bmcm

bmc messsysteme gmbh

NextView®4 Script

Event-oriented Scripting Language
for NextView®4

Programming Guide

Version 4.6

www.bmcm.de

bavarian measurement company munich

CONTENTS

1 OVERVIEW

1.1 INTRODUCTION

NextView®4 Script is an easy-to-use scripting language to implement specific measurement applications in combination with the professional data acquisition and processing software **NextView®4**. Mainly based on the programming language BASIC, **NextView®4 Script**, however, has been extended by specific commands and functions perfectly complementing the default commands and features of **NextView®4**.

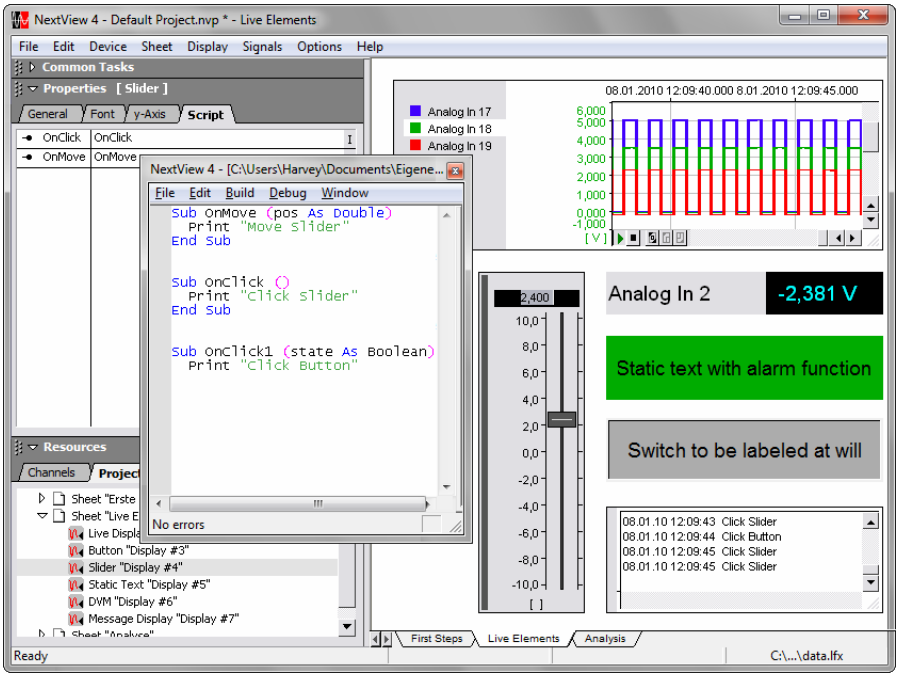

Figure 1

Customized measurement tasks can be realized directly with **NextView®4 Script** by simple programming. Processes are automated and controlled.

The optionally available add-on module (subject to charge) **NextView®4 Script** makes the software **NextView®4** even more multifunctional and powerful without loosing sight of the main feature of **NextView®4**: the user-friendliness. Applications can easily be implemented without any long training periods by using the popular and easy-to-learn programming language BASIC as a basis for **NextView®4 Script**.

This manual describes the structure of a **NextView®4 Script** program and explains the most important commands placing emphasis on the specific features and functions provided by the scripting language **NextView®4 Script**. It is not a reference for the programming language BASIC. To learn BASIC programming, a programming guide for BASIC should definitely be consulted.

Individual example scripts for bmcm measurement hardware are provided in the download area of the bmcm website.

1.2 BMC MESSSYSTEME GMBH

BMC Messsysteme GmbH stands for innovative measuring technology made in Germany. We provide all components required for the measuring chain, from sensor to software.

Our hardware and software components are perfectly tuned with each other to produce an extremely user-friendly integrated system. We put great emphasis on observing current industrial standards, which facilitate the interaction of many components.

Products by BMC Messsysteme are applied in industrial large-scale enterprises, in research and development and in private applications. We produce in compliance with ISO-9000-standards because standards and reliability are of paramount importance to us - for your profit and success.

Please visit us on the web (http://www.bmcm.de/) for detailed information and latest news.

www.bmcm.de
bavarian measurement company munich

1.3 COPYRIGHTS

The programming language **NextView®4 Script** has been developed and tested with utmost care. BMC Messsysteme GmbH does not provide any guarantee in respect of this manual, the hardware and software described in it, its quality, its performance or fitness for a particular purpose. BMC Messsysteme GmbH is not liable in any case for direct or indirect damages or consequential damages, which may arise from improper operation or any faults whatsoever of the system. The system is subject to changes and alterations which serve the purpose of technical improvement.

Updated: 09/20/2014

BMC Messsysteme GmbH

Hauptstrasse 21
82216 Maisach
GERMANY

Phone: +49 8141/404180-1
Fax: +49 8141/404180-9
E-mail: info@bmcm.de

2 BASICS OF PROGRAMMING

2.1 GENERAL

2.1.1 FILE TYPE *.NVS

A program created with **NextView®4 Script** is saved as a script file with the file format **`*.nvs`**. It contains both a declaration part listing and naming all data types as well as an application part containing the used commands, procedures, and functions.

The code written in a script does not necessarily have to relate to a single application, but can also be used in different applications if written in a general way.

2.1.2 INVALID CHARACTERS

The following characters are not permitted in source code:

- umlauts (special German characters: ä, ö, ü)
- special characters (e.g. @, §, ...)
- language-specific characters (e.g. ß)

2.1.3 COMMENTS

Comments are used for a better understanding of source code. Everything written after the character **`'`**, will be ignored when the program is running.

Comments can be placed after a programming command (but not after a word-wrap!) or take a complete line. If they need more than one line, the character for comments must be placed at the beginning of each line.

```
' example of a comment using
' several lines

Print "Hello World!" ' comment example after a command
```

2.1.4 WORD-WRAPPING

To improve the clear arrangement of a program, word-wrapping is recommended for long program lines sometimes. In this case an underline character with a preceding space is added at the end of the line to be wrapped: (_)

```
Print "The area is radius * radius * Pi " _
      & radius * radius * 3.141592653
```

Comments are not permitted after a word-wrap!

2.1.5 USING NON-DECIMAL NUMERAL SYSTEMS

In contrast to decimal numbers, hexadecimal numbers are written with the prefix **&H**. In the same way, **&O** is added to numbers of the octal system, **&B** to binary numbers. The prefix **&D** can also be used for the decimal system but is not mandatory.

```
Dim i As Integer
i = &H10       ' set i to 10hex (decimal system: 16)
```

2.1.6 NAME CONVENTIONS

When naming variables, constants, programs, etc., the following name conventions must be regarded:

- uniqueness
- 1. character must be a letter
- max. 256 characters
- no use of spaces, dots, or other special characters (exception: underline _), especially short forms of variable types

2.2 DATA TYPE

Processing data of the most different type is an important task of programs. Data types define the properties of the values and their range. The following data types are available in **NextView®4 Script**:

Data type	Range	Notes
Byte	0..255	only integers within the range
Integer	-2147483648 .. 2147483647	only integers within the range
Double	$-1.79769313486232*10^{308}$ .. $1.79769313486232*10^{308}$	floating point number with 10-digit accuracy
String	all characters of the ASCII code	string (upper-case/lower-case letters, numerals, special characters, etc.)
Boolean	True, False	indicates the state of Boolean expressions or variables
Classes		class (e.g. channel)
Array		values of the same data type with indexing

Values of the *Double* type are written exponentially as follows:

<mantissa>D<exponent> or **<mantissa>d<exponent>** or
<mantissa>E<exponent> or **<mantissa>e<exponent**

The exponent always relates to the basis 10. Also note the sign of the exponent:

45D+3 = 45*10^3 = 45000 or **45D-3 = 45*10^{-3} = 0.045**

Data types are classified as simple data types, classes, and arrays. Simple data types are *byte*, *integer*, *double*, *string*, and *Boolean*.

Windows® date/time format:

A Windows® date/time information is saved as a double floating point number. The number of days since 12/30/1899 stands before the decimal point. The value after the decimal point is the fraction of the day (e.g. 0.5 is 12:00am).

2.3 VARIABLES

Variables consist of the following elements: variable name, value, and data type. When assigning a variable name, it must comply with the name conventions. Unlike constants, the value of a variable may change during a program run for variables to be used for saving interim values.

2.3.1 GLOBAL AND LOCAL VARIABLES

A difference must be made between global and local variables: Global variables are defined in the declaration part of the program and are always valid. Local variables are declared within a procedure and only exist while the procedure is processed.

If the same name is used for a global and a local variable, always the local variable is valid in the procedure the local variable has been defined for (see "Validity areas", p. 50).

2.3.2 VARIABLE DECLARATION

It is always necessary to explicitly declare variables with the key word **Dim** at the beginning of each line.

```
Dim Variablename As Datatype
```


If several variables are declared in one line, they must be separated from each other by a comma.

```
Dim Var1[, Var2, ...] As Datatype
```

```
Dim Number As Double
Dim i, j As Integer
```

2.3.3 ASSIGN A VALUE TO VARIABLES

The following command assigns a value to a variable. This must be within the defined variable range.

```
Variable_name = Variable_value
```

The expression on the right can also contain a variable.

```
Number = 23
Number = Number + 1
```

It is possible to assign a string to a numerical variable if it is a numerical value. The string is automatically converted into a number (see "Conversion of expressions", p. 31). Definitions of this kind, however, should be an exception as they are prone to errors.

```
Dim s As String
Dim i As Integer

s = 123
i = s + 1
```

2.4 CONSTANTS

Constant declarations consist of a name, a data type, and a constant value, which is already known at the beginning of the program. In contrast to variables, the value of constants cannot be changed during the program run. When choosing a name, the name conventions must be observed.

The following example shows how constants are declared. The value assigned on the right must be within the range of the used data type.

```
Const Constant_name As Datatype = Value
```

Only simple data types (*byte*, *integer*, *double*, *string*, *Boolean*) can be assigned. Constant *arrays* or *classes* are not possible.

2.5 ARRAYS

Unlike simple data types, such as *integer* or *string*, arrays are used to store several values of the same data type.

Access is possible with the array name and an index allowing for the elements of an array to be distinguishable from each other.

The great advantage of arrays is that the source code can considerably be simplified for repeating processes. If using loops, for example, arrays with large amounts of data can be processed only by changing the index number.

Arrays have an upper and a lower bound and the elements of an array have a defined position. The index always starts at the lower bound with 0 and is incremented by 1 if not defined otherwise.

2.5.1 ARRAY DECLARATION

The declaration of arrays always starts with the key word **`Dim`** and the array name followed by the highest index value in round brackets and the used data type:

```
Dim Arrayname(upper_index) As Datatype
```
or

```
Dim Arrayname(upper_index to lower_index) As Datatype
```

The array data type can only be a simple data type (*byte*, *integer*, *double*, *string*, *Boolean*).

2.5.2 ACCESS TO ELEMENTS OF AN ARRAY

An array element gets a value via the array name and the specific index. The assigned value must be within the range of the data type used.

```
Arrayname(Index) = Value
```

If a variable is used instead of the index, assigning values for arrays can be simplified considerably by using a counting loop.

The content of an array element can also be read out by writing the array name directly followed by the index in round brackets:

```
Arrayname(Index)
```

2.5.3 MULTIDIMENSIONAL ARRAYS

Sometimes elements of a (one-dimensional) array are related to elements of other arrays. To present this additional information, a multidimensional array is recommended.

Those are defined by means of multiple indexes. It must be pointed out, however, that the higher the dimension of the array, the more complex und confusing the structure, and the higher the memory capacity required.

```
Dim Arrayname(Index 1, Index 2, .. , Index n) As Datatype
```

The index of an array always starts with 0 and ends with the value set in the **`Dim`** definition. For example, **`Dim Array(3) as Integer`** generates an array with four integer elements.

Multidimensional arrays can efficiently be processed with nested counting loops. The following example initializes a two-dimensional array of the size 5x4 with the products of the index values:

```
Dim i, j As Integer
Dim Array(4, 3) As Integer

For i  = 0 to 4
    For j = 0 to 3
        Array(i, j) = i * j
    Next
Next
```

i j	0	1	2	3	4
0	(0, 0) 0	(1, 0) 0	(2, 0) 0	(3, 0) 0	(4, 0) 0
1	(0, 1) 0	(1, 1) 1	(2, 1) 2	(3, 1) 3	(4, 1) 4
2	(0, 2) 0	(1, 2) 2	(2, 2) 4	(3, 2) 6	(4, 2) 8
3	(0, 3) 0	(1, 3) 3	(2, 3) 6	(3, 3) 9	(4, 3) 12

2.5.4 DYNAMIC ARRAYS

The arrays provided by **NextView®4 Script** are always dynamic, i.e. their size is variable during the program run allowing for great flexibility. By permanently adjusting the array size, a large array, for example, can be used temporarily and be resized as soon as data are not needed anymore to save lots of memory space.

The declaration of a dynamic array is done with the key word **`Dim`** without specifying the bounds. Those are defined with the command **`ReDim`** later on, but without declaration of the data type.

```
Dim Arrayname() As Datatype
...
ReDim Arrayname(Index 1, Index 2, .. , Index n)
```

When adjusting the array size, all values stored in the array will be lost.

After the **`ReDim`** command has been executed, the elements of an array are reset to the values (depending on the data type) shown in the table below. This is an advantage, when new data are assigned to an array or memory space is to be saved.

Data type	New value
Byte	0
Integer	0
Double	0
String	""
Boolean	False
Classes	Nothing

2.5.5 COMMANDS CONCERNING ARRAYS

The following commands are frequently used with arrays:

Command	Description
`LBound(Arrayname)`	returns the index of the lower bound
`LBound(Arrayname, n)`	for multidimensional arrays: returns the index of the lower bound of the n-th dimension
`UBound(Arrayname)`	returns the index of the upper bound
`UBound(Arrayname, n)`	for multidimensional arrays: returns the index of the upper bound of the n-th dimension

2.6 EXPRESSIONS AND OPERATORS

Expressions are an important component of a script language. The easiest way to describe an expression would be: "everything that has a value". Examples for expressions are the number **5**, the addition **5+37**, or a function call such as **Rnd** to receive a random number. The value of an expression can be assigned to variables or be passed to procedures or functions .

```
Dim i As Integer

i = 5
i = i + 37
Print "The answer is " & i
Print "A random number is " & Rnd
```

Operators connect expressions to a new one. The **+** symbol in **5 + 37**, for example, is an operator combining the integer expressions **5** and **37** to a new integer expression.

2.6.1 OPERATOR PRECEDENCE

The following table lists the available operators beginning with the strongest binding operator. Expressions connected with a stronger binding operator are executed first. Operators in the same line have the same priority. Operations with equal priority are executed from left to right.

Symbol	Description
^	exponentiation
-	arithmetic negation
* /	multiplication, division
\	whole-number division
mod	modulo division
+ -	addition, subtraction

Symbol	Description
&	string connection
= <> < > <= >=	comparison
Not	logic and binary negation
And	logic and binary And
Or	logic and binary Or
Xor	logic and binary Xor

To influence the operator precedence, expressions can be put in round brackets. Brackets bind more than any operator. For example, the bracket in the expression **(2+3)*4** overrides the rule that multiplication goes before addition and the result is **20**.

2.6.2 ARITHMETIC OPERATORS

Arithmetic operators are exponentiation (**^**), arithmetic negation (-), addition (**+**), subtraction (-), multiplication (*****), division(**/**), whole-number division (\), and modulo division (**mod**). They connect and return numeric values (*Byte*, *integer*, *double*).

```
Dim i As Integer
Dim d As Double

i = 5 mod 3         ' the result is 2
Print i * i         ' returns 4
d = 1.0 / 2.0       ' the result is 0.5
Print - d ^ 2       ' returns -0.25
```

2.6.3 LOGICAL OPERATORS

Logical operators are negation (**Not**), conjunction (**And**), disjunction (**Or**), and the exclusive disjunction (**Xor**). They use values of the *Boolean* type and return a *Boolean* value. The so-called "truth tables" show the result of a logic operation:

x	y	Not x	Not y	x And y	x Or y	x Xor y
True	True	False	False	True	True	True
True	False	False	True	False	True	False
False	True	True	False	False	True	False
False	False	True	True	False	False	True

```
Dim b As Boolean

b = True Xor False ' the result is True
```

2.6.4 RELATIONAL OPERATORS

Relational operators compare values of the same data type and return a *Boolean* value. If the combination makes sense, all data types can be used. *Arrays* must have the same dimension. Numerical values (*Byte*, *integer*, *double*) being compared can be of different type also (see "CONVERSION OF EXPRESSIONS", p. 31).

Symbol	Description
=	equal to
<>	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

2.6.5 BITWISE OPERATORS

Bitwise operators are negation (**`Not`**), conjunction (**`And`**), disjunction (**`Or`**), and the exclusive disjunction (**`Xor`**). They take values of the *integer* type (0, 1) and return an *integer* value. So-called "bit tables" show the result of a bitwise operation.

x	y	Not x	Not y	x And y	x Or y	x Xor y
1	1	0	0	1	1	1
1	0	0	1	0	1	0
0	1	1	0	0	1	0
0	0	1	1	0	0	1

```
Dim i As Integer

i = &B01111111 And &B11110111 ' the result is &B01110111
i = &H7f And &Hf7             ' the result is &H77
```

2.6.6 STRING OPERATORS

The operator **&** connects strings and returns a continuous *string* value. Strings which have not been assigned to a variable before must be written with quotation marks.

```
Dim s As String

s = "Next"
s = s & "View"   ' the result is "NextView"
```

2.6.7 CONVERSION OF EXPRESSIONS

NextView®4 Script features an automatic expression conversion of simple data types. Without an additional conversion command being necessary in the program, a string consisting only of digits is recognized as a numerical value and vice versa.

The following example automatically converts the value of the integer variable **i** into a string and connects the two strings with **&**:

```
Dim i As Integer

i = 13
Print "i is " & i      ' the output is: i is 13
```

Conversely, two strings are converted into numerical values and multiplied with each other:

```
Print "14" * "1.5"      ' the output is: 21
```

2.7 SUBROUTINES

Using functions and procedures reduces the programming effort and improve the clarity and structuring. A process used several times can simply be realized by a call of a function or procedure and does not have to be entered repeatedly as source code.

NextView®4 Script provides predefined subroutines like, for example, the procedure **`Print`** frequently used in the examples or the function **`Rnd`**, which can be integrated directly in the program without previous declaration (see "STANDARD FUNCTIONS / PROCEDURES", p. 153).

Subroutines are classified as procedures (**`Sub`**) executing command sequences or functions (**`Function`**) providing a return value.

```
Print "Asterix"     ' example of a procedure
                    ' Print has no return value

Dim i, j As Integer ' example of a function
i = Rnd             ' Rnd returns a random number as
                    ' return value
```

2.7.1 ARGUMENTS

The information for a subroutine is passed by means of arguments. A string is passed to the procedure **Print** as an argument, for example, and **Print** passes it to the standard output (e.g. printer). Subroutines can also have optional arguments, i.e. arguments can but do not have to be entered. An index value can be passed to the procedure **UBound**, for example. If the index position is not specified, always the first value of the array is used.

```
Print "Asterix"          ' the output is: Asterix

Dim i(5,15) As Integer

Print UBound (i)         ' the output is: 5
Print UBound (i,1)       ' the output is: 5
Print UBound (i,2)       ' the output is: 15
```

2.7.2 PROCEDURES

A procedure is defined as follows:

```
Sub Procedurename ([arg1 As Datatype1, arg2 As Datatype2, ...])
  [commands]
End Sub
```

If the procedure does not require arguments, there is an empty (round) bracket in the declaration part after the procedure name.

