Labview Utilities for ECE331

Request for Comment:
The Simple ASCII Data Format (SADF)
Version 0.1

Mark Woolston

2008 January 03

Summary

While the LabView instrument control software makes it possible to control and query test equipment such as oscilloscopes, power supplies and function generators, saving the data generated for use in lab reports, spreadsheets, and math-package post-processing is hampered by the lack of a commonly readable measurement file format. To be of the most use in an undergraduate environment, such a file format should meet the following goals:

It should:

- Be easy to use and not distract students from the laboratory exercises.
- Be human-readable for easy inclusion into lab reports.
- Be writeable from LabView.
- Be readable by common spreadsheet software such as Microsoft Excel and OpenOffice.org Calc.
- Be readable by common math packages such as Matlab.
- Encourage good laboratory practices such as recording instrument setup and experiment purpose.

This document introduces one possible file format that is intended to meet these goals. For lack of a catchier name, it has been named Simple ASCII Data Format (SADF). At the time of writing, LabView VI's have already been created to write data in the SADF format and Matlab helper functions have been written to import SADF data. Since SADF is based upon the Comma Separated Value (CSV) format and uses the .csv file extension, Excel can already read the format directly.

Details

The Simple ASCII Data Format (SADF) is based upon the widely supported Comma Separated Value (CSV) format. This is a plain-text format used to store 2-dimensional tables of data. Data column entries are separated by a separator character (usually a space or comma), and frequently delimited by a delimiter character (usually a single or double quote). Data rows are separated by end-of-line characters, generally the carriage return – linefeed combination on Windows machines.

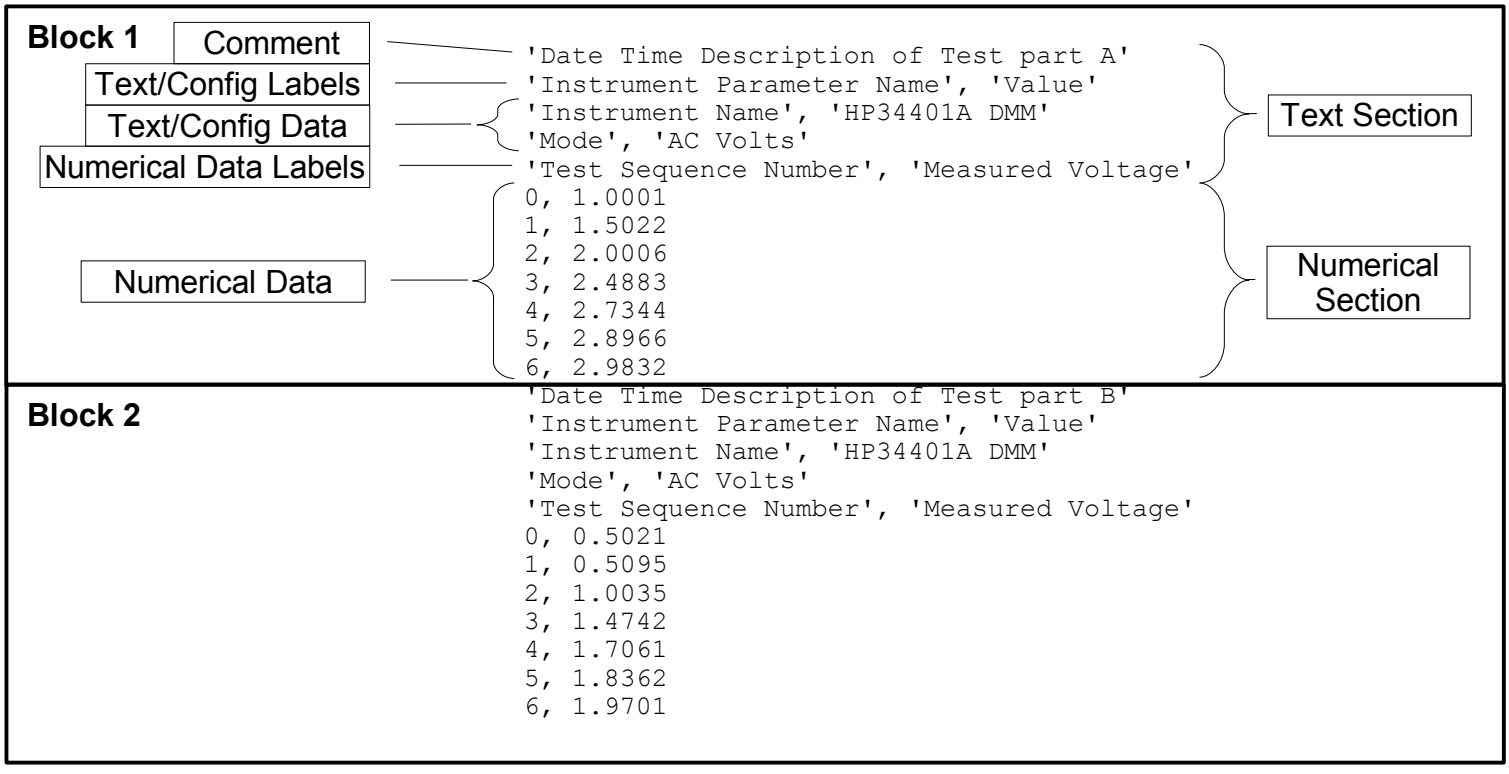
A Simple ASCII Data Formatted file is essentially a CSV file with a particular format. The format is based on 'blocks', where each block is a set of data from a particular measurement event. Multiple blocks can be saved into a single file.

