日本語 Compaq TCP/IP Services for OpenVMS

日本語機能の手引き

AA-PUL3J-TE

2002 年 9 月

本書は，日本語 Compaq TCP/IP Services for OpenVMS の日本語処理機能について説明します。

改訂情報: 改訂版

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

ソフトウェア・バージョン: 日本語 Compaq TCP/IP Services for OpenVMS V5.3

コンパックコンピュータ株式会社

2002 年 9 月

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

目次

1 **日本語機能の概要**

1.1 **漢字フィルタ** 1–1

1.2 **標準で提供される漢字フィルタの構成と種類** 1–6

2 **漢字フィルタの使用方法**

2.1 FTP **での漢字フィルタ使用方法** 2–1

2.2 TELNET **での漢字フィルタ使用方法** 2–3

2.3 SMTP **での漢字フィルタ使用方法** 2–4

2.4 **リモート・プリントでの漢字フィルタ使用方法** 2–5

2.5 COPY/FTP **での漢字フィルタ使用方法** 2–6

2.6 SET HOST/TELNET **での漢字フィルタの使用方法** 2–7

3 **漢字フィルタ・プログラミング・ガイド**

3.1 **漢字フィルタ・プログラミングの概要** 3–1

3.1.1 フィルタの使用 3–1

3.1.2 ANET+ 用に作成されたフィルタの使用 3–2

3.2 **漢字フィルタの設計** 3–5

3.2.1 フィルタ設定ルーチン 3–5

3.2.2 フィルタ・ストリーム 3–6

3.2.3 バッファ資源管理 3–7

3.3 **フィルタ・ルーチン** 3–8

3.3.1 フィルタ設定ルーチン 3–8

TCPIP_FILTER (control-block) 3–9

3.3.2 フィルタ・ストリーム初期化ルーチン 3–11

BEGIN_FILTER (stream, application, resource) 3–12

3.3.3 入力フィルタ・ルーチン 3–14

NTOH_FILTER (stream, src, srclen,dst, dstlen) 3–15

3.3.4 出力フィルタ・ルーチン 3–17
HTON_FILTER (stream, src, srclen, dst, dstlen) 3–18
3.3.5 バッファ資源解放ルーチン 3–20
RELEASE_FILTER (stream, buffer) 3–21
3.3.6 フィルタ・ストリーム解放ルーチン 3–22
END_FILTER (stream) 3–23
3.4 **フィルタ・ルーチンの例** 3–24

4 **標準漢字フィルタの仕様**

4.1 7 **ビット** JIS **漢字フィルタ** 1 (JIS) 4–1
4.2 7 **ビット** JIS **漢字フィルタ** 2 (JISM) 4–3
4.3 **シフト** JIS **漢字フィルタ** (SJIS) 4–5
4.4 **日本語** EUC **漢字フィルタ** (UJIS) 4–7

索引

例

3–1 UPCASE.C の場合 3–25

図

1–1 TELNET の場合 1–2
1–2 FTP 送信の場合 1–3
1–3 FTP 受信の場合 1–3
1–4 SMTP 送信の場合 1–4
1–5 SMTP 受信の場合 1–4
1–6 リモート・プリント送信の場合 1–5
1–7 リモート・プリント受信の場合 1–5
3–1 control-block のデータ構造 3–10

表

1–1 漢字フィルタの設定 1–6
1–2 標準で提供される漢字フィルタ 1–6
2–1 標準で提供される漢字フィルタ 2–2
2–2 標準で提供される漢字フィルタ 2–3
2–3 標準で提供される漢字フィルタ 2–5
2–4 標準で提供される漢字フィルタ 2–6
2–5 標準で提供される漢字フィルタ 2–6
2–6 標準で提供される漢字フィルタ 2–7
3–1 フィルタ・ルーチン群 3–5
3–2 通信ユーティリティの種類と規定の値 3–13

1

日本語機能の概要

1.1 漢字フィルタ

日本語 OpenVMS では，日本語 (漢字) の符号化に関して DEC 漢字が使われていますが，各社のコンピュータ上での日本語 (漢字) の符号化にはシフト JIS，日本語 EUC， JIS 7 ビットなどさまざまな符号化方法が使用されているのが現状です。TCP/IP に基づいたマルチベンダ環境のコンピュータ・ネットワークを構築する場合，これらの符号化方法の違いを考慮する必要があります。

たとえば SMTP(Simple Mail Transfer Protocol) では， 8 ビット・データの送信が保証されていません。そのため多くの場合， ISO 2022 に基づいた JIS 7 ビット符号化が使用され， 8 ビット構成の DEC 漢字をそのままメール・メッセージとして送信しても正しく届けられません。同様に， SMTP でメールを受信する場合も，JIS 7 ビットのままで受信したメール・メッセージは，日本語 OpenVMS 上では，正しく表示されません。

また，シフト JIS を採用しているシステムへ OpenVMS から TELNET によりリモート・ログインを行った場合，漢字文字が正しく表示されません。

日本語 Compaq TCP/IP Services for OpenVMS では，このような場合に起こる不都合を解決する手段として， DEC 漢字以外の日本語 (漢字) コードから DEC 漢字への変換，およびその逆の変換を行うフィルタ機能を提供します (本書では， DEC 漢字は SS2 付きの半角カタカナを含む SuperDEC 漢字のことを指します)。以下の TCP/IP アプリケーションにおいて，これらの変換を行うためのフィルタが設定できます。

