTPU$EDIT to
  provide the names of the input and output files
*/

#include descrip

int return_status;

static $DESCRIPTOR (input_file, "infile.dat");
static $DESCRIPTOR (output_file, "outfile.dat");

main (argc, argv)
    int argc;
    char *argv[];
```

```
    {
    /*
      Call DEC XTPU to edit text in "infile.dat" and write the result
      to "outfile.dat".  Return the condition code from DEC XTPU as the
      status of this program.
    */

    return_status = XTPU$EDIT (&input_file, &output_file);
    exit (return_status);
    }
```

以下の例は，上の例と同じ動作をします。ここではエントリ・ポイントとして XTPU$XTPU を使います。 XTPU$XTPU は引数として "XTPU"で始まるコマンド文字列を受け取ります。コマンド文字列には "EDIT/XTPU"コマンドに使用できるすべての修飾子を使用できます。

例

```
/*
  Sample C program that calls DEC XTPU.  This program uses XTPU$XTPU and
  specifies a command string
*/

#include descrip

int return_status;

static $DESCRIPTOR (command_prefix, "XTPU/NOJOURNAL/NOCOMMAND/OUTPUT=");
static $DESCRIPTOR (input_file, "infile.dat");
static $DESCRIPTOR (output_file, "outfile.dat");
static $DESCRIPTOR (space_desc, " ");

char command_line [100];
static $DESCRIPTOR (command_desc, command_line);

main (argc, argv)
    int argc;
    char *argv[];

    {
    /*
      Build the command line for DEC XTPU.  Note that the command verb
      is "XTPU".  The string we construct in the buffer command_line
      will be
        "XTPU/NOJOURNAL/NOCOMMAND/OUTPUT=outfile.dat infile.dat"
    */

    return_status = STR$CONCAT (&command_desc,
                                &command_prefix,
                                &output_file,
                                &space_desc,
                                &input_file);
    if (! return_status)
        exit (return_status);
```

```
/*
  Now call DEC XTPU to edit the file
*/
return_status = XTPU$XTPU (&command_desc);
exit (return_status);
}
```

次節では，完全な呼び出し可能インタフェースによって呼び出されるルーチンについて詳しく説明します。単純な呼び出し可能インタフェースを使用した場合には，これらのルーチンがインタフェースによって呼び出されます。完全な呼び出し可能インタフェースを使用する場合には，ユーザ・プログラムが直接これらのルーチンを呼び出します。

6.3 完全な呼び出し可能インタフェース

DEC XTPU の完全な呼び出し可能インタフェースは，以下の作業を実行するために使用できる複数のルーチンから構成されています。

- 初期化パラメータを指定する
- ファイルの入出力を制御する
- エディタが実行するコマンドを指定する
- 条件の処理方法を制御する

単純な呼び出し可能インタフェースを使用する場合には，上記の操作は自動的に実行されます。これらの機能を実行する各 DEC XTPU ルーチンはユーザ作成プログラムから呼び出すことができ， DEC XTPU の完全な呼び出し可能インタフェースと呼ばれます。このインタフェースには 2 種類のルーチンが含まれています。すなわち DEC XTPU の呼び出し可能なメイン・ルーチンと DEC XTPU ユーティリティ・ルーチンです。これらの DEC XTPU ルーチンと， DEC XTPU ルーチンにパラメータを渡すユーザ・ルーチンを使用することにより，アプリケーション・プログラムが DEC XTPU を制御します。

以降の節では，呼び出し可能なメイン・ルーチン，これらのルーチンに対するパラメータの受渡し方法， DEC XTPU ユーティリティ・ルーチン，およびユーザ作成ルーチンの必要条件について説明します。

6.3.1 主な呼び出し可能な DEC XTPU ユーティリティ・ルーチン

この節では，以下の呼び出し可能な DEC XTPU ルーチンについて説明します。

- XTPU$INITIALIZE
- XTPU$EXECUTE_INIFILE

- XTPU$CONTROL
- XTPU$EXECUTE_COMMAND
- XTPU$CLEANUP

注意

これらのルーチンを呼び出す前に，条件ハンドラとして XTPU$HANDLER **かまたはユーザ独自のルーチンを設定しておかなければなりません。条件ハンドラの設定については，この章の** XTPU$HANDLER **ルーチンの説明および**『OpenVMS Calling Standard』**を参照してください。**

6.3.2 その他の DEC XTPU ユーティリティ・ルーチン

完全な呼び出し可能インタフェースにはユーティリティ・ルーチンがいくつか含まれており，それらに対してはパラメータを渡すことができます。アプリケーションによっては，ユーザ独自のルーチンの代わりにこれらのルーチンが利用できる場合があります。ユーティリティ・ルーチンには以下のものがあります。

XTPU$CLIPARSE	コマンド・ラインを解析し，XTPU$INITIALIZE 用のアイテム・リストを作成する
XTPU$PARSEINFO	コマンドを解析し，XTPU$INITIALIZE 用のアイテム・リストを作成する
XTPU$FILEIO	省略時のファイル入出力ルーチン
XTPU$MESSAGE	MESSAGE 組込みプロシージャを使ってエラー・メッセージや文字列を出力する
XTPU$HANDLER	省略時の条件ハンドラ
XTPU$CLOSE_TERMINAL	CALL_USER ルーチンの実行中，DEC XTPU のターミナルへのチャネルをクローズする
XTPU$SPECIFY_ASYNC_ACTION	XTPU$CONTROL ルーチンを中断するための非同期イベントを指定する
XTPU$TRIGGER_ASYNC_ACTION	指定の非同期イベントで XTPU$CONTROL ルーチンを中断する

XTPU$CLIPARSE および XTPU$PARSEINFO は，コマンド解析のために CLI$ルーチンによって保持されているデータを破壊することに注意してください。

6.3.3 ユーザ作成ルーチン

この節では，ユーザ作成ルーチンの必要条件を定義します。これらのルーチンを DEC XTPU に渡す場合には，バウンド・プロシージャ値として渡さなければなりません (バウンド・プロシージャ値については第 6.1.3 項を参照してください)。アプリケーションに応じて，以下に示すルーチンのうち，1 つまたはすべてを作成しなければなりません。

- 初期化コールバックのためのルーチン

このルーチンは，初期化パラメータの値を入手するために， XTPU$INITIALIZE によって呼び出されます。初期化パラメータはアイテム・リストとして返します。

- ファイル入出力のためのルーチン

  このルーチンはファイル操作を処理するルーチンです。独自のファイル入出力ルーチンを作成する代わりに， XTPU$FILEIO ユーティリティ・ルーチンを使用することもできます。ただし，ジャーナル・ファイル操作や SAVE 組込みプロシージャによって実行される動作ではこのルーチンは使用されません。

- 条件処理のためのルーチン

  このルーチンはエラー条件を処理するルーチンです。独自の条件ハンドラを作成する代わりに，省略時の条件ハンドラである XTPU$HANDLER を使用することもできます。

- CALL_USER 組込みプロシージャのためのルーチン

  このルーチンは CALL_USER 組込みプロシージャによって呼び出されるルーチンです。この仕組みを利用すれば，編集セッション中に制御をユーザ・プログラムに渡すことができます。

6.4 DEC XTPU ルーチンの使用例

例 6–1，例 6–2，および例 6–3 は，呼び出し可能な DECXTPU を使用したものです。これらの例は解説を目的としたものであり， DEC はこれらの信頼性について，その責任を負いません。

例 6–1 DEC FORTRAN における通常の DEC XTPU セットアップ

```
C       A sample FORTRAN program that calls DEC XTPU to act
C       normally, using the programmable interface.
C
C       IMPLICIT NONE

        INTEGER*4       CLEAN_OPT       !options for clean up routine
        INTEGER*4       STATUS          !return status from DEC XTPU routines
        INTEGER*4       BPV_PARSE(2)    !set up a Bound Procedure Value
        INTEGER*4       LOC_PARSE       !a local function call

C       declare the DEC XTPU functions

        INTEGER*4       XTPU$CONTROL
        INTEGER*4       XTPU$CLEANUP
        INTEGER*4       XTPU$EXECUTE_INIFILE
        INTEGER*4       XTPU$INITIALIZE
        INTEGER*4       XTPU$CLIPARSE

C       declare a local copy to hold the values of DEC XTPU cleanup variables
```

(次ページに続く)

例 6–1 (続き) DEC FORTRAN における通常の DEC XTPU セットアップ

```
        INTEGER*4       RESET_TERMINAL
        INTEGER*4       DELETE_JOURNAL
        INTEGER*4       DELETE_BUFFERS,DELETE_WINDOWS
        INTEGER*4       DELETE_EXITH,EXECUTE_PROC
        INTEGER*4       PRUNE_CACHE,KILL_PROCESSES
        INTEGER*4       CLOSE_SECTION
        INTEGER*4       CLOSE_KANJI_DIC

C       declare the DEC XTPU functions used as external

        EXTERNA         XTPU$HANDLER
        EXTERNAL        XTPU$CLIPARSE

        EXTERNAL        XTPU$_SUCCESS    !external error message

        EXTERNAL        LOC_PARSE       !user supplied routine to

C                                       call TPUCLIPARSE and setup
C       declare the DEC XTPU cleanup variables as external these are the
C       external literals that hold the value of the options

        EXTERNAL        XTPU$M_RESET_TERMINAL
        EXTERNAL        XTPU$M_DELETE_JOURNAL
        EXTERNAL        XTPU$M_DELETE_BUFFERS,XTPU$M_DELETE_WINDOWS
        EXTERNAL        XTPU$M_DELETE_EXITH,XTPU$M_EXECUTE_PROC
        EXTERNAL        XTPU$M_PRUNE_CACHE,XTPU$M_KILL_PROCESSES
        EXTERNAL        XTPU$M_CLOSE_KANJI_DIC

100     CALL LIB$ESTABLISH ( XTPU$HANDLER ) !establish the condition handler

C       set up the Bound Procedure Value for the call to XTPU$INITIALIZE

        BPV_PARSE( 1 ) = %LOC( LOC_PARSE )
        BPV_PARSE( 2 ) = 0

C       call the DEC XTPU initialization routine to do some set up work

        STATUS = XTPU$INITIALIZE ( BPV_PARSE )

C       Check the status if it is not a success then signal the error

        IF ( STATUS .NE. %LOC ( XTPU$_SUCCESS ) ) THEN

                CALL LIB$SIGNAL( %VAL( STATUS ) )
                GOTO 9999

        ENDIF

C       execute the XTPU$_ init files and also a command file if it
C       was specified in the command line call to DEC XTPU

        STATUS = XTPU$EXECUTE_INIFILE ( )

        IF ( STATUS .NE. %LOC ( XTPU$_SUCCESS ) ) THEN

                CALL LIB$SIGNAL( %VAL( STATUS ) )
                GOTO 9999

        ENDIF

C       invoke the editor as it normally would appear
```

(次ページに続く)

例 6–1 (続き) DEC FORTRAN における通常の DEC XTPU セットアップ

```
        STATUS = XTPU$CONTROL ( )       !call the DEC XTPU editor

        IF ( STATUS .NE. %LOC ( XTPU$_SUCCESS ) ) THEN

                CALL LIB$SIGNAL( %VAL( STATUS ) )
C               GOTO 9999
        ENDIF

C       Get the value of the option from the external literals.  In FORTRAN you
C       cannot use external literals directly so you must first get the value
C       of the literal from its external location.  Here we are getting the
C       values of the options that we want to use in the call to XTPU$CLEANUP.

        DELETE_JOURNAL  = %LOC ( XTPU$M_DELETE_JOURNAL )
        DELETE_EXITH    = %LOC ( XTPU$M_DELETE_EXITH )
        DELETE_BUFFERS  = %LOC ( XTPU$M_DELETE_BUFFERS )
        DELETE_WINDOWS  = %LOC ( XTPU$M_DELETE_WINDOWS )
        EXECUTE_PROC    = %LOC ( XTPU$M_EXECUTE_PROC )
        RESET_TERMINAL  = %LOC ( XTPU$M_RESET_TERMINAL )
        KILL_PROCESSES  = %LOC ( XTPU$M_KILL_PROCESSES )
        CLOSE_SECTION   = %LOC ( XTPU$M_CLOSE_SECTION )
        CLOSE_KANJI_DIC = %LOC ( XTPU$M_CLOSE_KANJI_DIC )

C       Now that we have the local copies of the variables we can do the
C       logical OR to set the multiple options that we need.

