XMLプロセッサにおけるマルチ文字エンコーディングスキーム処理機能の実装

阿　部　　司*

Implementing multiple character encoding scheme hander in XML processor

Tsukasa ABE

Abstruct

XML was developed for wide variety of applications. A software module called an XML processor is used to read XML documents and produce output stream for an application. XML processor not only read Unicode streame, but also should be read character stream based other encoding scheme. This paper shows implementing multiple character encoding scheme handler in XML processor. This handler can translate an encoded input stream to Unicode stream read by XML processor nucleus. And also this handler can be replaced dynamaticaly.

1．はじめに

HTMLの後継言語，汎用的データ表現によるデータベース，システム間の汎用的プロトコルとしてXMLが注目されている。XMLでは，国際化・多言語対応は基本的事項である。1998年の勧告案[1),2)]では，XMLを処理するXMLプロセッサは，入出力の文字エンコーディングスキームとしてUnicode[3)]およびISO/IEC 10646-1[4)]だけを解釈できれば良い。しかし，各国での文字エンコーディングスキームの利用状況からこれでは不十分である。従って，図1のように，ファイル等からの文字エンコーディングスキームをUnicodeに変換してXMLプロセッサ本体に入力し，XMLプロセッサ本体の出力を逆変換しファイル等へ出力する機能が必要である。日本でも，各種文字エンコーディングスキームがあり，Unicodeとの相互変換機能が必要である。

また，XML文書の文字エンコーディングスキーム指定と，先頭の数文字の推定による相互変換の切替え機能も重要な課題である。

本研究では，各種文字エンコーディングスキームとUnicodeとの相互変換機能と切替え機能のXMLプロセッサへの実装を行った。

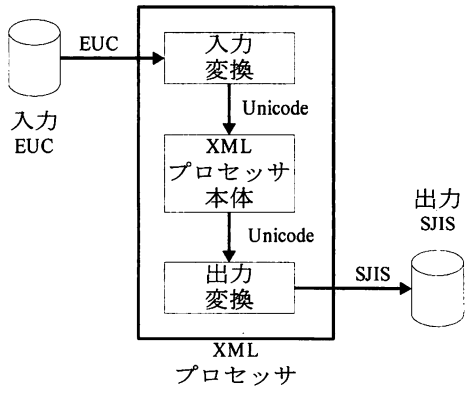

図1　エンコーディング変換の実装

2．文字エンコーディングスキーム変換と切替え

2．1　文字エンコーディングスキーム

一般に，計算機や通信で使用されている文字の集合を，文字コードや文字集合と呼んでおり，JIS X 0201[5)]，JIS X 0208[6)]とJIS X 0212[7)]を元にしたシフトJISやEUCなどが日本で使用されている。しかし，シフトJISとEUCは文字に割り当てられた値が異なるだけで，JIS X 0201とJIS X 0208での値を，異なる数値体系に置き換えたものである。従って，元の数値体系は同じで，その表現方法としてシフトJISやEUCがある。つまり，文字集合を表現する符号化文字集合と，それを実際に計算機等内部の数値として表現する文字エンコーディングスキームと区別することができる[8)]。

今回は，表1に示す符号化文字集合と文字エンコーディングスキームに対するUnicodeの相互

＊　助教授　情報工学科

変換機能の実装を行った。表から符号化文字集合と文字エンコーディングスキームとの対応は大きく分けると3種類になる。

(1) JIS X 0201とJIS X 0208に対応するシフトJIS

(2) JIS X 0201, JIS X 0208, JIS X 0212に対応するEUCとISO-2022-JP-2[9]

(3) Unicode, ISO/IEC10646-1とJIS X 0221[10]に対応するUTF-7[11], UTF-8[12], UTF-16/UCS-2[3),4)](以降UTF-16と記述する), UCS-4[3),4)]

表1 文字エンコーディングスキーム

符号化文字集合	文字エンコーディングスキーム	エンディアン
JIS X 0201 JIS X 0208	シフトJIS	なし
JIS X 0201 JIS X 0208 JIS X 0212	EUC	なし
	ISO-2022-JP-2	なし
ASCII	ASCII	なし
Unicode ISO/IEC 10646-1 JIS X 0221	UTF-7	なし
	UTF-8	なし
	UTF-16/UCS-2	あり
	UCS-4	あり