- FTP
- TELNET
- SMTP

- リモート・プリント

この機能により，前述した TCP/IP ネットワーク環境における日本語 (漢字) の符号化にともなう問題点に対処することができます。

これらの TCP/IP アプリケーションにおけるフィルタの利用例を以下に示します。

図 1–1 TELNET の場合

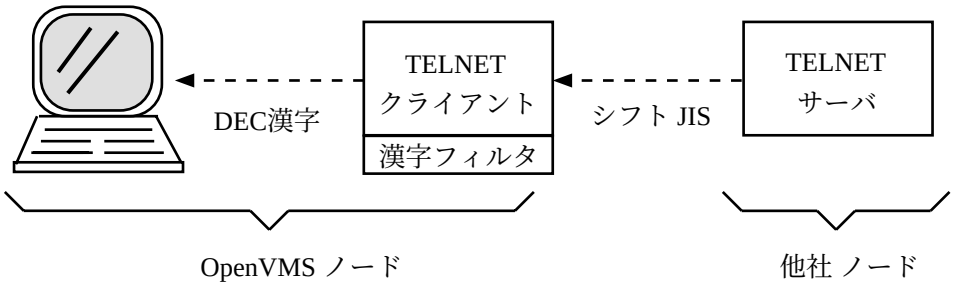


図 1–2 FTP 送信の場合

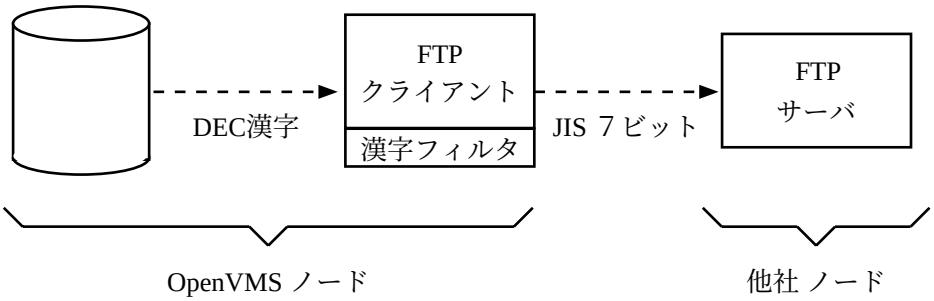


図 1–3 FTP 受信の場合

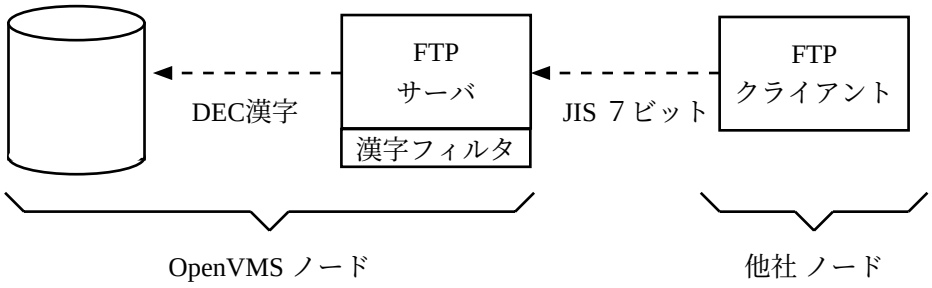


図 1–4　SMTP 送信の場合

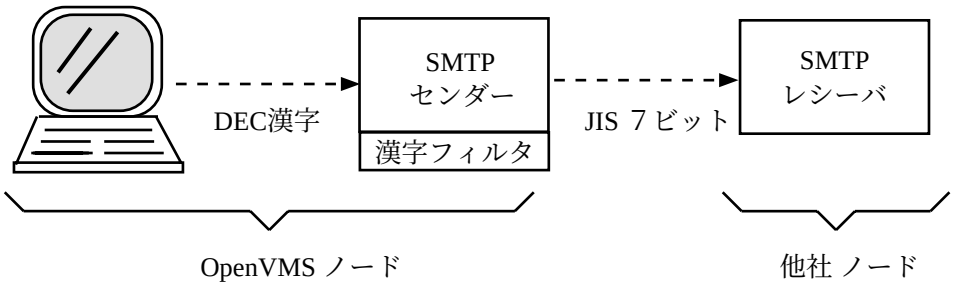


図 1–5　SMTP 受信の場合

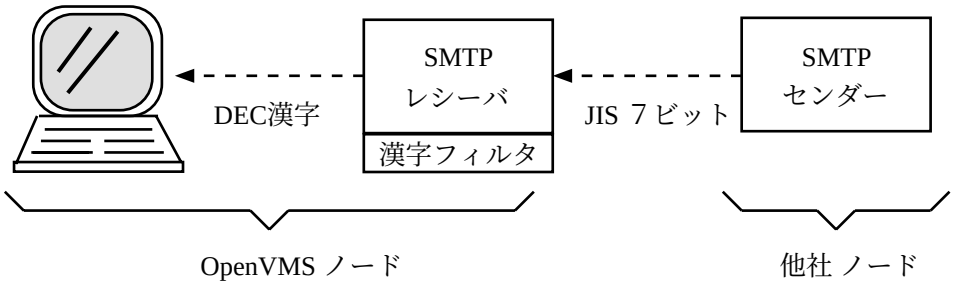


図 1–6 リモート・プリント送信の場合

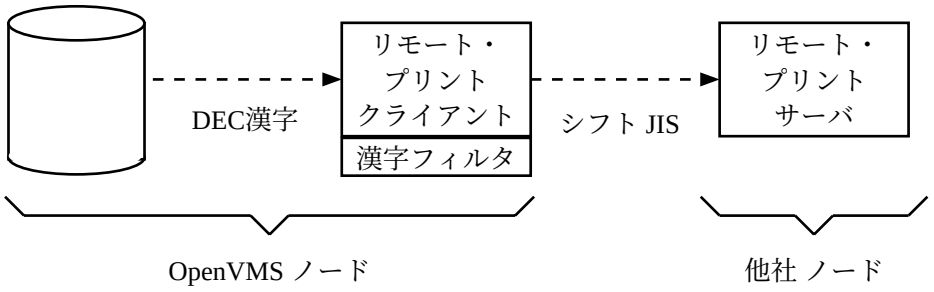
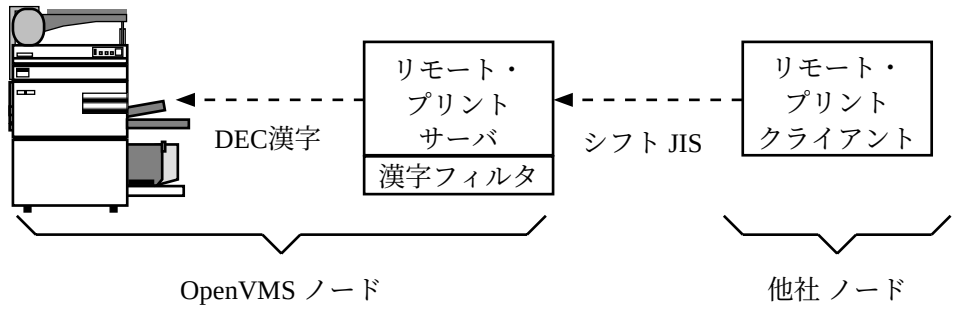


図 1–7 リモート・プリント受信の場合



1.2 標準で提供される漢字フィルタの構成と種類

日本語 Compaq TCP/IP Services for OpenVMS では， FTP， TELNET，SMTP，リモート・プリントそれぞれのユーティリティに対して，以下の構成で漢字フィルタの設定が可能です。

表 1–1 漢字フィルタの設定

サービス	クライアント	サーバ
FTP	○	○
TELNET	○	×
SMTP	○	○
リモート・プリント	○	○

以下の漢字フィルタが標準で提供されています。これらのフィルタの詳細な仕様については，第 4 章を参照してください。

表 1–2 標準で提供される漢字フィルタ

	フィルタ名
7 ビット JIS 漢字フィルタ (1)	JIS
7 ビット JIS 漢字フィルタ (2)	JISM
シフト JIS 漢字フィルタ	SJIS
UJIS 漢字フィルタ	UJIS

ユーザが必要に応じて，独自のフィルタ・ルーチンを作成して使用することも可能です。この方法については，第 3 章を参照してください。

また日本語 Compaq TCP/IP Services for OpenVMS では， ANET+ で用いられている漢字フィルタを使用することも可能です。この方法についても第 3 章を参照してください。

注意

ANET+ **は，日本語** Compaq TCP/IP Services for OpenVMS **と同様に** OpenVMS **上で** TCP/IP **の通信プロトコルに基づいたサービスを提供するソフトウェアで，従来より弊社が開発，および販売しています。**

2

漢字フィルタの使用方法

2.1 FTP での漢字フィルタ使用方法

FTP クライアントでの漢字フィルタの設定には，コマンド・インタフェースによる方法と，論理名で指定する方法の 2 通りがあります。コマンドと論理名の両方で別々のフィルタを指定した場合は，コマンドによる指定が優先されます。

UNIX 形式のコマンドを用いる場合

(UNIX 形式のコマンドの使用方法については，『*Compaq TCP/IP Services for OpenVMS User's Guide*』を参照してください。)

$ FTP/FILTER=フィルタ名

または

FTP> filter [フィルタ名]

OpenVMS 形式のコマンドを用いる場合

$ FTP/FILTER=フィルタ名

または

FTP> set filter フィルタ名

論理名を用いる場合

$ DEFINE [/SYSTEM] TCPIP$FTP_KANJI_FILTER フィルタ名

設定したフィルタの確認は，以下のコマンドで行います。

UNIX **形式のコマンドを用いる場合**

```
FTP> status
```

OpenVMS **形式のコマンドを用いる場合**

```
FTP> show status
```

あらかじめ提供されるフィルタは，以下の 4 つです。

表 2–1 標準で提供される漢字フィルタ

	フィルタ名
7 ビット JIS 漢字フィルタ (1)	JIS
7 ビット JIS 漢字フィルタ (2)	JISM
シフト JIS 漢字フィルタ	SJIS
UJIS 漢字フィルタ	UJIS

それぞれのフィルタについての詳細は第 4 章を参照してください。

FTP サーバでのフィルタの指定は， FTP のプロトコルで定義されている "QUOTE"と "SITE"コマンドを使用してリモートの FTP クライアントから行います。以下に，設定例を示しますが，ユーザの使用するコマンドの形式は，クライアント・システムにより，異なる場合もありますのでご注意ください。

なお， FTP サーバでは論理名でフィルタを指定することはできません。

```
ftp> quote "site filter フィルタ名 "
```

設定したフィルタを解除するには，次のようにフィルタ名を指定せずに SITE FILTER コマンドを実行します。

```
ftp> quote "site filter"
```

注意

FTP では，バイナリ・モードのときには FILTER の指定を無視します。

2.2 TELNET での漢字フィルタ使用方法

TELNET クライアントでの漢字フィルタの設定には，コマンド・インタフェースによる方法と，論理名で指定する方法の 2 通りがあります。コマンドと論理名の両方で別々のフィルタを指定した場合は，コマンドによる指定が優先されます。

コマンドでの指定は以下のようになります。

$ TELNET/FILTER=フィルタ名

または

TELNET> CONNECT/FILTER=フィルタ名

論理名での指定は以下のようになります。

$ DEFINE [/SYSTEM] TCPIP$TELNET_KANJI_FILTER フィルタ名

TELNET クライアントでは，サーバ (リモート・ノード) からクライアント (自ノード) 方向の転送のみにフィルタ機能が提供されます。また， TELNET サーバとしては，フィルタ設定はできません。あらかじめ提供されるフィルタは，以下の 4 つです。

表 2–2 標準で提供される漢字フィルタ

	フィルタ名
7 ビット JIS 漢字フィルタ (1)	JIS
7 ビット JIS 漢字フィルタ (2)	JISM
シフト JIS 漢字フィルタ	SJIS
UJIS 漢字フィルタ	UJIS

フィルタについての詳細は第 4 章を参照してください。

設定したフィルタの確認は，以下のコマンドで行います。

TELNET> status

または

TELNET> show status

注意

マルチセッションの場合， "$ TELNET/FILTER=**フィルタ名 ホスト名** " **で指定した漢字フィルタは最初のセッションのみに有効で，** "TELNET> CONNECT/FILTER=**フィルタ名 ホスト名** " **で指定した漢字フィルタは，そのセッションのみに有効です。**

マルチセッションの詳細に関しては，『*Compaq TCP/IP Services for OpenVMS User's Guide*』**を参照してください。**

2.3 SMTP での漢字フィルタ使用方法

SMTP では以下のように， TCPIP$SMTP_KANJI_FILTER という論理名に，設定したいフィルタ名を定義します。

形式:

```
$ DEFINE [/SYSTEM] TCPIP$SMTP_KANJI_FILTER フィルタ名
```

SMTP 受信フィルタの設定は，システム (/SYSTEM) 論理名定義で行います。個別ユーザの設定は行えません。 SMTP 送信フィルタの設定は，/SYSTEM で定義された設定が全てのユーザに適用されますが，各ユーザが個別に (プロセス) 論理名を定義すれば，そのフィルタ設定が有効となります。

SMTP で漢字フィルタを使用する場合，以下のことに注意してください。

- 漢字フィルタが漢字コードの変換対象として扱うのは，メールの本文だけです。メール・ヘッダ部は漢字コードの変換対象外です。

- 漢字フィルタは，メール本文全体を漢字コードの変換対象として扱います。たとえば， MIME(Multipurpose Internet Mail Extention) 標準 (RFC1521) のメールであっても， MIME は解釈されずに漢字コードの変換が行なわれます。

