ReadMe: ‘DataGov_API_GeoMD_Tool_v1.R’ Script

Code

Andrew Beggs (General Engineer), Department of Commerce - Office of Inspector General

Purpose, Source, Scope, Conclusion

Purpose:

This software script intends to assist the user(s) quickly and easily:

1. read in data across about a number of geospatial metadata records via the Data.gov Application Programming Interface (API)
2. parse (i.e., process) the information sent from the Data.gov API
3. perform the selected series of reliability and quality tests on the metadata present, including:
   3.1 determine if data exists (reliability)
   3.2 determine if data allowed/disallowed values and/or stored in proper format(s) (quality)
   3.3 determine and test other attributes (e.g., data reliability, date validation, keyword formatting) (quality)
4. aggregate results across all data records read in via the Data.gov API (*see step (1)*)
5. automatically write results for further analysis and review

This notebook documents the key parts of the script’s code that are most impactful to meeting these objectives. All code will be reproduced, in full, at the end of this file.

Source:

This resource was developed in response to the administrative and technical challenges associated with both retrieving and then reviewing any reasonable number of geospatial metadata files, consistent with effective Office of Inspector General oversight onto an agency’s geospatial data resources.

Without a resource like this, auditors can still locate, download, and open an individual records’ geospatial metadata (e.g., in a web browser). However, in order to determine the availability and relative quality of information present in the metadata file, auditors would need to locate each valid data location within the file (a.k.a. searching for the correct “XPath”), copy or view that information, and then overlay best practices to determine suitability of the information supplied.

Whereas, computers are well-suited to this exact tedium of locating, downloading / obtaining, reviewing large amounts of data by using a robust-yet-adaptable series of logical checks, aggregating results, and reporting results.

Specifically, this script sources its data from the Data.gov API (https://catalog.data.gov/) (script variable: `datagov_url`). The Data.gov API returns information when we pass it a request (using the `package_search()` function). The script iteratively asks for, receives, and then digests information on geospatial metadata from the Data.gov API.

Scope:

The availability, completeness, and quality of (AGENCY NAME HERE) (#123,456#) geospatial metadata hosted on Data.gov.

Conclusion:

In short, this script establishes the basic logical pattern to reliably and accurately test an agency’s geospatial metadata records’ content fields for (1) completeness and (2) quality of information against metadata standards and/or best practices, where applicable. This script also assist with automating the aggregation of information about the group of geospatial metadata tested, enabling subsequent analysis (or discussion) on potential data quality issues like: invalid dates, future dates, disallowed metadata values, improperly formatted keywords, duplicated keywords, and other such issues.

R Package Dependencies

tidyverse
stringr
tibble
xml2
curl
crul
R6
urltools
httpcode
jsonlite
ckanr
purrr
httr
glue

Package loading function (ipak)

This R script leveraged a script (Method 3) (https://towardsdatascience.com/fastest-way-to-install-load-libraries-in-r-f6fd56e3e4c4) written by Catherine Williams. This function seeks to quickly install and load the required libraries inside the R script before calling any package-dependent functions. This function attempts to semi-automate package install/loading, (visually) shrink package loading code, and make the script more interperable by others.

Hide

```
ipak <- function(pkg){
  new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
  if (length(new.pkg))
    install.packages(new.pkg, dependencies = TRUE)
  sapply(pkg, require, character.only = TRUE)
}

listofpackages <- c("stringr","tibble","xml2","tidyverse","curl","crul","R6",
                    "urltools","httpcode","jsonlite","ckanr","purrr","httr",
                    "glue","lintr","styler","microbenchmark")

ipak(listofpackages)
```

Analysis Background and Overview

R is a multipurpose programming language typically used in statistics and data visualization. This is due, in part, because it is flexible when handling data (e.g., mixed data types), is open source (free code base), and has a large user community that developed further packages and functions (e.g., the `ckanr` function used here to pass requests to Data.gov's API).

API selection and setup:

As a new user to this script, you will need to obtain obtain an API key (https://api.data.gov/signup/). Once you have a key, please place it into `api_key` variable name (inside the placeholder quotes in the script file itself). Note that this key is unique to each requester/user and should be treated as confidential information. As always, please consult with your agency's CIO / CISO on any additional protections —if any—that may be required by policy to protect such information.

