Guide To Using GPSS/H External Ampervariables

www.wolverinesoftware.com

1. INTRODUCTION

This guide contains information on how to develop External Ampervariables (External Routines) in C or C++ and use them with GPSS/H Professional. Although you can develop routines using either C or C++, in the descriptions that follow, we will use "C++" to refer to the development language.

When Should I Use an External Ampervariable?

Wolverine has worked hard over the years to make it unnecessary to go outside the GPSS/H language when developing a model. Only in very special circumstances should it be necessary or desirable to use an External Ampervariable. Examples of such circumstances are:

- The use of pre-existing or live-system software in a simulation model.
- The need for very complex computational code.
- The need for special I/O processing to interface with non-ASCII data files.
- Specialized user-interface requirements.

Mixed-language development (no matter which languages are involved) requires great care and careful attention to detail. If your modeling situation merits the use of one or more External Ampervariables, be sure to read these instructions thoroughly and follow them carefully.

How Do I Create External Ampervariables?

Four steps are required:

1. Write the External Ampervariables, paying careful attention to the rules by which arguments are passed to the routine by GPSS/H.
2. Compile the External Ampervariables with a C++ compiler, following the requirements specified below.
3. Link the External Ampervariables into a Windows Dynamic Link Library (DLL), "exporting" the names to be called from GPSS/H.
4. Test the External Ampervariables by using a C++ main program that mimics the way your GPSS/H model will call the routine.

In the sections that follow, each of these steps is discussed in detail. The discussion assumes that you have a basic understanding of programming in C++, and explains only the special considerations necessary to create GPSS/H-callable External Ampervariables.

2. WRITING EXTERNAL AMPERVARIABLES IN C++

2.1 General Guidelines

The Wolverine-provided header file `gpsshdll.h` should be `#include`d in any External Ampervariable that you write in C++. It contains data and structure definitions, function prototypes, and documentation for Wolverine-provided utility routines. For an example of the use of `gpsshdll.h`, see the Wolverine-provided `hdll1.cpp`. You should copy `gpsshdll.h` into the directory that will contain your C++ source code.

Be sure to specify __cdecl calling conventions. Prefixing the names of your C++ functions with "__cdecl" will force your C++ compiler and linker to (1) generate function names that GPSS/H can recognize in your DLLs, and (2) follow the calling conventions GPSS/H expects.

Be sure you are familiar with the contents of Chapter 8 ("EXTERNAL AMPERVARIABLES") of the GPSS/H Reference Manual.

The following rules apply to all External Ampervariables written in C++:

- The name of each C++ function should match the name of its corresponding External Ampervariable's in a GPSS/H model. Be sure to be consistent in your use of upper- and lower-case.
- An External Ampervariable that does not include a “main” program shares GPSS/H's stack. While Windows may be able to expand the stack when necessary, you are *guaranteed* approximately 25,000 bytes of stack space upon entry to the AMPER-VARIABLE. If you need more memory, use malloc() to get it and free() to relinquish it when it is no longer needed, rather than allocating large local (stack-based) variables.
- An External Ampervariable may return a value of type `int`, a value of type `double`, or no value at all. ***If an External Ampervariable returns a value of type*** `int`***, its name in the GPSS/H model must begin with one of the letters I through N. (This is a very old Fortran convention.)***
- A maximum of 20 arguments can be passed to the routine.
- Each argument can be a constant, an SNA/SCA/SLA, or a calculated expression.
- All arguments to the routine are passed by address, so all argument declarations in an External Ampervariable must be of the form "***pointer to*** argument-type".
- If a data item that can have a value assigned to it in the GPSS/H model is used as an argument, the address of the *actual data item* is passed to the External Amper-variable. Thus such data items can have values assigned to them in the External Ampervariable. For all other arguments, including constants other than character constants, GPSS/H makes a copy of the argument's value and passes the address of the *copy* to the External Ampervariable. This prevents the External Ampervariable from modifying something that it should not be able to change. One exception to this rule is that character constants are passed in a way that does not preclude their modification, although modifying a string constant is obviously a bad idea.
- If an element of an array data-type (Ampervariable Array or Matrix Savevalue) is specified as an argument, GPSS/H passes the address of *that single element* to the External Ampervariable. To gain access to an entire array in an External Ampervariable, specify as an argument either element (1) of a one-dimensional array or element (1,1) of a two-dimensional array. Instructions for accessing the array are presented in a later section. Only *individual* elements of *Character* Ampervariable Arrays may be accessed.