あらかじめ提供されるフィルタは，以下の 4 つです。

表 2–3 標準で提供される漢字フィルタ

	フィルタ名
7 ビット JIS 漢字フィルタ (1)	JIS
7 ビット JIS 漢字フィルタ (2)	JISM
シフト JIS 漢字フィルタ	SJIS
UJIS 漢字フィルタ	UJIS

それぞれのフィルタについての詳細は第 4 章を参照してください。

2.4 リモート・プリントでの漢字フィルタ使用方法

リモート・プリント・クライアントで漢字フィルタを使用するには TCPIP$PRINTCAP.DAT 中に以下のエントリを追加します。 PRINTCAP ファイルの設定方法については，『*Compaq TCP/IP Services for OpenVMS Management*』を参照してください。

:kf=フィルタ名

この時，以下のように， PRINT コマンドの中にフィルタの指定がある場合は，指定されたフィルタが優先的に選択されます。

```
$ PRINT/PARAMETER=(FILTER=フィルタ名)
```

あらかじめ提供されるフィルタは，以下の 4 つです。

表 2–4 標準で提供される漢字フィルタ

	フィルタ名
7 ビット JIS 漢字フィルタ (1)	JIS
7 ビット JIS 漢字フィルタ (2)	JISM
シフト JIS 漢字フィルタ	SJIS
UJIS 漢字フィルタ	UJIS

それぞれのフィルタについての詳細は，第 4 章を参照してください。

リモート・プリント・サーバで漢字フィルタを使用する場合，同様に TCPIP$PRINTCAP.DAT 中に以下のエントリを追加します。この場合は，リモート・ノードからのフィルタ設定の変更はできません。

:kf=フィルタ名

2.5 COPY/FTP での漢字フィルタ使用方法

COPY コマンドを用いて FTP を行う場合，以下のように TCPIP$FTP_KANJI_FILTER という論理名に，設定したいフィルタ名を定義します。なお， COPY コマンドでは，/FILTER 修飾子は指定できません。

形式:

```
$ DEFINE [/SYSTEM] TCPIP$FTP_KANJI_FILTER フィルタ名
```

表 2–5 標準で提供される漢字フィルタ

	フィルタ名
7 ビット JIS 漢字フィルタ (1)	JIS
7 ビット JIS 漢字フィルタ (2)	JISM
シフト JIS 漢字フィルタ	SJIS

(次ページに続く)

表 2–5 (続き) 標準で提供される漢字フィルタ

	フィルタ名
UJIS 漢字フィルタ	UJIS

2.6 SET HOST/TELNET での漢字フィルタの使用方法

SET HOST コマンドを用いて TELNET を行う場合，以下のように TCPIP$TELNET_KANJI_FILTER という論理名に，設定したいフィルタ名を定義します。なお， SET HOST コマンドでは/FILTER 修飾子は指定できません。

表 2–6 標準で提供される漢字フィルタ

	フィルタ名
7 ビット JIS 漢字フィルタ (1)	JIS
7 ビット JIS 漢字フィルタ (2)	JISM
シフト JIS 漢字フィルタ	SJIS
UJIS 漢字フィルタ	UJIS

3

漢字フィルタ・プログラミング・ガイド

3.1 漢字フィルタ・プログラミングの概要

日本語 Compaq TCP/IP Services for OpenVMS は， FTP， TELNET， SMTP，リモート・プリントにおいて，漢字フィルタの機能が提供されています。シフト JIS 漢字， 7 ビット JIS 漢字 (2 種類)，および UJIS 漢字と DEC 漢字コードとの変換フィルタが標準で提供されています。上記以外の漢字コードと DEC 漢字コードとの変換フィルタも，ユーザが必要に応じて作成して利用することができます。

また，このフィルタは漢字コードの変換にとどまらず，より一般的なフィルタと考えることができます。本章の最後に示す UPCASE フィルタは，すべての小文字を大文字に変換するフィルタを作成して，利用する場合のプログラム例です。

作成した漢字フィルタ・ルーチンは，以下に示すセットアップを行うことにより，上記アプリケーションで使用することが可能になります。

3.1.1 フィルタの使用

作成したフィルタを使用するには次の作業を行ってください。

1. フィルタ・ルーチンを TCPIP$フィルタ名_FSHR.EXE というファイル名で SYS$SHARE: ディレクトリに置くか，あるいは論理名 TCPIP$フィルタ名_FSHR で定義する。
2. フィルタ・イメージを INSTALL ユーティリティで/OPEN を指定してインストールする。
3. フィルタ・イメージのファイル保護を W:RE にする。

以下に詳細を示します。

作成したフィルタ・ルーチンを TCPIP$フィルタ名_FSHR.EXE というファイル名で SYS$SHARE: ディレクトリに置きます。ただし，論理名 TCPIP$フィルタ名_FSHR を定義すれば，必ずしも SYS$SHARE: ディレクトリに置く必要はありません。サーバのフィルタ指定に論理名を使用する場合は，システム (/SYSTEM) 論理名の定義が必要です。システム (/SYSTEM) 論理名を定義する場合は，/EXEC も同時に指定してください。

漢字フィルタ共用イメージは， INSTALL ユーティリティで/OPEN を指定してインストールします。 SYS$SHARE: ディレクトリに標準で提供されているフィルタ共用イメージ TCPIP$フィルタ名_FSHR.EXE は，日本語 Compaq TCP/IP Services for OpenVMS 起動コマンド・プロシージャ (SYS$STARTUP:TCPIP$FILTER_STARTUP.COM) が自動的にインストールします。 SYS$SHARE: ディレクトリ以外にフィルタ共用イメージを置く場合には，論理名の定義を行い INSTALL ユーティリティで/OPEN を指定してインストールしてください。漢字フィルタ共用イメージのファイル保護を W:RE にすれば，あるユーザが作成したフィルタをすべてのユーザが使用できるようになります。

3.1.2 ANET+ 用に作成されたフィルタの使用

日本語 Compaq TCP/IP Services for OpenVMS 漢字フィルタ機能は， ANET+ に提供されていた漢字フィルタ機能に基づいて実装されています。以下のようなコマンドにより， ANET+ 用に作成されたフィルタを，日本語 Compaq TCP/IP Services for OpenVMS においても，使用することができます。以下の例において，フィルタ名は ANET+ 用に使用する場合のフィルタの名称を指定します。

ANETP 用に作成されたフィルタを使用するには次の作業を行ってください。

1. フィルタ・ルーチンを ANETP_フィルタ名_FSHR.EXE というファイル名で SYS$SHARE: ディレクトリに置くか，あるいは論理名 ANETP_フィルタ名_FSHR で定義する。
2. フィルタ・イメージを INSTALL ユーティリティで/OPEN を指定してインストールする。
3. フィルタ・イメージのファイル保護を W:RE にする

前述の TCPIP 漢字フィルタのセットアップの説明中において,
TCPIP$フィルタ名_FSHR.EXE は， ANETP_フィルタ名_FSHR.EXE と,
TCPIP$フィルタ名_FSHR は， ANETP_フィルタ名_FSHR と置き換わります。

TELNET クライアントの場合

$ TELNET/ANETP/FILTER=フィルタ名

FTP クライアントの場合

UNIX 形式のコマンドを用いる場合

$ FTP/ANETP/FILTER=フィルタ名

または

FTP> filter [フィルタ名_ANETP]

OpenVMS 形式のコマンドを用いる場合

$ FTP/ANETP/FILTER=フィルタ名

または

FTP> set filter フィルタ名_ANETP

FTP サーバの場合

ftp> quote "site filter フィルタ名_ANETP"

SMTP の場合

$ DEFINE [/SYSTEM] TCPIP$SMTP_KANJI_FILTER フィルタ名_ANETP

リモート・プリントの場合

TCPIP$PRINTCAP.DAT 中のエントリでのフィルタ指定。

:kf=フィルタ名_ANETP

PRINT **コマンドでのフィルタ指定**

```
$ PRINT /PARAMETER=(FILTER=フィルタ名_ANETP)
```

3.2 漢字フィルタの設計

漢字フィルタは 2 つの要素から構成されています。コード変換を行うためのサブルーチン群と，それを呼び出す FTP， TELNET などの通信ユーティリティです。コード変換を行うサブルーチン群は共用イメージです。 SYS$SHARE:TCPIP$SJIS_FSHR.EXE は， DEC 漢字コードと SJIS 漢字コードの変換フィルタです。フィルタ共用イメージは，次に説明するフィルタ設定ルーチンと，以下の 5 つのフィルタ・ルーチン群を用意する必要があります。

表 3–1 フィルタ・ルーチン群

ルーチン名	内容
BEGIN_FILTER	フィルタ・ストリーム初期化ルーチン
NTOH_FILTER	入力フィルタ・ルーチン
HTON_FILTER	出力フィルタ・ルーチン
RELEASE_FILTER	バッファ資源解放ルーチン
END_FILTER	フィルタ・ストリーム解放ルーチン

3.2.1 フィルタ設定ルーチン

FTP， TELNET などの通信ユーティリティは，各ユーティリティのコマンド行，または修飾子/FILTER によりフィルタ名が指定されると，フィルタ名からフィルタ共用イメージのファイル名を作成します。たとえば FTP/FILTER=SJIS コマンドで FTP を起動すると， FTP は TCPIP$フィルタ名_FSHR.EXE という規則にしたがい， TCPIP$SJIS_FSHR.EXE を得ます。

次に通信ユーティリティは LIB$FIND_IMAGE_SYMBOL を呼び出し，共用イメージを起動し， tcpip_filter というユニバーサル・シンボルの値を得ます。すべてのフィルタ共用イメージは tcpip_filter というユニバーサル・シンボルを持たなければなりません。そしてこのユニバーサル・シンボルは，フィルタ設定ルーチンのエントリ・ポイントのアドレスとして使われます。

