DRAFT

DOCUMENTATION FOR THE FRAMES-HWIR TECHNOLOGY SOFTWARE SYSTEM, VOLUME 10: FACILITATING DYNAMIC LINK LIBRARIES

Project Officer
and Technical Direction:

Mr. Gerard F. Laniak
U.S. Environmental Protection Agency
Office of Research and Development
National Environmental Research Laboratory
Athens, Georgia 30605

Prepared by:

Pacific Northwest National Laboratory
Battelle Boulevard, P.O. Box 999
Richland, Washington 99352
Under EPA Reference Number DW89937333-01-0

U.S. Environmental Protection Agency
Office of Research and Development
Athens, Georgia 30605

October 1999

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC06-76RLO 1830

Acknowledgments

A number of individuals have been involved with this effort. Mr. Gerard F. Laniak of the U.S. Environmental Protection Agency (EPA), Office of Research and Development, National Environmental Research Laboratory, Athens, Georgia, provided the overall technical direction and review throughout this work. This report was prepared by the Pacific Northwest National Laboratory[1] (PNNL) staff of Karl Castleton, Mitch Pelton, Gene Whelan, John Buck, Gariann Gelston, Bonnie Hoopes, Regina Lundgren, John McDonald, and Randal Taira. Additional PNNL staff supporting this effort include Wayne Cosby, Nancy Foote, Kristin Manke, Jill Pospical, Debbie Schulz, and Barbara Wilson. Useful inputs were provided by many U.S. EPA individuals working on the Hazardous Waste Identification Rule, including Messrs. Barnes Johnson, Stephen Kroner, and David Cozzie, and Drs. David Brown, Robert Ambrose, Zubair Saleem, Donna Schwede, and Sharon LeDuc, among many others.

[1]Operated by Battelle for the U.S. Department of Energy under Contract DE-AC06-76RLO 1830.

Summary

The U.S. Environmental Protection Agency is developing a comprehensive environmental exposure and risk analysis software system for agency-wide application. The software system will be applied to the technical assessment of exposures and risks relevant to the Hazardous Waste Identification Rule (HWIR). The software system adapted to automate this assessment is the Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES), developed by the Pacific Northwest National Laboratory (PNNL). The process used to develop the FRAMES-HWIR Technology Software System includes steps for requirements analysis, design, specification, and development with testing and quality assurance composing a critical portion of each step.

This report documents the input, scientific, and output requirements of a set of facilitating dynamic link libraries (DLLs) shared by system components. The purpose of the two facilitating DLLs is described, including their ability to allow proper functioning of all FRAMES-HWIR Technology Software System components. The two appendixes to this document provide additional details on the testing program for the facilitating DLLs. Other components developed by PNNL are described in companion documents listed in Section 6.0. The system is documented in a summary report entitled *Overview of the FRAMES-HWIR Technology Software System.*

Acronyms and Abbreviations

ASCII	American Standard Code for Information Exchange
DLL	Dynamic Link Library
ELP	Exit Level Processor
EPA	U.S. Environmental Protection Agency
FRAMES	Framework for Risk Analysis in Multimedia Environmental Systems
GRF	Global Results Files
HWIR	Hazardous Waste Identification Rule
HWIRMC.DLL	HWIR Monte Carlo Dynamic Link Library
HWIRIO.DLL	HWIR Input/Output Dynamic Link Library
MET	meteorological
MMSP	Multimedia Multipathway Simulation Processor
MSVCP	MicroSoft® Visual C ++
OCRWM	Office of Civilian Radioactive Waste Management
PNNL	Pacific Northwest National Laboratory
SDF	Site Definition Files
SDP	Site Definition Processor
SSF	Site Simulation Files
SUI	System User Interface

Contents

Figures

Tables

1.0 Introduction

The U.S. Environmental Protection Agency (EPA) is developing a comprehensive environmental exposure and risk analysis software system for agency-wide application. The software system will be applied to the technical assessment of exposures and risks relevant to the Hazardous Waste Identification Rule (HWIR). The HWIR is designed to determine quantitative criteria for allowing a specific class of industrial waste streams to no longer require disposal as a hazardous waste (that is, such streams may *exit* Subtitle C) and to allow disposal in Industrial Subtitle D facilities. Hazardous waste constituents with values less than these exit criteria levels would be reclassified as nonhazardous wastes under the Resource Conservation and Recovery Act.

The software system adapted to automate this assessment is the Framework for Risk Analysis in Multimedia Environmental Systems (FRAMES), developed by the Pacific Northwest National Laboratory (PNNL). The FRAMES-HWIR Technology Software System consists of a series of components within a system framework (Figure 1.1). The process used to develop the FRAMES-HWIR Technology Software System includes steps for requirements analysis, design, specification, and development with testing and quality assurance comprising a critical portion of each step.

To allow consistent, efficient functioning of all system components, a collection of facilitating dynamic link libraries (DLLs) has been developed. A DLL is a modular set of routines that link to their calling programs at run time, rather than during compilation. The set of such files is somewhat comparable to the library routines provided with programming languages such as FORTRAN, C, and C++.

The facilitating DLLs are the HWIR Input/Output Dynamic Link Library (HWIRIO.DLL) and the HWIR Monte Carlo Dynamic Link Library (HWIRMC.DLL). This document covers those two DLLs. Another system component also implemented as a DLL is the Chemical Properties Processor. This component is covered in separate documentation (see Section 6.0 for additional information).

This report includes information on requirements and design information for the three facilitating DLLs. It also discusses their testing plans, testing results, and the quality assurance program under which they were developed. Specifications for the three facilitating DLLs are described in *Documentation of the FRAMES-HWIR Technology Software System, Volume 8: Specifications*. References cited in the text are listed in Section 6.0. Appendices A, B, and C provide additional details on the testing program for the three facilitating DLLs. Other components developed by PNNL are described in companion documents as listed in the reference list; the system itself is documented in a summary report entitled, *Overview of the FRAMES-HWIR Technology Software System.*

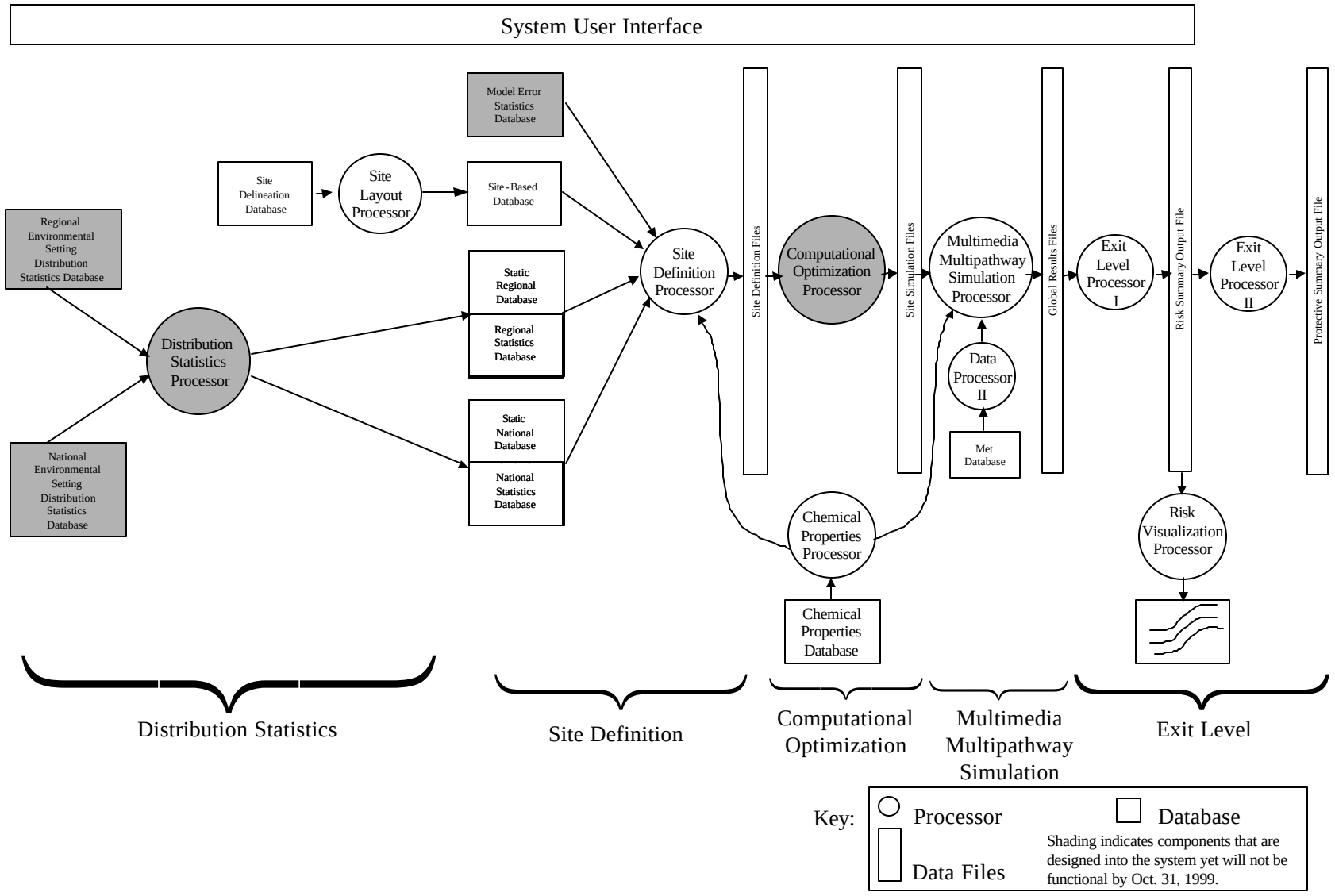

Figure 1.1 Overview of the FRAMES-HWIR Technology Software System

2.0 Requirements

Requirements are characteristics and behaviors that a piece of software must possess to function adequately for its intended purpose. The following sections describe the purpose and requirements for each of the two facilitating DLLs of the FRAMES-HWIR Technology Software System: HWIRIO.DDL and HWIRMC.DLL.

2.1 HWIR Input/Output DLL

The FRAMES-HWIR Technology Software System includes a number of processors. One of the processors, the Multimedia Multipathway Simulation Processor (MMSP), contains modules that interact to develop a complete picture of contaminant movement through the environment to the human and ecological receptors. Some of these modules are newly developed; others have existed for some time. To simplify the interface between existing and new software codes and to provide a storage format that is both flexible and capable of expanding to meet system needs, modules make use of the HWIRIO.DLL to facilitate input/output. In addition, the processors within the system use the HWIRIO.DLL to facilitate data transfer.

2.1.1 Input Requirements

The first step for input is when a subroutine within these processors and modules calls a subroutine in the HWIRIO.DLL. To facilitate these calls, the HWIRIO.DLL reads a specified formatted-American Standard Code for Information Exchange (ASCII) file. The term "formatted ASCII" refers to a text file that contains only 7-bit ASCII characters and uses only ASCII-standard control characters (that is, it has no embedded codes specific to a particular text-formatter markup language or output device and no meta-characters). Additional information on the formatted-ASCII file is presented in *Documentation for the FRAMES-HWIR Technology Software System: Volume 8, Specifications* (see Section 6.0).

The Site Simulation Files (SSFs) contain all necessary data required by the MMSP, including information describing module connectivity. For the SSFs, the HWIRIO.DLL recognizes only the variables in the following data groups:

- Header
- Site Layout
- Source In
- Air In
- Vadose Zone In
- Aquifer In
- Watershed In
- Surface Water In
- Terrestrial Foodchain In
- Aquatic Foodchain In
- Ecological Risk In
- Farm Foodchain In
- Human Exposure In
- Human Risk In

Specific variables in each of these data groups are defined in *Documentation for the FRAMES-HWIR Technology Software System: Volume 8, Specifications* (see Section 6.0).

2.1.2 Scientific Requirements

Development of the HWIRIO.DLL assumes that only two languages need to be supported: C++ and FORTRAN 77 (or later versions). Four compilers are also supported. The only limitation of the HWIRIO.DLL is that it must be a 32-bit DLL that must run under Windows® 95.

2.1.3 Output Requirements

A subroutine within the processors and modules provides input to a subroutine within the HWIRIO.DLL. To facilitate this input, the HWIRIO.DLL will output a formatted-ASCII file. The term "formatted ASCII" refers to a text file that contains only 7-bit ASCII characters and uses only ASCII-standard control characters (that is, has no embedded codes specific to a particular text-formatter markup language or output device, and no meta-characters). Additional information on the formatted-ASCII format can be found in *Documentation for the FRAMES-HWIR Technology Software System: Volume 8, Specifications* (see Section 6.0).

The Global Results Files (GRFs) are a collection of data groups populated by executing appropriate modules in the appropriate sequence in the MMSP and used to provide input to connecting modules. To facilitate output from the processors and modules, the HWIRIO.DLL recognizes only the variables in the following data groups for the GRF:

- Source Out
- Air Out
- Vadose Zone Out
- Aquifer Out
- Watershed Out
- Surface Water Out
- Terrestrial Foodchain Out
- Aquatic Foodchain Out
- Ecological Risk Out
- Farm Foodchain Out
- Human Exposure Out
- Human Risk Out

Specific variables in each of these data groups are defined in *Documentation for the FRAMES-HWIR Technology Software System: Volume 8, Specifications* (see Section 6.0).

In addition, the HWIRIO.DLL creates and manages the error and warning files. The error and warning files are free-format ASCII files. Free format means that no particular format is required for these ASCII files, as was required for the formatted-ASCII files previously defined. The HWIRIO.DLL echos all data written to these files back to the user of the FRAMES-HWIR Technology Software System through the System User Interface (SUI).

Errors are written to the error file. Error files are expected to be populated only when a critical error occurs. Critical errors are those errors that require the computation to be terminated. Examples include when a file or variable called for is not found in the location specified, or the variable does not match the one requested. The string description of the error that is written to this file should be detailed enough to allow the component developer to diagnose the problem, but at a minimum should include component name and subroutine name.

Warnings are written to the warning file. Warnings are descriptions of suspect calculations. The description of the warnings written to the file should be detailed enough to allow the component developer to diagnose whether an error has occurred.

2.2 HWIR Monte Carlo DLL

The FRAMES-HWIR Technology Software System includes a number of processors. One of the processors, the Site Definition Processor (SDP), interacts with the input databases to develop a complete set of parameters to make the SSF. Some of these parameters have distributions associated with them. The SDP uses the HWIRMC.DLL to sample from these distributions to determine a value for a given parameter. The CPP also uses the HWIRMC to generate some chemical parameter values. The HWIRMC was developed to ensure consistent sampling between the processors.

The core functionality of the HWIRMC.DLL comes from previously existing EPA subroutines. (The design and testing document was provided by EPA and TetraTech.) These subroutines were written in FORTRAN 77 and need to be modified only to the extent required for the HWIR software system. Since the HWIR software system uses a number of different compilers and languages (FORTRAN and C++), there was a need to have the legacy subroutines "wrapped" in a manner that is more acceptable to all languages and compilers. In particular, the legacy subroutine required arguments that were a real number with higher than one dimension. Arrays of this nature are difficult to pass between languages and compilers.

2.2.1 Input and Functionality Requirements

The "wrapper" for the legacy FORTRAN takes the approach of breaking the array arguments into a number of subroutines that change and allow the caller to set sections of the array. A clear example of this is the VarU array that the InitMC subroutine expects. The VarU array is used to define the distribution types and the parameters for each distribution. For example, an entry of VarU(1,1)=3.0, VarU(1,2)=1.0, VarU(1,3)=0.0, VarU(1,4)=6.0, and VarU(1,5)=1.0 indicates that the first distribution is a normal distribution. This is determined by VarU(1,5)=1.0. Array elements VarU(1,1) and VarU(1,2) indicate that the distribution has a mean of 3.0 and a standard deviation of 1.0. The distribution is limited to values between 0.0 and 1.0, which is indicated by elements VarU(1,3) and VarU(1,4).

A number of distributions and the associated parameters could be loaded in the VarU array using the manner described above. But the fact that VarU is a two-dimensional array will cause problems in the implementation across compilers. So rather than having the one InitMC subroutine, the "wrapper" has a subroutine named StatNormal.

StatNormal takes the mean, standard deviation, minimum, and maximum and sets the values in the current array element to define a normal distribution. For example, a call to StatNormal(3.0,1.0,0.0,6.0) will set the current array element in the VarU array to be a normal distribution with a mean of 3.0 and a standard deviation of 1.0 bounded between 0.0 and 6.0.

The StatNormal and other distribution subroutines automatically set the VarU(Current,5) array value to the appropriate distribution value, and they all increment the current variable. The current variable represents the current element in the VarU array that is being populated. Very similar behavior is used to populate the empirical distributions as well as the correlation matrix. The HWIR specification document, Section 2.0, describes the associated subroutines and function in more detail.

2.2.2 Scientific Requirements

Development of the HWIRMC.DLL assumes that only two languages need to be supported: C++ and FORTRAN 77 (or later versions). Four compilers are also supported. The only limitation of the HWIRMC.DLL is that it must be a 32-bit DLL that must run under Windows® 95.

2.2.1 Output Requirements

The only output requirement is that the value returned is in range and fits the distribution specified.

2.3 Future Directions for All Shared Subroutines

In the future, additional interpolation schemes may be provided, such as splines.

3.0 Design Elements

Design elements are strategies for meeting requirements. The facilitating DLLs are designed to meet the requirements identified in Section 2.0. Key to meeting those requirements is the interface between the various system components. The following sections describe the various design elements of the three facilitating DLLs.

3.1 HWIR Input/Output DLL

The design of the HWIRIO.DLL encompasses several subroutines, including those for initialization, input, output, and finalization. These subroutines are described in detail in *Documentation for the FRAMES-HWIR Technology Software System, Volume 8: Specifications* (see Section 6.0). In summary, the subroutines include the following:

- Subroutine OpenGroups—this subroutine is expected to be the first call in a program and to first retrieve the call arguments and open the warning and error files. If the warning and error files are opened successfully, all appropriate data groups are opened with the correct read/write mode.

- Subroutine CloseGroups—this subroutine is expected to be the last call in a program and to close all data group, warning, and error files. It then destroys the error file, signaling to the SUI that no error occurred in the component.

- Function ReadInt—this function returns the integer value for the given data group, variable name, and set of indexes.

- Function ReadReal—this function returns the real value for the given data group, variable name, and set of indexes.

- Function ReadLogical—this function returns the logical value for the given data group, variable name, and set of indexes.

- Subroutine ReadString—this subroutine returns for the variable *String*, the string value for the given data group, variable name, and set of indexes.

- Subroutine NumArgs—this subroutine returns the number of call arguments handed to a module.

- Function GetArgInt—this function returns the integer value for the given call-argument index.

- Subroutine GetArgString—this subroutine returns the string value for the call-argument index.

- Subroutine WriteInt—this subroutine stores the given integer value in the given data group, variable name, and set of indexes.

- Subroutine WriteReal—this subroutine stores the given real value in the given data group, variable name, and set of indexes.

- Subroutine WriteLog—this subroutine stores the given logical value in the given data group, variable name, and set of indexes.

- Subroutine WriteString—this subroutine stores a string of up to 80 character values in the given data group, variable name, and set of indexes.

- Subroutine Error—this subroutine will add the string value passed by the calling program to the error file and then terminate the calling program.

- Subroutine Warning—this subroutine will add the string value passed by the calling program to the warning file and then return control to the calling program.

In addition, the HWIRIO.DLL will test for consistency between the specifications and the calling program. For example, if a FORTRAN program module contained the following line of code:

MyInts(i,j,k)=ReadInt3("Source_SSF," "XYZ," "cm/s,"i,j,k)

the DLL could make the following tests for consistency with the specified information in the data table:

1) Does the data group "Source_SSF" exist? If not, raise an error.
2) Does the variable "XYZ" exist in "Source_SSF"? If not, raise an error.
3) Are any of the indexes negative? If so, raise an error.
4) Do the specified units for "XYZ" in "Source_SSF" match the units passed by the calling program? If not, raise an error.
5) Does the specified dimensionality of "XYZ" match the calling program's assumptions of dimensionality? If not, raise an error.
6) Is any one of the indexes larger than the maximum indexes stored in the file? If so, raise a warning and return a 0.
7) Is the value stored in the data file outside the specified bounds for that variable? If so, raise an error.
8) Does the specified type of the variable match the assumed type from the calling program? If not, raise an error.

3.2 HWIR Monte Carlo DLL

The design of the HWIRMC.DLL encompasses several subroutines, including those for initialization, input, output, and finalization. These subroutines are described in detail in *Documentation for the FRAMES-HWIR Technology Software System, Volume 8: Specifications* (see Section 2.0). In summary, the subroutines include the following:

This Dynamic Link Library contains the core set of Monte Carlo functions and subroutines for the HWIR Software System. Its intended to be used to perform all sampling, from parameterized distributions, for the DSP and SDP. Typically, the calling program will make use of the NumDist and NumCor subroutines to inform the HWIRMC.DLL as to the number of distributions and correlations. Then the calling program will make use of the various distribution definition functions to set the

parameters for each distribution. The Cor subroutine is then used to inform the HWIRMC.DLL as to the correlations between the specified distributions. Then the sample subroutine is called to perform the actual sampling.

- Subroutine StatNumDist—This subroutine sets the number of distributions that will have values sampled from them.

- Subroutine StatNumCor—This subroutine sets the number of correlations between the given distributions that will have values sampled from them.

- Subroutine StatDebugOn—This subroutine will turn on debug messages in the HWIRMC.DLL.

- Subroutine StatSeed—This subroutine sets the seed to be used by the HWIRMC.DLL. This allows for a reproducible set of samples from the given set of distributions.

- Subroutine StatClear—This subroutine deallocates any memory resources used by the HWIRMC.dll.

- Subroutine StatCor—This subroutine sets a correlation between two distributions.

- Subroutine StatSample—This subroutine performs the sampling from the given set of distributions.

- Distribution Definition Functions—In general, the Distribution Definition Functions provide the HWIRMC.dll with the parameters for a specific distribution type. For example, the Normal() function sets the mean, standard deviation, minimum, and maximum for a normal distribution. The distribution types available in the HWIRMC.DLL are Normal, LogNormal, JohnsonSB, Transformed LogNormal, Exponential, Triangular, Uniform, Integer Uniform, SB, SU, Gamma, Weibull, and Empirical.

The HWIRMC.DLL also optimizes the use of the sampling subroutines within it. For example, if a set of distributions contains few if any correlations, the HWIRMC.DLL will break the set of distributions into uncorrelated subsets. This is simply done by ensuring that if, for example, distribution A is correlated to distribution B, then they will be assigned to the same subset. But if distribution A is uncorrelated with distribution B, they will be assigned to different subsets. When the StatSample routine is executed, this divides the distribution set into subsets. This increases the efficiency of the HWIRMC.DLL because many times a set of distributions contains a number of independent subsets, but the underlying sample methodology assumes that all distributions in a set might be correlated. Therefore, the number of samples from the distributions that are rejected will be less with the subsets than with the set of distributions as a whole.

4.0 Testing Approach and Results

As noted, a DLL provides a convenient mechanism for sharing subroutines and functions because they dynamically link to their calling programs at run time rather than at the time of compilation. This linking allows a DLL to be compiled and tested separately from the program that uses it. The following sections describe how the three facilitating DLLs were tested.

4.1 HWIR Input/Output DLL

The HWIRIO.DLL was tested to ensure it met its requirements, as described in Section 2.0. The test cases described in Section 4.1.1 address the basic functionality listed in the requirements (black box testing). The DLL was tested in depth because it is used by many programs in the FRAMES-HWIR Technology Software System.

Requirements for the HWIRIO.DLL are described in Section 2.0. These requirements were reworded for Table 4.1 to list the concise, fundamental requirements suitable for testing.

Table 4.1 Fundamental Testing Requirements for the HWIRIO.DLL

Requirement Number	Requirement
1	Correctly read data (all strings, logicals, integers, and floats having from 0 to 6 dimensions) from the format required for the Site Definition Files (SDF)/SSF and the GRF
2	Correctly write data (all strings, logicals, integers, and floats having from 0 to 6 dimensions) to the format required for the SDF/SSF and the GRF
3	Recognize only the variables listed in a data group dictionary (*.dic)
4	Correctly access arguments passed from the calling program
5	Create an error file (ASCII)
6	Create a warning file (ASCII)
7	Write an error message to the error file and halt program execution when a designated error condition occurs (for instance, the program attempts to access a data group or variable that is undefined in a data dictionary, data accessed is outside the range allowed for that variable as defined in the data dictionary, and so forth)
8	Write a warning message to the warning file when a designated warning condition occurs (example, calling subroutine OpenGroups twice before calling CloseGroups)
9	Destroy the error file
10	Destroy the warning file
11	Support the Borland C++ Version 4.0 compiler
12	Support the Microsoft® Visual C++ Version 5.0 compiler
13	Support the Lahey FORTRAN-90 Version 4.0 compiler
14	Support the Digital Visual FORTRAN-90 Version 5.0 compiler
15	Be a 32-bit DLL
16	Run under Windows® 95

The formats of the SDF/SSF and GRF, along with the defined data groups in these files and variables within those groups, are listed in *Documentation for the FRAMES-HWIR Technology Software System, Volume 8: Specifications* (see Section 6.0 References). Table 4.2 shows the relationship between these requirements and the test cases described in Section 4.1.1. Requirements 11 through 16 are omitted from this table because they will be addressed by all the test cases.

Table 4.2 Test Cases Related to Requirements for the HWIRIO.DLL

		Test Case Number												
		1	2	3	4	5	6	7	8	9	10	11	12	13
Requirement	1		X			X		X						
	2			X	X									
	3													
	4	X												
	5		X											
	6		X											
	7						X		X	X	X	X	X	X
	8													
	9			X										
	10			X										

		Test Case Number												
		14	15	16	17	18	19	20	21	22	23	24	25	26
Requirement	1													X
	2													
	3												X	
	4													
	5				X	X	X							
	6													
	7	X	X		X		X	X	X	X	X	X	X	
	8			X										
	9													
	10													

DLLs are called by programs. Thus, to test the HWIRIO.DLL, a testing program was used that performed subroutine and function calls to the DLL, as specified in a test-specific input file (testname.tst). Appendix A

- documents the general procedure for conducting each test along with the format of the test-specific input file
- lists the FORTRAN-90 version of the test program along with the "include files" necessary for both the Lahey FORTRAN-90 4.0 and Digital Visual FORTRAN-90 5.0 compilers
- lists the C++ version along with the "include files" necessary for both the Borland C++ 4.0 and Microsoft® Visual C++ 5.0 compilers

- lists an example SSF along with its data dictionary
- lists an example GRF, also with its data dictionary.

All tests were conducted under Windows® 95 because this is the only operating system required to be supported. In addition, all tests were conducted with Borland C++ 4.0, Microsoft® Visual C++ 5.0, Lahey FORTRAN-90 4.0, and Digital Visual FORTRAN-90 5.0 compiled versions of the testing program. All of the following tests were successfully executed using the HWIRIO.DLL.

4.1.1 IODL_1

4.1.1.1 Description and Rationale

This case verifies that the NumArgs, GetArgInt, and GetArgString functions of the DLL are operating correctly. These functions are used by the calling program to determine the location and names of the data group files to read and a data group file to write. This information is passed to the calling program using an environment variable, called ARGUMENTS. Function NumArgs returns the number of arguments in the environment variable. GetArgInt gets a specified integer argument from the environment variable, and GetArgString gets a specified string argument.

4.1.1.2 Input Data

The input file to the test program follows. NumArgs, GetArgInt, and GetArgString are called to verify that these functions work properly. After each call, a comment stating the expected result is included (comments occur after the semicolon). Calls to OpenGroups and CloseGroups are included; however, they are not required in order to call these three functions.

IODL_1.TST

```
“OpenGroups”
“NumArgs”   ; should return 7
“GetArgString”
  1         ; should return 1:05pm
“GetArgString”
  2         ; should return 03/30/1998
“GetArgInt”
  5         ; should return 1
“GetArgString”
  6         ; should return hd.ssf
“GetArgString”
  7         ; should return hdjunk.grf
“CloseGroups”
“Stop”
```

The batch file that runs the C++ version of the test program and sets the arguments is as follows:

```
echo The call arguments are:
```

```
set ARGUMENTS=1:05pm 03/30/1998 \HWIR98\IODLL\SSF \HWIR98\IODLL\GRF 1 hd.ssf
hdjunk.grf[2]
echo %ARGUMENTS%
del %1c.out
del %1.out
call cdlltst %1
```

4.1.1.3 Expected Results

Functions NumArgs, GetArgInt, and GetArgString are expected to execute without error and return the values listed in the IODL_1.TST file.

4.1.1.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_1" and press return. The program runs and creates an output file called "iodl_1c.out," located in the SSF directory. As shown, this file lists the functions called and the results returned. The results returned can be compared to the expected result comments in the "iodl_1.tst" file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using "runmscp.bat." The output file is called "iodl_1m.out." To run the Lahey Fortran-90 test program, repeat the previous steps using "runlf.bat." The output file is called "iodl_1l.out." To run the Digital Visual Fortran-90 test program, repeat the same steps using "rundv.bat." The output file is called "iodl_1d.out."

IODL_1C.OUT

```
Call OpenGroups
 Call NumArgs
 Returned   7
 Call GetArgString
 Returned    1:05pm
 Call GetArgString
 Returned    03/30/1998
 Call GetArgInt
 Returned    1
 Call GetArgString
 Returned    hd.ssf
 Call GetArgString
 Returned    hdjunk.grf
    Call CloseGroups
```

[2] This line (line #3) is actually a continuation of the previous line (line #2) that has wrapped around.

4.1.2 IODL_2

4.1.2.1 Description and Rationale

This simple case verifies that the DLL can correctly read from the format required for the SDF/SSF and GRF. The example SSF File (HD.SSF) is used to verify that read operations can be performed successfully for all four data types (string, logical, integer, and float) from 0 to 6 dimensions. In addition, the error and warning files are created.

4.1.2.2 Input Data

The input file to the testing program is as follows:

IODL_2.TST

```
“OpenGroups”
“ReadInt”
“hd.ssf”,“Int”,“days”, ; should return 20
“ReadInt1”
“hd.ssf”,“Int1”,“days”,1, ; should return 1
“ReadInt2"
“hd.ssf”,“Int2”,“days”,2,6, ; should return 11
“ReadInt3”
“hd.ssf”,“Int3”,“days”,2,2,5, ; should return 20
“ReadInt4”
“hd.ssf”,“Int4”,“days”,2,2,2,5, ; should return 40
“ReadInt5”
“hd.ssf”,“Int5”,“days”,2,2,2,2,5, ; should return 80
“ReadInt6”
“hd.ssf”,“Int6”,“days”,2,2,2,2,2,5, ; should return 160
“ReadReal1”
“hd.ssf”,“Real1”,“days”,5, ; should return 5.1
“ReadReal2”
“hd.ssf”,“Real2”,“days”,2,6, ; should return 11.1
“ReadReal3”
“hd.ssf”,“Real3”,“days”,2,2,5, ; should return 20.1
“ReadReal4”
“hd.ssf”,“Real4”,“days”,2,2,2,5, ; should return 40.1
“ReadReal5”
“hd.ssf”,“Real5”,“days”,2,2,2,2,5, ; should return 80.1
“ReadReal6”
“hd.ssf”,“Real6”,“days”,2,2,2,2,2,5, ; should return 160.1
“ReadLog”
“hd.ssf”,“Log”, ; should return T
“ReadLog1”
“hd.ssf”,“Log1”,“”,5, ; should return T
“ReadLog2”
“hd.ssf”,“Log2”,“”,2,6, ; should return F
“ReadLog3”
“hd.ssf”,“Log3”,“”,2,2,5, ; should return T
```

```
“ReadLog4”
“hd.ssf”,“Log4”,“”,2,2,2,4, ; should return F
“ReadLog5”
“hd.ssf”,“Log5”,“”,2,2,2,2,5 ; should return T
“ReadLog6”
“hd.ssf”,“Log6”,“”,2,2,2,2,2,4, ; should return F
“ReadString”
“hd.ssf”,“String”,“”, ; should return A
“ReadString1”
“hd.ssf”,“String1”,“”,5, ; should return E
“ReadString2”
“hd.ssf”,“String2”,“”,2,5 ; should return J
“ReadString3”
“hd.ssf”,“String3”,“”,2,2,5, ; should return T
“ReadString4”
“hd.ssf”,“String4”,“”,2,2,2,5, ; should return NN
“ReadString5”
“hd.ssf”,“String5”,“”,2,2,2,2,5, ; should return DDDD
“ReadString6”
“hd.ssf”,“String6”,“”,2,2,2,2,2,5, ; should return AAAAAAA
“Stop”
```

The subroutine CloseGroups is omitted to test for the creation of the error and warning files (CloseGroups would delete these files).

