

MathParser User Manual

Richard Spooner

April 17, 2008

Contents

1	Overview	2
1.1	Supported Functions	2
1.2	Supported Operators	3
2	Usage	4
2.1	Variables	4
2.1.1	Component Variables	4
2.1.2	Signal Variables	4
2.2	Offline Configuration	5
2.2.1	Model XML Files	5
2.2.2	Component XML File	5
2.3	Online Configuration	5
3	Example Operations	6
3.1	Counter	6
3.2	Counter and test	7
3.3	Add two CDP signals (alias routing)	8
3.4	Add two CDP signals (full routing)	9
3.5	Nested if's	10

Chapter 1

Overview

MathParser is a CDP component for creating CDP signals from mathematical manipulation of existing signals and internal variables. It is based on muParser, a fast math parser library freely available from <http://muparser.sourceforge.net/>

1.1 Supported Functions

Table 1.1 shows the functions supported in version 2.001 of MathParser.

Name	Description
sin	sine function
cos	cosine function
tan	tangent function
asin	arc sine function
acos	arc cosine function
atan	arc tangent function
sinh	hyperbolic sine function
cosh	hyperbolic cosine function
tanh	hyperbolic tangent function
asinh	hyperbolic arc sine function
acosh	hyperbolic arc cosine function
atanh	hyperbolic arc tangent function
log2	logarithm to the base 2
log10	logarithm to the base 10
log	logarithm to the base 10
ln	logarithm to the base e
exp	e raised to the power of x
sqrt	square root of a value
sign	sign function -1 if x<0; 1 if x>0
rint	round to nearest integer
abs	absolute value
if	if ... then ... else
min	min of all arguments
max	max of all arguments
sum	sum of all arguments
avg	mean value of all arguments

Table 1.1: Supported functions

As the source code is distributed along with this component it is a straight forward task to add additional functions. Consult the muParser documentation at <http://muparser.sourceforge.net/> for further information. However, if you feel that the function you require might be of use to other MathParser users please contact me at richard.spooner@rolls-royce.com and I'll add the desired function to this distribution.

1.2 Supported Operators

Table 1.2 shows the binary operators supported in version 2.001 of MathParser.

Name	Description
=	assignment
and	logical and
or	logical or
xor	logical exclusive or
<=	less than or equal to
>=	greater than or equal to
!=	not equal to
==	equal to
<	less than
>	greater than
+	addition
-	subtraction
*	multiplication
/	division
^	raise x to the power of y

Table 1.2: Supported operators

As the source code is distributed along with this component it is a straight forward task to add additional binary operators. Consult the muParser documentation at <http://muparser.sourceforge.net/> for further information. However, if you feel that the operator you require might be of use to other MathParser users please contact me at richard.spooner@rolls-royce.com and I'll add the desired operator to this distribution.

Chapter 2

Usage

For each mathematical expression required, an Operation element must be defined within the Operations element of a MathParser component. See section 3 for some simple examples.

Table 2.1 shows the attributes available within the Operation element:

Name	Required	Description
Name	Yes	Name of output signal
Expression	Yes	Mathematical expression to be evaluated
Alias	No	Alias of output signal
Routing	No	Push routing of output signal
Unit	No	Engineering unit

Table 2.1: Operation attributes

For each Operation an CDP output signal will be created. The expression will be evaluated $\langle fs \rangle$ times per second and the output signal will contain the result of the evaluation. Operations will be executed in the order in which they appear in the Operations tag of the component XML file. It is therefore possible to build up complex multi-line expressions.

The first time an expression is evaluated it is optimised and compiled to bytecode. Subsequent evaluations are performed using the compiled bytecode resulting in much improved performance.

2.1 Variables

Variables used in expressions can be of two types: *component variables* and *signal variables*.

2.1.1 Component Variables

Component variable are variables defined within the context of the CDP component. They are defined by prefixing an alpha-numeric string with a \$ symbol.

The following characters can be used in component variable names:

`.$0123456789_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ`

Examples of valid component variables are:

`$test, $test123, $test_one`

2.1.2 Signal Variables

Signal variables are those which read values from CDP signals. They are defined by giving the variable the same name as the CDP signal (or preferably its alias as this leads to shorter and more readable expressions).

For each signal variable you define in your MathParser component a corresponding input signal will automatically be created with the suffix `_ip`. This input signal will be automatically routed to the CDP signal you wish to access.

The following characters can be used in signal variable names:

`.0123456789_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ`

Examples of valid signal variables are:

`Application.Component.Signal, test123, test_one`

2.2 Offline Configuration

2.2.1 Model XML Files

```
<?xml version="1.0" ?>
<Model>
  <Type>MathParser</Type>
  <ModelTypeClass>CDPComponent</ModelTypeClass>
  <BaseModel>CDPComponent</BaseModel>
  <TypeHelp>MathParser</TypeHelp>
  <Attributes></Attributes>
  <Messages></Messages>
</Model>
```

```
<?xml version="1.0" ?>
<Model>
  <Type>Operation</Type>
  <ModelTypeClass>CDPObject</ModelTypeClass>
  <BaseModel>Signal<double></BaseModel>
  <TypeHelp>MathParser Operation</TypeHelp>
  <Attributes></Attributes>
  <Messages></Messages>
</Model>
```

2.2.2 Component XML File

```
<?xml version="1.0" ?>
<Component Name="Example" Type="MathParser">
  <Activate>1</Activate>
  <fs>1</fs>
  <InstanceHelp></InstanceHelp>
  <Alarms></Alarms>
  <Signals></Signals>
  <Parameters></Parameters>
  <Operations>
    <Operation Name="PI" Expression="3.14159"></Operation>
  </Operations>
</Component>
```

2.3 Online Configuration

Using the CDP web interface it is possible to change the Alias, Routing and Unit attributes of the Operation element at run-time.