        CLEAN_OPT = DELETE_JOURNAL .OR. DELETE_EXITH .OR.
        1       DELETE_BUFFERS .OR. DELETE_WINDOWS .OR. EXECUTE_PROC
        1       .OR. RESET_TERMINAL .OR. KILL_PROCESSES .OR. CLOSE_SECTION
        1       .OR. CLOSE_KANJI_DIC


C       do the necessary clean up
C       XTPU$CLEANUP wants the address of the flags as the parameter so
C       pass the %LOC of CLEAN_OPT which is the address of the variable

        STATUS = XTPU$CLEANUP ( %LOC ( CLEAN_OPT ) )

        IF ( STATUS .NE. %LOC (XTPU$_SUCCESS) ) THEN

                CALL LIB$SIGNAL( %VAL(STATUS) )

        ENDIF


9999    CALL LIB$REVERT         !go back to normal processing -- handlers

        STOP
        END

C
C
        INTEGER*4  FUNCTION LOC_PARSE

        INTEGER*4       BPV(2)          !A local Bound Procedure Value

        CHARACTER*13    EDIT_COMM       !A command line to send to XTPU$CLIPARSE

C       Declare the DEC XTPU functions used
```

(次ページに続く)

例 6–1 (続き) DEC FORTRAN における通常の DEC XTPU セットアップ

```
        INTEGER*4       XTPU$FILEIO
        INTEGER*4       XTPU$CLIPARSE

C       Declare this routine as external because it is never called directly and
C       we need to tell FORTRAN that it is a function and not a variable

        EXTERNAL        XTPU$FILEIO

        BPV(1) = %LOC(XTPU$FILEIO)       !set up the BOUND PROCEDURE VALUE
        BPV(2) = 0

        EDIT_COMM(1:18) = 'EDIT/XTPU TEST.TXT'

C       parse the command line and build the item list for XTPU$INITIALIZE
9999     LOC_PARSE = XTPU$CLIPARSE (EDIT_COMM, BPV , 0)

        RETURN
        END
```

例 6–2 DEC FORTRAN でのコール・バック項目リストの作成

```
       PROGRAM  TEST_TPU
C
       IMPLICIT NONE
C
C       Define the expected DEC XTPU return statuses
C
       EXTERNAL        XTPU$_SUCCESS
       EXTERNAL        XTPU$_QUITTING

C
C       Declare the DEC XTPU routines and symbols used
C
       EXTERNAL        XTPU$M_DELETE_CONTEXT
       EXTERNAL        XTPU$HANDLER
       INTEGER*4       XTPU$M_DELETE_CONTEXT
       INTEGER*4       XTPU$INITIALIZE
       INTEGER*4       XTPU$EXECUTE_INIFILE
       INTEGER*4       XTPU$CONTROL
       INTEGER*4       XTPU$CLEANUP

C
C       Declare the external callback routine
C
       EXTERNAL        TPU_STARTUP       ! the DEC XTPU set-up function
       INTEGER*4       TPU_STARTUP

       INTEGER*4       BPV(2)            ! Set up a bound procedure value
```

(次ページに続く)

例 6–2 (続き) DEC FORTRAN でのコール・バック項目リストの作成

```
C
C      Declare the functions used for working with the condition handler
C
       INTEGER*4       LIB$ESTABLISH
       INTEGER*4       LIB$REVERT

C
C      Local Flags and Indices
C
       INTEGER*4       CLEANUP_FLAG       ! flag(s) for DEC XTPU cleanup
       INTEGER*4       RET_STATUS

C
C      Initializations
C
       RET_STATUS       = 0
       CLEANUP_FLAG     = %LOC(XTPU$M_DELETE_CONTEXT)

C
C      Establish the default DEC XTPU condition handler
C
       CALL LIB$ESTABLISH(%REF(XTPU$HANDLER))

C
C      Set up the bound procedure value for the initialization callback
C
       BPV(1)  =  %LOC (TPU_STARTUP)
       BPV(2)  =  0

C
C      Call the DEC XTPU procedure for initialization
C
       RET_STATUS = XTPU$INITIALIZE(BPV)

       IF (RET_STATUS .NE. %LOC(XTPU$_SUCCESS)) THEN
       CALL LIB$SIGNAL (%VAL(RET_STATUS))
       ENDIF
C
C      Execute the DEC XTPU initialization file
C
       RET_STATUS = XTPU$EXECUTE_INIFILE()

       IF (RET_STATUS .NE. %LOC(XTPU$_SUCCESS)) THEN
       CALL LIB$SIGNAL (%VAL(RET_STATUS))
       ENDIF

C
C      Pass control to DEC XTPU
C
       RET_STATUS = XTPU$CONTROL()

       IF (RET_STATUS .NE. %LOC(XTPU$_QUITTING)
     1         .OR. %LOC(XTPU$_QUITTING)) THEN
                CALL LIB$SIGNAL (%VAL(RET_STATUS))
       ENDIF
```

(次ページに続く)

例 6–2 (続き) DEC FORTRAN でのコール・バック項目リストの作成

```
C
C      Clean up after processing
C
       RET_STATUS = XTPU$CLEANUP(%REF(CLEANUP_FLAG))

       IF (RET_STATUS .NE. %LOC(XTPU$_SUCCESS)) THEN
       CALL LIB$SIGNAL (%VAL(RET_STATUS))
       ENDIF

C
C      Set the condition handler back to the default
C
       RET_STATUS = LIB$REVERT()

       END


       INTEGER*4 FUNCTION TPU_STARTUP

       IMPLICIT NONE

       INTEGER*4       OPTION_MASK       ! temporary variable for DEC XTPU
       CHARACTER*44       SECTION_NAME   ! temporary variable for DEC XTPU

C
C      External DEC XTPU routines and symbols
C
       EXTERNAL        XTPU$K_OPTIONS
       EXTERNAL        XTPU$M_READ
       EXTERNAL        XTPU$M_SECTION
       EXTERNAL        XTPU$M_DISPLAY
       EXTERNAL        XTPU$K_SECTIONFILE
       EXTERNAL        XTPU$K_FILEIO
       EXTERNAL        XTPU$FILEIO
       INTEGER*4       XTPU$FILEIO

C
C      The bound procedure value used for setting up the file I/O routine
C
       INTEGER*4       BPV(2)


C
C       Define the structure of the item list defined for the callback
C
       STRUCTURE /CALLBACK/
       INTEGER*2       BUFFER_LENGTH
       INTEGER*2       ITEM_CODE
       INTEGER*4       BUFFER_ADDRESS
       INTEGER*4       RETURN_ADDRESS
       END STRUCTURE
```

(次ページに続く)

例 6–2 (続き) DEC FORTRAN でのコール・バック項目リストの作成

```
C
C      There are a total of four items in the item list
C
       RECORD /CALLBACK/ CALLBACK (4)
C
C      Make sure it is not optimized!
C
       VOLATILE /CALLBACK/

C
C       Define the options we want to use in the DEC XTPU session
C
       OPTION_MASK = %LOC(XTPU$M_SECTION) .OR. %LOC(XTPU$M_READ)
     1        .OR. %LOC(XTPU$M_DISPLAY)

C
C      Define the name of the initialization section file
C
       SECTION_NAME = 'SYS$SHARE:JEVE$SECTION_V3.XTPU$SECTION'

C
C      Set up the required I/O routine.  Use the DEC XTPU default.
C
       BPV(1) = %LOC(XTPU$FILEIO)
       BPV(2) = 0

C
C      Build the callback item list
C
C      Set up the edit session options
C
       CALLBACK(1).ITEM_CODE = %LOC(XTPU$K_OPTIONS)
       CALLBACK(1).BUFFER_ADDRESS = %LOC(OPTION_MASK)
       CALLBACK(1).BUFFER_LENGTH = 4
       CALLBACK(1).RETURN_ADDRESS = 0

C
C      Identify the section file to be used
C
       CALLBACK(2).ITEM_CODE = %LOC(XTPU$K_SECTIONFILE)
       CALLBACK(2).BUFFER_ADDRESS = %LOC(SECTION_NAME)
       CALLBACK(2).BUFFER_LENGTH = LEN(SECTION_NAME)
       CALLBACK(2).RETURN_ADDRESS = 0

C
C      Set up the I/O handler
C
       CALLBACK(3).ITEM_CODE = %LOC(XTPU$K_FILEIO)
       CALLBACK(3).BUFFER_ADDRESS = %LOC(BPV)
       CALLBACK(3).BUFFER_LENGTH = 4
       CALLBACK(3).RETURN_ADDRESS = 0
```

(次ページに続く)

例 6–2 (続き) DEC FORTRAN でのコール・バック項目リストの作成

```
C
C      End the item list with zeros to indicate we are finished
C
       CALLBACK(4).ITEM_CODE = 0
       CALLBACK(4).BUFFER_ADDRESS = 0
       CALLBACK(4).BUFFER_LENGTH = 0
       CALLBACK(4).RETURN_ADDRESS = 0

C
C      Return the address of the item list
C
       TPU_STARTUP = %LOC(CALLBACK)

       RETURN
       END
```

例 6–3 DEC C でのユーザ作成ファイルの入出力ルーチンの指定

```
/*
Simple example of a C program to invoke DEC XTPU.  This program provides its
own FILEIO routine instead of using the one provided by DEC XTPU.
*/
#include descrip
#include stdio

/* data structures needed */

struct bpv_arg                  /* bound procedure value */
    {
    int *routine_add ;          /* pointer to routine */
    int env ;                   /* environment pointer */
    } ;

struct item_list_entry          /* item list data structure */
    {
    short int buffer_length;    /* buffer length */
    short int item_code;        /* item code */
    int *buffer_add;            /* buffer address */
    int *return_len_add;        /* return address */
    } ;

struct stream_type
    {
    int ident;                  /* stream id */
    short int alloc;            /* file size */
    short int flags;            /* file record attributes/format */
    short int length;           /* resultant file name length */
    short int stuff;            /* file name descriptor class & type */
    int nam_add;                /* file name descriptor text pointer */
    } ;
```

(次ページに続く)

例 6–3 (続き) DEC C でのユーザ作成ファイルの入出力ルーチンの指定

```
globalvalue xtpu$_success;       /* DEC XTPU Success code */
globalvalue xtpu$_quitting;      /* Exit code defined by DEC XTPU */

globalvalue                     /* Cleanup codes defined by DEC XTPU */
    xtpu$m_delete_journal, xtpu$m_delete_exith,
    xtpu$m_delete_buffers, xtpu$m_delete_windows, xtpu$m_delete_cache,
    xtpu$m_prune_cache, xtpu$m_execute_file, xtpu$m_execute_proc,
    xtpu$m_delete_context, xtpu$m_reset_terminal, xtpu$m_kill_processes,
    xtpu$m_close_section, xtpu$m_delete_others, xtpu$m_last_time;

globalvalue                     /* Item codes for item list entries */
    xtpu$_fileio, xtpu$_options, xtpu$_sectionfile,
    xtpu$_commandfile ;

globalvalue                     /* Option codes for option item */
    xtpu$m_display, xtpu$m_section, xtpu$m_command, xtpu$m_create ;

globalvalue                     /* Possible item codes in item list */
    xtpu$k_access, xtpu$k_filename, xtpu$k_defaultfile,
    xtpu$k_relatedfile, xtpu$k_record_attr, xtpu$k_maximize_ver,
    xtpu$k_flush, xtpu$k_filesize;

globalvalue                     /* Possible access types for xtpu$k_access */
    xtpu$k_io, xtpu$k_input, xtpu$k_output;

globalvalue                     /* RMS File Not Found message code */
    rms$_fnf;

globalvalue                     /* FILEIO routine functions */
    xtpu$k_open, xtpu$k_close, xtpu$k_close_delete,
    xtpu$k_get, xtpu$k_put;
int lib$establish ();           /* RTL routine to establish an event handler */
int xtpu$cleanup ();            /* XTPU routine to free resources used */
int xtpu$control ();            /* XTPU routine to invoke the editor */
int xtpu$execute_inifile ();    /* XTPU routine to execute initialization code */
int xtpu$handler ();            /* XTPU signal handling routine */
int xtpu$initialize ();         /* XTPU routine to initialize the editor */

/*
   This function opens a file for either read or write access, based upon
   the itemlist passed as the data parameter.  Note that a full implementation
   of the file open routine would have to handle the default file, related
   file, record attribute, maximize version, flush and file size item code
   properly.
 */
open_file (data, stream)

int *data;
struct stream_type *stream;
```

(次ページに続く)

例 6–3 (続き) DEC C でのユーザ作成ファイルの入出力ルーチンの指定

```
{
    struct item_list_entry *item;
    char *access;               /* File access type */
    char filename[256];         /* Max file specification size */