4.1.2.3 Expected Results

The read functions called are expected to return the values listed in the IODL_2.TST File. It is also expected that error and warning files will be created by the OpenGroups subroutine, which can be verified by a directory listing.

4.1.2.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_2” and press return. The program runs and creates an output file called “iodl_2c.out,” located in the SSF directory. As shown following, this file lists the functions called and the results returned. The results returned can be compared to the expected result comments in the “iodl_2.tst” file. Note that in C++, logical data are represented by 1s and 0s. A “T” is represented by a “1” and an “F” by a “0.”

To run the Microsoft® Visual C++ test program, repeat these steps using “runmscp.bat.” The output file is called “iodl_2m.out.” To run the Lahey Fortran-90 test program, repeat the same steps using “runlf.bat.” The output file is called “iodl_2l.out.” To run the Digital Visual Fortran-90 test program, repeat the steps using “rundv.bat.” The output file is called “iodl_2d.out.”

IODL_2C.OUT

```
Call OpenGroups
 Call ReadInt    hd.ssf   Int   days
 Returned   20
```

```
Call ReadInt1    hd.ssf   Int1   days  1
Returned   1
Call ReadInt2    hd.ssf   Int2   days  2 6
Returned   11
Call ReadInt3    hd.ssf   Int3   days  2 2 5
Returned   20
Call ReadInt4    hd.ssf   Int4   days  2 2 2 5
Returned   40
Call ReadInt5    hd.ssf   Int5   days  2 2 2 2 5
Returned   80
Call ReadInt6    hd.ssf   Int6   days  2 2 2 2 2 5
Returned   160
Call ReadReal1    hd.ssf   Real1   days  5
Returned   5.1
Call ReadReal2    hd.ssf   Real2   days  2 6
Returned   11.1
Call ReadReal3    hd.ssf   Real3   days  2 2 5
Returned   20.1
Call ReadReal4    hd.ssf   Real4   days  2 2 2 5
Returned   40.1
Call ReadReal5    hd.ssf   Real5   days  2 2 2 2 5
Returned   80.1
Call ReadReal6    hd.ssf   Real6   days  2 2 2 2 2 5
Returned   160.1
Call ReadLog    hd.ssf   Log
Returned   1
Call ReadLog1    hd.ssf   Log1     5
Returned   1
Call ReadLog2    hd.ssf   Log2     2 6
Returned   0
Call ReadLog3    hd.ssf   Log3     2 2 5
Returned   1
Call ReadLog4    hd.ssf   Log4     2 2 2 4
Returned   0
Call ReadLog5    hd.ssf   Log5     2 2 2 2 5
Returned   1
Call ReadLog6    hd.ssf   Log6     2 2 2 2 2 4
Returned   0
Call ReadString    hd.ssf   String
Returned    A
Call ReadString1    hd.ssf   String1     5
Returned    E
Call ReadString2    hd.ssf   String2     2 5
Returned    J
Call ReadString3    hd.ssf   String3     2 2 5
Returned    T
Call ReadString4    hd.ssf   String4     2 2 2 5
Returned    NN
Call ReadString5    hd.ssf   String5     2 2 2 2 5
Returned    DDDD
Call ReadString6    hd.ssf   String6     2 2 2 2 2 5
Returned    AAAAAAA
```

4.1.3 IODL_3

4.1.3.1 Description and Rationale

This case verifies that the DLL can correctly write to the format required for the SDF/SSF and GRF. A test-specific GRF (HD.GRF) is created by the testing program from data listed in the test input file (IODL_3.TST). Write operations for all four data types (string, logical, integer, and float) from 0 to 6 dimensions are performed. To verify that error and warning files are properly deleted at the end of a run (assuming no error or warnings occur), these files are created at the beginning of the test (by subroutine OpenGroups) and deleted at the end of the test (by subroutine CloseGroups).

4.1.3.2 Input Data

The input file to the testing program is as follows:

IODL_3.TST

```
“OpenGroups”
“WriteInt”
“hd.grf”,“Int”,“days”,11, ; should write 11
“WriteInt1”
“hd.grf”,“Int1”,“days”,1,22, ; should Write 22
“WriteInt2”
“hd.grf”,“Int2”,“days”,2,6,33, ; should Write 33
“WriteInt3”
“hd.grf”,“Int3”,“days”,2,2,5,44, ; should Write 44
“WriteInt4”
“hd.grf”,“Int4”,“days”,2,2,2,5,55, ; should Write 55
“WriteInt5”
“hd.grf”,“Int5”,“days”,2,2,2,2,5,66, ; should Write 66
“WriteInt6”
“hd.grf”,“Int6”,“days”,2,2,2,2,2,5,77, ; should Write 77
“WriteReal”
“hd.grf”,“Real”,“days”,11.1, ; should Write 11.1
“WriteReal1”
“hd.grf”,“Real1”,“days”,5,22.2, ; should Write 22.2
“WriteReal2”
“hd.grf”,“Real2”,“days”,2,6,33.3, ; should Write 33.3
“WriteReal3”
“hd.grf”,“Real3”,“days”,2,2,5,44.4, ; should Write 44.4
“WriteReal4”
“hd.grf”,“Real4”,“days”,2,2,2,5,55.5, ; should Write 55.5
“WriteReal5”
“hd.grf”,“Real5”,“days”,2,2,2,2,5,66.6, ; should Write 66.6
“WriteReal6”
“hd.grf”,“Real6”,“days”,2,2,2,2,2,5,77.7, ; should Write 77.7
“WriteLog”
“hd.grf”,“Log”,“”,T, ; should Write T
“WriteLog1”
```

```
“hd.grf”,“Log1”,“”,5,T, ; should Write T
“WriteLog2”
“hd.grf”,“Log2”,“”,2,6,F, ; should Write F
“WriteLog3”
“hd.grf”,“Log3”,“”,2,2,5,T, ; should Write T
“WriteLog4”
“hd.grf”,“Log4”,“”,2,2,2,4,F, ; should Write F
“WriteLog5”
“hd.grf”,“Log5”,“”,2,2,2,2,5,T, ; should Write T
“WriteLog6”
“hd.grf”,“Log6”,“”,2,2,2,2,2,4,F, ; should Write F
“WriteString”,
“hd.grf”,“String”,“”,“A1”, ; should Write A1
“WriteString1”
“hd.grf”,“String1”,“”,5,“E2”, ; should Write E2
“WriteString2”
“hd.grf”,“String2”,“”,2,5,“J3”, ; should Write J3
“WriteString3”
“hd.grf”,“String3”,“”,2,2,5,“T4”, ; should Write T4
“WriteString4”
“hd.grf”,“String4”,“”,2,2,2,5,“NN5”, ; should Write NN5
“WriteString5”
“hd.grf”,“String5”,“”,2,2,2,2,5,“DDDD6”, ; should Write DDDD6
“WriteString6”
“hd.grf”,“String6”,“”,2,2,2,2,2,5,“AAAAAAA7”, ; should Write AAAAAAA7
“CloseGroups”
“Stop”
```

4.1.3.3 Expected Results

It is expected that data listed in the test-specific GRF (HD.GRD) created by the testing program exactly match the corresponding data in the test program input file (IODL_3.TST). It is further expected that the format of the test-specific GRF (HD.GRF) meets the required specifications for this file and that error and warning files are not present (that is, they are properly deleted). The format of the GRF can be verified by visual inspection. However, the following test case (IODL_4) verifies that the format of the newly created GRF is correct by actually reading this file.

4.1.3.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_3" and press return. The program runs and creates an output file called "iodl_3c.out," located in the SSF directory, and an output file called "iodl_3c.grf," located in the GRF directory. The "iodl_3c.out" file lists the functions called and the results returned. The "iodl_3c.grf" file is in the GRF format that should be readable by the DLL. Both files are shown in the example that follows. The results returned can be compared to the expected result comments in the "iodl_3.tst" file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using "runmscp.bat." The output file is called "iodl_3m.out." To run the Lahey Fortran-90 test program, repeat these same steps using "runlf.bat." The output file is called "iodl_3l.out." To run the Digital Visual Fortran-90 test program, repeat these steps using "rundv.bat." The output file is called "iodl_3d.out."

The created GRF file is visually examined to ensure that the format is correct. To further check the format, the next case (IODL_4.TST) attempts to read the GRF file created (IODL_3C.GRF). If the GRF file can be read by the DLL, the format is correct.

IODL_3C.OUT

```
Call OpenGroups
 Call WriteInt    hd.grf   Int   days  11
 Call WriteInt1    hd.grf   Int1   days  1 22
 Call WriteInt2    hd.grf   Int2   days  2 6 33
 Call WriteInt3    hd.grf   Int3   days  2 2 5 44
 Call WriteInt4    hd.grf   Int4   days  2 2 2 5 55
 Call WriteInt5    hd.grf   Int5   days  2 2 2 2 5 66
 Call WriteInt6    hd.grf   Int6   days  2 2 2 2 2 5 77
 Call WriteReal    hd.grf   Real   days  11.1
 Call WriteReal1    hd.grf   Real1   days  5 22.2
 Call WriteReal2    hd.grf   Real2   days  2 6 33.3
 Call WriteReal3    hd.grf   Real3   days  2 2 5 44.4
 Call WriteReal4    hd.grf   Real4   days  2 2 2 5 55.5
 Call WriteReal5    hd.grf   Real5   days  2 2 2 2 5 66.6
 Call WriteReal6    hd.grf   Real6   days  2 2 2 2 2 5 77.7
 Call WriteLog    hd.grf   Log      T
 Call WriteLog1    hd.grf   Log1     5  T
 Call WriteLog2    hd.grf   Log2     2 6  F
 Call WriteLog3    hd.grf   Log3     2 2 5  T
 Call WriteLog4    hd.grf   Log4     2 2 2 4  F
 Call WriteLog5    hd.grf   Log5     2 2 2 2 5  T
 Call WriteLog6    hd.grf   Log6     2 2 2 2 2 4  F
 Call WriteString    hd.grf   String      A1
 Call WriteString1    hd.grf   String1     5  E2
 Call WriteString2    hd.grf   String2     2 5  J3
 Call WriteString3    hd.grf   String3     2 2 5  T4
 Call WriteStrin4    hd.grf   String4     2 2 2 5  NN5
 Call WriteString5    hd.grf   String5     2 2 2 2 5  DDDD6
 Call WriteString6    hd.grf   String6     2 2 2 2 2 5  AAAAAAA7
 Call CloseGroups
```

IODL_3C.GRF

```
1,
“hd.grf”,“data group”,
28,
“Int”,0, “INTEGER”,0, “days”,
11,
“Int1",1, “INTEGER”,0, "days",
1, 22,
“Int2”,2, “INTEGER”,0, “days”,
2,
0,
6, 0, 0, 0, 0, 0, 33,
“Int3”,3, “INTEGER”,0, “days”,
2, 0,
2,
0,
5, 0, 0, 0, 0, 44,
“Int4”,4, “INTEGER”,0, “days”,
2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 55,
“Int5”,5, “INTEGER”,0, “days”,
2, 0, 2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 66,
“Int6”,6, “INTEGER”,0, “days”,
2, 0, 2, 0, 2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 77,
“Real”,0, “FLOAT”,0, “days”,
11.1,
“Real1”,1, “FLOAT”,0, “days”,
5, 0, 0, 0, 0, 22.2,
“Real2”,2, “FLOAT”,0, “days”,
2,
0,
6, 0, 0, 0, 0, 0, 33.3,
“Real3”,3, “FLOAT”,0, “days”,
2, 0,
2,
0,
5, 0, 0, 0, 0, 44.4,
“Real4”,4, “FLOAT”,0, “days”,
2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 55.5,
```

```
“Real5”,5, “FLOAT”,0, “days”,
2, 0, 2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 66.6,
“Real6”,6, “FLOAT”,0, “days”,
2, 0, 2, 0, 2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 77.7,
“Log”,0, “LOGICAL”,0, “”,
“T”
“Log1”,1, “LOGICAL”,0, “”,
5, “F”,“F”,“F”,“F”,“T”,
“Log2”,2, “LOGICAL”,0, “”,
2,
0,
6, “F”,“F”,“F”,“F”,“F”,“F”,
“Log3”,3, “LOGICAL”,0, “”,
2, 0,
2,
0,
5, “F”,“F”,“F”,“F”,“T”,
“Log4”,4, “LOGICAL”,0, “”,
2, 0, 2, 0,
2,
0,
4, “F”,“F”,“F”,“F”,
“Log5”,5, “LOGICAL”,0, “”,
2, 0, 2, 0, 2, 0,
2,
0,
5, “F”,“F”,“F”,“F”,“T”,
“Log6”,6, “LOGICAL”,0, “”,
2, 0, 2, 0, 2, 0, 2, 0,
2,
0,
4, “F”,“F”,“F”,“F.”
“String”,0, “STRING”,0, “”,
“A1”,
“String1”,1, “STRING”,0, “”,
5, “”,“”,“”,“”, “E2”,
“String2”,2, “STRING”,0, “”,
2,
0,
5, “”,“”,“”,“”,“J3”,
“String3”,3, “STRING”,0, “”,
2, 0,
2,
0,
5, “”,“”,“”,“”,“T4”,
```

```
“String4”,4, “STRING”,0, “”,
2, 0, 2, 0,
2,
0,
5, “”,“”,“”,“”,“NN5”,
“String5”,5, “STRING”,0, “”,
2, 0, 2, 0, 2, 0,
2,
0,
5, “”,“”,“”,“”,“DDDD6”,
“String6”,6, “STRING”,0, “”,
2, 0, 2, 0, 2, 0, 2, 0,
2,
0,
5, “”,“”,“”,“”,“AAAAAAA7”
```

4.1.4 IODL_4

4.1.4.1 Description and Rationale

Test case IODL_2 was established so the DLL could correctly read from an SSF/GRF. The previous test case (IODL_3) was tasked with creating a GRF, using the write functions in the DLL. Visual inspection was used to verify that the format of this GRF is correct, but a more rigorous check is to verify that this GRF can be read using the DLL. This case was designed to further verify that the previous case was executed correctly.

4.1.4.2 Input Data

The input file to the testing program is as follows:

IODL_4.TST

```
“OpenGroups”
“ReadInt”
“hd.grf”,“Int”,“days”, ; should Read 11
“ReadInt1”
“hd.grf”,“Int1”,“days”,1, ; should Read 22
“ReadInt2”
“hd.grf”,“Int2”,“days”,2,6, ; should Read 33
“ReadInt3”
“hd.grf”,“Int3”,“days”,2,2,5, ; should Read 44
“ReadInt4”
“hd.grf”,“Int4”,“days”,2,2,2,5, ; should Read 55
“ReadInt5”
“hd.grf”,“Int5”,“days”,2,2,2,2,5, ; should Read 66
“ReadInt6”
“hd.grf”,“Int6”,“days”,2,2,2,2,2,5, ; should Read 77
“ReadReal”,
“hd.grf”,“Real”,“days”, ; should Read 11.1
```

```
“ReadReal1”,
“hd.grf”,“Real1”,“days”,5, ; should Read 22.2
“ReadReal2”,
“hd.grf”,“Real2”,“days”,2,6, ; should Read 33.3
“ReadReal3”,
“hd.grf”,“Real3”,“days”,2,2,5, ; should Read 44.4
“ReadReal4”,
“hd.grf”,“Real4”,“days”,2,2,2,5, ; should Read 55.5
“ReadReal5”,
“hd.grf”,“Real5”,“days”,2,2,2,2,5, ; should Read 66.6
“ReadReal6”,
“hd.grf”,“Real6”,“days”,2,2,2,2,2,5, ; should Read 77.7
“ReadLog”,
“hd.grf”,“Log”,“”, ; should Read T
“ReadLog1”,
“hd.grf”,“Log1”,“”,5, ; should Read T
“ReadLog2”,
“hd.grf”,“Log2”,“”,2,6, ; should Read F
“ReadLog3”,
“hd.grf”,“Log3”,“”,2,2,5, ; should Read T
“ReadLog4”,
“hd.grf”,“Log4”,“”,2,2,2,4, ; should Read F
“ReadLog5”,
“hd.grf”,“Log5”,“”,2,2,2,2,5, ; should Read T
“ReadLog6”,
“hd.grf”,“Log6”,“”,2,2,2,2,2,4, ; should Read F
“ReadString”,
“hd.grf”,“String”,“”, ; should Read A1
“ReadString1”,
“hd.grf”,“String1”,“”,5, ; should Read E2
“ReadString2”,
“hd.grf”,“String2”,“”,2,5, ; should Read J3
“ReadString3”,
“hd.grf”,“String3”,“”,2,2,5, ; should Read T4
“ReadString4”,
“hd.grf”,“String4”,“”,2,2,2,5, ; should Read NN5
“ReadString5”,
“hd.grf”,“String5”,“”,2,2,2,2,5, ; should Read DDDD6
“ReadString6”,
“hd.grf”,“String6”,“”,2,2,2,2,2,5, ; should Read AAAAAAA7
“CloseGroups”
“Stop”
```

4.1.4.3 Expected Results

It is expected that the data listed in the test output file (IODL_4.OUT) will exactly match all the data in the GRF created in the previous test.

4.1.4.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_4" and press return. The program runs and creates an output file called "iodl_4c.out," located in the SSF directory, and an output file called "hdjunk.grf," located in the GRF directory. The "iodl_4c.out" file lists the functions called and the results returned. The "hdjunk.grf" file is empty because no write calls are made. The "iodl_4c.out" file is shown as follows. The results returned can be compared to the expected result comments in the iodl_4.tst file. Note that in C++, logical data are represented by 1s and 0s. A "T" is represented by a "1" and an "F" by a "0."

To run the Microsoft® Visual C++ test program, repeat the previous steps using "runmscp.bat." The output file is called "iodl_4m.out." To run the Lahey Fortran-90 test program, repeat these steps using "runlf.bat." The output file is called "iodl_4l.out." To run the Digital Visual Fortran-90 test program, repeat the same steps using "rundv.bat." The output file is called "iodl_4d.out."

IODL_4C.OUT

```
Call OpenGroups
 Call ReadInt    hd.grf   Int   days
 Returned   11
 Call ReadInt1    hd.grf   Int1   days  1
 Returned   22
 Call ReadInt2    hd.grf   Int2   days  2 6
 Returned   33
 Call ReadInt3    hd.grf   Int3   days  2 2 5
 Returned   44
 Call ReadInt4    hd.grf   Int4   days  2 2 2 5
 Returned   55
 Call ReadInt5    hd.grf   Int5   days  2 2 2 2 5
 Returned   66
 Call ReadInt6    hd.grf   Int6   days  2 2 2 2 2 5
 Returned   77
 Call ReadReal    hd.grf   Real   days
 Returned   11.1
 Call ReadReal1    hd.grf   Real1   days  5
 Returned   22.2
 Call ReadReal2    hd.grf   Real2   days  2 6
 Returned   33.3
 Call ReadReal3    hd.grf   Real3   days  2 2 5
 Returned   44.4
 Call ReadReal4    hd.grf   Real4   days  2 2 2 5
 Returned   55.5
 Call ReadReal5    hd.grf   Real5   days  2 2 2 2 5
 Returned   66.6
 Call ReadReal6    hd.grf   Real6   days  2 2 2 2 2 5
 Returned   77.7
 Call ReadLog    hd.grf   Log
 Returned   1
 Call ReadLog1    hd.grf   Log1     5
```

```
Returned   1
Call ReadLog2    hd.grf   Log2     2 6
Returned   0
Call ReadLog3    hd.grf   Log3     2 2 5
Returned   1
Call ReadLog4    hd.grf   Log4     2 2 2 4
Returned   0
Call ReadLog5    hd.grf   Log5     2 2 2 2 5
Returned   1
Call ReadLog6    hd.grf   Log6     2 2 2 2 2 4
Returned   0
Call ReadString    hd.grf   String
Returned    A1
Call ReadString1    hd.grf   String1     5
Returned    E2
Call ReadString2    hd.grf   String2     2 5
Returned    J3
Call ReadString3    hd.grf   String3     2 2 5
Returned    T4
Call ReadString4    hd.grf   String4     2 2 2 5
Returned    NN5
Call ReadString5    hd.grf   String5     2 2 2 2 5
Returned    DDDD6
Call ReadString6    hd.grf   String6     2 2 2 2 2 5
Returned    AAAAAAA7
Call CloseGroups
```

4.1.5 IODL_5

4.1.5.1 Description and Rationale

This case verifies that the DLL can accurately locate and read parameters from a very large (6.9 MB) data group file. The DLL reads from the SSF file "SW.SSF." This file is too large to be included in this test plan, but it is available electronically. Only a sampling of reads is performed. It is assumed that if one or two parameters can be read from this file without error, the DLL should be able to read all the parameters in this file.

4.1.5.2 Input Data

The input file to the testing program is as follows:

```
"OpenGroups"
"ReadReal1",
"sw.ssf","sw_var100","m",10, ; should return 6.00000001
"ReadReal4",
"sw.ssf","sw_var53","g/day",1,1,1,4, ; should return 12.4
"CloseGroups"
"Stop"
```

4.1.5.3 Expected Results

The data listed in the test output file (IODL_5.OUT) are expected to exactly match the corresponding data in SW.SSF.

4.1.5.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_5" and press return. The program runs and creates an output file called "iodl_5c.out," located in the SSF directory, and an output file called "hdjunk.grf," located in the GRF directory. The "iodl_5c.out" file lists the functions called and the results returned. The "hdjunk.grf" file is empty because no write calls are made. The "iodl_5c.out" file is shown as follows. The results returned can be compared to the expected results comment in the "iodl_5.tst" file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using "runmscp.bat." The output file is called iodl_5m.out. To run the Lahey Fortran-90 test program, repeat these same steps using "runlf.bat." The output file is called "iodl_5l.out." To run the Digital Visual Fortran-90 test program, repeat these steps using "rundv.bat." The output file is called "iodl_5d.out."

IODL_5C.OUT

```
Call OpenGroups
 Call ReadReal1    sw.ssf   sw_var100   m  10
 Returned   6.000000001
 Call ReadReal4    sw.ssf   sw_var53   g/day  1 1 1 4
 Returned   12.4
 Call CloseGroups
```

4.1.6 IODL_6

4.1.6.1 Description and Rationale

This case verifies that the DLL does not allow the user to write to a data group file that it is reading from if the file is not the last file listed in the ARGUMENTS environment variable. Of the files listed in ARGUMENTS, all files can be read from, but only the last file can be written to. The file HD.GRF is read from and an attempt is made to write to it also, even though it is not the last listed file in ARGUMENTS.

4.1.6.2 Input Data

The input file to the testing program is as follows:

IODL_6.TST

```
"OpenGroups"
"ReadInt"
"hd.grf","Int","days", ; should return 11
"WriteInt1"
```

```
"hd.grf","Int1","days",1,25, ; error - writing to an input file
"CloseGroups"
"Stop"
```

The batch file for the C++ version of the test program is as follows, showing that HD.GRF is not the last data group file in ARGUMENTS:

IODL_6C.BAT

```
echo The call arguments are
set ARGUMENTS=1:05pm 03/30/1998 \HWIR98\IODLL\SSF \HWIR98\IODLL\GRF 1 hd.grf
hd_junk.grf[3]
echo %ARGUMENTS%
del %1c.out
del %1.out
call cdlltst %1
copy %1.out %1c.out.
```

4.1.6.3 Expected Results

The read function is expected to execute properly, but the write function fails, and an appropriate error message is written to the error file.

4.1.6.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_6" and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and located in the GRF directory. The output files "iodl_6.out" and "hdjunk.grf" are created, but are empty. The expected result can be compared to the expected result comments in the "iodl_6.tst" file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using "runmscp.bat." The output file is called "iodl_6m.out." To run the Lahey Fortran-90 test program, repeat the same steps using "runlf.bat." The output file is called "iodl_6l.out." To run the Digital Visual Fortran-90 test program, repeat these steps using "rundv.bat." The output file is called "iodl_6d.out."

IODL_6C.ERR

```
"Failed to call CloseGroups writing","hdjunk.grf",
"Error writing data group [hd.grf], parameter [Int1(0,0,0,0,0,1)], unit [days]
       Can not write to an input file",
```

[3] This line (line #3) is actually a continuation of the previous line (line #2) that has wrapped around.

4.1.7 IODL_7

4.1.7.1 Description and Rationale

This case verifies that the DLL allows a program to read simultaneously from several different data group files. The files HD.SSF, AR.SSF, and EE.SSF are all read from.

4.1.7.2 Input Data

The input file to the testing program is as follows:

IODL_7.TST

```
“OpenGroups”
“ReadReal”
“Ee.ssf”,“BW_s”,“kg”, ; should return 52.4
“ReadInt”
“hd.ssf”,“Int”,“days”,; should return 20
“ReadReal”,
“Ar.ssf”,“fv”,“”, ; should return 52.4
“CloseGroups”
“Stop”
```

Data group files EE.SSF and AR.SSF and their associated dictionary files are as follows:

EE.SSF

```
3,
“Time: 12:00:00pm Date: 08/01/1998",
“SSFGen created this input file”,
“Executable 02/27/1998",
7,
“Ecd_acomm”,0,“FLOAT”,0,“mg/L water”,
52.4,
“Ecpcomm”,0,“FLOAT”,0,“Fg/g soil”,
3.1,
“Ecscomm”,0,“FLOAT”,0,“Fg/g soil”,
54.5,
“Ecsedcomm”,0,“FLOAT”,0,“Fg/g sediment”,
1,
“Ect_acomm”,0,“FLOAT”,0,“mg/L water”,
13.6,
“Tbaq_s”,0,“FLOAT”,0,“Fg/g-d”,
85.1,
“Tbter_s”,0,“FLOAT”,0,“Fg/g-d”,
14.5,
```

EE.DIC

```
16,
“Code”,“Dimensions”,“DataType”,“Min”,“Max”,“Units”,
“BW_s”,0,“FLOAT”,31,92,“kg”,
“CR_s”,0,“FLOAT”,3,11,“kg DW/d”,
“Crsed_s”,0,“FLOAT”,34,66,“kg DW/d”,
“Crsoil_s”,0,“FLOAT”,0,11,“kg DW/d”,
“Crw_s”,0,“FLOAT”,3,73,“kg DW/d”,
“Faqinv_s”,1”,FLOAT”,0,1,“unitless”,
“Faqp_s”,1,“FLOAT”,0,1,“unitless”,
“Finvert_s”,1,“FLOAT”,0,1,“unitless”,
“Fshell_s”,1,“FLOAT”,0,1,“unitless”,
“FT3_s”,1,“FLOAT”,0,1,“unitless”,
“FT4_s”,1,“FLOAT”,0,1,“unitless”,
“FT5_s”,1,“FLOAT”,0,1,“unitless”,
“Ftp_s”,1,“FLOAT”,0,1,“unitless”,
“Fvert_s”,1,“FLOAT”,0,1,“unitless”,
“Fworm_s”,1,“FLOAT”,0,1,“unitless”,
```

AR.SSF

```
3,
“Time: 12:00:00pm Date: 08/01/1998”,
“SSFGen created this input file”,
“Executable 02/27/1998",
7,
“fv”,0,“FLOAT”,0,“”,
52.4,
“Lat_Long”,2,“FLOAT”,0,“degrees”,
6,
1,3.1,
1,8.2,
1,7.1,
1,8.1,
1,3.9,
1,3,
“Pdens_p”,1,“FLOAT”,0,“g/cm³”,
8,6.1,6.4,8.8,7.9,8.1,8.7,6.4,6,
“Phi_p”,1,“FLOAT”,0,“percent”,
5,10.3,10.4,9.9,18.2,13,
“Scavcoef1”,0,“FLOAT”,0,“hr/s-mm”,
61.6,
“Scavcoef2”,0,“FLOAT”,0,“hr/s-mm”,
58,
“Scavcoef3”,0,“FLOAT”,0,“hr/s-mm”,
35.4,
```

AR.DIC

```
8,
“Code”,“Dimensions”,“DataType”,“Min”,“Max”,“Units”,
“fv”,0,“FLOAT”,31,92,“”,
“Lat_Long”,2,“FLOAT”,3,11,“degrees”,
“Pdens_p”,1,“FLOAT”,6,9,“g/cm³”,
“Phi_p”,1,“FLOAT”,9,19,“percent”,
“Scavcoef1”,0,“FLOAT,“49,63,“hr/s-mm”,
“Scavcoef2”,0,“FLOAT”,54,87,“hr/s-mm”,
“Scavcoef3”,0,“FLOAT”,21,42,“hr/s-mm”,
```

4.1.7.3 Expected Results

All reads are expected to accomplish without error.

4.1.7.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_7” and press return. The program runs and creates an output file called “iodl_7c.out,” located in the SSF directory, and an output file called “hdjunk.grf,” located in the GRF directory. The “iodl_7c.out” file lists the functions called and the results returned. The “hdjunk.grf” file is empty because no write calls are made. The “iodl_7c.out” file is shown as follows. The results returned can be compared to the expected result comments in the “iodl_7.tst” file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using “runmscp.bat.” The output file is called “iodl_7m.out.” To run the Lahey Fortran-90 test program, repeat these same steps using “runlf.bat.” The output file is called “iodl_7l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_7d.out.”

IODL_7C.OUT

```
Call OpenGroups
 Call ReadReal    Ee.ssf   BW_s   kg
 Returned   52.4
 Call ReadInt    hd.ssf   Int   days
 Returned   20
 Call ReadReal    Ar.ssf   fv
 Returned   52.4
    Call CloseGroups
```

4.1.8 IODL_8

4.1.8.1 Description and Rationale

This test attempts to read a variable of a particular data type as if it were some other data type (that is, attempt to read an integer variable as if it were a float). Because this DLL will be widely used

throughout the FRAMES-HWIR Technology Software System, all 12 possible data type errors will be tested (IODL_8A through IODL_8L).

4.1.8.2 Input Data

The input files to the testing program are as follows:

IODL_8A.TST (attempts to read a real variable as if it were an integer)

```
“OpenGroups”
“ReadInt”
“hd.ssf”,“Int”,“days”, ; should return 20
“ReadInt1”,
“hd.ssf”,“Real1”,“days”,1, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8B.TST (attempts to read a logical variable as if it were an integer)

```
“OpenGroups”
“ReadInt”
“hd.ssf”,“Int”,“days”, ; should return 20
“ReadInt1”
“hd.ssf”,“Log1”,“days”,1, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8C.TST (attempts to read a string variable as if it were an integer)

```
“OpenGroups”
“ReadInt”
“hd.ssf”,“Int”,“days.” ; should return 20
“ReadInt1”
“hd.ssf”,“String1”“days”,1, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8D.TST (attempts to read an integer variable as if it were a real)

```
“OpenGroups”
“ReadReal”,
“hd.ssf”,“Real”,“days”, ; should return 20.1
“ReadReal1”,
“hd.ssf”,“Int1”“days”,5, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8E.TST (attempts to read a logical variable as if it were a real)

```
“OpenGroups”
“ReadReal”,
“hd.ssf”,“Real”,“days”, ; should return 20.1
“ReadReal1",
“hd.ssf”,“Log1”,“days”,5, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8F.TST (attempts to read a string variable as if it were a real)

```
“OpenGroups”
“ReadReal”,
“hd.ssf”,“Real”,“days”, ; should return 20.1
“ReadReal1”,
“hd.ssf”,“String1”,“days”,5, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8G.TST (attempts to read an integer variable as if it were a logical)

```
“OpenGroups”
“ReadLog”,
“hd.ssf”,“Log”,“”, ; should return T
“ReadLog1”,
“hd.ssf”,“Int1”,“”,5, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8H.TST (attempts to read a real variable as if it were a logical)

```
“OpenGroups”
“ReadLog”,
“hd.ssf”,“Log”,“”, ; should return T
“ReadLog1”,
“hd.ssf”,“Real1”,“”,5, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8I.TST (attempts to read a string variable as if it were a logical)

```
“OpenGroups”
“ReadLog”,
“hd.ssf”,“Log”,“”, ; should return T
“ReadLog1”,
“hd.ssf”,“String1”,“”,5, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8J.TST (attempts to read an integer variable as if it were a string)

```
“OpenGroups”
“ReadString”,
“hd.ssf”,“String”,“”, ; should return A
“ReadString1”,
“hd.ssf”,“Int1”,“”,5, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8K.TST (attempts to read a real variable as if it were a string)

```
“OpenGroups”
“ReadString”,
“hd.ssf”,“String”,“”, ; should return A
“ReadString1”,
“hd.ssf”,“Real1”,“”,5, ; should return an error
“CloseGroups”
“Stop”
```

IODL_8L.TST (attempts to read a logical variable as if it were a string)

```
“OpenGroups”
“ReadString”,
“hd.ssf”,“String”,“”, ; should return A
“ReadString1”,
“hd.ssf”,“Log1”,“”,5, ; should return an error
“CloseGroups”
“Stop”
```

4.1.8.3 Expected Results

An error is expected to be generated for each of the 12 cases with an appropriate error message written to the error file; program execution is expected to immediately cease upon discovering the error.

