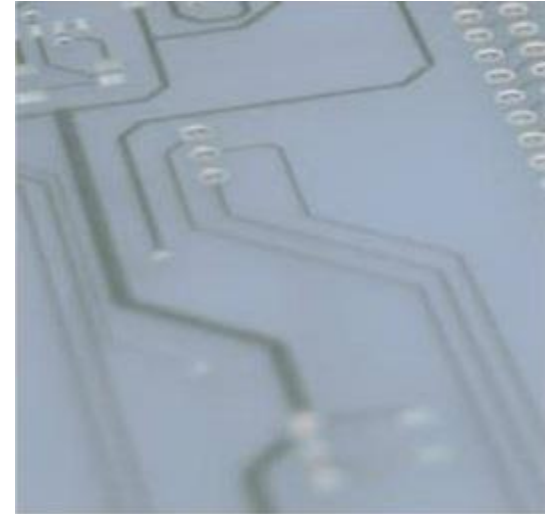
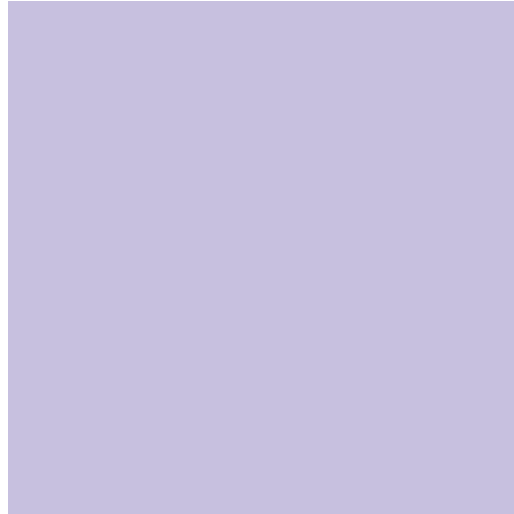
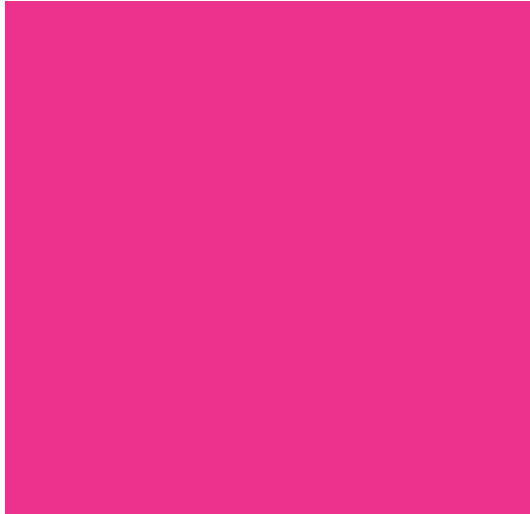