When calling a procedure in the script, the name of the procedure and then the arguments to be used are entered, if necessary. Several arguments are separated from each other by a comma.

```
Procedurename
```

or

```
Procedurename [expression1, expression2, ...]
```

Some examples:

```
Sub ShowName ()
  Print "Asterix"
End Sub

Sub Mult (a As Double, b As Double)
  Print a * b
End Sub
...
Sub Test ()
  ShowName           ' the output is: Asterix
  Mult 3, 1.5        ' the output is: 4.5
End Sub
```

To drop out of a procedure, the command **`Exit Sub`** is used.

```
Sub Div (a As Double, b As Double)
  If b = 0 Then Exit Sub Else Print a / b
End Sub
```

2.7.3 FUNCTIONS

A function is defined as follows:

```
Function Functionname ([arg1 As Datatyp1, _
                        arg2 As Datatyp2, ...]) As Datatype
  [commands]
End Function
```

The function definition must always assign the result of the function (return value) to the function name.

```
Functionname = Functionresult
```

When calling a function in the script, the function name is used. The arguments to be passed are parenthesized **( )** after the function name. The return value of a function, resulting from calling the function name, is either assigned to a variable or processed as an expression.

```
Variable = Functionname       or

Variable = Functionname (expression1, expression2, ...)
```

```
Function Pi () As Double  ' function declaration
  Pi = 3.1415926535       ' assign return value
End Function

Function Mult (a As Integer, b As Integer) As Integer
  Mult = a * b            ' assign return value
End Function
...
Sub Test ()
Dim i as Double
  i = Pi                  ' i is set to: 3.1415926535
  Print Mult(3, 2)        ' the output is: 6
  Print 1 + Mult(3, i)    ' the output is: 10.4247779605
End Sub
```

To drop out of a function, the command **`Exit Function`** is used.

```
Function Div (a As Integer, b As Integer) As Integer
  Div = 1
  If b = 0 Then Exit Function Else Div = a / b
End Function
```

2.7.4 OPTIONAL ARGUMENTS

If optional arguments are assigned to a subroutine, the key word **`Optional`** is used in the declaration part of the subroutine before the argument and the assignment of the default value. They are used if no other argument has been used when calling the function/procedure. All arguments following the first optional argument are optional, too.

```
Optional argx As Datatypex = Defaultvalue
```

```
Sub Name (Optional s As String = "Asterix")
  Print s
End Sub

Function Mult (a As Integer, Optional _
               b As Integer = 1) As Integer
  Mult = a * b
End Function

Sub Test ()
  Name                ' the output is: Asterix
  Name "Obelix"       ' the output is: Obelix
  Print Mult(3)       ' the output is: 3
  Print Mult(3, 2)    ' the output is: 6
End Sub
```

2.7.5 BYVAL AND BYREF

The key words **ByVal** and **ByRef** define how an argument is passed to a subroutine.

ByVal argx As Datatypex or

ByRef argy As Datatypey

If the standard method **ByVal** is used, a copy of the expression is passed to the subroutine. If the subroutine changes the value, the value of the copy is changed, but the original stays the same.

If a value is changed and the **ByRef** method used, however, the original is changed, too. Therefore, only variables can be passed to a subroutine with the **ByRef** method but no constants, for example.

```
Sub Name1 (ByVal s As String)
  s = "Asterix"
End Sub

Sub Name2 (ByRef s As String)
  s = "Asterix"
End Sub

Sub Test ()
  Dim s As String

  s = "Obelix"
  Name1 "Obelix"       ' correct
  Name1 s
  Print s              ' the output is: Obelix

  s = "Obelix"
' Name2 "Obelix"         WOULD NOT BE CORRECT!
  Name2 s
  Print s              ' the output is: Asterix
End Sub
```

2.8 CONTROL STRUCTURES

In addition to the declaration part, every **NextView®4 Script** consists of a number of commands. This can be an assignment, a procedure call, or a control structure such as a loop or a conditional statement, for example.

2.8.1 CONDITIONAL STATEMENTS

Conditional statements verify a *Boolean* expression. If the condition is true, certain parts of the script will be processed.

2.8.1.1 IF ... THEN ... ELSE

An **`If ... Then`** structure is used to execute one or more commands if the condition placed after the **`If`** key word is fulfilled. Both single-line syntax and multiline notation are possible. The single-line type is only used if only one command is to be executed.

- single-line syntax: `If condition Then command`
- multiline syntax:

```
If condition1 Then
  [commands]

  ...

End If
```

Please note the final **`End If`** when writing in multiline syntax to signalize the end of the conditional statements.

```
' single-line syntax
If value < 10 Then Print "Less then 10"

' multiline syntax
If value < 10 Then
  Print "Less then 10 "
  value = value + 1        ' increment value by one
End If
```

To check other conditions if the **`if`** expression is not true, **`Else If`** is used for every additional query. If neither the **`If`** nor the **`Else If`** conditions are valid, the alternative commands are stated with the key word **`Else`**.

- single-line syntax: `If condition Then command Else command`
- multiline syntax:

```
If condition1 Then
  [commands1]

[Else If condition2 Then
  [commands2]]

  ...
[Else
  [commandsN]]

End If
```

That means, condition1 is checked first and - if true - the commands1 are executed, or condition2 is checked, etc. If no condition is fulfilled, the commandsN of the **`Else`** block are processed if defined.

```
' single-line syntax
If value = 0 Then Print "Zero" Else Print "Non-zero"
' multiline syntax
If value = 0 Then
  Print "Zero"
Else If value = 10 Then
  Print " 10 "
Else
  Print "Not Zero or 10"
End If
```

Nesting of multiple **`if`** statements is allowed. However, this can have a negative effect on the clarity and may lead to errors.

```
' multiline syntax
If value > 0 Then
  If value > 0 und value <10 then
    Print "positive value is single-digit"
  Else
    Print "positive value is multi-digit"
  End If
Else
  Print "invalid value"
End If
```

2.8.1.2 SELECT CASE

`Select Case` is suitable to compare several expressions and execute different commands. Compared to the **`If ... Then ... Else`** statement, **`Select Case`** is often clearer.

```
Select Case expression
Case list_of_expressions1
  [commands1]
[Case list_of_expressions2
  [commands2]]
...
[Case Else
  [commandsN]]
End Select
```

The **`If ... Then ... Else`** example could also be written as follows:

```
Select Case value
Case 0, 1
  Print "Zero or 1"
Case 10
  Print "10"
Case Else
  Print "Non-zero"
End Select
```

With the key word **Case**, it is ensured whether the specified value corresponds to the expression defined with **Select Case**.

2.8.2 LOOPS

Loop structures are used to repeat commands and program parts.

2.8.2.1 DO ... LOOP

The loop **Do ... Loop** repeats the embedded statements as long as an expression is true (**While** or **Until**). The condition is placed either before (after the key word **Do**) or behind (after the key word **Loop**) the program instructions. Depending on the key word used, subtle but significant differences may result when running the program.

Before the loop:

- with **While**:

```
Do While expression
  [commands]

Loop
```

- with **Until**:

```
Do Until expression
  [commands]

Loop
```

After the loop:

- with **While**:

```
Do
  [commands]

Loop While expression
```

- with **Until**:

```
Do
  [commands]

Loop Until expression
```

If the expression is in front, it is verified first if the condition is fulfilled. If it is not true the first time, the commands in the loop will be skipped. If the condition is placed at the end of the loop, the statements of the loop are always executed at least once. When programming the expression, it should be ensured that it becomes invalid (**While**) or valid (**Until**) sometime to avoid an infinite loop.

The following example shows how important it is to choose the suitable notation.

```
Do
  j = 1 / i
  Print j
  i = i - 1
Loop While i <> 0
```

If the variable **i** is **0** when reaching the loop **Do ... Loop**, there could be a division by **0**. A negative **i** would result in an infinite loop. Correct would be the following loop:

```
Do While i > 0
  j = 1 / i
  Print j
  i = i - 1
Loop
```

2.8.2.2 WHILE ... WEND

The **While ... Wend** statement corresponds to the **Do ... Loop** variant mentioned before.

- **While ... Wend**:

```
While expression
  [commands]

Wend
```

2.8.2.3 FOR ... NEXT

If it is clear how often a loop is to be run through, the **`For ... Next`** statement is recommended. The advantage is that infinite loops can be avoided. This type uses an integer variable named **`Counter`**, which is automatically increased by the specified increment or reduced (negative increment). If an increment has not been defined, the start value is incremented continuously by one for each run until the final value has been reached.

```
For Counter = Start To End [Step Increment]
[statements]
Next
```

`Counter` is a global or local integer variable, **`Start`**, **`End`**, and **`Increment`** are integer constants.

```
Dim i As Integer

For i = 0 To 42
  Print i                    ' output: 0 1 2 3 or till 42
Next

For i = 100 To 0 Step -2
  Print i                    ' output: 100 98 96 or till 0
Next
```

2.8.2.4 REPEAT, EXIT

To leave a loop or rerun a loop at an earlier point in time, the key words **`Exit`** and **`Repeat`** are provided.

- **`Do ... Loop`**:

```
Do [While | Until] expression
  [Repeat]

  [Exit Do]

Loop
```

- **`Do ... Loop`**:

```
Do
  [Repeat]

  [Exit Do]

Loop [While | Until] expression
```

- **`For ... Next`**:

```
For Counter = Start To End [Step Increment]
  [Exit For]

Next
```

A **`While ... Wend`** loop does not feature this possibility. Only the **`Exit For`** statement is provided for the **`For ... Next`** loop.

```
Dim i as Integer
Do while True
  i = InputBox ("range 0..10, -1 ends loop", _
                "enter number", 0)
  If i = -1 Then Exit Do
  ...
  If i > 10 then Repeat
  ...
Loop
```

2.9 CLASSES AND OBJECTS

NextView®4 Script operated object-oriented. Objects are generated from classes. The *class* data type is a collection of variables, procedures and functions using these variables. The procedures and functions defined in the relevant class determine which operations can be done with a generated object of this class. The access to objects is possible with reference variables pointing to an object or **`Nothing`**.

NextView®4 Script provides classes. The class "NvSheet" (represents a sheet in **NextView®4**), for example, contains functions only applying to a sheet, such as the function **`NvSheet.DisplayCount`**.

In the following example, the reference variable **`mySheet`** is pointing to an object of the class "NvSheet", which is a real existing sheet in **NextView®4**. The

command **mySheet.DisplayCount** returns the number of displays on the sheet.

```
Dim mySheet as NvSheet
Set mySheet = NvCurrentProject.Sheet(1)
print mySheet.DisplayCount
```

2.9.1 CLASS DECLARATION

In addition to classes predefined in **NextView®4 Script**, individual classes can be defined, too. The declaration of a class is done with the key word **Class**.

```
Class MyClass
  ...
End Class
```

The variables, procedures and functions available to an object are defined in a class declaration.

2.9.1.1 VARIABLES

The declaration of class variables is done with the key word **Dim** in the same way as the general variable declaration. It must be placed within the class declaration, however.

```
Class MyClass
  Dim i As Integer
End Class
```

2.9.1.2 PROCEDURES AND FUNCTIONS

The procedures and functions available in a class are defined with the key words **Sub** and **Function** within the class declaration following the common conventions.

```
Class MyClass
  Sub MySub ()
    ...
  End Sub
  Function MyFunction () As Integer
    ...
  End Function
End Class
```

2.9.1.3 ME

There is always a reference variable (see "REFERENCE VARIABLES", p. 46) in a class pointing to the own class. It does not have to be defined explicitly and is called with the key word **Me**.

```
Class MyClass
  Dim i As Integer

  Sub MyOtherSub ()
    Me.i = 42
  End Sub

  Sub MySub ()
    Me.MyOtherSub
  End Sub
End Class
```

With the **Me** reference variable, it is possible to access variables, procedures, and functions of a class. The following example demonstrates how to proceed.

```
Class MyClass
  Dim i As Integer                ' i as class variable
  ...
  Sub MySub (i As Integer)        ' i as local variable
    Print "Your answer is " & i   ' uses local i
    Print "My answer is " & Me.i  ' uses class variable i
  End Sub
End Class
```

2.9.2 WORKING WITH OBJECTS

2.9.2.1 REFERENCE VARIABLES

A reference variable is either related to a newly created object (see "CREATE OBJECTS", p. 46), to an object returned by a function (see "OBJECTS AS RETURN VALUE OF FUNCTIONS", p. 47), or to no object at all (**Nothing**, see p. 47). It is declared with the key word **Dim**.

```
Dim Referencevariable As class
```

The value of a reference variable after declaration is always **Nothing**. To assign an object to a reference variable, the command **Set** is used.

```
Dim cha As NvChannel        ' AnalogIn is a function
Set cha = AnalogIn (5)      ' predefined in NextView Script
```

2.9.2.2 CREATE OBJECTS

Objects of self-defined classes are created with the key word **New**.

```
Class MyClass
  Dim i As Integer

  Sub MySub ()
    Print "The answer is " & i & "."
  End Sub
End Class
...
Sub Test ()
  Dim my As MyClass
  Set my = New MyClass      ' object is created and reference
                            ' variable is set to it
  my.i = 42
  my.MySub                  ' output: The answer is 42.
End Sub
```

NextView®4 Script provides functions to create a new object for classes predefined by **NextView®4 Script** such as the class "NvDisplay". **New** must not be used in this case.

```
Dim g As NvDisplay
Set g = NvCurrentProject.Sheet(1).Display(1)
' calls the first display of the first sheet in the project
' and sets a reference variable to it
```

2.9.2.3 OBJECTS AS RETURN VALUE OF FUNCTIONS

An object can be the return value of a function. The predefined function **NvProject.FindSheet (name As String)** of the class "NvProject", for example, returns the **NextView®4** sheet named **name** in the project. **NvCurrentProject** provides the current project as a reference to a "NvProject".

```
Dim s As NvSheet
Set s = NvCurrentProject.FindSheet ("FFT")
' find sheet named FFT and set a reference variable to it
```

2.9.2.4 NOTHING

If a reference variable is not related to an object, it is set to **Nothing**. It is not possible to set a reference variable to an object with this key word or to query a reference variable with **Is Nothing** to check if it is related to a valid object.

```
Dim s As NvSheet

Set s = Nothing
If s Is Nothing Then Print "Nothing!"
```

2.9.2.5 DELETING OBJECTS

Objects are automatically deleted in **NextView®4 Script** as soon as there is no reference variable or any other existing reference pointing to an object.

```
Dim my As MyClass
Set my = New MyClass
Set my = Nothing
```

The code in the example above would create an object of the class "MyClass". It is eliminated in the next line right away as the only reference to the object is deleted.

If, however, a second reference variable points to object **my1**, it will not be deleted when set to **Nothing**, because the reference from **my2** to **my1** is still existing.

```
Dim my1, my2 As MyClass
Set my1 = New MyClass
Set my2 = my1
Set my1 = Nothing
```

2.9.3 PROGRAMMING WITH CLASSES

To get access to the properties, procedures and functions provided to an object by a class, always the object name followed by a dot is put in front.

```
Print NvCurrentProject.SheetCount

Print NvCurrentProject.Sheet(1).Name
```

If a reference variable does not point to an object, i.e. to **Nothing**, **NextView®4 Script** generates a so-called runtime error (Runtime Error 13, Nothing Referenced) displayed in the status bar of **NextView®4 Script**.

Check the validity of the reference variable in this case. Reference variables to newly created objects are always valid. If errors should occur, a runtime error would be generated already when creating the object.

Objects returned by functions can also be **`Nothing`**. The function **`NvProject.FindSheet`**, for example, returns **`Nothing`** if the demanded sheet does not exist.

```
Dim s As NvSheet

Set s = NvCurrentProject.FindSheet ("something funny")

If s Is Nothing Then
  Print "Sheet not found"
  Exit Sub
End If
```

A special case occurs if a class derives from a "parent" class. The class "NvGraphDisplay", the class "NvButton", and the class "NvSlider", for example, are special classes of the class "NvDisplay". To make sure the returned display object is a special object, the "?=" assignment operator is used. If the object is not of the type of the reference variable class, it will be set to **`Nothing`**.

```
Dim g as NvGraphDisplay
set g ?= NvCurrentProject.Sheet(1).Display(1)
' tries to assign the first display in the first sheet of
' the project to the NvGraphDisplay reference variable
' If the display is not a NvGraphDisplay, the reference
' variable will be set to Nothing
if g Is Nothing then print "no NvGraphDisplay"
```

If the usual "=" assignment operator was used in this case, there would be a runtime error already during the script compilation.

2.10 VALIDITY AREAS

If the same names are used for global and local variables (see "GLOBAL AND LOCAL VARIABLES", p. 22) in procedures, functions, or classes, the question of validity areas occurs. The following table lists the priority of variables, procedures, or functions in descending order.

Variables
return value of a function
local variable, procedure, or function arguments
class variables
global variables

Procedures / Functions
class procedures and functions
global procedures and functions

For example, local variables within functions, procedures, or classes are preceding global variables, i.e. the local variable is valid if using identical variable names.

```
Dim i As Integer          ' i as global variable (to 0)

Sub Test ()
  Dim i As Integer        ' i as local variable
  i = 42
  Print "Result: " & i    ' the output is: Result: 42
End Sub
```

If another procedure / function is called within a procedure / function declaration, the local variables of the defined procedure / function do not apply to the one called. Only the own local variables and the global variables are valid as demonstrated in the following example.

```
Dim i As Integer          ' i as global variable (to 0)

Sub Answer ()
  Print "Result: " & i    ' uses the global variable i
End Sub
Sub Test ()
  Dim i As Integer        ' i as local variable
  i = 42
  Answer                  ' the output is: Result: 0
End Class
```

Class procedures or class functions are given priority to global procedures or functions. This is shown in the following example:

```
Sub MyOtherSub ()          ' global procedure
  Print "Hello World!"
End Sub

Class MyClass
  Sub MyOtherSub ()        ' class procedure
    Print "Result: 42"
  End Sub
  Sub MySub ()
    MyOtherSub             ' the output is: Result: 42
  End Sub
End Class
```

3 SCRIPT CLASSES/ROUTINES

3.1 NV4 FUNCTIONS AND PROCEDURES

3.1.1 NV4 FUNCTIONS / PROCEDURES: OVERVIEW

Function	Description
`NvCurrentProject`	returns the current project
`NvStartScan`	starts a scan
`NvStopScan`	stops a scan
`NvScanState`	returns the current state of the scan
`NvSetTimerInterval`	sets the sampling interval for the `OnTimer` event
`NvExitProgram`	closes **NextView®4**
`NvAnalogIn`	provides the interface to an analog input channel
`NvAnalogOut`	provides the interface to an analog output channel
`NvDigital`	provides the interface to a digital channel
`NvDigitalLine`	provides the interface to a digital line
`NvCounter`	provides the interface to a counter channel
`NvFormula`	provides the interface to a formula channel

3.1.2 NVCURRENTPROJECT

```
Function NvCurrentProject () As NvProject
```

Returns the current project of the class "NvProject".

```
' prints the number of sheets of the project
Print NvCurrentProject.SheetCount
```

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.3 NVSTARTSCAN

```
Sub NvStartScan (Optional checkExisting As Boolean = True)
```

Starts a scan. This is equivalent to the "Start scan" command (menu item "Device") in **NextView®4**. If **`checkExisting`** is "False", any existing measurement files of the same name will be overwritten without further inquiry.

```
' starts a scan with inquiry
NvStartScan

' starts a scan without inquiry
NvStartScan False
```

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.4 NVSTOPSCAN

```
Sub NvStopScan ()
```

Stops a scan. This is equivalent to the "Stop scan" command (menu item "Device") in **NextView®4**.

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.5 NVSCANSTATE

```
Function NvScanState () as Integer
```

Returns the scan state of **NextView®4**. The following predefined constants are provided:

Scan state	Description
scanStatePreparing	data storage is prepared
scanStateBeforeTrigger	data are stored, trigger condition not fulfilled yet
scanStateAfterTrigger	data are stored, trigger condition fulfilled
scanStateBusy	scan system is busy after the last data storage
scanStateRunning	live data are scanned
scanStateReady	scan system is ready
scanStateUnknown	scan state is unknown

Example: Output of the scan state in a static text display called "Scan state":

```
...
Dim T as NVTextField
Set T ?= NvCurrentProject.FindDisplay("Scan state")

if Not(T is Nothing) then
  Select case NvScanState
    case scanStateBeforeTrigger
      T.Title = "Trigger...?"     ' wait for trigger
    case scanStateAfterTrigger
      if T.Title = "Scanning..." then  ' Blink "Scanning..."
        T.Title = ""
      Else
        T.Title = "Scanning..."
      End IF
    case scanStatePreparing
      T.Title = "Busy..."
    case scanStateBusy
      T.Title = "Busy..."
    case scanStateRunning
      T.Title = "Live data..."
    case scanStateRunning
      T.Title = "Live data..."
    case scanStateReady
      T.Title = "Ready..."
    case else
      T.Title = "State..."
  end Select
End If
...
```

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.6 NVSETTIMERINTERVAL

```
Sub NvSetTimerInterval (ms As Integer)
```

Sets the time interval in milliseconds used to call the **`OnTimer`** event. This event cannot be called more often than every 15msec. Due to the operating system Windows®, there may be time offsets, which are usually in the 15msec range.