なお, ASCIIは別になっているが, UTF-8の0x00から0x7fまではASCIIと全く同じである。

シフトJISはMS漢字コードとも呼ばれ, Windows95では, OS内部でも使用されている。EUCは主にUNIXで使用されており, 国際化対応で, 日本以外の言語のEUCも存在している。

ISO-2022-JP-2は, メールの国際化・多言語対応のため開発され, 複数の符号化文字集合をエスケープシーケンスで切り替える。

Unicode, ISO/IEC 10646-1とJIS X 0221は, 全世界の符号化文字集合を一つの符号化文字集合で表現し, それまでの符号化文字集合と全く異なるコードポイントを持つ符号化文字集合である。

UTF-7とUTF-8は, それぞれ7, 8ビット単位の複数オクテットでUCS-4を表現し, UTF-16は2または4オクテットでUCS-4を表現する。

UTF-16とUCS-4は, オクテットの転送順の規定がないため, エンディアンを考慮する必要がある。今回は4種類のエンディアンに対応している。

2．2 シフトJISの実装上の課題

シフトJISとUnicodeとの相互変換の実装には以下のような課題がある。

(1) 制定年が異なるJIS X 0208：1978とJIS X 0208：1983及び1990を区別できない。

(2) メーカで, ベースの制定年が異なっている。

(3) メーカ独自の文字を追加している[13)]。

(4) メーカにより, シフトJISとUnicodeとの相互変換表が異なっている[8)]。

(1)については, 外部からの指定が必要になる。(2)と(3)については, 各メーカのハード・ソフトの入手・調査・検証が困難なため, (4)の問題に含めて解決した。(4)については, The Unicode Consortiumからの変換表と, 入手可能システムの調査より, 以下に示す副変換表を選択できる。

(1) JIS X 0208の定義文字のみの副変換表。

(2) IBM社のシフトJISの副変換表。

(3) CP932の副変換表。

(4) CP932とNEC選定IBM拡張文字の副変換表。

(5) CP932とIBM拡張文字の副変換表。

(6) JDK 1.1.8互換の副変換表。

(7) Windows NT 4.0互換の副変換表。

2．3 文字エンコーディングスキームの指定と切替え

文字エンコーディングスキームの決定には, 明示的指定方法と, 自動的識別による方法がある。今回は, 以下の手順で決定する。

(1) XMLプロセッサを起動するときに, コマンドラインオプションとして指定する。

(2) (1)がない場合は, XML文書の先頭数文字で決定する。ただし, シフトJIS, EUC, ISO-2022-JP-2, UTF-8は区別できないので, XML勧告案に規定されているUTF-8と決定する。

(3) 最終的に, XML文書のXML宣言またはテキスト宣言で決定する。以下のように文字エンコーディングスキームの指定ができる。

＜?xml...encoding="EUC-JP"?＞

上記の(2), (3)では, 何文字か入力後, 文字エンコーディングスキームを切替える。この時, 入

力ストリームの乱れないことが重要である。

3．ソフトウェア構成

3．1 ターゲットシステムとプログラミング言語

ターゲットシステムは，Windows NT，プログラミング言語はC++。ホストシステムはWindows NT 4.0，コンパイラはMS Visual C++ 5.0である。

3．2 ソフトウェアの階層構成

今回作成したシステムはXMLプロセッサと，文字エンコーディングスキーム変換の大部分を実装したクラスライブラリからなっている。XMLプロセッサは，コマンドラインとXML宣言での文字エンコーディングスキームの解読を行い。実際の変換と切替えはクラスライブラリが行っている。また，XMLプロセッサクラスの入出力と内部処理はUTF-16とUCS-4で行っている。

図2に，XMLプロセッサの階層構成を示す。[1]は，[2]のXMLプロセッサクラスを実際に動作させるためのプログラムであり，今回は，XML文書構造を表示するプログラムを作成した。[2]と[3]は図1のXMLプロセッサ本体に対応するクラスである。[4]から[7]までが，入出力ストリームを処理する一般クラスライブラリである。

