UdReader User's Manual

Matt Donnelly

March 1, 2014

Contents

1 Introduction

UdReader is a library of reading/parsing routines that attempt to parse most common utility data formats used for storing time series data. The goal of the software is to provide a flexible library, useful on most any platform and for most any set of analysis or plotting tool available and commonly used in the electric power industry.

2 Quick Start

In Matlab:

```
NET.addAssembly('fullpathtolibrary\UdReader.dll');
import MTech.UdReader.*
rdr = UdReader('filename.ext');

% Get all analog data from the file
alldata = rdr.GetData();

% Create a time vector
t = alldata.Data(:,1);

% Plot the first channel
plot(t, alldata.Data(:,2)), legend(alldata.Name{2})
```

3 API Overview

3.1 Namespace

The API classes are defined in the *MTech.UdReader* namespace:

```
using MTech.UdReader;
```

3.2 Classes

At present there is exactly one class exposed in the MTech.UdReader namespace, the *UdReader* class. The only valid constructor takes a string argument containing the name of the file to be read using either relative or absolute path.

```
UdReader rdr = new UdReader("mycomtradefile.cfg");
```

3.3 Public Properties

Once a class is established one property, *FullFileInfo* is exposed to the user. The FullFileInfo property contains a full listing of the typical properties of all channels in the file. The property is a struct[1] in a format similar to the JSIS Matlab format. Assuming a valid UdReader named *rdr* has been instantiated, you would get the contents of the FullFileInfo property as follows:

```
FileContents info = new FileContents();
info = rdr.FullFileInfo;
```

The *FileContents* struct has the following format (in F#[2] syntax):

```
type FileContents = {
    StartTime : DateTime
    TimeUnits : float
    Name : string array
    Type : string array
    Units : string array
    Title : string
    Data : float [,]
    }
```

You would therefore get the StartTime, assuming you have a FileContents object named *info*, as follows:

```
DateTime start = new DateTime;
start = info.StartTime;
```

In F# the same code might look like this:

```
let start =
  UdReader("mycomtrade.cfg").FullFileContents.StartTime
```

The fields of the struct have the following meaning:

- StartTime is a System.DateTime object representing the time at which t = 0 in the data. The time at which t = 0 is commonly the first point in the data file, but not always. (Note: The time vector is the first column of the Data array. See below for more information.)

- TimeUnits is a System.Double representing the number of seconds elapsed in one interval of the time vector. In most cases TimeUnits will be 1.0, representing one second. Say, for example, that TimeUnits were 0.001. Then the elapsed time between 1.0 and 2.0 in the time vector would be one millisecond. In this manner we can accomodate a wide range of time spans from years to microseconds.

[1]It's actually an F# record, not a C# struct. They are close enough in useage to allow the mis-statement.

[2]Note that an F# *float* is a .NET System.Double.

- Name is an array of System.String containing the names of all channels in the Data array. The first entry in the Name array should be "Time". There is an exception for the FullFileInfo struct where the Data array is empty and Name contains all names available in the file. This feature is often used for searching through all of the channel names.
- Type is an array of System.String containing the type of data in the channel. A channel type is "V", "I" "F", etc. Providing informative type designators for all kinds of data files is a challenge, and we are certain that the Type field will evolve over time.
- Units is an array of System.String containing the units associated with the channel. Units are, for example, "kV", "A", "HZ".
- Title is a System.String containing the title of the data file, if any.
- Data is a 2D array of System.Double containing the scaled data read from the file. The first column of Data is always time.

3.4 Public Methods

The UdReader class exposes three methods, each with overloads.

3.4.1 GetData

The GetData method takes one argument – an array of integers specifying which channels the user wishes to retrieve from the data file. An overload allows the user to get all channels by calling the GetData method without any arguments. The method returns a FileContents object containing metadata and results for the selected channels

```
// Returns all channels
FileContents allChans = rdr.GetData();

// Return channel 7
int[] c7 = new int[1] {7};
FileContents chan7 = rdr.GetData(c7);
```

3.4.2 FindByName

The *FindByName* method searches the *Name* field of FullFileInfo using regular expressions. Regular expressions provide a powerful search framework. To learn more about how to form regular expression search strings the reader can google it. Entering a simple search word works fine if you don't want to become an expert in regular expression syntax. The method returns an array of integers suitable for use by GetData.

```
int[] found;
// Returns indices found by searching on srchStr
found = rdr.FindByName("findme");

// Return indices matching the search
string[] substations = new string[3] {"Sub[0-9]", "Bus[45]"};
found = rdr.FindByName(substations);
```

3.4.3 FindByNamei

FindByNamei and its overload works exactly the same as FindByName but with case-insensitive matching.

4 Helper Functions and Common Workflows