FTP に対して/FILTER=SJIS が指定されていると， FTP は tcpip_filter というユニバーサル・シンボルの値を解決し，サブルーチンとして呼び出します。このサブルーチンをフィルタ設定ルーチンと呼びます。フィルタ設定ルーチンを呼び出すことにより，第 3.3 節で説明する 5 つのフィルタ・ルーチン群のエントリ・ポイントが明らかになります。

3.2.2 フィルタ・ストリーム

フィルタ・ルーチン群はフィルタ・ストリームを管理しなければなりません。FTP， TELNET などの通信ユーティリティは，フィルタ・ストリーム初期化ルーチン BEGIN_FILTER を呼び出して必要な数のフィルタ・ストリームを得ます。BEGIN_FILTER ルーチンはフィルタ・ストリームの管理に必要なデータ構造の割り当て，初期化などを行った後，その識別子を呼び出し者に与えます。以後，入力/出力フィルタ・ルーチン，バッファ資源解放ルーチン，フィルタ・ストリーム解放ルーチンの呼び出しは，この識別子でフィルタ・ストリームを指定します。

フィルタ・ストリームは，変換されるデータの流れに対応するものです。フィルタ・ルーチンに変換のために渡されるデータ，それ自体で変換可能な完結したレコードではなく，変換されるべきデータの流れの一部が連続して渡されると考えなければなりません。

7 ビット JIS 漢字コードを DEC 漢字コードに変換する場合を考えてみます。 7 ビット JIS 漢字コード自体は ASCII コードと見分けがつかず， (いわゆる漢字 IN，漢字 OUT と呼ばれる) エスケープ・シーケンスにより文字集合を指示しています。もし，第 N 回目の JIS 漢字から DEC 漢字への変換ルーチンの呼び出しで漢字文字集合が指示されたら，第 N+1 回目の変換は，漢字文字集合が指示された状態から開始されなければなりません。このように，現在 JIS 漢字文字集合が指示されているのか，それとも ASCII 文字集合が指示されているのか，という状態をフィルタ・ストリームに記憶する必要があります。また， 7 ビット JIS 漢字コードに限らず，変換されるべきデータが 2 バイト漢字コードの 1 バイト目で終了しているという場合もありえます。この場合は 2 バイト漢字コードの 1 バイト目をフィルタ・ストリームに記憶し，次の呼び出しで与えられたデータの先頭の 1 バイトと合わせて 2 バイト漢字コードとして変換しなければなりません。

フィルタ・ストリームはデータの流れに対応しますから，1つのストリームで入力フィルタ・ルーチンと出力フィルタ・ルーチンが混合して呼び出されることはありません。入力フィルタ（ネットワークから読み込んだデータの変換）と出力フィルタ（ネットワークへ書き込むデータの変換）の両方を同時に必要とするときは2つのストリームが取得されます。

3.2.3 バッファ資源管理

FTP，TELNETなどの通信ユーティリティは，BEGIN_FILTERルーチンの呼び出し時に，あらかじめ確保すべきバッファ資源の数を指示します。バッファ資源とは，入力/出力フィルタ・ルーチンが変換結果を格納して呼び出し者に一時的に与えるバッファです。入力/出力フィルタの呼び出しのたびにこのバッファ資源が消費されます。通信ユーティリティは変換結果が必要でなくなると，バッファ資源解放ルーチンを呼び出し，フィルタ・ストリームにバッファ資源を返します。通信ユーティリティはBEGIN_FILTERで指定したバッファ資源の数の回数分はRELEASE_FILTERを呼び出さずに，連続して資源を消費する場合があります。

ただし，BEGIN_FILTERルーチンに対してあらかじめ確保するバッファ資源の数1を指示した場合は，通信ユーティリティはバッファ資源解放ルーチンを呼び出しません。バッファ資源の数が1の場合は，入力/出力フィルタ・ルーチンの呼び出しで与えられたバッファ資源は，次の入力/出力フィルタ・ルーチンの呼び出しで自動的に解放されるものとして扱われます。

3.3 フィルタ・ルーチン

各フィルタ・ルーチンについて説明します。

- フィルタ設定ルーチン

  TCPIP_FILTER (control-block)

- フィルタ・ストリーム初期化ルーチン

  BEGIN_FILTER (stream, application, resource)

- 入力フィルタ・ルーチン

  NTOH_FILTER (stream, src, srclen,dst, dstlen)

- 出力フィルタ・ルーチン

  HTON_FILTER (stream, src, srclen, dst, dstlen)

- バッファ資源解放ルーチン

  RELEASE_FILTER (stream, buffer)

- フィルタ・ストリーム解放ルーチン

  END_FILTER (stream)

3.3.1 フィルタ設定ルーチン

TCPIP_FILTER (control-block)

フィルタ・ルーチン群のエントリ・ポイントを明らかにします。通信ユーティリティはフィルタ設定ルーチンを呼び出して 5 つのフィルタ・ルーチンのエントリ・ポイントのアドレスを得ます。

戻り値

VMS **用法**: cond_value
データ型: longword (unsigned)
アクセス: write only
受け渡し方: by value

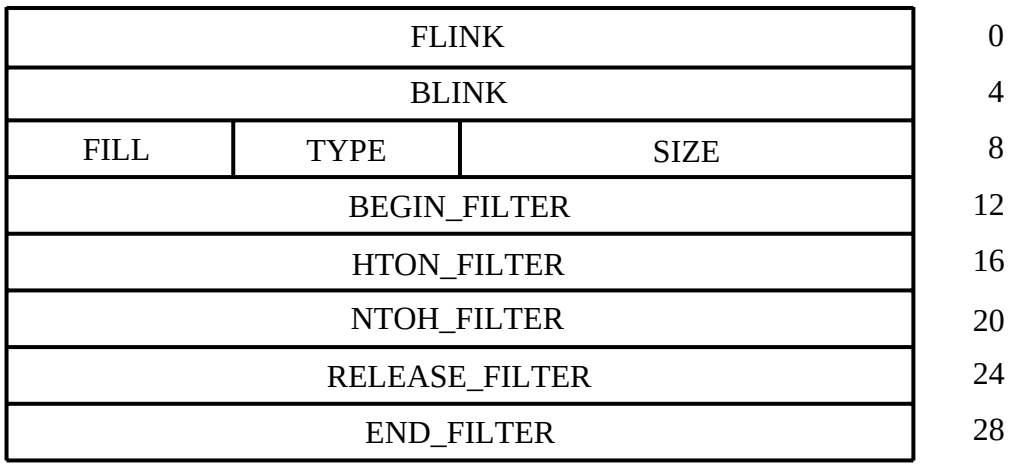
引数

control-block

データ型: record
アクセス: write only
受け渡し方: by reference

フィルタ・ルーチン群のアドレスを通信ユーティリティに対して明らかにするために使われる構造体です。次ページに control-block のデータ構造を示します。

図 3–1 control-block のデータ構造



説明

FTP, TELNET などの通信ユーティリティは，フィルタを指定されると LIB$FIND_IMAGE_SYMBOL を使ってフィルタ共用イメージ TCPIP$フィルタ名_FSHR.EXE を起動し， tcpip_filter というユニバーサル・シンボルの値を得ます。フィルタ共用イメージは， tcpip_filter というユニバーサル・シンボルを持つフィルタ設定ルーチンを必ず用意しなければなりません。共用イメージの起動後，フィルタ設定ルーチンが呼び出されます。フィルタ設定ルーチンは，Control-block パラメータで渡された構造体に 5 つのフィルタ・ルーチンのエントリ・ポイントを書き込みます。以後フィルタをサポートする通信ユーティリティは control-block 内に書き込まれたサブルーチンを必要に応じて呼び出します。

FLINK, BLINK, SIZE, TYPE, FILL は通信ユーティリティが使用します。フィルタ・ルーチンは FLINK, BLINK, SIZE, TYPE, FILL の内容を変更できません。 BEGIN_FILTER フィールドにはフィルタ・ストリーム初期化ルーチンの, HTON_FILTER フィールドには出力フィルタ・ルーチンの，また，NTOH_FILTER フィールドには入力フィルタ・ルーチンのエントリ・ポイントのアドレスを書き込みます。 RELEASE_FILTER フィールドにはバッファ資源解放ルーチンの, END_FILTER フィールドにはフィルタ・ストリーム解放ルーチンのエントリ・ポイントのアドレスを書き込みます。

戻り値がサクセスの場合，フィルタが設定され，以後必要に応じてフィルタ・ストリーム初期化ルーチンが呼び出されます。戻り値がエラーの場合，フィルタは設定されません。

3.3.2 フィルタ・ストリーム初期化ルーチン

BEGIN_FILTER (stream, application, resource)

フィルタ・ストリームを生成，初期化し，識別子を呼び出し者に与えます。

戻り値

VMS **用法**: cond_value
データ型: longword (unsigned)
アクセス: write only
受け渡し方: by value