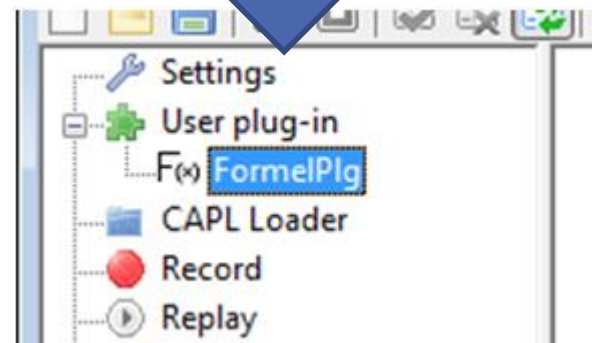
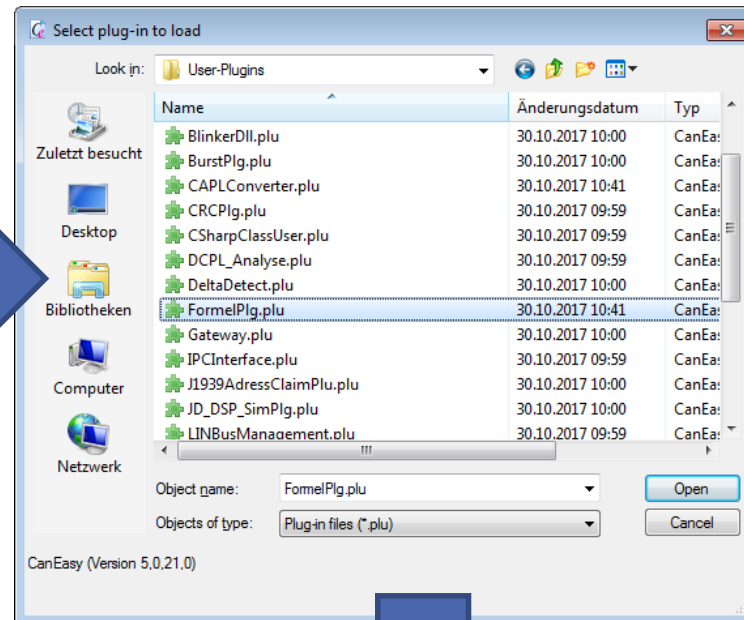
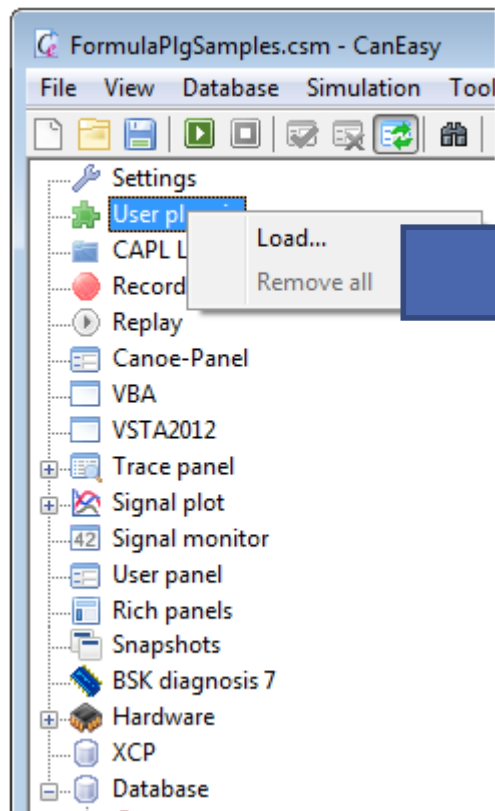
# Formula Plugin



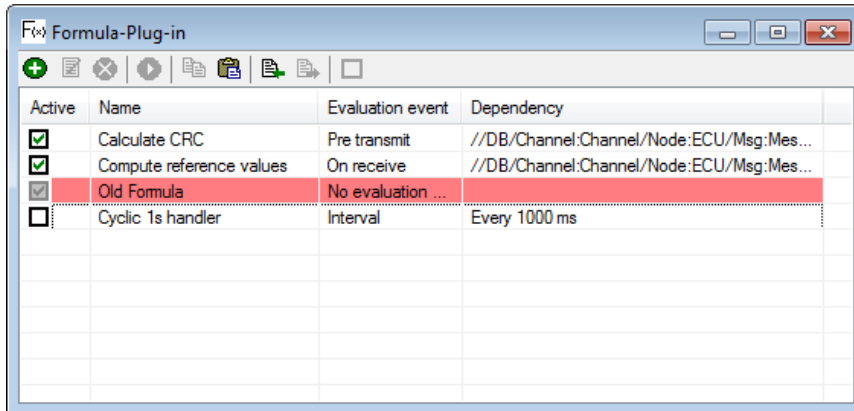
Michael Gerhardt  
Sven Schuchmann

- Instead of programming a behavior, a calculation of values based on simple equations is sometimes good enough
- Math is generally better understood than code
- Computed values based on signal values can easily give additional information instead of manually checking them, e.g.
  - Use signals to calculate reference values
  - Evaluate received checksums / alive counters