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.7 NVEXITPROGRAM

```
Sub NvExitProgram (checkModified As Boolean)
```

This command closes **NextView®4**. If **`checkModified`** is **`True`**, any unsaved changes of the project lead to an inquiry whether to save the project. If **`checkModified`** is **`False`**, **NextView®4** is closed immediately without saving possible changes.

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.8 NVANALOGIN

```
Function NvAnalogIn (index As Integer) As NvChannel
```

Returns the analog input channel of the number **`index`** or **`Nothing`** if not existing. The channels are numbered consecutively starting at **`index`**=1 with the first analog input of the device installed at first (see "CLASS "NVCHANNEL"", p. 60).

Only scanned channels are available in NextView®4 Script during a running scan.

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.9 NVANALOGOUT

```
Function NvAnalogOut (index As Integer) As NvChannel
```

Returns the analog output channel of the number **index** or **Nothing** if not existing. The channels are numbered consecutively starting at **index**=1 with the first analog output of the device installed at first (see "CLASS "NVCHANNEL"", p. 60).

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.10 NVDIGITAL

```
Function NvDigital (index As Integer) As NvChannel
```

Returns the digital channel of the number **index** or **Nothing** if not existing. The channels are numbered consecutively starting at **index**=1 with the first digital channel of the device installed at first (see "CLASS "NVCHANNEL"", p. 60).

```
' returns the current value of the 1. digital channel
' in the message display

Print NvDigital(1).Value
```

Only scanned channels are available in NextView®4 Script during a running scan.

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.11 NVDIGITALLINE

```
Function NvDigitalLine (index As Integer) As NvChannel
```

Returns the digital line of the number **index** or **Nothing** if not existing. The channels are numbered consecutively starting at **index**=1 with the first digital line of the device installed at first (see "CLASS "NVCHANNEL"", p. 60).

```
' returns the current value of the 1. digital line
' in the message display and sets it to high

Print NvDigitalLine(1).Value
Nv.DigitalLine(1).Value = 1
```

Only scanned channels are available in NextView®4 Script during a running scan.

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.12 NVCOUNTER

```
Function NvCounter (index As Integer) As NvChannel
```

Returns the counter channel of the number **index** or **Nothing** if not existing. The channels are numbered consecutively starting at **index**=1 with the first counter of the device installed at first (see "CLASS "NVCHANNEL"", p. 60).

Only scanned channels are available in NextView®4 Script during a running scan.

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.1.13 NVFORMULA

```
Function NvFormula (index As Integer) As NvChannel
```

Returns the formula channel of the number **`index`** or **`Nothing`** if not existing. The channels are numbered consecutively starting at **`index`**=1 with the first formula channel of the device installed at first (see "CLASS "NVCHANNEL"", p. 60).

Only scanned channels are available in NextView®4 Script during a running scan.

A list of available NV4 functions and procedures is provided in chapter "NV4 functions / procedures: Overview" on page 52.

3.2 CLASS "NVCHANNEL"

The class "NvChannel" describes the properties and functions of the available input and output channels in **NextView®4**. An object of the **NextView®4** class "NvChannel" cannot be created with the command **New**. It is returned by **NvSetTimerInterval**, **NvAnalogOut**, **NvDigital**, **NvDigitalLine**, **NvFormula** instead.

```
' outputs the 1. analog input channel as NvChannel object

Dim sig As NvChannel
Set sig = NvAnalogIn(1)

If sig Is Nothing Then
  Print "Channel does not exist"
Else
  Print "Current value: " & sig.Value & " " & sig.Unit
End If
```

3.2.1 NVCHANNEL: OVERVIEW

Element function	Description
NvChannel.Name	name of the channel
NvChannel.Group	group of the channel
NvChannel.Comment	comment of the channel
NvChannel.Unit	unit of the channel
NvChannel.Value	current value of the channel

3.2.2 NVCHANNEL.NAME

```
Function Name () As String
```

Returns the name of the channel.

```
' outputs the name of the 1. analog input channel
' in the message display
Print NvAnalogIn(1).Name
```

A list of all available element functions of the class "NvChannel" is provided in chapter "NvChannel: Overview" on page 60.

3.2.3 NVCHANNEL.GROUP

```
Function Group () As String
```

Returns the group of the channel.

```
' outputs the group of the 1. analog input channel
' in the message display
Print NvAnalogIn(1).Group
```

A list of all available element functions of the class "NvChannel" is provided in chapter "NvChannel: Overview" on page 60.

3.2.4 NVCHANNEL.COMMENT

```
Function Comment () As String
```

Returns the comment for the channel.

```
' outputs the comment of the 1. analog input channel
' in the message display
Print NvAnalogIn(1).Comment
```

A list of all available element functions of the class "NvChannel" is provided in chapter "NvChannel: Overview" on page 60.

3.2.5 NVCHANNEL.UNIT

```
Function Unit () As String
```

Returns the unit of the channel.

```
' outputs the unit of the 1. analog input channel
' in the message display
Print NvAnalogIn(1).Unit
```

A list of all available element functions of the class "NvChannel" is provided in chapter "NvChannel: Overview" on page 60.

3.2.6 NVCHANNEL.VALUE

```
' read value:
  Function Value () As Double
' set value:
  Value = newValue      'newValue As Double
```

Sets or returns the output value of a channel. If the channel is an input, always the last measured live data value is returned. How often this value will be updated, depends on the scan settings of the DAQ hardware used.

```
' sets the value of the 1. analog output channel
' and outputs it in the message display

Dim sig As NvChannel

Set sig = NvAnalogOut(1)

sig.Value = -3.14
Print sig.Value
```

A list of all available element functions of the class "NvChannel" is provided in chapter "NvChannel: Overview" on page 60.

3.3 CLASS "NVPROJECT"

An object of the **NextView®4** class "NvProject" cannot be created with the command **New**. It is returned by the **NextView®4** function **NvCurrentProject** instead.

```
' finds a sheet

Const SN As String = "Live elements"

Sub OnClick (State As Boolean)
  Dim p As NvProject
  Set p = NvCurrentProject
  If p.FindSheet(SN) Is Nothing Then
    Print "Sheet " & SN & " not found"
    Exit Sub
  End If
End Sub
```

3.3.1 NVPROJECT: OVERVIEW

Element function	Description
NvProject.SheetCount	returns the number of sheets in the project
NvProject.Sheet	returns a sheet of the project
NvProject.FindSheet	finds a sheet in the project
NvProject.FindDisplay	finds a display in the project
NvProject.Name	returns the name of the project
NvProject.SetPrintInfo	set header information for the printout
NvProject.SetPrintOptions	set print options
NvProject.ActiveSheet	returns the active sheet
NvProject.ActiveDisplay	returns the active display

3.3.2 NVPROJECT.SHEETCOUNT

```
Function SheetCount () As Integer
```

Returns the number of sheets in the project.

A list of all available element functions of the class "NvProject" is provided in chapter "NvProject: Overview" on page 63.

3.3.3 NVPROJECT.SHEET

```
Function Sheet (index As Integer) As NvSheet
```

Returns the sheet number **index** or **Nothing** if not existing (see "CLASS "NVSHEET"", p. 68).

A list of all available element functions of the class "NvProject" is provided in chapter "NvProject: Overview" on page 63.

3.3.4 NVPROJECT.FINDSHEET

```
Function FindSheet (name As String) As NvSheet
```

Returns the sheet with the name **name** or **Nothing** if not existing (see "CLASS "NVSHEET"", p. 68).

A list of all available element functions of the class "NvProject" is provided in chapter "NvProject: Overview" on page 63.

3.3.5 NVPROJECT.FINDDISPLAY

```
Function FindDisplay (name As String) As NvDisplay
```

Returns the display with the name **`name`** or **`Nothing`** if not existing (see "CLASS "NVDISPLAY"", p. 71).

A list of all available element functions of the class "NvProject" is provided in chapter "NvProject: Overview" on page 63.

3.3.6 NVPROJECT.NAME

```
Function Name () as String
```

Returns the full name of the **NextView®4** project.

A list of all available element functions of the class "NvProject" is provided in chapter "NvProject: Overview" on page 63.

3.3.7 NVPROJECT.SETPRINTINFO

```
Sub SetPrintInfo (header As String, comment As String,
  responsible as String, company As String)
```

This command specifies the following input fields for the printout: header, comment, person in charge ("Responsible"), and company. The comment can be multiline. Lines are wrapped with a linefeed (**`chr(10)`**) in the passed string.

A list of all available element functions of the class "NvProject" is provided in chapter "NvProject: Overview" on page 63.

3.3.8 NVPROJECT.SETPRINTOPTIONS

```
Sub SetPrintOptions (noFrame as Boolean, printHeading As
  Boolean, useColors as Boolean, usePatterns as Boolean,
  singleaxis as Boolean, legend as Boolean, pages as
  Integer)
```

This commands specifies the following layout options for the printout:

Parameter	Description
noFrame	page is printed without frame
printHeading	heading included in the printout
legend	legend with information about the displayed signals (e.g. storage location, sample period, number of measuring values, signal information, cursor values)
useColors	colored printout
usePatterns	signal lines are printed as a pattern and not as a continuous line (for a better differentiation if printing black-and-white)
singleAxis	use the same y-axis for all signals (0..100%)
pages	number of pages used for printing the signals

A list of all available element functions of the class "NvProject" is provided in chapter "NvProject: Overview" on page 63.

3.3.9 NVPROJECT.ACTIVESHEET

```
Function ActiveSheet () As NvSheet
```

Returns the currently active sheet or **Nothing** if no sheet exists (see "CLASS "NVSHEET"", p. 68).

A list of all available element functions of the class "NvProject" is provided in chapter "NvProject: Overview" on page 63.

3.3.10 NVPROJECT.ACTIVEDISPLAY

```
Function ActiveDisplay () As NvDisplay
```

Returns the currently active display or **`Nothing`** if no display exists or has been selected (see "CLASS "NVDISPLAY"", p. 71).

A list of all available element functions of the class "NvProject" is provided in chapter "NvProject: Overview" on page 63.

3.4 CLASS "NVSHEET"

An object of the **NextView®4** class "NvSheet" cannot be created with the command **`New`**. It is generated by the functions **`NvProject.Sheet`** and **`NvProject.FindSheet`** of the class "NvProject" instead.

```
' finds a sheet and activates it

Const SN As String = "Live elements"

Sub OnClick (State As Boolean)
  Dim s As NvSheet
  Set s = NvCurrentProject.FindSheet(SN)
  If s Is Nothing Then
    Print "Sheet " & SN & " not found"
    Exit Sub
  End If
  s.Activate
End Sub
```

3.4.1 NVSHEET: OVERVIEW

Element function	Description
`NvSheet.Name`	set or read name of the sheet
`NvSheet.DisplayCount`	returns the number of displays on a sheet
`NvSheet.Display`	returns a display of a sheet
`NvSheet.Activate`	activates this sheet
`NvSheet.Print`	prints the sheet

3.4.2 NVSHEET.NAME

```
' read name:
  Function Name () As String
' set name:
  Name = newName      ' newName As String
```

Sets or reads the name of a sheet.

```
' sets the name of the first sheet
If NvCurrentProject.SheetCount < 1 Then Exit Sub
NvCurrentProject.Sheet(1).Name = "Beer froth decay"
```

A list of all available element functions of the class "NvSheet" is provided in chapter "NvSheet: Overview" on page 68.

3.4.3 NVSHEET.DISPLAYCOUNT

```
Function DisplayCount () As Integer
```

Returns the number of displays on the sheet.

A list of all available element functions of the class "NvSheet" is provided in chapter "NvSheet: Overview" on page 68.

3.4.4 NVSHEET.DISPLAY

```
Function Display (index As Integer) As NvDisplay
```

Returns the display with the number **index** or **Nothing** if not existing (see "CLASS "NVDISPLAY"", p. 71).

A list of all available element functions of the class "NvSheet" is provided in chapter "NvSheet: Overview" on page 68.

3.4.5 NVSHEET.ACTIVATE

```
Sub Activate ()
```

Activates the sheet, i.e. makes it visible.

A list of all available element functions of the class "NvSheet" is provided in chapter "NvSheet: Overview" on page 68.

3.4.6 NVSHEET.PRINT

```
' print sheet:
  Sub Print (useDefaults as Boolean)
```

Prints the displayed sheet with the selected printer. If the variable **`useDefaults`** is set to **`True`**, the standard printer parameters will be used.

A list of all available element functions of the class "NvSheet" is provided in chapter "NvSheet: Overview" on page 68.

3.5 CLASS "NVDISPLAY"

An object of the **NextView®4** class "NvDisplay" cannot be created with the command **`New`**. It is generated by the function **`NvSheet.Display`** of the class "NvSheet" instead.

```
' outputs the display positions on a sheet
Sub OnClick (State As Boolean)
  Dim d As NvDisplay
  Dim s As NvSheet

  If NvCurrentProject.SheetCount < 1 Then Exit Sub
  Set s = NvCurrentProject.Sheet (1)
  Print "Number of displays: " & s.DisplayCount
  For i = 1 To s.DisplayCount
    Set d = s.Display (i)
    Print "Display " & i & ": " & d.Left & ", " & d.Top & ", " _
                  & d.Width & ", " & d.Height
  Next
End Sub
```

3.5.1 NVDISPLAY: OVERVIEW

Element function	Description
`NvDisplay.Name`	set or read display name
`NvDisplay.Sheet`	output related sheet
`NvDisplay.Left`	set or read left position of the display
`NvDisplay.Top`	set or read upper position of the display
`NvDisplay.Width`	set or read display width
`NvDisplay.Height`	set or read display height
`NvDisplay.Bounds`	set display size and position
`NvDisplay.Print`	prints the display
`NvDisplay.SetFont`	set font parameters for the display

3.5.2 NVDISPLAY.NAME

```
' read name:
  Function Name () As String
' set name:
  Name = newName      ' newName As String
```

Sets or returns the name of a single display.

```
' sets the name of the first display of the first sheet

If NvCurrentProject.Sheet(1).DisplayCount < 1 Then Exit Sub
NvCurrentProject.Sheet(1).Display(1).Name = "Test Display"
```

A list of all available element functions of the class "NvDisplay" is provided in chapter "NvDisplay: Overview" on page 71.

3.5.3 NVDISPLAY.SHEET

```
Sub Sheet () as NvSheet
```

Returns the sheet in which the display is integrated (see "CLASS "NVSHEET"", p. 68).

A list of all available element functions of the class "NvDisplay" is provided in chapter "NvDisplay: Overview" on page 71.

3.5.4 NVDISPLAY.LEFT

```
' read left position:
  Function Left () As Integer
' set left position:
  Left = newLeft      ' newLeft As Integer
```

Sets or returns the position of the display on the left side.

A list of all available element functions of the class "NvDisplay" is provided in chapter "NvDisplay: Overview" on page 71.

3.5.5 NVDISPLAY.TOP

```
' read top position:
  Function Top () As Integer
' set top position:
  Top = newTop      ' newTop As Integer
```

Sets or returns the position of the display on the top side.

A list of all available element functions of the class "NvDisplay" is provided in chapter "NvDisplay: Overview" on page 71.

3.5.6 NVDISPLAY.WIDTH

```
' read width:
  Function Width () As Integer
' set width:
  Width = newWidth      ' newWidth As Integer
```

Sets or returns the display width.

A list of all available element functions of the class "NvDisplay" is provided in chapter "NvDisplay: Overview" on page 71.

3.5.7 NVDISPLAY.HEIGHT

```
' Read height:
  Function Height () As Integer
' Set height:
  Height = newHeight      ' newHeight As Integer
```

Sets or returns the display height.

A list of all available element functions of the class "NvDisplay" is provided in chapter "NvDisplay: Overview" on page 71.

3.5.8 NVDISPLAY.BOUNDS

```
Sub Bounds (left as Integer, top as Integer,
            right as Integer, bottom as Integer)
```

Specifies the size of the rectangular display by entering the corner positions **`left`**, **`top`** and **`right`**, **`bottom`** in pixels. The position (0,0) is the top left corner of the sheet.

A list of all available element functions of the class "NvDisplay" is provided in chapter "NvDisplay: Overview" on page 71.

3.5.9 NVDISPLAY.PRINT

```
' print display:
  Sub Print (useDefaults as Boolean)
```

Prints the display on the default printer. If the variable **`useDefaults`** is set to **`True`**, the default printer parameters will be used.

A list of all available element functions of the class "NvDisplay" is provided in chapter "NvDisplay: Overview" on page 71.

3.5.10 NVDISPLAY.SETFONT

```
Sub SetFont (name as String, size as Integer,
             bold as Integer, italic as Integer)
```

Defines the font parameters for the display.

Parameter	Description
name	font name
size	font size in didot points
bold	font weight: 1 = bold; 0 = normal
italic	font style: 1 = italic; 0 = normal

```
...
Dim d As NvDisplay
Set d = NvCurrentProject.Sheet(1).Display(1)
' set font to Arial, 16 point, bold, not italic.
d.SetFont "Arial", 16, 1, 0
...
```

A list of all available element functions of the class "NvDisplay" is provided in chapter "NvDisplay: Overview" on page 71.

3.6 CLASS "NVBUTTON"

The class "NvButton" describes the properties of a button placed on a sheet. An object of the **NextView®4** class "NvButton" cannot be created with the command **New**. It is returned by the routine **`NvProject.FindDisplay`** instead.

```
' Example for NvButton

Sub ToggleState ()  ' continuously turn button on/off
   Dim B as NvButton
   Set B ?= NvCurrentProject.FindDisplay("Button Test")
   if Not(B is Nothing) then
      B.State = Not B.State
   End If
End Sub
```

3.6.1 NVBUTTON: OVERVIEW

Element function	Description
`NvButton.Title`	read/enter title of the button
`NvButton.State`	read/set state of the button
`NvButton.SetColor`	specify color settings of the button
`NvButton.GetColor`	read color settings of the button
`NvButton.SetActiveColor`	specify color settings of the button when active
`NvButton.GetActiveColor`	read color settings of the button when active
`NvButton.SetInactiveColor`	specify color settings of the button when inactive
`NvButton.GetInactiveColor`	read color settings of the button when inactive
`NvButton.ActiveTitle`	read/enter title of the button when active
`NvButton.InactiveTitle`	read/enter title of the button when inactive

3.6.2 NVBUTTON.TITLE

```
' read title:
  Function Title () As String
' set title:
  Title = newTitle        ' newTitle As String
```

This command is used to enter or read the title of a button.

A list of all available element functions of the class "NvButton" is provided in chapter "NvButton: Overview" on page 76.

3.6.3 NVBUTTON.STATE

```
' read state:
  Function State () As String
' set state:
  State = newState        ' newState As String
```

A button has two states: ON (**`True`**) or OFF (**`False`**). This command sets or reads the state of a button.

A list of all available element functions of the class "NvButton" is provided in chapter "NvButton: Overview" on page 76.

3.6.1 NVBUTTON.SETCOLOR

```
Sub SetColor (textColor as Integer, bgColor as Integer)
```

Specifies the color settings of the button. The values to be passed are calculated with the standard function **RGBColor**.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvButton" is provided in chapter "NvButton: Overview" on page 76.