2.2 Accessing GPSS/H Data in C++

The table below specifies the way GPSS/H passes each possible type of argument. For more details on accessing array and character data types, see Sections 2.3, 2.4, and 2.5.

GPSS/H Item	Size in Bytes	Passed As
Savevalue		
Byte	1	char *
Halfword	2	short *
Fullword	4	int *
Float	8	double *
Matrix Savevalue Element		
Byte	1	char *
Halfword	2	short *
Fullword	4	int *
Float	8	double *

GPSS/H Item	Size in Bytes	Passed As
Transaction Parameter		
Byte	1	char *
Halfword	2	short *
Fullword	4	int *
Float	8	double *
Ampervariable		
Integer	4	int *
Real	8	double *
Character*n	4	void *
Varying Character*n	4	void *
Ampervariable Array Element		
Integer	4	int *
Real	8	double *
Character*n	4	void *
Varying Character*n	4	void *
Integer Expression	4	int *
Floating-point Expression	8	double *
SNA		
Integer	4	int *
Real	8	double *
SCA		
Character*n	4	void *
Varying Character*n	4	void *
SLA	4	int * (Note: Value pointed to: T=1, F=0)

2.3 Accessing Integer and Real AMPERVARIABLE Arrays in C++

In GPSS/H, elements of an integer or real Ampervariable Array are stored contiguously in ascending order of the array index. To gain array-style access to an integer or real Ampervariable Array, specify its *first element* as an argument to the External Ampervariable. Then, in the External Ampervariable, declare the argument as a pointer to the type of element that the array contains. Any element in the array can then be accessed by employing a subscript in conjunction with the pointer that you have declared.

Remember that in C++ subscripts for an array with n elements start with 0 and run through n-1, while in GPSS/H they start with 1 and run through n.

For example, if you wanted to access the integer Ampervariable Array `&intarray,` you would write in the GPSS/H model:

```
         integer  &intarray(10)
         external &extamp
...
callext bcall    &extamp(&intarray(1))
```

Then, in the External Ampervariable, you would write:

```
EXTERN_C EXPORT void CDECL extamp(int * intarray)

        {
        intarray[0] = 1;          /* Assigns a value of 1 to &intarray(1) */
        intarray[1] = 2;          /* Assigns a value of 2 to &intarray(2) */
        }
```

In most cases, you will want to write the External Ampervariable so that it does not need to be changed if the size of the Ampervariable Array is changed within the GPSS/H model. This is easy to do if you pass the size of the Ampervariable Array, along with the address of its first element, to the External Ampervariable.

The above example thus could be extended as follows:

```
callext bcall    &extamp(&intarray(1),10)    (modified line in the gpss/h model)

EXTERN_C EXPORT void CDECL extamp(int * intarray, int * size)

        {
        int i;                    /* Subscript for accessing array elements*/
        int limit;                /* To hold the array limit */

        limit = *size - 1;        /* Express limit as a C++ subscript value
*/

        intarray[0] = 1;          /* Assigns a value of 1 to &intarray(1) */
        intarray[1] = 2;          /* Assigns a value of 2 to &intarray(2) */

        for (i = 2; i <= limit; ++i)  /* Same pattern for the rest */
                {
                intarray[i] = i + 1;  /* intarray[i] is &intarray(i+1) */
                }
        }
```

2.4 Accessing Matrix Savevalues in C++

In GPSS/H, elements of a Matrix Savevalue are stored contiguously. All the columns comprising a row are stored together, followed by all the columns of the succeeding row. For example, the elements of a 3x3 Matrix Savevalue would be stored in memory in the following sequence:

(1,1) (1,2) (1,3) (2,1) (2,2) (2,3) (3,1) (3,2) (3,3)

To gain array-style access to a Matrix Savevalue, specify its (1,1) element as an argument to the External Ampervariable. Then, in the External Ampervariable, declare the argument as a pointer to the type of element that the array contains. You can then access any element in the array by employing row and column subscripts in conjunction with the pointer that you have declared.