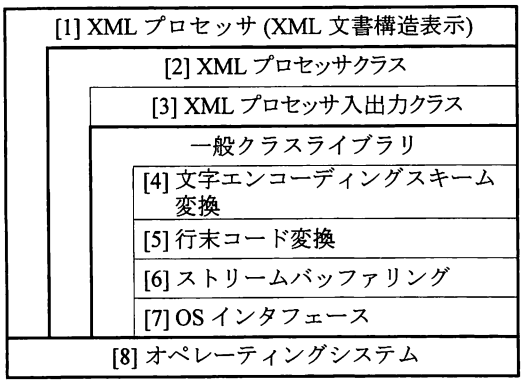

図2 XMLプロセッサの階層構造

3．3 XMLプロセッサクラス

XMLプロセッサクラスの名前はPCXMLParserである。各種システムの組み込むため，これ自体クラスライブラリであり，厳密には一般クラスライブラリの一部である。文字エンコーディングスキームの決定と切り替えについて以下に示す。

(1) 文字エンコーディングスキームの決定

XMLプロセッサクラスは，図2の[3]のXMLプロセッサ入出力クラスのPCXMLInputSourceで入出力を行う。ファイルの場合は，派生クラスのファイル入出力クラスPCXMLFileInputを使用する。ファイル入出力クラスは，ファイルのオープン後，XML文書の先頭数文字から文字エンコーディングスキーム変換を決定する。

また，XML文書のXML宣言，テキスト宣言からも文字エンコーディングスキームを決定する。

(2) 文字エンコーディングスキームの切替え

文字エンコーディングスキームが決定したら，XMLプロセッサ入出力用クラスに変換クラスの切替えを指示する。

3．4 一般クラスライブラリ

一般クラスライブラリの各階層について説明する。以下の説明ではファイルからXMLプロセッサクラスへの入力について述べる。

［4］ 文字エンコーディングスキーム変換クラス

文字エンコーディングスキーム変換の中核部分であり，基本クラスはPCTranslatorである。PCTranslatorからの派生クラスと，文字エンコーディングスキームとの対応を表2に示す。クラスによっては，エンディアンと副変換表が必要である。文字エンコーディングスキーム変換クラスは，内部で行末コード変換クラスを呼出し，入力をUCS-4に変換して，XMLプロセッサクラスに渡す。

表2 PCTranslatorからの派生クラス

クラス名	文字エンコーディングスキーム	エンディアン	副変換表
PCTransASCII	ASCII	なし	なし
PCTransUCS4	UCS-4	あり	なし
PCTransUTF16	UTF-16/UCS-2	あり	なし
PCTransUTF7	UTF-7	なし	なし
PCTransUTF8	UTF-8	なし	なし
PCTransSJIS	シフトJIS	なし	あり
PCTransEUC	EUC	なし	あり
PCTransISO2022JP	ISO-2022-JP-2	なし	あり

［5］ 行末コード変換クラス

行末コードをC言語の標準行末コードである¥n（0x0a）に変換する。基本クラスはPCLineTermである。これから，表3に示すように4種類の行末コードに対応する派生クラスがあるが，入力に関しては，自動識別が容易なため，自動識別派生クラスPCLineTermAutoを一般に使用する。

表3 PCLineTermからの派生クラス

クラス名	行末コード	入出力
PCLineTermCRLF	0x0d,0x0a	入出力
PCLineTermNL	0x0a	入出力
PCLineTermCR	0x0d	入出力
PCLineTermAuto	自動	入力のみ

［6］ ストリームバッファリングクラス

入出力効率を向上するため，OSインタフェースとの入出力をバッファリングして，1文字単位で，行末コード変換クラスに渡す。基本クラスはPCPointerである。派生クラスについて以下に述べる。

