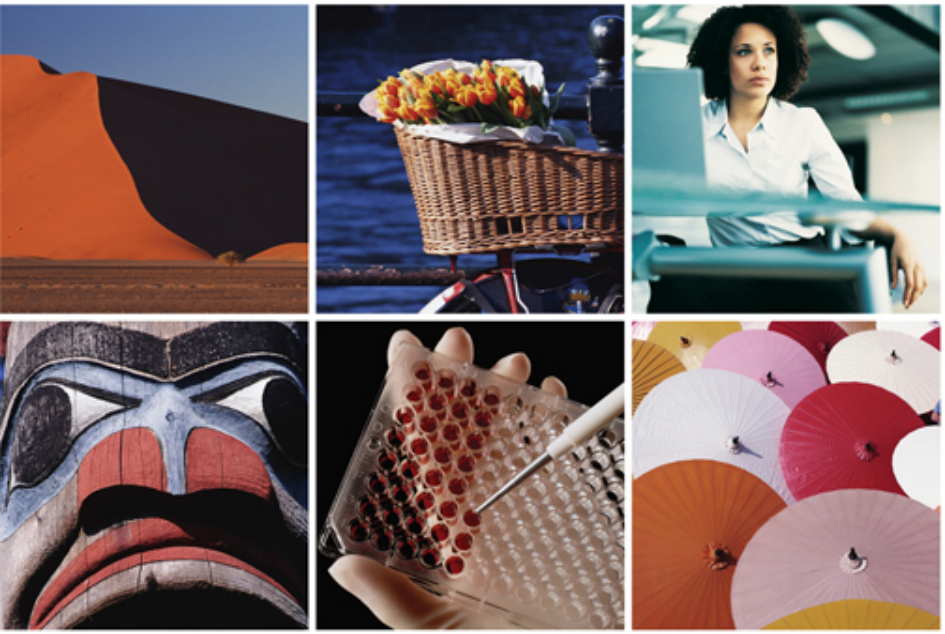
ICON
A Symbol of Excellence



CS-02 Parsing ISO 8601 Standard Partial Dates using Perl Regular Expressions

Mark Crangle

2012-10-17

Introduction: The Problem (1)

- Dates are received in SDTM datasets as a character variable in ISO 8601 format

  YYYY-MM-DD
- SDTM IG states:
  - *Information is human and machine readable*
  - *Completeness of the date can be seen from presence or absence of components*
- Commonly manipulation of these values is required
  - *Values are reformatted into more commonly used format for data presentation*

    eg. DDMMMYYYY
  - *May require conversion to numeric for calculation*

Introduction: The Problem (2)

- For complete dates, this is straightforward as SAS provides informats

```
num_date=input(in_date,is8601da.);
char_date=put(num_date,date9.);
```

in_date	num_date	char_date
2012-05-25	19138	25MAY2012
2012-05-25	19138	25MAY2012
2012-05-25	19138	25MAY2012
2012-05-28	19141	28MAY2012
2012-05-28	19141	28MAY2012
2012-05-25	19138	25MAY2012
2012-05-25	19138	25MAY2012
2012-05-25	19138	25MAY2012

- But how do we do this for partial dates
  - *First identify which parts are known/unknown*
  - *Separate the components and combine in the required format*
  - *If numeric values are required should missing information be imputed*

in_date
2012-05
2012
2012---2

Partial Dates in ISO 8601 Format (1)

- As SDTM IG says, partial dates are identified by an absence of components
- Missing components are represented by
  - *Right truncation*
    - *Missing components are from the right hand side of the string*

      Eg. 2009-09, or 2010.
  - *Replacement with a hyphen*
    - *More unlikely scenario*
    - *"Intermediate" components that are missing are replaced with a single hyphen*

      Eg. 2009---10, --09-15

    - *Hyphen replacement is used when missing components are from the right, but time part is known*

      eg. 2009-09--T12:00

Partial Dates in ISO 8601 Format (2)