Remember that in C++ subscripts for an array with n elements start with 0 and run through n-1, while in GPSS/H they start with 1 and run through n.

For example, if you wanted to access the Matrix Savevalue `MX$EXAMPLE`, you would write in your GPSS/H model:

```
Example matrix   MX,10,20
        external &extamp
...
callext bcall    &extamp(MX$example(1,1))
```

In the External Ampervariable, you would then write:

```
EXTERN_C EXPORT  CDECL extamp(int intarray[10][20])

        {
        intarray[0][0] = 1;          /* Assigns 1 to MX$example(1,1) */
        intarray[9][19] = 2;         /* Assigns 2 to MX$example(10,20) */
        }
```

In most cases, you will want to write the External Ampervariable so that it does not need to be changed if the size of the Matrix Savevalue is changed within the GPSS/H model. This is easy to do if you pass the size of the Matrix Savevalue, along with the address of its first element, to the External Ampervariable.

The C++ language does not allow the size of an array to be automatically "inherited" from a subroutine's caller, but the desired effect can be accomplished through the use of a C++ MACRO provided by Wolverine. By converting a two-dimensional subscript into an *equivalent* one-dimensional subscript, the MACRO will allow the Matrix Savevalue argument to be declared in the External Ampervariable as a pointer instead of as a two-dimensional array. The MACRO, named `HROW_HCOL()`, is defined in `gpsshdll.h`, and its use is illustrated below. Note that `HROW_HCOL()` also converts GPSS/H subscripts (1 to n) into C++ subscripts (0 to n-1).

Using the MACRO, the above example could be extended as follows:

```
callext bcall    &extamp(MX$example(1,1),10,20)    (modified line in the gpss/h model)


#include "gpsshdll.h"  /* GPSS/H header file for External Ampervariables */


EXTERN_C EXPORT void CDECL extamp(int * intarray, int * numrows, int *
numcols)

   {
   int i, j;                    /* Subscripts for accessing array elements*/
   int nrows;                   /* To hold number of rows in array */
   int ncols;                   /* To hold number of cols in array */

   nrows = *numrows;            /* A having to use "*" all the time */
   ncols = *numcols;            /* Likewise */

   intarray[HROW_HCOL(1,1,ncols)] = 1;    /* Assigns 1 to MX$EXAMPLE(1,1) */
   intarray[HROW_HCOL(1,2,ncols)] = 2;    /* Assigns 2 to MX$EXAMPLE(1,2) */

   for (i = 1; i <= nrows; ++i)                 /* Outer loop on row */
        {
        for (j = 1; j <= ncols; ++j)            /* Inner loop on column */
                {
                intarray[HROW_HCOL(i,j,ncols)] = 17;     /* Set all to 17 */
                }
        }
   }
```

2.5 Accessing Character-type Constants, AMPERVARIABLEs, and AMPERVARIABLE Arrays in C++

As described in the GPSS/H Reference Manual, values assigned to *fixed*-length character Ampervariables are stored left-justified, and either extended with blanks or truncated if the value being assigned is shorter or longer, respectively, than the destination Ampervariable. Values assigned to *varying*-length character Ampervariables are also stored left justified, but are never extended with blanks. Instead, the Ampervariable's current length is set to the length of the value being assigned. If the value being assigned is longer than the Ampervariable's maximum length, the current length is set to the maximum length and the text to be assigned is truncated to the maximum length.

When a character constant, character Ampervariable, or element of a character Ampervariable Array is used as an argument to an External Ampervariable, GPSS/H does not pass the address of the actual character data item. Instead, it passes a pointer to a special descriptor that facilitates accessing the item in C++. We do not publish the details of this descriptor; rather, we provide two utility functions for exchanging character data between GPSS/H and C++. These functions are described in Section 6.

NOTE: Take care not to assign a new value to a character constant. Because it uses the same kind of descriptor as does a character Ampervariable, a character constant is susceptible to modification. ***If the value of a character constant is changed, the original value is lost and will not be restored when the External Ampervariable is next called.***

Elements of GPSS/H ***character Ampervariable Arrays are not always stored contiguously, so they cannot be accessed as an array*** in an External Ampervariable. Individual elements of a character Ampervariable Array, however, can be accessed in an External Ampervariable.

