```
/* These declarations have been automatically
   generated by the cif file input/output function
   generator.  This file should be included in any
   routines that call these functions */
typedef char cifstring[84];
           /* to avoid array complications later */
#define MYLABELMAX 50
#define ATSITEUMAX  1000
typedef double atsiteutype [6];
#ifdef CIFVARDEC
  cifstring errormes;         /* an error message */
  int       errornum;         /* an error number  */
  cifstring mylabel[50];
                     /*data_atom_site_aniso_label*/
  atsiteutype atsiteu[1000];
                        /*data_atom_site_aniso_U_*/
  atsiteutype atsiteuesd[1000];
  cifstring extmet;
               /*data_refine_ls_extinction_method*/
  double reftot [2];         /*data_reflns_number_*/
  double reftotesd [2];
#else
  extern cifstring errormes;  /* an error message */
  extern int       errornum;  /* an error number  */
  extern cifstring mylabel[50];
                     /*data_atom_site_aniso_label*/
  extern atsiteutype atsiteu[1000];
                        /*data_atom_site_aniso_U_*/
  extern atsiteutype atsiteuesd[1000];
  extern cifstring extmet;
               /*data_refine_ls_extinction_method*/
  extern double reftot [2];  /*data_reflns_number_*/
  extern double reftotesd [2];
#endif
```

Fig. 5.3.7.5. Header file cifvars.h for a C application built by *BuildSiv* from the augmented DDL dictionary of Fig. 5.3.7.4.

```
/* A simple example application of the automatically
   generated cifsiv_ function */

#include <stdio.h>
#include "cifvars.h"

main(int argc, char *argv[])
{
  int i;
  char filename[80];
  char block[80];
  printf("Please enter CIF file name: ");
  scanf("%s", filename);
  printf("Please enter data block name ");
  printf("(without data_ prepended): ");
  scanf("%s", block);
  errornum = 0;
  cifsiv_(filename,block);
  if(errornum != 0)  /* an error, we have problems */
    {
    printf("An error occurred in reading the
CIF:\n");
    printf("%s",errormes);
    }
  for(i=0;i<5;i++)
  {
    printf("Atom %d: %s %f %f\n", i, mylabel[i],
            atsiteu[i][0], atsiteu[i][1]);
  }
  printf("Total reflections: %.f\n", reftot[0]);
  printf("Extinction method: %s\n", extmet);
}
```

Fig. 5.3.7.6. An example C program designed to read CIF data as tagged in the augmented DDL dictionary of Fig. 5.3.7.4.

5.3.7.2.3. *Input to a C application program*

When a DDL dictionary *dictfile* has been edited in accordance with the description above, the program *BuildSiv* may be run under a Unix-like operating system with a command of the form

`BuildSiv` *`dictfile ddlversion`*

where *ddlversion* takes the values '1' or '2' to indicate that a DDL1 or DDL2 parser is appropriate. If the option '*-e*' is given before *dictfile*, variable definitions and read capability for standard uncertainty values will be included as well. The name of the variable that will hold the standard uncertainty is the name given by the programmer with the string `esd` appended.

An object file cifsiv.o is produced together with a header file cifvars.h. Some source-code files are also produced as intermediate files in the lexical analysis and parse phases of the software build; these may be deleted. The object file must be linked against the other object files when the application program is compiled and references to the header files must be introduced (generally through C preprocessor `#include` directives) within the application code where access to the imported data structures is required.

Fig. 5.3.7.5 is an example of the header file cifvars.h built when *BuildSiv* reads the augmented dictionary of Fig. 5.3.7.4 with the '*-e*' option to interpret and store standard uncertainties.

The integer variable *errornum* stores a nonzero value if an error occurs in attempting to read a CIF, and an error message is stored in the character array *errormes*, indicating the nature of the problem. Errors generated by the input subroutine *cifsiv_* are not fatal to the parent application program, and will at worst discard the particular loop block or data item affected. The parser operates by discarding CIF data upon encountering an error until it reaches an understandable set of input values. So, for example, if three numbers appear after an item name instead of one, the second two will be ignored after the error variables have been set, and parsing will continue. Similarly, if a serious error occurs within a loop, such as the appearance of an item name not matching an array variable, the entire loop is normally ignored. If a new packet of looped data exceeds the specified array limits, all further data in that loop are ignored.

The *cifsiv_* function has prototype

`void` *`cifsiv_`*`(char*` *`filename`*`, char*` *`blockname`*`)`

and requires pointers to character strings containing the name of the input file and the data-block code from which input is required.

A simple example C application illustrating the use of the *cifsiv_* subroutine is given in Fig. 5.3.7.6.

5.3.7.2.4. *Input to a Fortran application program*

A Fortran program can make use of the C input function generated by *BuildSiv* as long as the compiler used is capable of linking C and Fortran modules. For Fortran applications, the '*-f*' command-line option is used:

`BuildSiv -f` *`dictfile ddlversion`*

A C structure is defined for use within the *cifsiv_* subroutine and an identically constructed Fortran common block is built for use within Fortran routines. The first variable within the common block *must* be passed as an additional argument when the *cifsiv_* function is called. In the current implementation, that variable is