［i］ 一般ファイルバッファリング入出力クラス（PCBufIO）

OSの一般的なファイル入出力APIによる文字をバッファリングする。標準入出力に使われるクラスである。

［ii］ メモリマップトファイル入出力クラス（PCPointerMap）

ファイルを仮想メモリ空間にマップする機能を利用している。メモリに対するアクセスと同様に，ファイルに対しランダムアクセスできるため，ポインタの操作だけでファイルから入出力可能である。

［7］ OSインタフェース

OSにより構成が異なるが，WindowsNT用では，以下のクラスから構成されている。

［i］ 一般ファイル入出力クラス（PCFile）

ファイルを識別するクラスである。ストリームバッファリングクラスの両派生クラスで使用されている。

［ii］ メモリマップトファイルクラス（PCFileMap）

［i］のファイルを，仮想メモリ空間にマップする。メモリマップトファイル入出力クラスで使用されている。

4. 変換と切り替えの実装

4．1 文字エンコーディングスキーム変換の開始

(1) XMLプロセッサのXML文書オープン

リスト1に，XMLプロセッサクラスでXML文書をオープンする関数Parseを示す。引数のXML文書ファイル名（aFileName），外部文字エンコーディングスキーム（aEncoding），エンディアン（aEndian）により，ファイル入出力クラスPCXMLFileInputで，XML文書をオープンする。

リスト1 XMLプロセッサクラスでのXML文書のオープン

```
PCXMLParser::Parse (
  const utf16_t* aFileName,
  EncodingType   aEncoding
  EndianType     aEndian
{
  PCXMLFileInput*Input
  Input = new PCXMLFileInput;
  Input->Open(aFileName,
              aEncoding,
              aEndian);
  }
```

(2) ファイル入出力クラスのXML文書オープン

リスト2に，ファイル入出力クラスでXML文書をオープンする関数Openを示す。まず，メモリマップトファイル入出力クラスの関数OpenによりXML文書をオープンし，行末コード変換クラスに設定する。次に，関数SelectEncodingにより，XML文書の先頭1文字から文字エンコーディングスキームを決定するが，不明の場合はUTF-8としておく。

決定された文字エンコーディングスキームから，関数SelectTranslatorにより，文字エンコーディングスキーム変換クラスを選択し，リスト3に示すデータメンバmTranslatorに設定する。次に，関数Openで行末コード変換クラスを設定する。

リスト2　ファイル入力クラスでのXML文書のオープン

```
PCXMLFileInput::Open (
  const utf16_t* aFileName,
  EncodingType   aEncoding,
  EndianType     aEndian
  ) {
  PCPointer* Point
    = new PCPointerMap;
  PCLineTerm* LineTerm
    = new PCLineTermAuto;
  Point->Open(aFileName);
  LineTerm->Open(Point);
  Point->SelectEncoding(aEncoding,
                        aEndian, L'<');
  if (aEncoding == EncodingUndefined)
    aEncoding = EncodingUTF8;
  mTranslator
    = SelectTranslator(aEncoding);
  mTranslator->Open(LineTerm, aEndian);
  }
```

リスト3　ファイル入力クラスのデータメンバ

```
class PCXMLFileInput :
  public PCXMLInputSource {
private:
  PCTranslator* mTranslator;
  }
```

(3) 外部文字エンコーディングスキームの決定

リスト4に，外部文字エンコーディングスキームを決定する関数SelectEncodingを示す。以下の優先度にしたがって外部文字エンコーディングスキームとエンディアンを設定する。

リスト4　文字エンコーディングスキームの決定

```
PCPointer::SelectEncoding (
  EncodingType&  aEncoding,
  EndianType&    aEndian,
  utf16_t        aPattern
  ) {
  ucs4_t I4 = _GetQuadChar(),
  utf16_t I2 = _GetDoubleChar();
  if (I4==UCS4BOMBigEndian) {
    aEndian  = BigEndian;
    aEncoding= EncodingUCS4;
    }
  else if (I4==aPattern) {
    aEndian  = BigEndian;
    aEncoding= EncodingUCS4;
    }
  else if (I2==UTF16BOMBigEndian) {
    aEndian  = BigEndian;
    aEncoding= EncodingUTF16;
    }
  else if (I2==aPattern) {
    aEndian  = BigEndian;
    aEncoding= EncodingUTF16;
    }
  }
```

