Linking ADAS routines with Fortran, C and C++

A D Whiteford, M G O'Mullane and H P Summers

Generated on December 23, 2008

Contents

1 Introduction

This document gives information on linking ADAS subroutines with Fortran, C and C++. Typical flags for mainstream compilers are covered along with information on the directory structure inside ADAS. In particular, the topic of passing strings (typically filenames) and Fortran unit numbers between C and Fortran is covered.

This document should be read in conjunction with documentation on a specific subroutine, these are available as single documents for each subroutine or make up one large document (Appendix B of the ADAS manual). These documents give information on the input parameters of each subroutine.

The text in this document is an extract from end of Appendix B of the ADAS manual and is provided here in shorter form for easier browsing and printing.

Information in this document should not be taken as definitive and if in doubt about cross-linking subroutines you should consult your own compilers documentation.

2 Calling subroutines from Fortran

Since almost all ADAS subroutines are written in Fortran, calling them from Fortran tends to be much more straightforward than using C or C++. A typical compilation command to include the (e.g.) adas3xx library might be:

```
f77 myfile.f -o myfile.x -L/home/adas/lib -ladas3xx
```

the location of `/home/adas/lib` is installation/site specific and should be determined locally. It is frequently the case that the subroutine libraries rely on other libraries, particularly the generic `adaslib` library which underpins many of the series specific routines so it is often necessary to include more than one library. Examples of Fortran programs which use ADAS can be found in sections 4 and 5. Suggested compilation commands are given in the documentation of these programs.

3 Calling subroutines from C and C++

Calling ADAS subroutines from C is more problematic than Fortran. A particular problem is that a C compiler will not, by default, pass on underlying Fortran system (i.e. non-ADAS) libraries to the linker stage. The compilation options can be quite involved. This is an issue generic to linking Fortran & C and is not specific to ADAS. Guidance can be sought from either local system support or ADAS. It is also possible to compile a short dummy Fortran program and use the `--verbose` (or similar) compiler option to see what is passed to the linker.

However, for modern GNU/Linux systems with gcc, it is possible to compile C programs with the "g77" command, this has the side effect of also passing Fortran specific options to the linker, thus:

```
g77 myfile.c -L/home/adas/lib -ladaslib -o myfile.x
```

can often be effective (although note that gfortran does not have this useful side effect if no fortran codes are specified at the command line). A typical compilation command only using a C compiler (in this case gcc version 3) could look like:

```
gcc myfile.c -o myfile.x -L/home/adas/lib -ladaslib -lg2c
```

but other libraries in addition to "g2c" may be necessary. On some systems it may also be necessary to explicitly use version 3 of gcc (the last version that provided backward compatibility with g77), this is typically installed as the command 'gcc32', 'gcc34' etc. (note that 'gcc32' is not related to 32 bit vs 64 bit compilation). If the ADAS libraries have been compiled with gfortran then a typical compilation command would look like:

```
gcc myfile.c -o myfile.x -L/home/adas/lib -ladaslib -lgfortran
```

but, as above, other libraries may be necessary[1]

A great deal of confusion can be caused by the appending of trailing underscores and trailing double underscores on to Fortran routines. All ADAS library routines are compiled with the '-fno-second-underscore' flag set. This means that a subroutine called 'FOO' in Fortran becomes a C void function called 'foo_'. However, care must be taken when linking routines with each other. Error messages will be given at the linker stage

[1] ADAS is typically compiled using g77 (i.e. gcc version 3) but gfortran is used if g77 is not present on a given system. A request can be made for libraries compiled using a different compiler.

unless you are building shared libraries (as opposed to an executable program) in which case error messages are delayed until runtime.

3.1 Variable references, strings and Fortran I/O units

All Fortran subroutines take in references to variables, even to scalar variables. This is in contrast to C which passes everything by value (in this context, the 'value' of an array is defined as a pointer to the start of the array — the semantics of this statement are unimportant). This means that given a Fortran subroutine 'FOO' defined thus:

```
      SUBROUTINE FOO(A)
      INTEGER*4 A

      PRINT *,A

      END
```

which would have a C prototype of:

```
extern foo_(int *a);
```

it is **NOT** acceptable to write a C program thus:

```
extern foo_(int *a);
int main(void)
{
        foo_(15);
}
```

instead, one needs to write:

```
extern foo_(int *a);
int main(void)
{
        int x=15;
        foo_(&x);
}
```

this is so that Fortran receives a memory address to an integer (note that it's also possible to solve the issue in Fortran using things like %VAL — this is not discussed here since all ADAS Fortran routines are written conventionally and expect memory addresses).

Passing arrays is the same but the syntax is slightly different. Again, given a subroutine 'FOO':

```
      SUBROUTINE FOO(A)
      INTEGER*4 A(10)

      PRINT *,A(5)

      END
```

then the correct calling sequence would be:

```
extern foo_(int *a);
int main(void)
{
        int z[10];
        z[4]=15;
        foo_(z);
}
```

note that C counts from 0 and Fortran counts from 1 so that z[4] in C translates to A(5) in Fortran. Also note that our explicit memory addressing (i.e. with &) of the integer variable is no longer necessary since an array is already a memory address). For a 2D array our Fortran routine becomes:

```
        SUBROUTINE FOO(A)
        INTEGER*4 A(10,20)

        PRINT *,A(5,8)

        END
```

and our C routine is now:

```
extern foo_(int *a);

int main(void)
{
         int z[20][10];
         z[7][4]=15;
         foo_(*z);
}
```

note specifically that we now dereference 'z' once (by writing '*z') since we want to find the memory address of the first element. Also worthy of note is that the array indices swap between C and Fortran, i.e. [7][4] in C becomes (5,8) in Fortran. For a 3D array, as one might expect we end up with:

```
        SUBROUTINE FOO(A)
        INTEGER*4 A(10,20,30)

        PRINT *,A(5,8,19)

        END
```

and:

```
extern foo_(int *a);

int main(void)
{
         int z[30][20][10];
         z[18][7][4]=15;
         foo_(**z);
}
```

note the double dereferencing of 'z' (i.e. '**z'). Higher dimensioned arrays are as one would expect.

