A Configuration File Editor for a Wind Tunnel Data System

Stephen B. Jenkins*
Institute for Aerospace Research
National Research Council
Ottawa, Canada

Presented at YAPC::NA
St. Louis, Missouri
June, 2002

Abstract

A configuration file editor has been created at the Aerodynamics Laboratory of the Institute for Aerospace Research. This program allows the clients and staff of the 2m x 3m wind tunnel to modify the files that control the data acquisition and reduction processes. Because modifications to the large quantity of "C" code that interacts with these files were deemed to be impractical, a solution has been devised that is compatible with the legacy file format, while providing the benefits of a web-based interface.

1 Introduction

As part of the overall plan to provide web-based software tools for the Aerodynamics Laboratory, an application has been written to allow users to edit the contents of the configuration files used to control the data acquisition and reduction processes[1-3]. Previously, the clients and staff of the facility had been using a terminal-based (non-GUI) text editor to modify these files.

Each test at the 2m x 3m wind tunnel requires a minimum of two configuration files: **`Mod###.cfg`** and **`Run###.cfg`** where the test number replaces "`###`" (e.g.: **`Mod742.cfg`**). The **`Mod###.cfg`** file usually contains client-specified parameters that pertain to the configuration of the model in the wind tunnel (Fig. 1). Typical examples are the flap angle, the landing gear position, and the tail configuration. Ordinarily, changes to the model configuration file are the responsibility of the client or the test engineer. The **`Run###.cfg`** file normally contains parameters that relate to the current state of the data system (Fig. 2). Typical examples for this file are the number of tare points, the independent variable, and the total number of acquisition channels. The wind tunnel operator usually takes responsibility for the run configuration file. Occasionally, tests have one or two additional configuration files for information that does not fit into either of the two previously mentioned categories. The **`Red###.cfg`** file is used to provide additional parameters to control the data reduction processes, and the **`Ext###.cfg`** file is used to store "extra" control information for ancillary data acquisition systems.

2 Configuration file description

2.1 Legacy file format

The original structure of these configuration files was extremely simple. There was a single line of text for each parameter, with each line containing a description string, a colon as a separator, and the current value of that parameter.

```
Description : Value
```

It should be noted that the "Description" portion of each line was originally provided only as a prompt

* Senior Programmer/Analyst – http://www.nrc.ca/~jenkins

to the user, and is not used by the data acquisition/reduction programs. The assignment of the value to its corresponding variable in the code is done on a purely positional basis, i.e.: the value of the third line in the file is always assigned to the third input variable. In addition, the length of each line of the configuration file is restricted to 80 characters. Making any changes to this rather crude file structure was completely out of the question due to the large quantity of legacy "C" code making up the vast majority of the data system. Any new software would have to operate within the confines of this file format.