It is also possible to change the Expression attribute at run-time by editing the Description text of the generated output signal at the CDP web interface. Each time the Expression attribute is changed it is recommended that the ExpressionValid property of the CDP output signal (or the application message log) be checked to ensure that the entered expression is valid. Non-valid expressions are not evaluated. New variables and input signals are created at runtime as required. Existing variables and input signals no longer required will be removed next time the application is started.

Due to limitations in the CDP web interface the characters +, < and > cannot be entered into the description field. As a workaround, the sum() function can be used instead of the + operator. Otherwise the XML file must be edited directly and the application restarted.

It is currently not possible to add new operations at runtime.

Chapter 3

Example Operations

3.1 Counter

This example demonstrates the creation of a counter which increments each second.

```
<?xml version="1.0" ?>
<Component Name="Example1" Type="MathParser">
  <Description><![CDATA[Test]]></Description>
  <Activate>1</Activate>
  <fs>1</fs>
  <InstanceHelp></InstanceHelp>
  <Alarms></Alarms>
  <Signals></Signals>
  <Parameters></Parameters>
  <Operations>
    <Operation Name="Counter" Expression="$counter = $counter + 1"></Operation>
  </Operations>
</Component>
```

Notes:

- A component variable \$counter will be created
- A CDP output signal named Counter will be created which will hold the result of the expression \$counter = \$counter + 1
- The c/c++ construct \$counter++ is not valid in MathParser

3.2 Counter and test

This example demonstrates the creation of a counter which increments each second and a test for a value greater than a constant. The value of the signal 'Test' will be zero when the variable '\$counter' is less than 100 and 1 when the variable '\$counter' is greater then or equal to 100.

```
<?xml version="1.0" ?>
<Component Name="Example2" Type="MathParser">
  <Description><![CDATA[Test]]></Description>
  <Activate>1</Activate>
  <fs>1</fs>
  <InstanceHelp></InstanceHelp>
  <Alarms></Alarms>
  <Signals></Signals>
  <Parameters></Parameters>
  <Operations>
    <Operation Name="Counter" Expression="$counter = $counter + 1"></Operation>
    <Operation Name="Test" Expression="if( $counter >= 100, 1, 0 )"></Operation>
  </Operations>
</Component>
```

Notes:

- A component variable \$counter will be created
- A CDP output signal named Counter will be created which will hold the result of the expression \$counter = \$counter + 1
- A CDP output signal named Test will be created which will hold the result of the expression if(\$counter >= 100, 1, 0)
- The c/c++ construct \$counter++ is not valid in MathParser

3.3 Add two CDP signals (alias routing)

This example demonstrates the addition of two CDP signals using alias routing.

```
<?xml version="1.0" ?>
<Component Name="Example3" Type="MathParser">
  <Description><![CDATA[Test]]></Description>
  <Activate>1</Activate>
  <fs>1</fs>
  <InstanceHelp></InstanceHelp>
  <Alarms></Alarms>
  <Signals></Signals>
  <Parameters></Parameters>
  <Operations>
    <Operation Name="Output" Expression="Input1 + Input2"></Operation>
  </Operations>
</Component>
```

Notes:

- A signal variable named Input1_ip will be created which will automatically be routed to the CDP signal with alias Input1
- A signal variable named Input2_ip will be created which will automatically be routed to the CDP signal with alias Input2
- A CDP output signal with the name Output will be created which will hold the result of the expression Input1 + Input2

3.4 Add two CDP signals (full routing)

This example demonstrates the addition of two CDP signals using full name routing.

```
<?xml version="1.0" ?>
<Component Name="Example4" Type="MathParser">
  <Description><![CDATA[Test]]></Description>
  <Activate>1</Activate>
  <fs>1</fs>
  <InstanceHelp></InstanceHelp>
  <Alarms></Alarms>
  <Signals></Signals>
  <Parameters></Parameters>
  <Operations>
    <Operation Name="Output" Expression="App.Comp.Sig1 + App.Comp.Sig2"></Operation>
  </Operations>
</Component>
```

Notes:

- A signal variable named App.Com.Sig1_ip will be created which will automatically be routed to CDP signal App.Comp.Sig1
- A signal variable named App.Com.Sig2_ip will be created which will automatically be routed to CDP signal App.Comp.Sig2
- A CDP output signal with the name Output will be created which will hold the result of the expression Input1 + Input2

3.5 Nested if's

This example demonstrates the use of nested if's. It will keep record of the number of seconds, minutes, hours and days the component has been running.

```
<?xml version="1.0" ?>
<Component Name="Example5" Type="MathParser">
  <Description><![CDATA[Test]]></Description>
  <Activate>1</Activate>
  <fs>1</fs>
  <InstanceHelp></InstanceHelp>
  <Alarms></Alarms>
  <Signals></Signals>
  <Parameters></Parameters>
  <Operations>
    <Operation Name="S" Expression="$s = if(($s+1)>59,0,$s+1)"></Operation>
    <Operation Name="M" Expression="$m = if($s==0,if(($m+1)>59,0,$m+1),$m)"></Operation>
    <Operation Name="H" Expression="$h = if(($s+$m)==0,if(($h+1)>23,0,$h+1),$h)"></Operation>
    <Operation Name="D" Expression="$d = if(($s+$m+$h)==0,$d+1,$d)"></Operation>
  </Operations>
</Component>
```

Notes:

- Component variables \$s, \$m, \$h and \$d will be created
- CDP output signals with the names S (seconds), M (minutes), H (hours) and D (days) will be created