[1] UCS-4のバイトオーダーマークにより，UCS-4とエンディアンを設定する。

[2] 引数で指定されたパターン（aPattern）により，UCS-4とエンディアンを設定する。

[3] UTF-16のバイトオーダーマークにより，UTF-16とエンディアンを設定する。

[4] 引数で指定されたパターン（aPattern）により，UTF-16とエンディアンを設定する。

[5] いずれにも一致しない場合は文字エンコーディングスキームとエンディアンは変更しない。

関数_GetQuadChar，_GetDoubleCharは，ストリームバッファリングクラスから，それぞれ4，2オクテット入力する関数である。

(4) 変換クラスの選択

リスト5に，文字エンコーディングスキーム変換クラスを選択する関数SelectTranslatorを示す。引数から，適当な文字エンコーディングスキーム変換クラスを生成する。不明時はUTF-8の文字エンコーディングスキーム変換クラスを生成する。

リスト5　文字エンコーディングスキーム変換クラスの選択

```
PCTranslator* SelectTranslator (
  EncodingType aEncoding
  ) {
  PCTranslator* Translator;
  switch (aEncoding) {
  case EncodingASCII :
    Translator = new PCTransASCII;
    break;
  case EncodingISO2022JP :
    Translator = new PCTransISO2022JP;
    break;
  case EncodingUTF16 :
    Translator = new PCTransUTF16;
    break;
  case EncodingUTF8 :
  default :
    Translator = new PCTransUTF8;
    break;
    }
  return Translator;
  }
```

4．2　文字エンコーディングスキームの変換

文字エンコーディングスキームがシフトJISの場合の変換について説明する。

(1) ファイル入出力クラスでの文字入力

リスト6に，ファイル入出力クラスの文字入力関数 GetChar を示す。XML プロセッサクラスでは，全てこの関数から文字を入力しており，この関数からの文字は UCS-4である。動作は，文字エンコーディングスキーム変換基本クラスの関数 GetUCS4Char を呼び出すだけである。

リスト6　ファイル入力クラスの文字入力

```
virtual long
PCXMLFileInput::GetChar (void) {
  return mTranslator->GetUCS4Char(); };
```

(2) 文字エンコーディングスキーム変換基本クラスでの文字入力

リスト7に，文字エンコーディングスキーム変換基本クラスの文字入力関数 GetUCS4Char を示す。文字エンコーディングスキーム変換に対応する派生クラスの仮想関数_GetUCS4Char を呼出す。

リスト7　文字エンコーディングスキーム変換基本クラスの文字入力

```
long PCTranslator::GetUCS4Char(void) {
  return _GetUCS4Char(); };
```

(3) 文字エンコーディングスキーム変換派生クラスでの文字入力

リスト8に，文字エンコーディングスキーム変換派生クラスの文字入力関数__ GetUCS4Char を示す。関数 IsReadEnd によりファイルの終わりであれば EOF を返す。次に，4．1(1)で設定された行末コード変換派生クラスの関数 GetChar で文字を入力し，関数 IsLeadInSJIS により文字がシフト JIS の第1文字かを判定する。

第1文字であれば，もう一度関数 GetChar で第2文字を入力し，関数 ToWideChar でワイド文字に変換する。次に，関数 SJISToJIS によりシフト JIS から JIS X 0208に変換した後，関数 JISX0208ToUCS4により UCS-4に変換する。この時，副変換表による変換も行われる。

第1文字でなければ，関数 JISX0201ToUCS4 により UCS-4に変換する。この時，副変換表による変換も行われる。

リスト8　文字エンコーディングスキーム変換派生クラスの文字入力

```
long PCTransSJIS::_GetUCS4Char (void) {
  long UCS4Char;
  if (IsReadEnd())
    UCS4Char = EOF;
  else {
    char Char = mLineTerm->GetChar();
    if (IsLeadInSJIS(Char)) {
      char Trail; wchar_t Wide;
      Trail = mLineTerm->GetChar();
      Wide = ToWideChar(Char, Trail);
      Wide = SJISToJIS(Wide);
      UCS4Char = JISX0208ToUCS4(Wide);
      }
    else
      UCS4Char = JISX0201ToUCS4(Char);
    }
  return UCS4Char;
  }
```