# Start



# Main Dialog



Standard control elements to edit shortcuts

- Create
- Edit
- Delete
- Import / Export

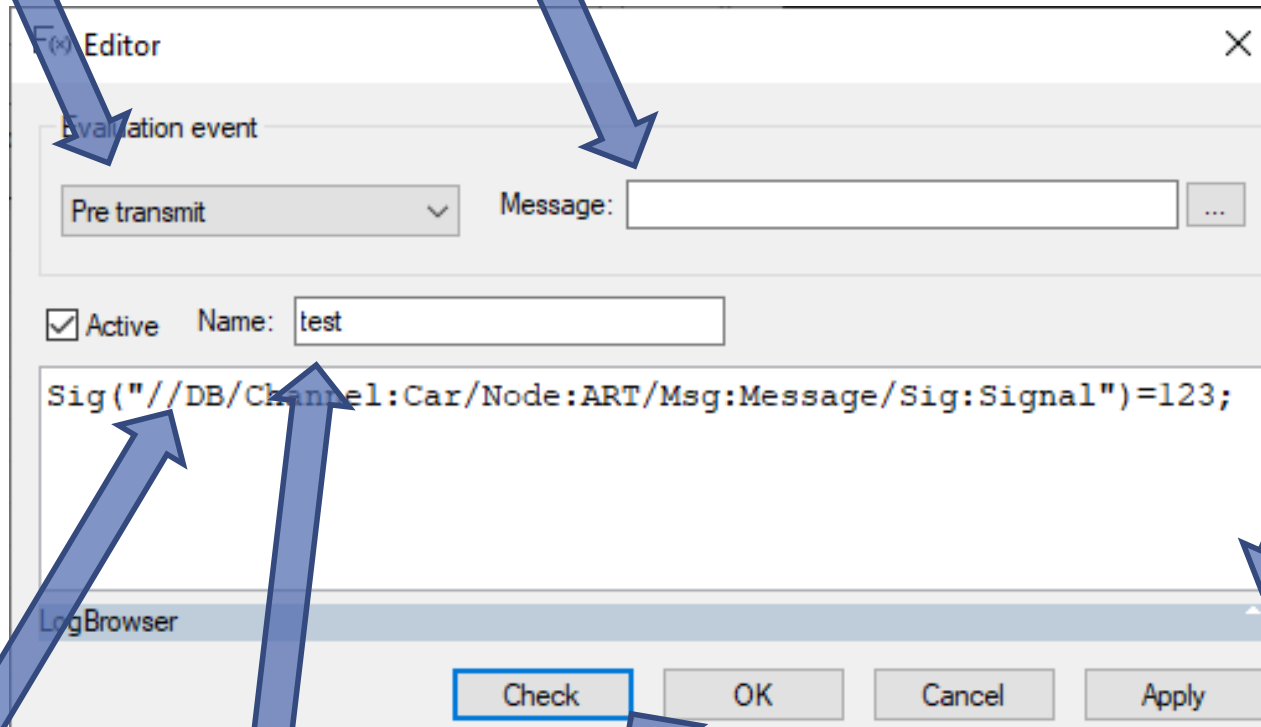
Information in overview	
Active	Formula will be evaluated / ignored
Name	descriptive name given by user
Evaluation event	when is this formula is executed [e.g. trigger, timer, etc.]
Dependency	additional information for evaluation event [e.g. linked Message, timing]
„Red line marker”	Formula can not be evaluated because of error

# Edit Dialog

Setting when formula should be evaluated

Additional config for evaluation event

Detailed error  
information



The screenshot shows the 'Formula Editor' dialog box. It has a title bar with a close button. The main area is divided into sections. The first section is 'Evaluation event' with a dropdown menu set to 'Pre transmit' and a 'Message:' text box. Below this is a section with a checked 'Active' checkbox and a 'Name:' text box containing 'test'. The main text area contains the formula: `Sig("//DB/Channel:Car/Node:ART/Msg:Message/Sig:Signal")=123;`. At the bottom is a 'LogBrowser' section. At the very bottom are four buttons: 'Check', 'OK', 'Cancel', and 'Apply'. Blue arrows point from text labels to specific parts of the dialog: 'Setting when formula should be evaluated' points to the 'Pre transmit' dropdown; 'Additional config for evaluation event' points to the 'Message:' text box; 'Unique name' points to the 'Name:' text box; 'Check for syntax/configuration errors' points to the 'Check' button; and 'Detailed error information' points to the 'LogBrowser' section.