Passing strings from C to Fortran is problematic and highly compiler dependent. Fortran subroutines pass the length of strings to each other as hidden parameters; if working only in the Fortran domain then the programmer never needs to worry about these parameters but if calling a Fortran routine from C you need to pass these parameters. It is implementation specific where the lengths are passed but most compilers append them at the very end of the argument list; if there is more than one string being passed then the order of the lengths are the same as the order of the original strings. As an example, a Fortran routine which accepts variable length strings such as:

```
         subroutine foo(a,b,c,d)

         character*(*)   a
         integer*4       b
```

```
        character*(*)   c
        integer*4       d

        print *,a,b
        print *,c,d

        end
```

has a prototype of:

```
extern foo_(char *a, int *b, char *c, int *d, int len_a, int len_c);
```

and should be called thus:

```
extern foo_(char *a, int *b, char *c, int *d, int len_a, int len_c);

int main(void)
{
        char a[20];
        int b;
        char c[20];
        int d;

        strcpy(a,"Example String A");
        b=1;
        strcpy(c,"Example String C");
        d=2;

        foo_(a,&b,c,&d,strlen(a),strlen(c));
}
```

note that we don't say '&a' or '&c' since 'a' and 'c' are already a memory addresses. Note also that len_a and len_c are passed by value rather than by reference.

If the Fortran routine has fixed length strings such as:

```
        subroutine foo(a,b,c,d)

        character*40    a
        integer*4       b
        character*40    c
        integer*4       d

        print *,a,b
        print *,c,d

        end
```

then it is necessary to fill the strings with NULL characters (i.e. '\0') in C before passing them to the Fortran. Note that C by default only puts a NULL character at the end of the string text, but the rest of the memory is essentially random, which is insufficient for Fortran. A C routine would look like:

```
extern foo_(char *a, int *b, char *c, int *d, int len_a, int len_c);

int main(void)
{
```

```
        char a[41];
        int b;
        char c[41];
        int d;
        int i;

        for (i=0;i<=40;i++)
        {
                a[i]='\0';
                c[i]='\0';
        }

        strcpy(a,"Example String A");
        b=1;
        strcpy(c,"Example String C");
        d=2;

        foo_(a,&b,c,&d,40,40);

}
```

most Fortran compilers will typically ignore the lengths being passed from C if the lengths are explictly defined in the Fortran. Note also that a C string has to be one character longer than a Fortran string because C strings require a trailing '\0' when at fully capacity whereas Fortran strings have an explicit length so do not. It's possible to code the C such that this isn't necessary but then there is a chance that your strings will behave poorly with functions such as printf, strcpy etc. Note that it is often possible to 'get away' with not padding out the strings with NULLs, particularly when only using the strings as filenames. This is not recommended.

Things are slightly more problematic for string arrays, given a Fortran subroutine:

```
        subroutine foo(a,b)

        character*40    a(15)
        character*20    b(10)

        print *,a(1)
        print *,b(4)

        end
```

we would use a C program such as:

```
extern foo_(char *a, char *b,int len_a, int len_b);

int main(void)
{
        char a[15][40];
        char b[10][20];
        int i,j;

        for (i=0;i<15;i++)
        {
                for (j=0;j<40;j++)
                {
                        a[i][j]='\0';
                }
```

```
        }

        for (i=0;i<10;i++)
        {
                for (j=0;j<20;j++)
                {
                        b[i][j]='\0';
                }
        }

        strcpy(a[0],"Example String A");
        strcpy(b[3],"Example String B");

        foo_(*a,*b,40,20);
}
```

there are a number of things worthy of note. Firstly, a and b are now (in the C sense) 2D arrays so we need to pass in '*a' as we did before with 2D integer arrays. There are clearly more efficient (and less verbose) ways of clearing the memory contents of 'a' and 'b' but the code was written as above for reasons of clarity.

The most subtle issue is now the C strings have to have the same number of characters as the Fortran strings (previously we dimensioned our C strings with one extra character). Due to the assumptions that Fortran makes about memory it's no longer possible to have this extra padding character for the sake of C. The consequences of this are not severe: even though we have, say, 40 character strings in Fortran we should only every treat them with an effective length of 39. This means that in C they can still end with a '\0'. If we use the whole 40 characters in Fortran then the strings will not behave well when passed to C functions such as printf, strcpy etc..

Many ADAS routines, especially the `xxdata` series of routines take Fortran unit numbers as an input rather than filenames. They expect the file to be opened by the calling subroutine, this is present for a number of reasons but can pose a problem since a C filehandle is not the same as a Fortran unit number. To get around this we suggest calling the ADAS library routine XXOPEN to obtain open a file on a unit number. The routine can be called:

```
        long lexist;
        int iunit=10;
        char dsnin[80];
        strcpy(dsnin,"/home/adas/adas/path/to/datafile.dat");
        xxopen_ (&iunit, dsnin, &lexist, strlen(dsnin));
```

which will open the above file on Fortran unit 10, the unit variable can then be passed in to the required routine. An example of using XXOPEN in a practical application is given in section 7. Note also that the ADAS routine XXOPCS can open a file and convert it to all lower case or all upper case as it is read.

