QSpice C-Block Components

The EngAtof() Function

Caveats & Cautions

First and foremost: Do not assume that anything I say is true until you test it yourself. In particular, I have no insider information about QSpice. I've not seen the source code, don't pretend to understand all of the intricacies of the simulator, or, well, anything. Trust me at your own risk.

If I'm wrong, please let me know. If you think this is valuable information, let me know. (As Dave Plummer, ex-Microsoft developer and YouTube celebrity (Dave's Garage) says, "I'm just in this for the likes and subs.")

Note: This is the ninth document in a clearly unplanned series:

- ***The first C-Block Basics document covered how the QSpice DLL interface works.***
- ***The second C-Block Basics document demonstrated techniques to manage multiple schematic component instances, multi-step simulations, and shared resources.***
- ***The third C-Block Basics document built on the second to demonstrate a technique to execute component code at the beginning and end of simulations.***
- ***The fourth C-Block Basics document revisited the Trunc() function and provided a basic "canonical" use example.***
- ***The fifth C-Block Basics document revisited the MaxExtStepSize() function with a focus on generating reliable clock signals from within the component.***
- ***The sixth C-Block Basics document examined connecting and using QSpice bus wires with components.***
- ***The seventh C-Block Basics document described recent QSpice changes (a new Display() function changes to Trunc()).***
- ***The eighth C-Block Basics document detailed post-processing techniques and much more in painful detail.***

This paper provides information about the EngAtof() function available in C-Block code. It assumes that you already have a good understanding of creating C-Block Components. See my GitHub repository here for prior documents in this series.

Overview

The following curiosity appeared in the QSpice release notes:

```
08/20/2025 In a C++ .DLL, added access to user-defined parameters, mathematical expressions, and
SPICE metric multipliers via extern "C" __declspec(dllexport) double (*EngAtof)(const char
**string) = 0;
```

I was intrigued. Mike Engelhardt, the author of QSpice, was kind enough to provide an example. `EngAtof()` is more clever than I would have guessed. Many thanks to Mike for his excellent support.

This paper gives a brief explanation of the `EngAtof()` function based on my investigations. Example schematic and DLL code are provided. (You're going to need them.)

The EngAtof() Function

Stated broadly, `EngAtof()` gives a DLL simulation-runtime access to the QSpice expression evaluation engine.

Yes, that's pretty vague. But `EngAtof()` is a bit complicated…

Take a look at the sample code. It works like this:

- You call `EngAtof()` with a pointer to a string that contains one or more "expressions" to evaluate.
- `EngAtof()` returns the value of the first expression (if valid) and advances the pointer to the beginning of the next expression in the string or the end of string marker (null byte). If the expression is invalid, the function returns `NaN` (Not a Number).

An "expression" is pretty much any expression that you can enter in a user-defined variable (i.e., a `.param` statement) or component value.

Using the `EngAtof()` function, you can do some interesting things:

- Obtain `.param` values without passing them to the DLL as String Attributes.
- Verify that a `.param` value exists.
- Have QSpice evaluate the result of a user-defined function (`.func`) with a runtime argument.

I could write for many pages and the above won't get any clearer. Let's move on to the examples.

The Example Files

The example files demonstrate how to use `EngAtof()`. They don't explain when and why you might find them useful. That's covered later in this paper.

The Schematic

CBlockBasics9.qsch passes a String Attribute parameter to the DLL:

```
char *ExprStrAttr="100mV 2e-3 {2 * EvilVal} rint(pi) UserFunc(2)*2"
```

`ExprStrAttr` is a list of space-delimited expressions to evaluate in the DLL. It doesn't really serve a practical purpose – it's here merely to demonstrate how `EngAtof()` parses a list of expressions.

The schematic also contains a user-defined variable and a user-defined function:

```
.param EvilVal=666
.func UserFunc(z) {z*z*z}
```

You should change all of the above, delete them (or comment them out), or whatever and run simulations to test the `EngAtof()` `*.cpp` code. (Really, please do play around with them to see what happens and why. No DLL recompiles required….)

The DLL Code

CBlockBasics9.cpp has three clearly marked sections that demonstrate using `EngAtof()` to evaluate those bits.

Code Section: Ex 1

This section is key. It demonstrates:

- How to set up a call to `EngAtof()` to parse a list of expressions.
- How to detect expression parsing errors.
- How to identify where in the expression list the parsing failed.

Things to note:

- `EngAtof()` understands metric multipliers (“K”, “m”, etc.) and applies them to the computed result.
- `EngAtof()` ignores units (“V”, “A”, “S”, etc.).
- `EngAtof()` understands engineering notation (“10e-5”), mathematical expressions, and built-in functions (see Simulator | General Conventions in QSpice Help for more information).
- `EngAtof()` applies `.param` and `.func` expressions.
- We test for `NaN` using `isnan()`. This is the only reliable way to detect `NaN`.

In particular, notice that all expression text is in lower case. This happens automatically when String Attributes are passed.

Also, since the expression parser treats spaces as delimiters, you must enclose expressions that contain spaces in “()” or “{}”.

Code Section: Ex 2

Before `EngAtof()`, the only way to get a `.param` value in a DLL was to pass it as a String Attribute parameter. This section uses `EngAtof()` to fetch a user-defined variable (`.param`) value directly from QSpice. It also demonstrates how to test whether or not a user-defined variable exists.

Note that the expression must be lower case.

Code Section: Ex 3

This section demonstrates how to construct an expression containing a runtime variable. It also demonstrates how to evaluate a user-defined function. Change `UserFunc()` and, without changing the DLL code, the DLL “magically” does something different.

When To Use EngAtof()

As we have seen, `EngAtof()` allows a DLL to parse expressions at simulation runtime. We can determine if user-defined variables are present and, if so, fetch their values. We can evaluate user-defined functions with runtime values. It’s all pretty cool. But when should we use it?

First, let's acknowledge that `EngAtof()` expression evaluation calls must – by definition – be computationally expensive – QSpice has to parse and evaluate the expression.

That said, as long as we only call `EngAtof()` within the one-time initialization section of the evaluation function, this seems a reasonable way for a DLL to get values.

By way of contrast, consider Example #3 where we dynamically constructed a call to `UserFunc()`. Imagine that we wanted to evaluate `UserFunc(x)` for a value of `x` that changed for every simulation point. While a sweet and clever thing to do, it is going to be slow. You should try not to do that.

Conclusion

I hope that you find the information useful. Please let me know if you find problems or suggestions for improving the code or this documentation.

--robert