引数

stream

データ型: longword (unsigned)
アクセス: write only
受け渡し方: by reference

フィルタ・ストリーム識別子。

application

データ型: longword (unsigned)
アクセス: read only
受け渡し方: by reference

フィルタ・ルーチンを呼び出す通信ユーティリティの種類を示します。現在，以下の値を規定しています。

表 3–2 通信ユーティリティの種類と規定の値

通信ユーティリティ	規定値
FTP	1
TELNET	2
SMTP	3
LPR	4

FTP, TELNET, SMTP, LPR の各通信ユーティリティはフィルタ・ストリーム初期化ルーチンを呼び出すとき，それぞれに割り当てられた番号を application パラメータとしてセットします。この番号はフィルタ・ルーチンのストリームに記憶するべきです。フィルタ・ルーチンはこの番号により上位通信ユーティリティが何であるか知ることができます。これは 1 つのフィルタ・ルーチンの中で，上位通信ユーティリティに依存する異なるフィルタ処理をしなければならない場合を予想しての準備です。

resource

データ型: longword integer (signed)
アクセス: read only
受け渡し方: by reference

フィルタ・ストリーム初期化時に確保するバッファ資源の数を指示します。

説明

フィルタ・ストリーム識別子は，以後，入力/出力フィルタ・ルーチン，バッファ資源解放ルーチン，フィルタ・ストリーム解放ルーチンを呼び出すとき，ストリームを識別するためにパラメータとして渡されます。フィルタ・ストリーム初期化ルーチンは，それが管理/識別できる形で，生成したストリームの識別子を呼び出し者に与えます。ストリーム識別子は 0 でない値でなければなりません。異なる 2 つのフィルタ・ストリームは異なる識別子を持たなければなりません。

バッファ資源の数は，フィルタ資源解放ルーチンを呼び出すことなく連続して入力/出力フィルタ・ルーチンを呼び出すことができる最大の回数を示します。入力/出力フィルタを呼び出すたびにバッファ資源は消費されます。入力/出力フィルタが変換結果を格納し， dst パラメータでサブルーチンの呼び出し者に与えるバッファは，その変換結果が不要となった時点でフィルタ・ストリームに返されます。通信ユーティリティはあらかじめ確保した資源数の回数以内で，フィルタ資源解放ルーチンを呼び出すことなく，入出力フィルタ・ルーチンを呼び出す場合があります。ただし，バッファ資源の数が 1 の場合，通信ユーティリティはフィルタ資源解放ルーチンを呼び出しません。資源の数が 1 の場合，つぎに入力/出力フィルタ・ルーチンが呼び出された時点で，前回の入力/出力フィルタ・ルーチンが与えたバッファ資源は解放されているものとみなします。

戻り値がサクセスの場合，以後必要に応じて入力/出力フィルタ・ルーチン，フィルタ資源解放ルーチンが呼び出されます。フィルタ・ストリームが不要になった時点でフィルタ・ストリーム解放ルーチンが呼び出されます。戻り値がエラーの場合，入力/出力フィルタ・ルーチンは呼び出されません。フィルタなしの状態になります。

3.3.3 入力フィルタ・ルーチン

NTOH_FILTER (stream, src, srclen,dst, dstlen)

ネットワーク (リモート・ホスト) から受信したデータ・ストリームを変換するフィルタ・ルーチンです。漢字変換フィルタは他機種漢字コードから DEC 漢字コードへの変換を行います。

戻り値

VMS **用法**: cond_value
データ型: longword (unsigned)
アクセス: write only
受け渡し方: by value

引数

stream

データ型: longword (unsigned)
アクセス: read only
受け渡し方: by reference

フィルタ・ストリーム初期化ルーチンで得たフィルタ・ストリーム識別子。変換ストリームの識別のために使用されます。

src

データ型: character string
アクセス: read only
受け渡し方: by reference

入力フィルタ・ルーチンにより変換されるソース文字列。文字列の先頭アドレスがパラメータとして渡されます。文字列のバイト数は srclen パラメータが示します。

srclen

データ型: word (unsigned)
アクセス: read only
受け渡し方: by reference

src パラメータで渡されるソース文字列のバイト数を示します。

dst

データ型: address of character string
アクセス: write only
受け渡し方: by reference

入力フィルタ・ルーチンが変換結果文字列の先頭アドレスを書き込みます。フィルタ初期化ルーチンでバッファ資源数 2 以上を指示した場合，このパラメータで返される値がバッファ資源解放ルーチンが呼び出されるときのパラメータになります。

dstlen

データ型: word (unsigned)
アクセス: write only
受け渡し方: by reference

入力フィルタ・ルーチンが変換結果文字列のバイト数を書き込みます。

説明

入力フィルタは，ネットワークあるいはリモート・ホストから受信したデータの変換を行います。変換結果は dst， dstlen パラメータで呼び出し者に返されます。入力フィルタ・ルーチンは，呼び出されるたびにストリームのバッファ資源を消費します。この資源は，不要になった時点でフィルタ資源解放ルーチンを呼び出して，フィルタ・ストリームに返されます。フィルタ資源解放ルーチンが呼び出されてバッファが解放されたら，その資源は再使用することができます。ただし，ストリーム初期化時にあらかじめ確保されたバッファ資源数が 1 の場合は，バッファ資源解放ルーチンが呼び出されなくても， 1 つのバッファ資源を繰り返し使用することになります。

漢字コードの変換においては，入力文字列と出力文字列の長さが異なる場合が予想されます。入力フィルタ・ルーチンは変換結果の格納のために，あらかじめ確保されていたバッファ資源，あるいはバッファ資源解放ルーチンでストリームに返されたバッファ資源を使用します。しかし，変換中に変換結果文字列を格納するための領域が不足することが判明したら，その時点でさらに大きな領域を取得する必要があります。

戻り値がサクセスの場合，変換結果の文字列が変換前の文字列の代わりに使われます。 FTP の場合変換結果文字列がファイルに出力され， TELNET の場合変換結果文字列は端末に表示されます。戻り値がエラーの場合，フィルタ処理は解除されフィルタなしの状態に戻ります。

3.3.4 出力フィルタ・ルーチン

HTON_FILTER (stream, src, srclen, dst, dstlen)

ネットワーク (リモート・ホスト) へ送信するデータを変換するフィルタ・ルーチンです。漢字変換フィルタは DEC 漢字コードから他機種漢字コードへの変換を行います。

戻り値

VMS **用法**: cond_value
データ型: longword (unsigned)
アクセス: write only
受け渡し方: by value

引数

stream

データ型: longword (unsigned)
アクセス: read only
受け渡し方: by reference

フィルタ・ストリーム初期化ルーチンで得たフィルタ・ストリーム識別子。変換ストリームの識別のために使用されます。

src

データ型: character string
アクセス: read only
受け渡し方: by reference

出力フィルタ・ルーチンにより変換されるソース文字列。文字列の先頭アドレスがパラメータとして渡されます。文字列のバイト数は srclen パラメータが示します。

srclen

データ型: word (unsigned)
アクセス: read only
受け渡し方: by reference

src パラメータで渡されるソース文字列のバイト数を示します。

dst

データ型: address of character string
アクセス: write only
受け渡し方: by reference

出力フィルタ・ルーチンが変換結果文字列の先頭アドレスを書き込みます。フィルタ初期化ルーチンでバッファ資源数 2 以上を指示した場合，このパラメータで返される値がバッファ資源解放ルーチンが呼び出される時のパラメータになります。

dstlen

データ型: word (unsigned)
アクセス: write only
受け渡し方: by reference

出力フィルタ・ルーチンが変換結果文字列のバイト数を書き込みます。

説明

出力フィルタは，ネットワークあるいはリモート・ホストへ送信するデータの変換を行います。変換結果は dst， dstlen パラメータで呼び出し者に返されます。出力フィルタ・ルーチンは呼び出されるたびにストリームのバッファ資源を消費します。この資源は，不要になった時点でフィルタ資源解放ルーチンを呼び出して，フィルタ・ストリームに返されます。フィルタ資源解放ルーチンが呼び出されてバッファが解放されたら，その資源は再使用することができます。ただし，ストリーム初期化時にあらかじめ確保されたバッファ資源数が 1 の場合は，バッファ資源解放ルーチンが呼び出されなくても， 1 つのバッファ資源を繰返し使用することになります。

漢字コードの変換においては，入力文字列と出力文字列の長さが異なる場合が予想されます。出力フィルタ・ルーチンは変換結果の格納のために，あらかじめ確保されていたバッファ資源，あるいはバッファ資源解放ルーチンでストリームに返されたバッファ資源を使用します。しかし，変換中に変換結果文字列を格納するための領域が不足することが判明したら，その時点でさらに大きな領域を取得する必要があります。

戻り値がサクセスの場合，変換結果の文字列が変換前の文字列の代わりに使われます。 FTP の場合変換結果文字列がリモート・ホストに送信されます。戻り値がエラーの場合，フィルタ処理は解除され，フィルタなしの状態に戻ります。

3.3.5 バッファ資源解放ルーチン

RELEASE_FILTER (stream, buffer)

入力/出力フィルタ・ルーチンの呼び出しにより消費されたバッファ資源をフィルタ・ストリームに解放するために呼び出されます。

戻り値

VMS **用法**: cond_value
データ型: longword (unsigned)
アクセス: write only
受け渡し方: by value

引数

stream

データ型: longword (unsigned)
アクセス: read only
受け渡し方: by reference

フィルタ・ストリーム識別子。フィルタ・ストリーム初期化ルーチンにより与えられた識別子です。

buffer

データ型: character string
アクセス: read only
受け渡し方: by reference

入力/出力フィルタ・ルーチンが dst パラメータで与えた変換結果文字列の先頭アドレスです。

説明

入力/出力フィルタ・ルーチンの呼び出しにより消費されたバッファ資源をフィルタ・ストリームに解放し，再使用可能にします。

ストリーム初期化ルーチンで指示した，あらかじめ確保する資源の数が 1 だった場合，フィルタ資源解放ルーチンを呼び出すことなく入力/出力フィルタ・ルーチンが繰り返し呼び出されます。

