УДК 004.457

ОСОБЕННОСТИ ОПРЕДЕЛЕНИЯ КОДИРОВКИ СИМВОЛОВ В C#

К.С. Захарченко (3 курс, каф. КИ), С.В. Теплинский, к.т.н., доц.
Л.И. Дорожко, к.т.н., доц,
Донецкий национальный технический университет, г. Донецк
кафедра компьютерной инженерии

Аннотация

Захарченко К.С., Теплинский С.В., Дорожко Л.И. Особенности определения кодировки символов в C#. *Выполнен анализ методики определения кодировки символов в C#. Определены основные особенности определения кодировки символов в C#.*

Ключевые слова: *Юникод, кодовая страница, кодировка символов.*

Постановка проблемы. Иногда бывает так, что текст, состоящий из букв русского алфавита, полученный с другого компьютера, невозможно прочитать. Это происходит оттого, что на компьютерах применяется разная кодировка символов русского языка. Таким образом, возникает необходимость определить, как с этой проблемой справляется C#.

Цель статьи – провести анализ методики определения кодировок символов строки, отличных от используемой по умолчанию, в C#. Показать механизм ее работы.

О кодировках UTF. Большинство кодировок работают с набором символов, которые являются лишь малой частью UCS(Universal Coded Character Set). Это становится проблемой для многоязычных данных, вот по этому необходима кодировка, которая использует все символы UCS. Кодировка 8-битных символов очень простая, так как вы получаете один символ из одного байта, но UCS использует 31 бит и вам необходимо 4 байта на символ. Появляется проблема порядка байтов так как некоторые системы используют порядок от старшего к младшему, другие - наоборот. Так же часть байтов всегда будут пустыми, это пустая трата памяти. Правильная кодировка должная использовать различное количество байтов для различных символов, но такая кодировка будет эффективная в одних случаях и не эффективна в других.

Решение этой головоломки – использовать несколько кодировок из которых вы можете выбрать подходящую. Они называются Unicode Transformation Formats, или UTF.

UTF-8 – самая распространенная кодировка в Интернете. Она использует для ASCII символов один байт, а для всех остальных символов UCS 2 или 4

байта. Это очень эффективно для языков использующих латинские буквы, так как они все входят в ASCII, достаточно эффективно для Греческого, Кириллицы, Японского, Армянского, Сирийского, Арабского и др., так как для них используется 2 байта на символ. Но это не эффективно для всех остальных языков из BMP(Basic Multilingual Plane), так как будет использоваться 3 байта на символ, а для всех остальных символов UCS, например Готическое письмо, будет использоваться 4 байта.

UTF-16 использует для всех символов из BMP одно 16-битное слово и два 16-битных слова для всех остальных символов. По этому, если вы не работает с одним из упомянутых выше языков, вам лучше использовать UTF-16. Так как UTF-16 использует 16-битные слова, мы получаем проблему порядка байтов. Она решена наличием трех вариантов: UTF-16BE для порядка байтов от старшего к младшему, UTF-16LE - от младшего к старшему, и просто UTF-16, который может быть UTF-16BE или UTF-16LE, при кодировании в начале используется маркер, который указывает порядок байтов. Этот маркер называется "byte order mark", или "BOM".

Так же существует UTF-32, который может быть в двух вариантах BE и LE как и UTF-16, и хранит символ Юникод как 32-битное целое число. Это не эффективно почти для всех символов, кроме тех, которые требуют 4 байта для хранения. Но при этом очень легко обрабатывать такие данные, так как у вас всегда 4 байта на символ.

Важно разделять кодированные данные от данных Юникод. Поэтому не думайте о UTF-8/16/32 данных как о Юникод. Таким образом, хотя кодировки UTF и определены в стандарте Unicode, в рамках модели понимания Юникода считаем что UTF - это не Юникод [1].

Определение кодировки символов строки в C#. В C# для кодировки символов по умолчанию используется кодировка Unicode (UTF-8), стоит проверить как он поведет себя с другими кодировками.

Итак, имеем файлы с разными кодировками (ANSI, UTF-8, Unicode и Unicode BE, в частности) со следующим текстом: «Hello! Привет!».

В языке C# для чтения текстового файла воспользуемся классом StreamReader. Перегрузим конструктор двумя параметрами: путь к файлу и значение «истина» для параметра автоматического определения кодировки по метке порядка следования байтов [2].

```
using (var sr = new StreamReader(of.FileName, true))
        {
textBox.Text = sr.ReadToEnd()+" "  sr.CurrentEncoding.EncodingName;
        }
```

Теперь поочередно передаем классу файлы в разных кодировках и следим за результатом. Файл с кодировками из семейства Unicode были прочитаны

корректно (рис. 1-3). Однако, из файл с кодировкой символов ANSI была получена лишь половина символов (рис.4).