3.2 Header files

For ease of use, many ADAS Fortran subroutines have C-style prototypes generated and these can normally be found in `/home/adas/include` but this path is site specific. A header file for each ADAS series exists and are called, e.g. adas3xx.h for ADAS series 3 subroutines and adaslib.h for generic ADAS library subroutines. These can be included at the top of C programs and the path to the include files should be specified with the '-I' command line. For example, a complete program to open a file on unit 10 (albeit then do nothing with it) would look like:

```
#include <string.h>
#include "adaslib.h"

int main(void)
{
```

```
        long lexist;
        int iunit=10;
        char dsnin[80];
        strcpy(dsnin,"/home/adas/adas/path/to/datafile.dat");
        xxopen_ (&iunit, dsnin, &lexist, strlen(dsnin));
}
```

and would be compiled with one of:

```
g77 -I$ADASTOP/include small.c -L$ADASTOP/lib -ladaslib
gcc -I$ADASTOP/include small.c -L$ADASTOP/lib -ladaslib -lg2c
gcc -I$ADASTOP/include small.c -L$ADASTOP/lib -ladaslib -lgfortran
```

where note that in the first example we're using the g77 command to compile C. You must use the same compiler version as ADAS was compiled with — see section 3 for more details on compiler issues with specific reference to g77 versus gfortran and the associated issue of gcc version 3 versus version 4 and the occasional neccesity to use a command like 'gcc32'.

Section 6 gives an example C program which calculates bremmstrahlung (analogous to the Fortran example given in 4) and section 8 gives an example of using C to extract beam stopping coefficients from ADF21 files.

3.3 Specific C++ issues and usage

The main difference between calling an ADAS code from C++ as opposed to C is that a prototype which would have been:

```
extern void foo_(int *a, char *b, float *c );
```

in C must become:

```
extern "C" { void foo_(int *a, char *b, float *c ); };
```

in C++. All of the ADAS header files automatically detect what language they are being compiled into (via looking for the definition of '__cplusplus') and prototypes are defined accordingly.

All of the information on linking with C given in the previous section is still valid but note that the C++ string type needs to converted to a character array before going in to Fortran. Given a Fortran routine such as:

```
        subroutine foo(a,b,c,d)

        character*30    a
        real*4          b(10,20)
        character*40    c(10)
        integer         d

        print *,a
        print *,b(10,18)
        print *,c(3)
        print *,d

        end
```

and a simple C++ program such as:

```
using namespace std;
#include <string>

int main(void)
{
```

```
        string a;
        float  b[20][10];
        string c[10];
        int    d;

        a="Example string in scalar";
        b[17][9]=123.0;
        c[2]="Example string in array";
        d=3;
}
```

then in order to call the Fortran FOO routine the C++ would need to look like:

```
extern "C" { void foo_(char *a,float *b       ,char *c,
                       int  *d,int   strlen_a ,int strlen_c);
           };
using namespace std;
#include <string>

int main(void)
{
        string a;
        float  b[20][10];
        string c[10];
        int    d;

        a="Example string in scalar";
        b[17][9]=123.0;
        c[2]="Example string in array";
        d=3;

        char a_temp[30];
        char c_temp[10][40];
        int i,j;

        for (i=0;i<30;i++) a_temp[i]='\0';
        if (!a.empty())
        {
                strcpy(a_temp,a.c_str());
        }

        for (i=0;i<10;i++)
        {
                for (j=0;j<40;j++)
                {
                        c_temp[i][j]='\0';
                }
                if (!c[i].empty())
                {
                        strcpy(c_temp[i],c[i].c_str());
                }
        }

        foo_(a_temp,*b,*c_temp,&d,30,40);
```

```
}
```

where the example given above is coded quite verbosely. Note that this shows an example of passing a string array, scalar string, scalar inteer and 2D floating point array in one example. Readers are encouraged to read the previous section on calling routines from C for a discussion of each of these separate issues.

In the routine above the '`c_str()`' method the C++ string class has been used to return an input suitable for strcpy but that it is necessary before invoking this method to check that the string isn't empty.

Compilation issues also arise when using C++. For straightforward C and Fortran linking using GNU tools it was suggested to simply invoke a Fortran compiler to take care of extra linking. However, using a Fortran compiler does not link in extra C++ libraries (conversely, using a C++ compiler doesn't link in any Fortran libraries). Suggested generic commands are:

```
g++ -I/home/adas/include myfile.cpp -L/home/adas/lib -ladaslib -lgfortran
```

for gcc 4 compiler (i.e. a gfortran implementation) and:

```
g++ -I/home/adas/include myfile.cpp -L/home/adas/lib -ladaslib -lg2c
```

for a gcc 3 compiler (i.e. a g77 implementation). Note that on modern systems a backward compatible gcc version 3 C++ compiler will typically be a command like 'g++32', 'g++34' or similar (see section 3).

Note that the C code given in section 8 which calculates a beam stopping coefficient is also perfectly valid C++ code. However, also see the next section for details on using C++ in a more idiomatic (i.e. object-based) way.

3.4 Class wrappers for C++ usage

There are a limited[2] number of class wrappers available for ADAS subroutines. The headers are arranged on a per-subroutine basis instead of a by series basis. The header files have an extension of `.hpp`. The definitive list of subroutines available with C++ wrappers can be found by looking for `.hpp` files in the `/home/adas/include` directory.

Note that it's possible to call every ADAS subroutine from C++ (see the previous section) and we are only presently discussing class-based solutions to calling ADAS subroutines.

Documentation for the classes appears in the header files but the relevant Fortran documentation should also be consulted for a fuller description of the workings of the particular subroutine.

All of the C++ classes are inside the 'adas' namespace to avoid symbol clashes with any other routines you may have, this means that to declare an ADAS object one would have something like:

```
#include "cxbms.hpp"     // Include header file
adas::cxbms myobject;    // Declare object
```

Compilation against these classes requires linking in the 'adascpp' library so generic compiler commands when using class wrappers become:

```
g++ -I/home/adas/include myfile.cpp -o program.x  \
         -L/home/adas/lib -ladascpp -ladaslib -lgfortran
```

for gcc 4 compiler (i.e. a gfortran implementation) and:

```
g++ -I/home/adas/include myfile.cpp -o program.x \
         -L/home/adas/lib -ladascpp -ladaslib -lg2c
```

for a gcc 3 compiler (i.e. a g77 implementation). Note that on modern systems a backward compatible gcc version 3 C++ compiler will typically be a command like 'g++32', 'g++34' or similar (see section 3). Also, it may be necessary to link in other ADAS libraries depending on what the C++ class is wrapping.