3.6.2 NVBUTTON.GETCOLOR

```
Sub GetColor (Byref textColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the button. The output values are RGB values.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvButton" is provided in chapter "NvButton: Overview" on page 76.

3.6.3 NVBUTTON.SETACTIVECOLOR

```
Sub SetActiveColor (textColor as Integer,
  bgColor as Integer)
```

Specifies the color settings of the button in active state (alarm state). The values to be passed are calculated with the standard function **RGBColor**.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvButton" is provided in chapter "NvButton: Overview" on page 76.

3.6.4 NVBUTTON.GETACTIVECOLOR

```
Sub GetActiveColor (Byref textColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the button in active state (alarm state). The output values are RGB values.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvButton" is provided in chapter "NvButton: Overview" on page 76.

3.6.5 NVBUTTON.SETINACTIVECOLOR

```
Sub SetInactiveColor (textColor as Integer,
  bgColor as Integer)
```

Specifies the color settings of the button in inactive state (normal state). The values to be passed are calculated with the standard function **RGBColor**.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvButton" is provided in chapter "NvButton: Overview" on page 76.

3.6.6 NVBUTTON.GETINACTIVECOLOR

```
Sub GetInactiveColor (Byref textColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the button in inactive state (normal state). The output values are RGB values.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvButton" is provided in chapter "NvButton: Overview" on page 76.

3.6.7 NVBUTTON.ACTIVETITLE

```
' read title:
  Function ActiveTitle () As String
' set title:
  ActiveTitle = newTitle      ' newTitle As String
```

Sets or returns the text displayed when the button is active.

A list of all available element functions of the class "NvButton" is provided in chapter "NvButton: Overview" on page 76.

3.6.8 NVBUTTON.INACTIVETITLE

```
' read title:
  Function InactiveTitle () As String
' set title:
  InactiveTitle = newTitle      ' newTitle As String
```

Sets or returns the text displayed when the button is inactive.

A list of all available element functions of the class "NvButton" is provided in chapter "NvButton: Overview" on page 76.

3.7 CLASS "NVSLIDER"

The class "NvSlider" describes the properties of a slider placed on a sheet. An object of the **NextView®4** class "NvSlider" cannot be created with the command **New**. It is returned by the routine **`NvProject.FindDisplay`** instead.

```
' Example for NvSlider

Dim Ampl As Double
Sub OnSliderAmplMove (pos As Double)
  Dim S as NvSlider
  Set S ?= NvCurrentProject.FindDisplay("Slider amplitude")
  if S is Nothing then Exit Sub
  Ampl = pos * (S.Maximum - S.Minimum) + S.Minimum
End Sub
```

3.7.1 NVSLIDER: OVERVIEW

Element function	Description
`NvSlider.Title`	returns or sets the title of the slider
`NvSlider.Value`	value at the current slider position
`NvSlider.Minimum`	minimum value of the slider scale
`NvSlider.Maximum`	maximum value of the slider scale

3.7.2 NVSLIDER.TITLE

```
' read title:
  Function Title () As String
' set title:
  Title = newTitle        ' newTitle As String
```

This command is used to enter or read the title of a slider.

A list of all available element functions of the class "NvSlider" is provided in chapter "NvSlider: Overview" on page 82.

3.7.3 NVSLIDER.VALUE

```
' read value:
  Function Value () As Double
' set value:
  Value = newValue      ' newValue As Double
```

Returns or sets the value at the current slider position.

A list of all available element functions of the class "NvSlider" is provided in chapter "NvSlider: Overview" on page 82.

3.7.4 NVSLIDER.MINIMUM

```
' read minimum:
  Function Minimum () As Double
' set minimum:
  Minimum = newMinimum    ' newMinimum As Double
```

Returns or specifies the lowest value of the slider scale.

A list of all available element functions of the class "NvSlider" is provided in chapter "NvSlider: Overview" on page 82.

3.7.5 NVSLIDER.MAXIMUM

```
' read maximum:
  Function Maximum () As Double
' set maximum:
  Maximum = newMaximum    ' newMaximum As Double
```

Returns the highest value of the slider scale or sets it to a new value.

A list of all available element functions of the class "NvSlider" is provided in chapter "NvSlider: Overview" on page 82.

3.8 CLASS "NVTEXTFIELD"

The class "NvTextField" describes the properties of a text field placed on a sheet. An object of the **NextView®4** class "NvTextField" cannot be created with the command **New**. It is returned by the routine **NvProject.FindDisplay** instead.

```
' Example for NvTextField

Sub Test ()
  Dim T as NvTextField
  Set T ?= NvCurrentProject.FindDisplay("Text Test")
  if Not(T is Nothing) then
    Print T.Title
  End If
End Sub
```

3.8.1 NVTEXTFIELD: OVERVIEW

Element function	Description
NvTextField.Title	read/enter content of the text field
NvTextField.SetColor	specify color settings of the text field
NvTextField.GetColor	read color settings of the text field
NvTextField.SetActiveColor	specify color settings of the text field when active
NvTextField.GetActiveColor	read color settings of the text field when active
NvTextField.SetInactiveColor	specify color settings of the text field when inactive
NvTextField.GetInactiveColor	read color settings of the text field when inactive
NvTextField.ActiveTitle	read/enter title of the text field when active
NvTextField.InactiveTitle	read/enter title of the text field when inactive

3.8.2 NVTEXTFIELD.TITLE

```
' read text:
  Function Title () As String
' set text:
  Title = newTitle        ' newTitle As String
```

This command is used to enter or read the content of a text field.

A list of all available element functions of the class "NvTextField" is provided in chapter "NvTextField: Overview" on page 84.

3.8.3 NVTEXTFIELD.SETCOLOR

```
Sub SetColor (textColor as Integer, bgColor as Integer)
```

Specifies the color settings of the text field. The values to be passed are calculated with the standard function **RGBColor**.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvTextField" is provided in chapter "NvTextField: Overview" on page 84.

3.8.4 NVTEXTFIELD.GETCOLOR

```
Sub GetColor (Byref textColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the text field. The output values are RGB values.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvTextField" is provided in chapter "NvTextField: Overview" on page 84.

3.8.5 NVTEXTFIELD.SETACTIVECOLOR

```
Sub SetActiveColor (textColor as Integer,
  bgColor as Integer)
```

Specifies the color settings of the text field in active state (alarm state). The values to be passed are calculated with the standard function **RGBColor**.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvTextField" is provided in chapter "NvTextField: Overview" on page 84.

3.8.6 NVTEXTFIELD.GETACTIVECOLOR

```
Sub GetActiveColor (Byref textColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the text field in active state (alarm state). The output values are RGB values.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvTextField" is provided in chapter "NvTextField: Overview" on page 84.

3.8.7 NVTEXTFIELD.SETINACTIVECOLOR

```
Sub SetInactiveColor (textColor as Integer,
  bgColor as Integer)
```

Specifies the color settings of the text field in inactive state (normal state). The values to be passed are calculated with the standard function **RGBColor**.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvTextField" is provided in chapter "NvTextField: Overview" on page 84.

3.8.8 NVTEXTFIELD.GETINACTIVECOLOR

```
Sub GetInactiveColor (Byref textColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the text field in inactive state (normal state). The output values are RGB values.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvTextField" is provided in chapter "NvTextField: Overview" on page 84.

3.8.9 NVTEXTFIELD.ACTIVETITLE

```
' read title:
  Function ActiveTitle () As String
' set title:
  ActiveTitle = newTitle      ' newTitle As String
```

Sets or returns the text displayed when the text field is active.

A list of all available element functions of the class "NvTextField" is provided in chapter "NvTextField: Overview" on page 84.

3.8.10 NVTEXTFIELD.INACTIVETITLE

```
' read title:
  Function InactiveTitle () As String
' set title:
  InactiveTitle = newTitle      ' newTitle As String
```

Sets or returns the text displayed when the text field is inactive.

A list of all available element functions of the class "NvTextField" is provided in chapter "NvTextField: Overview" on page 84.

3.9 CLASS "NVGRAPHDISPLAY"

The **NextView®4** class "NvGraphDisplay" derives from the class "NvDisplay". An object of the **NextView®4** class "NvGraphDisplay" cannot be created with the command **New**. An object of the class "NvDisplay" is generated by the function **NvSheet.Display** of the class "NvDisplay" instead. If it is an "NvGraphDisplay", it can be assigned with the "?=" operator.

```
' returns the positions of the displays on the sheet
Sub OnClick (State As Boolean)
  Dim graph As NvGraphDisplay
  Dim s As NvSheet

  If NvCurrentProject.SheetCount < 1 Then Exit Sub
  Set s = NvCurrentProject.Sheet (1)
  Print "Number of displays: " & s.DisplayCount

  For i = 1 To s.DisplayCount
    Set graph ?= s.Display (i)

    If graph Is Nothing Then
      Print "Display " & i & " is no GraphDisplay"
    Else
      Print "GraphDisplay " & i & ": " & graph.Left&  ", " _
            & graph.Top
      Print "  Number of signals: " & graph.SignalCount
    End If
  Next
End Sub
```

3.9.1 NVGRAPHDISPLAY: OVERVIEW

Element function	Description
NvGraphDisplay.SignalCount	returns the number of signals in the graph display
NvGraphDisplay.Signal	returns a signal of the graph display
NvGraphDisplay.WhiteCursor	returns the white (1.) cursor of the graph display
NvGraphDisplay.BlackCursor	returns the black (2.) cursor of the graph display
NvGraphDisplay.xAxis	returns the x-axis of a graph display
NvGraphDisplay.yAxis	returns the y-axis of a signal of a graph display

3.9.2 NVGRAPHDISPLAY.SIGNALCOUNT

```
Function SignalCount () As Integer
```

Returns the number of signals in the graph display.

A list of all available element functions of the class "NvGraphDisplay" is provided in chapter "NvGraphDisplay: Overview" on page 89.

3.9.3 NVGRAPHDISPLAY.SIGNAL

```
Function Signal (index As Integer) As NvSignal
```

Returns the signal with the number **`index`** or **`Nothing`** if not existing (see "CLASS "NVSIGNAL"", p. 120).

Signals in a display are read-only. If changing a signal, the related measuring file must be opened with write/read access first using the commands of the class "NvOpenDataFile" before the required changes can be done in the file.

A list of all available element functions of the class "NvGraphDisplay" is provided in chapter "NvGraphDisplay: Overview" on page 89.

3.9.4 NVGRAPHDISPLAY.WHITECURSOR

```
Function WhiteCursor () As NvCursor
```

Returns the white (1.) cursor of the graph display (see "CLASS "NVCURSOR"", p. 103).

A list of all available element functions of the class "NvGraphDisplay" is provided in chapter "NvGraphDisplay: Overview" on page 89.

3.9.5 NVGRAPHDISPLAY.BLACKCURSOR

```
Function BlackCursor () As NvCursor
```

Returns the black (2.) cursor of the graph display (see "CLASS "NVCURSOR"", p. 103).

A list of all available element functions of the class "NvGraphDisplay" is provided in chapter "NvGraphDisplay: Overview" on page 89.

3.9.6 NVGRAPHDISPLAY.XAXIS

```
Function xAxis () As NvAxis
```

Returns the x-axis as an element of the class "NvAxis" of the graph display (see p. 105).

A list of all available element functions of the class "NvGraphDisplay" is provided in chapter "NvGraphDisplay: Overview" on page 89.

3.9.7 NVGRAPHDISPLAY.YAXIS

```
Function yAxis (index as Integer) As NvAxis
```

Returns the y-axis as an element of the class "NvAxis" of the signal with the number **index** in the graph display (see p. 105).

A list of all available element functions of the class "NvGraphDisplay" is provided in chapter "NvGraphDisplay: Overview" on page 89.

3.10 CLASS "NVLEVELINDICATOR"

The class "NvLevelIndicator" describes the properties of a level indicator placed on a sheet. An object of the **NextView®4** class "NvLevelIndicator" cannot be created with the command **New**. It is returned by the routine **NvProject.FindDisplay** instead.

```
' Example for NvLevelIndicator

Sub Test ()
  Dim L as NvLevelIndicator
  Set L ?= NvCurrentProject.FindDisplay("Level Test")
  if Not(L is Nothing) then
' only makes sense if no analog channel is related to the display
    L.Title = "Analog*2"
    L.Value = 2 * NvAnalogIn(1).Value
  End If
End Sub
```

3.10.1 NVLEVELINDICATOR: OVERVIEW

Element function	Description
NvLevelIndicator.Title	read/enter channel name of the level indicator
NvLevelIndicator.SetColor	specify color settings of the level indicator
NvLevelIndicator.GetColor	read color settings of the level indicator
NvLevelIndicator.SetActiveColor	specify color settings of the level indicator when active
NvLevelIndicator.GetActiveColor	read color settings of the level indicator when active
NvLevelIndicator.SetInactiveColor	specify color settings of the level indicator when inactive
NvLevelIndicator.GetInactiveColor	read color settings of the level indicator
NvLevelIndicator.Value	read/set value
NvLevelIndicator.Minimum	read/set minimum value
NvLevelIndicator.Maximum	read/set maximum value

3.10.2 NVLEVELINDICATOR.TITLE

```
' read title:
  Function Title () As String
' set title:
  Title = newTitle        ' newTitle As String
```

This command is used to enter or read the channel name of a level indicator.

A list of all available element functions of the class "NvLevelIndicator" is provided in chapter "NvLevelIndicator: Overview" on page 92.

3.10.3 NVLEVELINDICATOR.SETCOLOR

```
Sub SetColor (fgColor as Integer, bgColor as Integer)
```

Specifies the color settings of the level indicator assigning the same values for the inactive and active state. The values to be passed are calculated with the standard function **`RGBColor`**.

Parameter	Description
`fgColor`	foreground color (fill, RGB value)
`bgColor`	background color (RGB value)

A list of all available element functions of the class "NvLevelIndicator" is provided in chapter "NvLevelIndicator: Overview" on page 92.

3.10.4 NVLEVELINDICATOR.GETCOLOR

```
Sub GetColor (Byref fgColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the level indicator. The output values are RGB values.

Parameter	Description
fgColor	foreground color (fill, RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvLevelIndicator" is provided in chapter "NvLevelIndicator: Overview" on page 92.

3.10.5 NVLEVELINDICATOR.SETACTIVECOLOR

```
Sub SetActiveColor (fgColor as Integer, bgColor as Integer)
```

Specifies the color settings of the level indicator in active state (alarm state). The values to be passed are calculated with the standard function **RGBColor**.

Parameter	Description
fgColor	foreground color (fill, RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvLevelIndicator" is provided in chapter "NvLevelIndicator: Overview" on page 92.

3.10.6 NVLEVELINDICATOR.GETACTIVECOLOR

```
Sub GetActiveColor (Byref fgColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the level indicator in active state (alarm state). The output values are RGB values.

Parameter	Description
fgColor	foreground color (fill, RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvLevelIndicator" is provided in chapter "NvLevelIndicator: Overview" on page 92.

3.10.7 NVLEVELINDICATOR.SETINACTIVECOLOR

```
Sub SetInactiveColor (fgColor as Integer,
  bgColor as Integer)
```

Specifies the color settings of the level indicator in inactive state (normal state). The values to be passed are calculated with the standard function **RGBColor**.

Parameter	Description
fgColor	foreground color (fill, RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvLevelIndicator" is provided in chapter "NvLevelIndicator: Overview" on page 92.

3.10.8 NVLEVELINDICATOR.GETINACTIVECOLOR

```
Sub GetInactiveColor (Byref fgColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the level indicator in inactive state (normal state). The output values are RGB values.

Parameter	Description
fgColor	foreground color (fill, RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvLevelIndicator" is provided in chapter "NvLevelIndicator: Overview" on page 92.

3.10.9 NVLEVELINDICATOR.VALUE

```
' read value:
  Function Value () As Double
' set value:
  Value = newValue      ' newValue As Double
```

Returns or sets the value at the current position of the level indicator.

A list of all available element functions of the class "NvLevelIndicator" is provided in chapter "NvLevelIndicator: Overview" on page 92.

3.10.10 NVLEVELINDICATOR.MINIMUM

```
' read minimum:
  Function Minimum () As Double
' set minimum:
  Minimum = newMinimum    ' newMinimum As Double
```

Returns or specifies the lowest value of the level indicator.

A list of all available element functions of the class "NvLevelIndicator" is provided in chapter "NvLevelIndicator: Overview" on page 92.

3.10.11 NVLEVELINDICATOR.MAXIMUM

```
' read maximum:
  Function Maximum () As Double
' set maximum:
  Maximum = newMaximum    ' newMaximum As Double
```

Returns the highest value of the level indicator or sets it to a new value.

A list of all available element functions of the class "NvLevelIndicator" is provided in chapter "NvLevelIndicator: Overview" on page 92.

3.11 CLASS "NVDVM"

The class "NvDVM" describes the properties of a button placed on a sheet. An object of the **NextView®4** class "NvDVM" cannot be created with the command **New**. It is returned by the routine **`NvProject.FindDisplay`** instead.

```
' example for NvDVM

Sub Test ()
  Dim D as NvDVM
  Set D ?= NvCurrentProject.FindDisplay("DVM Test")
  if Not(D is Nothing) then
' only makes sense if no analog channel is related to the display
    D.Title = "Analog*2"
    D.Value = 2 * NvAnalogIn(1).Value
  End If
End Sub
```

3.11.1 NVDVM: OVERVIEW

Element function	Description
`NvDVM.Title`	read/enter channel name of the digital voltmeter
`NvDVM.SetColor`	specify color settings of the digital voltmeter
`NvDVM.GetColor`	read color settings of the digital voltmeter
`NvDVM.SetActiveColor`	specify color settings of the digital voltmeter when active
`NvDVM.GetActiveColor`	read color settings of the digital voltmeter when active
`NvDVM.SetInactiveColor`	specify color settings of the digital voltmeter when inactive
`NvDVM.GetInactiveColor`	read color settings of the digital voltmeter when inactive
`NvDVM.Value`	read/set value

3.11.2 NVDVM.TITLE

```
' read title:
  Function Title () As String
' set title:
  Title = newTitle        ' newTitle As String
```

This command is used to enter or read the channel name indicated by the digital voltmeter.

A list of all available element functions of the class "NvDVM" is provided in chapter "NvDVM: Overview" on page 98.

3.11.3 NVDVM.SETCOLOR

```
Sub SetColor (textColor as Integer, bgColor as Integer)
```

Specifies the color settings of the digital voltmeter assigning the same values for the inactive and active state. The values to be passed are calculated with the standard function **`RGBColor`**.

Parameter	Description
`textColor`	text color (RGB value)
`bgColor`	background color (RGB value)

A list of all available element functions of the class "NvDVM" is provided in chapter "NvDVM: Overview" on page 98.

3.11.4 NVDVM.GETCOLOR

```
Sub GetColor (Byref textColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the digital voltmeter. The output values are RGB values.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvDVM" is provided in chapter "NvDVM: Overview" on page 98.

3.11.5 NVDVM.SETACTIVECOLOR

```
Sub SetActiveColor (textColor as Integer,
  bgColor as Integer)
```

Specifies the color settings of the digital voltmeter in active state (alarm state). The values to be passed are calculated with the standard function **RGBColor**.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvDVM" is provided in chapter "NvDVM: Overview" on page 98.

3.11.6 NVDVM.GETACTIVECOLOR

```
Sub GetActiveColor (Byref textColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the digital voltmeter in active state (alarm state). The output values are RGB values.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvDVM" is provided in chapter "NvDVM: Overview" on page 98.

3.11.7 NVDVM.SETINACTIVECOLOR

```
Sub SetInactiveColor (textColor as Integer,
  bgColor as Integer)
```

Specifies the color settings of the digital voltmeter in inactive state (normal state). The values to be passed are calculated with the standard function **RGBColor**.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvDVM" is provided in chapter "NvDVM: Overview" on page 98.

3.11.8 NVDVM.GETINACTIVECOLOR

```
Sub GetInactiveColor (Byref textColor as Integer,
  Byref bgColor as Integer)
```

Returns the color settings of the digital voltmeter in inactive state (normal state). The output values are RGB values.

Parameter	Description
textColor	text color (RGB value)
bgColor	background color (RGB value)

A list of all available element functions of the class "NvDVM" is provided in chapter "NvDVM: Overview" on page 98.

3.11.9 NVDVM.VALUE

```
' read value:
  Function Value () As Double
' set value:
  Value = newValue      ' newValue As Double
```

Returns or sets the current value of the digital voltmeter.

A list of all available element functions of the class "NvDVM" is provided in chapter "NvDVM: Overview" on page 98.

3.12 CLASS "NVCURSOR"

An object of the **NextView®4** class "NvCursor" cannot be created with the command **`New`**. An object of the **NextView®4** class "NvCursor**"** is generated by the functions **`NvGraphDisplay.BlackCursor`** and **`NvGraphDisplay.WhiteCursor`** of the class "NvGraphDisplay" instead.