あらかじめ確保するバッファ資源が 2 以上だった場合，入力/出力フィルタ・ルーチンの呼び出しで dst パラメータとして与えたバッファは，バッファ資源解放ルーチンが呼び出されてストリームに返されるまでは再び使用することはできません。

戻り値がエラーの場合，フィルタ処理は解除されフィルタなしの状態に戻ります。

3.3.6 フィルタ・ストリーム解放ルーチン

END_FILTER (stream)

不要となった変換ストリームを解放するために呼び出されます。

戻り値

VMS **用法**: cond_value
データ型: longword (unsigned)
アクセス: write only
受け渡し方: by value

引数

stream

データ型: longword (unsigned)
アクセス: read only
受け渡し方: by reference

フィルタ・ストリーム識別子。フィルタ・ストリーム初期化ルーチンにより与えられた識別子です。

説明

フィルタ・ストリーム初期化ルーチンの呼び出しで生成したフィルタ・ストリームを解放します。

戻り値がエラーの場合，フィルタ処理は解除されフィルタなしの状態に戻ります。

3.4 フィルタ・ルーチンの例

この例は， STR$UPCASE ルーチンを使ってすべての小文字を大文字に変換するフィルタです。使用言語は VAXC V3.1 です。

次のようにコンパイル/リンクします。^Z は Ctrl/Z を表します。

```
$ CC UPCASE.C
$ LINK/SHARE=TCPIP$UPCASE_FSHR.EXE UPCASE, TT/OPT
  UNIVERSAL = tcpip_filter
^Z
$
```

このフィルタ共用イメージを使用するには， TCPIP$UPCASE_FSHR.EXE を SYS$SHARE: ディレクトリにコピーするか，あるいは TCPIP$UPCASE_FSHR.EXE のあるディレクトリで次のように論理名を定義してください。

```
$ DEFINE TCPIP$UPCASE_FSHR 'F$ENVIRON("DEFAULT")'TCPIP$UPCASE_FSHR.EXE
```

UPCASE.C のソース・プログラムの例を次ページ以降に示します。

例 3–1　UPCASE.C **の場合**

```
#module UPCASE "V2.2-021"
#include ssdef
#pragma builtins

#define KF_LENGTH sizeof(struct KnjFilter)
struct KnjFilter {
    struct KnjFilter *kf_next;          /* Forward link                    */
    struct KnjFilter *kf_prev;          /* Backward link                   */
    unsigned short int kf_size;         /* Length                          */
    unsigned char kf_type;              /* Type                            */
    unsigned char kf_fill;              /* Unused                          */
    int (*kf_begin_filter)();           /* Allocate filter stream          */
    int (*kf_hton_filter)();            /* Host to net filter              */
    int (*kf_ntoh_filter)();            /* Net to host filter              */
    int (*kf_release_filter)();         /* Resource deallocation           */
    int (*kf_end_filter)();             /* Deallocate filter stream        */
    } ;
#define KB_LENGTH (sizeof(struct KnjBuf)-sizeof(char))
struct KnjBuf {                         /*                                 */
    struct KnjBuf *kb_next;             /* Forward link                    */
    struct KnjBuf *kb_prev;             /* Backward link                   */
    long int kb_size;                   /* Length                          */
    char kb_data [1];                   /* Data                            */
    } ;
#define KC_LENGTH sizeof(struct KnjContext)
struct KnjContext {                     /*                                 */
    struct KnjContext *kc_next;         /* Forward link                    */
    struct KnjContext *kc_prev;         /* Backward link                   */
    unsigned short int kc_size;         /* Length                          */
    unsigned char kc_type;              /* Type                            */
    char kc_fill;                       /* Unused                          */
    long int kc_application;            /* Name of caller                  */
    struct KnjBuf *kc_kb [2];           /* Queue header of free KnjBuf     */
    struct KnjBuf *kc_kbinuse [2];      /* Queue header of in-use KnjBuf   */
    long int kc_flags;                  /* Flags, see KC_                  */
    } ;
#define KC_M_RELREQ 1
#define KC_M_PRESERVE 2
#define HLA_FTP 1                       /* File Transfer Protocol          */
```

(**次ページに続く**)

例 3–1 (続き) UPCASE.C の場合

```
#define HLA_TELNET 2                    /* TELNET, virturl terminal        */
#define HLA_SMTP 3                      /* Simple Mail Transfer Protocol   */
#define HLA_LPR 4                       /* LPR                             */
#define HLA_MAX 4

typedef unsigned char u_char;
typedef unsigned short u_short;
typedef unsigned long u_long;
/*
 * The macros used to check condition value
 */
#define issuccess(r) (1&(r))
#define iserror(r) (!issuccess(r))
#define check(f) {long $$r=(f);if(iserror($$r))lib$stop($$r);}


#define onerror(f,a) {long $$r=(f);if(iserror($$r)) a;}
/*
 * When filter routines realize that KanjiBuf is too short to
 * store resultant string, ReplaceKB macro is executed.
 */
#define ReplaceKB(kc,kb,d,dl){\
        long l = (kb)->kb_size - (dl);\
        if(((kb) = NewBuf((kc),(kb),l)) == 0) return(SS$_INSFMEM);\
        (d) = (kb)->kb_data + l; (dl) = (kb)->kb_size - l;}

#define Display(m){\
        static char d$[]=m; long d[]={sizeof(d$)-1,d$}; lib$put_output(d);}

static begin_filter();
static end_filter();
static hton_filter();
static ntoh_filter();
static release_filter();
static struct KnjBuf *GetBuf();
static struct KnjBuf *NewBuf();
static handler();

static long KnjContext[] = {KnjContext,KnjContext};
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
/*
 * tcpip_filter
 *
 *      This routine fills Kanji Filter block with entry
 *      points of all filter routines. To make the address
 *      of this routine known to external, TCPIP$UPCASE_FSHR
 *      is an universal symbol.
 */
tcpip_filter(kf)
register struct KnjFilter *kf;
{
        /*
         * Place known entry points of filter routine in KnjFilter block.
         */
        kf->kf_begin_filter = begin_filter;
        kf->kf_hton_filter = hton_filter;
        kf->kf_ntoh_filter = ntoh_filter;
        kf->kf_end_filter = end_filter;
        kf->kf_release_filter = release_filter;

        return(SS$_NORMAL);
}
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
/*
 * BEGIN_FILTER
 *
 *      Begin_filter routine is called by TCPIP applications
 *      to get filter stream. Filter stream preserves several
 *      stream specific information.
 *      When a value of resource arguent is 1, TCPIP application
 *      do not call release_filter routine after a call of
 *      ntoh_filter or hton_filter. When a value of the argument
 *      is greater than 1, release_filter routine must be called
 *      after a call of ntoh_filter or hton_filter.
 */
static begin_filter(stream,application,resource)
u_long *stream;
u_long *application;
long *resource;
{
        register struct KnjContext *kc;
        register struct KnjBuf *kb;
        static char *t;
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
/*
 * Establish a condition handler.
 * R13 = FP.
 */
((long *)_READ_GPR(13))[0] = handler;
/*
 * Allocate Kanji Filter Context block for the stream.
 */
onerror(lib$get_vm(&sizeof(struct KnjContext),&t), return($$r));
kc = (struct KnjContext *)t;
kc->kc_size = sizeof(struct KnjContext);
kc->kc_type = 0;
kc->kc_fill = 0;
kc->kc_application = *application;
kc->kc_kb[1] = kc->kc_kb[0] = kc->kc_kb;
kc->kc_kbinuse[1] = kc->kc_kbinuse[0] = kc->kc_kbinuse;
kc->kc_flags = 0;
/*
 * Queue it
 */
_INSQUE((void *)kc,(void *)KnjContext);
/*
 * Allocate 1 Kanji Buff. Length of buffer
 * depends on high level application.
 */
kb = GetBuf(kc,512);
if(kb == 0) {
        end_filter(kc);
        *stream = 0;
        return(SS$_INSFMEM);
}
/*
 * Queue the Kanji Buff into Kanji Context.
 */
_INSQUE((void *)kb,(void *)kc->kc_kb);
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
        if(*resource > 1)
                kc->kc_flags |= KC_M_RELREQ;
        *stream = (long)kc;
        return(SS$_NORMAL);
}

/*
 * END_FILTER
 *
 *      This routine is called when TCPIP application no longer
 *      needs filter stream, that was previously allocated by
 *      begin_filter call. Note that this routine must be called
 *      after all resource has been released by release_filter
 *      calls.
 */
static end_filter(stream)
u_long *stream;
{
        register struct KnjContext *kc;
        register struct KnjBuf *kb;
        static char *t;
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
                /*
                 * Establish a condition handler.
                 * R13 = FP.
                 */
                ((long *)_READ_GPR(13))[0] = handler;
                /*
                 * Remove all Kanji Buffers from queue.
                 * Deallocate memory.
                 */
                kc = (struct KnjContext *)*stream;
                while(_REMQUE((void *)kc->kc_kbinuse[0],(void **)&kb) != 2) {
                        kb->kb_size += KB_LENGTH;
                        lib$free_vm(&kb->kb_size,&kb);
                }
                while(_REMQUE((void *)kc->kc_kb[0],(void **)&kb) != 2) {
                        kb->kb_size += KB_LENGTH;
                        lib$free_vm(&kb->kb_size,&kb);
                }
                /*
                 * Release Kanji Context block.
                 */
                _REMQUE((void *)kc,(void **)&t);
                lib$free_vm(&kc->kc_size,&t);

                return(SS$_NORMAL);
        }
```

(次ページに続く)