(4) 行末コード変換クラスでの文字の入力

リスト9に，行末コード変換派生クラスでの文字入力関数 GetChar を示す。行末コード変換基本クラスの文字入力関数 GetChar と文字先読み関数 LookUpChar により，行末コードを自動識別する。

リスト9　行末コード変換派生クラスの文字入力

```
int PCLineTermAuto::GetChar (void) {
  int Next, Char=PCLineTerm::GetChar();
  if (Char == '\r') {
    Next = PCLineTerm::LookUpChar();
    if (Next == '\n')
      PCLineTerm::GetChar();
    Char = '\n';
    }
  return Char;
  }
```

リスト10にように，行末コード変換基本クラスでの文字入力の関数 GetChar では，ストリームバッファリング基本クラスの GetChar を呼び出して，文字を入力する。

リスト10　行末コード変換基本クラスの文字入力

```
int PCLineTerm::GetChar (void) {
  return mPointer->GetChar(); };
```

(5) ストリームバッファリング基本クラスでの文字の入力

リスト11に，ストリームバッファリング基本クラスでの文字の入力関数 GetChar を示す。関数 IsReadEnd により，バッファに文字が残ってい

なければ関数Fillにより，バッファに文字を読み込む。次に関数Readで1文字読み，ポインタを1文字分進め，読み込んだ文字を返す。

リスト11 ストリームバッファリング基本クラスの文字入力

```
int PCPointer::GetChar (void) {
  if (IsReadEnd()&&Fill()==StatusEOF)
    return EOF;
  char Char = Read();
  SeekRead((RelOffType)1);
  return Char;
  }
```

この関数内で呼び出されているIsReadEndのような関数はストリームバッファリング派生クラスで仮想関数として定義されている。従って，派生クラスではIsReadEndのような関数だけを実装し，関数GetCharは派生クラスで実装する必要はない。

4．3 文字エンコーディングスキームの切替え

文字エンコーディングスキームがXML文書のXML宣言で指定された場合，文字エンコーディングスキームの切替えについて以下に述べる。

(1) XMLプロセッサクラスでの文字エンコーディングスキームの検出と切替え

リスト12に，XMLプロセッサクラスで文字エンコーディングスキームを検出・切替える関数ParseXMLDeclを示す。属性名を返す関数Nameの値がencodingであれば，属性値をEncodingType型に変換する。次に，元の文字エンコーディングスキームと異なっていればファイル入出力クラスの関数ChangeTranslatorにより文字エンコーディングスキーム変換クラスを切替える。

リスト12 XMLプロセッサクラスでの文字エンコーディングスキーム検出と切替え

```
PCStatus PCXMLParser::ParseXMLDecl (
  PCXMLInputSource& aInput
  ) {
  EncodingType New, Last;
  Last = aInput.Encoding();
  if (Name() == L"encoding") {
    EncodingStringToType(New,Value());
    if (New != Last)
      aInput.ChangeTranslator(New);
    }
  }
```

(2) ファイル入出力での文字エンコーディングスキームの切替え

リスト13に，ファイル入出力クラスでの文字エンコーディングスキーム切替え関数ChangeTranslatorを示す。文字エンコーディングスキーム変換クラス（以降，変換クラス）の関数ChangeTranslatorを呼出し，新しい変換クラスを生成して，元の変換クラスに設定されていた行末コード変換クラス等を引き継ぐ。次に，元の変換クラス破棄し，データメンバmTranslatorに新しい変換クラスを設定する。

リスト13 ファイル入力クラスでの文字エンコーディングスキーム切替え

```
PCXMLFileInput::ChangeTranslator (
  EncodingType aEncoding,
  EndianType   aEndian
  ) {
  PCTranslator* New;
  New = mTranslator
    ->ChangeTranslator(aEncoding,
                       aEndian);
  delete mTranslator;
  mTranslator = New;
  }
```

