OGR shapefile encoding

Roger Bivand

December 12, 2022

1 Introduction

Changes have taken place in the way that OGR, the part of GDAL that handles vector data, treats character strings, both the contents of string attribute fields, and the names of fields. A discussion document covers the principles recognised in work on encoding strings.[1] The document states that "it is proposed to implement the CPLRecode() method using the iconv() and related functions when available." These mechanisms are similar to, but not identical with, those used in R itself.

The implementation[2] distinguishes between the use of iconv mechanisms in OGR, when GDAL is built with iconv support, and a fallback setting when iconv is not available. In both settings, UTF-8 encoding is intended to be use internally. Conversion from the layer in the data source to what we see inside R will differ depending on whether iconv support is available in GDAL, and on the encoding used in the R session.

This has had particular impact on the ESRI Shapefile driver, because this OGR format uses DBF files for storing attribute data. Impacts on other drivers are not known at present. A subsection has been created in the R wiki[3] to gather user experiences.

Recoding support for the ESRI Shapefile driver was introduced in GDAL/OGR 1.9.0. Two mechanisms are described in the driver documentation.[4]

We can read thet "the SHAPE_ENCODING configuration option may be used to override the encoding interpretation of the shapefile with any encoding supported by CPLRecode or to "" to avoid any recoding." Reference is made here to option values set in a number of ways, possibly by an environment (shell) variable, but maintained within the running instance of GDAL.[5] It emerged during discussion of a ticket on this issue,[6] that the use of shell variables may not be portable, so functions in **rgdal** set and get the configure options through compiled code, in particular the SHAPE_ENCODING configuration option.

The second mechanism does not seem very robust, and is based on storing and retrieving values set in the LDID byte of the DBF file header. The driver documentation says "an attempt is made to read the LDID/codepage setting from the .dbf file and use it to translate string fields to UTF-8 on read, and back when writing. LDID "87 / 0x57" is treated as ISO8859_1 which may not be appropriate." In the driver code, a listing

[1] `http://trac.osgeo.org/gdal/wiki/rfc23_ogr_unicode`.

[2] `http://trac.osgeo.org/gdal/browser/trunk/gdal/port/cpl_recode.cpp`.

[3] `http://rwiki.sciviews.org/doku.php#ogr_string_encodings`.

[4] `http://www.gdal.org/ogr/drv_shapefile.html`.

[5] `http://trac.osgeo.org/gdal/browser/trunk/gdal/port/cpl_conv.cpp`, functions CPLSetConfigOption() and CPLGetConfigOption().

[6] `http://trac.osgeo.org/gdal/ticket/4920`.

is provided[7], referring to a Russian website[8] as authority. Crucially, the most common case for ESRI Shapefiles written by ArcGIS under Windows is LDID 87, which this function treats as ISO8859_1. In addition, users of this format may provide an extra file with the extension CPG, which may or may not be respected.

This mechanism is related to the rather incomplete description provided by ESRI in their knowledge base,[9] but which may contain hints that are helpful in exchanging shapefiles between R/**rgdal** via the OGR ESRI Shapefile driver. It is made plain that it is the user's responsibility to divine the appropriate encoding.

For obvious reasons, it is very difficult to give a sensible representation in this vignette of the character strings involved, because the vignette itself contains encoded strings. I have encapsulated results on two platforms available to me, and have consulted with a user of OSX. Some results are given using `charToRaw` to provide a neutral framework permitting encodings to be compared across platforms. The key string used in this example may be found at this location: `http://goo.gl/maps/PNRgX` (Střítež nad Ludinou), using a shapefile provided by Lukáš Marek as a result of work provoked by Jeff Ranara reporting contorted Swedish strings during a course in Bergen.

2 Does GDAL have iconv support?

When the **rgdal** is loaded, a set of startup messages is displayed (unless supressed). They may include the line:

```
GDAL does not use iconv for recoding strings.
```

which is generated if:

```
> library(rgdal)
```

```
> .Call("RGDAL_CPL_RECODE_ICONV", PACKAGE="rgdal")
```

is `FALSE`. This test is not absolutely trustworthy when GDAL is dynamically linked to **rgdal**, because it reports the state of the GDAL configure variables set when GDAL and **rgdal** were built. It is trustworthy for the CRAN Windows and OSX binaries, because they are built with static linking to GDAL and its dependencies. In other cases, if the GDAL runtime binaries have been updated but the header files have not been, or **rgdal** has not been re-installed, the value returned may be misleading.