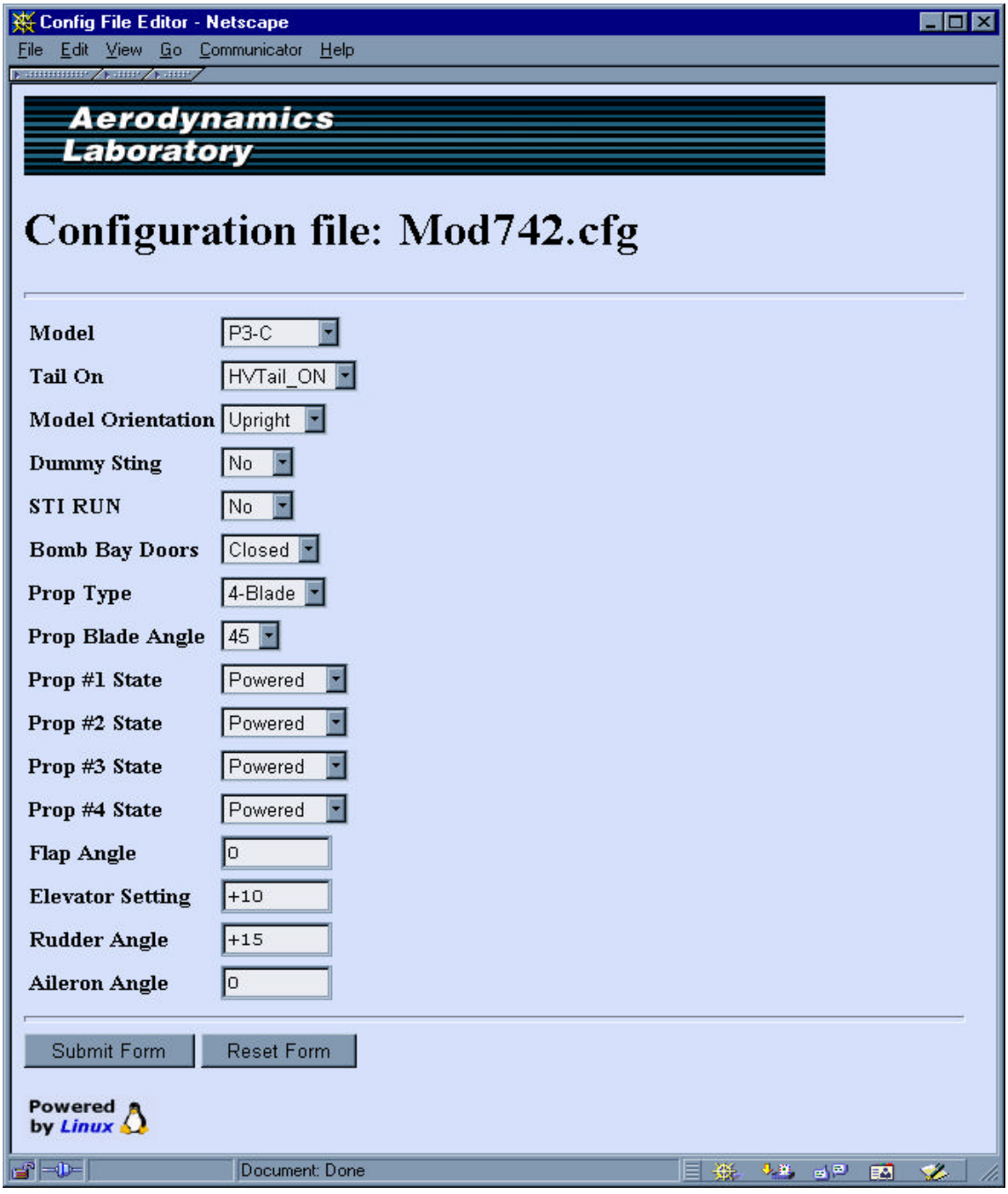

Figure 1: Sample model configuration file

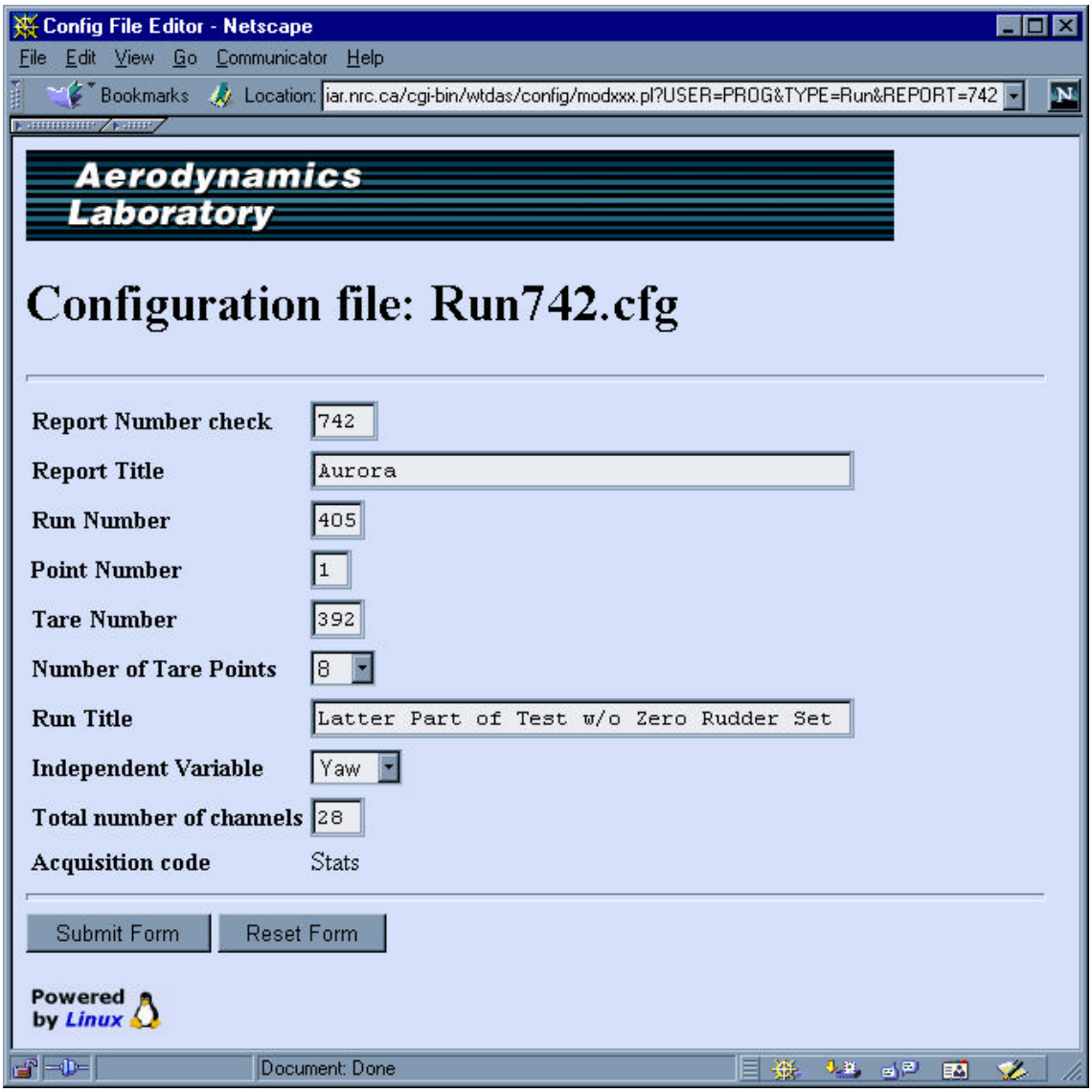

Figure 2: Sample run configuration file

2.2 New file format

During the design phase for the web-based editor (called **`modxxx.pl`**) it was realized that one of the weaknesses of the legacy file format could be turned into an advantage. Since the parameter description portion of each line is effectively ignored by the data system, it seemed to be the obvious place to embed any additional information required by the editor. It was decided to modify the format of each line to the following: a description string; a semi-colon separator; an editor control string; a colon separator; and the current value of the parameter.

```
Description ; Control string : Value
```

The addition of a different separating character allows the legacy code to continue operating without changes (it still need only extract the value to the right of the colon) and also provides an unambiguous method of providing the editor with the information that it requires.