Once R installs `ckanr` (along with all its dependencies), you should be able to begin accessing geospatial metadata by running this script from the top (which then configures your access to the API with the provided key and begins to request information from Data.gov's API). If the script fails to access the API, again please coordinate with your CIO/CISO on what further steps may be needed to ensure access to the Data.gov API via R.

Sampling vs population:

This script will attempt to survey the population (i.e., all) of a given agency's geospatial metadata. See "varname" for information on how to set up an agency and its sub-agencies (or bureaus).
Note: The Data.gov API is hard-limited to providing responses back of a maximum of 1,000 rows. For agencies with larger volumes of geospatial metadata (e.g., Department of Commerce) this could mean the script will run for hours in order to complete the population census.

API information vs. metadata:

This script defaults to using the information the Data.gov API supplies about each record which, as far as we can determine, is consistent with the information actually contained in the underlying record's metadata file(s).

This means that it is not necessary to download each geospatial metadata file, though the script is capable of automatically handling that task as well (if such a step is desired by the audit team). See "downloading metadat files" section below for more details. At a minimum, this script does record the URL location of each metadata file: see `api_MDfiles$MD_Link` for the Data.gov download request URL (such that a download could be performed later).

Downloading metadata files:

There is a section of the code that looks like this:

Hide

```
# Uncomment this code and the "else" conditional below to download MD
        # files at the same time as the analysis.
        # Warning: This dramatically slows down script performance
        # (best run this overnight / as a background process)

        # mdfiledownload <- tryCatch(download.file(url = paste0(ds_harvestlink), destfile = paste0(mdff,sprintf("%0
06d", x), "_", sprintf("%003d", u), "_",sprintf("%003d", b), "_", sprintf("%003d", k), "_", sprintf("%004d", j),
"_", dsname, ".xml")), error=function(ef) ef)
        # dllim <- 0
        # if (inherits(mdfiledownload,"error")){
        #   while (inherits(mdfiledownload,"error") || dllim < 6){
        #     Sys.sleep(5)
        #     if (dllim < 5){
        #       mdfiledownload <- tryCatch(download.file(url = paste0(ds_harvestlink), destfile = paste0(mdff,sprin
tf("%006d", x), "_", sprintf("%003d", u), "_",sprintf("%003d", b), "_", sprintf("%003d", k), "_", sprintf("%004d",
j), "_", dsname, ".xml")), error=function(ef) ef)
        #     } else if (dllim ==5){
        #       dstempname <- str_to_lower(glue_collapse(str_remove_all(str_remove_all(ds_title,"\\s"),"[:punc
t:]"), sep = ""))
        #       mdfiledownload <- tryCatch(download.file(url = paste0(ds_harvestlink), destfile = paste0(mdff,sprin
tf("%006d", x), "_", sprintf("%003d", u), "_",sprintf("%003d", b), "_", sprintf("%003d", k), "_", sprintf("%004d",
j), "_", dstempname, ".xml")), error=function(ef) ef)
        #     }
        #     dllim <- dllim + 1
        #   }
        #   if (inherits(mdfiledownload,"error") && dllim >=6 ){
        #     api_MDfiles$MD_DownloadedFlag[x] <- "Fail"
        #     api_qcresults$`MD-DownloadURL-Exists`[x] <- TRUE
        #     api_qcresults$`MD-DownloadURL-Score`[x] <- 1
        #     api_MDfiles$MD_Link[x] <- ds_harvestlink
        #   }
        # } else {
```

If a user uncomments (CTRL + SHIFT + C) from the line beginning with "mdfiledownload," then the script will begin downloading metadata files to the directory specified in the `mdff` variable. The metadata files are typically quite small but caution is warranted if your agency has thousands of records because the storage requirements will climb exponentially.

Each file will be serialized to match the `x`, `u`, `b`, `k`, and `j` values (i.e., where the script recorded interacting with that record). Example filename structure (not a real file): 153256_05_014_05_0975_climate-and-weather.xml

Metadata Criteria:

As described in the GDA, the Federal Geospatial Data Committee (FGDC) is the federal entity responsible for setting geospatial data standards, including standards for metadata (https://www.fgdc.gov/metadata) for geospatial data. Notably, FGDC has yet to publish any geospatial data standards but does maintain a technical guidance document (pdf, last updated: December 13, 2022) (https://www.fgdc.gov/technical-guidance/metadata/fgdc-technical-guidance-datagov-geoplatform-ngda.pdf) describing both the expectations for and locations of relevant metadata. This guidance applies to both geospatial metadata hosted on Data.gov (https://catalog.data.gov/dataset/?metadata_type=geospatial) and the GeoPlatform (https://www.geoplatform.gov/).

