Using the Perl module Data2any

Marcel Timmerman

Using the Perl module Data2any

Marcel Timmerman

Abstract

Data2any is capable of transforming an internally build data structure into xml after which it is possible to store it in a file or send it to any program for conversion or some other type of processor.

Version of yaml2xml when book is written: v0.0.4

Version of Data2any when book is written: $version_Data2any

Generated on date 2014-04-14.

Dedication

To my wife Juhi, who finds me too often behind the computer.

Table of Contents

Introduction iii
1. Who this book is for iii
2. Structure of the book iv
1. Introducing Data2any 1
1.1. lorem 1
1.2. Simple example 1
2. How to use YAML 3
2.1. Short introduction to YAML 3
3. Defining the data structure 4
3.1. The program layout 4
3.2. The data 4
3.3. Use of arrays 4
A. Availability 5
A.1. Availability 5
A.1.1. Installation of the module and program 5
Index 6

Introduction

Welcome to . Data2any is a Perl module to convert data structures into XML. This book will explain all intricasies of the data structure needed by the module and to explain how to write a plugin. Also, explanation of the data structure in Yaml format is given.

1. Who this book is for

There are plenty XML aware editors around with automatic coloring and autocompletion of words to name a few items such editors make use of. So why bother? Well, this book is for users who want to create XML without thinking about the syntax all the time. While this module will solve this for you, it is not a very easy task. The people who aren't strong in programming or programming Perl in particular. For them there is the program and they only need to study the information of the YAML data description language and the installed plugins.

Some more reasons to use the module and/or program are itemized below:

- When using YAML as input for the program a nice input format can give a better view of your text than in full xml code.
- Use of shortcuts in the text such as b[real] will translate to real. This is also an enhancement of visibility. This facility makes it easy to insert tags inlined into your text when the tag doesn't need any attributes.
- Use of preset variables such as $time and $date are nice to make your document a bit more dynamic. The variables are set by the module for later use. It is possible to set new variables at the start of the document, in the text and in the plugins code. There are also special variables called counters.
- Use of plugins can make tedious work simple. Take for example an imaginary plugin which can insert a set of elements to build a table row. Now, instead of writing a static piece of XML:

```
<table>
   <tr>
      <td>table row 1 data col 1</td>
      <td>table row 1 data col 2</td>
   </tr>
   <tr>
      td>table row 2 data col 1</td>
      td>table row 2 data col 2</td>
   </tr>
</table>
```

one could write the following:

```
- table:
   - !perl/Data2any::Html::TableRow
      - table row 1 data col 1
      - table row 1 data col 2
   - !perl/Data2any::Html::TableRow
      - table row 2 data col 1
      - table row 2 data col 2
```

You can easily extend this idea to let this plugin gather information from other sources such as system log files, network servers, websites and databases and put that into a table. Very nice to make a report of some sort.

- Plugins can also help to devide a big document into smaller pieces to make the whole more managable. The pieces can then be maintained by different authors. With some xml processors like it is possible to use xincludes which can do the same.

- Once you've edited your text in the Yaml format you are not tied to that system. Just generate the XML and work from that moment with some editor. However, it will then not be possible (yet) to return back to the Yaml format and you loose the use of plugins and the other facilities.

2. Structure of the book

This book will make its introduction by starting with an example of the Perl code needed to do a minimal conversion. Then makes a comparison with a similar datastructure defined in yaml and introduces the conversion program . Then a lot will be explained using Yaml before coming back to the Perl data structures. These will be shown side by side with the yaml code. Besides converting, the program is capable to send the results to other agents like xslt processors or just to file or the standard output.

- Chapter 1, Introducing Data2any

  This chapter introduces you to the module with a simple example showing the data structure and its result. The same example will then also be compared with a representable piece of yaml code and the use of .

- Chapter 2, How to use Yaml

  This chapter will show you how to write Yaml code which the program will convert to XML. A full description of all the possebilities are given here.

- Chapter 3, Defining the data structure

  In this chapter we will return to the Perl code and describe the datastructure for conversions to XML.