    /* Process the item list */

    item = data;
    while (item->item_code != 0 && item->buffer_length != 0)
        {
        if (item->item_code == xtpu$k_access)
            {
            if (item->buffer_add == xtpu$k_io) access = "r+";
            else if (item->buffer_add == xtpu$k_input) access = "r";
            else if (item->buffer_add == xtpu$k_output) access = "w";
            }
        else if (item->item_code == xtpu$k_filename)
            {
            strncpy (filename, item->buffer_add, item->buffer_length);
            filename [item->buffer_length] = 0;
            lib$scopy_r_dx (&item->buffer_length, item->buffer_add,
                                                  &stream->length);
            }
        else if (item->item_code == xtpu$k_defaultfile)
            {                           /* Add code to handle default file  */
            }                           /* spec here                        */
        else if (item->item_code == xtpu$k_relatedfile)
            {                           /* Add code to handle related       */
            }                           /* file spec here                   */
        else if (item->item_code == xtpu$k_record_attr)
            {                           /* Add code to handle record        */
            }                           /* attributes for creating files    */
        else if (item->item_code == xtpu$k_maximize_ver)
            {                           /* Add code to maximize version     */
            }                           /* number with existing file here   */
        else if (item->item_code == xtpu$k_flush)
            {                           /* Add code to cause each record    */
            }                           /* to be flushed to disk as written */
```

(次ページに続く)

例 6–3 (続き) DEC C でのユーザ作成ファイルの入出力ルーチンの指定

```
        else if (item->item_code == xtpu$k_filesize)
            {                           /* Add code to handle specification */
            }                           /* of initial file allocation here  */
        ++item;         /* get next item */
        }
    stream->ident = fopen(filename,access);
    if (stream->ident != 0)
        return xtpu$_success;
    else
        return rms$_fnf;
}

/*
  This procedure closes a file
 */
close_file (data,stream)
struct stream_type *stream;

{
    close(stream->ident);
    return xtpu$_success;
}

/*
  This procedure reads a line from a file
 */
read_line(data,stream)
struct dsc$descriptor *data;
struct stream_type *stream;

{
    char textline[984];                 /* max line size for XTPU records */
    int len;

    globalvalue rms$_eof;               /* RMS End-Of-File code */

    if (fgets(textline,984,stream->ident) == NULL)
        return rms$_eof;
    else
        {
        len = strlen(textline);
        if (len > 0)
            len = len - 1;
        return lib$scopy_r_dx (&len, textline, data);
        }
}

/*
  This procedure writes a line to a file
 */
write_line(data,stream)
struct dsc$descriptor *data;
struct stream_type *stream;
```

(次ページに続く)

例 6–3 (続き) DEC C でのユーザ作成ファイルの入出力ルーチンの指定

```
{
    char textline[984];               /* max line size for XTPU records */

    strncpy (textline, data->dsc$a_pointer, data->dsc$w_length);
    textline [data->dsc$w_length] = 0;
    fputs(textline,stream->ident);
    fputs("\n",stream->ident);
    return xtpu$_success;
}

/*
   This procedure will handle I/O for XTPU
 */
fileio(code,stream,data)
int *code;
int *stream;
int *data;

{
    int status;

/* Dispatch based on code type.  Note that a full implementation of the     */
/* file I/O routines would have to handle the close and delete code properly */
/* instead of simply closing the file                                       */

    if (*code == xtpu$k_open)                  /* Initial access to file */
        status = open_file (data,stream);
    else if (*code == xtpu$k_close)            /* End access to file */
        status = close_file (data,stream);
    else if (*code == xtpu$k_close_delete)     /* Treat same as close */
        status = close_file (data,stream);

    else if (*code == xtpu$k_get)              /* Read a record from a file */
        status = read_line (data,stream);
    else if (*code == xtpu$k_put)              /* Write a record to a file */
        status = write_line (data,stream);
    else
        {                                     /* Who knows what we got? */
        status = xtpu$_success;
        printf ("Bad FILEIO I/O function requested");
        }
    return status;
}
```

(次ページに続く)

例 6–3 (続き) DEC C でのユーザ作成ファイルの入出力ルーチンの指定

```
/*
   This procedure formats the initialization item list and returns it as
   is return value.
 */
callrout()
{
    static struct bpv_arg add_block =
        { fileio, 0 } ;         /* BPV for fileio routine */
    int options ;
    char *section_name = "XTPU$SECTION";
    static struct item_list_entry arg[4];

    /* Setup file I/O routine item entry */
    arg[0].item_code = (unsigned short int) xtpu$_fileio;
    arg[0].buffer_length = 4;
    arg[0].buffer_add = &add_block;
    arg[0].return_len_add = 0;

    /* Setup options item entry.  Leave journaling off. */
    options = xtpu$m_display | xtpu$m_section;
    arg[1].item_code = (unsigned short int) xtpu$_options;
    arg[1].buffer_length = 4;
    arg[1].buffer_add = &options;
    arg[1].return_len_add = 0;

    /* Setup section file name */
    arg[2].item_code = (unsigned short int) xtpu$_sectionfile;
    arg[2].buffer_length = strlen(section_name);
    arg[2].buffer_add = section_name;
    arg[2].return_len_add = 0;

    arg[3].item_code = 0;
    arg[3].buffer_length = 0;
    arg[3].buffer_add = 0;
    arg[3].return_len_add = 0;

    return arg;
}

/*
   Main program.  Initializes XTPU, then passes control to it.
 */
main()
{
    int return_status ;
    int cleanup_options;
    struct bpv_arg add_block;

/* Establish as condition handler the normal DEC XTPU handler */

    lib$establish(xtpu$handler);

/* Setup a BPV to point to the callback routine */
```

(次ページに続く)

例 6–3 (続き) DEC C でのユーザ作成ファイルの入出力ルーチンの指定

```
    add_block.routine_add = callrout ;
    add_block.env = 0;

/* Do the initialize of XTPU */

    return_status = xtpu$initialize(&add_block);
    if (!return_status)
        exit(return_status);

/* Have XTPU execute the procedure XTPU$INIT_PROCEDURE from the section file */
/* and then compile and execute the code from the command file */

    return_status = xtpu$execute_inifile();
    if (!return_status)
        exit (return_status);

/* Turn control over to XTPU */

    return_status = xtpu$control ();
    if (!return_status)
        exit(return_status);

/* Now clean up. */

    cleanup_options = xtpu$m_last_time | xtpu$m_delete_context;
    return_status = xtpu$cleanup (&cleanup_options);
    exit (return_status);

    printf("Experiment complete");
}
```

6.5 DEC XTPU ルーチン

この節では，個々の DEC XTPU ルーチンについて，ルーチン・プレート形式で説明します。

FILEIO

ユーザ作成ルーチン FILEIO は， DEC XTPU のファイル操作を処理するのに使用されます。このルーチンの名前は，ユーザ独自のファイル入出力ルーチンのものであるか，または DEC XTPU のファイル入出力ルーチンの名前 (XTPU$FILEIO) のいずれかです。

形式

FILEIO *code, stream, data*

戻り値

OpenVMS **用法** **条件値**
データ型 **符号なしロングワード**
アクセス **書き込みのみ**
受け渡し方 **参照による**

ロングワードの条件値です。ユーティリティ・ルーチンはほとんど，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値は， "戻される条件値 "にまとめられています。

引数

code

OpenVMS **用法** **符号なしロングワード**
データ型 **符号なしロングワード**
アクセス **読み取りのみ**
受け渡し方 **ディスクリプタによる**

実行する機能を指定する DEC XTPU からのアイテム・コードです。code 引数は実行する機能を指定する DEC XTPU からのアイテム・コードを含むロングワードのアドレスです。

stream

OpenVMS **用法** **未指定**
データ型 **符号なしロングワード**
アクセス **変更**
受け渡し方 **参照による**

ファイル記述です。stream 引数は， 4 つのロングワードを含むデータ構造のアドレスです。このデータ構造は，操作するファイルを記述するために使用されます。

data

OpenVMS **用法**　**アイテム・リスト** 3
データ型　**符号なしロングワード**
アクセス　**変更**
受け渡し方　**参照による**

ストリーム・データです。data 引数はアイテム・リストのアドレス，またはディスクリプタのアドレスです。

このパラメータの値は，どのアイテム・コードを指定したかによって異なります。

説明

このルーチンのバウンド・プロシージャ値は，コールバック・ルーチンによって作成されたアイテム・リストの中に指定されているものです。このルーチンは，ファイル操作を実行するために呼び出されます。ユーザが作成した独自のファイル入出力ルーチンを使用する代わりに，処理させるファイル操作をパラメータとして XTPU$FILEIO を呼び出すことができます。ただし， XTPU$FILEIO は，それがオープンしたファイルについてはすべての入出力要求を処理しなければならず，また，オープンしていないファイルに対しては何らの入出力要求も処理できないことに注意してください。言い換えれば，ファイル操作に関し，ユーザ独自のファイル入出力ルーチンと DEC XTPU の用意しているルーチンとを組み合わせることはできません。

説明

戻される条件値はユーザによって決められ，操作の正常終了あるいは失敗を指示します。

HANDLER

ユーザ作成ルーチン HANDLER は，条件処理を実行します。

形式

HANDLER *signal_vector, mechanism_vector*

戻り値

OpenVMS **用法　条件値**
データ型　符号なしロングワード
アクセス　書き込みのみ
受け渡し方　値による

ロングワードの条件値です。

引数

signal_vector

OpenVMS **用法　引数リスト**
データ型　符号なしロングワード
アクセス　変更
受け渡し方　参照による

シグナル・ベクタです。条件ハンドラに渡されるシグナル・ベクタについての説明は，『OpenVMS System Services Reference Manual』を参照してください。

mechanism_vector

OpenVMS **用法　引数リスト**
データ型　符号なしロングワード
アクセス　読み取りのみ
受け渡し方　参照による

メカニズム・ベクタです。条件ハンドラに渡されるメカニズム・ベクタについての説明は，『OpenVMS System Services Reference Manual』を参照してください。

説明

条件ハンドラの作成方法と，OpenVMS 条件処理標準についての詳しい説明が必要な場合には，『OpenVMS Programming Interfaces: Calling a System Routine』または『OpenVMS Calling Standard』を参照してください。

ユーザが独自に条件ハンドラを作成する代わりに，省略時の条件ハンドラである XTPU$HANDLER を利用することができます。条件ハンドラを独自に作成する場合には，DEC XTPU の内部シグナルを操作するため，そのルーチンは受け取ったのと同じパラメータで XTPU$HANDLER を呼び出さなければなりません。

INITIALIZE

初期化コールバック・ルーチン INITIALIZE は， XTPU$INITIALIZE にバウンド・プロシージャ値として渡され， DEC XTPU の初期化に必要な情報を得るために呼び出されるユーザ作成ルーチンです。

形式

INITIALIZE *[user_arg]*

戻り値

OpenVMS **用法** **アイテム・リスト**
データ型 **符号なしロングワード**
アクセス **書き込みのみ**
受け渡し方 **参照による**

このルーチンは，アイテム・リストのアドレスを返します。

引数

user_arg

OpenVMS **用法** **ユーザ引数**
データ型 **符号なしロングワード**
アクセス **読み取りのみ**
受け渡し方 **値による**

説明

初期化コールバック・ルーチン INITIALIZE は， XTPU$INITIALIZE にバウンド・プロシージャ値として渡され， DEC XTPU の初期化に必要な情報を得るために呼び出されるユーザ作成ルーチンです。

XTPU$INITIALIZE で user_arg パラメータが指定された時には，初期化コールバック・ルーチンは，そのパラメータだけを使って呼ばれます。 XTPU$INITIALIZE で user_arg パラメータが指定されなかった時には，初期化コールバック・ルーチンはパラメータなしで呼ばれます。

パラメータ user_arg によって，アプリケーションは XTPU$INITIALIZE を通してユーザによって書かれた初期化ルーチンに情報を渡すことができます。 DEC XTPU はこのデータを解釈しません。

この初期化コールバック・ルーチンは，初期化パラメータのアイテム・リストのアドレスを返すものと期待されています。そのアイテム・リストはこの初期化コールバック・ルーチンのスコープの外で用いられるため，静的メモリ内に割り当てられる必要があります。

このアイテム・リスト・エントリは XTPU$INITIALIZE のセクションに記述されています。ほとんどの初期化パラメータには省略値があります。省略時の文字列はヌル文字列です。省略時のフラグは FALSE です。唯一必要な初期化パラメータはファイル入出力のためのルーチンのアドレスです。ファイル入出力ルーチンのアドレスがアイテム・リストにないと， XTPU$INITIALIZE はエラー・ステータスを返します。

USER

ユーザ作成ルーチン USER を使用することによって，ユーザ・プログラムは DEC XTPU の編集セッション中に制御を取ることができます (たとえば，一時的にエディタを離れ，計算を実行するなど)。

形式

USER *integer, stringin, stringout*

戻り値

OpenVMS **用法** **条件値**
データ型 **符号なしロングワード**
アクセス **書き込みのみ**
受け渡し方 **値による**

ロングワードの条件値です。

引数

integer

OpenVMS **用法** **符号なしロングワード**
データ型 **符号なしロングワード**
アクセス **読み取りのみ**
受け渡し方 **ディスクリプタによる**

CALL_USER 組込みプロシージャに対する最初のパラメータです。これは入力専用パラメータであり，変更してはなりません。

stringin

OpenVMS **用法** **文字列**
データ型 **文字列**
アクセス **読み取りのみ**
受け渡し方 **ディスクリプタによる**

CALL_USER 組込みプロシージャに対する 2 番目のパラメータです。これは入力専用パラメータであり，変更してはなりません。

stringout

OpenVMS **用法** **文字列**
データ型 **文字列**
アクセス **読み取りのみ**

受け渡し方　　ディスクリプタによる

CALL_USER 組込みプロシージャへ返される文字列です。ユーザ・プログラムは，ランタイム・ライブラリが提供する文字列ルーチン (LIB$SGET1_DD など) によって割り当てられた動的文字列をこのディスクリプタに格納しなければなりません。 DEC XTPU エディタは必要に応じてこの文字列を解放します。

説明

ユーザ作成ルーチン USER は， DEC XTPU の CALL_USER 組込みプロシージャによって呼び出されます。 CALL_USER 組込みプロシージャはこのルーチンに 3 つのパラメータを渡します。このうち 2 つのパラメータは，ユーザ・アプリケーションがその目的に沿って用途を決めます (たとえば， FORTRAN プログラムで計算のオペランドとして使用することができます)。アプリケーションは，ランタイム・ライブラリが提供する文字列ルーチンを使い， USER ルーチンの stringout パラメータに値を格納することによって，その値を CALL_USER 組込みプロシージャに返します。

『DEC Text Processing Utility Reference Manual』には， BASIC による USER ルーチンのプログラム例が示されています。同マニュアルの CALL_USER 組込みプロシージャの説明を参照してください。

例