```
Dim g As NvGraphDisplay

' check if sheet and display exist
If NvCurrentProject.SheetCount < 1 Then Exit Sub
If NvCurrentProject.Sheet(1).DisplayCount < 1 Then Exit Sub

Set g ?= NvCurrentProject.Sheet(1).Display(1)
' is the display a graph display?
If g Is Nothing Then
  Print "Display is no graph display"
  Exit Sub
End If

Dim c As NvCursor
Set c = g.WhiteCursor

If c.Enabled Then
  Dim ax As NvAxis
  Set ax = g.xAxis
  If ax.IsDateMode Then
    print "WhiteCursor: " & TimeStampStr(c.Value)
  Else
    Print "WhiteCursor: " & c.Value
  End If
Else
  Print "WhiteCursor: not active"
End If
```

3.12.1 NVCURSOR: OVERVIEW

Element function	Description
`NvCursor.Value`	set or read cursor value
`NvCursor.Enabled`	set or read cursor state'

3.12.2 NVCURSOR.VALUE

```
' read value:
  Function Value () As Double
' set value:
  Value = newValue      ' newValue As Double
```

Sets or returns the cursor value. The value depends on the time format of the graph display. If using the date/time format, the value is displayed in absolute time. In relative time, the display shows seconds.

A list of all available element functions of the class "NvCursor" is provided in chapter "NvCursor: Overview" on page 103.

3.12.3 NVCURSOR.ENABLED

```
' read cursor state:
  Function Enabled () As Boolean
' set cursor state:
  Enabled = newState   ' newState As Boolean
```

Returns or sets the cursor state in the display. The cursor is either ON (**`True`**) or OFF (**`False`**).

A list of all available element functions of the class "NvCursor" is provided in chapter "NvCursor: Overview" on page 103.

3.13 CLASS "NVAXIS"

An object of the **NextView®4** class "NvAxis" cannot be created with the command **New**. An object of the **NextView®4** class "NvAxis**"** is generated by the functions **NvGraphDisplay.xAxis** and **NvGraphDisplay.yAxis** of the class "NvGraphDisplay" instead.

```
Dim g As NvGraphDisplay
Set g ?= NvCurrentProject.Sheet(1).Display(1)
' Is the display a graph display?
If g Is Nothing Then
  Print "Display is no graph display"
  Exit Sub
End If

Dim ax As NvAxis
Set ax = g.xAxis
If ax.IsDateMode Then
  print "x-axis minimum: " & TimeStampStr(ax.Min)
  print "x-axis maximum: " & TimeStampStr(ax.Max)
Else
  print "x-axis minimum: " & ax.Min
  print "x-axis maximum: " & ax.Max
End If
Dim ay As NvAxis
Set ay = g.yAxis(1)
ay.Min = 0.1
ay.Max = 1.1

print "y-axis set to: " & ay.Min & " .. " & ay.Max
```

3.13.1 NVAXIS: OVERVIEW

Element function	Description
NvAxis.Min	returns the lowest value of the axis
NvAxis.Max	returns the highest value of the axis
NvAxis.SetDateMode	sets the date/time mode of the x-axis to absolute or relative time
NvAxis.IsDateMode	returns the date/time mode of the x-axis (absolute or relative time)

3.13.2 NVAXIS.MIN

```
' read value:
  Function Min () As Double
' set value:
  Min = newMin      ' newMin As Double
```

Returns or specifies the lowest value of the axis. The value of the x-axis depends on the display format. If using the date/time format, the value is displayed in absolute time (PC local time). In relative time, the display shows seconds.

A list of all available element functions of the class "NvAxis" is provided in chapter "NvAxis: Overview" on page 105.

3.13.3 NVAXIS.MAX

```
' read value:
  Function Max () As Double
' set value:
  Max = newMax      ' newMax As Double
```

Sets or returns the upper value of the axis. The value of the x-axis depends on the display format. If using the date/time format, the value is displayed in absolute time (PC local time). In relative time, the display shows seconds.

A list of all available element functions of the class "NvAxis" is provided in chapter "NvAxis: Overview" on page 105.

3.13.4 NVAXIS.SETDATEMODE

```
' x-Axis only
Sub SetDateMode (mode as Boolean)
```

Sets the data format of the x-axis. If set to **`True`**, the x-axis uses the data/time format. If set to **`False`**, the relative time is displayed. This function does not influence the y-axis.

A list of all available element functions of the class "NvAxis" is provided in chapter "NvAxis: Overview" on page 105.

3.13.5 NVAXIS.ISDATEMODE

```
' x-Axis only
Function IsDateMode () as Boolean
```

Returns the data/time mode set for the x-axis. If **`True`** is returned, the display uses the date/time format. If **`False`** is returned, the relative time is displayed.

```
Dim g As NvGraphDisplay
set g ?= NvCurrentProject.Sheet(1).Display(1)
If Not(g is Nothing) Then
  Dim lTime as Double
  lTime = g.Signal(1).TimeStamp
  print "Start (Localtime) " & TimeStampStr(lTime)
  ' dependent on x-axis mode the position values on
  ' the x-axis has to be set differently
  If g.xAxis.IsDateMode Then
    g.xAxis.Min = lTime + TimeSerial(0,0,30)
    g.xAxis.Max = lTime + TimeSerial(0,0,90)
    g.WhiteCursor.Enabled
    g.WhiteCursor.Value = lTime + TimeSerial(0,0,60)
  Else
    g.xAxis.Min = 30 ' s
    g.xAxis.Max = 90 ' s
    g.WhiteCursor.Enabled
    g.WhiteCursor.Value = 60 ' s
  End If
End If
```

A list of all available element functions of the class "NvAxis" is provided in chapter "NvAxis: Overview" on page 105.

3.14 CLASS "NVPROGRESS"

An object of the **NextView®4** class "NvProgress" shows a progress bar in **NextView®4**. An object of the **NextView®4** class "NvProgress" is created with the command **New**.

```
Dim prog as NvProgress
set prog = new NvProgress
prog.Init (200)
Dim i as Integer
i = 0

Do While (i < 200)
  prog.setstep "Step " & i & " " & rnd, I
  sleep 10
  if prog.aborted then exit do
  i = i + 1
loop

prog.done
```

3.14.1 NVPROGRESS: OVERVIEW

Element function	Description
`NvProgress.Init`	initializes the progress bar
`NvProgress.SetStep`	sets the progress bar to the specified step
`NvProgress.Aborted`	informs about early abort of the progress bar
`NvProgress.Done`	closes the progress bar

3.14.2 NVPROGRESS.INIT

```
' set up progress bar:
  Sub Init (steps as Integer)
```

This command sets up a progress bar with the number of steps specified.

A list of all available element functions of the class "NvProgress" is provided in chapter "NvProgress: Overview" on page 108.

3.14.3 NVPROGRESS.SETSTEP

```
' set to step:
  Sub SetStep (steps as Integer)
```

This command sets the progress bar to the specified step.

A list of all available element functions of the class "NvProgress" is provided in chapter "NvProgress: Overview" on page 108.

3.14.4 NVPROGRESS.ABORTED

```
' query abort of the progress bar:
  Function Aborted () as Boolean
```

Returns **`True`** if the "Abort" key of the progress bar has been pressed.

A list of all available element functions of the class "NvProgress" is provided in chapter "NvProgress: Overview" on page 108.

3.14.5 NVPROGRESS.DONE

```
' close progress bar:
  Sub Done ()
```

This command is used to close the progress bar.

A list of all available element functions of the class "NvProgress" is provided in chapter "NvProgress: Overview" on page 108.

3.15 CLASS "NVOPENDATAFILE"

The class "NvOpenDataFile" is used to open data files for analysis in **NextView®4 Script**. An object of the **NextView®4** class "NvOpenDataFile" is created with the command **New**".

```
Dim o As NvOpenDataFile
Dim f As NvDataFile

Set o = New NvOpenDataFile
If Not o.Browse Then Exit Sub
o.Flags = nvSfReadOnly

Set f = o.Open
If f Is Nothing Then
  Print "File " & o.Path & " not found."
  Exit Sub
End If

Print "File " & o.Path & " contains " & f.SignalCount & " signals."
```

3.15.1 NVOPENDATAFILE: OVERVIEW

Element function	Description
`NvOpenDataFile.Browse`	shows the dialog box "Open file"
`NvOpenDataFile.Path`	sets or returns the file name
`NvOpenDataFile.Flags`	sets or returns special options
`NvOpenDataFile.Open`	opens the file

3.15.2 NVOPENDATAFILE.BROWSE

```
Function Browse () As Boolean
```

Shows the dialog box "Open file". If the user opens a file, the file name will be set and **`Browse`** returns **`True`**. The selected file name is retrieved and modified via the property **`NvOpenDataFile.Path`**. The function **`Open`** (see

"**NvOpenDataFile.Open**", p. 112) opens the selected file. If the user aborts the dialog box "Open file" when calling **Browse**, **False** will be returned.

A list of all available element functions of the class "NvOpenDataFile" is provided in chapter "NvOpenDataFile: Overview" on page 110.

3.15.3 NVOPENDATAFILE.PATH

```
' read path:
  Function Path () As String
' set path:
  Path = newPath          ' newPath As String
```

Returns or sets the file name of the file to be opened.

```
...
Dim o As NvOpenDataFile
Dim f As NvDataFile
Set o = New NvOpenDataFile
o.Path = "c:\data.lfx"
o.Flags = nvSfReadOnly
Set f = o.Open
If f Is Nothing Then
  Print "File " & o.Path & " not found."
  Exit Sub
End If
...
```

A list of all available element functions of the class "NvOpenDataFile" is provided in chapter "NvOpenDataFile: Overview" on page 110.

3.15.4 NVOPENDATAFILE.FLAGS

```
' read flags:
  Function Flags () As Integer
' set flags:
  Flags = newFlags        'newFlags as Integer
```

Returns or sets the options how to open the file.

flags	Description
nvSfReadOnly	opens the file write-protected
nvSfReadWrite	opens the file with reading and writing access

A list of all available element functions of the class "NvOpenDataFile" is provided in chapter "NvOpenDataFile: Overview" on page 110.

3.15.5 NVOPENDATAFILE.OPEN

```
Function Open (Optional path as String, Optional flags as
Integer) As NvDataFile
```

Opens the measuring file specified in **NvOpenDataFile.Path** with the options defined in **NvOpenDataFile.Flags** and returns them as a **NvDataFile** object . If the file does not exist or cannot be opened, **Nothing** will be returned (see "CLASS "NVDATAFILE"", p. 118).

```
Dim o As NvOpenDataFile
Dim f As NvDataFile
...
Set o = New NvOpenDataFile
Set f = o.Open ("c:\data.lfx", nvSfReadOnly)
...
```

Alternative syntax:

```
Dim o As NvOpenDataFile
Dim f As NvDataFile
...
Set o = New NvOpenDataFile
o.Path = "c:\data.lfx"
o.Flags = nvSfReadOnly
Set f = o.Open
...
```

A list of all available element functions of the class "NvOpenDataFile" is provided in chapter "NvOpenDataFile: Overview" on page 110.

3.16 CLASS "NVCREATEDATAFILE"

The class "NvCreateDataFile" is provided in **NextView®4 Script** to create data files. An object of the **NextView®4** class "NvCreateDataFile" is created with the command **New**.

```
Const pi As Double = 3.14159265358979323846264338327 95
Const samples As Integer = 1000

Sub Test()
  Dim c As NvCreateDataFile
  Set c = New NvCreateDataFile

  c.Add nvSfAnalog, samples
  c.Path = "c:\sinus.lfx"

  Dim f As NvDataFile
  Set f = c.Create

  Dim s As NvSignal
  Set s = f.Signal (1)

  s.xResolution = 0.1    ' set measuring frequency to 10Hz

 ' definition of the y-signal parameters
  s.yUnit = "mm"
  s.yRangeMin = -5
  s.yRangeMax = 5
  s.yMin = -5
  s.yMax = 5

  Dim i As Integer
  For i = 1 To samples
    s.Value(i) = 4 * Sin (2 * pi * (i-1)/samples)
  Next
End Sub
```

3.16.1 NVCREATEDATAFILE: OVERVIEW

Element function	Description
NvCreateDataFile.Add	adds a signal to the list used for **Create**
NvCreateDataFile.Reset	removes all signals from the **Create** list
NvCreateDataFile.Browse	shows the "Save file" dialog box
NvCreateDataFile.Path	specifies or returns the file name
NvCreateDataFile.FileType	specifies or returns the file format
NvCreateDataFile.AskOverwrite	sets or returns the option for the "Overwrite" query
NvCreateDataFile.Create	creates the file

3.16.2 NVCREATEDATAFILE.ADD

```
Sub Add (type As Integer, samples As Integer)
```

Adds a signal with **sample** measuring values to the list of signals to be created.

The following table shows the constants for **type**. Please, use only these constants.

type	Description
nvSfAnalog	analog signal
nvSfDigital	digital signal

A list of all available element functions of the class "NvCreateDataFile" is provided in chapter "NvCreateDataFile: Overview" on page 114.

3.16.3 NVCREATEDATAFILE.RESET

```
Sub Reset ()
```

Resets the list of signals to be created, i.e. deletes all items.

A list of all available element functions of the class "NvCreateDataFile" is provided in chapter "NvCreateDataFile: Overview" on page 114.

3.16.4 NVCREATEDATAFILE.BROWSE

```
Function Browse () As Boolean
```

Shows the dialog box "File open". If the user aborts the dialog, **`False`** will be returned. If the user selects a file, the file name will be set and **`Browse`** returns **`True`**. The file name can be retrieved and modified via the property **`NvCreateDataFile.Path`**.

A list of all available element functions of the class "NvCreateDataFile" is provided in chapter "NvCreateDataFile: Overview" on page 114.

3.16.5 NVCREATEDATAFILE.PATH

```
' read path:
  Function Path () As String
' set path:
  Path = newPath          ' newPath As String
```

Returns or specifies the file name of the file to be created.

A list of all available element functions of the class "NvCreateDataFile" is provided in chapter "NvCreateDataFile: Overview" on page 114.

3.16.6 NVCREATEDATAFILE.FILETYPE

```
' read file format:
  Function FileType () As String
' set file format:
  FileType = newType          ' newType As String
```

Returns or specifies the file format of the file to be created.

The following formats are supported:

Name	Description
LFX	**NextView®4** signal file
DAFF	TurboLab signal file
DIADEM	DIAdem signal file
ASCII	ASCII text signal file

A list of all available element functions of the class "NvCreateDataFile" is provided in chapter "NvCreateDataFile: Overview" on page 114.

3.16.7 NVCREATEDATAFILE.ASKOVERWRITE

```
' read value:
  Function AskOverwrite () As Boolean
' write value:
  AskOverwrite = newMode   ' newMode As Boolean
```

Returns or sets the option for the prior query to overwrite an already existing file or not. If the value is **`False`**, the file will be overwritten without any further request.

A list of all available element functions of the class "NvCreateDataFile" is provided in chapter "NvCreateDataFile: Overview" on page 114.

3.16.8 NVCREATEDATAFILE.CREATE

```
Function Create () As NvDataFile
```

Creates the measuring file specified in **`NvCreateDataFile.Path`** with the signal list created by **`NvCreateDataFile.Add`** and returns it as **`NvDataFile`** object, or **`Nothing`** if the file does not exist or cannot be opened (see "CLASS "NVDATAFILE"", p. 118).

A list of all available element functions of the class "NvCreateDataFile" is provided in chapter "NvCreateDataFile: Overview" on page 114.

3.17 CLASS "NVDATAFILE"

An object of the **NextView®4** class "NvDataFile" cannot be created with the command **New**. It is returned by the routine **NvOpenDataFile.Open** instead.

```
Dim o As NvOpenDataFile
Dim f As NvDataFile

Set o = New NvOpenDataFile
If Not o.Browse Then Exit Sub

Set f = o.Open
If f Is Nothing Then
  Print "File " & o.Path & " not found."
  Exit Sub
End If
Print "File " & o.Path & " contains " & f.SignalCount & " signals."
```

3.17.1 NVDATAFILE: OVERVIEW

Element function	Description
NvDataFile.Name	returns the name of the measuring file
NvDataFile.SignalCount	returns the number of signals in the measuring file
NvDataFile.Signal	returns a signal from the measuring file
NvDataFile.CreateTrain	converts the existing measuring file to a file train and returns an object of the class "NvDataTrain"

3.17.2 NVDATAFILE.NAME

```
Function Name () As String
```

Returns the name of the measuring file.

A list of all available element functions of the class "NvDataFile" is provided in chapter "NvDataFile: Overview" on page 118.

3.17.3 NVDATAFILE.SIGNALCOUNT

```
Function SignalCount () As Integer
```

Returns the number of signals in the measuring file.

A list of all available element functions of the class "NvDataFile" is provided in chapter "NvDataFile: Overview" on page 118.

3.17.4 NVDATAFILE.SIGNAL

```
Function Signal (index As Integer) As NvSignal
```

Returns the signal with the number **`index`** or **`Nothing`** if it does not exist (see "CLASS "NVSIGNAL"", p. 120").

A list of all available element functions of the class "NvDataFile" is provided in chapter "NvDataFile: Overview" on page 118.

3.17.5 NVDATAFILE.CREATETRAIN

```
Function CreateTrain () As NvDataTrain
```

This command is used to convert a measuring file into a train file. Other suitable measuring files can be connected to the train file with the command **`NvDataTrain.Dock`** (see "CLASS "NVDATATRAIN"", p. 135).

A list of all available element functions of the class "NvDataFile" is provided in chapter "NvDataFile: Overview" on page 118.

3.18 CLASS "NVSIGNAL"

An object of the **NextView®4** class "NvSignal" cannot be created with the command **`New`**. It is generated by the function **`NvDataFile.Signal`** of the class "NvDataFile" or **`NvGraphDisplay.Signal`** of the class "NvGraphDisplay" instead.

