宮坂 優大

株式会社ミガロ.

システム事業部 システム1課

[Delphi/400]

Delphi/400バージョンアップに伴う文字コードの違いと制御

略歴
1982年11月19日生まれ
2006年 近畿大学 理工学部卒業
2006年4月 株式会社ミガロ. 入社
2006年4月 システム事業部配属

現在の仕事内容
主にDelphi/400を利用したシステムの受託開発とMKSサポートを担当している。Delphiおよび Delphi/400のスペシャリストを目指して精進する毎日。

1.はじめに

2015年にWindowsの大きな変革となるWindows 10がリリースされたことにより、ここ数年でDelphi/400のアプリケーションも、Windows 10への正式対応を目的にバージョンアップを実施する企業が増えている。

Delphiはバージョンの互換性が非常に高い言語だが、バージョンアップでの大きなポイントの1つが文字コードである。Windows 9Xなど旧日本語環境のWindowsは、Ansi（Shift_JIS）を標準文字コードとして扱ってきたが、最近のWindowsではUnicodeが標準になっている。Delphi言語も同様に、以前はAnsiが標準文字コードであったが、最近のバージョンではUnicodeに変わっている。

プログラムには文字を扱うコードが非常に多く、その基盤となる文字コードが変わると、アプリケーションの動作も影響を受ける可能性が高い。そのため、バージョン互換が高いDelphi言語であっても、標準文字コードが異なるバージョン間でのバージョンアップでは、影響を考慮した対応が必要となる。

本稿では、こうしたDelphiのバージョンアップで重要なポイントとなる文字コードの扱いについて、違いや制御方法を考察する。

2.Delphi言語で扱う文字コード

冒頭で述べたように、Delphiのバージョンアップで注意すべき重要なポイントとして、文字コードが挙げられる。Delphiが扱う文字コードについて、バージョンで整理すると、Delphi 2007以前ではAnsi型（AnsiString）、Delphi 2009以降ではUnicode型（UnicodeString）が標準となっている。

具体的にはUnicode対応により、プログラム上のString型、Char型、PChar型の標準仕様が変更されている。そのため、Delphi 2007以前のバージョンからDelphi 2009以降のバージョンへプログラムをバージョンアップする場合には、文字コードへの配慮が必要となる。

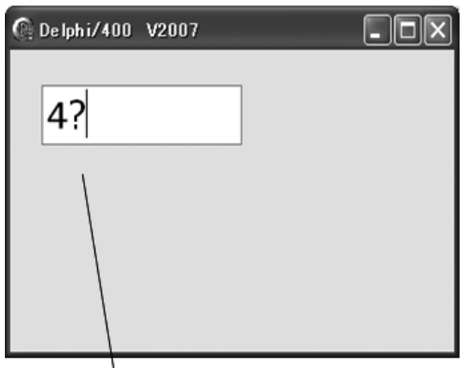
Unicodeとは、符号化文字集合や文字符号化方式などを定めた文字コードの規格である。UnicodeはAnsiと比べると、使える文字種類が非常に多く、世界中で表現されているほぼすべての文字に対応できる。

たとえば、「m^3」という記号文字はAnsiString型には存在しないが、UnicodeString型には存在している。これをDelphiアプリケーション上で入力すると、Unicodeに対応していないDelphi 2007以前のバージョンでは「?」で表示されるが、Unicodeに対応したバージョンでは正しく表示できる。【図1】

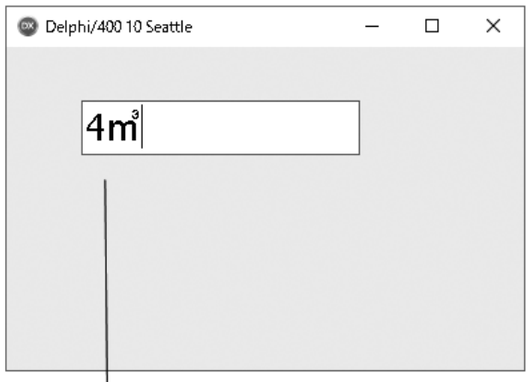
Unicodeについて、もう少し補足する。Unicodeの制御文字、文字、記号にはそれぞれに数値が割り当てられており、それをコードポイントと呼んでいる。たとえば、“A”のコードポイントは65（バイナリでは41）、“B”のコードポイントは66（バイナリでは42）と、それぞれの文字に一意のコードポイントが割り当