A SADF block consists of the following:

- A Text section consisting of single-quote (') delimited text fields:
  - An optional one-line Comment
  - An optional one-line row of comma-separated Text/Config labels
  - Optional multiple lines of comma-separated Text/Config Data strings, usually consisting of instrument parameter name and value pairs with each pair on a separate line
  - A one-line row of comma-separated Numerical Data Labels

- A Numerical section (non-delimited) consisting of:
  - Multiple lines of comma-separated Numerical Data formatted as ASCII text. Typically each line will consist of all measurements by the instrument taken “at the same time”, so that time increases with each line.

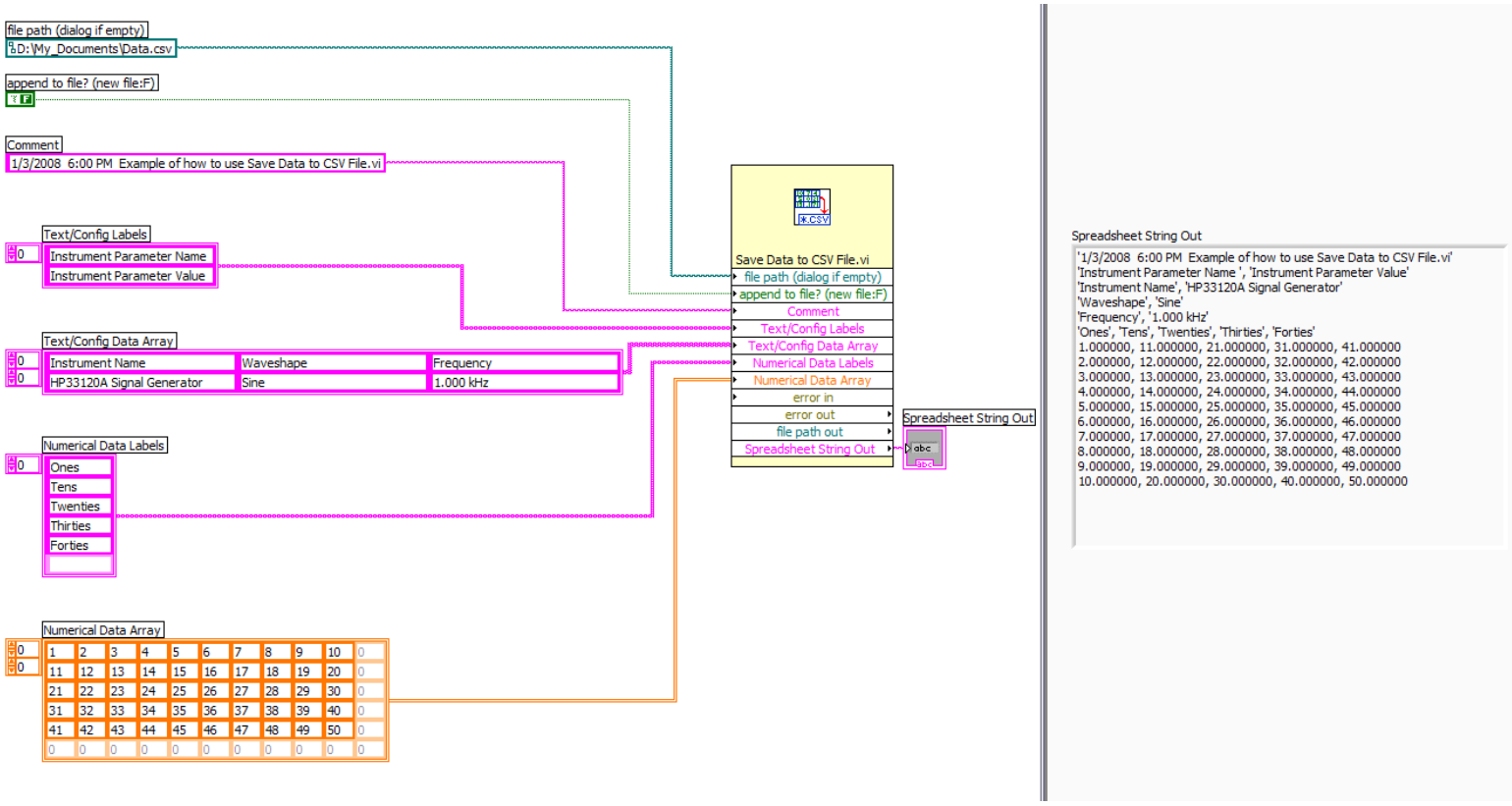
Example: `example.csv`



LabView Support

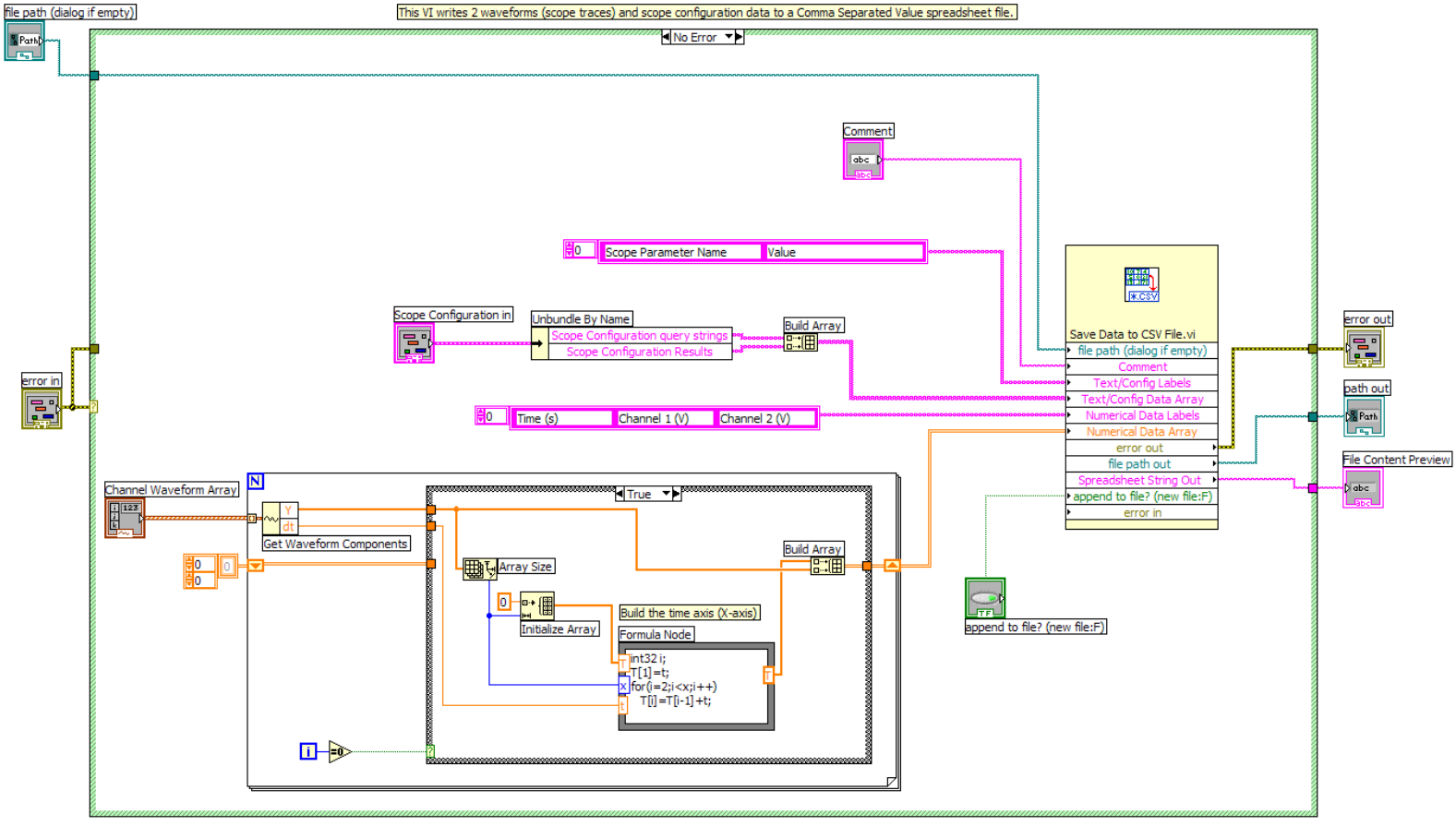
LabView support for the Simple ASCII Data Format (SADF) is provided by the custom VI “Save Data to CSV File.vi”. The very simple example below saves one block of constant data to a new file. In actual use the constant data would be replaced by data read from instruments and the user's entries.