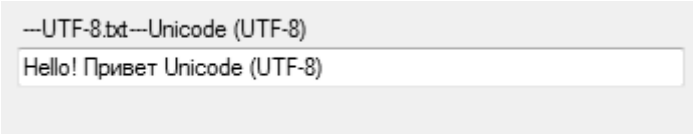

Рисунок 1 – Файл с кодировкой UTF-8

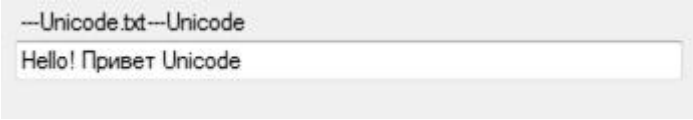

Рисунок 2 – Файл с кодировкой Юникод

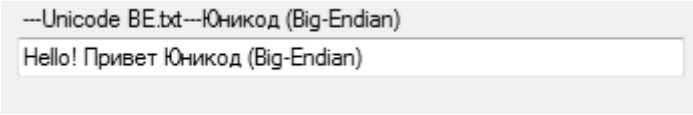

Рисунок 3 – Файл с кодировкой Юникод Big Endian

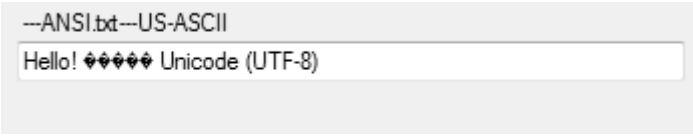

Рисунок 4 – Файл с кодировкой ANSI прочитан не верно

Теперь вызовем другой конструктор класса StreamReader перегрузим его, указав вторым параметром кодовую страницу 1251.

```
using (var sr = new StreamReader(of.FileName, Encoding.GetEncoding(1251)))
        {
    textBox.Text = sr.ReadToEnd() + " " + sr.CurrentEncoding.EncodingName;
      }
```

В результате получаем полный текст файла с кодировкой ANSI (рис.5). На правильность чтения файлов с кодировками Юникод изменение кодовой страницы не повлияло.

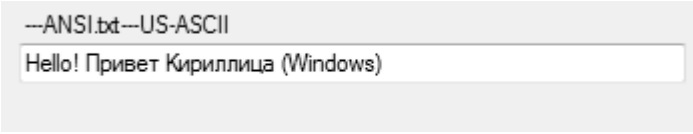

Рисунок 5 – Файл с кодировкой ANSI прочитан верно

Аналогичный эксперимент был проведен с файлом в кодировке Mac. Используя конструктор класса StreamReader без указания кодировки – файл читается не верно (рис.6), при использовании же метода с указанием кодовой страницы Mac (10007) – результат полный и правильный (рис.7).

Рисунок 6 – Файл с кодировкой Mac прочитан не верно

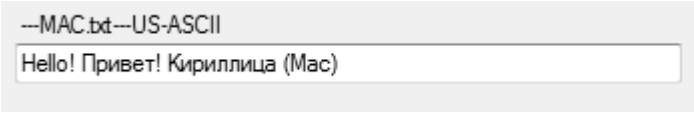

Рисунок 7 – Файл с кодировкой Mac прочитан верно

Как показали проведенные эксперименты – надежного способа определения произвольной кодовой страницы не существует, хотя и проводились попытки создать его, основываясь на вероятности обнаружения определенных последовательностей байтов в тексте. В StreamReader это не реализовано [3,4].

Вывод. На основе изложенной выше информации была разработана программа, которая определяет кодировку текстового файла, отображает кодировку на основании BOM(маркер порядка следования байт) выбранного текстового файла и его текст.

Результаты проведенных экспериментов показали, что C# успешно справляется с определением кодировки текстовых файлов, однако все же требует предварительной подстройки.

Список литературы

1. Unconfusing Unicode: What is Unicode? [электронный ресурс] // Lennart Regebro: Python, Plone, Web [сайт]. [2015]. URL: https://regebro.wordpress.com/2011/03/23/unconfusing-unicode-what-is-unicode/
2. Юникод [электронный ресурс] // Википедия – свободная энциклопедия: [сайт]. [2017]. URL: www.wikipedia.org/wiki/ Юникод
3. Microsoft Developer Network. [сайт]. URL:https://msdn.microsoft.com/
4. Гордеев А.В, Молчанов А.Ю. Системное программное обеспечение. – СПб.: Питер, 2002. – 736 с.: ил.