Bladed user-defined controllers in versions prior to version 4.4

In Bladed version 4.4 the interface method to a user defined external controller was changed from the “swap” array to a function based interface. This document describes the old “swap” array interface.

The user-defined controller may be written in any language, either as a DOS or Windows executable program (.exe) capable of reading and writing to shared files, or as a 32-bit DLL (dynamic link library). A DLL is preferable as it will result in faster simulations, and communication with *Bladed* may be more reliable.

The user-defined controller is ignored in the case of Hardware Test simulations.

Writing a user-defined controller as an executable program

If the controller is written as an executable, it will use a shared file for two-way communication with *Bladed*. When the simulation starts up, the controller executable (.exe) file is first copied into the directory where *Bladed* is installed, and renamed **discon.exe**. When **discon.exe** starts up, the *Bladed* directory is the current directory, and the files used to communicate between the simulation and the controller are in this directory. The controller program can therefore refer to these files by name without giving the full path.

Two files are used to communicate between the user-defined controller program and the simulation. One of these is a text file named **discon.aux**, which is written by the simulation and just contains the directory and run name for the simulation results. This may be useful if the controller wishes to write any permanent record of what it does to be stored with the simulation results. The file has just two lines: the first consists of the word **PATH** followed by a space and then the path for the simulation results (including the final backslash). The second line consists of the word **RUNNAME** followed by a space and then the run name, i.e. up to 8 characters. The controller may choose to ignore this file if the information is not required.

The second file is used for the dynamic information exchange between the two programs. It is called **discon.swp**, and is a binary file with a record length of 4 bytes. The file must be opened
as a shared file, allowing simultaneous read and write access to both programs. The file structure is given in Appendix A.

Although this file has many records, it may only be necessary for the external controller to read from and write to a small number of these, depending on the turbine type and the tasks which the external controller is performing.

Handshaking: record 1 of **discon.swp** is used for handshaking, to ensure that neither program starts reading data until the other program has finished writing it. The sequence of events to be followed by the controller program is as follows:

1. Controller program starts by creating **discon.swp** and writing a zero to record 1.
2. Controller program waits until record 1 becomes 1 or -1. If it is 1, this indicates that the simulation has finished writing data, and also that the file **discon.aux** is ready if required. If it is -1, the simulation is about to finish and the controller program should stop.
3. If the first record is 1, the controller may read any of the parameters written by the simulation, perform its calculations, and then write the appropriate outputs. Once all data is written, the controller writes a zero to record 1 to tell the simulation that it is ready. If the controller decides to abort the simulation, it should write -1 to record 1, and write an appropriate message to **discon.swp** as described in Appendix A.
4. Controller returns to step 2.

In step 2, the controller waits until record 1 becomes non-zero. It is important that the controller closes and re-opens **discon.swp** every time around the loop, otherwise the contents of the file as represented in the disk cache may not have been updated.

Writing a user-defined controller as a dynamic link library

A dynamic link library provides faster and more reliable communications between the controller and the ***Bladed*** simulation, and is recommended. The interface to ***Bladed*** is also simpler to write.

When the simulation starts up, the controller DLL file is loaded into memory from the path supplied by the user in **Supervisory Control**. In ***Bladed*** v3.82 and earlier, the controller DLL file is first copied into a temporary subfolder of the directory where ***Bladed*** is installed (known as the "run folder") and renamed **discon.dll**. In all versions of ***Bladed*** the run folder is the working directory while the simulation is running, but contains **discon.dll** only in ***Bladed*** v3.82 and earlier.

The controller is written as a subroutine or procedure. The DLL export name of the procedure must evaluate to DISCON (note: this name must be in upper case). Depending on the language system being used, it may be necessary to define this by means of an alias. The procedure does not generate a return value. It has five arguments, as follows (the names given here are arbitrary, and are given purely for ease of reference within this manual. Only the order is important):

"DATA" The address of the first record of an array of single-precision (4-byte) real numbers which is used for data exchange between the simulation and the controller. The contents of the array is given in Appendix A.

"FLAG" A 4-byte integer (passed by reference) which the DLL should set as follows:

0 if the DLL call was successful
>0 if the DLL call was successful but the "MESSAGE" should be issued as a warning message. The simulation will continue.
<0 if the DLL call was unsuccessful or for any other reason the simulation is to be stopped at this point. "MESSAGE" is then issued as an error message.