Unique name

Check for syntax/configuration errors

Editable Formula, different equations divided by semicolon

# Evaluation types

## Determines evaluation time of formula

Trigger	
No evaluation	No automatic evaluation of the formula. An evaluation can be performed using the context menu of the formula list in the main window.
Pre Transmit	Evaluation is performed before the selected message is sent. Changes in message data at the time of evaluation are considered when the next message is sent.
After Transmit	Evaluation is performed after the specified message is sent.
On Receive	Evaluation is performed when the specified message is received (only available for real ECUs).
On Simulation Start	One-time evaluation at the start of the simulation.
On Simulation Stop	One-time evaluation at the end of the simulation.
As Action	Defines the formula as an action within CanEasy to make the formula available in the alarm and shortcut plug-ins as a response to events.
Interval	Cyclic evaluation according to the cycle time.
After evaluation unit	Evaluation of the formula according to the selected evaluation unit [Formula].

# Syntax of formulas

- The syntax is equation based  
[first assignment also creates this variable, see Values/Variables[3]],  
e.g.  
`a = 1;`
- End of an equation is marked by semicolon, this can be used use multiple equations in a single formula, e.g.  
`a = 1; b = 2; c = a * b;`
- Comparisons can be used to determine boolean values, e.g.  
`boolResult = a == 3;`
- Linebreaks can be inserted to structure the formula
- Comments can be added by using C/C++ syntax, e.g.  
`// rest of line is a comment`  
`/* comment until the comment tags are closed */`
- C-Function style syntax are used to access values from the database,  
e.g.  
`Sig(„CorrectPathToSignalInDatabase“) = 42;`
- Formula variables do not need any declaration

# Values and Variables (1)

## Path (Database String reference)

CanEasy style unique path to element in database

Can be determined by copying database element or by drag&drop

Type of DB element	Path example
CAN Signal	//DB/Channel:Channel1/Node:ECU/Msg:Message3/Sig:SignalH
CANMessage	//DB/Channel:Channel1/Node:ECU/Msg:Message1
LIN Signal	//DB/Channel:Channel1/Node:ECU/UFrame:Frame/Sig:Signal
Environment Variable	//DB/Channel:Channel1/Node:ECU/Env:Variables/FormulaResult

## Short names for path identifiers

Short name	Long name
//DB/Channel1/ECU/Message3/SignalH or Sig:SignalH(*)	//DB/Channel:Channel1/Node:ECU/Msg:Message3/Sig:SignalH
//DB/Channel1/ECU/Message1 or Msg:Message1 (*)	//DB/Channel:Channel1/Node:ECU/Msg:Message1
//DB/Channel1/ECU/Variables/Formul aResult	//DB/Channel:Channel1/Node:ECU/Env:Variables/FormulaResult

[\*] Respective name must be unique in database if the shortest variant is used

# Values and Variables (2)

## Accessing values (getter/setter)

Type of value	Code to use in Plugin
Physical value of a signal	<b>Sig</b> ("<PATH>") Value range according to signal configuration.
Single Byte of a message	<b>MsgByte</b> ("<PATH>", <byte position>) Single byte of message [0..7], 8-Bit value range.
Environment variable	<b>UV</b> ("<PATH>")

<PATH>: See prev. page for details/information

---

# Values and Variables (3)

## Formula variables

- No declaration of variables, first assignment creates variable
- Variables are type less
- Defined variables are shared between all formulas  
[e.g. calculate a value in one formula and use it in another]
- Variables can be used as temporary storage or to structure complex formulas
- Variables keep their value regardless of simulation start/stop
- Add timing elements to formulas  
[e.g. increase variables on each evaluation]
- Type casts available:  
bool / int / double  
e.g. a = bool [n];