```
INTEGER FUNCTION XTPU$CALLUSER  (x,y,z)
    IMPLICIT NONE
    INTEGER X
    CHARACTER*(*) Y
    STRUCTURE /dynamic/ Z
              INTEGER*2 length
              BYTE    dtype
              BYTE    class
              INTEGER ptr
    END STRUCTURE
    RECORD /dynamic/ Z
    CHARACTER*80 local_copy
    INTEGER rs,lclen
    INTEGER STR$COPY_DX
    local_copy = '<' // y // '>'
    lclen = LEN (Y) + 2

    RS = STR$COPY_DX(Z,local_copy(1:lclen))
    XTPU$CALLUSER = RS
    END
```

この FORTRAN プログラムを呼び出す XTPU プロシージャの例を次に示します。

```
PROCEDURE MY_CALL
     local status;
     status := CALL_USER (0,'ABCD');
     MESSAGE('"' + status + '"');
ENDPROCEDURE
```

XTPU$CLEANUP

XTPU$CLEANUP ルーチンは，内部データ構造の後片付けをし，メモリを解放し，ターミナルを初期状態に戻します。これは DEC XTPU とのやりとりの最後に呼び出すルーチンです。

形式

XTPU$CLEANUP *flags*

戻り値

OpenVMS **用法** **条件値**
データ型 **符号なしロングワード**
アクセス **書き込みのみ**
受け渡し方 **値による**

ロングワードの条件値です。ユーティリティ・ルーチンはすべて，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値については， "戻される条件値 "にまとめられています。

引数

flags

OpenVMS **用法** **ロングワード・マスク**
データ型 **符号なしロングワード**
アクセス **読み取りのみ**
受け渡し方 **参照による**

クリーンアップ・オプションを定義するフラグ (あるいはマスク) です。flags 引数は，クリーンアップ・オプションを定義するロングワード・ビット・マスクのアドレス，または 32 ビット・マスクのアドレスです。このマスクは，セットしたいフラグ・ビットの論理和 (OR) です。 XTPU$V... はビット・アイテムを示し， XTPU$M... はマスクを示します。クリーンアップ・オプションは表 6–1 に示されているとおりです。

表 6–1 クリーンアップ・オプション

シンボル[1]	機能
XTPU$M_CLOSE_KANJI_DIC	かな漢字変換辞書をオープンしていた場合，かな漢字変換辞書をクローズします。
XTPU$M_CLOSE_KANJI_DIC	かな漢字変換辞書をオープンしていた場合，かな漢字変換辞書をクローズします。
XTPU$M_CLOSE_SECTION[2]	セクション・ファイルをクローズし，関連するメモリを解放します。バッファ，ウィンドウ，およびプロセスがすべて削除されます。キャッシュは整理され，初期化ファイルと初期化プロシージャの再実行のためのフラグがセットされます。セクションがクローズされ，かつオプション・ビットが/SECTION 修飾子の存在を示している場合，次に XTPU$INITIALIZE を呼び出すと，新しいセクション・ファイルが読み込まれます。
XTPU$M_DELETE_BUFFERS	すべてのテキスト・バッファを削除します。この後も DEC XTPU を呼び出す場合には，これらのデータ構造を参照する変数はすべて， DELETE 組込みプロシージャの場合と同様にリセットされます。バッファが削除される場合には，そのバッファに含まれるレンジとマーカ，およびそのバッファを使用するサブプロセスもすべて削除されます。
XTPU$M_DELETE_CACHE	仮想ファイル・マネージャのデータ構造とキャッシュを削除します。この削除が要求された場合，バッファもすべて削除されます。キャッシュが削除されると，初期化ルーチンは次に呼び出されたときに仮想ファイル・マネージャを初期化し直さなければならなくなります。
XTPU$M_DELETE_CONTEXT	DEC XTPU のコンテキスト全体を削除します。このオプションが指定されている場合には，初期化ファイルと初期化プロシージャの実行のためのオプションを除き，他のオプションもすべて指定されているものと解釈されます。
XTPU$M_DELETE_EXITH	DEC XTPU の終了ハンドラを削除します。
XTPU$M_DELETE_JOURNAL	ジャーナル・ファイルがオープンされている場合には，そのファイルをクローズし，削除します。
XTPU$M_DELETE_OTHERS	前もって割り当てられている種々のデータ構造を削除します。これらのデータ構造のために使用されていたメモリは，次に XTPU$INITIALIZE が呼び出されたときに再割り当てされます。
XTPU$M_DELETE_WINDOWS	すべてのウィンドウを削除します。その後も DEC XTPU を呼び出す場合には，これらのデータ構造を参照する変数はすべて， DELETE 組込みプロシージャの場合と同様にリセットされます。

[1]プリフィックスは XTPU$M_または XTPU$V_ です。 XTPU$M_ は，ビットがセットされるマスクが特定のフィールドに対応することを示しています。 XTPU$V_ はビット番号です。

[2]単純な呼び出し可能インタフェースを使用する場合には， XTPU$CLOSE_SECTION はセットされません。そのため， XTPU$XTPU を複数回呼び出す場合でも，そのたびにセクション・ファイルをオープンしたりクローズしたりする必要はありません。

(次ページに続く)

表 6–1 (続き) クリーンアップ・オプション

シンボル[1]	機能
XTPU$M_EXECUTE_FILE	次に XTPU$EXECUTE_INIFILE が呼び出されたときに，コマンド・ファイルを再び実行します。異なるコマンド・ファイルを指定するつもりならば，このビットをセットしなければなりません。このオプションは，/COMMAND 修飾子の存在を示すのに XTPU$INITIALIZE に渡されるオプション・ビットと併用します。
XTPU$M_EXECUTE_PROC	次に XTPU$EXECUTE_INIFILE が呼び出されたときに，XTPU$INIT_PROCEDURE を検索し，それを実行します。
XTPU$M_KILL_PROCESSES	セッションで生成されたすべてのサブプロセスを削除します。
XTPU$M_LAST_TIME	このビットは，DEC XTPU に対する最後の呼び出しである場合にだけセットしなければなりません。このビットをセットした後に再度 DEC XTPU を呼び出した場合，結果は予測できません。
XTPU$M_PRUNE_CACHE	バッファにページが割り当てられていない仮想ファイル・マネージャ・キャッシュをすべて解放します。このオプションは，セッションで作成されたあと，現在は不要になったキャッシュを解放します。
XTPU$M_RESET_TERMINAL	ターミナルを，DEC XTPU を起動したときの状態にリセットします。ターミナル・メールボックスおよびすべてのウィンドウが削除されます。リセットされたターミナルは，次に XTPU$INITIALIZE を呼び出したときに初期化し直されます。

[1]プリフィックスは XTPU$M_または XTPU$V_ です。 XTPU$M_ は，ビットがセットされるマスクが特定のフィールドに対応することを示しています。 XTPU$V_ はビット番号です。

説明

クリーンアップ・ルーチン XTPU$CLEANUP は，DEC XTPU とのやりとりの最後に呼び出すルーチンです。このルーチンは DEC XTPU に対し，内部データ構造をクリーンアップし，次の呼び出しに備えるよう指示します。上記のフラグをセットまたはクリアすることで，このルーチンによってリセットされるものを制御することができます。

DEC XTPU の使用を終了する場合には，このルーチンを呼び出すことにより，メモリを解放し，ターミナルの設定を初期状態に戻しておかなければなりません。

XTPU$CLEANUP を呼び出したあと，すぐにユーザ・アプリケーションを終了する場合には，データ構造を削除しないでください。オペレーティング・システムがそれを自動的に削除します。データ構造の削除をオペレーティング・システムに任せることで，プログラムの性能が向上します。

注意

1. **単純なインタフェースを使用する場合には，** DEC XTPU **は以下のフラグを自動的にセットします。**
   - XTPU$V_DELETE_BUFFERS
   - XTPU$V_DELETE_EXITH
   - XTPU$V_DELETE_JOURNAL
   - XTPU$V_DELETE_WINDOWS
   - XTPU$V_EXECUTE_FILE
   - XTPU$V_EXECUTE_PROC
   - XTPU$V_KILL_PROCESSES
   - XTPU$V_PRUNE_CACHE
   - XTPU$V_RESET_TERMINAL
2. **このルーチンがサクセス・ステータスを戻さなかった場合には，再度エディタを呼び出さないようにしなければなりません。**

戻される条件値

XTPU$_SUCCESS	正常終了したことを示します。

XTPU$CLIPARSE

コマンド・ラインを解析し， XTPU$INITIALIZE 用のアイテム・リストを作成します。

このルーチンは， CLI$DCL_PARSE を呼び出し，解析するコマンドとコマンド・テーブルを設定します。次に XTPU$PARSEINFO を呼び出し， XTPU$INITIALIZE 用のアイテム・リストを作成します。

ユーザのアプリケーションがコマンド・ラインを解析して DEC XTPU の操作とは無関係な情報を得る場合， XTPU$CLIPARSE を呼び出す前に，そうした情報をすべて取り出して使用してしまう必要があります。 XTPU$CLIPARSE は呼び出される前に格納されていた解析情報をすべて破壊してしまうからです。

形式

XTPU$CLIPARSE *string, fileio, call_user*

戻り値

OpenVMS **用法** **アイテム・リスト**
データ型 **符号なしロングワード**
アクセス **読み取りのみ**
受け渡し方 **参照による**

このルーチンはアイテム・リストのアドレスを返します。

引数

string

OpenVMS **用法** **文字列**
データ型 **文字列**
アクセス **読み取りのみ**
受け渡し方 **ディスクリプタによる**

コマンド・ラインです。string 引数は， DEC XTPU コマンドのディスクリプタのアドレスです。

fileio

OpenVMS **用法** **符号なしロングワード・ベクタ**
データ型 **バウンド・プロシージャ値**
アクセス **読み取りのみ**

受け渡し方　　ディスクリプタによる

ファイル入出力ルーチンです。fileio 引数は，ファイル入出力ルーチンのデスクリプタのアドレスです。

call_user

OpenVMS **用法　符号なしロングワード・ベクタ**
データ型　　バウンド・プロシージャ値
アクセス　　読み取りのみ
受け渡し方　　ディスクリプタによる

USER ルーチンです。call_user 引数は， USER ルーチンのディスクリプタのアドレスです。

XTPU$CLOSE_TERMINAL

このルーチンはターミナルへのチャネルをクローズします。

形式

XTPU$CLOSE_TERMINAL

戻り値

OpenVMS **用法　条件値**
データ型　符号なしロングワード
アクセス　書き込みのみ
受け渡し方　値による

ロングワードの条件値です。ユーティリティ・ルーチンのほとんどは，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値については， "戻される条件値 "にまとめられています。

引数

なし

説明

このルーチンは， CALL_USER 組込みプロシージャおよびそれに結び付けられた USER ルーチンにおいて， DEC XTPU のターミナルへのアクセスを制御するのに用いられます。 USER ルーチンが XTPU$CLOSE_TERMINAL を呼び出すと， DEC XTPU はターミナルへのチャネルと付随するメールボックスのチャンネルとをクローズします。

USER ルーチンから制御が戻されると， DEC XTPU は自動的にターミナルへのチャネルを再オープンし，他のウィンドウで隠されていないウィンドウを再表示します。

USER ルーチンは，そのプログラム中でいつでも XTPU$CLOSE_TERMINAL を使用することができます。また，必要に応じて何度でも使用することができます。 XTPU$CLOSE_TERMINAL を使用したときに，すでにターミナルがクローズされている場合は，その呼び出しは無視されます。

戻される条件値

XTPU$_SUCCESS	正常終了したことを示します。

XTPU$CONTROL

DEC XTPU エディタのメイン処理ルーチンです。このルーチンは，テキストとコマンドを読み取り，それを実行します。このルーチンを呼び出すと (XTPU$INITIALIZE を呼び出したあと)，制御は DEC XTPU に渡されます。

形式

XTPU$CONTROL *[integer]*

戻り値

OpenVMS **用法** **条件値**
データ型 **符号なしロングワード**
アクセス **書き込みのみ**
受け渡し方 **値による**

