Кодировки и Unicode в FPC

Иван Шихалев
http://freepascal.ru/

26 ноября 2005 г.

Одним из насущнейших вопросов для всякого русскоязычного программиста является вопрос кодировок. Так уж исторически сложилось, что компьютеры пришли к нам из англопишуших стран, чей алфавит насчитывает всего лишь 26 букв, а мы, спасибо Кириллу с Мефодием, пользуемся несколько другими. Впрочем, Кирилла и Мефодия винить не за что — они это не со зла, да и родной им греческий алфавит тоже далек от латиницы.

Как бы то ни было, мы в этом вопросе не одиноки. Более того, народам с иероглифической письменностью — еще хуже. Всемирное братание, когда все будут говорить на одном языке и писать на одном алфавите пока не предвидится, а посему возник некоторый стандарт кодирования символов, неэкономный для отдельно взятого языка, зато позволяющий работать сразу со многими.

Содержание

Некоторые общие сведения

Было бы заманчиво назвать Unicode "универсальной кодировкой", но, к сожалению, такой единой кодировки не существует. Стандарт определяет порядковый номер каждого символа, однако их очень уж много. Отводить под каждый символ 4 байта и более как-то никому не хочется. Поэтому используется несколько способов кодирования: в одних случаях используется ограниченное подмножество символов, кодируемое одним или двумя байтами; в других — количество байт на символ зависит от самого символа.

Среди ограниченных вариантов главное место занимают однобайтные, которые, впрочем, к Unicode никакого отношения не имеют... Для кириллицы таких кодировок, к сожалению, много: KOI8-r, используемая в *nix-системах и ставшая стандартом de facto для электронной почты; кодировка Windows 1251, используемая в MS Windows, cp866 — альтернативная кодировка DOS... И это только наиболее распространенные. Что особенно «приятно», так это то, что кодировка, утвержденная Международной Организацией по Стандартизации — ISO, в силу ряда причин неудобна и практически никем не используется. Среди двухбайтных следует выделить UCS, которая использовалась в операционных системах Windows до версии 2000/XP.

Кодировки с переменным числом байт на символ тоже бывают разные — различается «минимальный символ». Для нас наиболее интересными будут кодировки UTF-8, где минимум — один байт, и UTF-16, где, соответственно, два. UTF-16 используется в Windows 2000/XP (и, скорее всего, будет использоваться в последующих версиях), а UTF-8 широко распространена на *nix'овых платформах, но любим мы ее не только за это...

Дело в том, что FPC, как и большинство компиляторов и подобных им программ, воспринимает тексты только в однобайтной кодировке, причем в качестве значимых используются только символы из первой половины кодовой таблицы. В UTF-8 эти символы кодируются точно также. Иными словами, исходный код в UTF-8 компилятор прекрасно воспримет, какими бы ни были символы в строках и комментариях.

Что касается UTF-16, следует заметить, что все буквы европейских языков и знаки препинания кодируются одним двухбайтным символом. Таким образом, для большинства наших задач работа с ней и с UCS-кодировкой отличаться не будут.

Чтобы не останавливаться подробно на вопросе Unicode в целом, рекомендую статью «Unicode» в русской «Википедии» — там довольно ясно и подробно изложены основные сведения.

Строки в Free Pascal

Типы

В Free Pascal могут использоваться два типа символов, из которых составляются строки: одно- и двух-байтные. Для первых есть тип `Char` (он же — `AnsiChar`), для вторых — `WideChar`. И соответственно: `string`, `AnsiString`, `WideString`, `PChar` и `PWideChar`. Таким образом, мы можем использовать одно- и двух-байтные кодировки, причем как ограниченные, так и с переменным объемом символа, поскольку нулевой символ, имеющий специальное значение, везде одинаков.

Преобразование

Free Pascal поддерживает автоматическое преобразование Ansi и Wide-строк друг в друга. Тем не менее, кто пытался этим воспользоваться, испытал, наверное, жестокое разочарование — преобразуются только символы из первой части таблицы, ради которых переходить к двухбайтным кодировкам было б странно. Но не надо думать, что эта часть разработчиками FPC недоделана. Дело в том, что кодировки у всех разные, и задать такое преобразование «для всех» невозможно. В принципе, можно было б привязаться к текущим настройкам локализации, однако это добавляет платформо-зависимый код, и резко ухудшает переносимость программ. Вместо этого, определение перекодировки отдано на откуп программисту.

Управление перекодировкой зависит от структуры следующего типа:

```
type
  TWideStringManager = record
    Wide2AnsiMoveProc         : procedure (
                                          Source : PWideChar;
                                    var Dest   : AnsiString;
                                          Len    : SizeInt
                                         );
    Ansi2WideMoveProc         : procedure (
                                          Source : PChar;
                                    var Dest   : WideString;
                                          Len    : SizeInt
                                         );
    UpperWideStringProc       : function (const S : WideString) : WideString;
    LowerWideStringProc       : function (const S : WideString) : WideString;
    CompareWideStringProc     : function (const S1, S2 : WideString) : PtrInt;
    CompareTextWideStringProc : function (const S1, S2 : WideString) : PtrInt;
    CharLengthPCharProc       : function (const Str : PChar) : PtrInt;
    UpperAnsiStringProc       : function (const S : AnsiString) : AnsiString;
    LowerAnsiStringProc       : function (const S : AnsiString) : AnsiString;
    CompareStrAnsiStringProc  : function (const S1, S2 : AnsiString) : PtrInt;
    CompareTextAnsiStringProc : function (const S1, S2 : AnsiString) : PtrInt;
    StrCompAnsiStringProc     : function (S1, S2 : PChar) : PtrInt;
    StrICompAnsiStringProc    : function (S1, S2 : PChar) : PtrInt;
```

```
    StrLCompAnsiStringProc    : function (
                                          S1, S2 : PChar;
                                          MaxLen : PtrUInt
                                               ) : PtrInt;
    StrLICompAnsiStringProc   : function (
                                          S1, S2 : PChar;
                                          MaxLen : PtrUInt
                                               ) : PtrInt;
    StrLowerAnsiStringProc    : function (Str : PChar) : PChar;
    StrUpperAnsiStringProc    : function (Str : PChar) : PChar;
  end;
```