### E.g. structuring:

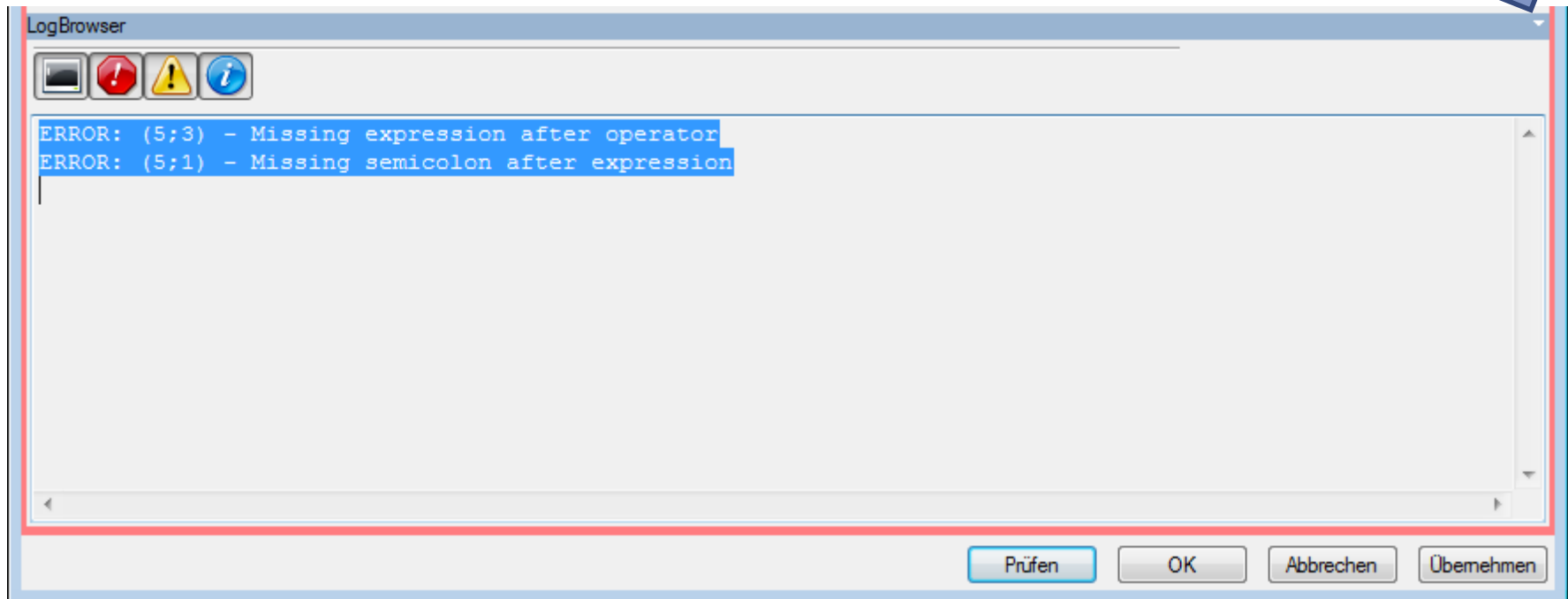
```
TRes = UV(„Resolution“) * 2.45;  
Result = Sig(„Signal1“) /  
    TRes;
```

### E.g. timing/half sinus:

```
T = (T + 1) % 100;  
Sig(„Signal1“) = sin(T/100)*255;
```

- Prints output from the parser behind the formula to allow the user better error analysis of complex formulas.
- Three categories: errors, warnings and information.

Open LogBrowser



# Examples (1)

## Easy checksums:

- Variant 1 [few/specific signals]:

```
Sig("Sig:MsgChkSum") =  
Sig("Sig:SignalA") XOR Sig("Sig:SignalB");
```

- Variant 2 [full message bytes]:

```
Sig("Sig:MsgChkSum") =  
MsgByte("Msg:Message3", 0) XOR  
MsgByte("Msg:Message3", 1) XOR  
MsgByte("Msg:Message3", 2) XOR  
MsgByte("Msg:Message3", 3) XOR  
MsgByte("Msg:Message3", 4) XOR  
MsgByte("Msg:Message3", 5) XOR  
MsgByte("Msg:Message3", 6);
```