ロングワードの条件値です。ユーティリティ・ルーチンはほとんど，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値は，"戻される条件値 "にまとめられています。

引数

integer

OpenVMS **用法** **整数**
データ型 **符号なしロングワード**
アクセス **読み取りのみ**
受け渡し方 **参照による**

呼び出しプログラムが DEC XTPU に制御を渡した時に，"Editing Session is not being journaled"というメッセージが表示されないようにします。将来のバージョンとの互換性を保つために，真 (奇数) の整数を指定してください。引数を省略した場合，ジャーナリングされていないときには，DEC XTPU はメッセージを表示します。

説明

このルーチンは編集セッションを制御します。このルーチンは，テキストとコマンドを読み取り，それを実行します。実行された編集内容を示すように，スクリーン上のウィンドウが更新されます。 XTPU$SPECIFY_ASYN_ACTION ルーチンを XTPU$TRIGGER_ASYN_ACTION ルーチンとともに使って， DEC XTPU を中断しユーザ・プログラムがコントロールを取り戻すようにすることもできます。

注意

制御がユーザ・プログラムに戻されるのは，エラーが発生した場合，または QUIT **組込みプロシージャか** EXIT **組込みプロシージャを実行したあとだけです。**

戻される条件値

XTPU$_EXITING	EXIT の結果として戻されます (省略時の条件ハンドラが設定されている場合)。
XTPU$_NONANSICRT	処理の中断の結果として戻されます。この条件値が戻されるのは， XTPU$_DISPLAYFILE を NODISPLAY に設定して DEC XTPU を呼び出し，スクリーン関係のコマンドを実行しようとしたときです。
XTPU$_QUITTING	QUIT の結果として戻されます (省略時の条件ハンドラが設定されている場合)。
XTPU$_RECOVERFAIL	回復操作が異常終了したことを示します。

XTPU$EDIT

このルーチンは，パラメータからコマンド文字列を作成し，そのコマンド文字列を XTPU$XTPU ルーチンに渡します。

XTPU$EDIT は，もうひとつの DEC XTPU の単純な呼び出し可能インタフェースのエントリ・ポイントです。

形式

XTPU$EDIT *input, output*

戻り値

OpenVMS **用法　条件値**
データ型　符号なしロングワード
アクセス　書き込みのみ
受け渡し方　値による

ロングワードの条件値です。ユーティリティ・ルーチンはほとんど，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値については，"戻される条件値 "にまとめられています。

引数

input

OpenVMS **用法　文字列**
データ型　文字列
アクセス　読み取りのみ
受け渡し方　ディスクリプタによる

入力ファイル名です。input 引数は，ファイル指定のディスクリプタのアドレスです。

output

OpenVMS **用法　文字列**
データ型　文字列
アクセス　読み取りのみ
受け渡し方　ディスクリプタによる

出力ファイル名です。output 引数は，出力ファイル指定のディスクリプタのアドレスです。これは/OUTPUT コマンド修飾子で使用される文字列です。

説明

このルーチンはコマンド文字列を作成し，それを XTPU$XTPU に渡します。出力ファイル名の長さがゼロでなければ，コマンド文字列に/OUTPUT 修飾子を付加します。

ユーザ・アプリケーションがコマンド・ラインを解析して DEC XTPU の操作とは無関係な情報を得る場合， XTPU$EDIT を呼び出す前に，そうした情報をすべて取り出して使用してしまう必要があります。 XTPU$EDIT は呼び出される前に格納されていた解析情報をすべて破壊してしまうからです。

戻される条件値

XTPU$XTPU から戻される値。

XTPU$EXECUTE_COMMAND

このルーチンは，ユーザ・プログラムが DEC XTPU ステートメントを実行できるようにします。

形式

XTPU$EXECUTE_COMMAND *string*

戻り値

OpenVMS **用法　条件値**
データ型　符号なしロングワード
アクセス　書き込みのみ
受け渡し方　値による

ロングワードの条件値です。ユーティリティ・ルーチンはほとんど，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値は， "戻される条件値 "にまとめられています。

引数

string

OpenVMS **用法　文字列**
データ型　文字列
アクセス　読み取りのみ
受け渡し方　ディスクリプタによる

DEC XTPU ステートメントです。string 引数は， 1 つかまたはそれ以上の DEC XTPU ステートメントを示す文字列のディスクリプタのアドレスです。

説明

このルーチンは，『DEC Text Processing Utility Reference Manual』に説明されている EXECUTE 組込みプロシージャと同じ機能を実行します。

戻される条件値

XTPU$_SUCCESS	正常終了したことを示します。
XTPU$_EXITING	EXIT 組込みプロシージャが呼び出されたことを示します。
XTPU$_QUITTING	QUIT 組込みプロシージャが呼び出されたことを示します。
XTPU$_EXECUTEFAIL	実行が異常終了したことを示します。これは，実行エラーまたはコンパイラ・エラーが発生したためです。

XTPU$EXECUTE_INIFILE

このルーチンは，ユーザがユーザ作成初期化ファイルを実行できるようにします。

このルーチンは，エディタを初期化したあとで，他のコマンドを処理する前に実行しなければなりません。

形式

XTPU$EXECUTE_INIFILE

戻り値

OpenVMS **用法　条件値**
データ型　符号なしロングワード
アクセス　書き込みのみ
受け渡し方　値による

ロングワードの条件値です。ユーティリティ・ルーチンはほとんど，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値は，"戻される条件値 "にまとめられています。

引数

なし

説明

XTPU$EXECUTE_INIFILE ルーチンを呼び出すと， DEC XTPU は以下の手順で実行します。

1. バッファにコマンド・ファイルが読み込まれます。省略時のファイルは XTPU$COMMAND.TPU です。もし，コマンド・ラインで指定したファイルが見つからなかった場合には，エラー・メッセージが表示され，ルーチンは強制終了されます。
2. /DEBUG 修飾子をコマンド・ラインで指定した場合には，バッファに DEBUG ファイルが読み込まれます。省略時のファイルは SYS$SHARE:XTPU$DEBUG .TPU です。

3. DEBUG ファイルをコンパイルし，実行します。
4. TPU$INIT_PROCEDURE を実行します。
5. コマンド・バッファをコンパイルし，実行します。
6. TPU$INIT_POSTPROCEDURE を実行します。

注意

XTPU$CLEANUP を呼び出したあと，このルーチンを呼び出す場合には，XTPU$_EXECUTEPROCEDURE フラグと XTPU$_EXECUTEFILE フラグをセットしなければなりません。これらのフラグをセットしなかった場合には，初期化ファイルは実行されません。

戻される条件値

XTPU$_SUCCESS	正常終了。
XTPU$_EXITING	EXIT の結果として戻されます。省略時の条件ハンドラを使用している場合には，セッションは終了します。
XTPU$_QUITTING	QUIT の結果として戻されます。省略時の条件ハンドラを使用している場合には，セッションは終了します。
XTPU$_COMPILEFAIL	初期化ファイルのコンパイルが異常終了したことを示します。
XTPU$_EXECUTEFAIL	初期化ファイルに含まれるステートメントの実行が異常終了したことを示します。
XTPU$_FAILURE	他のすべてのエラーを示す汎用コード。

XTPU$FILEIO

このルーチンはファイル操作を処理します。ユーザが作成したファイル入出力ルーチンは，何らかの操作を実行するためにこのルーチンを呼び出すことができます。しかし，ファイルをオープンするルーチンは，そのファイルに対してすべての操作を実行しなければなりません。たとえば，XTPU$FILEIO がファイルをオープンする場合には，そのファイルをクローズする操作も実行しなければなりません。

形式

XTPU$FILEIO *code, stream, data*

戻り値

OpenVMS **用法** **条件値**
データ型 **符号なしロングワード**
アクセス **書き込みのみ**
受け渡し方 **値による**

ロングワードの条件値です。ユーティリティ・ルーチンはほとんど，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値は，"戻される条件値 "にまとめられています。

引数

code

OpenVMS **用法** **符号なしロングワード**
データ型 **符号なしロングワード**
アクセス **読み取りのみ**
受け渡し方 **参照による**

ある DEC XTPU 機能を指定するアイテム・コードです。code 引数は，実行する機能を指定する DEC XTPU からのアイテム・コードを含むロングワードのアドレスです。

ファイル入出力ルーチンに指定できるアイテム・コードは，以下のとおりです。

- XTPU$K_OPEN ―このアイテム・コードは，data 引数がアイテム・リストのアドレスであることを指定します。このアイテム・リストには，ファイルをオープンするのに必要な情報が含まれます。stream 引数には，このファイルを将来参照するために使用される固有の値を指定しなければなりません。この結果作成され

たファイル名は，動的な文字列ディスクリプタといっしょにコピーしなければなりません。

- XTPU$K_CLOSE ― stream 引数によって指定されるファイルがクローズされます。構造が使用しているメモリをすべて解放することができます。
- XTPU$K_CLOSE_DELETE ― stream 引数によって指定されるファイルがクローズされ，削除されます。構造が使用しているメモリをすべて解放することができます。
- XTPU$K_GET ―data 引数は，stream 引数によって指定されるファイルの，次のレコードが格納される動的な文字列ディスクリプタのアドレスです。このルーチンは，テキストをこのディスクリプタにコピーするために， OpenVMS ランタイム・ライブラリが提供するルーチンを使用しなければなりません。 DEC XTPU は，ファイルの最後に到達したことをファイル入出力ルーチンが示したときに，読み取られるデータのために割り当てられたメモリを解放します。
- XTPU$K_PUT ―data 引数は，stream 引数によって指定されるファイルに書き込まれるデータのディスクリプタのアドレスです。

stream

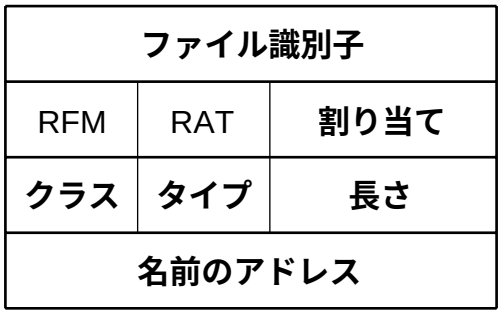
OpenVMS **用法** **不定**
データ型 **符号なしロングワード**
アクセス **変更**
受け渡し方 **参照による**

ファイル指定です。stream 引数は， 4 つのロングワードで構成されるデータ構造のアドレスです。このデータ構造は，操作されるファイルを記述するために使用されます。

このデータ構造は，すべてのファイルを参照するために使用されます。ファイル・オープン要求が実行されたときには，このデータ構造にデータが書き込まれます。他の要求はすべて，この構造の情報を使って，どのファイルが参照されているかを判断します。

図 6–2 はストリーム・データ構造を示しています。

図 6–2 ストリーム・データ構造



最初のロングワードは，各ファイルの固有の識別子を格納するために使用されます。ユーザ作成ファイル入出力ルーチンは， 0 から 511 までの値に制限されています。したがって，最大 512 のファイルを同時にオープンすることができます。

2 番目のロングワードは 3 つのフィールドに分かれています。下位ワードは， FAB (FAB$L_ALQ) から割り当てられるサイズ，つまり，このファイルに割り当てられるブロック数を格納するために使用されます。この値はあとで，ディスク空間を前もって割り当てるために出力ファイル・サイズを計算するときに使用されます。 2 ワード目の下位バイトは，既存のファイルをオープンするときに，レコード属性バイト (FAB$B_RAT) を格納するために使用されます。上位バイトは，既存のファイルをオープンするときに，レコード・フォーマット・バイト (FAB$B_RFM) を格納するために使用されます。これらは，入力ファイルと同じフォーマットで出力ファイルを作成するために使用されます。これらのフィールドには，ファイルをオープンするルーチンがデータを書き込みます。

最後の 2 つのロングワードは，作成または拡張されるファイル名のディスクリプタとして使用されます。この名前はあとで， EXIT を処理するときに使用されます。このディスクリプタには，オープン操作のあとで，ファイル名が書き込まれます。このディスクリプタは，ランタイム・ライブラリの LIB$SCOPY_R_DX ルーチンまたは LIB$SCOPY_DX ルーチンを使って割り当てなければなりません。この空間が不要になった場合には， DEC XTPU によって解放されます。

data

OpenVMS **用法** **アイテム・リスト** 3
データ型 **符号なしロングワード**
アクセス **変更**
受け渡し方 **参照による**