"INFILE" The address of the first record of an array of 1-byte characters giving the name of the parameter file, which is currently DISCON.IN (See section 5.9.3). This array should not be modified by the DLL. The number of characters in the name is given in "DATA" - see Appendix A.

"OUTNAME" The address of the first record of an array of 1-byte characters giving the simulation run name, prefixed by the full path to the directory which will contain the simulation results. This may be useful if the controller wishes to write a permanent record of what it does to be stored with the simulation results: the results should be stored in a file whose name (including path) is generated by appending ".xxx" to "OUTNAME", where xxx is any suitable file extension **not** beginning with "%". The number of characters in the name is given in "DATA" - see Appendix A. Alternatively, or in addition, the DLL may send information back to Bladed for output in the same form as the other simulation results. This is described in Appendix A.

"MESSAGE" The address of the first record of an array of 1-byte characters which may be used by the DLL to send a text message to ***Bladed***, which appears on the screen and is stored together with any other calculation messages generated by ***Bladed***.

Communication Between *Bladed* And External Controllers

The following describes in detail the communication between ***Bladed*** and the user's external controller code.

Data exchange records

External controllers compiled as executable (.EXE) files exchange information with the ***Bladed*** simulation through a shared binary file consisting of a number of 4-byte records. External controllers compiled as DLLs exchange information through an array passed as the first argument to the DLL. The structure of the binary file used for EXEs and the array used for DLLs is similar and is described in the tables which follow. The type of each record of the file or element of the array may be integer, real or character, as specified in the tables. In the EXE case, the 4-byte records in the file should be written to and/or read in as 4-byte integers, 4-byte (single precision) real (i.e. floating point) numbers, or groups of 4 characters as appropriate. In the DLL case, all the array elements are passed as real numbers, so if an element is described as type Integer, the real number must be converted to the nearest integer (and integers being sent back to the simulation must be converted to real values). Character variables are passed in separate arrays in the DLL case.

Table 1 shows the array elements or binary file records which are used for data exchange between the ***Bladed*** simulation and the external controller. As shown by the 'Data flow' column, some records are used to pass information from the simulation to the controller, some are used to pass information from the controller back to the simulation, and a few are used for two-way communication.

Note that the first binary file record or array element is referred to as record or element number 1 (not 0).

Table 1

Record number	Data flow[8]	Data type[9]	Description	See note(s)	Units
1	in	I	See Section A.2		-
2	in	R	Current time		s
3	in	R	Communication interval		s
4	in	R	Blade 1 pitch angle		rad
5	in	R	Below-rated pitch angle set-point	1	rad
6	in	R	Minimum pitch angle	1	rad
7	in	R	Maximum pitch angle	1	rad
8	in	R	Minimum pitch rate (most negative value allowed)		rad/s
9	in	R	Maximum pitch rate		rad/s
10	in	I	0 = pitch position actuator, 1 = pitch rate actuator		-
11	in	R	Current demanded pitch angle		rad
12	in	R	Current demanded pitch rate		rad/s
13	in	R	Demanded power	2	W
14	in	R	Measured shaft power	3	W
15	in	R	Measured electrical power output		W
16	in	R	Optimal mode gain	3,5	$Nm/(rad/s)^2$
17	in	R	Minimum generator speed	3	rad/s
18	in	R	Optimal mode maximum speed	3	rad/s
19	in	R	Demanded generator speed above rated	1,3	rad/s
20	in	R	Measured generator speed		rad/s
21	in	R	Measured rotor speed		rad/s
22	in	R	Demanded generator torque above rated	3	Nm
23	in	R	Measured generator torque	3	Nm
24	in	R	Measured yaw error	4	rad
25	in	I	Start of below-rated torque-speed look-up table =R	3,5	Record no.
26	in	I	No. of points in torque-speed look-up table =N	3,5	-
27	in	R	Hub wind speed	4	m/s
28	in	I	Pitch control: 0 = collective, 1 = individual		-
29	in	I	Yaw control: 0 = yaw rate control, 1 = yaw torque control		-
30-32	in	R	Blade 1-3 root out of plane bending moment	18	Nm
33	in	R	Blade 2 pitch angle		rad
34	in	R	Blade 3 pitch angle		rad
35	both	I	Generator contactor	10	-
36	both	I	Shaft brake status: 0=off, 1=Brake 1 on	19	-
37	in	R	Nacelle angle from North		rad
38-40	out		Reserved		
41	out	R	Demanded yaw actuator torque	13,21	Nm
42	out	R	Demanded blade 1 individual pitch position or rate	12,14	rad or rad/s
43	out	R	Demanded blade 2 individual pitch position or rate	12,14	rad or rad/s
44	out	R	Demanded blade 3 individual pitch position or rate	12,14	rad or rad/s
45	out	R	Demanded pitch angle (Collective pitch)	12	rad
46	out	R	Demanded pitch rate (Collective pitch)	12	rad/s
47	out	R	Demanded generator torque		Nm
48	out	R	Demanded nacelle yaw rate	13,21	rad/s
49	out	I	Message length OR -M0	15	-
49	in	I	Maximum no. of characters allowed in the "MESSAGE"	6	-
50	in	I	No. of characters in the "INFILE" argument	6	-