An example of using a class interface to the same example of beam stopping as exemplified in section 8 is given in 9.

[2] These are added on a per-request basis.

4 Fortran example: Bremsstrahlung emission

```
       PROGRAM BREM
C-----------------------------------------------------------------------
C
C  ******************* FORTRAN77 PROGRAM: BREM ************************
C
C  PURPOSE:   TO EVALUATE BREMSSTRAHLUNG EMISSION USING ADAS AT A
C             SPECIFIED TEMPERATURE AND DENSITY. ORIGINALLY WRITTEN AS
C             AN EXAMPLE PROGRAM FOR RITU DEY AT THE IPR
C
C  ROUTINES:
C          ROUTINE    SOURCE    BRIEF DESCRIPTION
C          ------------------------------------------------------------
C          CONTINUO   ADAS      FOR AN GIVEN WAVELENGTH GENERATE
C                               RADIATIVE RECONBINATION AND
C                                BREMSSTRAHLUNG EMISSIVITY.
C
C  NOTES :    CONTAINS VERBOSE COMMENTS TO ILLUSTRATE HOW ADAS
C             MAY BE USED FROM STAND ALONE FORTRAN PROGRAMS.
C
C             THIS PROGRAM SHOULD BE LINKED TO BOTH THE ADAS3XX
C             AND ADASLIB LIBRARIES, COMPILATION WILL RESEMBLE:
C               f77 brem.for -L/home/adas/lib \
C                       -ladas3xx -ladaslib -o brem.x
C
C
C  AUTHOR:    ALLAN WHITEFORD
C             UNIVERSITY OF STRATHCLYDE
C
C  DATE:      30/03/06
C-----------------------------------------------------------------------

C-----------------------------------------------------------------------
C      DEFINE APPOPRIATE VARIABLES
C-----------------------------------------------------------------------
       IMPLICIT  NONE
C-----------------------------------------------------------------------
       INTEGER*4 IZ0  , IZ1
       REAL*8    WAVE , TEV , CONTFF , CONTIN
C-----------------------------------------------------------------------

C-----------------------------------------------------------------------
C      SPECIFY WE'RE DEALING WITH FULLY STRIPPED CARBON
C-----------------------------------------------------------------------
       IZ0=6
       IZ1=7

C-----------------------------------------------------------------------
C      SPECIFY A TEMPERATURE OF 3keV and a WAVELENGTH OF 6000A
C-----------------------------------------------------------------------
       TEV=3000
       WAVE=6000

C-----------------------------------------------------------------------
C      USE ADAS TO CALCULATE EMISSION
C-----------------------------------------------------------------------
       CALL CONTINUO(WAVE   , TEV    , IZ0  , IZ1  ,
     &               CONTFF , CONTIN
     &                                          )


C-----------------------------------------------------------------------
C      PRINT OFF FREE-FREE EMISSIVITY (Ph cm3 s-1 A-1)
C-----------------------------------------------------------------------
       PRINT *,CONTFF

C-----------------------------------------------------------------------
C      PRINT OFF TOTAL EMISSIVITY (Ph cm3 s-1 A-1)
C-----------------------------------------------------------------------
       PRINT *,CONTIN

       END
```