```
' opens the measuring file "c:\data.lfx", get the first signal
' and outputs its name in the message display

Dim o As NvOpenDataFile
Set o = New NvOpenDataFile
o.Path = "c:\data.lfx"
Dim file As NvDataFile
Set file = o.Open

If file Is Nothing Then
  Print "Cannot open data.lfx"
Else
  Dim sig As NvSignal
  Set sig = file.Signal(1)

  If sig Is Nothing Then
    Print "Cannot open Signal 1"
  Else
    Print "Signal 1 is " & sig.Name
  End If
End If
```

3.18.1 NVSIGNAL: OVERVIEW

Element function	Description
`NvSignal.Name`	set or read signal name
`NvSignal.File`	returns the corresponding data file as an object of the class "NvDataFile"
`NvSignal.Group`	set or read signal group
`NvSignal.Comment`	set or read signal comment
`NvSignal.Type`	outputs if signal is analog or digital
`NvSignal.Samples`	returns the number of measuring values of a signal
`NvSignal.Prehist`	returns the number of measuring values provided by the prehistory of a signal
`NvSignal.Posthist`	returns the number of measuring values provided by the posthistory of a signal
`NvSignal.Timestamp`	scan start in PC local time using Windows® date/time format
`NvSignal.xStart`	set or read signal start
`NvSignal.xResolution`	set or read the x-resolution (time between point of sampling)
`NvSignal.xUnit`	set or read unit of the signal's x-axis
`NvSignal.yMin`	set or read bottom value displayed by the y-axis
`NvSignal.yMax`	set or read top value displayed by the y-axis
`NvSignal.yRangeMin`	set or read bottom value of the measuring range
`NvSignal.yRangeMax`	set or read top value of the measuring range
`NvSignal.yUnit`	set or read unit of the signal's y-axis
`NvSignal.yUsing`	set or read output format of the signal's y-values
`NvSignal.Value`	set or read signal value
`NvSignal.ValueAt`	set or read a signal value at point x

3.18.2 NVSIGNAL.NAME

```
' read name:
  Function Name () As String
' set name:
  Name = newName      ' newName As String
```

Specifies or returns the signal name.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old name and sets the new name for the 1. signal
Print sig.Name
sig.Name = "My signal"
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.3 NVSIGNAL.FILE

```
Function File () As NvDataFile
```

The recorded signal file is returned as an object of the class "NvDataFile".

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.4 NVSIGNAL.GROUP

```
' read group:
  Function Group () As String
' set group:
  Group = newGroup      ' newGroup As String
```

Defines or returns the signal group.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old name and sets the new name for the 1. signal
Print sig.Group
sig.Group = "My group"
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.5 NVSIGNAL.COMMENT

```
' read comment:
  Function Comment () As String
' set comment:
  Comment = newComment     ' newComment As String
```

Sets or returns the signal comment.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old name and sets the new name for the 1. signal
Print sig.Comment
sig.Comment = "My comment for signal"
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.6 NVSIGNAL.TYPE

```
Function Type () As Integer
```

Outputs the signal type. The following values will be returned:

flags	Description
nvSfAnalog	signal is analog
nvSfDigital	signal is digital

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

If sig.Type = sfAnalog Then
  Print "Signal 1 is analog"
Else
  Print "Signal 1 is digital"
End If
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.7 NVSIGNAL.SAMPLES

```
Function Samples () As Integer
```

Returns the number of signal samples.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

Print "Signal 1 contains " & sig.Samples & " samples"
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.8 NVSIGNAL.PREHIST

```
Function Prehist () As Integer
```

Returns the number of samples belonging to the signal's prehistory.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

Print "Signal 1 contains " & sig.Prehist
Print " samples as prehistory"
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.9 NVSIGNAL.POSTHIST

```
Function Posthist () As Integer
```

Returns the number of samples belonging to the signal's posthistory.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

Print "Signal 1 contains " & sig.Posthist
Print " samples as posthistory"
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.10 NVSIGNAL.TIMESTAMP

```
Function Timestamp () As Double
```

Returns the time of the scan start. The value is written in PC local time and can be converted to UTC time with the function **`LocalToSystemTime`**.

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.11 NVSIGNAL.XSTART

```
' read time of signal start:
  Function xStart () As Double
' set time of signal start:
  xStart = newStart       ' newStart As Double
```

Defines or outputs the signal start. The value is written in seconds (e.g. 1.2sec).

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' read old signal start first, then set the new one for the 1. signal
Print sig.xStart
sig.xStart = 0.0 ' s
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.12 NVSIGNAL.XRESOLUTION

```
' read x-resolution:
  Function xResolution () As Double
' set x-resolution:
  xResolution = newResolution   ' newResolution As Double
```

Specifies or returns the x-resolution of the signal, i.e. generally the time in seconds between the samples.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old x-resolution first and then sets the new one for
' the 1. signal
Print sig.xResolution
sig.xResolution = 0.01 ' set to 10 msec
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.13 NVSIGNAL.XUNIT

```
' read unit of the x-values:
  Function xUnit () As String
' set unit of the x-values:
  xUnit = newUnit         ' newUnit As String
```

Specifies o returns the signal's x-unit.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old x-unit first and then sets the new one for
' the 1. signal
```

```
Print sig.xUnit
sig.xUnit = "sec"
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.14 NVSIGNAL.YMIN

```
' read lower bound of the y-range:
  Function yMin () As Double
' set lower bound of the y-range:
  yMin = newMin           ' newMin As Double
```

Defines or returns the minimum value for the default display range of the y-axis in a graph display.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old minimum first and then sets the new one for
' the 1. signal
Print sig.yMin
sig.yMin = -1.0
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.15 NVSIGNAL.YMAX

```
' read upper bound of the y-range:
  Function yMax () As Double
' set upper bound of the y-range:
  yMax = newMax           ' newMax As Double
```

Defines or returns the maximum value for the default display range of the y-axis in a graph display.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old maximum first and then sets the new one for
' the 1. signal
Print sig.yMax
sig.yMax = 1.0
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.16 NVSIGNAL.YRANGEMIN

```
' read bottom value of the measuring range:
  Function yRangeMin () As Double
' set bottom value of the measuring range:
  yRangeMin = newRangeMin   ' newRangeMin As Double
```

Defines or returns the minimum value for the measuring range of the DAQ system.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old minimum first and then sets the new one for
' the 1. signal
Print sig.yRangeMin
sig.yRangeMin = -10.0
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.17 NVSIGNAL.YRANGEMAX

```
' read top value of the measuring range:
  Function yRangeMax () As Double
' set top value of the measuring range:
  yRangeMax = newRangeMax   ' newRangeMax As Double
```

Defines or returns the maximum value for the measuring range of the DAQ system.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old maximum first and then sets the new one for
' the 1. signal
Print sig.yRangeMax
sig.yRangeMax = 10.0
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.18 NVSIGNAL.YUNIT

```
' read unit of the y-values:
  Function yUnit () As String
' set unit of the y-values:
  yUnit = newUnit         ' newUnit As String
```

Specifies o returns the signal's y-unit.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old y-unit first and then sets the new one for
' the 1. signal
Print sig.xUnit
sig.yUnit = "°C"
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.19 NVSIGNAL.YUSING

```
Function yUsing As NvUsing
```

Defines or returns the using of the y-values (e.g. exponential, scientific, see "CLASS "NVUSING"", p. 132).

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' returns the old using first and then changes it for the 1. signal
Print sig.yUsing
sig.yUsing.Width = 9
sig.yUsing.Fraction = 5
sig.yUsing.Type = nvUsingFixed
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.20 NVSIGNAL.VALUE

```
' set value:
  Function Value (pos As Integer) As Double
' read value:
  Value (pos) = newValue   'newValue As Double
```

Sets or returns a signal value at a certain position **pos**. The variable **pos** is between 1 and the number of samples.

```
' get signal
Dim sig As NvSignal
Set sig = file.Signal(1)
...

' sets sample no. 3
sig.Value(3) = 0

' outputs the first 5 samples
Dim i As Integer

For i = 1 To 5
  Print sig.Value(i)
Next
```

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.18.21 NVSIGNAL.VALUEAT

```
' read value:
  Function ValueAt (xPos As Double, Optional isDate as
                    Boolean = False) As Double
' set value:
  ValueAt (xPos, isDate) = newValue   'newValue As Double
```

Sets or returns a signal value at a certain point in time **xPos**. The value of the position **xPos** can be entered in data/time format (absolute time; **isDate** = **True**) or in seconds (relative time; **isDate** = **False**).

A list of all available element functions of the class "NvSignal" is provided in chapter "NvSignal: Overview" on page 121.

3.19 CLASS "NVUSING"

An object of the **NextView®4** class "NvUsing" cannot be created with the command **`New`**. It is generated by the function **`NvSignal.yUsing`** of the class "NvSignal" instead.

```
Dim o As NvOpenDataFile
set o = New NvOpenDataFile

Dim file As NvDataFile
Dim i As Integer
Set file = o.Open ("c:\data\test-1-1.lfx", nvSFReadWrite)

If file Is Nothing Then
  Print "Cannot open lfx"
Else
  Dim sig As NvSignal
  Set sig = file.Signal(1)

  If sig Is Nothing Then
    Print "Cannot open signal 1"
  Else
    Print "The name of signal 1 " & sig.Name
    Print "old yUsing: " & sig.yUsing

    ' new using in exponential format
    sig.yUsing.Width = 12
    sig.yUsing.Fraction = 3
    sig.yUsing.Type = nvUsingEngineering

    Print "new yUsing: " & sig.yUsing
 End If
End If
```

3.19.1 NVUSING: OVERVIEW

Element function	Description
`NvUsing.Width`	field width: number of digits per numerical value
`NvUsing.Type`	type of using
`NvUsing.Fraction`	number of decimal places
`NvUsing.Print`	prints a string describing the using

3.19.2 NVUSING.WIDTH

```
' read field width:
  Function Width () As Integer
' set field width:
  Width = newWidth        'newWidth as Integer
```

Specifies or returns the number of digits per numerical value, which is the y-value for a signal. One digit is reserved for the sign and one for the decimal point.

A list of all available element functions of the class "NvUsing" is provided in chapter "NvUsing: Overview" on page 133.

3.19.3 NVUSING.TYPE

```
' read using:
  Function Type () As Integer
' set using:
  Type = newType        'newType as Integer
```

Specifies or returns the presentation of the signal's y-values. The following values are provided. The example assumes a width (**`NvUsing.Width`**) of 9 digits including 4 decimal places (**`NvUsing.Fraction`**):

val	Description	Example 123.456 mm
`nvUsingDecimal`	decimal format	123.456 mm
`nvUsingHex`	hexadecimal format	7B mm
`nvUsingFixed`	fixed format	123.4560 mm
`nvUsingEngineering`	exponential format	1.2E+002 mm
`nvUsingScientific`	scientific format	12.3456cm
`nvUsingFixedScientific`	scientific format with fixed unit	123.456 mm
`nvUsingTime`	time format	e.g. 2min 3.5sec (basis unit: seconds)
`nvUsingDate`	date format	<day>.<month>.<year>

A list of all available element functions of the class "NvUsing" is provided in chapter "NvUsing: Overview" on page 133.

3.19.4 NVUSING.FRACTION

```
' read number of decimal places:
  Function Fraction () As Integer
' set number of decimal places:
  Fraction = newFraction    'newFraction as Integer
```

Configures the number of decimal places.

A list of all available element functions of the class "NvUsing" is provided in chapter "NvUsing: Overview" on page 133.

3.19.5 NVUSING.PRINT

```
Function Print () As String
```

Prints the set parameters of the using (**`NvUsing.Type`**, **`NvUsing.Width`**, **`NvUsing.Fraction`**).

A list of all available element functions of the class "NvUsing" is provided in chapter "NvUsing: Overview" on page 133.

3.20 CLASS "NVDATATRAIN"

An object of the **NextView®4** class "NvDataTrain" cannot be created with the command **New**. It is generated by the function **NvDataFile.CreateTrain** of the class "NvDataFile" instead. The **NextView®4** class "NvDataTrain" derives from the class "NvDataFile" and all commands of the class NvDataFile (see "NvDataFile: Overview", p. 118) are also valid for an object of the class "NvDataTrain".

3.20.1 NVDATATRAIN: OVERVIEW

Element function	Description
NvDataTrain.Dock	adds a signal file to a train
NvDataTrain.Undock	removes a signal file from a train

3.20.2 NVDATATRAIN.DOCK

```
Function Dock (wagon as NvDataFile) As Boolean
```

Adds the passed signal file to an existing train. If the call was successful, **True** will be returned.

```
Dim f, ftmp as NvDataFile
Dim t as NvDataTrain
Dim i as Integer
Dim str as String
' open first file
str = "c:\data\Test-1-1.lfx"
Set f = New NvOpenDataFile.Open (str, nvSfReadOnly)
If f Is Nothing Then
  Print "error when opening " & str
  Exit Sub
End If

' first file pulls the train (t)
set t = f.CreateTrain
```



```
' add test-1-2.lfx and test-1-3.lfx
for i = 2 to 3
  str = "c:\data\Test-1-" & i & ".lfx"
  Set ftmp = New NvOpenDataFile.Open (str, nvSfReadOnly)
  if Not(ftmp is Nothing) then
    if Not t.Dock(ftmp) then
      Print "Error when docking " & str
    end if
  else
    Print "Error when opening" & str
  end if
next

print "Number of samples test-1-1.lfx: " & f.Signal(1).Samples
print "Number of samples Train (Test-1-1+2+3.lfx): " & _
       t.Signal(1).Samples
```

A list of all available element functions of the class "NvDataTrain" is provided in chapter "NvDataTrain: Overview" on page 136.

3.20.3 NVDATATRAIN.UNDOCK

```
Function Undock (idx as Integer) as Boolean
```

Removes the signal file number **`idx`** from an existing train. The passed index range is between 1 and the number of signal files in the train. If the call was successful, **`True`** will be returned.

A list of all available element functions of the class "NvDataTrain" is provided in chapter "NvDataTrain: Overview" on page 136.

3.21 CLASS "NVFFT"

An object of the **NextView®4** class "NvFFT" is created with the command **New**.

```
Dim s As NvSheet
Dim d As NvGraphDisplay

' short check if script is working
If NvCurrentProject.SheetCount < 1 Then Exit Sub

Set s = NvCurrentProject.Sheet (1)
If s.DisplayCount < 1 Then Exit Sub

Set d ?= s.Display (1)
If d.SignalCount < 3 Then Exit Sub

Dim fft As NvFFT
Set fft = New NvFFT

' preparations for the FFT

fft.Add d.Signal(1), 0, nvFftWindowNone, _
        nvFftLines128, nvFftResultsMag, False
fft.Add d.Signal(2), 0, nvFftWindowHanning, _
        nvFftLines512, nvFftResultsComplex, False
fft.Add d.Signal(3), 0, nvFftWindowHamming, _
        nvFftLines128, nvFftResultsMagPhase, True

Dim f As NvDataFile
' run FFT
Set f = fft.Run ("C:\fft.lfx")
```

3.21.1 NVFFT: OVERVIEW

Element function	Description
NvFFT.Add	adds a signal to the list used for the FFT
NvFFT.Run	runs an FFT analysis
NvFFT.Reset	removes all signals from the FFT list

3.21.2 NVFFT.ADD

```
Sub Add (sig As NvSignal, xStart As Double,
         fftWindow As Integer, fftLines As Integer,
         fftResults As Integer, fftLog As Boolean)
```

Adds the signal **`sig`** to the list of signals to be analyzed by an FFT. **`NvSignal.xStart`** describes the start time, **`fftLog`** if the result is logarithmic.

The following tables show the constants for **`fftWindow`**, **`fftLines`**, and **`fftResults`**. Please, use only these constants.

FFT window	Description
`nvFftWindowNone`	without window
`nvFftWindowRoundLast`	Round Last Point
`nvFftWindowHanning`	Hanning window
`nvFftWindowHamming`	Hamming window
`nvFftWindowBlackman`	Blackman window
`nvFftWindowBartlet`	Bartlet window

FFT lines	Description
`nvFftLines64`	64 lines
`nvFftLines128`	128 lines
`nvFftLines256`	256 lines
`nvFftLines512`	512 lines
`nvFftLines1024`	1024 lines
`nvFftLines2048`	2048 lines
`nvFftLines4096`	4096 lines
`nvFftLines8192`	8192 lines

FFT result	Description
nvFftResultsMag	magnitude
nvFftResultsMagPhase	magnitude & phase
nvFftResultsPower	power spectrum
nvFftResultsComplex	complex result

A list of all available element functions of the class "NvFFT" is provided in chapter "NvFFT: Overview" on page 138.

3.21.3 NVFFT.RUN

```
Function Run (path As String,
              Optional askOverwrite As Boolean = True)
              As NvDataFile
```

Runs an FFT for the signals added to the list by **NvFFT.Add** and saves the result in the file **path**. An empty list causes a runtime error. If the file name **path** and the name of a signal file used are the same, a runtime error will be caused, too.

The option **askOverwrite** defines if a query is displayed before an existing file is overwritten. If **askOverwrite** is **False**, the file will be overwritten without further request.

A list of all available element functions of the class "NvFFT" is provided in chapter "NvFFT: Overview" on page 138.

3.21.4 NVFFT.RESET

```
Sub Reset ()
```

Resets the FFT list of signals, i.e. deletes all items.

A list of all available element functions of the class "NvFFT" is provided in chapter "NvFFT: Overview" on page 138.

3.22 CLASS "NVINTEGRATION"

An object of the **NextView®4** class "NvIntegration" is created with the command **New**.

```
Dim s As NvSheet
Dim d As NvGraphDisplay

' short check if script is working
If NvCurrentProject.SheetCount < 1 Then Exit Sub

Set s = NvCurrentProject.Sheet (1)
If s.DisplayCount < 1 Then Exit Sub

Set d = s.Display (1)
If d.SignalCount < 3 Then Exit Sub

Dim int As NvIntegration
Set int = New NvIntegration

' preparations for integration
int.Add d.Signal(1), 0, 1
int.Add d.Signal(2), 0, 1, False, 1, True, False, 1
int.Add d.Signal(3), 0, 1, True, 0, True

Dim f As NvDataFile
' run integration
Set f = int.Run ("C:\integration.lfx")
```

3.22.1 NVINTEGRATION: OVERVIEW

Element function	Description
`NvIntegration.Add`	adds a signal to the list used for integration
`NvIntegration.Run`	runs an integration
`NvIntegration.Reset`	removes all signals from the integration list

3.22.2 NVINTEGRATION.ADD

```
Sub Add (sig As NvSignal, xStart As Double,
         xEnd As Double,
         Optional intAutoOffset As Boolean = True,
         Optional intOffset As Double = 0.0,
         Optional int2nd As Boolean = False,
         Optional intAutoOffset2 As Boolean = True,
         Optional intOffset2 As Double = 0.0)
```

Adds the signal **sig** to the list of signals to be analyzed. **xStart** describes the start time, **xEnd** the end. If **int2nd** is set to **True**, a 2. integral is calculated (i.e. integral from the integral). **intAutoOffset** and **intAutoOffset2** define if the signal offset is calculated automatically or if **intOffset** and **intOffset2** is used.

A list of all available element functions of the class "NvIntegration" is provided in chapter "NvIntegration: Overview" on page 141.

3.22.3 NVINTEGRATION.RUN

```
Function Run (path As String) As NvDataFile
```

Runs an integration for the signals added to the list by **NvIntegration.Add** and saves the result in the file **path** (see "CLASS "NVDATAFILE"", p. 118).

An empty list causes a runtime error. If the file name **path** and the name of a signal file used are the same, a runtime error will be caused, too.

The option **askOverwrite** defines if a query is displayed before an existing file is overwritten. If **askOverwrite** is **False**, the file will be overwritten without further request.

A list of all available element functions of the class "NvIntegration" is provided in chapter "NvIntegration: Overview" on page 141.

3.22.4 NVINTEGRATION.RESET

```
Sub Reset ()
```

Resets the integration list of signals, i.e. deletes all items.

A list of all available element functions of the class "NvIntegration" is provided in chapter "NvIntegration: Overview" on page 141.

3.23 CLASS "NVDIFFERENTIATION"

An object of the **NextView®4** class "NvDifferentiation" is created with the command **New**.

```
Dim s As NvSheet
Dim d As NvGraphDisplay

' short check if script is working
If NvCurrentProject.SheetCount < 1 Then Exit Sub

Set s = NvCurrentProject.Sheet (1)
If s.DisplayCount < 1 Then Exit Sub

Set d ?= s.Display (1)
If d.SignalCount < 3 Then Exit Sub

Dim diff As NvDifferentiation
Set diff = New NvDifferentiation

' preparations for differentiation

diff.Add d.Signal(1), 0, 1, 5
diff.Add d.Signal(2), 0, 1, 3
diff.Add d.Signal(3), 0, 1, 7

Dim f As NvDataFile
' run differentiation
Set f = diff.Run ("C:\diff.lfx")
```

3.23.1 NVDIFFERENTIATION: OVERVIEW

Element function	Description
`NvDifferentiation.Add`	adds a signal to the list used for differentiation
`NvDifferentiation.Run`	runs a differentiation
`NvDifferentiation.Reset`	removes all signals from the differentiation list

3.23.2 NVDIFFERENTIATION.ADD

```
Sub Add (sig As NvSignal, xStart As Double,
         xEnd As Double, int diffArea)
```

Adds the signal **sig** to the list of signals to be analyzed. **xStart** describes the start time, **xEnd** the end. If **int2nd** is set to **True**, a 2. integral is calculated (i.e. integral from the integral). **intAutoOffset** and **intAutoOffset2** define if the signal offset is calculated automatically or if **intOffset** and **intOffset2** are used.

diffArea specifies the number of samples in the area used for differentiation.

A list of all available element functions of the class "NvDifferentiation" is provided in chapter "NvDifferentiation: Overview" on page 144.

3.23.3 NVDIFFERENTIATION.RUN

```
Function Run (path As String,
              Optional askOverwrite As Boolean = True)
              As NvDataFile
```

Runs a differentiation for the signals added to the list by **NvDifferentiation.Add** and saves the result in the file **path** (see "CLASS "NVDATAFILE"", p. 118).

An empty list causes a runtime error. If the file name **path** and the name of a signal file used are the same, a runtime error will be caused, too.

The option **askOverwrite** defines if a query is displayed before an existing file is overwritten. If **askOverwrite** is **False**, the file will be overwritten without further request.

A list of all available element functions of the class "NvDifferentiation" is provided in chapter "NvDifferentiation: Overview" on page 144.

3.23.4 NVDIFFERENTIATION.RESET

```
Sub Reset ()
```

Resets the differentiation list of signals, i.e. deletes all items.

A list of all available element functions of the class "NvDifferentiation" is provided in chapter "NvDifferentiation: Overview" on page 144.

3.24 CLASS "NVFILTER"

An object of the **NextView®4** class "NvFilter" is created with the command **New**.