....continued overleaf....

Record number	Data flow[8]	Data type[9]	Description	See note(s)	Units
51	in	I	No. of characters in the “OUTNAME” argument	6	-
52	in	I	DLL interface version number (reserved for future use)	6	-
53	in	R	Tower top fore-aft acceleration		m/s^2
54	in	R	Tower top side to side acceleration		m/s^2
55	out	I	Pitch override	16	-
56	out	I	Torque override	16	-
57-59	out		Reserved		
60	in	R	Rotor azimuth angle		rad
61	in	I	No. of blades		-
62	in	I	Max. number of values which can be returned for logging	7	-
63	in	I	Record number for start of logging output	7	-
64	in	I	Max. no. of characters which can be returned in “OUTNAME”	7	-
65	out	I	Number of variables returned for logging	17	-
66-68	in	R	Reserved		
69-71	in	R	Blade 1-3 root in plane bending moment	18	Nm
72	out	R	Generator start-up resistance		ohm/phase
73	in	R	Rotating hub My (GL co-ords)	18	Nm
74	in	R	Rotating hub Mz (GL co-ords)	18	Nm
75	in	R	Fixed hub My (GL co-ords)	18	Nm
76	in	R	Fixed hub Mz (GL co-ords)	18	Nm
77	in	R	Yaw bearing My (GL co-ords)	18	Nm
78	in	R	Yaw bearing Mz (GL co-ords)	18	Nm
79	out	I	Request for loads	18	-
80	out	I	1 = Variable slip current demand at position 81	11	-
81	both	R	Variable slip current demand	11	A
82	in	R	Nacelle roll acceleration	18	rad/s^2
83	in	R	Nacelle nodding acceleration	18	rad/s^2
84	in	R	Nacelle yaw acceleration	18	rad/s^2
85-89			Reserved		
90	in	R	Real time simulation time step		s
91	in	R	Real rime simulation time step multiplier		-
92	out	R	Mean wind speed increment	20	m/s
93	out	R	Turbulence intensity increment	20	%
94	out	R	Wind direction increment	20	rad
95-96			Reserved		
97	in	I	Safety system number that has been activated		-
98	out	I	Safety system number to activate		-
99	in	I	Reserved		
100	in	I	Reserved		
101	in	R	Reserved		
102	out	I	Yaw control flag	21	-
103	out	R	Yaw stiffness if record 102 = 1 or 3	21	-
104	out	R	Yaw damping if record 102 = 2 or 3	21	-
105	in	R	Reserved		
106	in	R	Reserved		
107	out	R	Brake torque demand	19, 22	Nm

....continued overleaf....