#	Year Known	Month Known	Day Known	ISO 8601 Pattern	Example Date
1	Yes	Yes	Yes	YYYY-MM-DD	2005-09-12
2a	Yes	Yes	No	YYYY-MM	2005-09
2b				YYYY-MM--	2005-09--
3a	Yes	No	No	YYYY	2005
3b				YYYY----	2005----
4	Yes	No	Yes	YYYY---DD	2005---12
5	No	Yes	Yes	--MM-DD	--09-12
6	No	No	Yes	----DD	----12
7	No	Yes	No	--MM--	--09--
8a	No	No	No	(missing)	(missing)
8b				-----	-----

Handling Partial Dates (1)

ICON
A Symbol of Excellence

- Identifying “common” partial date types can be done using the length of the string
  - *When missing components represented by right truncation the length is sufficient to identify the partial date type.*
  - *If handling uncommon partial date types may need to examine substrings as lengths are not unique.*
- Extracting date components using scan or substring functions

Handling Partial Dates (2)

#	Year Known	Month Known	Day Known	ISO 8601 Pattern	Length
1	Yes	Yes	Yes	`YYYY-MM-DD`	10
2a	Yes	Yes	No	`YYYY-MM`	7
3a	Yes	No	No	`YYYY`	4
4	Yes	No	Yes	`YYYY---DD`	9
5	No	Yes	Yes	`--MM-DD`	7
6	No	No	Yes	`----DD`	6
7	No	Yes	No	`--MM--`	6
8a	No	No	No	`(missing)`	0

- This leads us to consider methods designed for matching patterns!

Perl Regular Expressions (1)

ICON
A Symbol of Excellence

- Perl Regular Expressions added to SAS in version 9
  - *Provide a concise way to identify strings of text*
- Metacharacters allow matching of types of characters
  - *ie. Digits, whitespace, letters etc.*
  - *Quantifiers, alternation and grouping constructs add flexibility into the patterns*
- Lot of documentation available for Perl Regular Expressions online
  - *SAS online documentation contains exhaustive list (link)*
  - *Free online utilities exist to create and test regular expressions (link)*

Perl Regular Expressions (2)

ICON
A Symbol of Excellence

Metacharacter	Description	Example
^	matches the position at the beginning of the input string	^red matches “red” but not “it’s red”
$	matches the position at the end of the input string	red$ matches “red” but not “reds”
\d	matches a digit character	\d matches “0”
[123]	matches any one of the numbers inside the square brackets	1[123] matches “11” but not “14”
[0-2]	matches the numbers 0 to 2 inclusive	1[1-3] matches “11” but not “14”
{n}	matches the preceding sub-expression exactly n times	\d{2} matches “11” but not “1”
|	alternator - specifies the or condition	1|2 matches “1” and “2” but not “3”

Perl Regular Expressions in SAS - PRXPARSE

ICON
A Symbol of Excellence

- Define regular expressions using PRXPARSE
  - *Regular expression passed to PRXPARSE as a string starting and ending with /*
  - *Compiles the regular expression and function output is a reference to location where compiled expression is stored*
  - *Efficient practice to use (If _N_ = 1) block to initialise regular expressions*
- Example

```
data prog1;
  retain pattern;
  if _n_ = 1 then pattern = prxparse("/1[12]/");
  input text $30.;
datalines;
00 01 02
10 11 12
;
run;
```

Perl Regular Expressions in SAS - PRXMATCH

ICON
A Symbol of Excellence

- Basic matching of patterns is done using PRXMATCH
  - *2 arguments, regular expression and string*
    PRXMATCH (regular-expression-id, source)
  - *Usually takes result from PRXPARSE as first argument but can handle "raw" regexp*
  - *Returns the position where the pattern is first found in target string*
- Example

```
data prog1;
  retain pattern;
  if _n_ = 1 then pattern = prxparse("/1[12]/");
  input text $30.;
  match = prxmatch(pattern,text);
datalines;
00 01 02
10 11 12
;
run;
```