ストリーム・データです。data 引数はアイテム・リストのアドレス，またはディスクリプタのアドレスです。

注意

このパラメータの意味は，コード・フィールドに指定したアイテム・コードによって異なります。

XTPU$K_OPEN アイテム・コードを指定した場合には，data 引数は，オープン要求に関する情報を含むアイテム・リストのアドレスです。オープン要求に関する情報を指定する場合には，以下の DEC XTPU アイテム・コードを使用できます。

- XTPU$K_ACCESS －このアイテム・コードを使用すれば，バッファ・アドレス・フィールドに以下のいずれかのアイテム・コードを指定できます。
  - XTPU$K_IO
  - XTPU$K_INPUT
  - XTPU$K_OUTPUT
- XTPU$K_FILENAME －このアイテム・コードは，オープンしようとするファイル名として使用する文字列のアドレスを指定するために使用されます。長さフィールドには文字列の長さを指定し，アドレス・フィールドにはそのアドレスを指定します。
- XTPU$K_DEFAULTFILE －このアイテム・コードは，オープンしようとするファイルに省略時のファイル名を割り当てるために使用されます。バッファ長フィールドには長さを指定し，バッファ・アドレス・フィールドには省略時のファイル名のアドレスを指定します。
- XTPU$K_RELATEDFILE －このアイテム・コードは，オープンしようとするファイルの関連ファイル名を指定するために使用されます。バッファ長フィールドには長さを指定し，バッファ・アドレス・フィールドには，関連ファイル名として使用する文字列のアドレスを指定します。
- XTPU$K_RECORD_ATTR －このアイテム・コードは，ファイルを作成するために使用される FAB のレコード属性バイト (FAB$B_RAT) の値がバッファ・アドレス・フィールドに含まれていることを指定します。
- XTPU$K_RECORD_FORM －このアイテム・コードは，ファイルを作成するために使用される FAB のレコード・フォーマット・バイト (FAB$B_RFM) の値がバッファ・アドレス・フィールドに含まれていることを指定します。
- XTPU$K_MAXIMIZE_VER －このアイテム・コードは，出力ファイルのバージョン番号を現存する最大のバージョン番号より 1 つだけ大きな値にしなければならないことを指定します。
- XTPU$K_FLUSH －このアイテム・コードは，ファイルに書き込む操作のたびに各レコードが実際にファイルに書き出されることを指定します。
- XTPU$K_FILESIZE －このアイテム・コードは，ファイルを作成するときに，割り当てサイズとして使用される値を指定するために使用されます。値はバッファ・アドレス・フィールドに指定します。

説明

省略時の状態では，XTPU$FILEIO はキャリッジ・リターン・レコード属性を持った可変長ファイルを作成します (fab$b_rtm=var, fab$b_rat=cr)。XTPU$K_RECORD_ATTR アイテムまたは XTPU$K_RECORD_FORM アイテムを指定した場合には，それらのアイテムが使用されます。使用できるフォーマットと属性の組み合わせは以下のとおりです。

フォーマット	属性
STM, STMLF, STMCR	O, BLK, CR, BLK+CR
VAR	O, BLK, FTN, CR, BLK+FTN, BLK+CR

他の組み合わせはすべて，CR 属性を持った VAR フォーマットに変換されます。

このルーチンは常に，ストリーム・データ構造の最初のロングワードに 511 より大きな値をセットします。ユーザ作成ファイル入出力ルーチンは 0 ― 511 の値に制限されているため，このルーチンがセットするファイル制御ブロック (FCB) とユーザが作成したブロックは簡単に区別することができます。

注意

単純な呼び出し可能インタフェースを利用する場合，省略時の設定により，XTPU$FILEIO が使用されます。完全な呼び出し可能インタフェースを利用する場合は，明示的に XTPU$FILEIO を呼び出すか，またはユーザが独自に入出力ルーチンを用意しなければなりません。

戻される条件値

ファイル入出力ルーチンは，RMS ステータス・コードを DEC XTPU に戻します。エラー・メッセージが必要な場合には，ファイル入出力ルーチンがすべてのエラーを通知しなければなりません。

XTPU$HANDLER

このルーチンは DEC XTPU の条件ハンドラです。 DEC XTPU 条件ハンドラは, Put Message($PUTMSG) システム・サービスを呼び出し, XTPU$MESSAGE のアドレスを渡します。

形式

XTPU$HANDLER *signal_vector, mechanism_vector*

戻り値

OpenVMS **用法** **条件値**
データ型 **符号なしロングワード**
アクセス **書き込みのみ**
受け渡し方 **値による**

ロングワードの条件値です。

引数

signal_vector

OpenVMS **用法** **引数リスト**
データ型 **符号なしロングワード**
アクセス **変更**
受け渡し方 **参照による**

シグナル・ベクタです。条件ハンドラに渡されるシグナル・ベクタについての説明は, 『OpenVMS System Services Reference Manual』を参照してください。

mechanism_vector

OpenVMS **用法** **引数リスト**
データ型 **符号なしロングワード**
アクセス **読み取りのみ**
受け渡し方 **参照による**

条件ハンドラに渡されるメカニズム・ベクタについての説明は, 『OpenVMS System Services Reference Manual』を参照してください。

説明

XTPU$MESSAGE ルーチンは実際のメッセージ出力を実行します。 Put Message ($PUTMSG) システム・サービスはメッセージを書式整形するだけです。このルーチンは，第 6.1.2 項で説明した変数からメッセージ・フラグの設定と機能名を入手します。これらの値を変更できるのは， DEC XTPU の SET 組込みプロシージャだけです。

ハンドラが受け取った条件値に FATAL ステータスが含まれている場合や， DEC XTPU のファシリティ・コードが含まれていない場合には，条件が再通知されます。

条件が XTPU$_QUITTING， XTPU$_EXITING，または XTPU$_RECOVERFAIL の場合には，条件ハンドラを設定したルーチンに対して， UNWIND 要求が与えられます。

メッセージを処理したあと，条件ハンドラは継続ステータスを戻します。 DEC XTPU エラー・メッセージ要求は，どのメッセージを出力しなければならないかを示す条件を通知することによって実行されます。シグナル・アレイに含まれる引数は，正しく書式整形されたメッセージ引数ベクタです。このベクタには，関連するメッセージに対する複数の条件と書式整形された ASCII 出力 (FAO) 引数が含まれていることがあります。たとえば，存在しないファイルをエディタがオープンしようとした場合には， DEC XTPU のメッセージとして XTPU$_NOFILEACCESS が通知されます。このメッセージに対する FAO 引数は，ファイルの名前を示す文字列です。この条件にはエラー・ステータスが含まれており，そのあとに RMS のステータス・フィールド (STS) とステータス値フィールド (STV) が続きます。この条件には回復不可能な重大なエラーは含まれていないので，エラーを処理したあと，エディタは処理を継続できます。

エディタは XTPU$CONTROL から自動的に戻るわけではありません。したがって，XTPU$CONTROL ルーチンを呼び出す場合には，制御をエディタに戻すための方法を設定しておかなければなりません (たとえば， CALL_USER 組込みプロシージャを使用します)。また，ユーザ作成条件ハンドラを設定したものの，特定の条件に対しては DEC XTPU ハンドラを呼び出す場合には，プログラムの中で制御を戻したい場所に，省略時の条件ハンドラを設定しておかなければなりません。また，XTPU$SPECIFY_ASYN_ACTION および XTPU$TRIGGER_ASYNC_ACTION の非同期ルーチンを使って， XTPU$CONTROL を中断することもできます。

『OpenVMS Calling Standard』を参照してください。

XTPU$INITIALIZE

このルーチンはテキスト処理のために DEC XTPU を初期化します。このルーチンはグローバル・データ構造を割り当て，グローバル変数を初期化し，仮想ファイル・マネージャやスクリーン・マネージャ，入出力サブシステムも含めて，エディタの主要コンポーネントのそれぞれに対して，適切なセットアップ・ルーチンを呼び出します。

形式

XTPU$INITIALIZE *callback [,user_arg]*

戻り値

OpenVMS **用法** **条件値**
データ型 **符号なしロングワード**
アクセス **書き込みのみ**
受け渡し方 **値による**

ロングワードの条件値です。ユーティリティ・ルーチンはほとんど，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値は，"戻される条件値 "にまとめられています。

引数

callback

OpenVMS **用法** **符号なしロングワード・ベクタ**
データ型 **バウンド・プロシージャ値**
アクセス **読み取りのみ**
受け渡し方 **ディスクリプタによる**

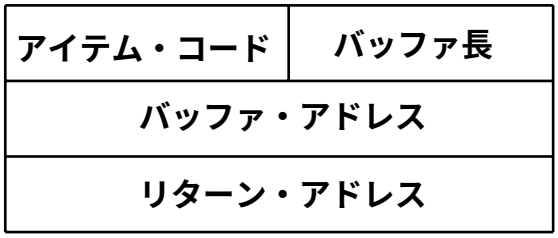
コールバック・ルーチンです。引数callbackは，初期化パラメータを含むアイテム・リストのアドレスを返すユーザ作成ルーチンか，またはファイル入出力操作を処理するルーチンのアドレスです。このコールバック・ルーチンは，コマンド・ライン解析ルーチン (XTPU$CLIPARSE か，またはユーザ作成の解析ルーチンのいずれか) を呼び出さなければなりません。

呼び出し可能な DEC XTPU は，初期化パラメータを指定するために使用できるアイテム・コードを定義しています。これらのコードを使用するときには，以下の基準に従ってください。

- XTPU$_OTHER_FILENAMES は，XTPU$_FILENAME の後で使わなければなりません。
- XTPU$_CHAIN または XTPU$_ENDLIST はリストの最後のアイテムでなければなりません。

図 6–3 は，アイテムディスクリプタの一般的な形式を示しています。アイテム・リストの作成方法についての詳しい説明は，使用する各言語のプログラマ・マニュアルを参照してください。

図 6–3 アイテムディスクリプタのフォーマット



アイテムディスクリプタのリターン・アドレスは，通常 0 です。

使用できるアイテム・コードは表 6–2 に示すとおりです。

表 6–2 使用できるアイテム・コード

アイテム・コード	説明
XTPU$_OPTIONS	コマンド修飾子を使用可能にします。バッファ・アドレス・フィールドの 14 ビットは，いろいろな DEC XTPU コマンド修飾子に対応しています。バッファ・アドレス・フィールドの残りの 18 ビットは使用されません。
XTPU$_JOURNALFILE	/JOURNAL 修飾子に指定されている文字列を渡します。バッファ長フィールドは文字列の長さであり，バッファ・アドレス・フィールドは文字列のアドレスです。これは， GET_INFO (COMMAND_LINE,"JOURNAL_FILE") で使用できる文字列です。この文字列はヌル文字列でもかまいません。
XTPU$_SECTIONFILE	マッピングされるバイナリ初期化ファイル (セクション・ファイル) の名前を示す文字列を渡します。バッファ長フィールドは文字列の長さであり，バッファ・アドレス・フィールドは文字列のアドレスです。 DEC XTPU の CLD ファイルには，この文字列の省略時の値が含まれています。 XTPU$V_SECTION ビットがセットされている場合には，このアイテム・コードを指定しなければなりません。

(次ページに続く)

表 6–2 (続き) 使用できるアイテム・コード

アイテム・コード	説明
XTPU$_OUTPUTFILE	/OUTPUT 修飾子に指定されている文字列を渡します。バッファ長フィールドは文字列の長さであり，バッファ・アドレス・フィールドは文字列のアドレスを指定します。これは， GET_INFO (COMMAND_LINE,"OUTPUT_FILE") 組込みプロシージャから戻される文字列です。文字列はヌル文字列でもかまいません。
XTPU$_DISPLAYFILE	/DISPLAY 修飾子に指定されている文字列を渡します。バッファ長フィールドは文字列の長さであり，バッファ・アドレス・フィールドは文字列のアドレスを指定します。
XTPU$_COMMANDFILE	/COMMAND 修飾子に指定されている文字列を渡します。バッファ長フィールドは文字列の長さであり，バッファ・アドレス・フィールドは文字列のアドレスです。この文字列は， GET_INFO (COMMAND_LINE,"COMMAND_FILE") 組込みプロシージャから戻される文字列です。文字列はヌル文字列でもかまいません。
XTPU$_FILENAME	コマンド・ラインに指定されている入力ファイルの名前を示す文字列を渡します。バッファ長フィールドはこの文字列の長さを指定し，バッファ・アドレス・フィールドはアドレスを指定します。これは， GET_INFO (COMMAND_LINE,"FILE_NAME") 組込みプロシージャから戻される文字列です。このファイル名はヌル文字列でもかまいません。