# Examples (2)

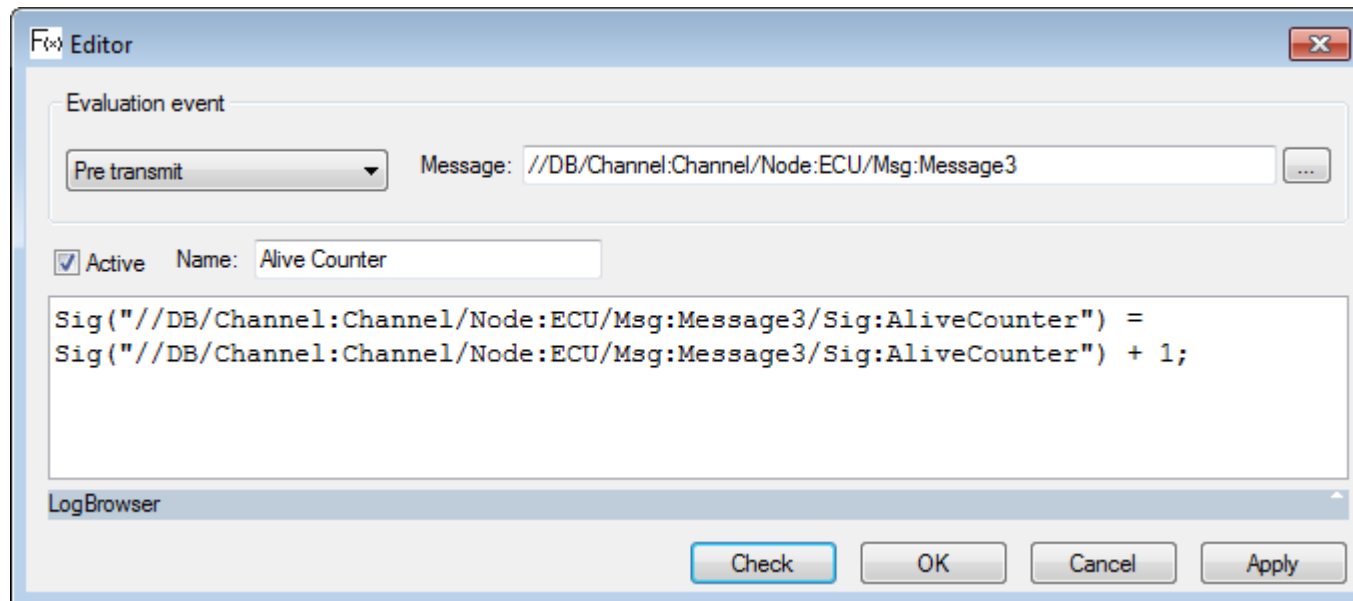
## Complex formulas with Temporary variables:

```
Part1 =  
(((((((( (MsgByte("Msg:Message3", 0) +  
  MsgByte("Msg:Message3", 1)) % 255) +  
  MsgByte("Msg:Message3", 2)) % 255) +  
  MsgByte("Msg:Message3", 3)) % 255) +  
  MsgByte("Msg:Message3", 4)) % 255) +  
  MsgByte("Msg:Message3", 5)) % 255) +  
  MsgByte("Msg:Message3", 6)) % 255);  
  
Part2 =  
(MsgByte("Msg:Message3", 0) +  
MsgByte("Msg:Message3", 1) +  
MsgByte("Msg:Message3", 2) +  
MsgByte("Msg:Message3", 3) +  
MsgByte("Msg:Message3", 4) +  
MsgByte("Msg:Message3", 5) +  
MsgByte("Msg:Message3", 6)) % 255;  
  
Sig("Sig:MsgChkSum") = (Part1 + Part2) % 255;
```

# Examples (3)

## Alive counters:

```
Sig("//DB/Channel:Channel/Node:ECU/Msg:Message3/Sig:AliveCounter") =  
Sig("//DB/Channel:Channel/Node:ECU/Msg:Message3/Sig:AliveCounter") + 1;
```