Perl Regular Expressions in SAS - PRXPOSN

- Perl Regular Expressions allow for a grouping construct to be used
  - *Sections of the regular expression contained within parenthesis*
  - *In SAS, the part of the pattern matched by each group is assigned to a capturing buffer*
  - *The value stored within each capturing buffer can be accessed by PRXPOSN*
- Example

```
data prog1;
  retain pattern;
  if _n_ = 1 then pattern = prxparse("/(\d{5}) (\d{6})/");
  input text $30.;
  part1 = prxposn(pattern, 1, text);
  part2 = prxposn(pattern, 2, text);
datalines;
12345 123456
98765 987654
```

Regular Expressions for Partial Dates (1)

ICON
A Symbol of Excellence

- Define group for each date component and combine into regular expression for each date type
- Year
  - *Four digits*

```
yyyy = "(\d{4})";
```

    - Month
  - *0 followed by digit in the range 1 to 9*
  - *Or 1 followed by a 0, 1 or 2*

```
mm = "(0[1-9]|1[012])";
```

    - Day
  - *0, 1 or 2 followed by a digit*
  - *Or 3 followed by a 0 or 1*

```
dd = "([0-2]\d|3[01])";
```

Regular Expressions for Partial Dates (2)

- 3 stage process
  - *Use PRXPARSE to create partial date regular expressions*

```
array pattern(8);
pattern1=PRXPARSE("/^"||yyyy||"-"||mm||"-"||dd||"$/");
pattern2=PRXPARSE("/^"||yyyy||"-"||mm||"(--)?$/");
etc.
```

  - *Use PRXMATCH to identify partial date types*

```
do index=1 to 8;
 if PRXMATCH(pattern(index), strip(in_date)) then match = index;
 if match ne . then
  leave;
end;
```

IN_DATE	MATCH
2010-09-18	1
2011-02	2
2010	3
2010---13	4
--07-18	5
---23	6
--08--	7
	8

Regular Expressions for Partial Dates (3)

- 3 stage process
  - *Use PRXPOSN to extract date components into separate variables for further processing*

IN_DATE	CHAR_OUT
2011-02	FEB2011
2010	2010
2013-08	AUG2013
2013	2013
2012	2012

```
select (match);
  when (1) do; *Matched pattern YYYY-MM-DD;
    ...
  end;
  when (2) do; *Matched pattern YYYY-MM or YYYY-MM--;
    year  = input(prxposn(pattern2, 1, strip(in_date)) , 4.);
    month = input(prxposn(pattern2, 2, strip(in_date)) , 2.);

    char_out = "  "||put(month,fmt_mm3.)||put(year,z4.);
  etc.
if _n_=1 then put "IN_DATE    " "CHAR_OUT   ";
put in_date= char_out=;
```

Imputing Missing Information in Partial Dates

ICON
A Symbol of Excellence

- Generalise the constraints or rules for imputing missing information
  - *Generalise imputation to earliest or latest possible value*
  - *Constraint from a "related" date*
  - *Impute within a window (ie. Treatment start to end date)*
  - *The related date could itself be a partial date so this will need to be processed in the same way*
  - *Pass both the input and the related date through the partial date identification process*
  - *Extract date components from both dates*
  - *Impute missing information according to user specification and subject to constraints*

Conclusion

ICON
A Symbol of Excellence

- Regular expressions provide a viable alternative to “traditional” coding methods
- Use of ISO 8601 standard defines a pattern for complete and partial dates
- Use of online tools require limited knowledge of regular expressions to build patterns
- Patterns can be used to:
  - *Identify partial date types*
  - *Flag unlikely types as requiring queries*
  - *Extract component parts to further processing*
  - *Identify missing information that requires imputation*

Contact the Presenter:

Mark Crangle
ICON Clinical Research
South County Business Park
Leopardstown
Dublin 18
Ireland

mark.crangle@iconplc.com