4.1.8.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_8a” and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files “iodl_8ac.out” and “hdjunk.grf”are created, but are empty. The expected result can be compared to the expected result comments in the “iodl_8a.tst” file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using “runmscp.bat.” The output file is called “iodl_8am.out.” To run the Lahey Fortran-90 test program, repeat the same steps using “runlf.bat.” The output file is called “iodl_8al.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_8d.out.”

Run cases “iodl_8b” through “iodl_8l” by repeating the this process and replacing “iodl_8a” with “iodl_8b” etc. The appropriate error message appears in the error file in each case.

IODL_8AC.ERR

```
“Failed to call CloseGroups writing”,“hdjunk.grf”,
“Error reading data group [hd.ssf], parameter [Real1(1,0,0,0,0,0)], unit
[days]
       Parameter type does not match dictionary”,
```

4.1.9 IODL_9

4.1.9.1 Description and Rationale

This case verifies that DLL writes an error message when an element is requested from an array that is outside the range of the elements defined in the data group file (for example, requesting element number 4 from an array with only three elements listed in the data group file).

4.1.9.2 Input Data

The input file to the testing program is as follows:

IODL_9.TST

```
“OpenGroups”
“ReadInt”
“hd.ssf”,“Int”,“days”, ; should return 20
“ReadInt1”
“hd.ssf”,“Int1”,““days”,1, ; should return 1
“ReadInt2”
“hd.ssf”,“Int2”,“days”,2,6, ; should return 11
“ReadInt3”
“hd.ssf”,“Int3”,“days”,2,3,5, ; should return error
“CloseGroups”
“Stop”
```

4.1.9.3 Expected Results

The first three reads are expected to occur properly, but the fourth fails with an error because the 3-dimensional array does not have an element 2,3,5 in the data group file.

4.1.9.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_9” and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files “iodl_9c.out” and “hdjunk.grf”

are created, but are empty. The expected result can be compared to the expected result comments in the "iodl_9.tst" file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using "runmscp.bat." The output file is called "iodl_9m.out." To run the Lahey Fortran-90 test program, repeat the same steps using "runlf.bat." The output file is called "iodl_9l.out." To run the Digital Visual Fortran-90 test program, repeat these steps using "rundv.bat." The output file is called "iodl_9d.out."

ERROR File

```
"Failed to call CloseGroups writing","hdjunk.grf",
"Error reading data group [hd.ssf], parameter [Int3(2,3,5,0,0,0)], unit [days]
       Parameter value has not been defined",
```

4.1.10 IODL_10

4.1.10.1 Description and Rationale

This case verifies that DLL can write an error message when a data group file that does not exist is listed in the ARGUMENTS environment variable as a file that can be read from (that is, an attempt is made to access a data group that does not exist).

4.1.10.2 Input Data

The input file to the testing program is as follows:

IODL_10.TST

```
"OpenGroups"
"ReadInt"
"hd10.grf","IntA","days", ; should return error "parameter not found"
"CloseGroups"
"Stop"
```

4.1.10.3 Expected Results

An attempt to open the group HD10.GRF is expected to fail, and an appropriate error message is written to the error file.

4.1.10.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_10" and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files "iodl_10c.out" and "hdjunk.grf" are created, but are empty. The expected result can be compared to the expected result comments in the "iodl_10.tst" file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using "runmscp.bat." The output file is called "iodl_10m.out." To run the Lahey Fortran-90 test program, repeat these same steps using "runlf.bat." The output file is called "iodl_10l.out." To run the Digital Visual Fortran-90 test program, repeat these steps using "rundv.bat." The output file is called "iodl_10d.out."

IODL_10C.ERR

```
"Failed to call CloseGroups writing","hdjunk.grf",
"Data group hd10.grf
       Can not find/open data group",
```

4.1.11 IODL_11

4.1.11.1 Description and Rationale

This case verifies that DLL does not allow access to an existing data group that is not listed in the ARGUMENTS environment variable. An attempt is made to read from the file HD11.GRF, which exists, but is not listed, in ARGUMENTS. Data group HD11.GRF is a copy of HD.GRF, which is listed in Appendix A.

4.1.11.2 Input Data

The input file to the testing program is as follows:

IODL_11.TST

```
"OpenGroups"
"ReadInt"
"hd11.grf","IntA","days", ;should return error "data group not defined"
"CloseGroups"
"Stop"
```

The C++ version of the batch file that runs this case is as follows:

```
echo The call arguments are:
set ARGUMENTS=1:05pm 03/30/1998 \HWIR98\IODLL\SSF \HWIR98\IODLL\GRF 1 hd.grf
hd_junk.grf
echo %ARGUMENTS%
del %1c.out
del %1.out
call cdlltst %1
copy %1.out %1c.out
```

This batch file shows that HD11.GRF is not listed in the ARGUMENTS environment variable.

4.1.11.3 Expected Results

The attempted read is expected to fail, and an appropriate error message is written to the error file.

4.1.11.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_11" and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files "iodl_11c.out" and "hdjunk.grf"are created, but are empty. The expected result can be compared to the expected result comment in the "iodl_11.tst" file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using "runmscp.bat." The output file is called "iodl_11m.out." To run the Lahey Fortran-90 test program, repeat these same steps using "runlf.bat." The output file is called "iodl_11l.out." To run the Digital Visual Fortran-90 test program, repeat these steps using "rundv.bat." The output file is called "iodl_11d.out."

IODL_11C.ERR

```
"Failed to call CloseGroups writing","hdjunk.grf",
"Error reading data group [hd.grf], parameter [IntA(1,0,0,0,0,0)], unit [days]
       Data group not found",
```

4.1.12 IODL_12

4.1.12.1 Description and Rationale

This test reads a variable from a test-specific SSF, in which the value for that variable falls outside the range allowed.

4.1.12.2 Input Data

The input file to the testing program is as follows:

IODL_12.TST

```
"OpenGroups"
"ReadInt"
"hd12.ssf","IntA","days", ;should return error "parameter out of range"
"CloseGroups"
"Stop"
```

The test-specific SSF is as follows:

HD12.SSF

```
2,
“Time: 12:00:00pm Date: 03/16/1998”,
“Created by hand by Karl Castleton”,
2,
“IntA”,0,“INTEGER”,0,“days”,
-1,
“IntB”,0,“INTEGER”,0,“days”
100,
```

The allowed range for variable IntA is specified as 0 to 100 in the data dictionary (HD.DIC, listed in Appendix A). In HD12.SSF, its value is shown as -1, which is outside the allowed range.

4.1.12.3 Expected Results

The attempted read is expected to fail, and an appropriate error message is written to the error file.

4.1.12.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_12” and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files “iodl_12c.out” and “hdjunk.grf”are created, but are empty. The expected result can be compared to the expected result comments in the “iodl_12.tst” file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using “runmscp.bat.” The output file is called “iodl_12m.out.” To run the Lahey Fortran-90 test program, repeat thee same steps using “runlf.bat.” The output file is called “iodl_12l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_12d.out.”

IODL_12C.ERR

```
“Failed to call CloseGroups writing”,“hdjunk.grf”,
“IntA”,“ parameter”,
“Data group hd12.ssf
       Parameter value out of range”,
```

4.1.13 IODL_13

4.1.13.1 Description and Rationale

This test reads a variable from a test-specific SSF, where the value for that variable falls exactly on an endpoint of the range allowed.

4.1.13.2 Input Data

The input file to the testing program is as follows:

IODLL_13.TST

```
“OpenGroups”
“ReadInt”
“hd.ssf”,“IntA”,“days”, ; should return 0
“ReadInt”
“hd.ssf”,“IntB”,“days”, ; should return 100
“ReadReal”,
“hd.ssf”,“RealA”,“days”, ; should return 0.0
“ReadReal”,
“hd.ssf”,“RealB”,“days”, ; should return 100.0
“CloseGroups”
“Stop”
```

Data group file HD.SSF can be found in Appendix A.

4.1.13.3 Expected Results

The DLL is expected to read the values correctly without error because the range given is inclusive (that is, it includes the endpoints).

4.1.13.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_13” and press return. The program runs and creates an output file called “iodl_13c.out,” located in the SSF directory, and an output file called “hdjunk.grf,” located in the GRF directory. The “iodl_13c.out” file lists the functions called and the results returned. The “hdjunk.grf” file is empty because no write calls are made. The “iodl_13c.out” file is shown as follows. The results returned can be compared to the expected result comments in the “iodl_13.tst” file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using “runmscp.bat.” The output file is called “iodl_13m.out.” To run the Lahey Fortran-90 test program, repeat the same steps using “runlf.bat.” The output file is called “iodl_13l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_13d.out.”

IODL_13C.OUT

```
Call OpenGroups
 Call ReadInt    hd.ssf   IntA   days
 Returned   0
 Call ReadInt    hd.ssf   IntB   days
 Returned   100
 Call ReadReal    hd.ssf   RealA   days
 Returned   0
```

```
 Call ReadReal    hd.ssf   RealB   days
 Returned   100
 Call CloseGroups
```

4.1.14 IODL_14

4.1.14.1 Description and Rationale

This test attempts to read a variable using a negative index.

4.1.14.2 Input Data

The input file to the testing program is as follows:

IODL_14.TST

```
“OpenGroups”
“ReadInt2"
“hd.ssf”,“Int2”,“days”,2,-6, ;should return an error for negative index
“CloseGroups”
“Stop”
```

4.1.14.3 Expected Results

An error is expected to be generated with an appropriate error message written to the error file. Program execution ceases immediately upon discovering the error.

4.1.14.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_14” and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files “iodl_14c.out” and “hdjunk.grf”are created, but are empty. The expected result can be compared to the expected result comment in the “iodl_14.tst” file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using “runmscp.bat.” The output file is called “iodl_14m.out.” To run the Lahey Fortran-90 test program, repeat these same steps using “runlf.bat.” The output file is called “iodl_14l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_14d.out.”

IODL_14C.ERR

```
“Failed to call CloseGroups writing”,“hdjunk.grf”,
“Error reading data group [hd.ssf], parameter [Int2(2,-6,0,0,0,0)], unit
[days]
       Parameter indices less than or equal to zero”,
```

4.1.15 IODL_15

4.1.15.1 Description and Rationale

This test attempts to read a variable in which the units specified by the calling program do not match the units in the Data Dictionary (HD.DIC, listed in Appendix A).

4.1.15.2 Input Data

The input file to the testing program is as follows:

IODL_15.TST

```
“OpenGroups”
“ReadInt2”
“hd.ssf”,“Int2”,“months”,2,6, ; should return an error for wrong units
“CloseGroups”
“Stop”
```

4.1.15.3 Expected Results

An error is expected to be generated with an appropriate error message written to the error file. Program execution ceases immediately upon discovering the error.

4.1.15.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_15” and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files “iodl_15c.out” and “hdjunk.grf”are created, but are empty. The expected result can be compared to the expected result comments in the “iodl_15.tst” file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using “runmscp.bat.” The output file is called “iodl_15m.out.” To run the Lahey Fortran-90 test program, repeat these same steps using “runlf.bat.” The output file is called “iodl_15l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_15d.out.”

IODL_15C.ERR

```
“Failed to call CloseGroups writing”,“hdjunk.grf”,
“Error reading data group [hd.ssf], parameter [Int2(2,6,0,0,0,0)], unit
[months]
       Parameter unit does not match dictionary”,
```

4.1.16 IODL_16

4.1.16.1 Description and Rationale

This case verifies that DLL generates a warning message when the subroutine OpenGroups is called twice in the same run before the subroutine CloseGroups is called.

4.1.16.2 Input Data

The input file to the testing program is as follows:

IODL_16.TST

```
“OpenGroups”
“OpenGroups” ; warning message should be generated
“CloseGroups”
“Stop”
```

4.1.16.3 Expected Results

A warning is generated with an appropriate warning message written to the warning file, and program execution is expected to continue.

4.1.16.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_16” and press return. The program runs and creates an output file called “iodl_16c.out,” located in the SSF directory, and an output file called “hdjunk.grf,” located in the GRF directory. A “Warning” file is also generated and is located in the GRF directory. The “iodl_16c.out” file lists the functions called and the results returned. The “hdjunk.grf” file is empty because no write calls are made. The “Warning” and “iodl_16c.out” files are shown as follows. The results returned can be compared to the expected results comment in the “iodl_16.tst” file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using “runmscp.bat.” The output file is called “iodl_16m.out.” To run the Lahey Fortran-90 test program, repeat thes ame steps using “runlf.bat.” The output file is called “iodl_16l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_16d.out.”

IODL_16C.WRN

```
“Starting OpenGroups”,
“Ending OpenGroups”,
“Data groups already opened”,
“Starting CloseGroups”,
“Closing group”,“hd.ssf”,
“Closing group”,“hdjunk.grf”,
“Ending CloseGroups”,
```

IODL_16C.OUT

```
Call OpenGroups
 Call OpenGroups
 Call CloseGroups
```

4.1.17 IODL_17

4.1.17.1 Description and Rationale

This case verifies that DLL produces an error message if an attempt is made to read a string (from the ARGUMENTS environment variable) that does not exist. In this case, the number of arguments is seven, and an attempt is made to read the eighth argument.

4.1.17.2 Input Data

The input file to the testing program is as follows:

IODL_17.TST

```
“OpenGroups”
“NumArgs”   ; should return 7
“GetArgString”
  8         ; should produce an error
“CloseGroups”
“Stop”
```

The C++ version of the batch file that sets the ARGUMENTS environment variable is as follows:

IODL_17C.BAT

```
echo The call arguments are:
set ARGUMENTS=1:05pm 03/30/1998 \HWIR98\IODLL\SSF \HWIR98\IODLL\GRF 1 hd.ssf
hd_junk.grf[4]
echo %ARGUMENTS%
del %1c.out
del %1.out
call cdlltst %1
copy %1.out %1c.out
```

4.1.17.3 Expected Results

The call to NumArgs is expected to occur properly, but the call to GetArgString asking for the eighth argument fails with an appropriate error message written to the error file.

[4] This line (line #3) is actually a continuation of the previous line (line #2) that has wrapped around.

4.1.17.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_17" and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files "iodl_17c.out" and "hdjunk.grf"are created, but are empty. The expected result can be compared to the expected result comments in the "iodl_17.tst" file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using "runmscp.bat." The output file is called "iodl_17m.out." To run the Lahey Fortran-90 test program, repeat the same steps using "runlf.bat." The output file is called "iodl_17l.out." To run the Digital Visual Fortran-90 test program, repeat these steps using "rundv.bat." The output file is called "iodl_17d.out."

IODL_17C.ERR

```
"Failed to call CloseGroups writing","hdjunk.grf,
"      Call argument does not exist",
```

4.1.18 IODL_18

4.1.18.1 Description and Rationale

This case verifies that DLL can write to and read from the same SSF if it is the last file listed in the ARGUMENTS environment variable.

4.1.18.2 Input Data

The input file to the testing program is as follows:

IODL_18.TST

```
"OpenGroups"
"WriteInt"
"hd18.ssf","Int","days",11, ; should write 11
"WriteInt1"
"hd18.ssf","Int1","days",1,22, ; should Write 22
"ReadInt"
"hd18.ssf","Int","days", ; should return 11
"ReadInt1"
"hd18.ssf","Int1","days",1, ; should return 22
"CloseGroups"
"Stop"
```

4.1.18.3 Expected Results

The DLL is expected to write to and read from the SSF without error.

4.1.18.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_18" and press return. The program runs and creates an output file called "iodl_18c.out," located in the SSF directory, and an output file called "hd18.ssf," located in the SSF directory. The "iodl_18c.out" file lists the functions called and the results returned. The "hd18.ssf" file contains the information written out in the correct format. The "iodl_18c.out" and "hd18.ssf" files are shown as follows. The results returned can be compared to the expected results comment in the "iodl_18.tst" file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using "runmscp.bat." The output file is called "iodl_18m.out." To run the Lahey Fortran-90 test program, repeat the same steps using "runlf.bat." The output file is called "iodl_18l.out." To run the Digital Visual Fortran-90 test program, repeat these steps using "rundv.bat." The output file is called "iodl_18d.out."

IODL_18C.OUT

```
Call OpenGroups
 Call WriteInt    hd18.ssf   Int   days  11
 Call WriteInt1    hd18.ssf   Int1   days  1 22
 Call ReadInt    hd18.ssf   Int   days
 Returned   11
 Call ReadInt1    hd18.ssf   Int1   days  1
 Returned   22
 Call CloseGroups
```

HD18.SSF

```
1,
"hd18.ssf","data group",
2,
"Int",0,"INTEGER",0,"days",
11,
"Int1",1,"INTEGER",0,"days",
1,22,
```

4.1.19 IODL_19

4.1.19.1 Description and Rationale

This case verifies that DLLl produces an error message if an attempt is made to read an integer that does not exist from the ARGUMENTS environment variable. In this case, the number of arguments is seven, and an attempt is made to read the eighth argument.

4.1.19.2 Input Data

The input file to the testing program is as follows:

IODL_19.TST

```
“OpenGroups”
“NumArgs”   ; should return 7
“GetArgInt”
  8         ; should produce an error
“CloseGroups”
“Stop”
```

The batch file that sets the ARGUMENTS environment variable is identical to test case IODL_17.

4.13.19.3 Expected Results

The call to NumArgs is expected to occur properly, but the call to GetArgInt asking for the eighth argument fails with an appropriate error message being written to the error file.

4.1.19.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_19” and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files “iodl_19c.out” and “hdjunk.grf” are created, but are empty. The expected result can be compared to the expected result comments in the “iodl_19.tst” file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using “runmscp.bat.” The output file is “iodl_19m.out.” To run the Lahey Fortran-90 test program, repeat the same steps using “runlf.bat.” The output file is called “iodl_19l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_19d.out.”

IODL_19C.ERR

```
“Failed to call CloseGroups writing”,“hdjunk.grf”,
“      Call argument does not exist”,
```

4.1.20 IODL_20

4.1.20.1 Description and Rationale

This case verifies that DLL generates an error when an attempt is made to read a parameter from an empty SDP/SSF or GRF file. The file is shown, if a directory listing is produced, but it contains no data.

4.1.20.2 Input Data

The input file to the testing program is as follows:

IODL_20.TST

```
“OpenGroups”
“ReadInt”
“hdjunk.grf”,“Int”,“days”, ;should return error - “hdjunk.grf” is empty
“ReadInt1”
“hdjunk.grf”,“Int1”,“days”,1, ; should error before reaching this line
“CloseGroups”
“Stop”
```

4.1.20.3 Expected Results

An error is expected to be generated with an appropriate error message being written to the error fil. Program execution immediately ceases upon discovering the error.

4.1.20.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_20” and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files “iodl_20c.out” and “hdjunk.grf” are created, but are empty. The expected result can be compared to the expected result comments in the “iodl_20.tst” file.

To run the Microsoft® Visual C++ test program, repeat the preceding steps using “runmscp.bat.” The output file is called “iodl_20m.out.” To run the Lahey Fortran-90 test program, repeat the same steps using “runlf.bat.” The output file is “iodl_20l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_20d.out.”

ERROR File

```
“Failed to call CloseGroups writing”,“hdjunk.grf”,
“Error reading data group [hdjunk.grf], parameter [Int(1,0,0,0,0,0)], unit
[days]
       Parameter value has not been defined”,
```

4.1.21 IODL_21

4.1.21.1 Description and Rationale

This case verifies that DLL does not write a value to a SDP/SSF or GRF file that is out of the range allowed for that parameter.

4.1.21.2 Input Data

The input file to the testing program is as follows:

IODL_21.TST

```
“OpenGroups”
“WriteInt”
“hdjunk.grf”,“Int”,“days”,11, ; should write 11
“WriteInt1”
“hdjunk.grf”,“Int1”,“days”,1,102, ;should error-max value allowed is 100
“CloseGroups”
“Stop”
```

4.1.21.3 Expected Results

An error is expected to be generated with an appropriate error message being written to the error file. Program execution immediately ceases upon discovering the error.

4.1.21.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_21” and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files “iodl_21c.out” and “hdjunk.grf”are created, but are empty. The expected result can be compared to the expected result comments in the “iodl_21.tst” file.

To run the Microsoft® Visual C++ test program, repeat the previous steps using “runmscp.bat.” The output file is called “iodl_21m.out.” To run the Lahey Fortran-90 test program, repeat the same steps using “runlf.bat.” The output file is called “iodl_21l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_21d.out.”

IODL_21C.ERR

```
“Failed to call CloseGroups writing”,“hdjunk.grf”,
“Error writing data group [hdjunk.grf], parameter [Int1(0,0,0,0,0,1)], unit
[days]
                                          Parameter value out of range”,
```

4.1.22 IODL_22

4.1.22.1 Description and Rationale

This case verifies that the DLL will halt execution if an attempt is made to read a parameter in a data group before subroutine OpenGroups is called. An attempt is made to read from HD.SSF before it is opened.

4.1.22.2 Input Data

The input file to the testing program is as follows:

IODL_22.TST

```
“ReadInt”
“hd.ssf”,“Int”,“days”, ; should halt execution - error open groups not called
“CloseGroups”
“Stop”
```

4.1.22.3 Expected Results

The attempt to read from HD.SSF without calling subroutine OpenGroups first is expected to fail, and exucution should halt with an illegal operation error.

4.1.22.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_22" and press return. The program runs and terminates due to an error. An error message, should appear that states that the program has performed an illegal operation and will be shut down.

To run the Microsoft® Visual C++ test program, repeat the preceding steps using “runmscp.bat.” To run the Lahey Fortran-90 test program, repeat these same steps using “runlf.bat.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.”

4.1.23 IODL_23

4.1.23.1 Description and Rationale

This case verifies that the DLL will halt execution if an attempt is made to write a parameter to a data group before subroutine OpenGroups is called. An attempt is made to write to HDJUNK.GRF before it is opened.

4.1.23.2 Input Data

IODL_23.TST

```
“WriteInt”
“hdjunk.grf”,“Int”,“days”,11, ; should error open groups not called
“CloseGroups”
“Stop”
```

4.1.23.3 Expected Results

The attempt to write to HDJUNK.GRF without first calling subroutine OpenGroups is expected to fail, and an error message will be written to the screen stating that program execution has been halted.

4.1.23.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_23" and press return. The program runs and terminates due to an error. An error message should appear that states that the program has performed an illegal operation and will be shut down.

To run the Microsoft® Visual C++ test program, repeat the preceding steps using "runmscp.bat." To run the Lahey Fortran-90 test program, repeat these same steps using "runlf.bat." To run the Digital Visual Fortran-90 test program, repeat these steps using "rundv.bat."

4.1.24 IODL_24

4.1.24.1 Description and Rationale

This case verifies that DLL produces an error message if an attempt is made to read anything besides a "T" or an "F" from a logical parameter. In this case, an attempt is made to read an "N" from a logical parameter in HD24.GRF.

4.1.24.2 Input Data

The input file to the testing program is as follows:

IODL_24.TST

```
"OpenGroups"
"ReadLog",
"hd24.grf","Log","",N, ; should return an error non T or F
"CloseGroups"
"Stop"
```

4.1.24.3 Expected Results

An error is expected to be generated with an appropriate error message written to the error file, and program execution will immediately cease upon discovering the error.

4.1.24.4 Conducting the Test

To run this case, double click on "runc.bat." At the prompt, type "iodl_24" and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files "iodl_24c.out" and "hdjunk.grf" are created, but are empty. The expected result can be compared to the expected result comments in the "iodl_24.tst" file.

To run the Microsoft® Visual C++ test program, repeat the preceding steps using "runmscp.bat." The output file is called "iodl_24m.out." To run the Lahey Fortran-90 test program, repeat these same steps using "runlf.bat." The output file is called "iodl_24l.out." To run the Digital Visual Fortran-90 test program, repeat these using "rundv.bat." The output file is called "iodl_24d.out."

IODL_24C.ERR

```
“Failed to call CloseGroups writing”,“hdjunk.grf”,
“Log”, “parameter”,
“Data group hd24.ssf
       Parameter value out of range”,
```

4.1.25 IODL_25

4.1.25.1 Description and Rationale

This case verifies that DLL produces an error message if an attempt is made to read an undefined variable (a variable that does not occur in the Data Dictionary) from the example SSF.

4.1.25.2 Input Data

The input file to the testing program is as follows:

IODL_25.TST

```
“OpenGroups”
“ReadInt”
“hd.ssf”,“IntZ”,“days”, ; should error out with appropriate message
“CloseGroups”
“Stop”
```

4.1.25.3 Expected Results

An error is expected to be generated with an appropriate error message written to the error file. Program execution immediately ceases upon discovering the error.

4.1.25.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_25” and press return. The program runs and terminates due to an error. The error file, shown as follows, is created with an appropriate message and is located in the GRF directory. The output files “iodl_25c.out” and “hdjunk.grf”are created, but are empty. The expected result can be compared to the expected result comments in the “iodl_25.tst” file.

To run the Microsoft® Visual C++ test program, repeat the preceding steps using “runmscp.bat.” The output file is called “iodl_25m.out.” To run the Lahey Fortran-90 test program, repeat these same steps using “runlf.bat.” The output file is called “iodl_25l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_25d.out.”

IODL_25C.ERR

```
“Failed to call CloseGroups writing”,“hdjunk.grf”,
“Error reading data group [hd.ssf], parameter [IntZ(1,0,0,0,0,0)], unit [days]
       Parameter not found in dictionary”,
```

4.1.26 IODL_26

4.1.26.1 Description and Rationale

This case verifies that DLL can read from all data groups (SSF files) that currently exist. An attempt is made to read a parameter from all data groups. The data groups read from are af.ssf, aq.ssf, aq.ssf, ar.ssf, ee.ssf, ff.ssf, he.ssf, hr.ssf, sl.ssf, sr.ssf, sw.ssf, tf.ssf, vz.ssf, and ws.ssf. These files and their associated dictionaries are not included in this test plan, but are available electronically.) It is assumed that if a single parameter can be read from each data group, all parameters can be read.

4.1.26.2 Input Data

The input file to the testing program is as follows:

IODL_26.TST

```
“OpenGroups”
“ReadReal1”,
“af.ssf”,“aqpBCFcalc_j”,“L/kg lipid”,1, ; should return 52.4
“ReadReal”,
“aq.ssf”,“AL”,“m”, ; should Read 299.1
“ReadReal”,
“ar.ssf”,“fv”,“”, ; should Read 52.4
“ReadReal”,
“ee.ssf”,“BW_s”,“kg”, ; should Read 52.4
“ReadReal”,
“er.ssf”,“ECpcomm”,“Fg/g soil”, ; should Read 3.1
“ReadReal”,
“ff.ssf”,“Br_forage”,“(Fg/g)/(Fg/g)”, ; should Read 54.5
“ReadReal”,
“he.ssf”,“Bri_cr_1”,“m³/day”, ; should Read 1
“ReadReal”,
“hr.ssf”,“URFInhal”,“per Fg/m³”, ; should Read 14.5
“ReadReal1",
“sl.ssf”,“AirLocElev”,“m”,8, ; should return 38.2
“ReadReal”,
“sr.ssf”,“bcA”,“unitless”, ; should Read 0.3
“ReadReal4”,
“sw.ssf”,“sw_var53”,“g/day”,1,1,1,4, ; should return 12.4
“ReadReal”,
“tf.ssf”,“VGag_forage”,“unitless”, ; should Read 8.5
“ReadReal”,
“vz.ssf”,“BETA”,“N/A”, ; should Read 1.4
```

```
“ReadReal”,
“ws.ssf”,“7Q10”,“m³/m²-d”, ; should Read 0.3
“CloseGroups”
“Stop”
```

4.1.26.3 Expected Results

The DLL is expected to read from all the SSF data group files without error.

4.1.26.4 Conducting the Test

To run this case, double click on “runc.bat.” At the prompt, type “iodl_26” and press return. The program runs and creates an output file called “iodl_26l.out,” located in the SSF directory. As follows, this file lists the functions called and the results returned. The results returned can be compared to the expected result comments in the “iodl_26.tst” file.

To run the Microsoft® Visual C++ test program, repeat the preceding steps using “runmscp.bat.” The output file is called “iodl_26m.out.” To run the Lahey Fortran-90 test program, repeat these same steps using “runlf.bat.” The output file is called “iodl_26l.out.” To run the Digital Visual Fortran-90 test program, repeat these steps using “rundv.bat.” The output file is called “iodl_26d.out.”

IODL_26L.OUT

```
Call OpenGroups
 Call ReadReal1    af.ssf   aqpBCFcalc_j   L/kg lipid  1
 Returned   52.4
 Call ReadReal    aq.ssf   AL   m
 Returned   299.1
 Call ReadReal    ar.ssf   fv
 Returned   52.4
 Call ReadReal    ee.ssf   BW_s   kg
 Returned   52.4
 Call ReadReal    er.ssf   ECpcomm   Fg/g soil
 Returned   3.1
 Call ReadReal    ff.ssf   Br_forage   (Fg/g)/(Fg/g)
 Returned   54.5
 Call ReadReal    he.ssf   Bri_cr_1   m³/day
 Returned   1
 Call ReadReal    hr.ssf   URFInhal   per Fg/m³
 Returned   14.5
 Call ReadReal1    sl.ssf   AirLocElev   m  8
 Returned   38.2
 Call ReadReal    sr.ssf   bcA   unitless
 Returned   0.3
 Call ReadReal4    sw.ssf   sw_var53   g/day  1 1 1 4
 Returned   12.4
 Call ReadReal    tf.ssf   VGag_forage   unitless
 Returned   8.5
 Call ReadReal    vz.ssf   BETA   N/A
```

```
Returned   1.4
Call ReadReal    ws.ssf   7Q10   m³/m²-d
Returned   0.3
Call CloseGroups
```

4.2 HWIR Monte Carlo DLL

Document to be provided by TetraTech.

5.0 Quality Assurance Program

The facilitating DLLs of the HWIRIO.DLL, and the HWIRMC.DLL were developed under a quality assurance program documented in Gelston et al. (1998). In that program, quality is defined as the ability of the software to meet client needs. Meeting client needs starts with a shared understanding of how the software must perform and continues with attention to details throughout the software life cycle of design, development, testing, and implementation.

Figure 5.1 outlines the software development process used for the facilitating DLLs, highlighting the quality check points. (Note that the activities for the facilitating DLLs flow down the left side of the figure because it is software developed for the first time, as opposed to a modification to existing software.) The process shown is designed for compatibility with similar processes used by other government agencies. For example, this quality process compares favorably with that in the U.S. Environmental Protection Agency Directive 2182, *System Design and Development Guidance* (EPA 1997). It also compares favorably with the Office of Civilian Radioactive Waste Management's *Quality Assurance Requirements and Description, Supplement I, Software* (OCRWM 1995). Activities roughly equivalent across these processes are shown in Table 5.1.