“Save Data to CSV File.vi” is also used by the higher-level VI's specifically designed for particular instruments, such as the HP54600B oscilloscope.



LabView Support continued

The example below shows how “Save Data to CSV File.vi” can be used to store oscilloscope data.



Matlab Support

Matlab support for the Simple ASCII Data Format is provided by the custom helper function “Import_SADF_Data.m”, a portion of which is shown below.

```
function [D,S]=Import_SADF_Data()
% Presents a file-selection dialog and imports text (typically instrument
% setup information) and numerical data (typically instrument measurements)
% from a Simple ASCII Data Formatted (SADF) Comma Delimited Value (.csv)
% file, as saved by the LabView program HP54600B_ReadTraces.vi.
% [Data, Setup]=Import_SADF_Data
%
% Data is accessed as follows: Data(blocknumber,samplenumber,column)
%  where:
%   "blocknumber" is the acquisition number in the file, starting with 1.
%   (Blocknumber is used when multiple captures (acquisitions) were
%    saved to the same file.)
%   "samplenumber" is the measurement sample number, starting with
%    1 as the first data point taken. If the data is from an oscilloscope,
%    this will correspond to leftmost point visible on the scope screen.
%   "column" is the column in file, starting with 1. Typically each column
%    represents a different measurement type, such as time of measurement,
%    voltage at point A, current at point A, voltage at point B, etc.
%    If the data is from an oscilloscope, this will typically correspond
%    to the time axis (as channel 1) and the scope's input channels 1 and 2
%    as columns 2 and 3 respectively.
%
% Setup is accessed as follows:
% Setup{blocknumber,itemnumber}{column}{1}
%  where:
%   "blocknumber" is as above.
%   "itemnumber" is a row index into the list of text items.
%    Typically the first item is a user comment and does not have a 2nd
%    column.
%   "column" is a column index into the list of text items.
%    Usually column 1 is an item name, and column 2 is a value.
%    If the data is from an oscilloscope, the following is typical:
%    column=1 will retrieve the parameter name (frequently a GPIB command)
%    column=2 will retrieve the value returned by the scope.
```

This example shows how the “Import_SADF_Data” function can be used:

```
[Data,Setup]=Import_SADF_Data;       % Import scope data and scope settings information

                                     % This file had multiple blocks:
Time=Data(:,:,1);                    % Extract all Time vectors
Vds=Data(:,:,2);                     % Extract all Channel 1 data vectors
Vdd=Data(:,:,3);                     % Extract all Channel 2 data vectors
Ids=Vdd-Vds;    % note Rd = 1 ohm    % Subtract the two channels

figure(1);                           % Bring up Figure Window 1
plot(Vds',Ids');                     % Plot Ids vs Vds. Note the transpose operator (')
title(Setup{1,1}{1}{1});             % Extract the first Comment field and use it as the plot title
xlabel('Vds (V)');                   % Assign an X-axis label
ylabel('Id (A)');                    % Assign a Y-axis label
grid on;                             % Enable grid lines on the plot
% Give a legend to show which traces are which (it was known there were 8 blocks)
legend({'Vgs=1V','Vgs=2V','Vgs=3V','Vgs=4V','Vgs=5V','Vgs=6V','Vgs=7V','Vgs=8V'});
```