The National Oceanic and Atmospheric Administration (NOAA) also maintains a workbook (pdf) (https://www.ncei.noaa.gov/sites/default/files/2020-04/ISO%2019115-2%20Workbook_Part%20II%20Extentions%20for%20imagery%20and%20Gridded%20Data.pdf), which guides a user in implementing the relevant features of ISO 19115-2:2009(E) (paid standard) (https://www.iso.org/standard/39229.html) edition: "*Geographic information—Metadata Part 2: Extensions for imagery and gridded data*". Note: ISO 19115-2:2009 was withdrawn and, subsequently, updated (around 2019) to the ISO 19115-2:2019 (paid standard) (https://www.iso.org/standard/67039.html) edition: "*Geographic information—Metadata Part 2: Extensions for acquisition and processing*."

Metadata content fields and score totals:

At present, this script will attempt to examine the following 20 metadata content fields:

- title
- abstract
- responsible party (role code)
- progress code
- maintenance frequency code
- citation identifier
- metadata identifier
- reference system identifier
- metadata download
- data set download
- web service(s)
- temporal extent begin
- temporal extent end
- data set publication date
- data set creation date
- data set revision or last update date
- metadata publish date
- metadata revision or last update date
- geographic extent (4x bounding boxes)
- keywords

Each of the 20 tested elements had a maximum point value associated with its corresponding score (as captured by the script in the `api_qcresults` variable). The script will run (at least) 30 different tests on the metadata information from the API. **NOTE: We did not use these scores to tabulate or otherwise rank results for the purposes of a finding; this is an informational view into a relative level of completeness (expected versus actual).**

Metadata element	NGDA Max score:	NGDA Min. Score	Non-NGDA Max score:	Non-NGDA Min. Score
title	2	1	1	1
abstract	1	0	1	0
responsible party (role code)	1	0	1	0
progress code	1	0	1	0

Metadata element	NGDA Max score:	NGDA Min. Score	Non-NGDA Max score:	Non-NGDA Min. Score
maintenance frequency code	1	0	1	0
citation identifier	1	0	1	0
metadata identifier	1	0	1	0
reference system identifier	1	0	1	0
metadata download	1	0	1	0
data set download	1	0	1	0
temporal extent begin	5	0	5	0
temporal extent end	5	0	5	0
data set publication date	5	0	5	0
data set creation date	5	0	5	0
data set revision or last update date	5	0	5	0
metadata publish date	5	1	5	1
metadata revision or last update date	5	1	5	1
geographic extent (4x bounding boxes)	10	0	10	0
keywords	1	0	1	0
keywords-duplicated tags	1	0	1	0
keywords-duplicated words	1	0	1	0
keywords-multiple words	1	0	1	0
keywords-punctuation	1	0	1	0
keywords-NGDA	2	0	1	0
keywords-NGDAID	2	0	1	0
keywords-NGDA long	2	0	1	0
keywords-NGDA theme	2	1	1	0
web service	1 point for each (current limit 9)	0	1 point for each (current limit 9)	0
Total possible	**79**	**3**	**71**	**3**

Discovery of data and basic validation approach for metadata fields:

Generally, we made every reasonable attempt to find the information contained within the 20 metadata elements listed above (if it were present). We built the code by using some test cases/samples, manually inspected the data package sent by the Data.gov API for those handful of samples, cross-checked that information with the samples' original metadata files, and then carefully constructed the API data extractor to consistently pull the data from that specific location. As the API should always be very consistent in delivering the exact same data package each time we request data, this provided a very consistent way to read-in the same data across all records requested.

Strong caveat: This script generally expects to find information in one or just a handful of locations. If data are present in another location, the script is not prepared to scan through each and every spot of a given record to attempt to identify and give further credit for information that factually exists but is not uncovered using this automated approach. Teams should

proceed with caution and discuss analysis findings with agency representatives to ensure this script's outputs match known gaps (or not) in geospatial metadata. This is an especially important step for those agencies with "fewer" metadata than agencies with thousands of records to ensure a fair and accurate representation of the facts surrounding the relative completeness and quality of an agency's geospatial metadata holdings.