3. COMPILING EXTERNAL AMPERVARIABLES WRITTEN IN C++

For use as GPSS/H External Ampervariables, C++ functions must (1) follow __cdecl calling conventions, and (2) be exported to a Windows Dynamic Link Library (DLL). In some compilers, "__cdecl" is the default, while in others, e.g., Open Watcom, it is not. The Wolverine-provided header file gpsshdll.h includes C++ macros named CDECL and EXTERN_C that you can use when defining C++ functions to be exported. Exporting function names can be done in at least two different ways. The simplest way is to prefix functions to be exported with the Wolverine-provided EXPORT macro. If this approach doesn't work with your C++ compiler, you may have to manually specify to your linker the names of functions to be exported. The ease of doing so depends on the quality of the development environment you are using.

The following is a representative C++ function using Wolverine-provided macros:

```
EXTERN_C EXPORT int CDECL myfunc(int *i, int *j)
```

GPSS/H recognizes two special function names in each DLL you use. If a DLL contains a function named "Connect", GPSS/H will call "Connect" when it loads the DLL. If a DLL contains a function named "Disconnect", GPSS/H will call it at the conclusion of running your GPSS/H program. You can take advantage of this convention to perform initialization before any of your other DLL functions are called and to clean up at the conclusion of execution.

If you wish to use Wolverine-provided utility functions in a DLL, you *must* provide a "Connect" function, and in that function you must save a static copy of a pointer to a vector of pointers to the utility functions. Details are given in Section 6.

4. TESTING EXTERNAL AMPERVARIABLES WRITTEN IN C++

GPSS/H is powerless to protect itself against pointer errors, array-bounds errors, and other mishaps that may occur in an External Ampervariable. Thus ***you must be especially careful to test your routine thoroughly before actually using it with a GPSS/H model.*** The best way to do this is to write a C++ main program that calls the External Ampervariable, passing it the same values that will be passed when the routine is used with your model. Even better, pass it some values that it doesn't expect to "see", and ensure that it can properly handle these error conditions. ***Pay special attention to code that determines pointer values, starting and ending points of loops, whether an array reference is safely within the array bounds, and so on.*** You should also test any code that does input or output.

5. LINKING EXTERNAL AMPERVARIABLES WRITTEN IN C++

After an External Ampervariable has been compiled and tested, it must be incorporated into a Windows Dynamic Link Library (DLL). Depending on which C++ development environment you're using, this may be very easy or a bit tedious. The process by which routines linked into a DLL are "published" to the outside world is known as "exporting." The use of the Wolverine-provided EXPORT C++ macro may make it easier to export C++ function names. Please review the discussion of EXPORT in Section 3.

Note: An External Ampervariable runs using GPSS/H's stack, so changing the stack size specified in hext.lnk has no effect on the stack space available to your routine.

6. SPECIAL UTILITY ROUTINES PROVIDED BY GPSS/H

GPSS/H provides a number of built-in special-purpose functions that can be called from External Ampervariables. These functions are located within GPSS/H. You do not need to load them from a library. GPSS/H passes a pointer to an array of pointers to these functions to the "Connect" entry point of a DLL. The following example comprises a minimal "Connect" function:

```
#include "gpsshdll.h"

        extern struct GPSSHTransferVector *GPSSHTV = NULL;

//      Define Connect function (to setup transfer vector of utility routines)

EXTERN_C EXPORT void CDECL Connect(struct GPSSHTransferVector *gpsshtv)
        {
        GPSSHTV = gpsshtv;      // Save address of transfer vector
        return;
        }
```

The sample code above uses macros and names declared in gpsshdll.h. GPSS/H provides the following functions for your use:

Function	Purpose
`GPSSH_GetCharacterAmper`	Transfers GPSS/H character data to C++
`GPSSH_SetCharacterAmper`	Transfers character data fro C++ to GPSS/H
`GPSSH_WriteToScreen`	Writes a message to the screen in an interactive run
`GPSSH_WriteToListing`	Writes a message to a program's listing file

The calling sequences for these functions are given in `gpsshdll.h`, and examples of their use are given in a sample program named `hdll1.gps`, and `hdll1.cpp`.