```
Dim s As NvSheet
Dim d As NvGraphDisplay

' short check if script is working
If NvCurrentProject.SheetCount < 1 Then Exit Sub

Set s = NvCurrentProject.Sheet (1)
If s.DisplayCount < 1 Then Exit Sub

Set d ?= s.Display (1)
If d.SignalCount < 3 Then Exit Sub

Dim filter As NvFilter
Set filter = New NvFilter

' preparations for filter

filter.Add d.Signal(1), 0, 1, nvFilterCriticalDamp, _
           nvFilterOrder2, nvFilterLowPass, 10
filter.Add d.Signal(2), 0, 1, nvFilterChebycheff30, _
           nvFilterOrder6, nvFilterBandPass, 5
filter.Add d.Signal(3), 0, 1, nvFilterRunningMean, 0, 0, 20

Dim f As NvDataFile
' run filter
Set f = filter.Run ("C:\filter.lfx")
```

3.24.1 NVFILTER: OVERVIEW

Element function	Description
`NvFilter.Add`	adds a signal to the list the filter is applied to
`NvFilter.Run`	runs a filter
`NvFilter.Reset`	removes all signals from the filter list

3.24.2 NVFILTER.ADD

```
Sub Add (sig As NvSignal, xStart As Double,
         xEnd As Double, filterType As Integer,
         filterOrder As Integer, filterPass As Integer,
         filterCutoff As Double)
```

Adds the signal **sig** to the list of signals to be analyzed. **xStart** describes the start time, **xEnd** the end.

The parameter **filterCutoff** defines the filter cut-off frequency for band-pass filters. If using a running mean or median filter, **filterCutoff** is the number of samples around the relevant sample used for calculation.

filterType	Description
nvFilterCriticalDamp	critical damping
nvFilterBessel	Bessel filter
nvFilterButterworth	Butterworth filter
nvFilterChebycheff05	Tschebyscheff filter, 0.5db
nvFilterChebycheff10	Tschebyscheff filter, 1.0db
nvFilterChebycheff20	Tschebyscheff filter, 2.0db
nvFilterChebycheff30	Tschebyscheff filter, 3.0db
nvFilterRunningMean	running mean filter
nvFilterRunningMedian	running median filter
nvFilterNothing	signal data are written to the output file without filtering

filterOrder	Description
nvFilterOrder2	2. order
nvFilterOrder4	4. order
nvFilterOrder6	6. order
nvFilterOrder8	8. order

filterPass	Description
nvFilterLowPass	low-pass
nvFilterHighPass	high-pass
nvFilterBandPass	band-pass
nvFilterBandElimination	band-elimination
nvFilterByPass	does a heart surgery

A list of all available element functions of the class "NvFilter" is provided in chapter "NvFilter: Overview" on page 147.

3.24.3 NVFILTER.RUN

```
Function Run (path As String,
              Optional askOverwrite As Boolean = True)
              As NvDataFile
```

Applies a filter to the signals added to the list by **NvFilter.Add** and saves the result in the file **path** (see "CLASS "NVDATAFILE"", p. 118). An empty list causes a runtime error. If the file name **path** and the name of a signal file used are the same, a runtime error will be caused, too. The option **askOverwrite** defines if a query is displayed before an existing file is overwritten. If **askOverwrite** is **False**, the file will be overwritten without further request.

A list of all available element functions of the class "NvFilter" is provided in chapter "NvFilter: Overview" on page 147.

3.24.4 NVFILTER.RESET

```
Sub Reset ()
```

Resets the filter list of signals, i.e. deletes all items.

A list of all available element functions of the class "NvFilter" is provided in chapter "NvFilter: Overview" on page 147.

3.25 CLASS "NVREDUCTION"

An object of the **NextView®4** class "NvReduction" is created with the command **New**.

```
Dim s As NvSheet
Dim d As NvGraphDisplay

' short check if script is working
If NvCurrentProject.SheetCount < 1 Then Exit Sub

Set s = NvCurrentProject.Sheet (1)
If s.DisplayCount < 1 Then Exit Sub

Set d ?= s.Display (1)
If d.SignalCount < 3 Then Exit Sub

Dim reduce As NvReduction
Set reduce = New NvReduction

' preparations for reduction

reduce.Add d.Signal(1), 0, 1, nvReductionMin, 10
reduce.Add d.Signal(2), 0, 1, nvReductionAverage
reduce.Add d.Signal(3), 0, 1, nvReductionRMS, 3

Dim f As NvDataFile
' run reduction
Set f = reduce.Run ("C:\reduction.lfx")
```

3.25.1 NVREDUCTION: OVERVIEW

Element function	Description
`NvReduction.Add`	adds a signal to the list used for reduction
`NvReduction.Run`	runs a reduction
`NvReduction.Reset`	removes all signals from the reduction list

3.25.2 NVREDUCTION.ADD

```
Sub Add (sig As NvSignal, xStart As Double,
         xEnd As Double, reduceType As Integer,
         Optional reduceRatio As Integer = 1)
```

Adds the signal **sig** to the list of signals used for data reduction. **xStart** describes the start time, **xEnd** the end.

reduceRatio specifies the number of samples the reduction is applied to.

reduceType	Description
nvReductionMin	saves the minimum of the signal values for every other interval
nvReductionMax	saves the maximum of the signal values for every other interval
nvReductionAverage	saves the average of the signal values for every other interval
nvReductionRMS	saves the r.m.s. of the signal values for every other interval

A list of all available element functions of the class "NvReduction" is provided in chapter "NvReduction: Overview" on page 150.

3.25.3 NVREDUCTION.RUN

```
Function Run (path As String,
              Optional askOverwrite As Boolean = True)
              As NvDataFile
```

Runs a reduction for to the signals added to the list by **`NvFilter.Add`** and saves the result in the file **`path`** (see "CLASS "NVDATAFILE"", p. 118).

An empty list causes a runtime error. If the file name **`path`** and the name of a signal file used are the same, a runtime error will be caused, too.

The option **`askOverwrite`** defines if a query is displayed before an existing file is overwritten. If **`askOverwrite`** is **`False`**, the file will be overwritten without further request.

A list of all available element functions of the class "NvReduction" is provided in chapter "NvReduction: Overview" on page 150.

3.25.4 NVREDUCTION.RESET

```
Sub Reset ()
```

Resets the reduction list of signals, i.e. deletes all items.

A list of all available element functions of the class "NvReduction" is provided in chapter "NvReduction: Overview" on page 150.

4 STANDARD COMMANDS

4.1 STANDARD FUNCTIONS / PROCEDURES

4.1.1 STANDARD FUNCTIONS / PROC.: OVERVIEW

Function	Description
`Print`	prints the specified string in the message display of **NextView®4**
`Timer`	returns the number of seconds since midnight
`TickCount`	returns the number of milliseconds since the start of Windows®
`TimestampStr`	converts the scan start of a signal (timestamp) into a string
`Format`	converts a number into a formatted string
`Hex`	returns a number as a string in hexadecimal format
`LCase`	returns a lower-case string
`UCase`	returns an upper-case string
`Asc`	returns the ASCII code of the first letter of a string
`Chr`	returns the character of the specified ASCII code
`Tab`	returns string with tabs
`Spc`	returns string with spaces
`Len`	returns the length of a string
`Left`	returns a certain number of characters of a string beginning from the left
`Right`	returns a certain number of characters of a string beginning from the right
`Mid`	returns a certain number of characters of a string beginning from the middle
`InStr`	searches a string within a string
`Date`	returns date as string
`Time`	returns time as string

Function	Description
Now	returns current date and time as string
LBound	returns the lower index value of an array
UBound	returns the upper index value of an array
GetDim	returns the dimension of an array
Sleep	waits for a certain time
Randomize	initializes a random function
Rnd	returns a random value
Int	returns the integer part of a number
MsgBox	shows message box
InputBox	returns the string of an input box
Import	imports standard class modules into **NextView®4 Script**
Include	integrates the stated script in **NextView®4 Script** when compiling
System	processes a system command and returns the state of the instruction execution
quote	quotes a string
RGBColor	returns an RGB color value
IsNumber	checks if a string is a number and converts it into a double type when indicated
LocalTime	returns the local time in Windows® date/time format
SystemTime	returns the system time in Windows® date/time format
SystemToLocalTime	converts the system time into local time
LocalToSystemTime	converts the local time into system time
GetDate	read the date
GetTime	read the time
DateValue	converts a date as string into Windows® date format
TimeValue	converts a time as string into Windows® time format
DateTimeValue	converts a date and time indication as string into Windows® date/time format
GetDateFormat	converts a Windows® date into a date string optionally specifying certain date formats

Function	Description
GetTimeFormat	converts a Windows® time into a time string optionally specifying certain time formats
DateSerial	converts the specified year, month, day into Windows® date format
TimeSerial	converts the specified hour, minute, second into Windows® time format
ExtractDate	extracts year, month, day from a Windows® date
ExtractTime	extracts hour, minute, second, millisecond from a Windows® date
GetYear	returns the year from a date/time indication
GetMonth	returns the month from a date/time indication
GetDay	returns the day from a date/time indication
GetHour	returns the hour from a date/time indication
GetMinute	returns the minute from a date/time indication
GetSecond	returns the second from a date/time indication
GetMillisecond	returns the millisecond from a date/time indication
GDN	converts day, month, year into the number of days in the Gregorian calendar
IsNaN	checks the validity of the measuring value

4.1.2 PRINT

```
Sub Print (str As String)
```

Outputs the string **`str`** at the standard printout. The standard printout in **NextView®4 Script** is the message display, which can be inserted on a sheet in **NextView®4** via the menu item "Display".

Print instructions of a script will not be shown in the message display until the script is finished.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.3 TIMER

```
Function Timer () As Integer
```

Returns the number of seconds in Coordinated Universal Time (UTC) passed since midnight.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.4 TICKCOUNT

```
Function TickCount () As Integer
```

Returns the number of milliseconds passed since the start of Windows®.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.5 TIMESTAMPSTR

```
Function TimeStampStr (time as Double) As String
```

Converts the time **time** at scan start into a string using the following format: **DD.MM.YYYY hh:mm:ss.sss**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.6 FORMAT

```
Function Format (x As Double, fmt As String) As String
```

Returns the number **x** as formatted string according to the options in **fmt**.

The following characters have a special meaning in **fmt**:

Symbol	Description
#	replacement character, substituted by a space if the number is not big enough
0	replacement character, substituted by 0 if the number is not big enough
,	replacement character, substituted by the country-specific thousand separator
.	indicates the start of the decimal places replacing the country-specific separator
+ or **-**	a preceding + or - forces the output of the sign

```
Dim d As Double

  d = 1234.1234

  Print Format (d, "0.00")     ' the output is "1234.12"
  Print Format (d, "0.0#")     ' the output is "1234.12"
  Print Format (d, "0.##")     ' the output is "1234.12"

  Print Format (d, "+000000")  ' the output is "+001234"
  Print Format (d, "X######")  ' the output is "X  1234"

  Print Format (d, "0,000.00") ' the output is "1,234.00"
  Print Format (d, "##test##") ' the output is "12test34"

  Print Format (1000, "#,###") ' the output is "1,000"
  Print Format (1, "#,###")    ' the output is "    1"

  Print Format (d, "-000000")  ' the output is "+001234"
  Print Format (d, "X######")  ' the output is "X  1234"
```

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.7 HEX

```
Function Hex (x As Double) As String
```

Returns the number **x** as string in hex format.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.8 LCASE

```
Function LCase (str As String) As String
```

Returns the string **str** in lowercase letters.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.9 UCASE

```
Function UCase (str As String) As String
```

Returns the string **str** in uppercase letters.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.10 ASC

```
Function Asc (str As String) As Integer
```

Returns the ASCII code of the first character in a string **str**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.11 CHR

```
Function Chr (asc As Integer) As String
```

Returns the character described by the ASCII code **asc**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.12 TAB

```
Function Tab (n As Integer) As String
```

Returns a string with **n** tabulators.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.13 SPC

```
Function Spc (n As Integer) As String
```

Returns a string with **n** space characters.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.14 LEN

```
Function Len (str As String) As Integer
```

Returns the number (length) of characters belonging to a string **str**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.15 LEFT

```
Function Left (str As String, n As Integer) As String
```

Returns the first **n** characters in a string **str**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.16 RIGHT

```
Function Right (str As String, n As Integer) As String
```

Returns the last **n** characters in a string **str**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.17 MID

```
Function Mid (str As String, pos As Integer, _
              n As Integer) As String
```

Returns the following **n** characters starting from the position **pos** in the string **str**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.18 INSTR

```
Function InStr (str as String, search as String) as Integer
```

The function **InStr** starts a case-sensitive search for the first existing string **search** in the string **str**. As soon as **search** has been found, the position of the first character in **str** is returned. If **search** does not exist, 0 will be returned.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.19 DATE

```
Function Date () As String
```

Returns the current date as string using the country-specific format.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.20 TIME

```
Function Time () As String
```

Returns the current time as string using the country-specific format.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.21 NOW

```
Function Now () As String
```

Returns the current date and time as string using the country-specific format.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.22 LBOUND

```
Function LBound (Array[, dim As Integer]) As Integer
```

Returns the lower index bound of the **dim**-th dimension of an array.

```
Dim a(1 To 10) As Integer
Dim b(2 To 11,3 To 12) As Integer

Print LBound(b, 2)          ' the output is 3
Print LBound(a)             ' the output is 1
```

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.23 UBOUND

```
Function UBound (Array[, dim As Integer]) As Integer
```

Returns the upper index bound of the **dim**-th dimension of an array.

```
Dim a(1 To 10) As Integer
Dim b(2 To 11,3 To 12) As Integer

Print UBound(b, 2)           ' the output is 12
Print UBound(a)              ' the output is 10
```

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.24 GETDIM

```
Function GetDim (Array) As Integer
```

Returns the dimension of an array.

```
Dim a(1 To 10) As Integer
Dim b(2 To 11,3 To 12) As Integer

Print GetDim(a)           ' the output is 1
Print GetDim(b)           ' the output is 2
```

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.25 SLEEP

```
Sub Sleep (ms As Integer)
```

Keeps the program waiting for approximately **ms** milliseconds.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.26 RANDOMIZE

```
Sub Randomize (seed As Integer)
```

Initializes the random generator with the value **seed**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.27 RND

```
Function Rnd () As Integer
```

Returns the next value of the random generator (range: 0 .. 32767).

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.28 INT

```
Function Int (x As Double) As Integer
```

Returns the integer part of **x** (see "Floor", p. 184).

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.29 MSGBOX

```
Function MsgBox (msg As String[, buttons As Integer, _
                 title As String]) As Integer
```

Opens a dialog box containing the message **msg**. Style definitions are possible via the variable **buttons**. A title can be entered with **title**.

Buttons	Description
mbOkOnly	only show "OK" button
mbOKCancel	show "OK" and "Cancel" buttons
mbAbortRetryIgnore	show "Cancel", "Retry", and "Ignore" buttons
mbYesNoCancel	show "Yes", "No", and "Cancel" buttons
mbYesNo	show "Yes" and "No" buttons
mbRetryCancel	show "Retry" and "Cancel" buttons

The return value provided by **MsgBox** is:

MsgBox output	Description
mbOk	user chose "OK"
mbCancel	user chose "Cancel"
mbAbort	user chose "Abort"
mbRetry	user chose "Retry"
mbIgnore	user chose "Ignore"
mbYes	user chose "Yes"
mbNo	user chose "No"

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.30 INPUTBOX

```
Function InputBox (prompt As String,  title As String, _
                   def As String) As String
```

Opens an input box containing the prompt **prompt** and the title **title**. The default value is defined by **def**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.31 IMPORT

```
Import "Module"
```

Some standard classes must be imported in **NextView®4 Script** first before their procedures and functions can be used in the script.

Module	Description	Import of the standard classes
db	database module	standard class "Database", standard class "Recordset"
os	operating system module	standard class "Stream", standard class "Directory", standard class "Clipboard"
xnf	Windows INI file module	standard class "XNF"

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.32 INCLUDE

```
Include "filename.nvs"
```

This command is used to include the script **filename.nvs** into **NextView®4 Script** during compilation.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.33 SYSTEM

```
Function System(cmd as String) as Integer
```

Runs the system command **cmd** and returns the state of the instruction execution. Some parameters may have to be specified explicitly with the function **quote** in quotes ("").