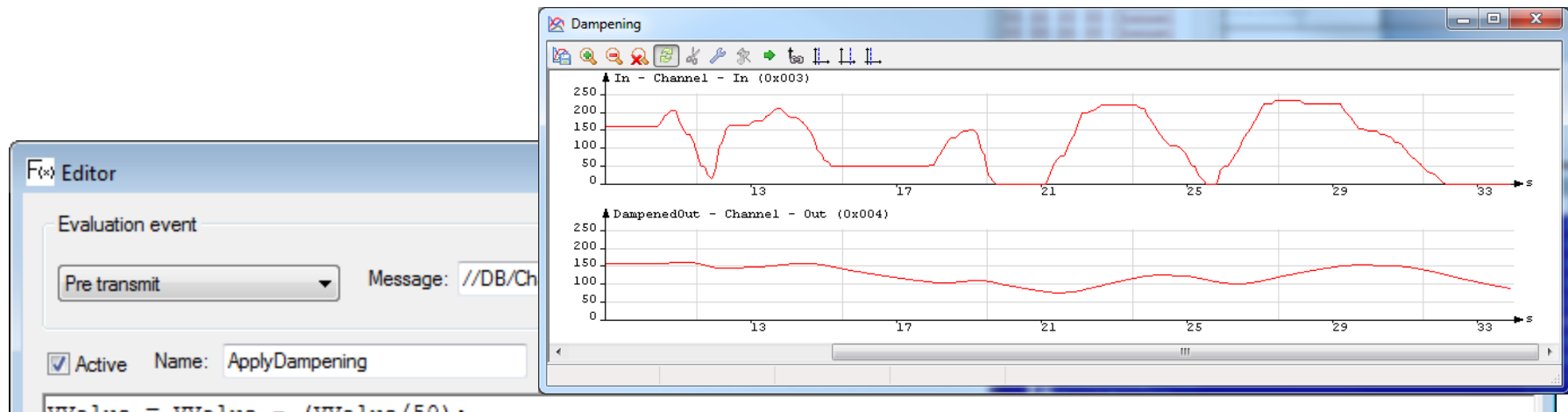


# Examples [4]

Simple dampening based on cycle time of sent signal:

```
DampeningTimer = 50;  
VValue = VValue - (Vvalue / DampeningTimer);  
VValue = VValue +  
    Sig("//DB/Channel:Channel/Node:PTS/Msg:In/Sig:In");  
Sig("//DB/Channel:Channel/Node:PTS/Msg:Out/Sig:DampenedOut") =  
    VValue / DampeningTimer;
```

DampeningTimer: iterations of formula evaluation needed to reach target value,  
e.g. cyclic time 100ms => 100ms \* 50 = 5 seconds



# Examples (5)

## Rising edge detection for signals:

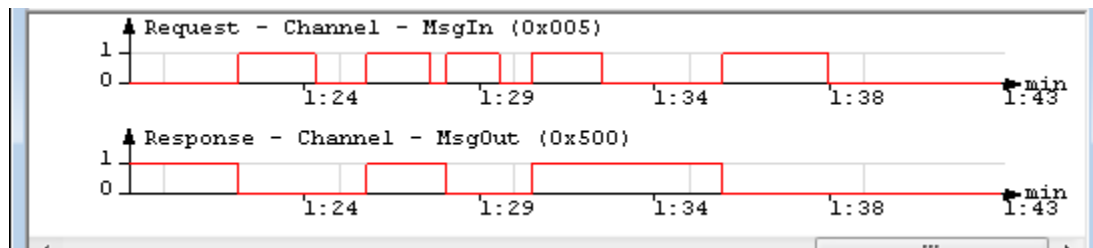
```
InputCur =
```

```
  Sig("//DB/Channel:Channel/Node:S2/Msg:MsgIn/Sig:Request");
```

```
RisingEdge = InputOld == 0 && InputCur == 1;
```

```
Sig("//DB/Channel:Channel/Node:S2/Msg:MsgOut/Sig:Response") =  
  Sig("//DB/Channel:Channel/Node:S2/Msg:MsgOut/Sig:Response") +  
  RisingEdge;
```

```
InputOld = InputCur;
```



- Not every logic needs to be coded
- Be careful with the evaluation order  
[e.g. mismatch of order for Alive counter /  
Checksum]
- Variables are shared between all formulas, so use  
meaningful and unique names if they should not be  
shared

Thank you for your attention!

---