5 Fortran example: Ionisation balance for carbon

```
       PROGRAM CARBON
C-----------------------------------------------------------------------
C
C  ****************** FORTRAN77 PROGRAM: CARBON ************************
C
C  PURPOSE:   TO EVALUATE EQUILIBRIUM IONISATION BALANCE FOR CARBON AT
C             A RANGE OF FIXED TEMPERATURE AND DENSITY PAIRS. WRITTEN
C             AS AN EXAMPLE PROGRAM FOR PAVEL GONCHAROV AT NIFS/LHD AS
C             A DEMONSTRATION OF HOW TO CALL ADAS LIBRARIES FROM A
C             FORTRAN PROGRAM.
C
C  ROUTINES:
C          ROUTINE    SOURCE    BRIEF DESCRIPTION
C          ------------------------------------------------------------
C          D5DATA     ADAS      FETCHES DATA FROM ADF11 MASTER FILES
C          D5MPOP     ADAS      PARTITIONED TRI-DIAG MATRIX INVERSION
C
C  NOTES :    CONTAINS VERBOSE COMMENTS TO ILLUSTRATE HOW ADAS
C             MAY BE USED FROM STAND ALONE FORTRAN PROGRAMS.
C
C             THIS PROGRAM SHOULD BE LINKED TO BOTH THE ADAS4XX
C             AND ADASLIB LIBRARIES, COMPILATION WILL RESEMBLE:
C               f77 carbon.for -L/home/adas/lib \
C                       -ladas4xx -ladaslib -o carbon.x
C
C
C  AUTHOR:    ALLAN WHITEFORD
C             UNIVERSITY OF STRATHCLYDE
C
C  DATE:      01/02/07
C-----------------------------------------------------------------------

C-----------------------------------------------------------------------
C      DEFINE APPOPRIATE VARIABLES
C-----------------------------------------------------------------------
       IMPLICIT  NONE
C-----------------------------------------------------------------------
       INTEGER   NTDIM   , ITDIMD , IZDIMD , IMDIMD ,
     &           NDFILE  , IPDIMD , ISDIMD , NDONE
C-----------------------------------------------------------------------
       PARAMETER ( NTDIM  = 30 , ITDIMD = 51 , IZDIMD = 83 )
       PARAMETER ( IPDIMD = 5  , ISDIMD = 83 , IMDIMD = ISDIMD+1 )
       PARAMETER ( NDONE = 1   , NDFILE = 6  )
C-----------------------------------------------------------------------
       LOGICAL LPART , LSOLVE
C-----------------------------------------------------------------------
       LOGICAL LSELA(8)  , LEXSA(8)   , LDEFA(8)
       LOGICAL LACDA(IZDIMD,IPDIMD,IPDIMD),
     &         LSCDA(IZDIMD,IPDIMD,IPDIMD),
     &         LCCDA(IZDIMD,IPDIMD,IPDIMD),
     &         LQCDA(IZDIMD,IPDIMD,IPDIMD),
     &         LXCDA(IZDIMD,IPDIMD,IPDIMD),
     &         LPRBA(IZDIMD,IPDIMD),
     &         LPRCA(IZDIMD,IPDIMD),
     &         LPLTA(IZDIMD,IPDIMD)
C-----------------------------------------------------------------------
       CHARACTER DSFLLA(8)*120
C-----------------------------------------------------------------------
       INTEGER   IZ0
       INTEGER   I
       INTEGER   ITMAX
       INTEGER   IZL     , IZH     ,
     &           IT      ,
     &           NSTAGE  , NMSUM
C-----------------------------------------------------------------------
       INTEGER   NPRT(IZDIMD)  ,  NPRTR(IZDIMD)
C-----------------------------------------------------------------------
       REAL*8    DNS        , DNSH
C-----------------------------------------------------------------------
       REAL*8    TEV(NTDIM) , DTEV(NTDIM) ,
     &           DENS(NTDIM), DDENS(NTDIM), DENSH(NTDIM),
     &           FPABUN(NTDIM,IMDIMD)
       REAL*8    DTEVD(ITDIMD) , DDENSD(ITDIMD) , ZDATA(ISDIMD),
     &           DRCOFD(ISDIMD,ITDIMD,ITDIMD),
```

```
     &            DRCOFI(NTDIM)
       REAL*8     ACDA(NTDIM,IZDIMD,IPDIMD,IPDIMD),
     &            SCDA(NTDIM,IZDIMD,IPDIMD,IPDIMD),
     &            CCDA(NTDIM,IZDIMD,IPDIMD,IPDIMD),
     &            QCDA(NTDIM,IZDIMD,IPDIMD,IPDIMD),
     &            XCDA(NTDIM,IZDIMD,IPDIMD,IPDIMD),
     &            PRBA(NTDIM,IZDIMD,IPDIMD),
     &            PRCA(NTDIM,IZDIMD,IPDIMD),
     &            PLTA(NTDIM,IZDIMD,IPDIMD)
       REAL*8     POPF(IMDIMD)
       REAL*8     CFREC(IPDIMD,IPDIMD,IZDIMD),
     &            CFION(IPDIMD,IPDIMD,IZDIMD),
     &            CFMET(IPDIMD,IPDIMD,IZDIMD)
       REAL*8     CPOPN(IPDIMD,IPDIMD,IZDIMD+1) ,
     &            CPOPND(IPDIMD,IPDIMD,IZDIMD+1),
     &            CPOPNZ(IPDIMD,IPDIMD,IZDIMD+1)
       REAL*8     POPNMO(IPDIMD,NDONE,IZDIMD+1),
     &            POPNPO(IPDIMD,NDONE,IZDIMD+1),
     &            POPN(IPDIMD,NDONE,IZDIMD+1)
       REAL*8     RDUM(IPDIMD),RHS(2*IPDIMD-1),
     &            SOLVE(2*IPDIMD-1,2*IPDIMD-1)
       REAL*8     XTEMP(IPDIMD,IPDIMD),YTEMP(IPDIMD,IPDIMD)
       REAL*8     YTEM(IZDIMD)

C-----------------------------------------------------------------------
C      SET LOCATION OF RECOMBINATION (ACD) AND IONISATION (SCD) FILES
C-----------------------------------------------------------------------
       DSFLLA(1)='/home/adas/adas/adf11/acd96/acd96_c.dat'
       DSFLLA(2)='/home/adas/adf11/scd96/scd96_c.dat'
       LSELA(1)=.TRUE.
       LSELA(2)=.TRUE.
       LEXSA(1)=.TRUE.
       LEXSA(2)=.TRUE.

C-----------------------------------------------------------------------
C      SPECIFY THAT FILES ARE NOT METASTABLE
C-----------------------------------------------------------------------
       LPART=.FALSE.
       NSTAGE=7
       NMSUM=7
       NPRT(1)=1
       NPRT(2)=1
       NPRT(3)=1
       NPRT(4)=1
       NPRT(5)=1
       NPRT(6)=1
       NPRT(7)=1

C-----------------------------------------------------------------------
C      INDICATE THAT FILES CONTAIN COMPLETE INFORMATION
C-----------------------------------------------------------------------
       IZ0=6
       IZL=1
       IZH=7

C-----------------------------------------------------------------------
C      SET 13 TEMPERATURE AND DENSITY PAIRS TO COMPUTE IONISATION
C      BALANCE AT. IN THIS EXAMPLE WE COVER A WIDE RANGE AND ASSUME
C      CONSTANT DENSITY. HERE YOU COULD USE THE DATA FROM THOMSON
C      SCATTERING MEASUREMENTS OR SIMILAR
C-----------------------------------------------------------------------
       ITMAX=13

       DENS(1)=1d14
       DENS(2)=1d14
       DENS(3)=1d14
       DENS(4)=1d14
       DENS(5)=1d14
       DENS(6)=1d14
       DENS(7)=1d14
       DENS(8)=1d14
       DENS(9)=1d14
       DENS(10)=1d14
       DENS(11)=1d14
       DENS(12)=1d14
       DENS(13)=1d14
```

```
       TEV(1)= .1d0
       TEV(2)= .2d0
       TEV(3)= .5d0
       TEV(4)= 1d0
       TEV(5)= 2d0
       TEV(6)= 5d0
       TEV(7)= 10d0
       TEV(8)= 20d0
       TEV(9)= 50d0
       TEV(10)= 100d0
       TEV(11)= 200d0
       TEV(12)= 500d0
       TEV(13)= 1000d0

C-----------------------------------------------------------------------
C      SUBROUTINE ALSO REQUIRE THAT LOG10 OF THE ABOVE ARE PASSED
C-----------------------------------------------------------------------
       DO 10 I=1,ITMAX
10             DDENS(I)= DLOG10(DENS(I))
       DO 11 I=1,ITMAX
11             DTEV(I)= DLOG10(TEV(I))

C-----------------------------------------------------------------------
C      CALL ADAS ROUTINE TO READ THE ACD AND SCD FILES, THIS ROUTINE
C      ALSO INTERPOLATES ON TO THE DESIRED TEMPERATURE AND DENSITIES
C-----------------------------------------------------------------------
       CALL D5DATA( DSFLLA , LSELA  , LEXSA  , LDEFA  , LPART  ,
     &              IZ0    , IZL    , IZH    , NPRT   ,
     &              NTDIM  , ITMAX  ,
     &              ISDIMD , IZDIMD , ITDIMD , IPDIMD , NPRTR  ,
     &              DTEV   , DDENS  ,
     &              DTEVD  , DDENSD , DRCOFD , ZDATA  , DRCOFI ,
     &              ACDA   , LACDA  ,
     &              SCDA   , LSCDA  ,
     &              CCDA   , LCCDA  ,
     &              PRBA   , LPRBA  ,
     &              PRCA   , LPRCA  ,
     &              QCDA   , LQCDA  ,
     &              XCDA   , LXCDA  ,
     &              PLTA   , LPLTA  )


C-----------------------------------------------------------------------
C      LOOP OVER TEMERATURE AND DENSITY PAIRS
C-----------------------------------------------------------------------
       DO 12 IT=1,ITMAX
           DNS  = DENS(IT)
           DNSH = DENSH(IT)

C-----------------------------------------------------------------------
C          CALL ADAS ROUTINE TO CALCULATE THE FRACTIONAL ABUNDANCIES
C          OF THE DIFFERENT STAGES
C-----------------------------------------------------------------------

           CALL D5MPOP( NTDIM , IZDIMD, IPDIMD,
     &                  NSTAGE, ITMAX , NPRT  , NMSUM ,
     &                  ACDA  , SCDA  , CCDA  , QCDA  , XCDA  ,
     &                  DENS  , DENSH ,
     &                  IT    ,
     &                  CFREC , CFION , CFMET ,
     &                  POPN  , POPNMO, POPNPO,
     &                  CPOPN , CPOPND, CPOPNZ,
     &                  POPF  ,
     &                  XTEMP , YTEMP , YTEM  ,
     &                  RHS   , RDUM  , SOLVE , LSOLVE        )

C-----------------------------------------------------------------------
C          STORE THE RESULTS IN FPABUN, SET MINIMUM TO 1D-74 TO PROTECT
C          FOR UNDERFLOWS LATER IN THE PROGRAM
C-----------------------------------------------------------------------
           DO 13 I=1,NMSUM
   13          FPABUN(IT,I) = MAX(POPF(I),1D-74)

C-----------------------------------------------------------------------
C      END LOOP OVER TEMERATURE AND DENSITY PAIRS
```

```
C-----------------------------------------------------------------------
  12   CONTINUE

C-----------------------------------------------------------------------
C      WRITE THE OUTPUT TO THE SCREEN, IN A REAL EXAMPLE OF COURSE
C      YOU WOULD THEN USE THE CALCULATED DATA
C-----------------------------------------------------------------------
       WRITE(6,1000) (I-1,I=1,7)
       DO 14 IT=1,ITMAX
  14       WRITE(6,1001) TEV(IT),(FPABUN(IT,I),I=1,7)

1000   FORMAT('#',2X,'T / eV',5X,7('C+',I1,7X))
1001   FORMAT(8(1PE10.2))

C-----------------------------------------------------------------------
C      END OF EXAMPLE
C-----------------------------------------------------------------------
       END
```