Record number	Data flow[8]	Data type[9]	Description	See note(s)	Units
108	out	R	Yaw brake torque demand		Nm
109	in	R	Shaft torque (= hub Mx for clockwise rotor)	18	Nm
110	in	R	Hub Fixed Fx	18	N
111	in	R	Hub Fixed Fy	18	N
112	in	R	Hub Fixed Fz	18	N
113	in	R	Network voltage disturbance factor		-
114	in	R	Network frequency disturbance factor		-
115-116			Reserved		
117	in	I	Controller state	23	-
118	in	R	Settling time (time to start writing output)		s
119			Reserved		
120-129	both	R	User-defined variables 1 to 10	24	
130-142			Reserved		
143	in	R	Teeter angle		rad
144	in	R	Teeter velocity		rad/s
145-160			Reserved		
161	in	I	Controller failure flag		-
162	in	R	Yaw bearing angular position		rad
163	in	R	Yaw bearing angular velocity		rad/s
164	in	R	Yaw bearing angular acceleration		rad/s^2
R	in	R	First generator speed in look-up table	3,5	rad/s
R+1	in	R	First generator torque in look-up table	3,5	Nm
R+2	in	R	Second generator speed in look-up table	3,5	rad/s
R+3	in	R	Second generator torque in look-up table	3,5	Nm
...		...	... etc., until ...		...
R+2N-2	in	R	Last generator speed in look-up table	3,5	rad/s
R+2N-1	in	R	Last generator torque in look-up table	3,5	Nm
M_0	out	I	Message length, only if record 49 < 0	15	
M_1 - M_n	out	C	Message text, 4 characters per record	15	-
L_1 onwards	out	R	Variables returned for logging output	17	SI

Notes:

1. Pitch regulated case only.
2. Not for variable speed pitch regulated case.
3. Variable speed case only.
4. Based on free wind at hub position - no modelling of actual nacelle anemometer or wind vane.
5. If the look-up table option is selected for the optimal mode below rated control, then record 16 is zero, record 25 contains the record number (R) of the start of the look-up table, and record 26 contains the number of points in the table (N).
6. DLL case only.
7. DLL case only.
8. in = data supplied by simulation, which may be used but not changed by the external controller.
   out = data supplied by the external controller to the simulation.
   both = data which is written by the simulation but which may be changed by the external controller.
9. Record type for EXE case. I = integer, R = real (floating point), C = character. In the DLL case, all records are actually passed as 4-byte real (floating point) numbers.
10. 0 = off, 1 = main (high speed) or variable speed generator, 2 = low speed generator.
11. Only used with the variable slip generator electrical model. Set record 80 to 1 if using record 81 to send a rotor current demand. If record 80 is 0 (default), then the torque demand (record 47) will be used to control the generator.
12. See record 28.
13. See record 29.
14. Depending on record 10.
15. EXE case only.
16. See Below.
17. DLL case only.
18. Record 79 is used to request additional measured loads and accelerations to be provided by the simulation:

Record 79	Blade loads and accelerations	Hub rotating loads	Hub fixed loads	Yaw bearing loads
0				
1	√			
2	√	√		
3	√	√	√	
4	√	√	√	√

19. For shaft brake 1; to apply additional brakes, this is a binary flag: specify a value of

$$\sum_i 2^{i-1} B_i$$

where B_i = 1 if the brake with index number i is applied, otherwise 0. The brake index numbers are as follows:

Index number	Brake description
1	Shaft brake 1
2	Shaft brake 2
3	Generator brake
4	Shaft brake 3
5	Brake torque set in record 107

20. For the Real Time Test facility, it is useful for the user to be able to change the wind conditions manually during a simulation from code in the external controller. Bladed will increase the mean wind speed, turbulence intensity (of all components) and wind direction by the value set in the respective field.
21. Yaw control flag in record 102 (affects the flexible yaw model only):
    0: Default (record 48 sets the yaw rate demand).
    1: As 0 but change the linear yaw stiffness according to record 103 (no effect on hydraulic accumulator model).
    2: As 0 but change the yaw damping according to record 104.
    3: As 1 but also change the yaw damping according to record 104.
    4: Use record 41 (yaw torque demand) to override the yaw spring and damper.
22. Brake torque demand used for brake index 5 (see note 19).
23. Controller state flag is set by the Bladed internal controller as follows:
    0: Power production
    1: Parked
    2: Idling
    3: Start-up
    4: Normal stop
    5: Emergency stop
24. May be used to share information between user-defined DLLs for different turbine components.

Table 1: Communication records

Note the strict use of SI units for all variables.