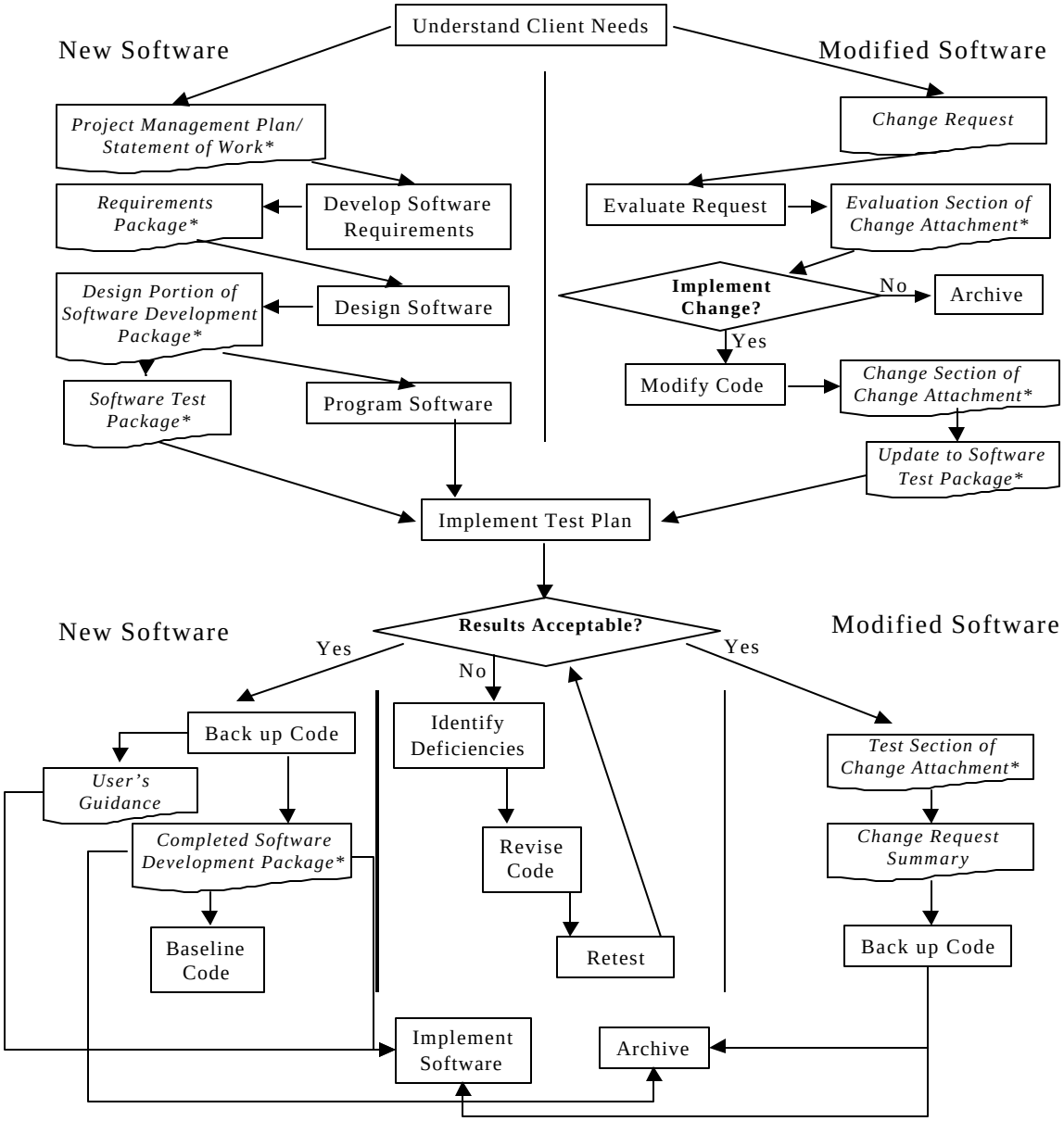

Figure 5.1 Ensuring Quality in the Environmental Software Development Process
(* *Indicates quality review stage; box with wavy bottom line and italics font indicates a document rather than an activity*)

Table 5.1 Relationship of PNNL Environmental Software Development Process to Quality Assurance Requirements (OCRWM 1995; EPA 1997)

OCRWM [a] Quality Assurance Requirement[b]	**EPA Essential Element of Information[c]**	**Environmental Software Process Equivalent (Section)**
	4—System Implementation Plan	Project Management Plan or Statement of Work
I.2.5A Functional Requirements Information Documentation; I.2.5C Requirements and Design Documentation	5—System Detailed Requirements Document	Requirements Package
I.2.1 Software Life Cycles, Baselines (see Appendix C), and Controls	6—Software Management Plan	Project Management Plan or Statement of Work and Gelston et al. (1998)
I.2.2 Software Verification[d] and Software Validation; I.2.4 Software Validation[e]	7—Software Test and Acceptance Plan	Software Test Package
I.2.3 Software Verification; I.2.5C Requirements and Design Information Documentation	8—Software Design Document	Design Portion of Software Development Package
I.2.6A Configuration Identification		Completed Software Development Package
I.2.6B Configuration Control; I.2.6C Configuration Status; I.2.7 Defect Reporting and Resolution[f]	9—Software Maintenance Document	Modification Documentation
	10—Software Operations Document	User"s Guidance and Training
I.2.5B User Information Documentation	11—Software User's Reference Guide	User's Guidance and Training
	12—System Integration Test Reports	Software Test Package

(a) OCRWM = Office of Civilian Radioactive Waste Management.
(b) Note that OCRWM requirement I.2.8, Control of the Use of Software, is the responsibility of the OCRWM-related client.
(c) Elements 1 through 3 are generally completed by clients in the U.S. Environmental Protection Agency before contract initiation with the project team.
(d) Verification includes informal code testing by software engineers to ensure that code functions as required.
(e) Validation includes testing by those other than the software engineers who developed the code to provide an independent confirmation that software functions as required.
(f) Note that some changes requested by clients may not be made in the software unless funding has been allocated for such modifications.

6.0 References

Documentation for the FRAMES-HWIR Technology Software System

Volume 1: Overview of the FRAMES-HWIR Technology Software System. 1998. PNNL-11914, Vol. 1, Pacific Northwest National Laboratory, Richland, Washington.

Volume 2: System User Interface Documentation. 1998. PNNL-11914, Vol. 2, Pacific Northwest National Laboratory, Richland, Washington.

Volume 3: Distribution Statistics Processor Documentation. 1998. TetraTech, Lafayette, California.

Volume 4: Site Definition Processor Documentation. 1998. PNNL-11914, Vol. 4, Pacific Northwest National Laboratory, Richland, Washington.

Volume 5: Computational Optimization Processor Documentation. 1998. TetraTech, Lafayette, California.

Volume 6: Multimedia Multipathway Simulation Processor Documentation. 1998. PNNL-11914, Vol. 6, Pacific Northwest National Laboratory, Richland, Washington.

Volume 7: Exit Level Processor Documentation. 1998. PNNL-11914, Vol. 7, Pacific Northwest National Laboratory, Richland, Washington.

Volume 8: Specifications. 1998. PNNL-11914, Vol. 8, Pacific Northwest National Laboratory, Richland, Washington.

Volume 9: Software Development and Testing Strategies. 1998. PNNL-11914, Vol. 9, Pacific Northwest National Laboratory, Richland, Washington.

Volume 10: Facilitating Dynamic Link Libraries. 1998. PNNL-11914, Vol. 10, Pacific Northwest National Laboratory, Richland, Washington.

Volume 11: User's Guidance. 1998. PNNL-11914, Vol. 11, Pacific Northwest National Laboratory, Richland, Washington.

Volume 12: Dictionary. 1998. PNNL-11914, Vol. 12, Pacific Northwest National Laboratory, Richland, Washington.

Volume 13: Chemical Properties Processor Documentation. 1998. PNNL-11914, Vol. 13, Pacific Northwest National Laboratory, Richland, Washington.

Volume 14: Site Layout Processor Documentation. 1998. PNNL-11914, Vol. 14, Pacific Northwest National Laboratory, Richland, Washington.

Volume 15: Risk Visualization Tool Documentation. 1998. PNNL-11914, Vol. 15, Pacific Northwest National Laboratory, Richland, Washington.

Quality Assurance Program Document

Gelston, G. M., R. E. Lundgren, J. P. McDonald, and B. L. Hoopes. 1998. *An Approach to Ensuring Quality in Environmental Software*. PNNL-11880, Pacific Northwest National Laboratory, Richland, Washington.

Additional References

Office of Civilian Radioactive Waste Management (OCRWM). 1995. *Quality Assurance Requirements and Description, Software*. U.S. Department of Energy, Washington, D.C.

U.S. Environmental Protection Agency (EPA). 1997. *System Design and Development Guidance*. EPA Directive Number 2182, Washington, D.C.

Appendix A

Additional Testing Information for the HWIR Input/Output Dynamic Link Library

Appendix A--Additional Testing Information for the HWIR Input/Output Dynamic Link Library

Dynamic Link Libraries (DLLs) are called by programs. Thus, to test HWIRIO.DLL, a small program is used that performs the function calls to the DLL specified in a test-specific input file. This appendix describes the general procedure for conducting each test and includes a description of the input file to the test program. The FORTRAN-90 and C++ versions of the test program are listed in this appendix.

A.1 General Procedure for Test Case Implementation

To run a test case, batch files are used. An example batch file is listed below for the Borland C++ test program:

```
echo The call arguments are:
set ARGUMENTS=1:05pm 03/30/1998 \HWIR\TESTING\IODLL\TESTING\ssf
\HWIR\TESTING\IODLL\TESTING\grf 1 hd.ssf hd.grf[e]
echo %ARGUMENTS%
del %1c.out
del %1.out
cdlltst %1
copy %1.out %1c.out
cd \HWIR\TESTING\IODLL\TESTING\grf
copy hd.grf %1c.grf
cd \HWIR\TESTING\IODLL\TESTING\ssf
```

This batch file first sets the ARGUMENTS environment variable. The arguments are the time, date, path to the location of the SSF data group files, and path to the location of the GRF data group files. Then, the data group files themselves are listed. All the files can be read from, but only the last file can be written to. The last file is opened with write access, so if it is an existing file, its contents are erased when it is opened. The next line lists (or echos) the ARGUMENTS to the screen. Then, pre-existing output files from the test program are deleted (%1 is the test case name passed to the batch file as an argument). The test program itself is then run (cdlltst), passing the test case name (the .tst extension is added by the test program). Then the output file from the test program is copied with a new name having a "c" as the last character to designate the file as output from the Borland C++ test program. Finally, this particular batch file changes to the GRF directory and copies a data group file to a test-specific name. The batch files used to test the Lahey FORTRAN-90 and Digital Visual FORTRAN-90 compiled executables differ only on line seven, where iodlltst is called instead of cdlltst. The batch file used to test the Microsoft® Visual C++ compiled executable differs only on line seven, where mscpp5 is called instead of cdlltst.

To run a batch file in Windows® 95, have the batch file, test program executables, and test program input file in the same directory (the preceding example is run from the SSF directory). Double

[e] This line (line #3) is actually a continuation of the previous line (line #2) that has wrapped around.

click on “My Computer” and navigate to the location of the batch file. Select the batch file by right clicking once, and select “properties” at the bottom of the pop-up menu. Select the “program” tab in the window that appears, and at the end of the “Cmd line,” type in the test case name or just enter a question mark (in which case, you will be prompted for the argument list—which is just the test case name—when you run the batch file). Then click “OK,” and then double click on the batch file name to run the case. The test program generates an output file (test case name with the extension “out”) that shows which functions or subroutines were called and the values returned (if any).

A.1.1 Test Program Input File Format

The general format of the test program input file is similar to the format of other flat-ASCII files used in the FRAMES-HWIR Technology Software System. Strings are denoted by double quotes, and values are comma separated. Each record in the input file consists of the name of the function or subroutine in the DLL to call (as a string in double quotes), followed on the next line by values forming an argument list to pass to the function or subroutine (comma separated) if required. The last line of the input file must contain the string “Stop,” to tell the test program the end of the input file has been reached.

This is the input file for test case IODL_7:

“OpenGroups”
“ReadReal”
“Ee.ssf”,“BW_s”,“kg”, ; should return 52.4
“ReadInt”
“hd.ssf”,“Int”,“days”, ; should return 20
“ReadReal”,
“Ar.ssf”,“fv”,“”, ; should return 52.4
“CloseGroups”
“Stop”

The first line in the preceding example directs the test program to call subroutine OpenGroups. The subroutine has no argument, so no argument line follows. The second line directs the test program to call ReadReal, and an argument list follows. The first argument is the data group file to read a real, the second is the variable name to read, and the third is the units designation for the variable. A comment statement is allowed in the input file after all arguments have been listed. In this case, the expected value to be returned is stated. The fourth and fifth lines also call ReadInt with another argument list. The sixth and seventh lines call ReadReal with its argument list. Finally, CloseGroups is called with no argument list, and then the end of file designator is listed. The Data Dictionary associated with each data group file is determined by the first two letters of the data group filename. For data group file EE.SSF, Data Dictionary EE.DIC is used.

A.2 FORTRAN-90 Test Program Listing

This section lists the FORTRAN-90 version of the test program used to prepare the Lahey FORTRAN-90 and the Digital Visual FORTRAN-90 compiled executables.

IODLLTST.F90

```
program IODLLTst

!     DLL function types
!     THIS MUST BE INCLUDED FOR THINGS TO WORK
        use lf90hwirio

        Character Key
        Integer*2 I1,I2,I3,I4,I5,I6,RetI
        Logical*1 RetL
        Integer IOCheck1,IOCheck2
        Real*8 RetR
        CHARACTER*80 TestName
        CHARACTER*20 FunName,DataGroup
        Character*20 VarName,Units
        CHARACTER*80 Junk,RetS

        CALL GetCL(TESTNAME)
        OPEN (8, FILE=Trim(TESTNAME) // ".TST", STATUS="OLD", &
     &        IOSTAT=IOCheck1,Delim="Apostrophe",pad="NO")
        OPEN (9, FILE=trim(TESTNAME) // ".OUT", STATUS="NEW", &
     &        IOSTAT=IOCheck2)
        if (IOCheck1+IOCheck2.NE.0) stop
        do while (FunName.NE."Stop")
          Read (8,*) FunName
          Write(6,*) FunName
          units=""
          If (FunName.EQ. "Pause") Then
            Write(9,*) "Call Pause"
            Write(6,*) "Press a key then press the return key"
            Read(*",(a1)") Key

          Else If (Trim(FunName).EQ. "OpenGroups") Then
            Write(9,*) "Call OpenGroups"
            Call OpenGroups()

          Else If (FunName.EQ. "CloseGroups") Then
            Write(9,*) "Call CloseGroups"
            Call CloseGroups()

          Else if (FunName.EQ. "Error") Then
            Read(8,*) Junk
            Write(9,*) "Call Error",Junk
            Close(8)
            Close(9)
            Call Error(Junk)

          Else If (FunName.EQ. "Warning") Then
            Read(8,*) Junk
            Write(9,*) "Call Warning",Junk
            Call Warning(Junk)

          Else if (FunName.EQ."NumArgs") Then
            I1=NumArgs()
            Write(9,*) "Call NumArgs"
            Write(9,*) "Returned",I1

          Else if (FunName.EQ. "GetArgInt") Then
            Read(8,*) I1
            I2=GetArgInt(I1)
            Write(9,*) "Call GetArgInt"
            Write(9,*) "Returned",I2
```

```
          Else if (FunName.EQ. “GetArgString”) Then
            Read(8,*) I1
            Write(9,*) “Call GetArgString”,I1
            Call GetArgString(I1,Junk)
            Write(9,*) “Returned”,Junk

!****  (Section containing calls to ReadInt and ReadInt[1-6])  ******

          Else If (FunName.EQ. “ReadInt”) Then
            Read(8,*) DataGroup,VarName,Units
            Write(9,*) “Call ReadInt”,DataGroup,VarName,Units
            RetI=ReadInt(DataGroup,VarName,Units)
            Write(9,*) “Returned”, RetI

          Else If (FunName.EQ. “ReadInt1”) Then
            Read(8,*) DataGroup,VarName,Units,I1
            Write(9,*) “Call ReadInt1”,DataGroup,VarName,Units, I1
            RetI=ReadInt1(DataGroup,VarName,Units,I1)
!            RetI=fReadInt1(DataGroup,VarName,Units,CArg(I1))
            Write(9,*) “Returned”, RetI

         Else If (FunName.EQ. “ReadInt2”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2
            RetI=ReadInt2(DataGroup,VarName,Units,I1,I2)
            Write(9,*) “Call ReadInt2”,DataGroup,VarName,Units, &
                       I1,I2
            Write(9,*) “Returned”, RetI

          Else If (FunName.EQ. “ReadInt3”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3
            RetI=ReadInt3(DataGroup,VarName,Units,I1,I2,I3)
            Write(9,*) “Call ReadInt3”,DataGroup,VarName,Units, &
                       I1,I2,I3
            Write(9,*) “Returned”, RetI

         Else If (FunName.EQ. “ReadInt4”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4
            RetI=ReadInt4(DataGroup,VarName,Units,I1,I2,I3,I4)
            Write(9,*) “Call ReadInt4”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4
            Write(9,*) “Returned”, RetI

          Else If (FunName.EQ.”ReadInt5”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5
            RetI=ReadInt5(DataGroup,VarName,Units,I1,I2,I3,I4,I5)
            Write(9,*) “Call ReadInt5”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5
            Write(9,*) “Returned”, RetI

          Else If (FunName.EQ. “ReadInt6”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6
            RetI=ReadInt6(DataGroup,VarName,Units,I1,I2,         &
                          I3,I4,I5,I6)
            Write(9,*) “Call ReadInt6”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5,I6
            Write(9,*) “Returned “, RetI

!****  (Section containing calls to ReadReal and ReadReal[1-6])  ******

          Else If (FunName.EQ. “ReadReal”) Then
            Read(8,*) DataGroup,VarName,Units
            RetR=ReadReal(DataGroup,VarName,Units)
            Write(9,*) “Call ReadReal”,DataGroup,VarName,Units
            Write(9,*) “Returned”,RetR
```

```
          Else If (FunName.EQ. “ReadReal1”) Then
            Read(8,*) DataGroup,VarName,Units,I1
            RetR=ReadReal1(DataGroup,VarName,Units,I1)
            Write(9,*) “Call ReadReal1”,DataGroup,VarName,Units,I1
            Write(9,*) “Returned”, RetR

          Else If (FunName.EQ. “ReadReal2”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2
            RetR=ReadReal2(DataGroup,VarName,Units,I1,I2)
            Write(9,*) “Call ReadReal2”,DataGroup,VarName,Units,I1,I2
            Write(9,*) “Returned “, RetR

          Else If (FunName.EQ. “ReadReal3”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3
            RetR=ReadReal3(DataGroup,VarName,Units,I1,I2,I3)
            Write(9,*) “Call ReadReal3”,DataGroup,VarName,Units, &
                       I1,I2,I3
            Write(9,*) “Returned”, RetR

          Else If (FunName.EQ. “ReadReal4”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4
            RetR=ReadReal4(DataGroup,VarName,Units,I1,I2,I3,I4)
            Write(9,*) “Call ReadReal4”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4
            Write(9,*) “Returned”, RetR

          Else If (FunName.EQ. “ReadReal5” Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5
            RetR=ReadReal5(DataGroup,VarName,Units,I1,I2,         &
                           I3,I4,I5)
            Write(9,*) “Call ReadReal5”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5
            Write(9,*) “Returned”, RetR

          Else If (FunName.EQ. “ReadReal6”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6
            RetR=ReadReal6(DataGroup,VarName,Units,I1,I2,         &
                           I3,I4,I5,I6)
            Write(9,*) “Call ReadReal6”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5,I6
            Write(9,*) “Returned”, RetR

!****  (Section containing calls to ReadLog and ReadLog[1-6])  ******

          Else If (FunName.EQ. “ReadLog”) Then
            Read(8,*) DataGroup,VarName,Units
            RetL=ReadLog(DataGroup,VarName,Units)
            Write(9,*) “Call ReadLog”,DataGroup,VarName,Units
            Write(9,*) “Returned”, RetL

          Else If (FunName.EQ. “ReadLog1”) Then
            Read(8,*) DataGroup,VarName,Units,I1
            RetL=ReadLog1(DataGroup,VarName,Units,I1)
            Write(9,*) “Call ReadLog1”,DataGroup,VarName,Units,I1
            Write(9,*) “Returned”, RetL

          Else If (FunName.EQ. “ReadLog2”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2
            RetL=ReadLog2(DataGroup,VarName,Units,I1,I2)
            Write(9,*) “Call ReadLog2”,DataGroup,VarName,Units, &
                       I1,I2
            Write(9,*) “Returned”, RetL

          Else If (FunName.EQ. “ReadLog3”) Then
```

```
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3
            RetL=ReadLog3(DataGroup,VarName,Units,I1,I2,I3)
            Write(9,*) "Call ReadLog3",DataGroup,VarName,Units, &
                       I1,I2,I3
            Write(9,*) "Returned", RetL

          Else If (FunName.EQ. "ReadLog4") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4
            RetL=ReadLog4(DataGroup,VarName,Units,I1,I2,I3,I4)
            Write(9,*) "Call ReadLog4",DataGroup,VarName,Units, &
                       I1,I2,I3,I4
            Write(9,*) "Returned", RetL

          Else If (FunName.EQ. "ReadLog5") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5
            RetL=ReadLog5(DataGroup,VarName,Units,I1,I2,         &
                          I3,I4,I5)
            Write(9,*) "Call ReadLog5",DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5
            Write(9,*) "Returned", RetL

          Else If (FunName.EQ. "ReadLog6") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6
            RetL=ReadLog6(DataGroup,VarName,Units,I1,I2,         &
                          I3,I4,I5,I6)
            Write(9,*) "Call ReadLog6",DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5,I6
            Write(9,*) "Returned", RetL

!****  (Section containing calls to Readstring and ReadString[1-6])  ******

          Else If (FunName.EQ. "ReadString") Then
            Read(8,*) DataGroup,VarName,Units
            CaLL ReadString(DataGroup,VarName,Units,RetS)
            Write(9,*) "Call ReadString",DataGroup,VarName,Units
            Write(9,*) "Returned ", RetS

          Else If (FunName.EQ. "ReadString1") Then
            Read(8,*) DataGroup,VarName,Units,I1
            CaLL ReadString1(DataGroup,VarName,Units,I1,RetS)
            Write(9,*) "Call ReadString1",DataGroup,VarName,Units,I1
            Write(9,*) "Returned", RetS

          Else If (FunName.EQ. "ReadString2") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2
            CaLL ReadString2(DataGroup,VarName,Units,I1,I2,RetS)
            Write(9,*) "Call ReadString2",DataGroup,VarName,Units, &
                       I1,I2
            Write(9,*) "Returned", RetS

          Else If (FunName.EQ."ReadString3") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3
            CaLL ReadString3(DataGroup,VarName,Units,I1,I2,I3,RetS)
            Write(9,*) "Call ReadString3",DataGroup,VarName,Units, &
                       I1,I2,I3
            Write(9,*) "Returned", RetS

          Else If (FunName.EQ. "ReadString4") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4
            CaLL ReadString4(DataGroup,VarName,Units,I1,I2,I3,I4,RetS)
            Write(9,*) "Call ReadString4",DataGroup,VarName,Units, &
                       I1,I2,I3,I4
            Write(9,*) "Returned", RetS
```

```
          Else If (FunName.EQ. “ReadString5”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5
            CaLL ReadString5(DataGroup,VarName,Units,I1,I2,         &
                             I3,I4,I5,RetS)
            Write(9,*) “Call ReadString5”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5
            Write(9,*) “Returned”, RetS

          Else If (FunName.EQ. “ReadString6”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6
            CaLL ReadString6(DataGroup,VarName,Units,I1,I2,         &
                             I3,I4,I5,I6,RetS)
            Write(9,*) “Call ReadString6”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5,I6
            Write(9,*) “Returned”, RetS

!****  (Section containing calls to WriteInt and WriteInt[1-6])  ******

          Else If (FunName.EQ. “WriteInt”) Then
            Read(8,*) DataGroup,VarName,Units,RetI
            CaLL WriteInt(DataGroup,VarName,Units,RetI)
            Write(9,*) “Call WriteInt”,DataGroup,VarName,Units,RetI

          Else If (FunName.EQ. “WriteInt1”) Then
            Read(8,*) DataGroup,VarName,Units,I1,RetI
            CaLL WriteInt1(DataGroup,VarName,Units,I1,RetI)
            Write(9,*) “Call WriteInt1”,DataGroup,VarName,Units,  &
                       I1,RetI

          Else If (FunName.EQ. “WriteInt2”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,RetI
            CaLL WriteInt2(DataGroup,VarName,Units,I1,I2,RetI)
            Write(9,*) “Call WriteInt2”,DataGroup,VarName,Units,  &
                       I1,I2,RetI

          Else If (FunName.EQ. “WriteInt3”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,RetI
            CaLL WriteInt3(DataGroup,VarName,Units,I1,I2,I3,RetI)
            Write(9,*) “Call WriteInt3”,DataGroup,VarName,Units,  &
                       I1,I2,I3,RetI

          Else If (FunName.EQ. “WriteInt4”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,RetI
            CaLL WriteInt4(DataGroup,VarName,Units,I1,I2,I3,I4,RetI)
            Write(9,*) “Call WriteInt4 “,DataGroup,VarName,Units,  &
                       I1,I2,I3,I4,RetI

          Else If (FunName.EQ. “WriteInt5”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,RetI
            CaLL WriteInt5(DataGroup,VarName,Units,I1,I2,          &
                           I3,I4,I5,RetI)
            Write(9,*) “Call WriteInt5”,DataGroup,VarName,Units,  &
                       I1,I2,I3,I4,I5,RetI

          Else If (FunName.EQ. “WriteInt6”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6,RetI
            CaLL WriteInt6(DataGroup,VarName,Units,I1,I2,          &
                           I3,I4,I5,I6,RetI)
            Write(9,*) “Call WriteInt6”,DataGroup,VarName,Units,  &
                       I1,I2,I3,I4,I5,I6,RetI

!****  (Section containing calls to WriteReal and WriteReal[1-6])  ******

          Else If (FunName.EQ. “WriteReal”) Then
```

```
            Read(8,*) DataGroup,VarName,Units,RetR
            CaLL WriteReal(DataGroup,VarName,Units,RetR)
            Write(9,*) “Call WriteReal”,DataGroup,VarName,Units,RetR

          Else If (FunName.EQ. “WriteReal1” Then
            Read(8,*) DataGroup,VarName,Units,I1,RetR
            CaLL WriteReal1(DataGroup,VarName,Units,I1,RetR)
            Write(9,*) “Call WriteReal1”,DataGroup,VarName,Units, &
                       I1,RetR

          Else If (FunName.EQ. “WriteReal2”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,RetR
            CaLL WriteReal2(DataGroup,VarName,Units,I1,I2,RetR)
            Write(9,*) “Call WriteReal2”,DataGroup,VarName,Units, &
                       I1,I2,RetR

          Else If (FunName.EQ. “WriteReal3") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,RetR
            CaLL WriteReal3(DataGroup,VarName,Units,I1,I2,         &
                            I3,RetR)
            Write(9,*) “Call WriteReal3”,DataGroup,VarName,Units, &
                       I1,I2,I3,RetR

          Else If (FunName.EQ. “WriteReal4”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,RetR
            CaLL WriteReal4(DataGroup,VarName,Units,I1,I2,         &
                            I3,I4,RetR)
            Write(9,*) “Call WriteReal4”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,RetR

          Else If (FunName.EQ. “WriteReal5”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,RetR
            CaLL WriteReal5(DataGroup,VarName,Units,I1,I2,         &
                            I3,I4,I5,RetR)
            Write(9,*) “Call WriteReal5”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5,RetR

          Else If (FunName.EQ. “WriteReal6’) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6,RetR
            CaLL WriteReal6(DataGroup,VarName,Units,I1,I2,         &
                            I3,I4,I5,I6,RetR)
            Write(9,*) “Call WriteReal6”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5,I6,RetR

!****  (Section containing calls to WriteLog and WriteLog[1-6])  ******

          Else If (FunName.EQ. “WriteLog”) Then
            Read(8,*) DataGroup,VarName,Units,RetL
            CaLL WriteLog(DataGroup,VarName,Units,RetL)
            Write(9,*) “Call WriteLog”,DataGroup,VarName,Units,RetL

          Else If (FunName.EQ. “WriteLog1”) Then
            Read(8,*) DataGroup,VarName,Units,I1,RetL
            CaLL WriteLog1(DataGroup,VarName,Units,I1,RetL)
            Write(9,*) “Call WriteLog1”,DataGroup,VarName,Units,  &
                        I1,RetL

          Else If (FunName.EQ. “WriteLog2”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,RetL
            CaLL WriteLog2(DataGroup,VarName,Units,I1,I2,RetL)
            Write(9,*) “Call WriteLog2”,DataGroup,VarName,Units,  &
                       I1,I2,RetL

          Else If (FunName.EQ. “WriteLog3”) Then
```

```
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,RetL
            CaLL WriteLog3(DataGroup,VarName,Units,I1,I2,          &
                           I3,RetL)
            Write(9,*) "Call WriteLog3",DataGroup,VarName,Units,  &
                        I1,I2,I3,RetL

          Else If (FunName.EQ. "WriteLog4") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,RetL
            CaLL WriteLog4(DataGroup,VarName,Units,I1,I2,          &
                           I3,I4,RetL)
            Write(9,*) "Call WriteLog4",DataGroup,VarName,Units,  &
                       I1,I2,I3,I4,RetL

          Else If (FunName.EQ. "WriteLog5") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,RetL
            CaLL WriteLog5(DataGroup,VarName,Units,I1,I2,          &
                           I3,I4,I5,RetL)
            Write(9,*) "Call WriteLog5",DataGroup,VarName,Units,  &
                       I1,I2,I3,I4,I5,RetL

          Else If (FunName.EQ. "WriteLog6") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6,RetL
            CaLL WriteLog6(DataGroup,VarName,Units,I1,I2,          &
                           I3,I4,I5,I6,RetL)
            Write(9,*) "Call WriteLog6",DataGroup,VarName,Units,  &
                       I1,I2,I3,I4,I5,I6,RetL

!****  (Section containing calls to WriteString and WriteString[1-6])  ******

          Else If (FunName.EQ. "WriteString") Then
            Read(8,*) DataGroup,VarName,Units,RetS
            CaLL WriteString(DataGroup,VarName,Units,RetS)
            Write(9,*) "Call WriteString",DataGroup,VarName,Units,RetS

          Else If (FunName.EQ. "WriteString1") Then
            Read(8,*) DataGroup,VarName,Units,I1,RetS
            CaLL WriteString1(DataGroup,VarName,Units,I1,RetS)
            Write(9,*) "Call WriteString1",DataGroup,VarName,Units, &
                       I1,RetS

          Else If (FunName.EQ. "WriteString2") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,RetS
            CaLL WriteString2(DataGroup,VarName,Units,I1,I2,RetS)
            Write(9,*) "Call WriteString2",DataGroup,VarName,Units, &
                       I1,I2,RetS

          Else If (FunName.EQ. "WriteString3") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,RetS
            CaLL WriteString3(DataGroup,VarName,Units,I1,I2,         &
                              I3,RetS)
            Write(9,*) "Call WriteString3",DataGroup,VarName,Units, &
                       I1,I2,I3,RetS

          Else If (FunName.EQ. "WriteString4") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,RetS
            CaLL WriteString4(DataGroup,VarName,Units,I1,I2,         &
                              I3,I4,RetS)
            Write(9,*) "Call WriteString4",DataGroup,VarName,Units, &
                       I1,I2,I3,I4,RetS

          Else If (FunName.EQ. "WriteString5") Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,RetS
            CaLL WriteString5(DataGroup,VarName,Units,I1,I2,         &
                              I3,I4,I5,RetS)
```

```
            Write(9,*) “Call WriteString5”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5,RetS

          Else If (FunName.EQ. “WriteString6”) Then
            Read(8,*) DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6,RetS
            CaLL WriteString6(DataGroup,VarName,Units,I1,I2,         &
                              I3,I4,I5,I6,RetS)
            Write(9,*) “Call WriteString6”,DataGroup,VarName,Units, &
                       I1,I2,I3,I4,I5,I6,RetS
          End If
        end do
        Close(8)
        Close(9)
      End
```

The following file is needed to prepare an executable using the Lahey FORTRAN-90 Version 4.0 compiler:

LAHEY.F90

```
module lf90hwirio
        implicit none
        DLL_Import fNumArgs
        DLL_Import fGetArgInt
        DLL_Import fGetArgString
        DLL_Import fOpenGroups
        DLL_Import fCloseGroups
        DLL_Import fError
        DLL_Import fWarning
        DLL_Import fReadInt
        DLL_Import fReadInt1
        DLL_Import fReadInt2
        DLL_Import fReadInt3
        DLL_Import fReadInt4
        DLL_Import fReadInt5
        DLL_Import fReadInt6
        DLL_Import fReadReal
        DLL_Import fReadReal1
        DLL_Import fReadReal2
        DLL_Import fReadReal3
        DLL_Import fReadReal4
        DLL_Import fReadReal5
        DLL_Import fReadReal6
        DLL_Import fReadLog
        DLL_Import fReadLog1
        DLL_Import fReadLog2
        DLL_Import fReadLog3
        DLL_Import fReadLog4
        DLL_Import fReadLog5
        DLL_Import fReadLog6
        DLL_Import fReadString
        DLL_Import fReadString1
        DLL_Import fReadString2
        DLL_Import fReadString3
        DLL_Import fReadString4
        DLL_Import fReadString5
        DLL_Import fReadString6
        DLL_Import fWriteInt
        DLL_Import fWriteInt1
        DLL_Import fWriteInt2
        DLL_Import fWriteInt3
        DLL_Import fWriteInt4
```

```
DLL_Import fWriteInt5
DLL_Import fWriteInt6
DLL_Import fWriteReal
DLL_Import fWriteReal1
DLL_Import fWriteReal2
DLL_Import fWriteReal3
DLL_Import fWriteReal4
DLL_Import fWriteReal5
DLL_Import fWriteReal6
DLL_Import fWriteLog
DLL_Import fWriteLog1
DLL_Import fWriteLog2
DLL_Import fWriteLog3
DLL_Import fWriteLog4
DLL_Import fWriteLog5
DLL_Import fWriteLog6
DLL_Import fWriteString
DLL_Import fWriteString1
DLL_Import fWriteString2
DLL_Import fWriteString3
DLL_Import fWriteString4
DLL_Import fWriteString5
DLL_Import fWriteString6
integer*2 fNumArgs
integer*2 fGetArgInt
integer*2 fReadInt
integer*2 fReadInt1
integer*2 fReadInt2
integer*2 fReadInt3
integer*2 fReadInt4
integer*2 fReadInt5
integer*2 fReadInt6
Real*8    fReadReal
Real*8    fReadReal1
Real*8    fReadReal2
Real*8    fReadReal3
Real*8    fReadReal4
Real*8    fReadReal5
Real*8    fReadReal6
logical*1 fReadLog
logical*1 fReadLog1
logical*1 fReadLog2
logical*1 fReadLog3
logical*1 fReadLog4
logical*1 fReadLog5
logical*1 fReadLog6
contains

  integer function NumArgs()
    NumArgs=fNumArgs()
  end function NumArgs

  integer*2 function GetArgInt(ArgIndex)
    integer*2 ArgIndex
    intent(in) :: ArgIndex
    GetArgInt=fGetArgInt(CArg(ArgIndex))
  end function

  subroutine GetArgString(ArgIndex,Argument)
    integer*2 ArgIndex
    intent(in) :: ArgIndex
    character*80, intent(out) :: Argument
    Call fGetArgString(CArg(ArgIndex),Argument)
  end subroutine
```

```
          subroutine OpenGroups()
            Call fOpenGroups()
          end subroutine

          subroutine CloseGroups()
            Call fCloseGroups()
          end subroutine

          subroutine Error(Description)
            character*80, intent(in) :: Description
            Call fError(Description)
          end subroutine

          subroutine Warning(Description)
            character*80, intent(in) :: Description
            Call fWarning(Description)
          end subroutine

!////////////////////////////////////////////////////////////////////////////
!//   routines for reading integer variables
!////////////////////////////////////////////////////////////////////////////

          integer*2 function ReadInt(group,var,unit)
            character*20, intent(in) :: group,var,unit
            ReadInt=fReadInt(group,var,unit)
          end function ReadInt

          integer*2 function ReadInt1(group,var,unit,i1)
            character*20, intent(in) :: group,var,unit
            integer*2 i1
            intent(in) :: i1
            ReadInt1=fReadInt1(group,var,unit,CArg(i1))
          end function ReadInt1

          integer*2 function ReadInt2(group,var,unit,i1,i2)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2
            intent(in) :: i1,i2
            ReadInt2=fReadInt2(group,var,unit,CArg(i1),CArg(i2))
          end function ReadInt2

          integer*2 function ReadInt3(group,var,unit,i1,i2,i3)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3
            intent(in) :: i1,i2,i3
            ReadInt3=fReadInt3(group,var,unit,CArg(i1),CArg(i2),CArg(i3))
          end function ReadInt3

          integer*2 function ReadInt4(group,var,unit,i1,i2,i3,i4)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4
            intent(in) :: i1,i2,i3,i4
            ReadInt4=fReadInt4(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                               ,CArg(i4))
          end function ReadInt4

          integer*2 function ReadInt5(group,var,unit,i1,i2,i3,i4,i5)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4,i5
            intent(in) :: i1,i2,i3,i4,i5
            ReadInt5=fReadInt5(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                               ,CArg(i4),CArg(i5))
          end function ReadInt5
```

```
          integer*2 function ReadInt6(group,var,unit,i1,i2,i3,i4,i5,i6)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4,i5,i6
            intent(in) :: i1,i2,i3,i4,i5,i6
            ReadInt6=fReadInt6(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                              ,CArg(i4),CArg(i5),CArg(i6))
          end function ReadInt6

!////////////////////////////////////////////////////////////////////////
!//   routines for reading real(double); variables
!////////////////////////////////////////////////////////////////////////

          Real*8 function ReadReal(group,var,unit)
            character*20, intent(in) :: group,var,unit
            ReadReal=fReadReal(group,var,unit)
          end function ReadReal

          Real*8 function ReadReal1(group,var,unit,i1)
            character*20, intent(in) :: group,var,unit
            integer*2 i1
            intent(in) :: i1
            ReadReal1=fReadReal1(group,var,unit,CArg(i1))
          end function ReadReal1

          Real*8 function ReadReal2(group,var,unit,i1,i2)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2
            intent(in) :: i1,i2
            ReadReal2=fReadReal2(group,var,unit,CArg(i1),CArg(i2))
          end function ReadReal2

          Real*8 function ReadReal3(group,var,unit,i1,i2,i3)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3
            intent(in) :: i1,i2,i3
            ReadReal3=fReadReal3(group,var,unit,CArg(i1),CArg(i2),CArg(i3))
          end function ReadReal3

          Real*8 function ReadReal4(group,var,unit,i1,i2,i3,i4)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4
            intent(in) :: i1,i2,i3,i4
            ReadReal4=fReadReal4(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                                ,CArg(i4))
          end function ReadReal4

          Real*8 function ReadReal5(group,var,unit,i1,i2,i3,i4,i5)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4,i5
            intent(in) :: i1,i2,i3,i4,i5
            ReadReal5=fReadReal5(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                                ,CArg(i4),CArg(i5))
          end function ReadReal5

          Real*8 function ReadReal6(group,var,unit,i1,i2,i3,i4,i5,i6)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4,i5,i6
            intent(in) :: i1,i2,i3,i4,i5,i6
            ReadReal6=fReadReal6(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                                ,CArg(i4),CArg(i5),CArg(i6))
          end function ReadReal6

!////////////////////////////////////////////////////////////////////////
!//   routines for reading logical variables
```

```
!////////////////////////////////////////////////////////////////////////////

          Logical*1 function ReadLog(group,var,unit)
            character*20, intent(in) :: group,var,unit
            ReadLog=fReadLog(group,var,unit)
          end function ReadLog

          Logical*1 function ReadLog1(group,var,unit,i1)
            character*20, intent(in) :: group,var,unit
            integer*2 i1
            intent(in) :: i1
            ReadLog1=fReadLog1(group,var,unit,CArg(i1))
          end function ReadLog1

          Logical*1 function ReadLog2(group,var,unit,i1,i2)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2
            intent(in) :: i1,i2
            ReadLog2=fReadLog2(group,var,unit,CArg(i1),CArg(i2))
          end function ReadLog2

          Logical*1 function ReadLog3(group,var,unit,i1,i2,i3)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3
            intent(in) :: i1,i2,i3
            ReadLog3=fReadLog3(group,var,unit,CArg(i1),CArg(i2),CArg(i3))
          end function ReadLog3

          Logical*1 function ReadLog4(group,var,unit,i1,i2,i3,i4)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4
            intent(in) :: i1,i2,i3,i4
            ReadLog4=fReadLog4(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                               ,CArg(i4))
          end function ReadLog4

          Logical*1 function ReadLog5(group,var,unit,i1,i2,i3,i4,i5)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4,i5
            intent(in) :: i1,i2,i3,i4,i5
            ReadLog5=fReadLog5(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                               ,CArg(i4),CArg(i5))
          end function ReadLog5

          Logical*1 function ReadLog6(group,var,unit,i1,i2,i3,i4,i5,i6)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4,i5,i6
            intent(in) :: i1,i2,i3,i4,i5,i6
            ReadLog6=fReadLog6(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                               ,CArg(i4),CArg(i5),CArg(i6))
          end function ReadLog6

!////////////////////////////////////////////////////////////////////////////
!//   routines for reading string variables
!////////////////////////////////////////////////////////////////////////////

          subroutine ReadString(group,var,unit,str)
            character*20, intent(in) :: group,var,unit
            character*80, intent(out) :: str
            Call fReadString(group,var,unit,str)
          end subroutine ReadString

          subroutine ReadString1(group,var,unit,i1,str)
            character*20, intent(in) :: group,var,unit
```

```
  integer*2 i1
  intent(in) :: i1
  character*80, intent(out) :: str
  Call fReadString1(group,var,unit,CArg(i1),str)
end subroutine ReadString1

subroutine ReadString2(group,var,unit,i1,i2,str)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2
  intent(in) :: i1,i2
  character*80, intent(out) :: str
  Call fReadString2(group,var,unit,CArg(i1),CArg(i2),str)
end subroutine ReadString2

subroutine ReadString3(group,var,unit,i1,i2,i3,str)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3
  intent(in) :: i1,i2,i3
  character*80, intent(out) :: str
  Call fReadString3(group,var,unit,CArg(i1),CArg(i2),CArg(i3),str)
end subroutine ReadString3

subroutine ReadString4(group,var,unit,i1,i2,i3,i4,str)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3,i4
  intent(in) :: i1,i2,i3,i4
  character*80, intent(out) :: str
  Call fReadString4(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                    ,CArg(i4),str)
end subroutine ReadString4

subroutine ReadString5(group,var,unit,i1,i2,i3,i4,i5,str)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3,i4,i5
  intent(in) :: i1,i2,i3,i4,i5
  character*80, intent(out) :: str
  Call fReadString5(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                   ,CArg(i4),CArg(i5),str)
end subroutine ReadString5

subroutine ReadString6(group,var,unit,i1,i2,i3,i4,i5,i6,str)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3,i4,i5,i6
  intent(in) :: i1,i2,i3,i4,i5,i6
  character*80, intent(out) :: str
  Call fReadString6(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                    ,CArg(i4),CArg(i5),CArg(i6),str)
end subroutine ReadString6


!////////////////////////////////////////////////////////////////////////////
!//  Export routines for writing integer variables
!////////////////////////////////////////////////////////////////////////////

subroutine WriteInt(group,var,unit,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 pval
  intent(in) :: pval
  Call fWriteInt(group,var,unit,CArg(pval))
end subroutine WriteInt

subroutine WriteInt1(group,var,unit,i1,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1
```

```
  intent(in) :: i1
  integer*2 pval
  intent(in) :: pval
  Call fWriteInt1(group,var,unit,CArg(i1),CArg(pval))
end subroutine WriteInt1

subroutine WriteInt2(group,var,unit,i1,i2,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2
  intent(in) :: i1,i2
  integer*2 pval
  intent(in) :: pval
  Call fWriteInt2(group,var,unit,CArg(i1),CArg(i2),CArg(pval))
end subroutine WriteInt2

subroutine WriteInt3(group,var,unit,i1,i2,i3,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3
  intent(in) :: i1,i2,i3
  integer*2 pval
  intent(in) :: pval
  Call fWriteInt3(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                  ,CArg(pval))
end subroutine WriteInt3

subroutine WriteInt4(group,var,unit,i1,i2,i3,i4,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3,i4
  intent(in) :: i1,i2,i3,i4
  integer*2 pval
  intent(in) :: pval
  Call fWriteInt4(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                   ,CArg(i4),CArg(pval))
end subroutine WriteInt4

subroutine WriteInt5(group,var,unit,i1,i2,i3,i4,i5,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3,i4,i5
  intent(in) :: i1,i2,i3,i4,i5
  integer*2 pval
  intent(in) :: pval
  Call fWriteInt5(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                   ,CArg(i4),CArg(i5),CArg(pval))
end subroutine WriteInt5

subroutine WriteInt6(group,var,unit,i1,i2,i3,i4,i5,i6,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3,i4,i5,i6
  intent(in) :: i1,i2,i3,i4,i5,i6
  integer*2 pval
  intent(in) :: pval
  Call fWriteInt6(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                   ,CArg(i4),CArg(i5),CArg(i6),CArg(pval))
end subroutine WriteInt6

!////////////////////////////////////////////////////////////////////////////
!//  Export routines for writing real(double); variables
!////////////////////////////////////////////////////////////////////////////

subroutine WriteReal(group,var,unit,pval)
  character*20, intent(in) :: group,var,unit
  Real*8 pval
  intent(in) :: pval
  Call fWriteReal(group,var,unit,CArg(pval))
```

```
            end subroutine WriteReal

            subroutine WriteReal1(group,var,unit,i1,pval)
              character*20, intent(in) :: group,var,unit
              integer*2 i1
              intent(in) :: i1
              Real*8 pval
              intent(in) :: pval
              Call fWriteReal1(group,var,unit,CArg(i1),CArg(pval))
            end subroutine WriteReal1

            subroutine WriteReal2(group,var,unit,i1,i2,pval)
              character*20, intent(in) :: group,var,unit
              integer*2 i1,i2
              intent(in) :: i1,i2
              Real*8 pval
              intent(in) :: pval
              Call fWriteReal2(group,var,unit,CArg(i1),CArg(i2),CArg(pval))
            end subroutine WriteReal2

            subroutine WriteReal3(group,var,unit,i1,i2,i3,pval)
              character*20, intent(in) :: group,var,unit
              integer*2 i1,i2,i3
              intent(in) :: i1,i2,i3
              Real*8 pval
              intent(in) :: pval
              Call fWriteReal3(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                              ,CArg(pval))
            end subroutine WriteReal3

            subroutine WriteReal4(group,var,unit,i1,i2,i3,i4,pval)
              character*20, intent(in) :: group,var,unit
              integer*2 i1,i2,i3,i4
              intent(in) :: i1,i2,i3,i4
              Real*8 pval
              intent(in) :: pval
              Call fWriteReal4(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                               ,CArg(i4),CArg(pval))
            end subroutine WriteReal4

            subroutine WriteReal5(group,var,unit,i1,i2,i3,i4,i5,pval)
              character*20, intent(in) :: group,var,unit
              integer*2 i1,i2,i3,i4,i5
              intent(in) :: i1,i2,i3,i4,i5
              Real*8 pval
              intent(in) :: pval
              Call fWriteReal5(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                               ,CArg(i4),CArg(i5),CArg(pval))
            end subroutine WriteReal5

            subroutine WriteReal6(group,var,unit,i1,i2,i3,i4,i5,i6,pval)
              character*20, intent(in) :: group,var,unit
              integer*2 i1,i2,i3,i4,i5,i6
              intent(in) :: i1,i2,i3,i4,i5,i6
              Real*8 pval
              intent(in) :: pval
              Call fWriteReal6(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                               ,CArg(i4),CArg(i5),CArg(i6),CArg(pval))
            end subroutine WriteReal6

!////////////////////////////////////////////////////////////////////////////
!//  Export routines for writing logical variables
!////////////////////////////////////////////////////////////////////////////
```

```
subroutine WriteLog(group,var,unit,pval)
  character*20, intent(in) :: group,var,unit
  Logical*1 pval
  intent(in) :: pval
  Call fWriteLog(group,var,unit,CArg(pval))
end subroutine WriteLog

subroutine WriteLog1(group,var,unit,i1,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1
  intent(in) :: i1
  Logical*1 pval
  intent(in) :: pval
  Call fWriteLog1(group,var,unit,CArg(i1),CArg(pval))
end subroutine WriteLog1

subroutine WriteLog2(group,var,unit,i1,i2,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2
  intent(in) :: i1,i2
  Logical*1 pval
  intent(in) :: pval
  Call fWriteLog2(group,var,unit,CArg(i1),CArg(i2),CArg(pval))
end subroutine WriteLog2

subroutine WriteLog3(group,var,unit,i1,i2,i3,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3
  intent(in) :: i1,i2,i3
  Logical*1 pval
  intent(in) :: pval
  Call fWriteLog3(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                 ,CArg(pval))
end subroutine WriteLog3

subroutine WriteLog4(group,var,unit,i1,i2,i3,i4,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3,i4
  intent(in) :: i1,i2,i3,i4
  Logical*1 pval
  intent(in) :: pval
  Call fWriteLog4(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                   ,CArg(i4),CArg(pval))
end subroutine WriteLog4

subroutine WriteLog5(group,var,unit,i1,i2,i3,i4,i5,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3,i4,i5
  intent(in) :: i1,i2,i3,i4,i5
  Logical*1 pval
  intent(in) :: pval
  Call fWriteLog5(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                   ,CArg(i4),CArg(i5),CArg(pval))
end subroutine WriteLog5

subroutine WriteLog6(group,var,unit,i1,i2,i3,i4,i5,i6,pval)
  character*20, intent(in) :: group,var,unit
  integer*2 i1,i2,i3,i4,i5,i6
  intent(in) :: i1,i2,i3,i4,i5,i6
  Logical*1 pval
  intent(in) :: pval
  Call fWriteLog6(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                   ,CArg(i4),CArg(i5),CArg(i6),CArg(pval))
end subroutine WriteLog6
```

```
!////////////////////////////////////////////////////////////////////////////
!//  Export routines for writing pvaling variables
!////////////////////////////////////////////////////////////////////////////

          subroutine WriteString(group,var,unit,str)
            character*20, intent(in) :: group,var,unit
            character*80, intent(in) :: str
            Call fWriteString(group,var,unit,str)
          end subroutine WriteString

          subroutine WriteString1(group,var,unit,i1,str)
            character*20, intent(in) :: group,var,unit
            integer*2 i1
            intent(in) :: i1
            character*80, intent(in) :: str
            Call fWriteString1(group,var,unit,CArg(i1),str)
          end subroutine WriteString1

          subroutine WriteString2(group,var,unit,i1,i2,str)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2
            intent(in) :: i1,i2
            character*80, intent(in) :: str
            Call fWriteString2(group,var,unit,CArg(i1),CArg(i2),str)
          end subroutine WriteString2

          subroutine WriteString3(group,var,unit,i1,i2,i3,str)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3
            intent(in) :: i1,i2,i3
            character*80, intent(in) :: str
            Call fWriteString3(group,var,unit,CArg(i1),CArg(i2),CArg(i3),str)
          end subroutine WriteString3

          subroutine WriteString4(group,var,unit,i1,i2,i3,i4,str)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4
            intent(in) :: i1,i2,i3,i4
            character*80, intent(in) :: str
            Call fWriteString4(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                              ,CArg(i4),str)
          end subroutine WriteString4

          subroutine WriteString5(group,var,unit,i1,i2,i3,i4,i5,str)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4,i5
            intent(in) :: i1,i2,i3,i4,i5
            character*80, intent(in) :: str
            Call fWriteString5(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                              ,CArg(i4),CArg(i5),str)
          end subroutine WriteString5

          subroutine WriteString6(group,var,unit,i1,i2,i3,i4,i5,i6,str)
            character*20, intent(in) :: group,var,unit
            integer*2 i1,i2,i3,i4,i5,i6
            intent(in) :: i1,i2,i3,i4,i5,i6
            character*80, intent(in) :: str
            Call fWriteString6(group,var,unit,CArg(i1),CArg(i2),CArg(i3) &
                              ,CArg(i4),CArg(i5),CArg(i6),str)
          end subroutine WriteString6

      end module lf90hwirio
```

The following file is needed to prepare an executable using the Digital Visual FORTRAN-90 Version 5.0 compiler:

DIGITAL.F90

```
module DVF90hwirio
      implicit none
                        interface
        integer*2 function NumArgs()
          !DEC$ Attributes C, Alias:"_dNumArgs" :: NumArgs
        end function

        integer*2 function GetArgInt(ArgIndex)
          !DEC$ Attributes C, Alias:"_dGetArgInt":: GetArgInt
          integer*2 ArgIndex
                                  !DEC$ Attributes Value::ArgIndex
          intent(in) :: ArgIndex
        end function

        subroutine GetArgString(ArgIndex,Argument)
          !DEC$ Attributes C, Alias:"_dGetArgString" :: GetArgString
          integer*2 ArgIndex
                                  !DEC$ Attributes Value::ArgIndex
          intent(in) :: ArgIndex
          character*80, intent(out) :: Argument
                                  !DEC$ Attributes Reference::Argument
        end subroutine

        subroutine OpenGroups()
          !DEC$ Attributes C, Alias:"_dOpenGroups" :: OpenGroups
        end subroutine

        subroutine CloseGroups()
          !DEC$ Attributes C, Alias:"_dCloseGroups" :: CloseGroups
        end subroutine

        subroutine Error(Description)
          !DEC$ Attributes C, Alias:"_dError" :: Error
          character*80, intent(in) :: Description
                                  !DEC$ Attributes Reference::Description
        end subroutine

        subroutine Warning(Description)
          !DEC$ Attributes C, Alias:"_dWarning" :: Warning
          character*80, intent(in) :: Description
                                  !DEC$ Attributes Reference::Description
        end subroutine

!////////////////////////////////////////////////////////////////////////////
!//   routines for reading integer variables
!////////////////////////////////////////////////////////////////////////////

        integer*2 function ReadInt(group,var,unit)
          !DEC$ Attributes C, Alias:"_dReadInt" :: ReadInt
          character*20, intent(in) :: group,var,unit
                                  !DEC$ Attributes Reference::group,var,unit
        end function ReadInt

        integer*2 function ReadInt1(group,var,unit,i1)
          !DEC$ Attributes C, Alias:"_dReadInt1" :: ReadInt1
          character*20, intent(in) :: group,var,unit
                                  !DEC$ Attributes Reference::group,var,unit
          integer*2 i1
```

```
                                !DEC$ Attributes Value:: i1
  intent(in) :: i1
end function ReadInt1

integer*2 function ReadInt2(group,var,unit,i1,i2)
  !DEC$ Attributes C, Alias:"_dReadInt2" :: ReadInt2
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2
                                !DEC$ Attributes Value:: i1,i2
  intent(in) :: i1,i2
end function ReadInt2

integer*2 function ReadInt3(group,var,unit,i1,i2,i3)
  !DEC$ Attributes C, Alias:"_dReadInt3" :: ReadInt3
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3
                                !DEC$ Attributes Value:: i1,i2,i3
  intent(in) :: i1,i2,i3
end function ReadInt3

integer*2 function ReadInt4(group,var,unit,i1,i2,i3,i4)
  !DEC$ Attributes C, Alias:"_dReadInt4" :: ReadInt4
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4
                                !DEC$ Attributes Value:: i1,i2,i3,i4
  intent(in) :: i1,i2,i3,i4
end function ReadInt4

integer*2 function ReadInt5(group,var,unit,i1,i2,i3,i4,i5)
  !DEC$ Attributes C, Alias:"_dReadInt5" :: ReadInt5
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4,i5
                                !DEC$ Attributes Value:: i1,i2,i3,i4,i5
  intent(in) :: i1,i2,i3,i4,i5
end function ReadInt5

integer*2 function ReadInt6(group,var,unit,i1,i2,i3,i4,i5,i6)
  !DEC$ Attributes C, Alias:"_dReadInt6" :: ReadInt6
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4,i5,i6
                                !DEC$ Attributes Value:: i1,i2,i3,i4,i5,i6
  intent(in) :: i1,i2,i3,i4,i5,i6
end function ReadInt6

!////////////////////////////////////////////////////////////////////////////
!//   routines for reading real(double); variables
!////////////////////////////////////////////////////////////////////////////

Real*8 function ReadReal(group,var,unit)
  !DEC$ Attributes C, Alias:"_dReadReal" :: ReadReal
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
end function ReadReal

Real*8 function ReadReal1(group,var,unit,i1)
  !DEC$ Attributes C, Alias:"_dReadReal1" :: ReadReal1
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1
```

```
                                !DEC$ Attributes Value:: i1
  intent(in) :: i1
end function ReadReal1

Real*8 function ReadReal2(group,var,unit,i1,i2)
  !DEC$ Attributes C, Alias:"_dReadReal2" :: ReadReal2
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2
                                !DEC$ Attributes Value:: i1,i2
  intent(in) :: i1,i2
end function ReadReal2

Real*8 function ReadReal3(group,var,unit,i1,i2,i3)
  !DEC$ Attributes C, Alias:"_dReadReal3" :: ReadReal3
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3
                                !DEC$ Attributes Value:: i1,i2,i3
  intent(in) :: i1,i2,i3
end function ReadReal3

Real*8 function ReadReal4(group,var,unit,i1,i2,i3,i4)
  !DEC$ Attributes C, Alias:"_dReadReal4" :: ReadReal4
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4
                                !DEC$ Attributes Value:: i1,i2,i3,i4
  intent(in) :: i1,i2,i3,i4
end function ReadReal4

Real*8 function ReadReal5(group,var,unit,i1,i2,i3,i4,i5)
  !DEC$ Attributes C, Alias:"_dReadReal5" :: ReadReal5
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4,i5
                                !DEC$ Attributes Value:: i1,i2,i3,i4,i5
  intent(in) :: i1,i2,i3,i4,i5
end function ReadReal5

Real*8 function ReadReal6(group,var,unit,i1,i2,i3,i4,i5,i6)
  !DEC$ Attributes C, Alias:"_dReadReal6" :: ReadReal6
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4,i5,i6
                                !DEC$ Attributes Value:: i1,i2,i3,i4,i5,i6
  intent(in) :: i1,i2,i3,i4,i5,i6
end function ReadReal6

!////////////////////////////////////////////////////////////////////////////
!//   routines for reading logical variables
!////////////////////////////////////////////////////////////////////////////

Logical*1 function ReadLog(group,var,unit)
  !DEC$ Attributes C, Alias:"_dReadLog" :: ReadLog
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
end function ReadLog

Logical*1 function ReadLog1(group,var,unit,i1)
  !DEC$ Attributes C, Alias:"_dReadLog1" :: ReadLog1
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1
```

```
                                !DEC$ Attributes Value:: i1
  intent(in) :: i1
end function ReadLog1

Logical*1 function ReadLog2(group,var,unit,i1,i2)
  !DEC$ Attributes C, Alias:"_dReadLog2" :: ReadLog2
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2
                                !DEC$ Attributes Value:: i1,i2
  intent(in) :: i1,i2
end function ReadLog2

Logical*1 function ReadLog3(group,var,unit,i1,i2,i3)
  !DEC$ Attributes C, Alias:"_dReadLog3" :: ReadLog3
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3
                                !DEC$ Attributes Value:: i1,i2,i3
  intent(in) :: i1,i2,i3
end function ReadLog3

Logical*1 function ReadLog4(group,var,unit,i1,i2,i3,i4)
  !DEC$ Attributes C, Alias:"_dReadLog4" :: ReadLog4
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4
                                !DEC$ Attributes Value:: i1,i2,i3,i4
  intent(in) :: i1,i2,i3,i4
end function ReadLog4

Logical*1 function ReadLog5(group,var,unit,i1,i2,i3,i4,i5)
  !DEC$ Attributes C, Alias:"_dReadLog5" :: ReadLog5
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4,i5
                                !DEC$ Attributes Value:: i1,i2,i3,i4,i5
  intent(in) :: i1,i2,i3,i4,i5
end function ReadLog5

Logical*1 function ReadLog6(group,var,unit,i1,i2,i3,i4,i5,i6)
  !DEC$ Attributes C, Alias:"_dReadLog6" :: ReadLog6
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4,i5,i6
                                !DEC$ Attributes Value:: i1,i2,i3,i4,i5,i6
  intent(in) :: i1,i2,i3,i4,i5,i6
end function ReadLog6

!////////////////////////////////////////////////////////////////////////////
!//   routines for reading string variables
!////////////////////////////////////////////////////////////////////////////

subroutine ReadString(group,var,unit,str)
  !DEC$ Attributes C, Alias:"_dReadString" :: ReadString
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  character*80, intent(out) :: str
                                !DEC$ Attributes Reference::str
end subroutine ReadString

subroutine ReadString1(group,var,unit,i1,str)
  !DEC$ Attributes C, Alias:"_dReadString1" :: ReadString1
  character*20, intent(in) :: group,var,unit
```

```
                        !DEC$ Attributes Reference::group,var,unit
  integer*2 i1
                        !DEC$ Attributes Value:: i1
  intent(in) :: i1
  character*80, intent(out) :: str
                        !DEC$ Attributes Reference::str
end subroutine ReadString1

subroutine ReadString2(group,var,unit,i1,i2,str)
  !DEC$ Attributes C, Alias:"_dReadString2" :: ReadString2
  character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2
                        !DEC$ Attributes Value:: i1,i2
  intent(in) :: i1,i2
  character*80, intent(out) :: str
                        !DEC$ Attributes Reference::str
end subroutine ReadString2

subroutine ReadString3(group,var,unit,i1,i2,i3,str)
  !DEC$ Attributes C, Alias:"_dReadString3" :: ReadString3
  character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3
                        !DEC$ Attributes Value:: i1,i2,i3
  intent(in) :: i1,i2,i3
  character*80, intent(out) :: str
                        !DEC$ Attributes Reference::str
end subroutine ReadString3

subroutine ReadString4(group,var,unit,i1,i2,i3,i4,str)
  !DEC$ Attributes C, Alias:"_dReadString4" :: ReadString4
  character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4
                        !DEC$ Attributes Value:: i1,i2,i3,i4
  intent(in) :: i1,i2,i3,i4
  character*80, intent(out) :: str
                        !DEC$ Attributes Reference::str
end subroutine ReadString4

subroutine ReadString5(group,var,unit,i1,i2,i3,i4,i5,str)
  !DEC$ Attributes C, Alias:"_dReadString5" :: ReadString5
  character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4,i5
                        !DEC$ Attributes Value:: i1,i2,i3,i4,i5
  intent(in) :: i1,i2,i3,i4,i5
  character*80, intent(out) :: str
                        !DEC$ Attributes Reference::str
end subroutine ReadString5

subroutine ReadString6(group,var,unit,i1,i2,i3,i4,i5,i6,str)
  !DEC$ Attributes C, Alias:"_dReadString6" :: ReadString6
  character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4,i5,i6
                        !DEC$ Attributes Value:: i1,i2,i3,i4,i5,i6
  intent(in) :: i1,i2,i3,i4,i5,i6
  character*80, intent(out) :: str
                        !DEC$ Attributes Reference::str
end subroutine ReadString6
```

```
!////////////////////////////////////////////////////////////////////////
!//  Export routines for writing integer variables
!////////////////////////////////////////////////////////////////////////

      subroutine WriteInt(group,var,unit,pval)
        !DEC$ Attributes C, Alias:"_dWriteInt" :: WriteInt
        character*20, intent(in) :: group,var,unit
                                  !DEC$ Attributes Reference::group,var,unit
        integer*2 pval
                                  !DEC$ Attributes Value:: pval
        intent(in) :: pval
      end subroutine WriteInt

      subroutine WriteInt1(group,var,unit,i1,pval)
        !DEC$ Attributes C, Alias:"_dWriteInt1" :: WriteInt1
        character*20, intent(in) :: group,var,unit
                                  !DEC$ Attributes Reference::group,var,unit
        integer*2 i1
        intent(in) :: i1
        integer*2 pval
                                  !DEC$ Attributes Value:: i1,pval
        intent(in) :: pval
      end subroutine WriteInt1

      subroutine WriteInt2(group,var,unit,i1,i2,pval)
        !DEC$ Attributes C, Alias:"_dWriteInt2" :: WriteInt2
        character*20, intent(in) :: group,var,unit
                                  !DEC$ Attributes Reference::group,var,unit
        integer*2 i1,i2
        intent(in) :: i1,i2
        integer*2 pval
        intent(in) :: pval
                                  !DEC$ Attributes Value:: i1,i2,pval
      end subroutine WriteInt2

      subroutine WriteInt3(group,var,unit,i1,i2,i3,pval)
        !DEC$ Attributes C, Alias:"_dWriteInt3" :: WriteInt3
        character*20, intent(in) :: group,var,unit
                                  !DEC$ Attributes Reference::group,var,unit
        integer*2 i1,i2,i3
        intent(in) :: i1,i2,i3
        integer*2 pval
        intent(in) :: pval
                                  !DEC$ Attributes Value:: i1,i2,i3,pval
      end subroutine WriteInt3

      subroutine WriteInt4(group,var,unit,i1,i2,i3,i4,pval)
        !DEC$ Attributes C, Alias:"_dWriteInt4" :: WriteInt4
        character*20, intent(in) :: group,var,unit
                                  !DEC$ Attributes Reference::group,var,unit
        integer*2 i1,i2,i3,i4
        intent(in) :: i1,i2,i3,i4
        integer*2 pval
        intent(in) :: pval
                                  !DEC$ Attributes Value:: i1,i2,i3,i4,pval
      end subroutine WriteInt4

      subroutine WriteInt5(group,var,unit,i1,i2,i3,i4,i5,pval)
        !DEC$ Attributes C, Alias:"_dWriteInt5" :: WriteInt5
        character*20, intent(in) :: group,var,unit
                                  !DEC$ Attributes Reference::group,var,unit
        integer*2 i1,i2,i3,i4,i5
        intent(in) :: i1,i2,i3,i4,i5
        integer*2 pval
```

```
  intent(in) :: pval
                              !DEC$ Attributes Value:: i1,i2,i3,i4,i5,pval
      end subroutine WriteInt5

      subroutine WriteInt6(group,var,unit,i1,i2,i3,i4,i5,i6,pval)
        !DEC$ Attributes C, Alias:"_dWriteInt6" :: WriteInt6
        character*20, intent(in) :: group,var,unit
                              !DEC$ Attributes Reference::group,var,unit
        integer*2 i1,i2,i3,i4,i5,i6
        intent(in) :: i1,i2,i3,i4,i5,i6
        integer*2 pval
        intent(in) :: pval
                              !DEC$ Attributes Value:: i1,i2,i3,i4,i5,i6,pval
      end subroutine WriteInt6

!////////////////////////////////////////////////////////////////////////
!//  Export routines for writing real(double); variables
!////////////////////////////////////////////////////////////////////////

      subroutine WriteReal(group,var,unit,pval)
        !DEC$ Attributes C, Alias:"_dWriteReal" :: WriteReal
        character*20, intent(in) :: group,var,unit
                              !DEC$ Attributes Reference::group,var,unit
        Real*8 pval
        intent(in) :: pval
                              !DEC$ Attributes Value:: pval
      end subroutine WriteReal

      subroutine WriteReal1(group,var,unit,i1,pval)
        !DEC$ Attributes C, Alias:"_dWriteReal1" :: WriteReal1
        character*20, intent(in) :: group,var,unit
                              !DEC$ Attributes Reference::group,var,unit
        integer*2 i1
        intent(in) :: i1
        Real*8 pval
        intent(in) :: pval
                              !DEC$ Attributes Value:: i1,pval
      end subroutine WriteReal1

      subroutine WriteReal2(group,var,unit,i1,i2,pval)
        !DEC$ Attributes C, Alias:"_dWriteReal2" :: WriteReal2
        character*20, intent(in) :: group,var,unit
                              !DEC$ Attributes Reference::group,var,unit
        integer*2 i1,i2
        intent(in) :: i1,i2
        Real*8 pval
        intent(in) :: pval
                              !DEC$ Attributes Value:: i1,i2,pval
      end subroutine WriteReal2

      subroutine WriteReal3(group,var,unit,i1,i2,i3,pval)
        !DEC$ Attributes C, Alias:"_dWriteReal3" :: WriteReal3
        character*20, intent(in) :: group,var,unit
                              !DEC$ Attributes Reference::group,var,unit
        integer*2 i1,i2,i3
        intent(in) :: i1,i2,i3
        Real*8 pval
        intent(in) :: pval
                              !DEC$ Attributes Value:: i1,i2,i3,pval
      end subroutine WriteReal3

      subroutine WriteReal4(group,var,unit,i1,i2,i3,i4,pval)
        !DEC$ Attributes C, Alias:"_dWriteReal4" :: WriteReal4
        character*20, intent(in) :: group,var,unit
```

```
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4
  intent(in) :: i1,i2,i3,i4
  Real*8 pval
  intent(in) :: pval
                                !DEC$ Attributes Value:: i1,i2,i3,i4,pval
end subroutine WriteReal4

subroutine WriteReal5(group,var,unit,i1,i2,i3,i4,i5,pval)
  !DEC$ Attributes C, Alias:"_dWriteReal5" :: WriteReal5
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4,i5
  intent(in) :: i1,i2,i3,i4,i5
  Real*8 pval
  intent(in) :: pval
                                !DEC$ Attributes Value:: i1,i2,i3,i4,i5,pval
end subroutine WriteReal5

subroutine WriteReal6(group,var,unit,i1,i2,i3,i4,i5,i6,pval)
  !DEC$ Attributes C, Alias:"_dWriteReal6" :: WriteReal6
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2,i3,i4,i5,i6
  intent(in) :: i1,i2,i3,i4,i5,i6
  Real*8 pval
  intent(in) :: pval
                                !DEC$ Attributes Value:: i1,i2,i3,i4,i5,i6,pval
end subroutine WriteReal6

!////////////////////////////////////////////////////////////////////////////
!//  Export routines for writing logical variables
!////////////////////////////////////////////////////////////////////////////

subroutine WriteLog(group,var,unit,pval)
                !DEC$ Attributes C, Alias:"_dWriteLog" :: WriteLog
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  Logical*1 pval
  intent(in) :: pval
                                !DEC$ Attributes Value:: pval
end subroutine WriteLog

subroutine WriteLog1(group,var,unit,i1,pval)
                !DEC$ Attributes C, Alias:"_dWriteLog1" :: WriteLog1
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1
  intent(in) :: i1
  Logical*1 pval
  intent(in) :: pval
                                !DEC$ Attributes Value:: i1,pval
end subroutine WriteLog1

subroutine WriteLog2(group,var,unit,i1,i2,pval)
                !DEC$ Attributes C, Alias:"_dWriteLog2" :: WriteLog2
  character*20, intent(in) :: group,var,unit
                                !DEC$ Attributes Reference::group,var,unit
  integer*2 i1,i2
  intent(in) :: i1,i2
  Logical*1 pval
  intent(in) :: pval
                                !DEC$ Attributes Value:: i1,i2,pval
end subroutine WriteLog2
```

```
        subroutine WriteLog3(group,var,unit,i1,i2,i3,pval)
                        !DEC$ Attributes C, Alias:"_dWriteLog3" :: WriteLog3
          character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
          integer*2 i1,i2,i3
          intent(in) :: i1,i2,i3
          Logical*1 pval
          intent(in) :: pval
                        !DEC$ Attributes Value:: i1,i2,i3,pval
        end subroutine WriteLog3

        subroutine WriteLog4(group,var,unit,i1,i2,i3,i4,pval)
                        !DEC$ Attributes C, Alias:"_dWriteLog4" :: WriteLog4
          character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
          integer*2 i1,i2,i3,i4
          intent(in) :: i1,i2,i3,i4
          Logical*1 pval
          intent(in) :: pval
                        !DEC$ Attributes Value:: i1,i2,i3,i4,pval
        end subroutine WriteLog4

        subroutine WriteLog5(group,var,unit,i1,i2,i3,i4,i5,pval)
                        !DEC$ Attributes C, Alias:"_dWriteLog5" :: WriteLog5
          character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
          integer*2 i1,i2,i3,i4,i5
          intent(in) :: i1,i2,i3,i4,i5
          Logical*1 pval
          intent(in) :: pval
                        !DEC$ Attributes Value:: i1,i2,i3,i4,i5,pval
        end subroutine WriteLog5

        subroutine WriteLog6(group,var,unit,i1,i2,i3,i4,i5,i6,pval)
                        !DEC$ Attributes C, Alias:"_dWriteLog6" :: WriteLog6
          character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
          integer*2 i1,i2,i3,i4,i5,i6
          intent(in) :: i1,i2,i3,i4,i5,i6
          Logical*1 pval
          intent(in) :: pval
                        !DEC$ Attributes Value:: i1,i2,i3,i4,i5,i6,pval
        end subroutine WriteLog6

!////////////////////////////////////////////////////////////////////////////
!//  Export routines for writing pvaling variables
!////////////////////////////////////////////////////////////////////////////

        subroutine WriteString(group,var,unit,str)
                        !DEC$ Attributes C, Alias:"_dWriteString" :: WriteString
          character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
          character*80, intent(in) :: str
                        !DEC$ Attributes Reference::str
        end subroutine WriteString

        subroutine WriteString1(group,var,unit,i1,str)
                        !DEC$ Attributes C, Alias:"_dWriteString1" :: WriteString1
          character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
          integer*2 i1
                        !DEC$ Attributes Value:: i1
          intent(in) :: i1
          character*80, intent(in) :: str
```

```
                        !DEC$ Attributes Reference::str
  end subroutine WriteString1

  subroutine WriteString2(group,var,unit,i1,i2,str)
                !DEC$ Attributes C, Alias:"_dWriteString2" :: WriteString2
   character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
   integer*2 i1,i2
                        !DEC$ Attributes Value:: i1,i2
   intent(in) :: i1,i2
   character*80, intent(in) :: str
                        !DEC$ Attributes Reference::str
  end subroutine WriteString2

  subroutine WriteString3(group,var,unit,i1,i2,i3,str)
                !DEC$ Attributes C, Alias:"_dWriteString3" :: WriteString3
   character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
   integer*2 i1,i2,i3
                        !DEC$ Attributes Value:: i1,i2,i3
   intent(in) :: i1,i2,i3
   character*80, intent(in) :: str
                        !DEC$ Attributes Reference::str
  end subroutine WriteString3

  subroutine WriteString4(group,var,unit,i1,i2,i3,i4,str)
                !DEC$ Attributes C, Alias:"_dWriteString4" :: WriteString4
   character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
   integer*2 i1,i2,i3,i4
                        !DEC$ Attributes Value:: i1,i2,i3,i4
   intent(in) :: i1,i2,i3,i4
   character*80, intent(in) :: str
                        !DEC$ Attributes Reference::str
  end subroutine WriteString4

  subroutine WriteString5(group,var,unit,i1,i2,i3,i4,i5,str)
                !DEC$ Attributes C, Alias:"_dWriteString5" :: WriteString5
   character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
   integer*2 i1,i2,i3,i4,i5
                        !DEC$ Attributes Value:: i1,i2,i3,i4,i5
   intent(in) :: i1,i2,i3,i4,i5
   character*80, intent(in) :: str
                        !DEC$ Attributes Reference::str
  end subroutine WriteString5

  subroutine WriteString6(group,var,unit,i1,i2,i3,i4,i5,i6,str)
                !DEC$ Attributes C, Alias:"_dWriteString6" :: WriteString6
   character*20, intent(in) :: group,var,unit
                        !DEC$ Attributes Reference::group,var,unit
   integer*2 i1,i2,i3,i4,i5,i6
                        !DEC$ Attributes Value:: i1,i2,i3,i4,i5,i6
   intent(in) :: i1,i2,i3,i4,i5,i6
   character*80, intent(in) :: str
                        !DEC$ Attributes Reference::str
  end subroutine WriteString6
 end interface
end module DVF90hwirio
```