例 3–1 (続き) UPCASE.C **の場合**

```
/*
 * HTON_FILTER
 *
 *      Translates from Host representation to Network representation.
 */
static hton_filter(stream,src,slen,dst,dlen)
u_long *stream;
u_char *src,**dst;
u_short *slen,*dlen;
{
        register struct KnjBuf *kb;
        register struct KnjContext *kc;
        register unsigned char *s,*d;
        register long sl,dl;
        long sdsc[2],ddsc[2];

        /*
         * Establish a condition handler.
         * R13 = FP.
         */
        ((long *)_READ_GPR(13))[0] = handler;

        kc = (struct KnjContext *)*stream;
        kb = kc->kc_kb[0];
        /*
         * If multi-buffer was requested in begin_filter,
         * we will remque Kanji buffer, then insque it
         * to in-use queue later. The buffer can be
         * replaced by larger one when we realize
         * the size of buffer is insufficient.
         */
        if(kc->kc_flags) {
                if(_REMQUE((void *)kb,(void **)&kb) == 2)
                        if((kb = GetBuf(kc,*slen)) == 0) {
                                *dlen = 0;
                                *dst = (char *)0;
                                return(SS$_INSFMEM);
                        }
        }
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
        s = src;
        d = kb->kb_data;
        dl = kb->kb_size;
        sl = *slen;
        if(sl == 0)
                goto end;

        if(sl > dl) {
                ReplaceKB(kc,kb,d,dl);
        }
        sdsc[0] = sl, sdsc[1] = s;
        ddsc[0] = sl, ddsc[1] = d;
        str$upcase(ddsc,sdsc);
end:
        /*
         * Insert Kanji Buf to in-use queue,


         * if release_filter should be called later.
         */
        if(kc->kc_flags)
                _INSQUE((void *)kb,(void *)kc->kc_kbinuse);
        /*
         * Give resultant buffer to user.
         */
        *dlen = sl;
        *dst = d;

        return(SS$_NORMAL);
}
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
/*
 * NTOH_FILTER
 *
 *      Translates Network representation to Host representation.
 */
static ntoh_filter(stream,src,slen,dst,dlen)
u_long *stream;
u_char *src,**dst;
u_short *slen,*dlen;
{
        register struct KnjBuf *kb;
        register struct KnjContext *kc;
        register unsigned char *s,*d;
        register long sl,dl;
        long sdsc[2],ddsc[2];

        /*
         * Establish a condition handler.
         * R13 = FP.
         */
        ((long *)_READ_GPR(13))[0] = handler;
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
kc = (struct KnjContext *)*stream;
kb = kc->kc_kb[0];
/*
 * If multi-buffer was requested in begin_filter,
 * we will remque Kanji buffer, then insque it
 * to in-use queue later. The buffer can be
 * replaced by larger one when we realize
 * the size of buffer is insufficient.
 */
if(kc->kc_flags) {
        if(_REMQUE((void *)kb,(void **)&kb) == 2) {
                if((kb = GetBuf(kc,*slen)) == 0) {
                        /*
                         * Woops no buffer available...
                         */
                        *dlen = 0;
                        *dst = (char *)0;
                        return(SS$_INSFMEM);
                }
        }

}
s = src;
d = kb->kb_data;
dl = kb->kb_size;
sl = *slen;
if(sl == 0)
        goto end;
```

(次ページに続く)

例 3–1 (続き) UPCASE.C の場合

```
        if(sl > dl) {
                ReplaceKB(kc,kb,d,dl);
        }
        sdsc[0] = sl, sdsc[1] = s;
        ddsc[0] = sl, ddsc[1] = d;
        str$upcase(ddsc,sdsc);
end:
        /*
         * Insert Kanji Buf to in-use queue,
         * if release_filter should be called later.
         */
        if(kc->kc_flags)
                _INSQUE((void *)kb,(void *)kc->kc_kbinuse);
        /*
         * Give resultant buffer to user.
         */
        *dlen = sl;
        *dst = d;

        return(SS$_NORMAL);
}

/*
 * RELEASE_FILTER
 *
 *      This routine is called to release resource. The buffers
 *      allocated by hton_filter or ntoh_filter routine must be
 *      deallocated, unless value of resource argument of begin_filter
 *      call was 1. When the value one for resource argument of
 *      begin_filter routine had been specified, release_filter
 *      must not be called.
 */
static release_filter(stream,c)
u_long *stream;
u_char *c;
{
        register struct KnjBuf *kb;
        register struct KnjContext *kc;
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
        /*
         * Establish a condition handler.
         * R13 = FP.
         */
        ((long *)_READ_GPR(13))[0] = handler;
        kc = *stream;
        kb = c - KB_LENGTH;


        {long z;_REMQUE((void *)kb,(void **)&z);}
        _INSQUE((void *)kb,(void *)kc->kc_kb);
        return(SS$_NORMAL);
}



/*
 * GETBUF
 *
 *      Allocates filter routine internal buffers. Ntoh_filter and
 *      hton_filter fills the buffer with resultant string, and
 *      give it to TCPIP application.
 */
static long newlength = 0;

static struct KnjBuf *
GetBuf(kc,min)
register struct KnjContext *kc;
long min;
{
        register struct KnjBuf *kb;
        long len;
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
                if(newlength == 0)
                        switch(kc->kc_application) {
                                case HLA_FTP:   newlength = 4096*2; break;
                                case HLA_TELNET:
                                case HLA_SMTP:
                                case HLA_LPR:
                                default:        newlength = 1024*2; break;
                        };
                len = (min * 125 / 100 + 511) & ~511;
                if(newlength < len)
                        newlength = len;
                onerror(lib$get_vm(&newlength,&kb), return(0));
                kb->kb_size = newlength - KB_LENGTH;
                return(kb);
        }

        /*
         * NEWBUF
         *
         *      This routine replaces a short buffer by a large buffer,
         *      which is newly allocated. Contents of a short buffer is
         *      copied to a large buffer.
         *      See also ReplaceKB macro.
         */
        static struct KnjBuf *
        NewBuf(kc,kb,l)
        register struct KnjContext *kc;
        register struct KnjBuf *kb;
        long l;
        {
                struct KnjBuf *kb0;

                kb->kb_size += KB_LENGTH;
                newlength = (kb->kb_size * 125 / 100 + 511) & ~511;
                if(iserror(lib$get_vm(&newlength,&kb0))) {
                        if(!kc->kc_flags)
                                _REMQUE(kb,&kb);
                        lib$free_vm(&kb->kb_size,&kb);
```

(次ページに続く)

例 3–1 **(続き)** UPCASE.C **の場合**

```
                return(0);
        }
        kb0->kb_size = newlength - KB_LENGTH;
        _MOVC3(l,kb->kb_data,kb0->kb_data);
        if(!kc->kc_flags) {
                _REMQUE(kb,&kb);
                _INSQUE(kb0,kc->kc_kb);
        }
        lib$free_vm(&kb->kb_size,&kb);
        return(kb0);
}

/*
 * HANDLER
 *
 *      The condition handler routine, which prevents TCPIP applications
 *      from fatal errors of filter routine.
 */
static handler(sig,mec)
u_long *sig;
u_long *mec;
{
        if(sig[1] == SS$_UNWIND)
                return;
        Display("Filter routine has detected fatal condition.");
        sys$putmsg(sig,0,0);
        mec[3] = sig[1];
        sys$unwind(&1,0);
}
```

4

標準漢字フィルタの仕様

4.1 7ビットJIS漢字フィルタ1 (JIS)

7 ビット JIS 漢字 (G0 集合[1]のみの使用を想定した漢字符号化方法) と DEC 漢字との変換を行います。

7 ビット JIS 側のエンコーディングは以下のものをサポートします。

```
(a) ASCII               ESC (B
(b) JIS X0201(LH)       ESC (J

(c) JIS X0208('78)      ESC $@
(d) JIS X0208('83)      ESC $B

(e) JIS X0201(RH)       ESC (I
                        SI,  SO
(f) JIS X0212           ESC $(D
```

1. ASCII コード・セットの変換

   ASCII と JIS X0201(LH) とを区別しません。すなわち (a) と (b) のどちらに対しても， DEC 漢字はエスケープ・シーケンスは使わず単に 0xxxxxxx で表示します。送信時は ESC (B を用います。

2. JIS X0208 コード・セットの変換

   JIS X0208('83) と JIS X0208('78) とを区別しません。すなわち (c) と (d) のどちらに対しても， DEC 漢字はエスケープ・シーケンスは使わず単に 1xxxxxxx 1xxxxxxx で表示します (未定義領域も含みます)。送信時は ESC $B を用い，1xxxxxxx 1xxxxxxx から 0xxxxxxx 0xxxxxxx へ変換します (未定義領域も含みます)。

[1] JIS X0202 情報交換用符号の拡張法参照

3. JIS X0212 コード・セットの変換

   JIS7 中に JIS X0212 の指示シーケンスがあると，それ以降ほかの文字セットの指示シーケンスがあるまで 0xxxxxxx 0xxxxxxx から SS3 1xxxxxxx 1xxxxxxx へ変換します。送信時は ESC $(D を用い， SS3 1xxxxxxx 1xxxxxxx から 0xxxxxxx 0xxxxxxx へ変換します (未定義領域も含みます)。

4. JIS X0201(RH) (半角カナ) コード・セットの変換

   JIS7 中に JIS X0201(RH) の指示シーケンスがあると，それ以降ほかの文字セットの指示シーケンスがあるまで，または SO から SI までの間 0xxxxxxx から SS2 1xxxxxxx へ変換します。送信時は SO， SI を用いて， SS2 1xxxxxxxx から 0xxxxxxx へ変換します (未定義領域も含みます)。