Note also that many of the parameters passed from the simulation to the controller are constants as defined in the **Control Systems** window, and some are variables such as measured signals. Some are only relevant for certain types of controllers, e.g. fixed or variable speed, stall or pitch control, and pitch position or pitch rate actuators. Although the record numbers are always the same, as shown in the tables above, the user-defined controller need only make use of those parameters which it actually requires, and only needs to output the demands which are relevant for the particular case, e.g.:

- demanded pitch angle(s) for pitch regulated machines with pitch position actuator
- demanded pitch rate(s) for pitch regulated machines with pitch rate actuator
- demanded generator torque for variable speed machines
- demanded nacelle yaw rate if the external controller option was selected for active yaw with yaw rate control
- demanded yaw actuator torque if yaw torque control was selected.

The controller may, if desired, change the status of the generator contactors and the brake.

Record 1: the Status flag

In the EXE case, record 1 of the shared binary file is used for handshaking.

In the DLL case, element 1 of the "DATA" array is set by the simulation as follows:

0	First call at time zero
1	All subsequent timesteps
-1	Final call at the end of the simulation.
2	Real Time update step (for Real Time Test simulations only). On a call with the status flag set to 2, the DLL must exchange data with the turbine controller.

The DLL may set the value to –1 to request the simulation to terminate.

Sending messages to the simulation

The controller may send a message to the simulation, which will then be displayed to the user.

In the DLL case, a separate argument to the DLL is provided for this purpose. Element 49 of the "DATA" array gives the maximum number of characters allowed. Each 1-byte element of the "MESSAGE" array can store one character of the message.

In the EXE case, there are two methods of specifying the message, which should not exceed 80 characters in length:

Method 1 (obsolete): Record 49 should contain the number of characters in the message, and the subsequent records should contain the message, four characters per record.

Method 2 (recommended): Place the message in records M_1 onwards, 4 characters per record. Enter the number of characters in the message as an integer in record number M_0 where $M_0 = M_1 - 1$, and set record 49 to $-M_0$ (note negative sign). Choose M_0 so that all these records occur after other output records, for example $M_0 = 61$. In practice it does not actually matter if any of the records in Table A.1 are overwritten since they are refreshed each timestep.

The EXE controller **must** write to record 49: a zero should be written if there is no message.

Pitch and torque override

If the external controller is used for supervisory control actions such as starts stops, while the built-in continous-time PI controllers are used for power production control, then it may be necessary for the external controller to specify the instant at which the supervisory control action takes over from the in-built controller action. Set record or element 55 to integer 0 whenever the external controller is to control the pitch, overriding the built-in PI controller. Set it to 1 when the built-in PI controller should be controlling the pitch.

For variable speed turbines, use record 56 in the same way to determine whether the external or the built-in controller should be controlling the generator torque.

Note that in the EXE case, messages should be written using Method 2 if the override control is to be used. The external controller will always take precedence if Method 1 is used.

A.5 Sending logging output to Bladed

In the DLL case only, additional data may be sent back to ***Bladed*** for logging in additional simulation output files in a similar format to other simulation outputs. This data can then be viewed directly using the Data View facility, or post-processed. This is particularly useful for debugging the controller, or for illustrating the details of its operation.

Element 62 of the "DATA" array gives the maximum number of logging outputs which can be returned. On the first call, the DLL should set element 65 to the number of logging outputs which will be returned, and their values should be returned starting at the element whose number is given by the value of element 63.

The "OUTNAME" array can be used to specify the names and units for the logging outputs. This should be set on at least the first and last calls to the DLL (overwriting the existing information in that array). This array should be set to a sequence of characters as follows:

Name:*Units*;

repeated for each logging output. *Name* is a description of the logging output, and *Units* should be one of table 2, provided the logging output is presented in strict SI units.

See below for an example in 'C'.

Allowed values for	Meaning (strict SI)
-	(No units specified)
1/T	s^{-1} (Hz)
A	rad
A/P	rad/W
A/PT	rad/Ws
A/PTT	rad/Ws^2
A/T	rad/s
A/TT	rad/s^2
F	N
F/L	N/m
F/LL	N/m^2
FL	Nm
FL/A	Nm/rad
FL/L	Nm/m
FLL	Nm^2
FLT/A	Nms/rad
FLTT/AA	Nms^2/rad^2
I	A
L	m
L/T	m/s
L/TT	m/s^2
LLL	m^3
LLL/A	m^3/rad
M	kg
M/L	kg/m
M/LLL	kg/m^3
M/LT	kg/ms
MLL	kgm^2
N	(No units specified)
P	W
PT	J
Q	VAr
T	s
V	V
VI	VA

Table 2: Allowed Units

Example External Controller Code In Selected Languages

To assist the user to get started with the coding required for external controllers, this section presents a few simple examples.