A.3 C++ Test Program Listing

This section lists the C++ version of the test program used to prepare both the Borland C++ Version 4.0 and Microsoft® Visual C++ 5.0 compiled executables.

CDLLTST.CPP

```
#include <stdio.h>
// DLL function types
// THIS MUST BE INCLUDED FOR THINGS TO WORK
#include "MSC++5.h"
// This is a collection of C++ routines for reading data
#include "csv.h"

void main(int argc,char **argv)
  {
     char Key;
     char FunName[20],DataGroup[20];
     char VarName[20],Units[20];
     char Junk[80],RetS[80];
     char IFile[80],OFile[80];
     int I1,I2,I3,I4,I5,I6,RetI;
     double RetR;
 // No such type as logical in C or C++
 // must read in as char string check for
 // T of F and assign the right value
     int RetL;

     icsv *input;
     ocsv *output;
     sprintf(IFile, "%s.tst",argv[1]);
     sprintf(OFile, "%s.out",argv[1]);
     input=new icsv(IFile",\"",",");
     output=new ocsv(OFile" "," ");
     if (input->Ok() && output->Ok())
       while (strcmpi(FunName,"Stop")!=0)
         {
           *input >> FunName >> NewLine;
           cout << FunName << "\n";
           if (strcmpi(FunName,"Pause")==0)
             {
               *output << "Call Pause" << NewLine;
               cout << "Press a key and then the return key to continue\n";
               cin >> Key;  // dummy get integer
             }
           else if (strcmpi(FunName,"OpenGroups")==0)
             {
               *output << "Call OpenGroups" << NewLine;
               OpenGroups();
             }
           else if (strcmpi(FunName,"CloseGroups")==0)
             {
               *output << "Call CloseGroups"<< NewLine;
               CloseGroups();
             }
           else if (strcmpi(FunName,"Error")==0)
             {
               *input >> Junk >> NewLine;
               *output << "Call Error " << Junk << NewLine;
               delete input;  // this doesn't delete the file
               delete output; // this doesn't delete the file
```

```
            Error(Junk);
            return ; // should never get here;
          }
        else if (strcmpi(FunName,“Warning”)==0)
          {
            *input >> Junk>> NewLine;
            *output << “Call Warning” << Junk << NewLine;
            Warning(Junk);
          }
        else if (strcmpi(FunName,“NumArgs”)==0)
          {
            I1=NumArgs();
            *output << “Call NumArgs” << NewLine;
            *output << “Returned” << I1 << NewLine;
          }
        else if (strcmpi(FunName,“GetArgInt”)==0)
          {
            *input >> I1 >> NewLine;
            I2=GetArgInt(I1);
            *output << “Call GetArgInt” << NewLine;
            *output << “Returned” << I2 << NewLine;
           }
        else if (strcmpi(FunName,“GetArgString”)==0)
          {
            *input >> I1 >> NewLine;
            GetArgString(I1,Junk);
            *output << “Call GetArgString” << NewLine;
            *output << “Returned” << Junk << NewLine;
          }

// ******** Section containing calls to ReadInt and ReadInt[1-6]   ***********

      else if (strcmpi(FunName,“ReadInt”)==0)
          {
            *input >> DataGroup >> VarName >>Units >> NewLine;
            RetI=ReadInt(DataGroup,VarName,Units);
            *output << “Call ReadInt” << DataGroup << VarName << Units <<
                    NewLine;
            *output << "Returned " << RetI << NewLine;
          }

      else if (strcmpi(FunName,“ReadInt1”)==0)
          {
            *input >> DataGroup >> VarName >>Units >>
                   I1 >> NewLine;
            RetI=ReadInt1(DataGroup,VarName,Units,I1);
            *output << “Call ReadInt1” << DataGroup << VarName << Units <<
                    I1 << NewLine;
            *output << “Returned” << RetI << NewLine;
          }

      else if (strcmpi(FunName,“ReadInt2”)==0)
          {
            *input >> DataGroup >> VarName >>Units >>
                   I1 >> I2 >> NewLine;
            RetI=ReadInt2(DataGroup,VarName,Units,I1,I2);
            *output << “Call ReadInt2” << DataGroup << VarName << Units <<
                    I1 << I2 << NewLine;
            *output << "Returned " << RetI << NewLine;
          }

      else if (strcmpi(FunName,"ReadInt3")==0)
          {
            *input >> DataGroup >> VarName >>Units >>
```

```
                    I1 >> I2 >> I3 >> NewLine;
                RetI=ReadInt3(DataGroup,VarName,Units,I1,I2,I3);
                *output << “Call ReadInt3” << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << NewLine;
                *output << “Returned” << RetI << NewLine;
              }

            else if (strcmpi(FunName,“ReadInt4”)==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                        I1 >> I2 >> I3 >> I4 >> NewLine;
                RetI=ReadInt4(DataGroup,VarName,Units,I1,I2,I3,I4);
                *output << “Call ReadInt4” << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << I4 << NewLine;
                *output << “Returned” << RetI << NewLine;
              }

            else if (strcmpi(FunName,"ReadInt5")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                        I1 >> I2 >> I3 >> I4 >> I5 >> NewLine;
                RetI=ReadInt5(DataGroup,VarName,Units,I1,I2,I3,I4,I5);
                *output << “Call ReadInt5” << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << I4 << I5 << NewLine;
                *output << “Returned” << RetI << NewLine;
              }

            else if (strcmpi(FunName,"ReadInt6")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                        I1 >> I2 >> I3 >> I4 >> I5 >> I6 >> NewLine;
                RetI=ReadInt6(DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6);
                *output << “Call ReadInt6” << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << I4 << I5 << I6 << NewLine;
                *output << “Returned” << RetI << NewLine;
              }

// ******** Section containing calls to ReadReal and ReadReal[1-6]   ***********

             else if (strcmpi(FunName, “ReadReal”)==0)
               {
                 *input >> DataGroup >> VarName >>Units >> NewLine;
                 RetR=ReadReal(DataGroup,VarName,Units);
                 *output << “Call ReadReal” << DataGroup << VarName << Units <<
                         NewLine;
                 *output << “Returned” << RetR << NewLine;
               }

             else if (strcmpi(FunName, “ReadReal1”)==0)
               {
                 *input >> DataGroup >> VarName >>Units >>
                        I1 >> NewLine;
                 RetR=ReadReal1(DataGroup,VarName,Units,I1);
                 *output << “Call ReadReal1” << DataGroup << VarName << Units <<
                         I1 << NewLine;
                 *output << “Returned” << RetR << NewLine;
               }

             else if (strcmpi(FunName,“ReadReal2”)==0)
               {
                 *input >> DataGroup >> VarName >>Units >>
                        I1 >> I2 >> NewLine;
                 RetR=ReadReal2(DataGroup,VarName,Units,I1,I2);
                 *output << “Call ReadReal2” << DataGroup << VarName << Units <<
```

```
                    I1 << I2 << NewLine;
                *output << "Returned" << RetR << NewLine;
              }

            else if (strcmpi(FunName, "ReadReal3")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> NewLine;
                RetR=ReadReal3(DataGroup,VarName,Units,I1,I2,I3);
                *output << "Call ReadReal3" << DataGroup << VarName << Units <<
                       I1 << I2 << I3 << NewLine;
                *output << "Returned" << RetR << NewLine;
              }

            else if (strcmpi(FunName,"ReadReal4")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> I4 >> NewLine;
                RetR=ReadReal4(DataGroup,VarName,Units,I1,I2,I3,I4);
                *output << "Call ReadReal4" << DataGroup << VarName << Units <<
                       I1 << I2 << I3 << I4 << NewLine;
                *output << "Returned" << RetR << NewLine;
              }

            else if (strcmpi(FunName,"ReadReal5")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> I4 >> I5 >> NewLine;
                RetR=ReadReal5(DataGroup,VarName,Units,I1,I2,I3,I4,I5);
                *output << "Call ReadReal5" << DataGroup << VarName << Units <<
                       I1 << I2 << I3 << I4 << I5 << NewLine;
                *output << "Returned" << RetR << NewLine;
              }

            else if (strcmpi(FunName, "ReadReal6")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> I4 >> I5 >> I6 >> NewLine;
                RetR=ReadReal6(DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6);
                *output << "Call ReadReal6" << DataGroup << VarName << Units <<
                       I1 << I2 << I3 << I4 << I5 << I6 << NewLine;
                *output << "Returned" << RetR << NewLine;
              }

// ******** Section containing calls to ReadLog and ReadLog[1-6]   ***********

            else if (strcmpi(FunName, "ReadLog")==0)
              {
                *input >> DataGroup >> VarName >> Units >> NewLine;
                RetL=ReadLog(DataGroup,VarName,Units);
                *output << "Call ReadLog" << DataGroup << VarName << Units <<
                       NewLine;
                *output << "Returned" << RetL << NewLine;
              }

            else if (strcmpi(FunName,"ReadLog1")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> NewLine;
                RetL=ReadLog1(DataGroup,VarName,Units,I1);
                *output << "Call ReadLog1" << DataGroup << VarName << Units <<
                       I1 << NewLine;
                *output << "Returned" << RetL << NewLine;
              }
```

```
            else if (strcmpi(FunName, “ReadLog2”)==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> NewLine;
                RetL=ReadLog2(DataGroup,VarName,Units,I1,I2);
                *output << “Call ReadLog2” << DataGroup << VarName << Units <<
                        I1 << I2 << NewLine;
                *output << “Returned” << RetL << NewLine;
              }

            else if (strcmpi(FunName,“ReadLog3”)==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> NewLine;
                RetL=ReadLog3(DataGroup,VarName,Units,I1,I2,I3);
                *output << “Call ReadLog3” << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << NewLine;
                *output << “Returned” << RetL << NewLine;
              }

            else if (strcmpi(FunName, “ReadLog4”)==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> I4 >> NewLine;
                RetL=ReadLog4(DataGroup,VarName,Units,I1,I2,I3,I4);
                *output << “Call ReadLog4” << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << I4 << NewLine;
                *output << “Returned” << RetL << NewLine;
              }

            else if (strcmpi(FunName, “ReadLog5”)==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> I4 >> I5 >> NewLine;
                RetL=ReadLog5(DataGroup,VarName,Units,I1,I2,I3,I4,I5);
                *output << “Call ReadLog5” << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << I4 << I5 << NewLine;
                *output << “Returned” << RetL << NewLine;
              }

            else if (strcmpi(FunName, “ReadLog6”)==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> I4 >> I5 >> I6 >> NewLine;
                RetL=ReadLog6(DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6);
                *output << “Call ReadLog6” << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << I4 << I5 << I6 << NewLine;
                *output << “Returned” << RetL << NewLine;
              }

// ******** Section containing calls to ReadString and ReadString[1-6]   ***********

            else if (strcmpi(FunName, “ReadString”)==0)
              {
                *input >> DataGroup >> VarName >>Units >> NewLine;
                ReadString(DataGroup,VarName,Units,RetS);
                *output << “Call ReadString” << DataGroup << VarName << Units <<
                        NewLine;
                *output << “Returned” << RetS << NewLine;
              }

            else if (strcmpi(FunName, “ReadString1”)==0)
              {
                *input >> DataGroup >> VarName >>Units >>
```

```
                I1 >> NewLine;
            ReadString1(DataGroup,VarName,Units,I1,RetS);
            *output << "Call ReadString1" << DataGroup << VarName << Units <<
                I1 << NewLine;
            *output << "Returned" << RetS << NewLine;
          }

        else if (strcmpi(FunName, "ReadString2")==0)
          {
            *input >> DataGroup >> VarName >>Units >>
                I1 >> I2 >> NewLine;
            ReadString2(DataGroup,VarName,Units,I1,I2,RetS);
            *output << "Call ReadString2" << DataGroup << VarName << Units <<
                I1 << I2 << NewLine;
            *output << "Returned" << RetS << NewLine;
          }

        else if (strcmpi(FunName, "ReadString3")==0)
          {
            *input >> DataGroup >> VarName >>Units >>
                I1 >> I2 >> I3 >> NewLine;
            ReadString3(DataGroup,VarName,Units,I1,I2,I3,RetS);
            *output << "Call ReadString3" << DataGroup << VarName << Units <<
                I1 << I2 << I3 << NewLine;
            *output << "Returned" << RetS << NewLine;
          }

        else if (strcmpi(FunName, "ReadString4")==0)
          {
            *input >> DataGroup >> VarName >>Units >>
                I1 >> I2 >> I3 >> I4 >> NewLine;
            ReadString4(DataGroup,VarName,Units,I1,I2,I3,I4,RetS);
            *output << "Call ReadString4" << DataGroup << VarName << Units <<
                I1 << I2 << I3 << I4 << NewLine;
            *output << "Returned" << RetS << NewLine;
          }

        else if (strcmpi(FunName, "ReadString5")==0)
          {
            *input >> DataGroup >> VarName >>Units >>
                I1 >> I2 >> I3 >> I4 >> I5 >> NewLine;
            ReadString5(DataGroup,VarName,Units,I1,I2,I3,I4,I5,RetS);
            *output << "Call ReadString5" << DataGroup << VarName << Units <<
                I1 << I2 << I3 << I4 << I5 << NewLine;
            *output << "Returned" << RetS << NewLine;
          }

        else if (strcmpi(FunName,"ReadString6")==0)
          {
            *input >> DataGroup >> VarName >>Units >>
                I1 >> I2 >> I3 >> I4 >> I5 >> I6 >> NewLine;
            ReadString6(DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6,RetS);
            *output << "Call ReadString6" << DataGroup << VarName << Units <<
                I1 << I2 << I3 << I4 << I5 << I6 << NewLine;
            *output << "Returned" << RetS << NewLine;
          }

// ******** Section containing calls to WriteInt and WriteInt[1-6]   ***********

        else if (strcmpi(FunName, "WriteInt")==0)
          {
            *input >> DataGroup >> VarName >>Units >> RetI >> NewLine;
            WriteInt(DataGroup,VarName,Units,RetI);
            *output << "Call WriteInt" << DataGroup << VarName << Units <<
```

```
                RetI << NewLine;
        }

      else if (strcmpi(FunName, "WriteInt1")==0)
        {
          *input >> DataGroup >> VarName >>Units >>
                 I1 >> RetI >> NewLine;
          WriteInt1(DataGroup,VarName,Units,I1,RetI);
          *output << "Call WriteInt1" << DataGroup << VarName << Units <<
                  I1 << RetI << NewLine;
        }

      else if (strcmpi(FunName, "WriteInt2")==0)
        {
          *input >> DataGroup >> VarName >>Units >>
                 I1 >> I2 >> RetI >> NewLine;
          WriteInt2(DataGroup,VarName,Units,I1,I2,RetI);
          *output << "Call WriteInt2" << DataGroup << VarName << Units <<
                  I1 << I2 << RetI << NewLine;
        }

      else if (strcmpi(FunName, "WriteInt3")==0)
        {
          *input >> DataGroup >> VarName >>Units >>
                 I1 >> I2 >> I3 >> RetI >> NewLine;
          WriteInt3(DataGroup,VarName,Units,I1,I2,I3,RetI);
          *output << "Call WriteInt3" << DataGroup << VarName << Units <<
                  I1 << I2 << I3 << RetI << NewLine;
        }

      else if (strcmpi(FunName, "WriteInt4")==0)
        {
          *input >> DataGroup >> VarName >>Units >>
                 I1 >> I2 >> I3 >> I4 >> RetI >> NewLine;
          WriteInt4(DataGroup,VarName,Units,I1,I2,I3,I4,RetI);
          *output << "Call WriteInt4" << DataGroup << VarName << Units <<
                  I1 << I2 << I3 << I4 << RetI << NewLine;
        }

      else if (strcmpi(FunName, "WriteInt5")==0)
        {
          *input >> DataGroup >> VarName >>Units >>
                 I1 >> I2 >> I3 >> I4 >> I5 >> RetI >> NewLine;
          WriteInt5(DataGroup,VarName,Units,I1,I2,I3,I4,I5,RetI);
          *output << "Call WriteInt5" << DataGroup << VarName << Units <<
                  I1 << I2 << I3 << I4 << I5 << RetI << NewLine;
        }

      else if (strcmpi(FunName,"WriteInt6")==0)
        {
          *input >> DataGroup >> VarName >>Units >>
                 I1 >> I2 >> I3 >> I4 >> I5 >> I6 >> RetI >> NewLine;
          WriteInt6(DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6,RetI);
          *output << "Call WriteInt6" << DataGroup << VarName << Units <<
                  I1 << I2 << I3 << I4 << I5 << I6 << RetI << NewLine;
        }

// ******** Section containing calls to WriteReal and WriteReal[1-6]   ***********

      else if (strcmpi(FunName, "WriteReal")==0)
        {
          *input >> DataGroup >> VarName >>Units >>
                 RetR >> NewLine;
          WriteReal(DataGroup,VarName,Units,RetR);
```

```
        *output << "Call WriteReal " << DataGroup << VarName << Units <<
                RetR << NewLine;
      }

    else if (strcmpi(FunName, "WriteReal1")==0)
      {
        *input >> DataGroup >> VarName >>Units >>
               I1 >> RetR >> NewLine;
        WriteReal1(DataGroup,VarName,Units,I1,RetR);
        *output << "Call WriteReal1 " << DataGroup << VarName << Units <<
                I1 << RetR << NewLine;
      }

    else if (strcmpi(FunName, "WriteReal2")==0)
      {
        *input >> DataGroup >> VarName >>Units >>
               I1 >> I2 >> RetR >> NewLine;
        WriteReal2(DataGroup,VarName,Units,I1,I2,RetR);
        *output << "Call WriteReal2 " << DataGroup << VarName << Units <<
                I1 << I2 << RetR << NewLine;
      }

    else if (strcmpi(FunName, "WriteReal3")==0)
      {
        *input >> DataGroup >> VarName >>Units >>
               I1 >> I2 >> I3 >> RetR >> NewLine;
        WriteReal3(DataGroup,VarName,Units,I1,I2,I3,RetR);
        *output << "Call WriteReal3" << DataGroup << VarName << Units <<
                I1 << I2 << I3 << RetR << NewLine;
      }

    else if (strcmpi(FunName, "WriteReal4")==0)
      {
        *input >> DataGroup >> VarName >>Units >>
               I1 >> I2 >> I3 >> I4 >> RetR >> NewLine;
        WriteReal4(DataGroup,VarName,Units,I1,I2,I3,I4,RetR);
        *output << "Call WriteReal4" << DataGroup << VarName << Units <<
                I1 << I2 << I3 << I4 << RetR << NewLine;
      }

    else if (strcmpi(FunName, "WriteReal5")==0)
      {
        *input >> DataGroup >> VarName >>Units >>
               I1 >> I2 >> I3 >> I4 >> I5 >> RetR >> NewLine;
        WriteReal5(DataGroup,VarName,Units,I1,I2,I3,I4,I5,RetR);
        *output << "Call WriteReal5" << DataGroup << VarName << Units <<
                I1 << I2 << I3 << I4 << I5 << RetR << NewLine;
      }

    else if (strcmpi(FunName, "WriteReal6")==0)
      {
        *input >> DataGroup >> VarName >>Units >>
               I1 >> I2 >> I3 >> I4 >> I5 >> I6 >> RetR >> NewLine;
        WriteReal6(DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6,RetR);
        *output << "Call WriteReal6" << DataGroup << VarName << Units <<
                I1 << I2 << I3 << I4 << I5 << I6 << RetR << NewLine;
      }

// ******** Section containing calls to WriteLog and WriteLog[1-6]   ***********

    else if (strcmpi(FunName, "WriteLog")==0)
      {
        *input >> DataGroup >> VarName >> Units >> Junk >> NewLine;
        if (Junk[0] == "T") RetL = 1;
```

```
    else RetL = 0;
    WriteLog(DataGroup,VarName,Units,RetL);
    *output << “Call WriteLog” << DataGroup << VarName << Units <<
            Junk << NewLine;
  }

else if (strcmpi(FunName, “WriteLog1”)==0)
  {
    *input >> DataGroup >> VarName >>Units >>
           I1 >> Junk >> NewLine;
    if (Junk[0] == “T”) RetL = 1;
    else RetL = 0;
    WriteLog1(DataGroup,VarName,Units,I1,RetL);
    *output << "Call WriteLog1” << DataGroup << VarName << Units <<
            I1 << Junk << NewLine;
  }

else if (strcmpi(FunName, “WriteLog2”)==0)
  {
    *input >> DataGroup >> VarName >>Units >>
           I1 >> I2 >> Junk >> NewLine;
    if (Junk[0] == “T”) RetL = 1;
    else RetL = 0;
    WriteLog2(DataGroup,VarName,Units,I1,I2,RetL);
    *output << “Call WriteLog2” << DataGroup << VarName << Units <<
            I1 << I2 << Junk << NewLine;
  }

else if (strcmpi(FunName, “WriteLog3”)==0)
  {
    *input >> DataGroup >> VarName >>Units >>
           I1 >> I2 >> I3 >> Junk >> NewLine;
    if (Junk[0] == “T”) RetL = 1;
    else RetL = 0;
    WriteLog3(DataGroup,VarName,Units,I1,I2,I3,RetL);
    *output << “Call WriteLog3” << DataGroup << VarName << Units <<
            I1 << I2 << I3 << Junk << NewLine;
  }

else if (strcmpi(FunName, “WriteLog4”)==0)
  {
    *input >> DataGroup >> VarName >>Units >>
           I1 >> I2 >> I3 >> I4 >> Junk >> NewLine;
    if (Junk[0] == “T”) RetL = 1;
    else RetL = 0;
    WriteLog4(DataGroup,VarName,Units,I1,I2,I3,I4,RetL);
    *output << “Call WriteLog4” << DataGroup << VarName << Units <<
            I1 << I2 << I3 << I4 << Junk << NewLine;
  }

else if (strcmpi(FunName, “WriteLog5”)==0)
  {
    *input >> DataGroup >> VarName >>Units >>
           I1 >> I2 >> I3 >> I4 >> I5 >> Junk >> NewLine;
    if (Junk[0] == “T”) RetL = 1;
    else RetL = 0;
    WriteLog5(DataGroup,VarName,Units,I1,I2,I3,I4,I5,RetL);
    *output << “Call WriteLog5” << DataGroup << VarName << Units <<
            I1 << I2 << I3 << I4 << I5 << Junk << NewLine;
  }

else if (strcmpi(FunName, “WriteLog6”)==0)
  {
    *input >> DataGroup >> VarName >>Units >>
```

```
                    I1 >> I2 >> I3 >> I4 >> I5 >> I6 >> Junk >> NewLine;
                if (Junk[0] == "T") RetL = 1;
                else RetL = 0;
                WriteLog6(DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6,RetL);
                *output << "Call WriteLog6" << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << I4 << I5 << I6 << Junk << NewLine;
              }

// ******** Section containing calls to WriteString and WriteString[1-6]   ***********

            else if (strcmpi(FunName, "WriteString")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       RetS >> NewLine;
                WriteString(DataGroup,VarName,Units,RetS);
                *output << "Call WriteString" << DataGroup << VarName << Units <<
                        RetS << NewLine;
              }

            else if (strcmpi(FunName, "WriteString1")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> RetS >> NewLine;
                WriteString1(DataGroup,VarName,Units,I1,RetS);
                *output << "Call WriteString1" << DataGroup << VarName << Units <<
                        I1 << RetS << NewLine;
              }

            else if (strcmpi(FunName, "WriteString2")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> RetS >> NewLine;
                WriteString2(DataGroup,VarName,Units,I1,I2,RetS);
                *output << "Call WriteString2" << DataGroup << VarName << Units <<
                        I1 << I2 << RetS << NewLine;
              }


            else if (strcmpi(FunName, "WriteString3")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> RetS >> NewLine;
                WriteString3(DataGroup,VarName,Units,I1,I2,I3,RetS);
                *output << "Call WriteString3" << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << RetS << NewLine;
              }

            else if (strcmpi(FunName, "WriteString4")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> I4 >> RetS >> NewLine;
                WriteString4(DataGroup,VarName,Units,I1,I2,I3,I4,RetS);
                *output << "Call WriteStrin4" << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << I4 << RetS << NewLine;
              }

            else if (strcmpi(FunName, "WriteString5")==0)
              {
                *input >> DataGroup >> VarName >>Units >>
                       I1 >> I2 >> I3 >> I4 >> I5 >> RetS >> NewLine;
                WriteString5(DataGroup,VarName,Units,I1,I2,I3,I4,I5,RetS);
                *output << "Call WriteString5" << DataGroup << VarName << Units <<
                        I1 << I2 << I3 << I4 << I5 << RetS << NewLine;
              }
```

```
          else if (strcmpi(FunName, "WriteString6")==0)
            {
              *input >> DataGroup >> VarName >>Units >>
                     I1 >> I2 >> I3 >> I4 >> I5 >> I6 >> RetS >> NewLine;
              WriteString6(DataGroup,VarName,Units,I1,I2,I3,I4,I5,I6,RetS);
              *output << "Call WriteString6" << DataGroup << VarName << Units <<
                      I1 << I2 << I3 << I4 << I5 << I6 << RetS << NewLine;
            }
        }
  }
```

This include file is needed so the test program can read/write data in CSV format.