図1

AnsiStringとUnicodeString



V2007（AnsiString）
では表示できない



10 Seattle（UnicodeString）
では表示可能

図2

AnsiStringとUnicodeStringの桁数とバイト数

AnsiString型

文字	A	B	あ		い	
バイト	1	2	3	4	5	6
バイナリ	41	42	82	A0	82	A2

UnicodeString型

文字	A		B		あ		い	
バイト	1	2	3	4	5	6	7	8
バイナリ	00	41	00	42	82	A0	82	A2

てられている。

ここで重要なのは、UnicodeString 型ではそれぞれの文字が 2 バイトで表現される点である。AnsiString 型では半角文字は 1 バイトで表現されるが、UnicodeString 型では半角・全角という概念がない。この変更のため、文字列のバイト数で計算していたロジックは、【図 2】のように Ansi と Unicode で計算や動作が一致しなくなる可能性がある。これが大きな影響点といえる。

ただし標準の文字コードが変わっているだけなので、これまで使用していた Ansi 文字列は「AnsiString」と明示的にコーディングすれば、新しい Delphi 上でも使用できる。また同様に、古い Char 型、PChar 型 は AnsiChar 型、PAnsiChar 型として使用できる。

実際に文字コードの違いで、どのような挙動の差異があるかを調べるため、【図 3】【ソース 1】のプログラムを Delphi 2007 と Delphi 10 Seattle で作成して比較した。

画面に TEdit と TButton を配置し、ボタンを押すと入力した文字列の 1 ～ 4 バイト目を Copy 関数で取得し、ShowMessage 関数で表示する。ここでは Delphi 2007 のアプリケーションを①、Delphi10Seattle のアプリケーションを②として比較する。

画面項目で「あいうえお」と入力してボタンを押すと、①のアプリケーションでは「あい」と表示される。【図 4】

しかし、②のアプリケーションで同じことを実行しても結果が異なる。同じように「あいうえお」と入力し、ボタンを押すと「あいうえ」と表示される。Delphi 2007 とは異なり、1 ～ 4 バイト目ではなく、1 ～ 4 文字目を取得していることがわかる。【図 5】

このように同じプログラムロジックであっても、文字コードの違いによって動作が異なることがある。では②のアプリケーションで、①のアプリケーションと同じように動作させる方法を説明する。

単純な方法としては、UnicodeString 型を AnsiString 型で扱うことで解消できる。Copy 関数に渡す文字列を AnsiString 型で明示的にキャストすることで、従来の AnsiString 型として文字列を扱える。【ソース 2】

修正したソースで再度実行すると、①のアプリケーションと同じ文字列で表示されることがわかる。【図 6】

ただし、AnsiString 型でキャストすると、UnicodeString 型では扱える特殊な文字が扱えなくなるので、デメリットも理解しておく必要がある。

IBMi 側の環境が日本語 Unicode を取り扱う CCSID1399 であれば、UnicodeString 型のまま扱ったほうがよいが、従来の 5026 ／ 5035 の CCSID であれば、もともと UnicodeString の文字種類を扱えないので、AnsiString で扱っても問題ない。【図 7】

Delphi 2007 以前から Delphi 2009 以降にバージョンアップする際には、こうした文字コードの制御を考慮してプログラムを変更すれば、スムーズに実施できる。

3.文字コードの違いによる制御ポイント

前章では文字コード自体の違いについて考察したが、文字コードが変わることで違いが発生する点はほかにもある。それはプロパティや関数の動作である。

たとえば TEdit では、入力できる最大桁数を制御する MaxLength プロパティがある。これもバイト数単位から文字数単位でカウントされるため、全角文字を入力した時にこれまでと挙動が異なる。ここでは TEdit で MaxLength プロパティを 5 で設定した例を説明する。【図 8】

この TEdit に対して、「123 あい」と入力する。Delphi 2007 では「123 あ」と入力した段階で規制がかかり、入力できなくなる。【図 9】

しかし、Delphi 10 Seattle の同じアプリケーションでは、「123 あい」とすべて文字を入力できるため、挙動が違っていることがわかる。【図 10】

これは Unicode なので、文字数としてカウントされている。これと同じ動作になるのが、文字長を判断する Length 関数である。「123 あいう」という文字列に対して、Length 関数で取得できる結果値は Delphi のバージョンによって異なる。

Delphi 2007 ではバイト数で算出されているため、9 が取得できるが、Delphi 10 Seattle では文字数で算出されるため、6 が取得される。同じ挙動に合わせる対処方法としては、前章で説明したように、対象の文字列を AnsiString 型でキャストすればよい。

関数については、Delphi 2009 以降でも互換性を考慮し、Ansi 専用の関数が別途用意されている場合もある。それらの関数は、「System.AnsiStrings」ユニットに定義されている。

たとえば、StringReplace や Upper/LowerCase、StuffString などよく使われる関数が多数含まれている。プログラムの Uses 節に「System.AnsiStrings」を追加すれば、これらの Ansi 型関数を使用できる。