Simple exists test for API-enforced value:

Here is a relatively simple code snippet that demonstrates how the script looks inside the API information and tests to see if a value exists (hint: API always puts it in put the metadata itself may not contain the value) and is allowed per FGDC guidance / best practices:

Hide

```
# ........................Title QC.............................
ds_title <- paste0(api_res$title[j])
ds_title <- str_replace_all(ds_title,"\\n$",blankify)
ds_title <- str_trim(str_replace_all(ds_title,"\\b\\s+\\b"," "))
ds_title <- str_replace_na(ds_title, "NA")
api_results$Title[x] <- ds_title
api_MDfiles$DS_Title[x] <- ds_title
if (str_length(ds_title) == 0) {
  # This condition should never trigger but it is here for thoroughness
  api_qcresults$Title[x] <- ds_title
  api_qcresults$`Title-Check`[x] <- FALSE
  api_qcresults$`Title-Score`[x] <- 0
} else {
  if (isngda) {
    # NGDA data set, looking to see that the Title on Data.gov
    # matches the title on FGDC
    # This is for "bonus points" that all sources agree, no score
    # penalty from non-NGDA for a mismatch with FGDC
    api_qcresults$Title[x] <- ds_title
    api_qcresults$`Title-Check`[x] <- TRUE
    ds_titletmp <- str_replace_all(str_trim(ds_title),"[[:punct:][:space:][:blank:]]","_")
    if (str_detect(ds_titletmp, ngdatitle)) {
      # assign +1 bonus point for a non-case sensitive match between
      # Data.gov to FGDC's website of approved NGDAs
      api_qcresults$`Title-Score`[x] <- 2
    } else {
      # Regular credit
      api_qcresults$`Title-Score`[x] <- 1
    }
  } else {
    # Non-NGDA DS should always have a title when pulling from the API
    api_qcresults$Title[x] <- ds_title
    api_qcresults$`Title-Check`[x] <- TRUE
    api_qcresults$`Title-Score`[x] <- 1
  }
}
```

Code snippet breakdown:
1. Obtain the data set title (`ds_title`) value from `api_res$title` at position j (line 1)
2. Cleanup on the string to remove new line characters (`"\\n$"`), multiple spaces with one space (`"\\b\\s+\\b"," "`), and any NA values with text (`"NA"`) (lines 2-4)
3. Write the title out (lines 5-6)
4. Conditional block, first check: is the string empty? If so, write blank and assign FALSE and 0 points (should never happen; lines 7-11)
5. Conditional block, else and next if: is this an NGDA record? If so, see if the found title matches FGDC's record for the title. If matched, assign bonus point, else keep one point. (lines 12-28)
6. Otherwise, write the title to the `api_qcresults` variable, write the check value, and assign the score (lines 29-34)

More complicated test for values non-enforced by the API:

Data set publication, creation, and revision/lastUpdate dates are not enforced by the API. In other words, the metadata files themselves supply the information—if it is populated inside the metadata file. This means that the API returns information in this location that can range from being blank to up to 3 different dates. This testing is a bit more complicated:

This intimidating-looking regular expression (`pubdate_extr` stands for publication date extractor, uses: `regx()` function from `stringr`) is just explicit pattern recognition logic to:

- Find a “date pattern” of a minimum of 4 digits in sequence: `\\d{4}` (*Note: the rest of the REGEX following this* `\\d{4}` *part is just more patterns of possible dates to match, ensuring the longest possible match, if it exists*)
- Look backward to see if we see the word “value” and then back further to see if we find a “publication” text tag
- If and only if the data matches yes to this pattern match will it succeed (i.e., match a YYYY date or longer within a string like: `"publication"{..some text or separators..}"value"{..some text or separators..}"YYYY"{a date in form YYYY or longer}"`).