6 C example: Bremsstrahlung emission

```
/*
 *      ********************* C Program: brem *********************
 *
 * Purpose:  To evaluate bremsstrahlung emission using adas at a
 *           specified temperature and density. originally written as
 *           an example program (in Fortran) for Ritu Dey at the IPR.
 *
 * ROUTINES:
 *         ROUTINE    SOURCE    BRIEF DESCRIPTION
 *         ------------------------------------------------------------
 *         continuo   ADAS      For a given wavelength generate
 *                              radiative reconbination and
 *                              bremsstrahlung emissivity.
 *
 * NOTES :    Contains verbose comments to illustrate how ADAS
 *            may be used from stand alone C programs.
 *
 *            This program should be linked to both the adas3xx
 *            and adaslib libraries, compilation will resemble:
 *              g77 -I/home/adas/include brem.c -L/home/adas/lib \
 *                       -ladas3xx -ladaslib -o brem.x
 *            (Note that g77 is used for compilation even though this
 *             program is written in C)
 *
 * AUTHOR:    Allan Whiteford
 *            University of Strathclyde
 *
 * DATE:      23/03/07
 *
 */

/*****************************************************************************
 Bremsstrahlung routine is in adaslib library so include appropriate header
*****************************************************************************/
#include "adaslib.h"

int main(int argc, char *argv)
{

/*****************************************************************************
        Declare appopriate variables
*****************************************************************************/
        int iz0, iz1;
        double wave,tev,contff,contin;

/*****************************************************************************
        Specify we're dealing with fully stripped carbon
*****************************************************************************/
        iz0=6;
        iz1=7;

/*****************************************************************************
        Specify a temperature of 3keV and a wavelength of 6000A
*****************************************************************************/
        wave=6000;
        tev=3000;

/*****************************************************************************
        Use ADAS to calculate emission
*****************************************************************************/
        continuo_(&wave   , &tev    , &iz0  , &iz1  ,&contff , &contin);

/*****************************************************************************
        Print off free-free emissivity (Ph cm3 s-1 A-1)
*****************************************************************************/
        printf("%e\n",contff);

/*****************************************************************************
        Print off total emissivity (Ph cm3 s-1 A-1)
*****************************************************************************/
        printf("%e\n",contin);

        return 0;
}
```