(3) 文字エンコーディングスキーム変換クラスでの変換クラスの生成

リスト14に，文字エンコーディングスキーム変換クラスでの切替え関数ChangeTranslatorを示す。関数SelectTranslatorで文字エンコーディングスキーム変換クラスを選択し，関数Openにより行末コード変換クラスを新しい文字エンコーディングスキーム変換クラスに移動する。これにより，すでに何文字かストリームから読み込んでいても，同じ時点から変換が可能である。

リスト14 文字エンコーディング変換クラスでの文字エンコーディングスキーム切替え

```
PCTranslator*
PCTranslator::ChangeTranslator (
  EncodingType aEncoding,
  EndianType   aEndian
  ) {
  PCTranslator* New;
  New = ::SelectTranslator(aEncoding);
  New->Open(mLineTerm, aEndian);
  mLineTerm = NULL;
  return New;
  }
```

5．評　価

5．1　文字エンコーディングスキームの変換

(1) 文字エンコーディングスキームが決定した時点で，変換クラスの動的生成が可能である。

(2) 文字エンコーディングスキーム変換派生クラスでは，文字エンコーディングスキームに対応した変換だけに集中できる。

(3) 階層化によって，文字エンコーディングスキームの変換を，このクラスだけに閉じ込めたため，このクラス内の行末コード変換クラス等には汎用的なクラスが利用できる。

(4) 文字エンコーディングスキームが8種類，副変換が7種類，UTF-16とUCS-4では4種類のエンディアンに対応可能である。

5．2　文字エンコーディングスキームの切替え

文字エンコーディングスキーム変換クラスを動的に生成することが可能で，階層化により行末コード変換クラス等とは完全独立である。文字エンコーディング変換クラスのみを単純に入替えるだけで，変換を切替えることができる。

5．3　XMLパーザ

XMLパーザも，一般ライブラリとなっており，XMLによる各種応用プログラムに利用できる。XMLパーザに，XML文書の文字エンコーディングスキームの変換と切替え機能が備わっているので応用プログラムで用意する必要はない。

6．おわりに

文字エンコーディングスキームの変換をクラス内への閉じ込めにより，他階層からの独立性を高めることができた。また，対応文字エンコーディングスキームの種類も多い。今後は，UNIXなど他のOSへの移植と，今回作成したXMLパーザクラスを組み込んだシステムの研究を行う。

7．参考文献

1) World Wide Web Consoritum : Extensible Mark-up Language (XML) 1.0 W3C Recommendation, World Wide Web Consoritum, 1998

2) XML/SGMLサロン：標準XML完全解説，技術評論社，1998

3) The Unicode Consortium : The Unicode Standard Version 3.0, The Unicode Consortium, 1999

4) ISO/IEC : ISO/IEC 10646-1 Information-technology－Universal Multiple Octet Coded Character Set (UCS) －PartI : Architectu-reand Basic Multilingual Plane, ISO/IEC, 1993

5) 日本規格化協会：7ビット及び8ビットの情報交換用符号化文字集合JIS X 0201，日本規格化協会，1997

6) 日本規格化協会：7ビット及び8ビットの2バイト情報交換用符号化漢字文字集合JIS X 0208，日本規格化協会，1997

7) 日本規格化協会：情報交換用漢字符号－補助漢字JIS X 0212，日本規格化協会，1990

8) 川俣晶：パソコンにおける日本語処理／文字コードハンドブック，1999

9) Masataka Ohta,Ken'ichi Handa : RFC1554 ISO-2022-JP-2 : Multilingual Extension of ISO-2022-JP, Internet Activities Board, 1993

10) 日本規格化協会：国際符号化文字集合(UCS)－第1部体系及び基本多言語面JIS X 0221,

11) David Goldsmith, Mark Davis : RFC1642 UTF-7 A Mail-Safe Transformation Format of Unicode, Internet Activities Board, 1994

12) Francois Yergeau : RFC2279 UTF-8, atrans formation format of ISO10646, Internet Activities Board, 1998

13) 上田純美礼：Windows NTに実装されたUNICODE（第1回日本語の文字コードの現状），ASCII, SUPER ASCII 1995年10月

（平成11年11月29日受理）