CSV.CPP

```
#include "csv.h"
#include <stdlib.h>

////////////////////////////////////////////////////////////////////////////
//   methods for class icsv
////////////////////////////////////////////////////////////////////////////
    int icsv::Ok()
      {
        if (fptr.rdbuf()==NULL) return 0;
        if (fptr.rdbuf()->is_open()) return 1;
        return 0;
      }
    int icsv::eof()
      {
        return fptr.eof();
      }
    icsv::icsv(char *name,char sdel,char del)
      {
        fptr.open(name,ios::in);
        sdelim=sdel;
        delim=del;
      }
    icsv::~icsv()
      {
        fptr.close();
      }
    icsv& icsv::operator >> (int &i)
      {
        *this >> temp;
        i=atoi(temp);
        return *this;
      }
    icsv& icsv::operator >> (long &i)
      {
        *this >> temp;
        i=atol(temp);
        return *this;
      }
    icsv& icsv::operator >> (float &f)
      {
        *this >> temp;
        f=atof(temp);
        return *this;
      }
    icsv& icsv::operator >> (double &d)
      {
        *this >> temp;
        d=atof(temp);
```

```
        return *this;
      }
    icsv& icsv::operator >> (char  *s)
      {
        int c;
        int q;  // q = 0 no quote q = 1 quote mode
        q=0;
        c=fptr.peek();
        while(c==" " && !fptr.eof())
          {
            fptr.get();
            c=fptr.peek();
          }
        do
          {
            c=fptr.get();
            if (c==sdelim && q==0)
              {
                q=1;
                c=fptr.get();
              }
            if (c==sdelim && q==1)
              {
                q=0;
                c=fptr.get();
// Added by Mitch Pelton
// Scan the rest of the line until a delim.
// Another option might be to consider the closing
// quote as a delimeter and start a new field.
// I don"t think that is the way to go.
                while(c!=delim && c!="\n" && !fptr.eof())
                {
                  fptr.get();
                  c=fptr.get();
                }
                break;
// End of addition
              }
// Comment by Mitch Pelton
// The \n should probably be taken out for multiline strings
            if ((c!=delim && c!="\n")|| q!=0)
              {
                *s=(char)c;
                s++;
              }
          }
        while (((c!=delim && c!="\n")|| q!=0) && !fptr.eof());
        if (c=="\n") fptr.putback((char)c);
        *s="\0";
        return *this;
      }
    icsv& icsv::operator >>(LineCommand lc)
      {
        if (lc==NewLine) return NextLine();
        else return *this;
      }
    icsv& icsv::NextLine()
      {
        while (fptr.get()!="\n" && !fptr.eof());
        return *this;
      }


////////////////////////////////////////////////////////////////////////
```

```
//   methods for class ocsv
////////////////////////////////////////////////////////////////////////////
    int ocsv::Ok()
      {
        if (fptr.rdbuf()==NULL) return 0;
        if (fptr.rdbuf()->is_open()) return 1;
        return 0;
      }
    ocsv::ocsv(char *name,char sdel,char del)
      {
        fptr.open(name);
        sdelim=sdel;
        delim=del;
      }
    ocsv::~ocsv()
      {
        fptr.close();
      }
    ocsv& ocsv::operator << (signed int i)
      {
        sprintf(temp,"%d%c",i,delim);
        fptr.write(temp,strlen(temp));
        return *this;
      }
    ocsv& ocsv::operator << (float f)
      {
        sprintf(temp,"%.10g%c",f,delim);
        fptr.write(temp,strlen(temp));
        return *this;
      }
    ocsv& ocsv::operator << (double d)
      {
        sprintf(temp,"%.10lg%c",d,delim);
        fptr.write(temp,strlen(temp));
        return *this;
      }
    ocsv& ocsv::operator << (char  *s)
      {
        fptr.put(sdelim);
        if (s!= NULL)
          fptr.write(s,strlen(s));
        fptr.put(sdelim);
        fptr.put(delim);
        return *this;
      }
    ocsv& ocsv::operator << (LineCommand lc)
      {
        if (lc==NewLine) return NextLine();
        else return *this;
      }
    ocsv& ocsv::NextLine()
      {
        char *LN="\n";
        fptr.write(LN,strlen(LN));
        fptr.flush();
        return *this;
      }
    void ocsv::Flush()
      {
        fptr.flush();
      }
```

This is the include file CSV.H.

```
#ifndef CSV_H
#define CSV_H
#include <string.h>
#include <stdio.h>
#include <fstream.h>

#define MAXFIELD 1024
typedef enum {NewLine}LineCommand;

class icsv
 {
    char temp[MAXFIELD];
    ifstream fptr;
    char sdelim,delim;
    public:
    int eof();
    icsv(char *name,char sdel="",”char del=,””);
    int Ok();
    ~icsv();
    icsv& operator >> (int &i);
    icsv& operator >> (long &i);
    icsv& operator >> (float &f);
    icsv& operator >> (double &d);
    icsv& operator >> (char  *s);
    icsv& operator >> (LineCommand lc);
    icsv& NextLine();
  };

class ocsv
  {
    char temp[MAXFIELD];
    ofstream fptr;
    char sdelim,delim;
    public:
    int Ok();
    ocsv(char *name,char sdel="",”char del=,””);
    ~ocsv();
    ocsv& operator << (signed int i);
    ocsv& operator << (float f);
    ocsv& operator << (double d);
    ocsv& operator << (char  *s);
    ocsv& operator << (LineCommand lc);
    ocsv& NextLine();
    void Flush();
  };

#endif
```

This include file is needed to prepare a Microsoft® Visual C++ 5.0 compiled executable.

MSC++5.H

```
#ifndef MSCPP5_H
#define MSCPP5_H
//#include <windows.h>

#define PASS _declspec(dllimport) __cdecl

////////////////////////////////////////////////////////////////////////
//  Code:    HWIR_IO.h
```

```
//  Date:    3/2/98
//  Programmer:  Mitch Pelton,Karl Castleton and Bonnnie Hooopes
//  Modified
//
////////////////////////////////////////////////////////////////////////
#ifdef __cplusplus
  extern "C" {
#endif
 int  PASS  NumArgs();
 int  PASS  GetArgInt(int ArgIndex);
 void  PASS  GetArgString(int ArgIndex, char *Argument);
 void  PASS  OpenGroups();
 void  PASS  CloseGroups(void);
 void  PASS  Error(char *Description);
 void  PASS  Warning(char *Description);

////////////////////////////////////////////////////////////////////////
//   routines for reading integer variables
////////////////////////////////////////////////////////////////////////
 int  PASS  ReadInt(char *_group,char *_var,char *_unit);
 int  PASS  ReadInt1(char *_group,char *_var,char *_unit, int _1);
 int  PASS  ReadInt2(char *_group,char *_var,char *_unit, int _1,int _2);
 int  PASS  ReadInt3(char *_group,char *_var,char *_unit, int _1,int _2,int _3);
 int  PASS  ReadInt4(char *_group,char *_var,char *_unit, int _1,int _2,int _3,int _4);
 int  PASS  ReadInt5(char *_group,char *_var,char *_unit, int _1,int _2,int _3,int
_4,int _5);
 int  PASS  ReadInt6(char *_group,char *_var,char *_unit, int _1,int _2,int _3,int
_4,int _5,int _6);

////////////////////////////////////////////////////////////////////////
//   routines for reading real(double); variables
////////////////////////////////////////////////////////////////////////
 double  PASS  ReadReal(char *_group,char *_var,char *_unit);
 double  PASS  ReadReal1(char *_group,char *_var,char *_unit,int _1);
 double  PASS  ReadReal2(char *_group,char *_var,char *_unit,int _1,int _2);
 double  PASS  ReadReal3(char *_group,char *_var,char *_unit,int _1,int _2,int _3);
 double  PASS  ReadReal4(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4);
 double  PASS  ReadReal5(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5);
 double  PASS  ReadReal6(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,int _6);

////////////////////////////////////////////////////////////////////////
//   routines for reading logical variables
////////////////////////////////////////////////////////////////////////
 int  PASS  ReadLog(char *_group,char *_var,char *_unit);
 int  PASS  ReadLog1(char *_group,char *_var,char *_unit,int _1);
 int  PASS  ReadLog2(char *_group,char *_var,char *_unit,int _1,int _2);
 int  PASS  ReadLog3(char *_group,char *_var,char *_unit,int _1,int _2,int _3);
 int  PASS  ReadLog4(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int _4);
 int  PASS  ReadLog5(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int _4,int
_5);
 int  PASS  ReadLog6(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int _4,int
_5,int _6);

////////////////////////////////////////////////////////////////////////
//   routines for reading string variables
////////////////////////////////////////////////////////////////////////
 void  PASS  ReadString(char *_group,char *_var,char *_unit,char *_str);
 void  PASS  ReadString1(char *_group,char *_var,char *_unit,int _1,char *_str);
 void  PASS  ReadString2(char *_group,char *_var,char *_unit,int _1,int _2,char *_str);
 void  PASS  ReadString3(char *_group,char *_var,char *_unit,int _1,int _2,int _3,char
*_str);
 void  PASS  ReadString4(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,char *_str);
 void  PASS  ReadString5(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,char *_str);
```

```
 void  PASS  ReadString6(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,int _6,char *_str);

////////////////////////////////////////////////////////////////////////
//  Export routines for writing integer variables
////////////////////////////////////////////////////////////////////////
 void  PASS  WriteInt(char *_group,char *_var,char *_unit,int _val);
 void  PASS  WriteInt1(char *_group,char *_var,char *_unit,int _1,int _val);
 void  PASS  WriteInt2(char *_group,char *_var,char *_unit,int _1,int _2,int _val);
 void  PASS  WriteInt3(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_val);
 void  PASS  WriteInt4(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _val);
 void  PASS  WriteInt5(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,int _val);
 void  PASS  WriteInt6(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,int _6,int _val);

////////////////////////////////////////////////////////////////////////
//  Export routines for writing real(double); variables
////////////////////////////////////////////////////////////////////////
 void  PASS  WriteReal(char *_group,char *_var,char *_unit,double _val);
 void  PASS  WriteReal1(char *_group,char *_var,char *_unit,int _1,double _val);
 void  PASS  WriteReal2(char *_group,char *_var,char *_unit,int _1,int _2,double _val);
 void  PASS  WriteReal3(char *_group,char *_var,char *_unit,int _1,int _2,int _3,double
_val);
 void  PASS  WriteReal4(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,double _val);
 void  PASS  WriteReal5(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,double _val);
 void  PASS  WriteReal6(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,int _6,double _val);
////////////////////////////////////////////////////////////////////////
//  Export routines for writing logical variables
////////////////////////////////////////////////////////////////////////
 void  PASS  WriteLog(char *_group,char *_var,char *_unit,int _val);
 void  PASS  WriteLog1(char *_group,char *_var,char *_unit,int _1,int _val);
 void  PASS  WriteLog2(char *_group,char *_var,char *_unit,int _1,int _2,int _val);
 void  PASS  WriteLog3(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_val);
 void  PASS  WriteLog4(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _val);
 void  PASS  WriteLog5(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,int _val);
 void  PASS  WriteLog6(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,int _6,int _val);

////////////////////////////////////////////////////////////////////////
//  Export routines for writing string variables
////////////////////////////////////////////////////////////////////////
 void  PASS  WriteString(char *_group,char *_var,char *_unit,char *_val);
 void  PASS  WriteString1(char *_group,char *_var,char *_unit,int _1,char *_val);
 void  PASS  WriteString2(char *_group,char *_var,char *_unit,int _1,int _2,char *_val);
 void  PASS  WriteString3(char *_group,char *_var,char *_unit,int _1,int _2,int _3,char
*_val);
 void  PASS  WriteString4(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,char *_val);
 void  PASS  WriteString5(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,char *_val);
 void  PASS  WriteString6(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,int _6,char *_val);
#ifdef __cplusplus
}
#endif
#endif
```

This include file is needed to prepare a Borland C++ 4.0 compiled executable.

BC4.H

```
#ifndef BC4_H
#define BC4_H
#include <stdlib.h>

#define FAR _cdecl

////////////////////////////////////////////////////////////////////////////
/
//  Code:    HWIR_IO.h
//  Date:    3/2/98
//  Programmer:  Mitch Pelton,Karl Castleton and Bonnnie Hooopes
//  Modified
//
////////////////////////////////////////////////////////////////////////////
/

extern "C" {
int  FAR  NumArgs();
int  FAR  GetArgInt(int ArgIndex);
void  FAR  GetArgString(int ArgIndex, char *Argument);
void  FAR  OpenGroups();
void  FAR  CloseGroups(void);
void  FAR  Error(char *Description);
void  FAR  Warning(char *Description);

////////////////////////////////////////////////////////////////////////////
/
//   routines for reading integer variables
////////////////////////////////////////////////////////////////////////////
/
int  FAR  ReadInt(char *_group,char *_var,char *_unit);
int  FAR  ReadInt1(char *_group,char *_var,char *_unit, int _1);
int  FAR  ReadInt2(char *_group,char *_var,char *_unit, int _1,int _2);
int  FAR  ReadInt3(char *_group,char *_var,char *_unit, int _1,int _2,int _3);
int  FAR  ReadInt4(char *_group,char *_var,char *_unit, int _1,int _2,int
_3,int _4);
int  FAR  ReadInt5(char *_group,char *_var,char *_unit, int _1,int _2,int
_3,int _4,int _5);
int  FAR  ReadInt6(char *_group,char *_var,char *_unit, int _1,int _2,int
_3,int _4,int _5,int _6);

////////////////////////////////////////////////////////////////////////////
/
//   routines for reading real(double); variables
////////////////////////////////////////////////////////////////////////////
/
double  FAR  ReadReal(char *_group,char *_var,char *_unit);
```

```
double  FAR  ReadReal1(char *_group,char *_var,char *_unit,int _1);
double  FAR  ReadReal2(char *_group,char *_var,char *_unit,int _1,int _2);
double  FAR  ReadReal3(char *_group,char *_var,char *_unit,int _1,int _2,int
_3);
double  FAR  ReadReal4(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4);
double  FAR  ReadReal5(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5);
double  FAR  ReadReal6(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,int _6);

////////////////////////////////////////////////////////////////////////////
/
//   routines for reading logical variables
////////////////////////////////////////////////////////////////////////////
/
int  FAR ReadLog(char *_group,char *_var,char *_unit);
int  FAR ReadLog1(char *_group,char *_var,char *_unit,int _1);
int  FAR ReadLog2(char *_group,char *_var,char *_unit,int _1,int _2);
int  FAR ReadLog3(char *_group,char *_var,char *_unit,int _1,int _2,int _3);
int  FAR ReadLog4(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4);
int  FAR ReadLog5(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5);
int  FAR ReadLog6(char *_group,char *_var,char *_unit,int _1,int _2,int _3,int
_4,int _5,int _6);

////////////////////////////////////////////////////////////////////////////
/
//   routines for reading string variables
////////////////////////////////////////////////////////////////////////////
/
void  FAR ReadString(char *_group,char *_var,char *_unit,char *_str);
void  FAR ReadString1(char *_group,char *_var,char *_unit,int _1,char *_str);
void  FAR ReadString2(char *_group,char *_var,char *_unit,int _1,int _2,char
*_str);
void  FAR ReadString3(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,char *_str);
void  FAR ReadString4(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,char *_str);
void  FAR ReadString5(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,char *_str);
void  FAR ReadString6(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,int _6,char *_str);

////////////////////////////////////////////////////////////////////////////
/
//  Export routines for writing integer variables
////////////////////////////////////////////////////////////////////////////
/
void  FAR WriteInt(char *_group,char *_var,char *_unit,int _val);
```

```
void  FAR WriteInt1(char *_group,char *_var,char *_unit,int _1,int _val);
void  FAR WriteInt2(char *_group,char *_var,char *_unit,int _1,int _2,int
_val);
void  FAR WriteInt3(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _val);
void  FAR WriteInt4(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _val);
void  FAR WriteInt5(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,int _val);
void  FAR WriteInt6(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,int _6,int _val);

////////////////////////////////////////////////////////////////////////
/
//  Export routines for writing real(double); variables
////////////////////////////////////////////////////////////////////////
/
void  FAR WriteReal(char *_group,char *_var,char *_unit,double _val);
void  FAR WriteReal1(char *_group,char *_var,char *_unit,int _1,double _val);
void  FAR WriteReal2(char *_group,char *_var,char *_unit,int _1,int _2,double
_val);
void  FAR WriteReal3(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,double _val);
void  FAR WriteReal4(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,double _val);
void  FAR WriteReal5(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,double _val);
void  FAR WriteReal6(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,int _6,double _val);
////////////////////////////////////////////////////////////////////////
/
//  Export routines for writing logical variables
////////////////////////////////////////////////////////////////////////
/
void  FAR WriteLog(char *_group,char *_var,char *_unit,int _val);
void  FAR WriteLog1(char *_group,char *_var,char *_unit,int _1,int _val);
void  FAR WriteLog2(char *_group,char *_var,char *_unit,int _1,int _2,int
_val);
void  FAR WriteLog3(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _val);
void  FAR WriteLog4(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _val);
void  FAR WriteLog5(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,int _val);
void  FAR WriteLog6(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,int _6,int _val);

////////////////////////////////////////////////////////////////////////
//  Export routines for writing string variables
```

```
//////////////////////////////////////////////////////////////////////////
void  FAR WriteString(char *_group,char *_var,char *_unit,char *_val);
void  FAR WriteString1(char *_group,char *_var,char *_unit,int _1,char *_val);
void  FAR WriteString2(char *_group,char *_var,char *_unit,int _1,int _2,char
*_val);
void  FAR WriteString3(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,char *_val);
void  FAR WriteString4(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,char *_val);
void  FAR WriteString5(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,char *_val);
void  FAR WriteString6(char *_group,char *_var,char *_unit,int _1,int _2,int
_3,int _4,int _5,int _6,char *_val);
}
#endif
```

A.4 Example SDP/SSF

This section lists the example SDP/SSF data group (HD.SSF) and its associated data dictionary (HD.DIC).

HD.SSF

```
2,“Time: 12:00:00pm Date: 03/16/1998”,“Created by hand by Karl Castleton”,32,
“IntA”,0,“INTEGER”,0,“days”,
0,
“IntB”,0,“INTEGER”,0,“days”,
100,
“Int”,0,“INTEGER”,0,“days”,20,“Int1”,1,“INTEGER”,0,“days”,
5,1,2,3,4,5,
“Int2”,2,“INTEGER”,0,“days”,
2,
5,1,2,3,4,5,
6,6,7,8,9,10,11,
“Int3”,3,“INTEGER”,0,“days”,
2,2,
5,1,2,3,4,5,
5,6,7,8,9,10,
2,
5,11,12,13,14,15,
5,16,17,18,19,20,
“Int4”,4,“INTEGER”,0,“days”,
2,2,2,
5,1,2,3,4,5,
5,6,7,8,9,10,
2,
5,11,12,13,14,15,
5,16,17,18,19,20,
2,2,
5,21,22,23,24,25,
5,26,27,28,29,30,
2,
5,31,32,33,34,35,
5,36,37,38,39,40,
“Int5”,5,“INTEGER”,0,“days”,
2,2,2,2,
```

```
5,1,2,3,4,5,
5,6,7,8,9,10,
2,
5,11,12,13,14,15,
5,16,17,18,19,20,
2,2,
5,21,22,23,24,25,
5,26,27,28,29,30,
2,
5,31,32,33,34,35,
5,36,37,38,39,40,
2,2,2,
5,41,42,43,44,45,
5,46,47,48,49,50,
2,
5,51,52,53,54,55,
5,56,57,58,59,60,
2,2,
5,61,62,63,64,65,
5,66,67,68,69,70,
2,
5,71,72,73,74,75,
5,76,77,78,79,80,
“Int6”,6,“INTEGER”,0,“days”,
2,2,2,2,2,
5,1,2,3,4,5,
5,6,7,8,9,10,
2,
5,11,12,13,14,15,
5,16,17,18,19,20,
2,2,
5,21,22,23,24,25,
5,26,27,28,29,30,
2,
5,31,32,33,34,35,
5,36,37,38,39,40,
2,2,2,
5,41,42,43,44,45,
5,46,47,48,49,50,
2,
5,51,52,53,54,55,
5,56,57,58,59,60,
2,2,
5,61,62,63,64,65,
5,66,67,68,69,70,
2,
5,71,72,73,74,75,
5,76,77,78,79,80,
2,2,2,2,
5,81,82,83,84,85,
5,86,87,88,89,90,
2,
5,91,92,93,94,95,
5,96,97,98,99,100,
2,2,
5,101,102,103,104,105,
5,106,107,108,109,110,
2,
5,111,112,113,114,115,
5,116,117,118,119,120,
2,2,2,
5,121,122,123,124,125,
5,126,127,128,129,130,
2,
```

```
5,131,132,133,134,135,
5,136,137,138,139,140,
2,2,
5,141,142,143,144,145,
5,146,147,148,149,150,
2,
5,151,152,153,154,155,
5,156,157,158,159,160,
“RealA”,0,“FLOAT”,0,“days”,
0.0,
“RealB”,0,“FLOAT”,0,“days”,
100.0,
“Real”,0,“FLOAT”,0,“days”,
20.1,
“Real1”,1,“FLOAT”,0,“days”,
5,1.1,2.1,3.1,4.1,5.1,
“Real2”,2,“FLOAT”,0,“days”,
2,
5,1.1,2.1,3.1,4.1,5.1,
6,6.1,7.1,8.1,9.1,10.1,11.1,
“Real3”,3,“FLOAT”,0,“days”,
2,2,
5,1.1,2.1,3.1,4.1,5.1,
5,6.1,7.1,8.1,9.1,10.1,
2,
5,11.1,12.1,13.1,14.1,15.1,
5,16.1,17.1,18.1,19.1,20.1,
“Real4”,4,“FLOAT”,0,“days”,
2,2,2,
5,1.1,2.1,3.1,4.1,5.1,
5,6.1,7.1,8.1,9.1,10.1,
2,
5,11.1,12.1,13.1,14.1,15.1,
5,16.1,17.1,18.1,19.1,20.1,
2,2,
5,21.1,22.1,23.1,24.1,25.1,
5,26.1,27.1,28.1,29.1,30.1,
2,
5,31.1,32.1,33.1,34.1,35.1,
5,36.1,37.1,38.1,39.1,40.1,
“Real5”,5,“FLOAT”,0,“days”,
2,2,2,2,
5,1.1,2.1,3.1,4.1,5.1,
5,6.1,7.1,8.1,9.1,10.1,
2,
5,11.1,12.1,13.1,14.1,15.1,
5,16.1,17.1,18.1,19.1,20.1,
2,2,
5,21.1,22.1,23.1,24.1,25.1,
5,26.1,27.1,28.1,29.1,30.1,
2,
5,31.1,32.1,33.1,34.1,35.1,
5,36.1,37.1,38.1,39.1,40.1,
2,2,2,
5,41.1,42.1,43.1,44.1,45.1,
5,46.1,47.1,48.1,49.1,50.1,
2,
5,51.1,52.1,53.1,54.1,55.1,
5,56.1,57.1,58.1,59.1,60.1,
2,2,
5,61.1,62.1,63.1,64.1,65.1,
5,66.1,67.1,68.1,69.1,70.1,
2,
5,71.1,72.1,73.1,74.1,75.1,
```

```
5,76.1,77.1,78.1,79.1,80.1,
“Real6”,6,“FLOAT”,0,“days”,
2,2,2,2,2,
5,1.1,2.1,3.1,4.1,5.1,
5,6.1,7.1,8.1,9.1,10.1,
2,
5,11.1,12.1,13.1,14.1,15.1,
5,16.1,17.1,18.1,19.1,20.1,
2,2,
5,21.1,22.1,23.1,24.1,25.1,
5,26.1,27.1,28.1,29.1,30.1,
2,
5,31.1,32.1,33.1,34.1,35.1,
5,36.1,37.1,38.1,39.1,40.1,
2,2,2,
5,41.1,42.1,43.1,44.1,45.1,
5,46.1,47.1,48.1,49.1,50.1,
2,
5,51.1,52.1,53.1,54.1,55.1,
5,56.1,57.1,58.1,59.1,60.1,
2,2,
5,61.1,62.1,63.1,64.1,65.1,
5,66.1,67.1,68.1,69.1,70.1,
2,
5,71.1,72.1,73.1,74.1,75.1,
5,76.1,77.1,78.1,79.1,80.1,
2,2,2,2,
5,81.1,82.1,83.1,84.1,85.1,
5,86.1,87.1,88.1,89.1,90.1,
2,
5,91.1,92.1,93.1,94.1,95.1,
5,96.1,97.1,98.1,99.1,100.1,
2,2,
5,101.1,102.1,103.1,104.1,105.1,
5,106.1,107.1,108.1,109.1,110.1,
2,
5,111.1,112.1,113.1,114.1,115.1,
5,116.1,117.1,118.1,119.1,120.1,
2,2,2,
5,121.1,122.1,123.1,124.1,125.1,
5,126.1,127.1,128.1,129.1,130.1,
2,
5,131.1,132.1,133.1,134.1,135.1,
5,136.1,137.1,138.1,139.1,140.1,
2,2,
5,141.1,142.1,143.1,144.1,145.1,
5,146.1,147.1,148.1,149.1,150.1,
2,
5,151.1,152.1,153.1,154.1,155.1,
5,156.1,157.1,158.1,159.1,160.1,
“Log”,0,“LOGICAL”,0,
T,
“Log1”,1,“LOGICAL”,0,,
5,T,F,T,F,T,
“Log2”,2,“LOGICAL”,0,,
2,
5,T,F,T,F,T,
6,T,F,T,F,T,F,
“Log3”,3,“LOGICAL”,0,,
2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
```

```
5,T,F,T,F,T,
“Log4”,4,“LOGICAL”,0,,
2,2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
“Log5”,5,“LOGICAL”,0,,
2,2,2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
“Log6”,6,“LOGICAL”,0,,
2,2,2,2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
```

```
2,2,2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,2,
5,T,F,T,F,T,
5,T,F,T,F,T,
2,
5,T,F,T,F,T,
5,T,F,T,F,T,
“String”,0,“STRING”,0,,
“A”,
“String1”,1,“STRING”,0,,
5,“A”,“B”,“C”,“D”,“E”,
“String2”,2,“STRING”,0,,
2,
5,“A”,“B”,“C”,“D”,“E”,
6,“F”,“G”,“H”,“I”,“J”,“K”,
“String3",3,“STRING”,0,,
2,2,
5,“A”,“B”,“C”,“D”,“E”,
5,“F”,“G”,“H”,“I”,“J”,
2,
5,“K”,“L”,“M”,“N”,“O”,
5,”P”,“Q”,“R”,“S”,“T”,
“String4",4,“STRING”,0,,
2,2,2,
5,“A”,“B”,“C”,“D”,“E”,
5,“F”,“G”,“H”,“I”,“J”,
2,
5,“K”,“L”,“M”,“N”,“O”,
5,“P”,“Q”,“R”,“S”,“T”,
2,2,
5,“U”,“V”,“W”,“X”,“Y”,
5,“Z”,“AA”,“BB”,“CC”,“DD”,
2,
5,“EE”,“FF”,“GG”,“HH”,“II”,
5,“JJ”,“KK”,“LL”,“MM”,“NN”,
“String5”,5,“STRING”,0,,
2,2,2,2,
5,“A”,“B”,“C”,“D”,“E”,
5,“F”,“G”,“H”,“I”,“J”,
2,
5,“K”,“L”,“M”,“N”,“O”,
5,“P”,“Q”,“R”,“S”,“T”,
2,2,
5,“U”,“V”,“W”,“X”,“Y”,
5,“Z”,“AA”,“BB”,“CC”,“DD”,
2,
5,“EE”,“FF”,“GG”,“HH”,“II”,
```

```
5,“JJ”,“KK”,“LL”,“MM”,“NN”,
2,2,2,
5,“OO”,“PP”,“QQ”,“RR”,“SS”,
5,“TT”,“UU”,“VV”,“WW”,“XX”,
2,
5,“YY”,“ZZ”,“AAA”,“BBB”,“CCC”,
5,“DDD”,“EEE”,“FFF”,“GGG”,“HHH”,
2,2,
5,“III”,“JJJ”,“LLL”,“MMM”,“NNN”,
5,“OOO”,“PPP”,“QQQ”,“RRR”,“SSS”,
2,
5,“TTT”,“UUU”,“VVV”,“WWW”,“YYY”,
5,“ZZZ”,“AAAA”,“BBBB”,“CCCC”,“DDDD”,
“String6”,6,“STRING”,0,,
2,2,2,2,2,
5,“A”,“B”,“C”,“D”,“E”,
5,“F”,“G”,“H”,“I”,“J”,
2,
5,“K”,“L”,“M”,“N”,“O”,
5,“P”,“Q”,“R”,“S”,“T”,
2,2,
5,“U”,“V”,“W”,“X”,“Y”,
5,“Z”,“AA”,“BB”,“CC”,“DD”,
2,
5,“EE”,“FF”,“GG”,“HH”,“II”,
5,“JJ”,“KK”,“LL”,“MM”,“NN”,
2,2,2,
5,“OO”,“PP”,“QQ”,“RR”,“SS”,
5,“TT”,“UU”,“VV”,“WW”,“XX”,
2,
5,“YY”,“ZZ”,“AAA”,“BBB”,“CCC”,
5,“DDD”,“EEE”,“FFF”,“GGG”,“HHH”,
2,2,
5,“III”,“JJJ”,“LLL”,“MMM”,“NNN”,
5,“OOO”,“PPP”,“QQQ”,“RRR”,“SSS”,
2,
5,“TTT”,“UUU”,“VVV”,“WWW”,“YYY”,
5,“111”,“222”,“333”,“444”,“555”,
2,2,2,2,
5,“ZZZ”,“AAAA”,“BBBB”,“CCCC”,“DDDD”,
5,“EEEE”,“FFFF”,“GGGG”,“HHHH”,“IIII”,
2,
5,“JJJJ”,“KKKK”,“LLLL”,“MMMM”,“NNNN”,
5,“OOOO”,“PPPP”,“QQQQ”,“RRRR”,“SSSS”,
2,2,
5,“TTTT”,“UUUU”,“VVVV”,“WWWW”,“XXXX”,
5,“YYYY”,“ZZZZ”,“AAAAA”,“BBBBB”,“CCCCC”,
2,
5,“DDDDDD”,“EEEEE”,“FFFFF”,“GGGGG”,“HHHHH”,
5,“IIIII”,“JJJJJ”,“KKKKK”,“LLLLL”,“MMMMM”,
2,2,2,
5,“NNNNN”,“OOOOO”,“PPPPP”,“QQQQQ”,“RRRRR”,
5,“SSSSS”,“TTTTT”,“UUUUU”,“VVVVV”,“WWWWW”,
2,
5,“XXXXX”,“YYYYY”,“ZZZZZ”,“AAAAAA”,“BBBBBB”,
5,“CCCCCC”,“DDDDDD”,“EEEEEE”,“FFFFFF”,“GGGGGG”,
2,2,
5,“HHHHHH”,“IIIIII”,“JJJJJJ”,“KKKKKK”,“LLLLLL”,
5,“MMMMMM”,“NNNNNN”,“OOOOOO”,“PPPPPP”,“QQQQQQ”,
2,
5,“RRRRRR”,“SSSSSS”,“TTTTTT”,“UUUUUU”,“VVVVVV”,
5,“WWWWWW”,“XXXXXX”,“YYYYYY”,“ZZZZZZ”,“AAAAAAA”,
```

HD.DIC

```
33,“Code”,“Dimensions”,“DataType”,“Min”,“Max”,“Units”,“IntA”,0,“INTEGER”,0,100,“days”,“I
ntB”,0,“INTEGER”,0,100,“days”,“Int”,0,“INTEGER”,0,100,“days”
“Int1”,1,“INTEGER”,0,100,“days”
“Int2”,2,“INTEGER”,0,100,“days”
“Int3”,3,“INTEGER”,0,100,“days”
“Int4”,4,“INTEGER”,0,100,“days”
“Int5”,5,“INTEGER”,0,100,“days”
“Int6”,6,“INTEGER”,0,160,“days”
“RealA”,0,“FLOAT”,0.0,100.0,“days”
“RealB”,0,“FLOAT”,0.0,100.0,“days”
“Real”,0,“FLOAT”,0.0,100.0,“days”
“Real1”,1,“FLOAT”,0.0,100.0,“days”
“Real2”,2,“FLOAT”,0.0,100.0,“days”
“Real3”,3,“FLOAT”,0.0,100.0,“days”
“Real4”,4,“FLOAT”,0.0,100.0,“days”
“Real5’,5,“FLOAT”,0.0,100.0,“days”
“Real6”,6,“FLOAT”,0.0,160.1,“days”
“Log”,0,“Logical”,,,,
“Log1”,1,“Logical”,,,,
“Log2”,2,“Logical”,,,,
“Log3”,3,“Logical”,,,,
“Log4”,4,“Logical”,,,,
“Log5”,5,“Logical”,,,,
“Log6”,6,“Logical”,,,,
“String”,0,”String”,,,,
“String1”,1,”String”,,,,
“String2”,2,”String”,,,,
“String3”,3,”String”,,,,
“String4”,4,”String”,,,,
“String5”,5,”String”,,,,
“String6”,6,”String”,,,,
```

A.5 Example GRF

This section lists the example GRF data group (HD.GRF) and its associated data dictionary (HD.DIC).