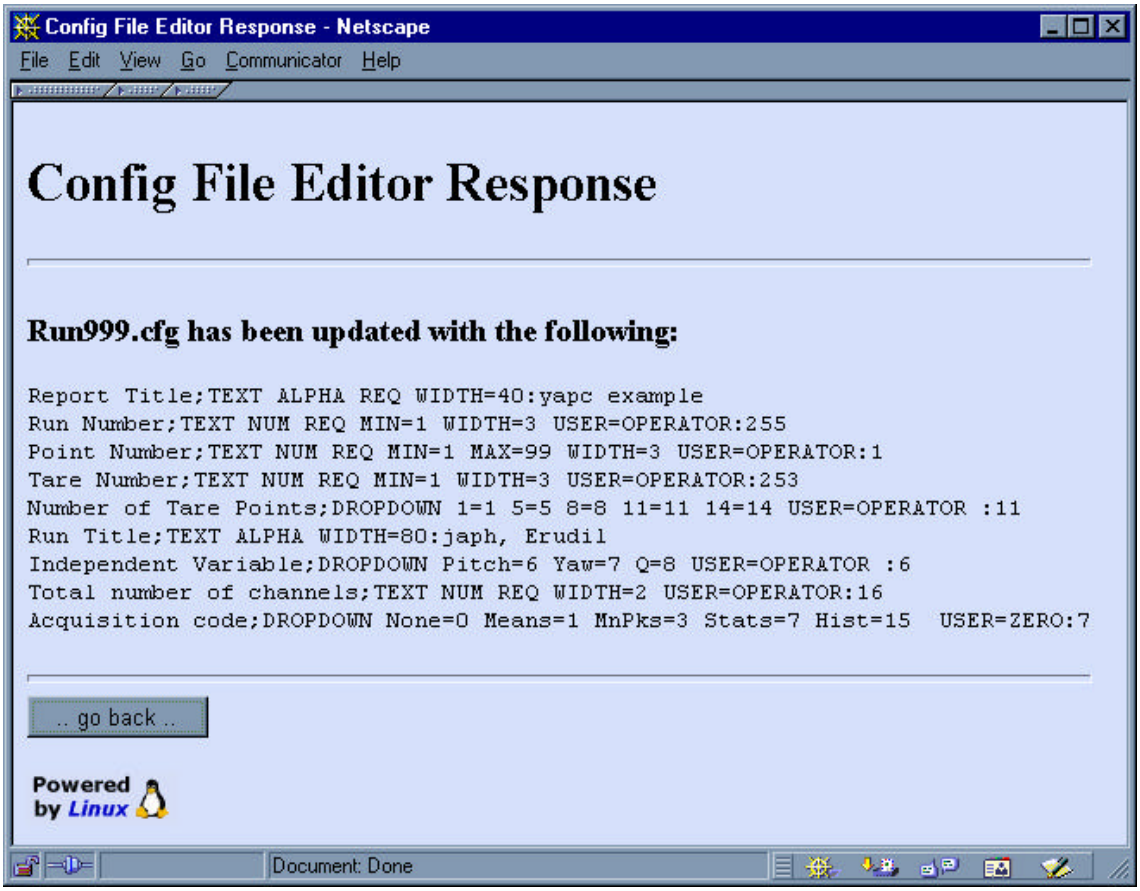

Figure 3: Editor response

3 Control string description

3.1 Input field types

The first portion of the editor control string is expected to be a tag that denotes the input field type. This information is then used to control how the remainder of the string is interpreted. It should be pointed out that the editor control string components were deliberately kept rather terse because of the previously mentioned 80-character limit on the length of the entire line.

It was decided that only two data input field types would be necessary: a dropdown menu, and a text entry box. The dropdown menu could be used for parameters with a small number of discrete choices (e.g. binary flags such as “bomb bay doors open/closed”). The text entry box could be used for parameters that required a large or unlimited number of choices (e.g. floating point values such as “rudder angle”). While the text entry box alone would have been sufficient for all types of input, the dropdown field type has been included, as selecting a value from a dropdown menu is more convenient and less prone to input errors than typing a value.

The web page shown in Fig. 3 is typical of the type of response generated by **`modxxx.pl`** after the successful update of a **`Run###.cfg`** file. The nine lines rendered in "`Courier`" font are the configuration file's contents. This response page allows the user to view the data exactly as it appears on disk, and provides a method of verifying the proper operation of the editor.