Simple example of DLL code written in C

```c
#include <stdio.h>
#include <string.h>
#define NINT(a) ((a) >= 0.0 ? (int)((a)+0.5) : (int)((a)-0.5))

extern "C"     //avoid mangled names
{ void __declspec(dllexport) __cdecl DISCON(float *avrSwap, int *aviFail,
char *accInfile, char *avcOutname, char *avcMsg);
}

//Main DLL routine
void __declspec(dllexport) __cdecl DISCON(float *avrSwap, int *aviFail,
        char *accInfile, char *avcOutname, char *avcMsg)
{
        char Message[257], InFile[257], OutName[1025];
        float rTime, rMeasuredSpeed, rMeasuredPitch;
        int iStatus, iFirstLog;
        static float rPitchDemand;

        //Take local copies of strings
        memcpy(InFile,accInfile, NINT(avrSwap[49]));
        InFile[NINT(avrSwap[49])+1] = '\0';
        memcpy(OutName,avcOutname, NINT(avrSwap[50]));
        OutName[NINT(avrSwap[50])+1] = '\0';

        //Set message to blank
        memset(Message,' ',257);

        //Set constants
        SetParams(avrSwap);

        //Load variables from Bladed (See Appendix A)
        iStatus = NINT (avrSwap[0]);
        rTime = avrSwap[1];
        rMeasuredPitch = avrSwap[3];
        rMeasuredSpeed = avrSwap[19];

        //Read any External Controller Parameters specified in the User Interface
        if (iStatus == 0)
        {
                *aviFail = ReadData(InFile, Message);  //User to supply this routine
                rPitchDemand = rMeasuredPitch;         //Initialise
        }

        //Set return values using previous demand if a sample delay is required
        avrSwap[44] = rPitchDemand;

        //Main calculation     //User to supply calcs routine
        if (iStatus >= 0 && *aviFail >= 0)
                *aviFail = calcs(iStatus, rMeasuredSpeed, rMeasuredPitch,
                        &rPitchDemand, OutName, Message);

        //Logging output - example
        avrSwap[64] = 2;              //No of outputs
        iFirstLog = NINT(avrSwap[62])-1;             //Address of first output
        strcpy(OutName, "Speed:A/T;Pitch:A");        //Names and units
        avrSwap[iFirstLog] = rMeasuredSpeed;         //First Value
        avrSwap[iFirstLog+1] = rMeasuredPitch;       //Second value

        //Return strings
        memcpy(avcOutname,OutName, NINT(avrSwap[63]));
        memcpy(avcMsg,Message,MIN(256,NINT(avrSwap[48])));

        return;
}
```

Simple example of DLL code written in FORTRAN 90

```
SUBROUTINE DISCON (avrSWAP, aviFAIL, accINFILE, avcOUTNAME, avcMSG)
IMPLICIT NONE

!Compiler specific: Tell the complier that this routine is the entry point for the DLL

!The next two lines are for the case of the Digital Visual Fortran compiler
CDEC$ ATTRIBUTES DLLEXPORT :: DISCON
CDEC$ ATTRIBUTES ALIAS:'DISCON' :: DISCON
!The Lahey LF90 compiler needs this line instead:
DLL_EXPORT DISCON
!For other compilers: read the documentation to find out how to do this

REAL AV_ avrSWAP(*)
INTEGER*1 accINFILE(*), avcOUTNAME(*), avcMSG(*)
INTEGER aviFAIL

INTEGER*1 iInFile(256), iOutName(1024), iMessage(256)
CHARACTER cInFile*256, cOutName*1024, cMessage*256
EQUIVALENCE (iInFile, cInFile), (iOutName, cOutName), (iMessage, cMessage)
INTEGER I, iStatus
REAL rTime, rMeasuredPitch, rMeasuredSpeed, rPitchDemand
SAVE rPitchDemand

!This just converts byte arrays to character strings, for convenience
DO I = 1,NINT(avrSWAP(50))
 iInFile(I) = accINFILE(I)    !Sets cInfile by EQUIVALENCE
ENDDO
DO I = 1,NINT(avrSWAP(51))
 iOutName(I) = avcOUTNAME(I)  !Sets cOutName by EQUIVALENCE
ENDDO

!Load variables from Bladed (See Appendix A)
iStatus = NINT(avrSwap(1))
rTime = avrSwap(2)
rMeasuredPitch = avrSwap(4)
rMeasuredSpeed = avrSwap(20)

!Read any External Controller Parameters specified in the User Interface
IF (iStatus .EQ. 0) THEN
  aviFail = ReadData(cInFile, cMessage)  !User to suppply this routine
  rPitchDemand = rMeasuredPitch          !Initialise
ENDIF

!Set return values using previous demand if a sample delay is required
avrSwap(45) = rPitchDemand

!Main calculation (User to suppply calcs routine)
IF (iStatus .GE. 0 .AND. aviFail .GE. 0) THEN
  aviFail = calcs(iStatus, rMeasuredSpeed, rMeasuredPitch, &
                  rPitchDemand, cOutName, cMessage)
ENDIF

!Return strings
DO I = 1,NINT(avrSwap(64))
 avcOutname(I) = iOutName(I)  !same as cOutName(I) by EQUIVALENCE
ENDDO
DO I = 1,MIN(256,NINT(avrSwap(49)))
 avcMsg(I) = iMessage(I)      !same as cMessage(I) by EQUIVALENCE
ENDDO

RETURN
END
```