XTPU$_OTHER_FILENAMES	コマンド・ラインに指定されている入力ファイルのうち一番目を除くものの名前を示す文字列を渡します。バッファ長フィールドはこの文字列の長さを指定し，バッファ・アドレス・フィールドはアドレスを指定します。 2 番目以降の入力ファイルはそれぞれ XTPU$_OTHER_FILENAMES のエントリが必要です。これは，GET_INFO (COMMAND_LINE,"NEXT_FILE_NAME") 組込みプロシージャから戻される文字列です。
XTPU$_FILEIO	ファイル操作を処理するために使用されるルーチンのバウンド・プロシージャ値を渡します。自分で作成したファイル入出力ルーチンを使用することもファイル操作を処理するために DEC XTPU が提供するユーティリティ・ルーチンである XTPU$FILEIO を呼び出すこともできます。ファイル入出力ルーチンのアドレスは，バッファ・アドレス・フィールドに指定します。
XTPU$_CALLUSER	CALL_USER 組込みプロシージャが呼び出すユーザ作成ルーチンのバウンド・プロシージャ値を渡します。このルーチンのアドレスはバッファ・アドレス・フィールドに指定します。
XTPU$_INIT_FILE	/INITIALIZATION 修飾子に指定されている文字列を渡します。バッファ長フィールドは文字列の長さであり，バッファ・アドレス・フィールドは文字列のアドレスです。この文字列は， GET_INFO (COMMAND_LINE, "INIT_FILE") 組込みプロシージャから戻される文字列です。

(次ページに続く)

表 6–2 (続き) 使用できるアイテム・コード

アイテム・コード	説明
XTPU$_KANJI_DICFILE	/KANJI_DICTIONARY 修飾子に指定されている文字列を渡します。バッファ長フィールドはこの文字列の長さを指定し，バッファ・アドレス・フィールドはアドレスを指定します。これは， GET_INFO (COMMAND_LINE, "KANJI_DICTIONARY_FILE") 組込みプロシージャから戻される文字列です。
XTPU$_START_LINE	その編集セッションの開始行番号を渡します。バッファ・アドレス・フィールドは/START_POSITION 修飾子で指定した 2 つの整数値の最初の値です。この値は GET_INFO (COMMAND_LINE,"LINE") 組込みプロシージャから戻される値です。通常，初期化プロシージャがこの情報を用いて，編集するバッファの先頭位置を設定します。バッファの先頭行は line 1 です。
XTPU$_START_CHAR	その編集セッションの開始カラム位置を渡します。バッファ・アドレス・フィールドは/START_POSITION 修飾子で指定した 2 つの整数値の 2 番目の値です。この値は GET_INFO (COMMAND_LINE,"CHARACTER") 組込みプロシージャから戻される値です。通常，初期化プロシージャがこの情報を用いて，編集するバッファの先頭位置をを設定します。バッファの先頭カラムは character 1 です。
XTPU$_WORKFILE	/WORK 修飾子で指定される文字列を渡します。バッファ長フィールドはこの文字列の長さを指定し，バッファ・アドレス・フィールドはアドレスを指定します。これは， GET_INFO (COMMAND_LINE, "WORK_FILE") 組込みプロシージャから戻される文字列です。
XTPU$_CHAIN	バッファ・アドレス・フィールドに次のアイテム・リストのアドレスを指定します。
XTPU$_ENDLIST	アイテム・リストの最後を示します。
XTPU$_CTRL_C_ROUTINE	Ctrl/C AST を処理するためのルーチンのバウンド・プロシージャ値を渡します。 DEC XTPU は Ctrl/C AST が発生すると，そのルーチンを呼び出します。そのルーチンが FALSE 値を返すと， DEC XTPU は Ctrl/C が処理されたものと仮定します。 TRUE 値が返されると， DEC XTPU は現在実行している DEC XTPU プロシージャの実行を強制終了させます。バッファ・アドレス・フィールドは， 2 つのロングワード・ベクタのアドレスを指定しています。最初のロングワードはルーチンのアドレスであり， 2 番目のロングワードはそのルーチンを呼ぶ前に R1 に DEC XTPU がロードする環境値です。
XTPU$_DEBUGFILE	/DEBUG 修飾子で指定された文字列を渡します。バッファ長フィールドは文字列の長さであり，バッファ・アドレス・フィールドは文字列のアドレスです。

表 6–3 は， XTPU$K_OPTIONS アイテム・コードによって与えられるビットと対応するマスクをまとめたものです。

表 6–3 XTPU$OPTIONS アイテム・コードによって与えられるビットと対応するマスク

マスク	フラグ	機能
XTPU$M_RECOVER[1]	XTPU$V_RECOVER[2]	回復操作を実行します。
XTPU$M_JOURNAL	XTPU$V_JOURNAL	編集セッションをジャーナルします。
XTPU$M_READ	XTPU$V_READ	メイン・バッファに対しては READ_ONLY 編集セッションとします。
XTPU$M_SECTION	XTPU$V_SECTION	起動時にバイナリ初期化ファイル (DEC XTPU セクション・ファイル) にマッピングします。
XTPU$M_CREATE	XTPU$V_CREATE	指定した入力ファイルが存在しない場合，入力ファイルを作成します。
XTPU$M_OUTPUT	XTPU$V_OUTPUT	セッションを終了するときに，変更された入力ファイルを書き出します。
XTPU$M_COMMAND	XTPU$V_COMMAND	起動時にコマンド・ファイルを実行します。
XTPU$M_DISPLAY	XTPU$V_DISPLAY	スクリーンを使用する編集と表示のために，ターミナルを使用します。
XTPU$M_INIT	XTPU$V_INIT	イニシャライゼーション・ファイルがあります。
XTPU$M_COMMAND_DFLTED	XTPU$V_COMMAND_DFLTED	コマンド・ラインのユーザが指定した省略値の名前があるかどうかを指示します。これが TRUE である場合には，ユーザがコマンド・ファイルを指定しなかったことを意味します。このビットが FALSE にセットされていて，かつファイルが指定されていなかったら，XTPU$INITIALIZE は失敗します。
XTPU$M_WRITE	XTPU$V_WRITE	/WRITE がコマンド・ラインで指定されたかどうかを示します。
XTPU$M_MODIFY	XTPU$V_MODIFY	/MODIFY がコマンド・ラインで指定されたかどうかを示します。
XTPU$M_NOMODIFY	XTPU$V_NOMODIFY	/NOMODIFY がコマンド・ラインで指定されたかどうかを示します。
XTPU$M_NOKANJI_DIC	XTPU$V_NOKANJI_DIC	/NOKANJI_DICTIONARY がコマンド・ラインで指定されたかどうかを示します。
XTPU$M_SEC_LNM_MODE	XTPU$V_SEC_LNM_MODE	イメージが特権付きでインストールされ，このビットがセットされていると，セクション・ファイルを開けるときに信用のある論理名のみが使用されます。

[1]XTPU$M... マスクを示します。

[2]XTPU$V... ビット・アイテムを示します。

フラグを作成するには，0 という値から開始し，セットしたい各アイテムのマスク (XTPU$M …) に対して， OR 演算子を使用します。また，フラグを作成する別の方法として， 32 ビットをビット・ベクタとして取り扱い，希望するアイテムに対応するビット (XTPU$V …) をセットする方法もあります。

user_arg

OpenVMS **用法　ユーザ引数**
データ型　符号なしロングワード
アクセス　読み取りのみ
受け渡し方　値による

ユーザ引数。user_arg 引数はユーザによって書かれた初期化ルーチン INITIALIZE に渡されます。

user_arg 引数によって，アプリケーションは XTPU$INITIALIZE を通してユーザによって書かれた初期化ルーチンに情報を渡すことができます。 DEC XTPU はこのデータを解釈しません。

説明

このルーチンは，条件ハンドラを設定したあとで最初に呼び出さなければならないルーチンです。

このルーチンは，コールバック・ルーチンから受け取った情報に応じてエディタを初期化します。初期化ルーチンは，省略時の値として，すべてファイル指定をヌル文字列に設定し，すべてのオプションをオフの状態に設定します。しかし，ファイル入出力ルーチンや USER ルーチンの省略時のアドレスは設定しません。

セクション・ファイルを指定しなかった場合には，エディタのソフトウェア機能は制限されます。

戻される条件値

XTPU$_SUCCESS	初期化が正常終了したことを示します。
XTPU$_SYSERROR	システム・サービスが正しく機能しなかったことを示します。
XTPU$_NONANSICRT	入力装置 (SYS$INPUT) がサポートされないターミナルであることを示します。
XTPU$_RESTOREFAIL	復元操作でエラーが発生したことを示します。
XTPU$_NOFILEROUTINE	ファイル操作を実行するためのルーチンが設定されていないことを示します。
XTPU$_OPENDIC	かな漢字変換辞書がオープンできなかったことを示します。
XTPU$_INSVIRMEM	エディタが初期化を実行するのに十分な仮想メモリが存在しないことを示します。
XTPU$_FAILURE	初期化を実行しているときに発生した他のすべてのエラーを示す汎用コードです。

XTPU$MESSAGE

このルーチンは，MESSAGE 組込みプロシージャを使って，エラー・メッセージや文字列を出力します。

このルーチンを使えば，DEC XTPU と同じ方法で，メッセージを出力し，処理することができます。このルーチンは，XTPU$EXECUTE_INFILE を実行したあとで使用しなければなりません。

形式

XTPU$MESSAGE *string*

戻り値

OpenVMS **用法　条件値**
データ型　符号なしロングワード
アクセス　書き込みのみ
受け渡し方　値による

ロングワードの条件値です。

注意

返される値は，Put Message($PUTMSG) **システム・サービスによる使用を意図したものであり，無視して下さい。**

引数

string

OpenVMS **用法　文字列**
データ型　文字列
アクセス　読み取りのみ
受け渡し方　ディスクリプタによる

書式整形されたメッセージです。string 引数は，出力されるテキストのディスクリプタのアドレスです。これは書式整形を終えておかなければなりません。このルーチンはメッセージ・プリフィックスを追加しません。しかし，メッセージ・バッファが存在する場合には，テキストがメッセージ・バッファに追加されます。さらに，バッファがウィンドウにマッピングされている場合には，そのウィンドウは更新されます。

XTPU$PARSEINFO

このルーチンはコマンドを解析し，XTPU$INITIALIZE 用のアイテム・リストを作成します。

形式

XTPU$PARSEINFO *fileio, call_user*

戻り値

OpenVMS **用法** **アイテム・リスト**
データ型 **符号なしロングワード**
アクセス **読み取りのみ**
受け渡し方 **参照による**

このルーチンは，アイテム・リストのアドレスを返します。

引数

fileio

OpenVMS **用法** **符号なしロングワード・ベクタ**
データ型 **バウンド・プロシージャ値**
アクセス **読み取りのみ**
受け渡し方 **ディスクリプタによる**

ファイル入出力ルーチンです。fileio 引数は，ファイル入出力ルーチンのディスクリプタのアドレスです。

call_user

OpenVMS **用法** **符号なしロングワード・ベクタ**
データ型 **バウンド・プロシージャ値**
アクセス **読み取りのみ**
受け渡し方 **ディスクリプタによる**

USER ルーチンです。call_user 引数は，USER ルーチンのディスクリプタのアドレスです。

説明

XTPU$PARSEINFO ルーチンはコマンドを解析し， XTPU$INITIALIZE 用のアイテム・リストを作成します。

このルーチンは，コマンド言語インタプリタ (CLI) ルーチンを使って現在のコマンドを解析し， DEC XTPU の期待するパラメータや修飾子について問い合わせます。問い合わせた結果はアイテム・リストに適切な情報を設定するのに使用されます。 2 つのユーザ・ルーチンのアドレスは，それぞれリスト中にアイテムとして格納されます。このリストのアドレスが XTPU$PARSEINFO ルーチンから返されます。

ユーザのアプリケーションが DEC XTPU の動作とは無関係な情報を解析する場合には，アプリケーションは XTPU$PARSEINFO を呼び出す前にそれらの情報をすべて使用してしまわなければなりません。 XTPU$PARSEINFO はそれが呼びだされる前に格納された解析情報をすべて破壊してしまうからです。

XTPU$SPECIFY_ASYNC_ACTION

このルーチンは DEC XTPU の完全な呼び出し可能インタフェースを使用したアプリケーションが DEC XTPU に非同期アクションを登録するときに使用されます。

形式

XTPU$SPECIFY_ASYNC_ACTION *facility_index ,[xtpu_statement]*

戻り値

OpenVMS **用法　条件値**
データ型　符号なしロングワード
アクセス　書き込みのみ
受け渡し方　値による

ロングワードの条件値です。ユーティリティ・ルーチンはほとんど，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値は， "戻される条件値 "にまとめられています。

引数

facility_index

OpenVMS **用法　符号なしロングワード**
データ型　符号なしロングワード
アクセス　読み取りのみ
受け渡し方　参照による