3.2 Text field options

The **`TEXT`** field type has six optional "tags" that allow the creator of the configuration file to constrain the value of the user supplied data. The simplest is the **`REQ`** tag, which signifies that the

parameter is required and may not be left blank. The **NUM** and **ALPHA** tags are mutually exclusive and cause the editor to require the value to contain only numeric or alphanumeric characters respectively. The last three tags each require an equals sign ("=") followed by an appropriate value. The **MIN** and **MAX** tags are used to limit the minimum and maximum numerical magnitude of the user-supplied input. The **WIDTH** tag sets the maximum allowable string length. For an example of the use of these tags, refer to the third line of the configuration file shown in Fig. 3 (the line beginning with "Point Number"). The value for that parameter must be numeric, from 1 to 99, and be a maximum of 3 characters in length.

3.3 Dropdown field options

Since the user is constrained by the options specified in the **DROPDOWN** field type, it does not require any tags for that purpose. Instead, the editor control string is expected to contain a list of key-value pairs. The keys are used in the HTML select element, but it is the value of the selected key that is used for that parameter. This was done in order to allow the user to see easily recognizable strings for the many parameters where the legacy data system requires numeric values. For an example, refer to the last line of the configuration file shown in Fig. 3 (the line beginning with “Acquisition code”). In that case, the value has been set to "7", implying that the user had selected the "Stats" option from the dropdown menu.

3.4 Field protection option

During beta testing, it became apparent that a method would be required to protect critical parameters from being inadvertently modified by unwary users. This was largely because it was now "too easy" to make changes to the configuration files. It was decided to provide for the addition of an optional **USER** tag to any **TEXT** or **DROPDOWN** field. As with the **MIN**, **MAX** and **WIDTH** described above, this tag also requires an equals sign followed by an appropriate value, which in this case is a string describing the "level" of user that is allowed to modify this parameter. For an example of the use of this tag, refer to the last three lines of the configuration file shown in Fig. 3.

4 Editor implementation

4.1 Configuration file creation

Before the beginning of a wind tunnel test, members of the Software and Instrumentation Group consult with the clients and/or the test engineer to acquire the information needed to produce the configuration files. A standard text editor is used to create each line of every file with an appropriate parameter description string, editor control string, and initial value.

4.2 Editor invocation

The editor is accessed through web “home pages” containing URLs that have been individually customized for clients, tunnel operators, and programming staff. When a user clicks on the link to modify a particular configuration file, **modxxx.pl** dynamically creates the editor web page from the description, control, and value information from each line of that file, as well as from the information contained within the URL string.

4.3 URL options

The “Location:” box of the browser shown in Fig. 2 contains an example of the three editor control parameters that may be specified in the URL: USER, TYPE and REPORT. (In the jargon that has developed within the laboratory, the report number is synonymous with the test number mentioned in the “Introduction” section.)

The first URL option, “USER”, is utilized in conjunction with the **USER** tag from the editor control string of each line of the configuration file to provide access control. The value of the USER option is compared to the value of each **USER** tag. If the "USER" is greater than the **USER** in an "ASCIIbetical" sense, **modxxx.pl** creates the appropriate HTML to allow the modification of that parameter's value. If not, the current value is displayed without the ability to make changes. See the last line of the configuration file in Fig. 2 for an example. In this case, the URL shows that the "USER" is "PROG", and that that user has not been given the ability to modify the value of the "Acquisition code". While this system provides no

real *security*, it fulfills its intended purpose of providing *protection.*

The second and third URL options, "TYPE" and "REPORT" are used by the editor to construct the name of the configuration file to be edited. In Fig. 2, the URL parameters are used to access the file named "Run742.cfg". If the "TYPE" option is omitted, **`modxxx.pl`** uses the default value "Run". If the "REPORT" option is omitted or contains any non-digits, the number for the current wind tunnel test is used.