So, in this snippet:

1. Line 1 is the `TRUE` or `FALSE` from the pattern recognition.
2. Line 2 tests the case where we do have a date
3. Line 3 extracts the publication date and coerces to char using `paste0()`
4. Line 4 replaces any `NA` values picked up (which will happen when using `str_extract_all()` if the pattern match fails) with the character text “NA”
5. Line 5 sets the `d8exists` (d8 = shorthand for “date”) variable as `TRUE` (used later to write results)
6. Line 6 (date validation) runs the custom `datevalidation()` function on the `D_PubDate` (data set publication date) value and writes those results out to the `dpubdateqc` (data set publication date quality control list-matrix) 7-8. Line 7-8 is a flag value for if the publication date is invalid (for coercion to a POSIXct date object). The validity and score values for analysis purposes are actually stored in the `dpubdateqc` list-matrix and written out separately (later).
7. Line 9 writes out the extracted data set publication date’s value to the results table.

Edge case handling:

While we generally we able to obtain data where it existed, sometimes data patterns were not consistent with expectations. We made every attempt to copy data that existed into the `api_results` variable before post-processing it for validity and recording results in the `api_qcresults` variable’s appropriate column(s).

Significant aspects of the script’s code

Code Structure & Loops

This code uses a triple for loop overall structure and an conditional fourth for loop if there are resources paired with the record currently under examination. Inside the second for loop, we use one while loop (with error handling and an alternative break criteria) to call the API as it will occasionally throw a 502-Bad Gateway error that stops the for loops from working.

The basic loops follow this pattern: (1) how much data and set API calling variables -> (2) call the API for as large a batch as required (between 1000 and 1 records) -> (3) parse the API’s information and perform various analyses, write-outs, checks, etc. (OPTIONAL)-> (4) if resources = yes and if we see at least 1 download link or 1 web mapping service, run a for loop to tabulate those results -> repeat (3) until API table parsing complete -> iterate one step forward, request new table (2) -> when all bureau data read, sequence to next bureau (until all bureaus completed) (1) -> record read-in complete. Here is a basic mock-up of the code’s structure:

REGEX (Regular Expressions)

This section defines various regular expressions (REGEX) used in detecting patterns within data (i.e., strings themselves and/or patterns within strings/character arrays).

Basic Date REGEX

This `date_regex` is written primarily to detect if something is a “date” form or not. The REGEX was designed to be intentionally broad such that it looks for a pattern of digits (i.e., the \\d means digits and the {#} sets the number of matches) within a string.

We assume, for the purposes of this script, that dates will be presented in a semi-standardized format. We make most reasonable attempts to identify a date by only searching in date fields and following standard conventions while making some allowances for variations or mixed formats.

While this is not an exhaustive list, we commonly anticipate dates in formats like ISO standard 8601:
YYYY-MM-DDTHH:MM:SS+00:00
YYYY-MM-DDTHH:MM:SS.123456Z
YYYY-MM-DD
YYYY-MM
YYYY

And some less common variants, like:

MM-YYYY
MM-DD-YYYY
DD-MM-YYYY
YYYY-MM-DDTHH:MM

Correspondingly, we wrote the `date_regex` regular expression to detect dates across a wide range of potential variability. This REGEX is written broadly so it is more interested in identifying the digit patterns than being caught up in any particular separator schema. For example, it should accept dates in these example (but not encountered) formats:
YYYY/MM;DD HH.MM\SS 00:00
YYYY,MM,DD HH;MM-SS/123456

Generally, it's been tested and confirmed to work on dates of pattern(s):
YYYY-MM-DDTHH:MM:SS+00:00
YYYY-MM-DDTHH:MM:SS+00:00Z
YYYY-MM-DDTHH:MM:SS + 00:00
YYYY-MM-DDTHH:MM:SS + 00:00Z
YYYY-MM-DDTHH:MM:SS.123456
YYYY-MM-DDTHH:MM:SS.123456Z
YYYY-MM-DDTHH:MM:SS123456
YYYY-MM-DDTHH:MM:SS123456Z
YYYY-MM-DDTHH:MM:SS
YYYY-MM-DDTHH:MM:SSZ
YYYY-MM-DDTHH:MM
YYYY-MM-DDTHH:MMZ
YYYY-MM-DDTHHMM
YYYY-MM-DDTHHMMZ
YYYY-MM-DD HH:MM:SS+00:00
YYYY-MM-DD HH:MM:SS+00:00Z
YYYY-MM-DD HH:MM:SS + 00:00
YYYY-MM-DD HH:MM:SS + 00:00Z
YYYY-MM-DD HH:MM:SS.123456
YYYY-MM-DD HH:MM:SS.123456Z
YYYY-MM-DD HH:MM:SS123456
YYYY-MM-DD HH:MM:SS123456Z
YYYY-MM-DD HH:MM:SS
YYYY-MM-DD HH:MM:SSZ
YYYY-MM-DD HH:MM
YYYY-MM-DD HH:MMZ
YYYY-MM-DD HHMM
YYYY-MM-DD HHMMZ
YYYY-MM-DD
YYYY-MM
YYYY
YYY
YY
Y
MM-YYYY