Depending on the outcome of this test, and concentrating here on the ESRI Shapefile driver, we can fork between two cases.

2.1 LDID and codepage values

First we will see how the LDID value may be retrieved. The `ogrInfo` reports the LDID for the ESRI Shapefile driver:

```
> dsn <- system.file("etc", package="rgdal")
> layer <- "point"
> oI <- ogrInfo(dsn, layer)
> attr(oI, "LDID")
```

[7] `http://trac.osgeo.org/gdal/browser/trunk/gdal/ogr/ogrsf_frmts/shape/ogrshapelayer.cpp`, in function ConvertCodePage(), after line 180.

[8] `http://www.autopark.ru/ASBProgrammerGuide/DBFSTRUC.HTM`, see Table 9.

[9] `http://support.esri.com/en/knowledgebase/techarticles/detail/21106`.

```
[1] 87
```

In this case, Lukáš Marek also provided a CPG file, as this is said to be common practice to inform ArcGIS of the appropriate codepage for the DBF file:

```
> scan(paste(dsn, .Platform$file.sep, layer, ".cpg", sep=""),
+  "character")

[1] "852"
```

This value does not match the LDID (which suits CP1252, the so-called ANSI codepage), but does match the Czech OEM codepage, for which the LDID should possibly be 31 (or 200). Most LDID values observed are 0 or 87. It is also possible that the value in the CPG file should be CP1250, rather than that used here.

To set the LDID value in writing a vector `Spatial*DataFrame` using the ESRI Shapefile driver, one may use the `layer_options=` argument to `writeOGR`:

```
> writeOGR(mySDF, dsn, layer, driver="ESRI Shapefile",
+  layer_options='ENCODING="LDID/31"')
```

but note that the OGR format documentation says: "The default value is "LDID/87". It is not clear what other values may be appropriate."

2.2 String representation in R

The encoding in use in the R session may be seen by:

```
> Sys.getlocale("LC_CTYPE")

[1] "en_GB.UTF-8"

> unlist(l10n_info())

   MBCS   UTF-8 Latin-1 codeset
 "TRUE"  "TRUE" "FALSE" "UTF-8"
```

where `l10n_info` may also return a `codepage` component (on Windows). Even when R is not running in a multibyte locale, as in the Windows GUI in some cases, it may be able to display marked multibyte charsets. OGR does not seem to mark multibyte charsets. R manuals document the ways in which internationalization, locales, and encoding are handled in general,[10] and for data import and export.[11] In addition, the OSX[12] and Windows[13] FAQ describe platform-specific questions.

[10] http://cran.r-project.org/doc/manuals/r-release/R-admin.html#Internationalization.

[11] http://cran.r-project.org/doc/manuals/r-release/R-data.html#Encodings.

[12] http://cran.r-project.org/bin/macosx/RMacOSX-FAQ.html#Internationalization-of-the-R_002eapp.

[13] http://cran.r-project.org/bin/windows/base/rw-FAQ.html#Languages-and-Internationalization.

3 GDAL with iconv

Despite the uncertainty present in the LDID and codepage definitions, it is possible to assert control both when GDAL is built with iconv and without iconv. First the GDAL with iconv case. The data displayed here are pre-stored from this system:

```
> sessionInfo()$locale

[1] "LC_CTYPE=en_US.UTF-8;LC_NUMERIC=C;LC_TIME=en_US.UTF-8;LC_COLLATE=en_US.UTF
```

This system has **rgdal** installed from source, using GDAL also installed from source. Similar behaviour will be found on other similar systems, also including **rgdal** installed from source under OSX, built against the Kyngchaos GDAL framework. Here, we can check that iconv is available in GDAL:

```
> .Call("RGDAL_CPL_RECODE_ICONV", PACKAGE="rgdal")

[1] TRUE
```

We proceed by checking the CPL configure option:

```
> getCPLConfigOption("SHAPE_ENCODING")

NULL
```