4.4 Data validation

During the creation of the HTML for the web form, **`modxxx.pl`** also dynamically creates the JavaScript code required to perform data validation and constraint enforcement within the browser. This technique has been used in order to provide a more responsive user interface than traditional CGI methods would allow [4,5]. If the user attempts to submit data that are not within the limits specified by the constraints, the JavaScript program displays a dialog box describing the errors. In order to facilitate the identification of the offending values, this program uses a standard image swapping technique to make flashing red arrows appear beside the text entry boxes. Figure 4 shows an example of an attempt to submit a form with two errors. Only when the user-supplied data conforms to the constraints will the form actually be submitted to the CGI program to update the configuration file on disk.

4.5 File update

Because of the fact that the legacy "C" code occasionally writes to the configuration files, it was decided to provide an extremely crude form of protection against "clobbering" the newly updated data. When the file is read by **`modxxx.pl`**, the system time is noted. After the user modifies and submits the file, but before it is actually written to disk, the file's last update time is read and compared to aforementioned system time. If the update time is more recent, the user is notified and instructed to press “Reload” in the browser, refreshing the form with the new contents. While this method is hardly foolproof – there is an inherent "race condition" – it is sufficiently robust for this particular application.

5 Concluding remarks

The CGI program **`modxxx.pl`** provides many significant improvements over the previous method of modifying the configuration files using a text editor. The most obvious benefit to end-users is from the familiar and robust web interface; they are not required to learn how to use a new application. The primary benefit to the staff of the software and instrumentation group is the ability to easily add configuration parameters and, through the use of the URL "TYPE" option, entirely new configuration files – all without the need to make changes to the editor program itself. In addition, because the data validation captures many input errors, there is less wind tunnel time lost due to configuration mistakes. Finally, because the editor can run on any web browser, it is trivially distributed, and there is no requirement to install special software for those clients who wish to use their own computers during the experiment.

The most serious shortcomings of the editor are a result of the necessity to maintain backwards compatibility with the legacy "C" code in the system. At the time of this writing, that code is being rewritten, and the new configuration information will be stored using XML format. The problems of asynchronous updates and file locking will also be addressed, possibly through the use of a database.

It should be stressed that **`modxxx.pl`** has been designed to be a laboratory tool, not a commercial application. Some of these design decisions (such as the use of JavaScript for field protection and data validation) only make sense within the confines of a secure Intranet. Also, the slight possibility of race condition errors makes the program rather unsuitable for use outside of a controlled research environment. The acceptance of these limitations was due to a conscious decision to follow Yourdon's philosophy of "good enough" software as discussed by Hunt and Thomas in "*The Pragmatic Programmer*"[6]. The additional time and effort that would have been required to write the program to meet commercial software standards would have far exceeded any potential gains in productivity. The correctness of this decision has been proven out by the universally positive response to the editor program by the clients and staff of the Aerodynamics Laboratory.