MM-DD-YYYY
MM-DD-YYYYTHHMM
MM-DD-YYYYTHHMMZ
MM-DD-YYYYTHH:MM
MM-DD-YYYYTHH:MMZ
MM-DD-YYYYTHH:MM:SS
MM-DD-YYYYTHH:MM:SSZ
MM-DD-YYYYTHH:MM:SS123456
MM-DD-YYYYTHH:MM:SS123456Z
MM-DD-YYYYTHH:MM:SS.123456
MM-DD-YYYYTHH:MM:SS.123456Z
MM-DD-YYYYTHH:MM:SS + 00:00
MM-DD-YYYYTHH:MM:SS + 00:00Z
MM-DD-YYYYTHH:MM:SS+00:00
MM-DD-YYYYTHH:MM:SS+00:00Z
MM-DD-YYYY HH:MM
MM-DD-YYYY HH:MMZ
MM-DD-YYYY HHMM
MM-DD-YYYY HHMMZ
MM-DD-YYYY HH:MM:SS
MM-DD-YYYY HH:MM:SSZ
MM-DD-YYYY HH:MM:SS123456
MM-DD-YYYY HH:MM:SS123456Z
MM-DD-YYYY HH:MM:SS.123456
MM-DD-YYYY HH:MM:SS.123456Z
MM-DD-YYYY HH:MM:SS + 00:00
MM-DD-YYYY HH:MM:SS + 00:00Z
MM-DD-YYYY HH:MM:SS+00:00
MM-DD-YYYY HH:MM:SS+00:00Z
DD-MM-YYYYTHH:MM:SS123456
DD-MM-YYYYTHH:MM:SS123456Z
DD-MM-YYYYTHH:MM:SS.123456
DD-MM-YYYYTHH:MM:SS.123456Z
DD-MM-YYYYTHH:MM:SS+00:00
DD-MM-YYYYTHH:MM:SS+00:00Z
DD-MM-YYYYTHH:MM:SS + 00:00
DD-MM-YYYYTHH:MM:SS + 00:00Z
DD-MM-YYYYTHH:MM:SS
DD-MM-YYYYTHH:MM:SSZ
DD-MM-YYYYTHH:MM
DD-MM-YYYYTHH:MMZ
DD-MM-YYYY HH:MM:SS123456
DD-MM-YYYY HH:MM:SS123456Z
DD-MM-YYYY HH:MM:SS.123456
DD-MM-YYYY HH:MM:SS.123456Z
DD-MM-YYYY HH:MM:SS+00:00
DD-MM-YYYY HH:MM:SS+00:00Z
DD-MM-YYYY HH:MM:SS + 00:00
DD-MM-YYYY HH:MM:SS + 00:00Z
DD-MM-YYYY HH:MM:SS
DD-MM-YYYY HH:MM:SSZ
DD-MM-YYYY HH:MM
DD-MM-YYYY HH:MMZ

Any and all combinations of the above separators should work because we used the “anything” character in between word boundaries (e.g., it looks for anything between the digits). More information on REGEX (https://stringr.tidyverse.org/articles/regular-expressions.html).

To be updated (2-28-2025), we have newer expressions: Lastly, the `dateymd` and `datemdy` REGEX are essentially the same thing as the `dateregex` variable except that they do not specifically use any separator class definitions. Instead, we rely on word boundary detection (`\\b`) to establish the digit pattern, essentially ignoring any separators that may/may not exist. The `dateymd` variable is used to detect dates of form: YYYY-MM-DD (or longer). The `datemdy` variable detects of form: MM-DD-YYYY (or longer).

Date REGEX required and used in datevalidation():

To be updated (2-28-2025); we are not using the datevalidation() function any more. These two headings cover the two main groups (pattern-format and valid date digits).

End of ReadMe