非同期アクションのインデックス番号を指定します。このインデックス番号は，XTPU$TRIGGER_ASYNC_ACTION ルーチンに渡し， DEC XTPU に実行すべきアクションをを識別させるのに使用します。また (引数xtpu_statementを省略することによって) アクションを削除するのにも使用します。非同期アクションは，必要に応じて複数登録することができます。インデックス番号は任意の正の整数です。

xtpu_statement

OpenVMS **用法　文字列**
データ型　文字列
アクセス　読み取りのみ
受け渡し方　ディスクリプタによる

XTPU$TRIGGER_ASYNC_ACTION が呼ばれたときに実行される DEC XTPU ステートメントです。ステートメントはコンパイルされ内部に格納されます。この引数

を省略した場合は DEC XTPU は非同期イベントのリストからアクションを削除します。

説明

XTPU$SPECIFY_ASYNC_ACTION ルーチンは，XTPU$TRIGGER_ASYNC_ACTION とともに使用することにより，XTPU$CONTROL を呼び出した後に DEC XTPU を中断させることを可能にします。指定された DEC XTPU ステートメントはコンパイルされ，保存されます。

このルーチンは，XTPU$INITIALIZE の後に呼び出さなければなりません。キーストローク・ジャーナリングが有効になっていると，正常終了しません。

戻される条件値

XTPU$_SUCCESS	正常終了したことを示します。
XTPU$_COMPILEFAIL	xtpu_statement 引数に指定されたコードが正常にコンパイルできません。
XTPU$_INVPARM	引数が正しくありません。
XTPU$_JNLACTIVE	キー・ジャーナリングが動作中です。このルーチンを正しく動作させるためにはバッファ・ジャーナリングが必要です。

XTPU$TRIGGER_ASYNC_ACTION

このルーチンは，DEC XTPU の完全な呼び出し可能インタフェースを使用したアプリケーションが DEC XTPU の XTPU$CONTROL ループを非同期に中断させるときに使用されます。

形式

XTPU$TRIGGER_ASYNC_ACTION *facility_index*

戻り値

OpenVMS **用法** **条件値**
データ型 **符号なしロングワード**
アクセス **書き込みのみ**
受け渡し方 **値による**

ロングワードの条件値です。ユーティリティ・ルーチンはほとんど，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値は，"戻される条件値 "にまとめられています。

引数

facility_index

OpenVMS **用法** **符号なしロングワード**
データ型 **符号なしロングワード**
アクセス **読み取りのみ**
受け渡し方 **参照による**

非同期アクションのインデックス番号です。このインデックス番号は，実行する DEC XTPU ステートメントを登録した際に XTPU$SPECIFY_ASYNC_ACTION ルーチンに渡したのと同じものです。

説明

XTPU$TRIGGER_ASYNC_ACTION ルーチンは，XTPU$SPECIFY_ASYNC_ACTION ルーチンとともに使用することにより，XTPU$CONTROL を呼び出した後に DEC XTPU を中断させることを可能にします。指定のインデックス番号に対応するコマンドが作業項目のキューに登録され，他の作業項目がなくなったときに処理さ

れます。これによって，DEC XTPU はコマンドを実行する前に環境を安定にし一定に保つことができます。このルーチンは，XTPU$CONTROL ルーチンによって制御を DEC XTPU に渡した後に呼び出さなければなりません。

戻される条件値

XTPU$_SUCCESS	正常終了したことを示します。
XTPU$_UNKFACILITY	このルーチンに渡されたインデックス番号は XTPU$SPECIFY_ASYNC_ACTION に渡されたものではありません。

XTPU$XTPU

このルーチンは DEC XTPU を起動します。これは DCL の DEC XTPU コマンドと同じです。

形式

XTPU$XTPU *command*

戻り値

OpenVMS **用法** **条件値**
データ型 **符号なしロングワード**
アクセス **書き込みのみ**
受け渡し方 **値による**

ロングワードの条件値です。ユーティリティ・ルーチンはほとんど，条件値を R0 に戻します。このルーチンから戻される可能性のある条件値は， "戻される条件値 "にまとめられています。

引数

command

OpenVMS **用法** **文字列**
データ型 **文字列**
アクセス **読み取りのみ**
受け渡し方 **ディスクリプタによる**

コマンド文字列です。command 引数は，コマンド・ラインのディスクリプタのアドレスです。コマンド名には， XTPU を使います。

説明

このルーチンは，指定されたコマンド文字列を使用し，それをエディタに渡します。DEC XTPU は， DCL レベルからコマンドをタイプした場合と同様に，初期化のために，このコマンド文字列の情報を使用します。

単純な呼び出し可能インタフェースを使用する場合には，XTPU$M_CLOSE_SECTION フラグはセットされません。そのため，XTPU$XTPU を繰り返し呼び出す場合でも，そのたびにセクション・ファイルをオープンしたりクローズしたりする必要はありません。

ユーザ・アプリケーションがコマンド・ラインを解析して DEC XTPU の操作とは無関係な情報を得る場合，XTPU$XTPU を呼び出す前に，そうした情報をすべて取り出して使用してしまう必要があります。XTPU$XTPU は呼び出される前に格納されていた解析情報をすべて破壊してしまうからです。

戻される条件値

XTPU$INITIALIZE，XTPU$EXECUTE_INIFILE，XTPU$CONTROL，および XTPU$CLEANUP から戻されるすべての条件値です。

A

DEC XTPU における端末装置のサポートと制限事項

A.1 サポートされる端末装置

DEC XTPU がスクリーン編集機能をサポートする端末装置は，VT80，VT280 シリーズ，VT382 の DEC 社製の漢字端末装置です。また，弊社の製品あるいは他社製の端末装置で VT80，VT280 シリーズの機能をエミュレートするものは，そのエミュレータの制限内においてサポートされます。

弊社の製品ではないターミナルやターミナル・エミュレーション・ソフトウェアなどで DEC XTPU を使用する場合には，ターミナルの設定の他に標準版 OpenVMS および日本語 OpenVMS におけるターミナルの設定について注意する必要があります。DEC XTPU に影響を及ぼすターミナル設定の属性については『DEC Text Processing Utility Reference Manual』を参照してください。

A.2 制限事項

サポートされる端末装置のうちいくつかのものは，DEC XTPU の持つ機能の一部を実行できない場合があります。

A.2.1 DEC 漢字 1978 年版対応の端末装置

VT280 シリーズの端末装置のうち DEC 漢字 1978 年版対応のもの，および VT80 端末装置では，DEC 漢字 1983 年版で変更になった漢字を正しく表示しません。

A.2.2 132 カラム・モード

VT80 端末装置や漢字ターミナル・エミュレーション・ソフトウェアの中には，132 カラム・モードがサポートされていないものがあります。これらの端末装置を用いる場合でも，DEC XTPU は 80 カラムを越えるウィンドウ幅の指定を受け付けますが，スクリーン上のウィンドウ幅は正しく設定されません。また，その後の DEC XTPU の動作は保証されません。これらの端末装置を用いる場合には，ウィンドウ幅を 80 カラムに固定して使用してください。

A.2.3 罫線コード

日本語 OpenVMS の V4.6 以降では，罫線コードとして， DEC 漢字 1983 年版の罫線コードが使われています (8 区に含まれる 32 文字)。したがって， DEC XTPU の SYMBOL 組込みプロシージャなどで生成される罫線コードは，すべて DEC 漢字 1983 年版の罫線コードとなります。 DEC 漢字 1983 年版対応の VT280 シリーズおよび VT382 のターミナルでは, 日本語 OpenVMS の機能によって， 1978 年版， 1983 年版両方の罫線を見ることが可能ですが， DEC XTPU が正しく動作するためには, テキストに含まれる罫線コードは， 1983 年版に統一されていることが必要です。

B

DEC XTPU メッセージ

この付録には，DEC XTPU が作成するメッセージのうち DEC XTPU で追加されたメッセージが示されています。各メッセージには説明とエラーから回復するための処置も示されています。メッセージは重大度レベルに応じて分類されており，各グループ内ではアルファベット順に示されています。

これ以外のメッセージについては，『DEC Text Processing Utility Reference Manual』を参照してください。

B.1 サクセス・メッセージ

DICADD, 'tango' has been added to a dictionary as 'yomigana'
説明: 指定された読みがなで単語を個人辞書に登録できたことを示します。
ユーザの処置: なし。

DICDEL, 'tango' has been removed from 'yomigana' of a dictionary
説明: 指定された読みがなの単語を個人辞書から削除できたことを示します。
ユーザの処置: なし。

B.2 インフォメーショナル・メッセージ

ROUND, FORWARD was rounded to the top
説明: 次候補選択が一巡して最初の候補に戻ったことを示します。
ユーザの処置: なし。

B.3 警告メッセージ

INVCODESET, unrecognized codeset name
説明: コードセットのキーワードが誤っています。
ユーザの処置: /CODESET 修飾子のパラメータに正しいキーワードを指定します。

NOCLA, No conversion source has been specified yet
説明: 変換操作を開始しないうちに文節に関する操作をしようとしたことを示します。
ユーザの処置: 文節に関する操作をする前に CONVERT_KANA (START_CONVERSION,...) で，変換操作を開始します。

NODIC, No dictionary available in this editing session
説明: 個人辞書が現在利用不可能にもかかわらず，辞書を使用しようとしたことを示します。
ユーザの処置: /NOKANJI_DICTIONARY 修飾子を付けて DEC XTPU を起動している場合は，/NOKANJI_DICTIONARY 修飾子を付けないで起動しなおします。 SPAWN/ATTACH の間に辞書が利用不可能になった場合は，もう一度 SPAWN/ATTACH を行ってその原因を取り除くか，またはそのセッションを終了します。

NODICENT, No entry found in a dictionary
説明: 個人辞書への単語登録，削除などで，指定された読みがなに対応する単語が存在しません。
ユーザの処置: 正しい読みがなを指定し，同じ操作をもう一度実行します。

TOOMANYCLA, Too many clauses are found while converting.
説明: 変換中に文節の数が多くなりすぎたことを示します。
ユーザの処置: 一度に変換する文字列の長さを短くして，もう一度最初から変換操作をやり直します。

B.4 エラー・メッセージ

CLOSEDIC, Error closing the dictionary file
説明: 現在使用中の個人辞書をクローズできなかったことを示します。
ユーザの処置: なし。

CNVERR, Error occurred in the conversion routine
説明: かな漢字変換ルーチン中でエラーが発生したことを示します。
ユーザの処置: SPR を提出してください。

DICUPDERR, Error updating dictionary file
説明: 個人辞書の更新中にエラーが発生したことを示します。
ユーザの処置: なし。

OPENDIC, Error opening the dictionary file
説明: 指定された，または省略時の値として指定されている個人辞書をオープンできなかったことを示します。
ユーザの処置: 起動時であれば，辞書をオープンできない原因を取り除き，もう一度起動します。 SPAWN/ATTACH のプロセスから戻ったときであれば，再度 SPAWN/ATTACH を行ってオープンできない原因を取り除くか，このセッションを中断します。

STRTOOLNG, String is too long for a conversion source
説明: かな漢字変換の読みがなとして指定された文字列が長すぎます。変換可能な文字列の長さは，全角ひらがなで 253 文字です。
ユーザの処置: 読みがなに短い文字列を指定します。

索引

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

ア

イ

エ

キ

ク

ケ

コ

シ

ス

セ

タ

テ

ニ

ハ

フ

ヘ

マ

メ

モ

ユ

ヨ

ル

レ

ロ

日本語 HP OpenVMS
DEC XTPU **リファレンス・マニュアル**

2005 年 4 月　発行

日本ヒューレット・パッカード株式会社

〒 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-PY6UE-TE