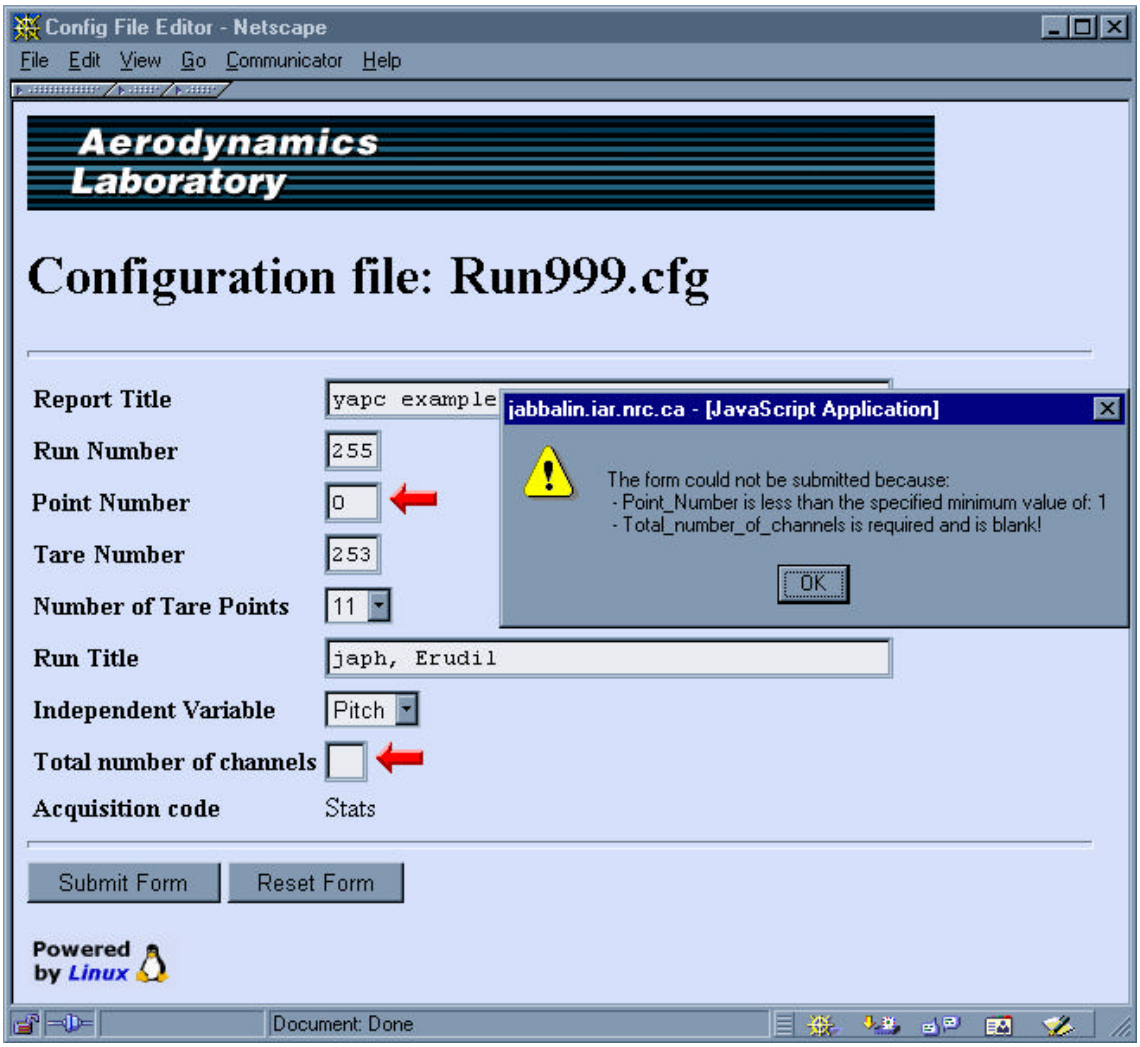

Figure 4: Submission error example

6 References

[1] - Jenkins, S.B., *Using Perl to Create Web-Based Software Tools for Wind Tunnel Testing.* AIAA 2001-905, American Institute of Aeronautics and Astronautics, 39th Aerospace Sciences Meeting & Exhibit, Reno, NV, Jan. 2001.

[2] - Jenkins, S.B., *Open Source Software at the Aerodynamics Laboratory*, Linux Journal Issue #90, Oct. 2001.

[3] - Johnson, K., *QNX System and Software Support Notes for the 2m by 3m Wind Tunnel*, LM-AL-2001-0098, National Research Council of Canada, Nov. 2001.

[4] - Jenkins, S.B., *Generating JavaScript from Perl*, Dr. Dobb's Journal Issue #336, May 2002.

[5] - Flanagan, D., *JavaScript: The Definitive Guide, Third Edition.* O'Reilly & Associates, 1998.

[6] - Hunt, A. and Thomas, D., *The Pragmatic Programmer*. Addison-Wesley, 2000, pp. 9-11.