NAME

qsage - C code generator for simplified access to qsage library

SYNOPSIS

**qsage **

**-p program-name **

**-i initialization-function **

**-o objective-function **

**-f finalization-function **

**[-b {debug|optimize}] [-a argspec] [-v] [-s] **

**C-source-1.c ... **

C-object-1.o ...

program-name -t -o -v -E [additional arguments]

DESCRIPTION

qsage simplifies the task of writing C code to access the sapi_solveQsage(...) function from the SAPI library. In its default mode, **qsage** generates C source files and compiles them with user-provided C to create an executable. When used as a code generator, **qsage** emits C source files that can be compiled and linked with user-provided C to create a complete executable.

[-a argspec]

The generated executable usually requires command line arguments to control its behavior. Each **-a** argument to **qsage** defines a command line argument which may be passed to the generated executable. The required **argspec** value following each -a specifies four pieces of information:

1) command option letter
2) variable type
3) variable name
4) variable default value

Variable type must be either integer, double or string. Variable name must be a legal C variable name. Variable default value specifies the value that the C variable will have if no corresponding command line argument is used.

All four values must be concatenated with a colon (:) separator. This combination of –a and an argspec

```
-a n:integer:N:0
```

causes the generated executable to look for a command line argument –n which would be followed by an integer value. This command line argument would be parsed and placed into a variable named N. If the –n command line argument is not present, then the N variable will have value 0.

Multiple –a arguments can be passed to **qsage**. Each variable defined in this way is at file scope and may be used anywhere in the user-provided C program files.

-b {debug|optimize}

The optional build flag (-b) controls the type of compilation when **qsage** is used to generate an executable. If –b debug is specified, the compiled code will retain symbol table information and so can be easily debugged. If –b optimize is specified, the compilation step will use optimization.

When **qsage** is used to generate C source files, the –b option controls the compilation commands in the compile.bash script.

If multiple –b options appear, all but the last one are ignored.

-i initialization-function

-o objective-function

-f finalization-function

These three required arguments allow the user to specify function names in the C program. The initialization-function is called once after parsing command line arguments. Any logic necessary to prepare for evaluation of the objective function must be included in the initialization-function. The objective-function is called

many times during execution. This is the function which is minimized by the qsage library routines using the D-Wave quantum computer. The finalization-function is called once at the end of execution and usually writes out the argument which minimizes the objective-function.

The return value from the initialization-function is an integer which is the number of Boolean variables over which the objective-function is defined.

-p program-name

The name of the generated executable is program-name.

-s

This optional argument controls whether **qsage** stops after generating source code (if **-s** is specified) or continues and generates an executable (if **-s** is not specified). If **-s** is specified, a new subdirectory of the current working directory is created and the generated source code is put there. The name of the new subdirectory is the concatenation of a dot, the program name, and underscore and a random four-digit numeric string. A shell script named compile.bash is also generated in the new subdirectory which contains compile and link commands to build the final executable. The name of the new subdirectory is written in a message on standard output.

-v

If present, this optional argument enables verbose output from **qsage** while it parses arguments. If **qsage** is invoked to generate an executable, the **-v** argument also causes the value of DWAVE_HOME and the individual compilation and link commands to be displayed.

In default mode **qsage** generates an executable named **program-name**, whose name is specified via the required –p argument. The generated executable has several built-in arguments which are always recognized (see below) as well as custom arguments which are defined via the **-a argspec** arguments to **qsage**. When invoking **program-name** the user will specify values for some built-in arguments followed by custom arguments. The **-E** option letter marks the end of the built-in arguments and following it, all remaining arguments are assumed to be custom.

-t NNN

This sets the timeout value. **Program-name** will generally execute until the search terminates at the specified objective value or until the timeout has been reached.

-o OOO

This sets the objective value.

-v

This turns on verbose output from the sapi_solveQsage(...) library function.

-E

This marks the end of the built-in arguments and the start of the custom arguments.

EXAMPLE

Suppose the user has written a C source file named hadamard.c which includes definitions for the functions hadamard_init(), hadamard_obj() and hadamard_final(). **Qsage** is invoked to generate an executable named hadamard. Hadamard is invoked with built-in arguments to determine the target objective value, the timeout and its verbosity mode. Following these arguments, an additional custom argument determines the specific search problem. At run time, the generated executable parses its command line arguments and then invokes hadamard_init(). The hadamard_init() function allocates data structures which depend on values passed via custom arguments. After hadamard_init() runs, the generated executable calls hadamard_obj() multiple times. Each invocation of hadamard_obj() evaluates the objective function once at its argument values. The function hadamard_final() displays the final result.

To support this calling sequence, invoke **qsage** with one argument specifier for the required integer value needed by hadamard_init(). Use **qsage** in its default mode by omitting the optional **-s** argument. Include the **-b debug** option to allow for debugging. Include the three functions mentioned above in a single C source file named hadamard.c. Invoke **qsage** as follows:

```
qsage -p hadamard –b debug –a n:integer:N:0 \

     -i hadamard_init -o hadamard_obj -f hadamard_final \

      hadamard.c
```

The above invocation of **qsage** creates an executable program named hadamard. In addition to the built-in command line arguments -t, -o and -v, the user provides the custom command line argument -n following the -E argument. The -n argument should be followed by an integer value. Corresponding to the custom argument is a single integer variable N in the generated C program.

The prototypes for the initialization, objective and finalization functions are as follows:

```
int initialization-function();

double objective-function(const int* state, size_t len);

void finalization-function(const int* state, size_t len);
```

The return value from the initialization function sets the number of Boolean variables over which the search takes place. The objective function should assume that **state** points to an array of length **len**. Each element in the **state** array contains 0 or 1. The objective function returns the objective value resulting from evaluating the objective at the point in search space specified by its input arguments. The arguments to the finalization function are the same as those for the objective function. The finalization function is called once after the search has terminated. The arguments to the finalization function represent the point in search space with the minimum objective value found by **qsage**, so the finalization function can write out the final result in whatever format is most convenient.

In addition to any standard include files needed in the C source files provided to **qsage**, the following header should also be included:

```
#include "qsage.h"
```

The file qsage.h is automatically generated by **qsage** and contains declarations of the initialization, objective and finalization function as well as variables corresponding to custom arguments.

The generated executable can be called as follows:

```
hadamard -v -t 100 -o 0 -E -n 4
```

BUGS

Please report bugs to dwsupport@dwavesys.com.

COPYRIGHT

SEE ALSO

Developer Guide for C, Release 2.2, Chapter 8