5. UDC(ユーザ定義文字， DEC 漢字) コード・セットの変換

   JIS7 側では， DEC 漢字の UDC(1xxxxxxx 0xxxxxxx) はすべて 0x2222 （全角の四角）に変換します。

6. C0 コード・セットの変換

   JIS7 中の 0x00-1f はその時のステートによらず C0 として扱いそのまま DEC 漢字の C0 とします。 DEC 漢字の C0(0x00-1f) のコードは (a) のシーケンスを付加して JIS 側へ出力します。

7. C1 コード・セットの変換

   DEC 漢字に含まれる SS2， SS3 以外の C1 コードは，同等の ESC コードと 7 ビット文字の組み合せに変換します。

8. (a) 〜 (f) 以外のエスケープ・シーケンス

   JIS7 側に (a) 〜 (f) 以外のエスケープ・シーケンスが現れた場合，そのエスケープ・シーケンスはそのまま DEC 漢字に送り，ステートはそれ以前のものを維持します。

4.2 7ビット JIS 漢字フィルタ 2 (JISM)

7 ビット JIS 漢字 (前節の JIS7 と同様に， G0 集合のみを使用した漢字符号， JIS X0201(RH) の符号化が異なる) と DEC 漢字との変換を行います。

JIS 側のエンコーディングは以下のものをサポートします。

```
(a) ASCII               ESC (B
(b) JIS X0201(LH)       ESC (J

(c) JIS X0208('78)      ESC $@
(d) JIS X0208('83)      ESC $B

(e) JIS X0201(RH)       ESC (I
                        SI,  SO
(f) JIS X0212           ESC $(D
```

1. ASCII コード・セットの変換

   ASCII と JIS X0201(LH) とを区別しません。すなわち (a) と (b) のどちらに対しても， DEC 漢字ではエスケープ・シーケンスは使わず単に 0xxxxxxx で表示します。送信時は ESC (J を用います。

2. JIS X0208 コード・セットの変換

   JIS X0208('83) と JIS X0208('78) とを区別しません。すなわち (c) と (d) のどちらに対しても， DEC 漢字ではエスケープ・シーケンスは使わず単に 1xxxxxxx 1xxxxxxx で表示します（未定義領域も含みます）。送信時は ESC $B を用いて 1xxxxxxx 1xxxxxxx から 0xxxxxxx 0xxxxxxx へ変換します (未定義領域も含みます)。

3. JIS X0212 コード・セットの変換

   JIS7 中に JIS X0212 の指示シーケンスがあると，それ以降ほかの文字セットの指示シーケンスがあるまで 0xxxxxxx 0xxxxxxx から SS3 1xxxxxxx 1xxxxxxx へ変換します。送信時は ESC $(D を用い SS3 1xxxxxxx 1xxxxxxx から 0xxxxxxx 0xxxxxxx へ変換します (未定義領域も含みます)。

4. JIS X0201(RH) (半角カナ) コード・セットの変換

   JIS7 中に JIS X0201(RH) の指示シーケンスがあると，それ以降ほかの文字セットの指示シーケンスがあるまで，または， SO から SI までの間 0xxxxxxx から SS2 1xxxxxxx へ変換します。送信時は ESC (I を用い， 1xxxxxxx から 0xxxxxxx へ変換します (未定義領域も含みます)。

5. UDC(ユーザ定義文字， DEC 漢字) コード・セットの変換

   JIS7 側では DEC 漢字の UDC(1xxxxxxx 0xxxxxxx) はすべて 0x2222 （全角の四角）に変換します。

6. C0 コード・セットの変換

   JIS7 中の 0x00-1f はその時のステートによらず C0 として扱いそのまま DEC 漢字の C0 とします。 DEC 漢字の C0(0x00-1f) のコードは (a) のシーケンスを付加して JIS7 側へ出力します。

7. C1 コード・セットの変換

   DEC 漢字に含まれる SS2， SS3 以外の C1 コードは，同等の ESC コードと 7 ビット文字の組み合せに変換します。

8. (a) 〜 (f) 以外のエスケープ・シーケンスの変換

   JIS7 側に (a) 〜 (f) 以外のエスケープ・シーケンスが現れた場合，そのエスケープ・シーケンスはそのまま DEC 漢字に送り，ステートはそれ以前のものを維持します。

4.3 シフト JIS 漢字フィルタ (SJIS)

シフト JIS と DEC 漢字との変換を行います。

シフト JIS 側のコード・セットは以下のものをサポートします。

```
JIS X0201(LH)      0xxxxxxx
JIS X0208('83)     1xxxxxxx+xxxxxxxx
JIS X0201(RH)      1xxxxxxx
```

1. JIS X0201(LH) コード・セットの変換

   受信/送信とも変換は行いません。

2. JIS X0208 コード・セットの変換

   シフト JIS および DEC 漢字の JIS X0208 に対応する部分はそれぞれの JIS X0208 コードに対応するように変換します (未定義部分も含みます)。

3. JIS X0212 コード・セットの変換

   DEC 漢字の JIS X0212(SS3 1xxxxxxx 1xxxxxxx) は，シフト JIS の 0x81a0 (全角の四角) に変換します。

4. JIS X0201(RH) (半角カナ) コード・セットの変換

   シフト JIS の JIS X0201RH (0xa1-df) は DEC 漢字の SS2(0x8e)+0xa1-df に，DEC 漢字の JIS X0201RH (SS2(0x8e)+0xa1-df) はシフト JIS の 0xa1-df に変換します。

5. UDC(ユーザ定義文字， DEC 漢字，シフト JIS) コード・セットの変換

   シフト JIS の UDC 領域は， DEC 漢字の 0xa2a2 (全角の四角) に変換します。DEC 漢字の UDC(1xxxxxxx 0xxxxxxx) はシフト JIS の 0x81a0 (全角の四角) に変換します。

6. C0 コード・セットの変換

   受信/送信とも変換は行いません。

7. C1 コード・セットの変換

   DEC 漢字に含まれる SS2， SS3 以外の C1 コードは，同等の ESC コードと 7 ビット文字の組み合せに変換します。

8. エスケープ・シーケンスの変換

   (他の C0 同様に) 受信/送信とも変換は行いません。

4.4 日本語 EUC 漢字フィルタ (UJIS)

日本語 EUC と DEC 漢字との変換を行います。

日本語 EUC 側のコード・セットは以下のものをサポートします。

```
CS0 ASCII              0xxxxxxx
CS1 JIS X0208('83)      1xxxxxxx+1xxxxxxx
CS2 JIS X0201(RH)       SS2+0xxxxxxx
CS3 JIS X0212           SS3+1xxxxxxx+1xxxxxxx
```

1. ASCII コード・セットの変換

   受信/送信とも変換は行いません。

2. JIS X0208 コード・セットの変換

   受信/送信とも変換は行いません。

3. JIS X0212 コード・セットの変換

   受信/送信とも変換は行いません。

4. JIS X0201(RH) (半角カナ) コード・セットの変換

   受信/送信とも変換は行いません。

5. UDC (ユーザ定義文字, DEC 漢字) コード・セットの変換

   DEC 漢字の UDC(1xxxxxxx 0xxxxxxx) の領域は 0xa2a2(全角の四角) に変換します。

6. C0 コード・セットの変換

   受信/送信とも変換は行いません。

7. C1 コード・セットの変換

   受信/送信とも変換は行いません。

8. エスケープ・シーケンスの変換

   (他の C0 同様に) 受信/送信とも変換は行いません。

索引

B

BEGIN_FILTER 3–5

C

control-block のデータ構造 3–10

E

END_FILTER 3–5

F

FTP での漢字フィルタの使用方法 2–1

H

HTON_FILTER 3–5

N

NTOH_FILTER 3–5

R

RELEASE_FILTER 3–5

S

SMTP での漢字フィルタの使用方法 2–4

T

TELNET での漢字フィルタの使用方法 2–3

U

UPCASE.C 3–25

カ

漢字フィルタの概要 1–1
- FTP 1–3
- SMTP 1–4
- TELNET 1–2
- リモート・プリント 1–5

漢字フィルタの使用方法 2–1
- FTP での 2–1
- SMTP での 2–4
- TELNET での 2–3
- リモート・プリントでの 2–5

シ

シフト JIS 漢字フィルタ (SJIS) 4–5
出力フィルタ・ルーチン 3–18

ソ

ソース・プログラム例
- UPCASE.C 3–25

ナ

7 ビット JIS 漢字フィルタ 1(JIS) 4–1
7 ビット JIS 漢字フィルタ 2(JISM) 4–3

ニ

日本語 EUC 漢字フィルタ (UJIS) 4–7
入力フィルタ・ルーチン 3–15

ハ

バッファ資源解放ルーチン 3–21
バッファ資源管理 3–7

フ

フィルタ・ストリーム 3–6
フィルタ・ストリーム解放ルーチン . . . 3–23
フィルタ・ストリーム初期化ルーチン 3–12
フィルタ設定ルーチン 3–5
フィルタ・ルーチン
出力フィルタ・ルーチン 3–18
入力フィルタ・ルーチン 3–15
バッファ資源解放ルーチン 3–21
フィルタ・ストリーム解放ルーチン 3–23
フィルタ・ストリーム初期化ルーチン 3–12
フィルタ設定ルーチン 3–5
例 3–25

リ

リモート・プリントでの漢字フィルタの使用方法 2–5

日本語 Compaq TCP/IP Services for OpenVMS
日本語機能の手引き

2002 年 9 月　発行

コンパックコンピュータ株式会社

〒 140-8641 東京都品川区東品川 2-2-24 天王洲セントラルタワー

電話　(03)5463-6600 (大代表)

AA-PUL3J-TE