HD.GRF

```
1,
“hd.grf”,“datagroup”,
28,
“Int”,0,“INTEGER”,0,“days”,
11,
“Int1”,1,“INTEGER”,0,“days”,
1,22,
“Int2”,2,“INTEGER”,0,“days”,
2,
0,
6, 0, 0, 0, 0, 0, 33,
“Int3”,3,“INTEGER”,0,“days”,
2, 0,
2,
0,
5, 0, 0, 0, 0, 44,
“Int4”,4, “INTEGER”,0, “days”,
```

```
2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 55,
“Int5”,5, “INTEGER”,0, “days”,
2, 0, 2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 66,
“Int6”,6, “INTEGER”,0, “days”,
2, 0, 2, 0, 2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 77,
“Real”,0,“FLOAT”,0,“days”,
11.1,
“Real1”,1,“FLOAT”,0,“days”,
5, 0, 0, 0, 0, 22.2,
“Real2”,2,“FLOAT”,0,“days”,
2,
0,
6, 0, 0, 0, 0, 0, 33.3,
“Real3”,3,“FLOAT”,0,“days”,
2, 0,
2,
0,
5, 0, 0, 0, 0, 44.4,
“Real4”,4,“FLOAT”,0,“days”,
2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 55.5,
“Real5”,5,“FLOAT”,0,“days”,
2, 0, 2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 66.6,
“Real6”,6,“FLOAT”,0,“days”,
2, 0, 2, 0, 2, 0, 2, 0,
2,
0,
5, 0, 0, 0, 0, 77.7,
“Log”,0,“LOGICAL”,0,“”,
“T”,
“Log1”,1,“LOGICAL”,0,“”,
5,“F”,“F”,“F”,“F”,“T”,
“Log2”,2,“LOGICAL”,0,“”,
2,
0,
6,“F”,“F”,“F”,“F”,“F”,“F”,
“Log3”,3,“LOGICAL”,0,“”,
2, 0,
2,
0,
5,“F”,“F”,“F”,“F”,“T”,
“Log4”,4,“LOGICAL”,0,“”,
2, 0, 2, 0,
2,
0,
4,“F”,“F”,“F”,“F”,
“Log5”,5,“LOGICAL”,0,“”,
2, 0, 2, 0, 2, 0,
2,
0,
5,“F”,“F”,“F”,“F”,“T”,
“Log6”,6,“LOGICAL”,0,“”,
2, 0, 2, 0, 2, 0, 2, 0,
2,
0,
```

```
4,“F”,“F”,“F”,“F”,
“String”,0,“STRING”,0,“”,
“A1”,
“String1”,1,“STRING”,0,“”,
5,“”,“”,“”,“”,”E2”,
“String2”,2,“STRING”,0,“”,
2,
0,
5,“”,“”,“”,“”,”J3”,
“String3”,3,“STRING”,0,“”,
2, 0,
2,
0,
5,“”,“”,“”,“”,”T4”,
“String4”,4,“STRING”,0,“”,
2, 0, 2, 0,
2,
0,
5,“”,“”,“”,“”,”NN5”,
“String5”,5,“STRING”,0,“”,
2, 0, 2, 0, 2, 0,
2,
0,
5,“”,“”,“”,“”,”DDDD6”,
“String6”,6,“STRING”,0,“”,
2, 0, 2, 0, 2, 0, 2, 0,
2,
0,
5,“”,“”,“”,“”,”AAAAAAA7”,
```

HD.DIC

```
33,
“Code”,“Dimensions”,“DataType”,“Min”,“Max”,“Units”,
“IntA”,0,“INTEGER”,0,100,“days”
“IntB”,0,“INTEGER”,0,100,“days”
“Int”,0,“INTEGER”,0,100,“days”
“Int1”,1,“INTEGER”,0,100,“days”
“Int2”,2,“INTEGER”,0,100,“days”
“Int3”,3,“INTEGER”,0,100,“days”
“Int4”,4,“INTEGER”,0,100,“days”
“Int5”,5,“INTEGER”,0,100,“days”
“Int6”,6,“INTEGER”,0,160,“days”
“RealA”,0,“FLOAT”,0.0,100.0,“days”
“RealB”,0,“FLOAT”,0.0,100.0,“days”
“Real”,0,“FLOAT”,0.0,100.0,“days”
“Real1”,1,“FLOAT”,0.0,100.0,“days”
“Real2”,2,“FLOAT”,0.0,100.0,“days”
“Real3”,3,“FLOAT”,0.0,100.0,“days”
“Real4”,4,“FLOAT”,0.0,100.0,“days”
“Real5”,5,“FLOAT”,0.0,100.0,“days”
“Real6”,6,“FLOAT”,0.0,160.1,“days”
“Log”,0,“Logical”,,,,
“Log1”,1,“Logical”,,,,
“Log2”,2,“Logical”,,,,
“Log3”,3,“Logical”,,,,
“Log4”,4,“Logical”,,,,
“Log5”,5,“Logical”,,,,
“Log6”,6,“Logical”,,,,
“String”,0,”String”,,,,
“String1”,1,”String”,,,,
“String2”,2,”String”,,,,
“String3”,3,”String”,,,,
```

```
“String4”,4,”String”,,,,
“String5”,5,”String”,,,,
“String6”,6,”String”,,,,
```

Appendix B

Testing the Distribution Statistics Processor

Appendix B-Testing the Statistical Dynamic Link Library

The twelve distributions available to the Statistical Dynamic Link Library (STAT.DLL) are

Normal
Log Normal
Exponential
Uniform
Johnson's SB
Johnson's SU
Empirical
Triangular
Trans Log Normal
Gamma
Weibull
Integer Uniform

Test Cases

The STAT.DLL was tested independently of the HWIR software. A front-end was created (RUN.BAT) which provided the input/output (I/O) parameters for the STAT.DLL. This front-end batch routine was given the test data file (xxx.tst) name as input, and generated an output file with the results (xxx.out). The input test data file contains all of the data parameters, calls to the various distribution subroutines, and correlation variables. A sample test data file is presented below.

Testing Each Distribution

Testing software to indicate, with some level of confidence, that the software is, indeed, creating the specified distribution within the specified parameters is a little different from any other typical statistical test. This is because in every other situation, the alternative hypothesis is the test in question, and the null hypothesis is the "status quo." In every other case, we are looking for enough information to reject the status quo in favor of a new hypothesis. In testing software to indicate that it is creating certain distributions within specified parameters, the status quo is the situation that is desired. Therefore, the favored output is that the tests *fail*; that is, enough information is gathered so that the status quo is *not* rejected in favor of a new hypothesis (the new hypothesis being that the output distribution is some distribution other than the one specified in the null hypothesis).

A p-value is the probability that the status quo is rejected when, in fact, it is true. In cases where testing is such that the desired outcome is to reject the null hypothesis (the status quo), a p-value of .05 is typically allowed, and it is desired that the calculated p-value be as close to zero as possible. In testing software for outcomes of distributions, the opposite is true. The desired p-value is one that is as close to one as possible. The closer it is to 1.0, the higher the probability that the distribution is not something other than that specified by the null hypothesis.

For each distribution in the following sections, only the results from the Kolmogorov-Smirnoff tests are shown. This is because all three tests will show the same results, and the redundancy of information is not needed. The Chi-Square Goodness of Fit tests and the Anderson-Darling tests are available, however, upon request.

B.1 Test of the Normal Distribution

The normal distribution was tested using established software containing a proven random number generator. Several tests were performed: the Chi-square Goodness of Fit Test, the Kolmogorov-Smirnoff Test, and the Anderson-Darling test. The normal distribution passed 93% (2 out of 30) of the tests, which is within normal parameters (distributions generated from established software will fail tests approximately 10% of the time). The results of the Kolmogorov-Smirnoff tests are shown below.

Run #	Mean	Std Dev	Min	Max	K-S	p-value
1)	29	3.56	0	100	0.036	0.5
2)	50	10	0	100	0.063	0.5
3)	12	8.291	0	100	0.072	0.5
4)	2	0.05	0	100	0.04	0.5
5)	73.6	6.66	30	100	0.064	0.5
6)	9	2.5	0	100	0.054	0.5
7)	38.4	11.1	0	100	0.052	0.5
8)	42.08	0.075	0	100	0.085	0.07
9)	2	0.01	0	100	0.068	0.5
10)	22.89	2.289	0	100	0.054	0.5
11)	19.481	3.502	0	80	0.058	0.5
12)	52.5	3.55	40	80	0.077	0.5
13)	72	2.5	0	80	0.067	0.5
14)	32	10.1	0	60	0.046	0.5
15)	9.375	2.5	0	60	0.042	0.5
16)	2.22	0.92	0	60	0.071	0.5
17)	8.32	0.03	0	40	0.062	0.5
18)	12	1.9	0	40	0.05	0.5
19)	12	3.5	0	40	0.074	0.5
20)	12	3.5	0	20	0.075	0.5
21)	10	1	0	20	0.069	0.5
22)	10	0.095	0	20	0.048	0.5
23)	2	0.4	0	10	0.078	0.5
24)	3	0.4	0	10	0.056	0.5
25)	4	0.4	0	10	0.044	0.5
26)	5	0.4	0	10	0.064	0.5
27)	6	0.4	0	10	0.054	0.5
28)	7	0.4	0	10	0.091	0.04*
29)	8	0.4	0	10	0.051	0.5
30)	5	2.5	0	10	0.092	0.04*

* This distribution did not show enough evidence to believe the status quo, that is, that the distribution being tested is a normal distribution.

B.2 Test of the Log Normal Distribution

The lognormal distribution was tested using established software containing a proven random number generator. Several tests were performed: the Chi-square Goodness of Fit Test, the Kolmogorov-Smirnoff Test, and the Anderson-Darling test. The lognormal distribution passed 100% of the tests, which is well within normal parameters (distributions generated from established software will fail tests approximately 10% of the time). The results of the Kolmogorov-Smirnoff tests are shown below.

Run	Meanlog	Sdlog	K-S	p-value
1)	10.000	1.000	0.570	0.91
2)	0.500	0.001	0.530	0.95
3)	0.120	0.008	0.590	0.88
4)	0.200	0.050	0.570	0.91
5)	0.736	0.660	0.710	0.70
6)	0.900	0.250	0.820	0.52
7)	0.384	0.111	0.520	0.95
8)	0.421	0.075	0.560	0.92
9)	0.891	0.010	0.610	0.86
10)	0.229	0.013	0.520	0.95
11)	0.019	<.0001	0.530	0.95
12)	1.020	0.355	0.550	0.93
13)	1.720	0.500	0.740	0.65
14)	0.320	0.100	0.520	0.95
15)	0.938	0.125	0.520	0.95
16)	0.220	0.009	0.530	0.95
17)	0.083	<.0001	0.510	0.96
18)	0.120	0.029	0.520	0.95
19)	0.120	0.005	0.580	0.90
20)	0.100	0.009	0.570	0.91
21)	0.100	0.005	0.510	0.96
22)	0.090	0.010	0.610	0.86
23)	0.200	0.004	0.500	0.97
24)	0.300	0.004	0.560	0.92
25)	0.040	<.0001	0.510	0.96
26)	0.500	0.004	0.560	0.92
27)	0.600	0.004	0.530	0.95
28)	0.700	0.005	0.530	0.95
29)	0.800	0.040	0.520	0.95
30)	0.500	0.010	0.530	0.95

B.3 Test of the Exponential Distribution

The exponential distribution was tested using established software containing a proven random number generator. Several tests were performed: the Chi-square Goodness of Fit Test, the Kolmogorov-Smirnoff Test, and the Anderson-Darling test. The exponential distribution passed all of its tests, which is within normal parameters (distributions generated from established software will fail tests approximately 10% of the time). Notice that run number 23 has a star by it. This is because the p-value, although above the threshold of .05, is very close to failing. However, this is the only distribution that came close to failing, and therefore, the outputs for the exponential distribution are well within expected parameters. The results of the Kolmogorov-Smirnoff tests are shown below:

Run #	Rate	Min	Max	K-S	p-value
1)	1/2.9	0	100	0.106	0.20
2)	1/5	0	100	0.072	0.50
3)	1/1.2	0	100	0.108	0.19
4)	1/2	0	100	0.071	0.50
5)	1/3.6	0	100	0.063	0.50
6)	1/.8	0	100	0.065	0.50
7)	1/.4	0	100	0.049	0.50
8)	1/.08	0	100	0.079	0.50
9)	1/1.34	0	100	0.117	0.12
10)	1/.89	0	100	0.091	0.36
11)	1/11.48	0	80	0.092	0.36
12)	1/12.5	0	800	0.066	0.50
13)	1/12.31	0	1800	0.061	0.50
14)	1/19.99	0	6000	0.063	0.50
15)	1/1.05	0	60	0.082	0.50
16)	1/1.22	0	60	0.079	0.50
17)	1/.94	0	40	0.108	0.19
18)	1/3	0	400	0.071	0.50
19)	1/4.01	0	400	0.067	0.50
20)	1/8.99	0	2000	0.076	0.50
21)	1/3.6	0	20	0.063	0.50
22)	1/.5	0	200	0.083	0.50
23)	1/.4	0	10	0.130	0.06*
24)	1/.3	0	10	0.077	0.50
25)	1/.2	0	100	0.065	0.50
26)	1/.1	0	100	0.080	0.50
27)	1/1.1	0	100	0.067	0.50
28)	1/1.2	0	100	0.063	0.50
29)	1/1.4	0	10	0.098	0.28
30)	1/1.6	0	10	0.114	0.14

B.4 Test of the Uniform Distribution

The uniform distribution was tested using established software containing a proven random number generator. Several tests were performed: the Chi-square Goodness of Fit Test, the Kolmogorov-Smirnoff Test, and the Anderson-Darling test. The uniform distribution passed 100% of the tests, which is well within normal parameters (distributions generated from established software will fail tests approximately 10% of the time). The p-value results of the Kolmogorov-Smirnoff tests are shown below. Notice that there are 29 runs instead of 30. This is because one of the runs had unreasonable inputs and was thrown out of the analysis.

Run #	Min	Max	K-S	p-value
1)	0.000	1	0.079	0.56
2)	0.000	2	0.070	0.71
3)	0.000	3.000	0.118	0.13
4)	0.000	4.000	0.078	0.58
5)	0.000	5.000	0.083	0.50
6)	0.000	6.000	0.103	0.24
7)	1.000	2.000	0.083	0.50
8)	1.000	3.000	0.050	0.96
9)	1.000	4.000	0.090	0.40
10)	1.000	5.000	0.093	0.35
11)	1.000	6.000	0.078	0.58
12)	2.000	3.000	0.094	0.33
13)	2.000	4.000	0.094	0.34
14)	2.000	5.000	0.078	0.58
15)	2.000	6.000	0.087	0.44
16)	2.000	10.000	0.096	0.32
17)	0.000	1.500	0.068	0.74
18)	0.000	2.500	0.069	0.73
19)	0.100	0.990	0.065	0.80
20)	48.8	73.200	0.090	0.39
21)	24.4	36.600	0.102	0.25
22)	24.4	36.600	0.075	0.63
23)	24.4	36.600	0.053	0.94
24)	48.8	73.200	0.112	0.17
25)	48.8	73.200	0.126	0.09
26)	73.2	110.000	0.063	0.83
27)	24.4	36.600	0.079	0.57
28)	24.4	36.600	0.066	0.77
29)	24.4	36.600	0.094	0.34

B.5 Test of the Johnson's SB Distribution

The Johnson SB distribution was tested using established software containing a proven random number generator. Several tests were performed: the Chi-square Goodness of Fit Test, the Kolmogorov-

Smirnoff Test, and the Anderson-Darling test. The Johnson SB distribution passed 90% of the tests, which is within normal parameters (distributions generated from established software will fail tests approximately 10% of the time). The p-value results of the Kolmogorov-Smirnoff tests are shown below.

Run #	A	B	min	max	K-S	p-value
1)	3.320	0.880	0	1000	0.053	0.5
2)	0.500	0.100	0	100	0.107	0.01
3)	0.070	0.011	0	100	0.051	0.5
4)	2.000	0.050	0	100	0.053	0.5
5)	3.600	6.660	30	10000	0.053	0.5
6)	1.000	2.500	0	10000	0.053	0.5
7)	0.400	1.081	0	100	0.053	0.5
8)	1.080	0.075	0	100	0.053	0.5
9)	2.000	0.010	0	1000	0.053	0.5
10)	2.890	0.289	0	1000	0.053	0.5
11)	1.481	1.502	0	80	0.053	0.5
12)	0.450	4.055	40	80	0.053	0.5
13)	2.000	12.500	0	80	0.053	0.5
14)	2.000	10.100	0	60	0.053	0.5
15)	0.375	2.500	0	60	0.053	0.5
16)	0.022	0.920	0	60	0.053	0.5
17)	0.083	0.030	0	40	0.056	0.5
18)	2.000	1.900	0	400	0.041	0.5
19)	2.900	3.500	0	4000	0.069	0.5
20)	0.001	3.500	0	200	0.061	0.5
21)	10.000	1.000	0	2000	0.065	0.5
22)	10.000	20.095	0	2000	0.114	0.01
23)	2.000	8.460	0	1000	0.09	0.05
24)	0.052	13.480	0	1000	0.046	0.5
25)	0.085	1.490	0	100	0.055	0.5
26)	0.010	4.440	0	100	0.101	0.01
27)	1.000	0.410	0	1000	0.078	0.5
28)	2.000	4.440	0	1000	0.062	0.5
29)	0.006	5.490	0	10000	0.048	0.5
30)	5.000	2.520	0	10000	0.072	0.5

B.6 Test of the Johnson's SU Distribution

There are problems with this distribution. This distribution is a transformation of the normal distribution. Therefore, the transformation back to a normal is implemented, and then the resulting normal distribution is tested. Only 50% of the 30 tests passed for this distribution. Therefore, it is recommended that this particular distribution not be used until further modifications are made. Actual results are not shown, but are available upon request.

B.7 Test of the Empirical Distribution

The empirical distribution outputs are integer uniform distributions, which have already been proven to pass 97% of the time, which is well within normal parameters.

B.8 Test of the Triangular Distribution

The triangular distribution is not a common distribution in statistics, and typical statistical packages do not carry the testing of the distribution within their realms. Steps are being taken to alleviate this problem, and testing results of this distribution are still pending. However, histograms of the output were plotted and visually inspected. The expectation, given these histograms, is that the triangular distribution will pass the tests and be available for use. These histograms are available upon request.

B.9 Test of the Trans Log Normal Distribution

The Trans Log Normal distribution was tested using established software containing a proven random number generator. Several tests were performed: the Chi-square Goodness of Fit Test, the Kolmogorov-Smirnoff Test, and the Anderson-Darling test. The Trans Log Normal distribution passed 100% of the tests, which is well within normal parameters (distributions generated from established software will fail tests approximately 10% of the time). The p-value results of the Kolmogorov-Smirnoff tests are shown below. Since 100% passed, the testing was stopped at 20 runs due to time constraints.

Run#	Mean	SD	Min	Max	K-S	P-value
1)	50.00	10.00	0.00	100.00	1.00	0.28
2)	1.70	1.00	0.00	2.00	0.55	0.93
3)	3.91	2.30	0.91	4.61	1.00	0.28
4)	50.00	10.00	0.00	100.00	0.91	0.39
5)	1.70	1.00	0.00	100.00	0.95	0.33
6)	3.91	2.30	0.00	100.00	0.91	0.39
7)	0.60	0.06	0.00	10.00	1.00	0.28
8)	1.46	0.55	0.01	64.00	1.00	0.28
9)	1.00	0.10	0.00	5.00	1.00	0.28
10)	0.20	0.02	0.00	3.00	1.00	0.28
11)	1.90	0.45	0.05	28.50	1.00	0.28
12)	2.80	0.85	0.05	900.00	1.00	0.28
13)	1.46	0.55	0.01	14.00	0.99	0.29
14)	1.00	0.10	0.00	5.00	1.00	0.28
15)	1.20	0.29	0.00	3.00	1.00	0.28
16)	1.90	0.45	0.05	280.50	1.00	0.28
17)	2.80	0.85	0.05	4000.40	1.00	0.28
18)	0.84	0.55	0.02	100.00	0.94	0.35
19)	1.00	0.26	0.00	1000.00	1.00	0.28
20)	5.85	0.95	0.00	1000000	1.00	0.28

B.10 Test of the Gamma Distribution

The gamma distribution was tested using established software containing a proven random number generator. Several tests were performed: the Chi-square Goodness of Fit Test, the Kolmogorov-Smirnoff Test, and the Anderson-Darling test. The gamma distribution passed 100% of the tests, which is well within normal parameters (distributions generated from established software will fail tests approximately 10% of the time). The p-value results of the Kolmogorov-Smirnoff tests are shown below.

Run	Shape	Scale	Min	Max	K-S	P-value
1)	1	0.25	0	1000	1.000	0.28
2)	1	0.50	0	1000	0.920	0.37
3)	1	1.00	0	1000	0.560	0.92
4)	1	1.50	0	1000	0.550	0.93
5)	1	2.00	0	1000	0.550	0.93
6)	1	2.50	0	1000	0.740	0.65
7)	1	3.00	0	1000	0.720	0.69
8)	2	0.25	0	1000	0.930	0.36
9)	2	0.50	0	1000	0.600	0.87
10)	2	1.00	0	1000	0.700	0.72
11)	2	1.50	0	1000	0.850	0.47
12)	2	2.00	0	1000	0.870	0.44
13)	2	2.50	0	1000	0.950	0.33
14)	2	3.00	0	1000	1.000	0.28
15)	3	0.10	0	1000	0.990	0.29
16)	3	0.70	0	1000	0.760	0.62
17)	3	1.60	0	1000	0.980	0.30
18)	3	2.10	0	1000	1.000	0.28
19)	3	3.00	0	1000	1.000	0.28
20)	4	0.50	0	1000	0.850	0.47
21)	4	1.00	0	1000	0.990	0.29
22)	4	2.00	0	1000	1.000	0.28
23)	4	3.00	0	1000	1.000	0.28
24)	0.5	0.10	0	1000	1.000	0.28
25)	0.5	0.70	0	1000	1.000	0.28
26)	0.5	1.40	0	1000	1.000	0.28
27)	0.5	2.50	0	1000	1.000	0.28
28)	0.25	0.20	0	1000	1.000	0.28
29)	0.25	0.40	0	1000	1.000	0.28
30)	0.25	0.75	0	1000	0.960	0.32

B.11 Test of the Weibull Distribution

The weibull distribution was tested using established software containing a proven random number generator. Several tests were performed: the Chi-square Goodness of Fit Test, the Kolmogorov-

Smirnoff Test, and the Anderson-Darling test. The weibull distribution passed 90% of the tests, which is within normal parameters (distributions generated from established software will fail tests approximately 10% of the time). The p-value results of the Kolmogorov-Smirnoff tests are shown below. Notice that for run number 30, the p-value is close to .05, indicating that there could be a possibility that the test could fail. However, other tests (not published here) were run, and the software was able to stay within the allowed 10% failure rate. Also, notice that the three that failed had a very high shape parameter, which may be an unreasonable parameter to input.

Run#	Shape	Scale	Min	Max	K-S	p-value
1)	29.000	3.560	0	100	0.117	0.13
2)	50.000	10.000	0	100	0.052	0.95
3)	12.000	8.291	0	100	0.089	0.40
4)	2.000	0.050	0	100	0.079	0.57
5)	73.600	6.660	30	100	1.000	0.00*
6)	9.000	2.500	0	100	0.054	0.93
7)	38.400	11.100	0	100	0.073	0.66
8)	42.080	0.075	0	100	0.055	0.92
9)	2.000	0.010	0	100	0.070	0.71
10)	22.890	2.289	0	100	0.080	0.54
11)	19.481	3.502	0	80	0.064	0.81
12)	52.500	3.550	40	80	1.000	0.00*
13)	72.000	2.500	0	80	0.069	0.73
14)	32.000	10.100	0	60	0.069	0.73
15)	9.375	2.500	0	60	0.080	0.55
16)	2.220	0.920	0	60	0.068	0.75
17)	8.320	0.030	0	40	0.115	0.14
18)	12.000	1.900	0	40	0.062	0.84
19)	12.000	3.500	0	40	0.052	0.95
20)	12.000	3.500	0	20	0.100	0.27
21)	10.000	1.000	0	20	0.071	0.69
22)	10.000	0.095	0	20	0.068	0.75
23)	2.000	0.400	0	10	0.149	0.02*
24)	3.000	0.400	0	10	0.062	0.84
25)	4.000	0.400	0	10	0.087	0.43
26)	5.000	0.400	0	10	0.061	0.85
27)	6.000	0.400	0	10	0.052	0.95
28)	7.000	0.400	0	10	0.059	0.88
29)	8.000	0.400	0	10	0.118	0.12
30)	5.000	2.500	0	10	0.134	0.06

B.12 Test of the Integer Uniform Distribution

The Integer uniform distribution was tested using established software containing a proven random number generator. Several tests were performed: the Chi-square Goodness of Fit Test, the

Kolmogorov-Smirnoff Test, and the Anderson-Darling test. The integer uniform distribution passed 97% of the tests, which is well within normal parameters (distributions generated from established software will fail tests approximately 10% of the time). The p-value results of the Kolmogorov-Smirnoff tests are shown below. Notice that two other runs (2 and 30) have stars next to them. These two runs barely passed the test, using a threshold p-value of .05 (typically used). However, the number of failures stays within the allowed 10%, and other tests were run, which confirm that the output for this distribution will pass 90% of the time. Results of the other tests that were performed are available upon request.

Run	Min	Max	K-S	P-value
1)	1	30	0.112	0.16
2)	1	10	0.129	0.07*
3)	1	50	0.087	0.44
4)	1	100	0.078	0.58
5)	1	300	0.096	0.32
6)	1	500	0.053	0.94
7)	10	500	0.074	0.65
8)	10	1000	0.055	0.92
9)	5	100	0.071	0.70
10)	6	200	0.082	0.51
11)	7	200	0.062	0.84
12)	8	200	0.066	0.78
13)	20	500	0.069	0.73
14)	10	1010	0.069	0.73
15)	13	2500	0.080	0.55
16)	22	920	0.068	0.75
17)	32	305	0.118	0.13
18)	4	190	0.063	0.82
19)	43	350	0.053	0.94
20)	19	3500	0.100	0.27
21)	4	100	0.075	0.62
22)	23	943	0.068	0.75
23)	2	4881	0.149	0.02*
24)	3	499	0.063	0.82
25)	4	40	0.098	0.29
26)	5	60	0.066	0.77
27)	6	90	0.052	0.95
28)	7	234	0.058	0.89
29)	8	100	0.118	0.12
30)	5	250	0.134	0.06*

B.13 Test of Correlated Variables: A Normal Distribution Correlated with another Normal Distribution, A Normal Distribution with a Lognormal Distribution, and a Lognormal Distribution with another Lognormal Distribution

Since there is no formal test for correlations, several runs specifying a range of correlations between distributions were run, their output correlations were calculated, and the results were compared. The output is shown below. Since a criteria is needed for "closeness," it was decided that +/-.1 would be a reasonable criteria. As can be seen below, 7 of the 27 runs did not pass this criteria. However, the fully bolded runs (#15-23) were for multiple correlations. In those 9 runs, 4 of the 7 did not pass criteria. Therefore, these runs are considered not valid to regular correlations, and the results are 3 out of the remaining 18 runs did not pass. This is 16%, but since the total number of runs was reduced, this is acceptable. However, it is recommended that the user not try to use multiple correlations, as the module does not perform well in this situation. This is the recommendation for *multiple correlations* until further modifications to the module are implemented.

Run#	Distributions being correlated	Specified Correlation	Actual Correlation
1)	normal, normal	0.5	**0.52429384**
2)	normal, lognormal	0.9	**0.89935545**
3)	lognormal, lognormal	0.7	**0.55413721***
4)	normal, normal	0.9	**0.8672213**
5)	normal, lognormal	0.5	**0.5313597**
6)	lognormal, lognormal	0.9	**0.88119401**
7)	normal, normal	0.9	**0.828575708**
8)	normal, normal	0.3	**0.456752599***
9)	lognormal, lognormal	0.5	**0.44220488**
10)	lognormal, lognormal	0.3	**0.21058561**
11)	normal, lognormal	0.9	**0.902628902**
12)	normal, lognormal	0.3	**0.431027811***
13)	normal, lognormal	0.1	**0.13391854**
14)	normal, lognormal	0.6	**0.5297802**
*15)	**normal, normal**	**0.9**	**0.12645320***
*16)	**normal, normal**	**0.3**	**0.41698873***
*17)	**normal, lognormal**	**0.1**	**-0.05542675***
*18)	**lognormal, lognormal**	**0.6**	**0.65182132**
*19)	**lognormal, lognormal**	**0.85**	**0.871903372**
*20)	**normal, normal**	**0.19**	**0.22515805**
*21)	**normal, normal**	**0.44**	**-0.06610354***
*22)	**normal, lognormal**	**0.81**	**0.85906118**
*23)	**lognormal, lognormal**	**0.66**	**0.71922156**
24)	normal, normal	0.19	**0.189618973**
25)	normal, lognormal	0.44	**0.46666838**
26)	normal, lognormal	0.81	**0.85755379**
27)	lognormal, lognormal	0.66	**0.761354305**