7 C example: Reading an ADF21 file

```
/*
 *      ********************* C Program: adf21 *******************
 *
 * Purpose:  To read an ADF21 file using ADAS. Originally written as
 *           an example program for Masaki Osakabe at LHD/NIFS.
 *
 * ROUTINES:
 *         ROUTINE    SOURCE    BRIEF DESCRIPTION
 *         ------------------------------------------------------------
 *         xxopen     ADAS      Opens a file on a Fortran unit number
 *         xxdata_21  ADAS      Reads an ADF21 file
 *
 * NOTES :    Contains verbose comments to illustrate how ADAS
 *            may be used from stand alone C programs.
 *
 *            This program should be linked to the adaslib libraries,
 *            compilation will resemble:
 *              g77 -I/home/adas/include adf21.c -L/home/adas/lib \
 *                    -ladaslib -o adf21.x
 *            (Note that g77 is used for compilation even though this
 *             program is written in C)
 *
 * AUTHOR:    Allan Whiteford
 *            University of Strathclyde
 *
 * DATE:      29/02/08
 *
 */

/*****************************************************************************
 xxdata_21 routine is in adaslib library so include appropriate header
*****************************************************************************/
#include "adaslib.h"

int main(void)
{
/*****************************************************************************
        Declare appopriate variables
*****************************************************************************/
        int    iunit=10 , mxbe=30     , mxtd=30     , mxtt=30;
        int    itz      , nbe         , ntdens      , nttemp;
        double svref    , beref       , tdref       , ttref;
        double be[mxbe] , tdens[mxtd] , ttemp[mxtt] , svt[mxtt];
        double sved[mxtd][mxbe];
        long lexist;
        char dsnin[81];
        char tsym[3]="  ";

/*****************************************************************************
        Set the filename appropriately
*****************************************************************************/
        strcpy(dsnin,"/home/adas/adas/adf21/bms97#h/bms97#h_c6.dat");

/*****************************************************************************
        Open the filename on a Fortran unit
*****************************************************************************/
        xxopen_ (&iunit, dsnin, &lexist, strlen(dsnin));

/*****************************************************************************
        Read the contents of the file
*****************************************************************************/
        xxdata_21_(     &iunit  , &mxbe   , &mxtd   , &mxtt   ,
                        &itz    ,  tsym   , &beref  , &tdref  ,
                        &ttref  , &svref  , &nbe    ,  be     ,
                        &ntdens ,  tdens  , &nttemp ,  ttemp  ,
                         svt    , *sved   ,  dsnin  ,  2 , 80 );

/*****************************************************************************
        Print off some example data from the file
*****************************************************************************/
        printf("%s %d %e %f %f\n",tsym,nttemp,svref,ttemp[0],ttemp[19]);

        return 0;
}
```

8 C example: Calculating a beam stopping coefficient

```
/*
 *      ********************** C Program: bms *********************
 *
 * Purpose:  To calculate beam effective stopping coefficients using
 *           ADAS. Originally written as an example program for
 *           Masaki Osakabe at LHD/NIFS.
 *
 *           The program calculates beam stopping coefficients at three
 *           temperatures (1keV, 2keV and 6keV) for a fixed beam energy
 *           of 165 keV / amu and a fixed plasma density of 2e13 cm-3.
 *
 *           The plasma is assumed to be 90% hydrogen and 10% carbon
 *           for all three temperatures but this can be easily changed.
 *
 * ROUTINES:
 *         ROUTINE    SOURCE    BRIEF DESCRIPTION
 *         ------------------------------------------------------------
 *         cxbms      ADAS      Calculates BMS coefficients
 *
 * NOTES :    Contains verbose comments to illustrate how ADAS
 *            may be used from stand alone C programs.
 *
 *            This program should be linked to the adas3xx and
 *            adaslib libraries, compilation will resemble:
 *              g77 -I/home/adas/include bms.c -L/home/adas/lib \
 *                     -ladas3xx -ladaslib -o adf21.x
 *            (Note that g77 is used for compilation even though this
 *             program is written in C)
 *
 * AUTHOR:    Allan Whiteford
 *            University of Strathclyde
 *
 * DATE:      29/02/08
 *
 */

/*****************************************************************************
 xxdata_21 routine is in adas3xx library so include appropriate header
*****************************************************************************/
#include "adas3xx.h"

/*****************************************************************************
 Additional required header files
*****************************************************************************/
#include <stdio.h>
#include <string.h>


int main(void)
{
/*****************************************************************************
        Declare appopriate variables
*****************************************************************************/
        int    mximp=5        , mxreqs=40      , iounit=10;
        int    nreq           , nsityp;
        double ubmeng[mxreqs] , utdens[mxreqs] , uttemp[mxreqs];
        char   dsnin[mximp][132];
        double sifrac[mximp][mxreqs];
        double bstot[mxreqs];

        int i;                  /* Variable for local use */

/*****************************************************************************
        Clear memory used for filename variable
*****************************************************************************/
        for (i=0;i<=nsityp*132;i++) (*dsnin)[i]='\0';

/*****************************************************************************
        Specify that we want 2 plasma species and 3 BMS coefficients
        calculated
*****************************************************************************/
        nsityp=2;
        nreq=3;
```

```
/*****************************************************************************
        Set filenames below for ADAS adf21 files, one for a H-beam
        into the hydrogen plasma species on to a carbon impurity
*****************************************************************************/
        strcpy(dsnin[0],"/home/adas/adas/adf21/bms97#h/bms97#h_h1.dat");
        strcpy(dsnin[1],"/home/adas/adas/adf21/bms97#h/bms97#h_c6.dat");

/*****************************************************************************
        Keep beam energy constant at 165 keV / amu for each of the
        calculated BMS coefficients
*****************************************************************************/
        ubmeng[0]=165e3;
        ubmeng[1]=165e3;
        ubmeng[2]=165e3;

/*****************************************************************************
        Keep plasma density constant at 2e13 cm-3 for each of the
        calculated BMS coefficients
*****************************************************************************/
        utdens[0]=2e13;
        utdens[1]=2e13;
        utdens[2]=2e13;

/*****************************************************************************
        Vary temperature from 1keV to 6keV for each of
        the values we get back
*****************************************************************************/
        uttemp[0]=1000e0;       /* Vary temperature from     */
        uttemp[1]=2000e0;       /* 1keV to 6keV for each of  */
        uttemp[2]=6000e0;       /* the values we get back    */

/*****************************************************************************
        90% hydrogen for each of the values we get back
*****************************************************************************/
        sifrac[0][0]=0.9;
        sifrac[0][1]=0.9;
        sifrac[0][2]=0.9;

/*****************************************************************************
        10% carbon for each of the values we get back
*****************************************************************************/
        sifrac[1][0]=0.1;
        sifrac[1][1]=0.1;
        sifrac[1][2]=0.1;

/*****************************************************************************
        Call ADAS routine to determine beam stopping
*****************************************************************************/
        cxbms_(  *dsnin,  &nsityp, &iounit, *sifrac, ubmeng, utdens,
                  uttemp, &nreq,   &mxreqs,  bstot,  132 );

/*****************************************************************************
        Print out temperature and beam stopping coefficient
*****************************************************************************/
        for (i=0;i<=2;i++)
        {
                printf("%f %e\n",uttemp[i],bstot[i]);
        }

        return 0;
}
```