It may be set using `setCPLConfigOption("SHAPE_ENCODING", value)`, where `value` is the appropriate encoding, or `NULL` to unset `SHAPE_ENCODING`. If `value` is the empty string, recoding is turned off in GDAL. Since we know that the encoding of the sample file is CP1250, we can import into R recoding to UTF-8 (the charset of the R session and the charset used internally by GDAL) in three ways (using `stringsAsFactors=FALSE` to make access to the string values easier):

```
> setCPLConfigOption("SHAPE_ENCODING", "CP1250")
> pt1 <- readOGR(dsn, layer, stringsAsFactors=FALSE)
> setCPLConfigOption("SHAPE_ENCODING", NULL)
> charToRaw(pt1$NAZEV[1])

 [1] 53 74 c5 99 c3 ad 74 65 c5 be 20 6e 61 64 20 4c 75 64 69 6e 6f 75

> pt2 <- readOGR(dsn, layer, stringsAsFactors=FALSE, encoding="CP1250")
> charToRaw(pt2$NAZEV[1])

 [1] 53 74 c5 99 c3 ad 74 65 c5 be 20 6e 61 64 20 4c 75 64 69 6e 6f 75

> setCPLConfigOption("SHAPE_ENCODING", "")
> pt3 <- readOGR(dsn, layer, stringsAsFactors=FALSE)
> setCPLConfigOption("SHAPE_ENCODING", NULL)
> pt1i_1 <- iconv(pt3$NAZEV[1], from="CP1250", to="UTF-8")
> charToRaw(pt1i_1)

 [1] 53 74 c5 99 c3 ad 74 65 c5 be 20 6e 61 64 20 4c 75 64 69 6e 6f 75
```

The three methods are either to set the CPL configure option directly import the data, and then unset it; to use the `encoding=` argument to `readOGR` which sets and unsets the CPL configure option internally; or to turn off GDAL recoding, and use `iconv` inside R. The final approach would need extra care if the field names also need recoding, and the string data are stored in the R object in their original charset. This, however, may be desirable if the same object is to be exported back to, for example, ArcGIS. A fourth method is to turn off GDAL recoding and use iconv within R inside `readOGR` to recode from the given encoding to the session charset, here UTF-8:

```
> setCPLConfigOption("SHAPE_ENCODING", "")
> pt4 <- readOGR(dsn, layer, stringsAsFactors=FALSE, use_iconv=TRUE,
+  encoding="CP1250")
> setCPLConfigOption("SHAPE_ENCODING", NULL)
> charToRaw(pt4$NAZEV[1])

 [1] 53 74 c5 99 c3 ad 74 65 c5 be 20 6e 61 64 20 4c 75 64 69 6e 6f 75
```

Note that `ogrInfo` also takes `encoding=` and `use_iconv=` arguments. If we do not set the encoding, GDAL recoding is from CP1252, not CP1250, and is wrongly rendered in UTF-8:

```
> pt5 <- readOGR(dsn, layer, stringsAsFactors=FALSE)
> charToRaw(pt5$NAZEV[1])

 [1] 53 74 c2 b0 c3 9d 74 65 c3 97 20 6e 61 64 20 4c 75 64 69 6e 6f 75

> all.equal(charToRaw(pt5$NAZEV[1]), charToRaw(pt4$NAZEV[1]))

[1] "5 element mismatches"
```

4 GDAL without iconv

Next we turn to the GDAL without iconv case. The data displayed here are pre-stored from this system:

```
> sessionInfo()$locale

[1] "LC_COLLATE=English_United States.1252;LC_CTYPE=English_United States.1252;

> unlist(l10n_info())

    MBCS    UTF-8  Latin-1 codepage
       0        0        1     1252
```

This system has **rgdal** installed using the CRAN binary. Here, we can check that iconv is available in GDAL:

```
> .Call("RGDAL_CPL_RECODE_ICONV", PACKAGE="rgdal")

[1] FALSE
```

This means that if GDAL recoding is not turned off, default non-iconv stub recoding will be used. It will often not be sensible to permit this to happen, so users may protect the raw data by either of these two methods:

```
> setCPLConfigOption("SHAPE_ENCODING", "")
> pt6 <- readOGR(dsn, layer, stringsAsFactors=FALSE)
> setCPLConfigOption("SHAPE_ENCODING", NULL)
> charToRaw(pt6$NAZEV[1])

 [1] 53 74 f8 ed 74 65 9e 20 6e 61 64 20 4c 75 64 69 6e 6f 75

> pt7 <- readOGR(dsn, layer, stringsAsFactors=FALSE, encoding="")
> charToRaw(pt7$NAZEV[1])

 [1] 53 74 f8 ed 74 65 9e 20 6e 61 64 20 4c 75 64 69 6e 6f 75
```

The assumption here is that there may not be a suitable recoding matching the imported strings and the R console; users will need to experiment to see how `iconv` may be used within R.