Simple example of EXE code written in FORTRAN 90

```
IMPLICIT NONE
LOGICAL lOK
INTEGER iERROR, iUNIT, iFail, iSTATUS, iStarted
REAL rTime, rPitchDemand, rMeasuredPitch, rMeasuredSpeed

!First open the swap file
L_UNIT = 99
OPEN(L_UNIT, FILE='DISCON.SWP', ACCESS='DIRECT', FORM='UNFORMATTED', RECL=4, &
     ACTION='READWRITE,DENYNONE', IOSTAT=iERROR)
IF (iERROR.NE.0) STOP 'Could not open swap file'

!Set initialisation flag
iStarted = 0

!Write zero to record 1
WRITE(iUNIT, REC=1, IOSTAT=iERROR) 0
CLOSE(iUNIT)
IF (iERROR.NE.0) STOP 'Could not write to swap file'

!Wait for Bladed
lOK = .TRUE.
DO WHILE (lOK)
  OPEN(iUNIT, FILE='DISCON.SWP', ACCESS='DIRECT', FORM='UNFORMATTED', RECL=4, &
       ACTION='READWRITE,DENYNONE', IOSTAT=iERROR)
  IF (iERROR.NE.0) STOP 'Could not re-open swap file'
  READ(iUNIT, REC=1, IOSTAT=iERROR) iSTATUS
  IF (iERROR.NE.0) STOP 'Could not read status from swap file'
  IF (iSTATUS.EQ.-1) THEN
    !End of simulation
    lOK = .FALSE.
  ELSEIF (iSTATUS.EQ.0) THEN
    !Still waiting
    CALL SLEEPQQ(1) !Wait 1 millisecond; Compiler-dependent subroutine.It may be
                    !unneccessary, but may help to prevent problems on a slow network.
  ELSEIF (iSTATUS.EQ.1) THEN
    !Read from swap file
    READ(iUNIT, REC=2, IOSTAT=iERROR) rTime
    READ(iUNIT, REC=4, IOSTAT=iERROR) rMeasuredPitch
    READ(iUNIT, REC=20, IOSTAT=iERROR) rMeasuredSpeed

    IF (iStarted .EQ. 0) THEN
      iFail = ReadData('DISCON.IN')  !User to suppply this routine
      rPitchDemand = rMeasuredPitch  !Initialise
    ENDIF

    !Set return values using previous demand if a sample delay is required
    WRITE(iUNIT, REC=45, IOSTAT=iERROR) rPitchDemand

    !Main calculation (User to suppply calcs routine)
    IF (iStarted .GE. 0 .AND. iFail .GE. 0) THEN
      iFail = calcs(iStarted, rMeasuredSpeed, rMeasuredPitch, rPitchDemand)
    ENDIF

    iStarted = 1

  ELSE
    STOP 'Handshake status incorrect'
  ENDIF

  CLOSE(iUNIT)

ENDDO

STOP
END
```