Впрочем, как видим, эта структура управляет не только преобразованием Ansi <-> Wide, но и другими действиями над строками, зависящими от кодировки — преобразованием регистра, порядком сортировки и т.д. Для управления используются следующие три процедуры:

```
procedure GetWideStringManager (var Manager : TWideStringManager);
procedure SetWideStringManager (const New : TWideStringManager);
procedure SetWideStringManager (
    const New : TWideStringManager;
      var Old : TWideStringManager
          );
```

Замечу, однако, что не все строковые функции модулей `System` и `SysUtils` используют данную структуру. Кроме того, для полноценной работы с UTF-8, например, ее недостаточно. В целом, работа над RTL в этом направлении еще не закончена.

UTF-8

Теперь посмотрим, что нам может предложить модуль `System` для работы с UTF-8.

```
type
  UTF8String = type AnsiString;

function UnicodeToUtf8 (
         Dest     : PChar;
         Source   : PWideChar;
         MaxBytes : SizeInt
                ) : SizeInt;
function UnicodeToUtf8 (
         Dest         : PChar;
         MaxDestBytes : SizeUInt;
         Source       : PWideChar;
         SourceChars  : SizeUInt
                    ) : SizeUInt;
```

```
function Utf8ToUnicode (
         Dest     : PWideChar;
         Source   : PChar;
         MaxChars : SizeInt
                ) : SizeInt;
function Utf8ToUnicode (
         Dest         : PWideChar;
         MaxDestChars : SizeUInt;
         Source       : PChar;
         SourceBytes  : SizeUInt
                    ) : SizeUInt;

function UTF8Encode (const S : WideString) : UTF8String;
function UTF8Decode (const S : UTF8String) : WideString;

function AnsiToUtf8 (const S : AnsiString) : UTF8String;
function Utf8ToAnsi (const S : UTF8String) : AnsiString;
```

Это то, что в разделе "`interface`". Если заглянуть в "`implementation`", то увидим, что последние две функции реализованы через умолчательное преобразование Ansi <-> Wide. Таким образом для их использования требуется адекватно определить это преобразование. Как — см. выше. В то же время преобразование UTF-8 <-> Wide определено практически полностью — включая 3-байтные символы (кириллица находится в 2-байтных).

Предварительные выводы

В целом на сегодняшний день работа с кодировками и Unicode в FPC вполне возможна. Далее мы еще поговорим, как проще всего добавить поддержку кириллицы. Заметим, что все вопросы, возникающие при локализации, решаются на уровне RTL, более того, их можно решить и не меняя стандартные модули — переопределение операторов и перегрузка функций могут производиться и в отдельных модулях (тут, правда, есть тонкости с текстовыми файлами). Управление строками на низком уровне, требующее поддежки компилятора, полностью реализовано, за исключением, разве что, совсем экзотических UTF-32 и UCS4 кодировок.

Шлифовка RTL, на мой взгляд, требуется в направлении большей стройности кода и ориентации на UTF-16 вместо UCS2 для `WideString` (то есть на использование переменной двухбайтной кодировки вместо ограниченной) — вдруг придется писать что-то для юго-восточной Азии... С другой стороны, строки с переменным числом байт на символ не позволяют уже обращаться с ними как с массивами символов и вообще значительно усложняют работу...

Дополнительные возможности

В стандартном наборе FPC наблюдается еще кое-что, что может облегчить работу с различными кодировками и Unicode. Это модуль `CharSet` и утилита `creumap`.

Модуль `CharSet` предлагает некое стандартное представление таблиц перекодировки. Вообще-то, он выглядит недоделанным и сам по себе малополезен, но в сочетании с программой `creumap` позволяет написать перекодирующие функции гораздо быстрее, чем с нуля.

Утилита формирует модули с таблицами перекодировки для `CharSet` из текстовых файлов специального вида. Что было бы тоже не особо полезно, если бы набор таких файлов не был включен в состав исходников FPC. Кстати, саму утилиту тоже придется компилировать из исходников — в бинарном дистрибутиве ее нет.

Поскольку без исходников все равно не обойтись, я не буду описывать модуль `CharSet` — его интерфейс умещается на одном экране — разобраться просто. Вместо этого замечу, что исходник `CharSet` находится в файле "`./rtl/inc/charset.pp`", `creumap` — "`./utils/creumap.pp`", а файлы кодировок — в каталоге "`./rtl/ucmaps/`", считая от корня исходников FPC. В частности, там наличествуют файлы для кодировок кириллицы: ISO-8859-5, cp855 (DOS-основная, практически не используемая), cp866 (DOS-альтернативная), cp1251 (Windows). KOI8-r, к сожалению, нет.

Таблицы перекодировки модуля `CharSet` устроены так, что нужный символ Unicode по ANSI выбирается сразу — как элемент таблицы по индексу, тогда как для обратного преобразования требуется перебор элементов, что не есть хорошо в случае перекодировки больших объемов текста.

Итоги

В целом картина типичная для Free Pascal: компилятор свое дело делает, RTL базовую функциональность реализует, хотя напильник бы не помешал, а для практического применения нужно немного поработать ручками и ясно представлять, что и как происходит. Надеюсь данная статья хотя бы немного поспособствует последнему — в документации, к сожалению, почти ничего на эту тему нет.