文字列を扱う関数で動作が違う場合には、互換の Ansi 関数が用意されていることが多いので、プログラムを作り変える前に、このユニットや WindowsAPI を確認するようお勧めする（WindowsAPI 関数についても、Microsoft 社が Ansi 互換の関数を多数用意している）。

4.おわりに

本稿では Delphi のバージョンによって異なる文字コードや制御方法について考察し、対処ポイントをまとめた。

文字コード自体は、普段のプログラミングではそれほど意識しないが、プログラムコードのほとんどは文字コードが密接に関係している。そのため、標準文字コードが変わると、アプリケーションの動作にも影響する可能性がある。

しかし前述の対処例により、文字コードの扱いや互換関数を把握していれば、バージョンアップ対応は定型的なソース変更で実施できる。

Delphi 2007 以前のプログラムから Delphi 2009 以降へバージョンアップをする場合には、本稿の対処ポイントを役立てていただければ幸いである。

M

図3

検証用設計画面1

ソース1

ボタン押下時の処理

```
procedure TSampleForm.Button1Click(Sender: TObject);
var
  sStr: String;
begin
  //Editに入力された1～4文字(ﾊﾞｲﾄ)目を取得
  sStr := Copy(Edit1.Text, 1, 4);

  //取得した文字列を表示
  ShowMessage(sStr);

end;
```

図4

実行結果（Delphi2007）

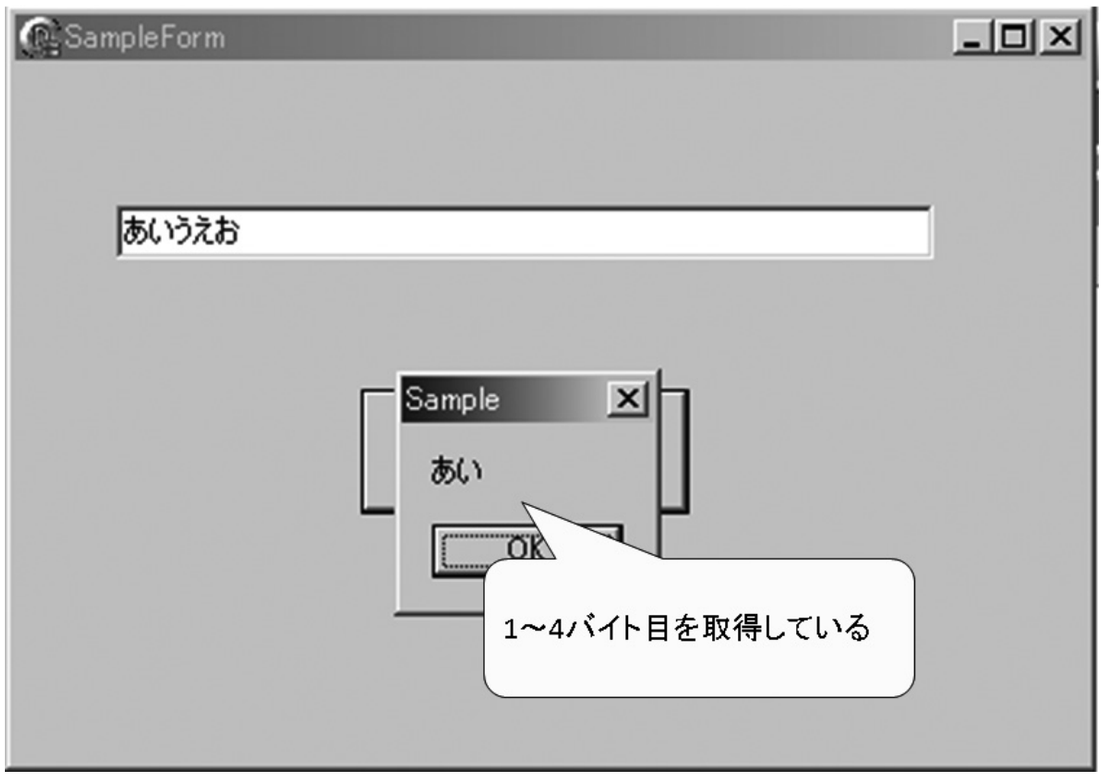


図5

実行結果（ Delphi10Seattle ）

ソース2

ボタン押下時の処理(AnsiString修正後)

```
procedure TSampleForm.Button1Click(Sender: TObject);
var
  sStr: String;
begin
  //Editに入力された1～4文字(ﾊﾞｲﾄ)目を取得
  sStr := Copy(AnsiString(Edit1.Text), 1, 4);

  //取得した文字列を表示
  ShowMessage(sStr);

end;
```

Editの文字列をAnsiString型で
キャストする

図6

AnsiString型修正後実行結果(Delphi10Seattle)

図7

図8

図9

実行結果（Delphi2007）

図10

実行結果（Delphi10Seattle）