9 C++ example: Calculating a beam stopping coefficient

```
/*
 *      ********************* C++ Program: bms ********************
 *
 * Purpose:  To calculate effective beam stopping coefficients
 *           using ADAS. Originally written as an example program
 *           for Masaki Osakabe at LHD/NIFS.
 *
 *           The program calculates beam stopping coefficients at three
 *           temperatures (1keV, 2keV and 6keV) for a fixed beam energy
 *           of 165 keV / amu and a fixed plasma density of 2e13 cm-3.
 *
 *           The plasma is assumed to be 90% hydrogen and 10% carbon
 *           for all three temperatures but this can be easily changed.
 *
 * REQUIRED CLASSES:
 *         CLASS      SOURCE    BRIEF DESCRIPTION
 *         ------------------------------------------------------------
 *         cxbms      ADAS      Calculates BMS coefficients
 *
 * NOTES :    Contains verbose comments to illustrate how ADAS
 *            may be used from stand alone C++ programs.
 *
 *            This program should be linked to the adas3xx and
 *            adaslib libraries, compilation will resemble:
 *              g++ -I/home/adas/include bms.cpp -L/home/adas/lib \
 *                      -ladascpp -ladas3xx -ladaslib -lg2c -o adf21.x
 *            (Note that it is necessary to link in the 'g2c' library
 *             but this may need to be gfortran depending on local settings)
 *
 * AUTHOR:    Allan Whiteford
 *            University of Strathclyde
 *
 * DATE:      29/02/08
 *
 */

/*************************************************************************
 Include the C++ header file which defines the cxbms class.
*************************************************************************/
#include "cxbms.hpp"

int main(void)
{
/*************************************************************************
        Declare 'mybms' as an adas::cxbms object.
*************************************************************************/
        adas::cxbms mybms;

/*************************************************************************
        Specify that we want 2 plasma species and 3 BMS coefficients
        calculated
*************************************************************************/
        mybms.set_nreq(3);
        mybms.set_nsityp(2);

/*************************************************************************
        Set filenames below for ADAS adf21 files, one for a H-beam
        into the hydrogen plasma species on to a carbon impurity
*************************************************************************/
        mybms.set_file(0,"/home/adas/adas/adf21/bms97#h/bms97#h_h1.dat");
        mybms.set_file(1,"/home/adas/adas/adf21/bms97#h/bms97#h_c6.dat");

/*************************************************************************
        Keep beam energy constant at 165 keV / amu for each of the
        calculated BMS coefficients
*************************************************************************/
        mybms.set_bmeng(0,165e3);
        mybms.set_bmeng(1,165e3);
        mybms.set_bmeng(2,165e3);
```

```
/*************************************************************************
        Keep plasma density constant at 2e13 cm-3 for each of the
        calculated BMS coefficients
*************************************************************************/
        mybms.set_dens(0,2e13);
        mybms.set_dens(1,2e13);
        mybms.set_dens(2,2e13);

/*************************************************************************
        Vary temperature from 1keV to 6keV for each of
        the values we get back
*************************************************************************/
        mybms.set_temp(0,1000);
        mybms.set_temp(1,2000);
        mybms.set_temp(2,6000);

/*************************************************************************
        90% hydrogen for each of the values we get back
*************************************************************************/
        mybms.set_frac(0,0,0.9);
        mybms.set_frac(0,1,0.9);
        mybms.set_frac(0,2,0.9);

/*************************************************************************
        10% carbon for each of the values we get back
*************************************************************************/
        mybms.set_frac(1,0,0.1);
        mybms.set_frac(1,1,0.1);
        mybms.set_frac(1,2,0.1);

/*************************************************************************
        Call ADAS routine to determine beam stopping
*************************************************************************/
        mybms.calculate();

/*************************************************************************
        Print out temperature and beam stopping coefficient
*************************************************************************/
        for (int i=0;i<=2;i++)
        {
                printf("%f %e\n",mybms.get_temp(i),mybms.get_bstot(i));
        }

        return 0;
}
```