- Chapter 4, Error messages

  This program makes use of several other modules which have all their own way of saying that there is something wrong. In this chapter I will explain the messages and what they all mean.

- Chapter A, Availability

Chapter 1. Introducing Data2any

1.1. lorem

1.2. Simple example

Lorem ipsum

```
use Modern::Perl;
require Data2any; (1)

my $data = (2)
[[{ DOCTYPE => { root => 'html'}}
 ,{ html =>
    [{ body =>
       [{ 'h1 id=hw class=helloWorld' => 'Hello World Example'}
       ,{ p => 'Hello World, this is <b /> easy! Wrote this'
             . ' on $date at $time!'
        }
       ]
     }
    ]
  }
 ]
];

my $d2xml = Data2any->new (3)
   ( inputData => $data
   , dataLabel => 'internal'
   , logging => 1
   , requestDocument => 0
   );

$d2xml->nodetreeFromData; (4)
$d2xml->xmlFromNodetree; (5)
$d2xml->postprocess; (6)
```

(1) An important ingredient, import the module Data2any. In Perl this is nessesary to get the module code known to the program.

(2) $data is the variable we use here to build our structure. It looks already complex while the output is quite simple.

(3) The next step is to call the constructor of the module and set a few attributes to work with.

(4) First step in translation is to convert the data into a tree of nodes. These are AppState::Node-Tree::Node objects belonging to another module. For simple translations it isn't nessesary to know the how they are build. The node tree can be retrieved from the $d2xml object if needed.

(5) Second step is to generate the xml in text format. The text can be retrieved from the $d2xml object.

(6) The last step is to add some extra's like a httpheader, or to send the xml to other processors for translation or storage.

Which will produce the following

```
<!DOCTYPE html>
<html>
  <body>
<h1 class="helloWorld" id="hw">Hello World Example</h1>
<p>Hello World, this is <b>real</b> easy! Wrote this on
2013-11-09 at 17:18:15!</p>
  </body>
</html>
```

Chapter 2. How to use YAML

2.1. Short introduction to YAML

YAML or “” (™)[1]is a data description language designed by Oren Ben-Kiki, Clark Evens and Ingy döt Net. The purpose of this language is to describe your data after which a program loads it using a Perl module. Other languages like Ruby and Python also have modules to read YAML. Data2any uses YAML and translates it to xml. Because of this the program will use only a subset of YAML and the definitions must be given in a specific way and order.

[1]Manual can be found here [http://www.yaml.org/spec/1.2/spec.html]

Chapter 3. Defining the data structure

3.1. The program layout

Before going into details I will show you again the global layout of the program. The program must be prepared before using any of the modules functions. This can be accompleshed by importing the module. Then the data must be prepared after which the initialization, the conversion and generation of XML will take place.

```
# Importing modules
use Modern::Perl;
require Data2any;

# Creating the data structure
my $data = [[...][...]];

# Initialize module with the data
my $d2xml = Data2any->new( inputData => $data
                         , dataLabel => 'internal'
                         , logging => 1
                         , requestDocument => 0
                         );

# Convert to XML
$d2xml->nodetreeFromData;
$d2xml->xmlFromNodetree;
$d2xml->postprocess;
```

We will first explain the data structure now before we go into the details of the used functions.

3.2. The data

The data specification describes a set of documents as a set of array references in an array reference like the following example is a declaration of a set of two douments.

```
my $document1 = [];
my $document2 = [];
my $data = [ $document1, $document2];
```

3.3. Use of arrays

This chapter describes the data structure which is needed for the Data2any module. The data structure needed for the program is designed in such a way that many situations can be described.

Appendix A. Availability

Abstract

lorem

A.1. Availability

The program is developed on a Linux platform. I suppose it will be able to work on any Unix platform with a reasonable recent Perl version and CPAN modules. Other operating systems still need to be investigated.

A.1.1. Installation of the module and program

Installation steps are simple for Linux systems. For this and other systems the following instructions can be followed:

- Linux

  > su
  ... password ...
  # yum
  # cpan Data2any
  ...

  Windows XP

  Windows 8

Index