```
Dim rc as Integer
rc = system(quote("c:\program files\alarm.exe") & " on")
```

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.34 QUOTE

```
Function Quote (str as String) As String
```

Puts quotes around a string **str** ("").

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.35 RGBCOLOR

```
Function RGBColor (red as Integer, green as Integer, blue as
Integer) as Integer
```

Returns the RGB color value containing the colors **red**, **green**, and **blue**. The range is between 0 and 255, 0 being 0% and 255 representing 100% of the relevant color. White, for example, is defined by **RGBColor(255,255,255)** and black by **RGBColor(0,0,0)**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.36 ISNUMBER

```
Function IsNumber(number as String, dbl as Double) as
Boolean
```

Checks if the string **number** is a number and converts it into the variable **dbl**. If **False** is returned, **number** is not a number.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.37 LOCALTIME

```
Function LocalTime () as Double
```

Returns the local time as Windows® date/time.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.38 SYSTEMTIME

```
Function SystemTime () as Double
```

Returns the system time of the PC as Windows® date/time.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.39 SYSTEMTOLOCALTIME

```
Function SystemToLocalTime (vDate as Double) as Double
```

Converts a date from system time to local time.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.40 LOCALTOSYSTEMTIME

```
Function LocalToSystemTime (vDate as Double) as Double
```

Converts a date from local time to system time.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.41 GETDATE

```
Function GetDate (datetime as Double) as Double
```

Returns the date of the **datetime** value.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.42 GETTIME

```
Function GetTime (datetime as Double) as Double
```

Returns the time of the **datetime** value.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.43 DATEVALUE

```
Function DateValue (date as String) as Double
```

Converts a date as string into a Windows® date (e.g. **DateValue ("20.1.2008")**). The passed string must be of the same format used by the PC.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.44 TIMEVALUE

```
Function TimeValue (time as String) as Double
```

Converts a time as string into a Windows® time value (e.g. **`TimeValue ("12:00:01")`**). The passed string must be of the same format used by the PC.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.45 DATETIMEVALUE

```
Function DateTimeValue (date as String) as Double
```

Converts a date as string into a Windows® date (e.g. **`DateTimeValue ("20.1.2008 12:00:01")`**). The passed string must be of the same format used by the PC.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.46 GETDATEFORMAT

```
Function GetDateFormat (vDate as Double,
  Optional format as String) as String
```

Converts a Windows® date into a string. The following format specifications can be added:

Parameter	Description
d	day of the month as a number without a preceding 0 for single-digit days
dd	day of the month as a number with a preceding 0 for single-digit days
ddd	day of the week abbreviated by 3 characters; the function uses the value LOCALE_SABBREVDAYNAME corresponding to the local system settings
dddd	day of the week fully written out; the function uses the value LOCALE_SDAYNAME corresponding to the local system settings
M	month as a number without a preceding 0 for single-digit months
MM	month as a number with a preceding 0 for single-digit months
MMM	month abbreviated by 3 characters; the function uses the value LOCALE_SABBREVMONTHNAME corresponding to the local system settings
MMMM	month fully written out; the function uses the value LOCALE_SMONTHNAME corresponding to the local system settings
y	two-digit year specification without a preceding 0 for single-digit years
yy	two-digit year specification with a preceding 0 for single-digit years
yyyy	four-digit specification of the year

The following example assigns the date "Thu 31.12.2009" to the variable **str**:

```
Dim str as String
str = GetDateFormat(40178,"ddd d.MM.yyyy")
```

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.47 GETTIMEFORMAT

```
Function GetTimeFormat (vTime as Double,
  Optional format as String) as String
```

Converts a Windows® time into a string. The following format specifications can be added:

Parameter	Description
h	hour without a preceding 0 for single-digit hours, 12-hour display
hh	hour with a preceding 0 for single-digit hours, 12-hour display
H	hour without a preceding 0 for single-digit hours, 24-hour display
HH	hour with a preceding 0 for single-digit hours, 24-hour display
m	minute without a preceding 0 for single-digit minutes
mm	minute with a preceding 0 for single-digit minutes
s	seconds without a preceding 0 for single-digit seconds
ss	seconds with a preceding 0 for single-digit seconds
t	one character as additional time information, e.g. A or P
tt	several characters as additional time information, e.g. AM or PM

Milliseconds are not displayed unless the string contains **.sss**.

The following example assigns the time "12:17:42.720" to the variable **str**:

```
Dim str as String
str = GetTimeFormat(0.5123,"hh:mm:ss.sss")
```

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.48 DATESERIAL

```
Function DateSerial (y as Integer, mm as Integer,
  dd as Integer) as Double
```

Converts the specified year **y**, the month **mm**, and the day **dd** into a Windows® date.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.49 TIMESERIAL

```
Function TimeSerial (h as Integer, m as Integer,
  s as Integer, Optional ms as Integer = 0) as Double
```

Converts the specified hour **h**, the minute **m**, and the second **s** into a Windows® time. The number of milliseconds can optionally be passed, too.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.50 EXTRACTDATE

```
Sub ExtractDate (vDate as Double, Byref y as Integer,
  Byref m as Integer, Byref d as Integer)
```

Extracts the year **y**, the month **mm** and the day **dd** from the passed Windows® date **vDate**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.51 EXTRACTTIME

```
Sub ExtractTime (vDate as Double, Byref h as Integer, Byref
m as Integer, Byref s as Integer, Byref ms as Integer)
```

Extracts the hour **h**, the minute **m**,and the second **s** from the passed Windows® time **vDate**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.52 GETYEAR

```
Function GetYear (vDate as Double) as Integer
```

Returns the year of the passed date/time indication **vDate**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.53 GETMONTH

```
Function GetMonth (vDate as Double) as Integer
```

Returns the month of the passed date/time indication **vDate**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.54 GETDAY

```
Function GetDay (vDate as Double) as Integer
```

Returns the day of the passed date/time indication **vDate**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.55 GETHOUR

```
Function GetHour (vDate as Double) as Integer
```

Returns the hour of the passed date/time indication **vDate**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.56 GETMINUTE

```
Function GetMinute (vDate as Double) as Integer
```

Returns the minute of the passed date/time indication **vDate**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.57 GETSECOND

```
Function GetSecond (vDate as Double) as Integer
```

Returns the second of the passed date/time indication **vDate**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.58 GETMILLISECOND

```
Function GetMillisecond (vDate as Double) as Integer
```

Returns the millisecond of the passed date/time indication **vDate**.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.59 GDN

```
Function GDN (d as Integer, m as Integer, y as Integer)
  as Integer
```

Converts day **d**, month **m**, and year **y** into the corresponding number of days in the Gregorian calendar.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.1.60 ISNAN

```
Function IsNaN (double val) as Boolean
```

Checks if a sample **val** contains a valid value.

A list of all available standard functions / procedures is provided in chapter "Standard functions / proc.: Overview" on page 153.

4.2 MATHEMATIC FUNCTIONS

4.2.1 MATHEMATIC FUNCTIONS: OVERVIEW

Function	Description
`Sin`	sine function
`Cos`	cosine function
`Tan`	tangent function
`Sinh`	hyperbolic sine function
`Cosh`	hyperbolic cosine function
`Tanh`	hyperbolic tangent function
`Asin`	arc sine function
`Acos`	arc cosine function
`Atan`	arc tangent function
`Sqrt`	square root function
`Sqr`	square function
`Log`	logarithm function (natural logarithm)
`Exp`	exponential function
`Round`	rounding function
`Floor`	rounding function (down)
`Ceil`	rounding function (up)
`Abs`	absolute function
`Pow`	power function
`Fmod`	floating-point modulo function

4.2.2 SIN

```
Function Sin (angle As Double) As Double
```

Returns the sine of **angle** . The variable **angle** must be declared in radians.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.3 COS

```
Function Cos (angle As Double) As Double
```

Returns the cosine of **angle** . The variable **angle** must be declared in radians.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.4 TAN

```
Function Tan (angle As Double) As Double
```

Returns the tangent of **angle** . The variable **angle** must be declared in radians.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.5 SINH

```
Function Sinh (angle As Double) As Double
```

Returns the hyperbolic sine of **angle** . The variable **angle** must be declared in radians.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.6 COSH

```
Function Cosh (angle As Double) As Double
```

Returns the hyperbolic cosine of **angle** . The variable **angle** must be declared in radians.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.7 TANH

```
Function Tanh (angle As Double) As Double
```

Returns the hyperbolic tangent of **angle** . The variable **angle** must be declared in radians.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.8 ASIN

```
Function Asin (x As Double) As Double
```

Returns the arc sine of **x** in radians between –π/2 and π/2. The variable **x** must be in the range of –1 and 1, otherwise **Asin** will return "NaN" (not a number).

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.9 ACOS

```
Function Acos (x As Double) As Double
```

Returns the arc cosine of **x** in radians between –π/2 and π/2. The variable **x** must be in the range of –1 and 1, otherwise **Acos** will return "NaN" (not a number).

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.10 ATAN

```
Function Atan (x As Double) As Double
```

Returns the arc tangent of **x** in radians between –π/2 and π/2. If **x** is 0, **Atan** will return "NaN" (not a number).

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.11 SQRT

```
Function Sqrt (x As Double) As Double
```

Returns the square root of **x**. If **x** is less than 0, **Sqrt** will return "NaN" (not a number).

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.12 SQR

```
Function Sqr (x As Double) As Double
```

Returns the square of **x**.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.13 LOG

```
Function Log (x As Double) As Double
```

Returns the natural logarithm of **x**. If **x** is less than 0, **Log** will return "NaN" (not a number). If **x** is 0, "INF" (infinite) will be returned.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.14 EXP

```
Function Exp (x As Double) As Double
```

Returns the value of the exponential function of **x**.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.15 ROUND

```
Function Round (x As Double) As Double
```

Returns the rounded, integer value of **x**.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.16 FLOOR

```
Function Floor (x As Double) As Double
```

Returns the rounded down, integer value of **x**.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.17 CEIL

```
Function Ceil (x As Double) As Double
```

Returns the rounded up, integer value of **x**.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.18 ABS

```
Function Abs (angle As Double) As Double
```

Returns the absolute value of **x**.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.19 POW

```
Function Pow (x As Double, y As Double) As Double
```

Returns the value of **x** to the power of **y**.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.2.20 FMOD

```
Function Fmod (x As Double, y As Double) As Double
```

Returns the floating-point modulo of **x / y** with **x = i * y + fmod(x,y)** and **i** being the highest integer for which applies that **x** is greater than **i * y**. Example: **fmod(9.8,5.5) = 4.3**.

A list of all available mathematic functions is provided in chapter "Mathematic functions: Overview" on page 179.

4.3 STANDARD CLASS "DATABASE"

The standard class "Database" provides access to existing databases of the system. An object of the class "Database**"** can be created with the command **New**.

To provide the standard class "Database" in NextView®4 Script, the module database (db) must be imported at the beginning of the script (see "IMPORT", p. 167).

```
' Import Database support at the beginning of the NextView Script
Import "db"
...

Sub Test ()
Dim db As Database

  Set db = New Database

  if db.Open ("DSN=myDSN") <> 0 then
    ...
  End If
End Sub
```

4.3.1 DATABASE: OVERVIEW

Element function	Description
Database.Open	opens a database
Database.Close	closes an open database
Database.Execute	processes an SQL command
Database.OpenRecordset	processes an SQL command and returns the corresponding Recordset

4.3.2 DATABASE.OPEN

```
Function Open (DSN As String) As Integer
```

Opens the database **DSN** (*Data Source Name*). If the database cannot be opened, **Open()** will return 0.

A list of all available element functions of the standard class "Database" is provided in chapter "Database: Overview" on page 186.

4.3.3 DATABASE.CLOSE

```
Sub Close ()
```

Closes an open database.

A list of all available element functions of the standard class "Database" is provided in chapter "Database: Overview" on page 186.

4.3.4 DATABASE.EXECUTE

```
Function Execute (sqlQuery As String) As RecordSet
```

Processes the SQL query **sqlQuery**. In case of an error, **Execute()** will cause a runtime error.

A list of all available element functions of the standard class "Database" is provided in chapter "Database: Overview" on page 186.

4.3.5 DATABASE.OPENRECORDSET

```
Function OpenRecordset (sqlQuery As String, Optional _
                        dbType As Integer) As Recordset
```

Processes the SQL query **sqlQuery** and returns the result as **Recordset** (see "STANDARD CLASS "RECORDSET", p. 189). In case of an error, **OpenRecordset()** will cause a runtime error.

If **dbType** is not set, **OpenRecordset()** will automatically try to set a cursor type. Please note that when choosing a cursor, the values permitted depend on the database settings and on the database used.

dbType	Description
dbNone	does not set a cursor type, the default cursor of the database is used instead
dbDynaset	sets the cursor type to "dynamic", i.e. you can move at will in the Recordset (This option must be possible in the database, otherwise **OpenRecordset** will fail!)
dbForwardOnly	sets the cursor type to "forward-only", i.e. the cursor can only be moved forward with **Recordset.MoveNext**

A list of all available element functions of the standard class "Database" is provided in chapter "Database: Overview" on page 186.

4.4 STANDARD CLASS "RECORDSET"

The standard class "Recordset" provides direct access to data of a database. An object of the class "Recordset" cannot be created with the command **New**. It is returned by Database.OpenRecordset of the standard class "Database" instead.

- **To make the standard class "Recordset" available in NextView®4 Script, the module database (db) must be imported at the beginning of the script (see "IMPORT", p. 167).**
- **All opened Recordsets must be shut before closing or reopening a database.**

```
' Import Database support at the beginning of the NextView Script
Import "db"
...

Sub Test ()
  Dim i As Integer
  Dim db As Database

  Set db = New Database

  if db.Open ("DSN=myDSN") <> 0 then

    Dim rs As Recordset
    Set rs = db.OpenRecordset ("SELECT * FROM Serial_Table")

    For i = 1 To rs.Columns
      Print rs.ColumnName (i)
    Next
' reset all Recordsets before reopening the database
    set rs = Nothing

  Else
    Print "not opened"
  End if
End Sub
```

4.4.1 RECORDSET: OVERVIEW

Element function	Description
`Recordset.Columns`	returns the number of columns
`Recordset.ColumnName`	returns the name of a column
`Recordset.IsBof`	returns if the cursor is in the first row
`Recordset.IsEof`	returns if the cursor is in the last row
`Recordset.AddNew`	adds a new row
`Recordset.Update`	writes changes to the database server
`Recordset.Edit`	makes a row editable
`Recordset.MoveFirst`	moves the cursor to the first row
`Recordset.MoveNext`	moves the cursor to the last row
`Recordset.Value`	sets or returns the column value of the current row

4.4.2 RECORDSET.COLUMNS

```
Function Columns () As Integer
```

The function **`Columns()`** returns the number of columns in a Recordset.

A list of all available element functions of the standard class "Recordset" is provided in chapter "Recordset: Overview" on page 190.

4.4.3 RECORDSET.COLUMNNAME

```
Sub ColumnName (index As Integer)
```

The function **`Columnname()`** returns the name of the column with the number **`index`** in the Recordset.

```
...
Set rs = db.OpenRecordset (...)

For i = 1 To rs.Columns
  Print rs.ColumnName (i)
Next
...
' reset all Recordsets before reopening the database
    set rs = Nothing
...
```

A list of all available element functions of the standard class "Recordset" is provided in chapter "Recordset: Overview" on page 190.

4.4.4 RECORDSET.ISBOF

```
Function IsBof () As Boolean
```

The function **IsBof()** returns if the cursor is in the first row of the Recordset (*Beginning of File*).

A list of all available element functions of the standard class "Recordset" is provided in chapter "Recordset: Overview" on page 190.

4.4.5 RECORDSET.ISEOF

```
Function IsEof () As Boolean
```

The function **IsEof()** returns if the cursor is in the last row of the Recordset (*End of File*).

A list of all available element functions of the standard class "Recordset" is provided in chapter "Recordset: Overview" on page 190.

4.4.6 RECORDSET.ADDNEW

```
Sub AddNew ()
```

The function **AddNew()** adds a new row at the end to the Recordset. Changes will apply not until the call of **Update** (see "RECORDSET.UPDATE", p. 192).

```
...
Dim rs As Recordset
Set rs = db.OpenRecordset ("SELECT FROM TestTable", dbDynaset)
...
rs.AddNew
rs.Value(rs.ColumnName (1)) = "xxx"
rs.Update
...
' reset all Recordsets before reopening the database
    set rs = Nothing
...
```

A list of all available element functions of the standard class "Recordset" is provided in chapter "Recordset: Overview" on page 190.

4.4.7 RECORDSET.UPDATE

```
Sub Update ()
```

The function **Update()** writes all changes made in the Recordset to the database server.

A list of all available element functions of the standard class "Recordset" is provided in chapter "Recordset: Overview" on page 190.

4.4.8 RECORDSET.EDIT

```
Sub Edit ()
```

The function **Edit()** makes the row in the Recordset in which the cursor is placed, editable.

A list of all available element functions of the standard class "Recordset" is provided in chapter "Recordset: Overview" on page 190.

4.4.9 RECORDSET.MOVEFIRST

```
Sub MoveFirst ()
```

The function **MoveFirst()** sets the cursor to the first row of the Recordset. If there are no data in the Recordset, **Recordset.IsEof()** will return "True".

A list of all available element functions of the standard class "Recordset" is provided in chapter "Recordset: Overview" on page 190.

4.4.10 RECORDSET.MOVENEXT

```
Sub MoveNext ()
```

The function **MoveNext()** sets the cursor to the next row of the Recordset. If **Recordset.IsEof()** returns "True" before the call of **MoveNext()**, **MoveNext()** will cause a runtime error.

A list of all available element functions of the standard class "Recordset" is provided in chapter "Recordset: Overview" on page 190.

4.4.11 RECORDSET.VALUE

```
read value:
  Function Value (columnName As String) As String

'set new value:
  Value (columnName) = newValue  ' newValue as string
```

`Value()` returns or sets the value for the column named **`columnName`** in the current row.

```
...
if db.Open ("DSN = DBNVTest") > 0 then

  Dim rs As Recordset
  Set rs = db.OpenRecordset ("SELECT FROM TestTable")
...
  rs.AddNew
  rs.Value(rs.ColumnName (1)) = "new product"
  rs.Update
print rs.Value(rs.ColumnName(1))
...
```

A list of all available element functions of the standard class "Recordset" is provided in chapter "Recordset: Overview" on page 190.

4.5 STANDARD CLASS "DIRECTORY"

The standard class "Directory" is provided in **NextView®4 Script** to access directories and files of the operating system. An object of the class "Directory" is created with the command **New**.

- **To make the standard class "Directory" available in NextView®4 Script, the module Operating System (OS) must be imported at the beginning of the script (see "IMPORT", p. 167).**
- **The user must have writing and/or reading access to the directories to run the routines of the class "Directory". The files to be modified must not be open in a program.**

The following example gets the file names of the existing files in the directory **c:\Program Files\Nextview 4.4**:

```
' Import Operating System support at the beginning of the NV Script
Import "os"
...

Sub Test ()
  Dim dir as Directory
  Dim str, names(40) as String
  Dim i as Integer

  set dir = new Directory
  i = 0
  names(i) = dir.FindFirst("c:\Program Files\NextView 4.4\*.*")
  do
    str = dir.FindNext
    if str <> "" then
     i= i+1
     names(i) = str
    End If
  Loop until (str = "" Or i >= ubound(names))
  dir.FindClose
  ...
End Sub
```

4.5.1 DIRECTORY: OVERVIEW

Element function	Description
`Directory.FindFirst`	returns the first file in the specified directory path
`Directory.FindNext`	returns the next file
`Directory.FindClose`	stops the file search
`Directory.MakeDir`	makes a new directory
`Directory.RemoveDir`	deletes an empty directory
`Directory.CopyFile`	copies an existing file
`Directory.MoveFile`	moves an existing file or directory
`Directory.DeleteFile`	deletes an existing file

4.5.2 DIRECTORY.FINDFIRST

```
Function FindFirst (path as String) As String
```

The function **`FindFirst`** returns the first file name in the specified directory path **`path`**. The variable **`path`** may contain replacement characters such as **`*`** or **`?`**.

```
Function FileExists (name as String) as Boolean
  Dim str as String
  Dim dir as Directory
  set dir = new Directory
  str = dir.FindFirst(name)
  if str <> "" then
    FileExists = True
  else
    FileExists = False
  end if
  dir.FindClose
end Function
```

A list of all available element functions of the standard class "Directory" is provided in chapter "Directory: Overview" on page 196.

4.5.3 DIRECTORY.FINDNEXT

```
Function FindNext () As String
```

The function **FindNext** returns the next file name in the directory path specified in the command "**Directory.FindFirst**".

A list of all available element functions of the standard class "Directory" is provided in chapter "Directory: Overview" on page 196.

4.5.4 DIRECTORY.FINDCLOSE

```
Sub FindClose ()
```

The procedure **FindClose** stops the file search of **Directory.FindFirst**.

A list of all available element functions of the standard class "Directory" is provided in chapter "Directory: Overview" on page 196.

4.5.5 DIRECTORY.MAKEDIR

```
Function MakeDir (path as String) As Boolean
```

The function **MakeDir** creates a directory named **path** and returns **True** if created successfully.

A list of all available element functions of the standard class "Directory" is provided in chapter "Directory: Overview" on page 196.

4.5.6 DIRECTORY.REMOVEDIR

```
Function RemoveDir (path as String) As Boolean
```

The function **RemoveDir** deletes an empty directory named **path** and returns **True** if deleting was successful. The directory will not be deleted and **False** will be returned if there is still a file in the directory or if another error has occurred.

A list of all available element functions of the standard class "Directory" is provided in chapter "Directory: Overview" on page 196.

4.5.7 DIRECTORY.COPYFILE

```
Function CopyFile (name as String, newname as String,
overwrite as Boolean) As Boolean
```

The function **CopyFile** copies an existing file **name** to a new file named **newname**. If **overwrite** is **True**, an existing file will be overwritten.

A list of all available element functions of the standard class "Directory" is provided in chapter "Directory: Overview" on page 196.

4.5.8 DIRECTORY.MOVEFILE

```
Function MoveFile (name as String, newname as String) As
Boolean
```

The function **MoveFile** moves an existing file or directory **name** to **newname**. If moving directories, **newname** must be stored on the same drive. Neither the file nor the directory may exist.

A list of all available element functions of the standard class "Directory" is provided in chapter "Directory: Overview" on page 196.

4.5.9 DIRECTORY.DELETEFILE

```
Function DeleteFile (name as String) As Boolean
```

The function **DeleteFile** deletes an existing file **name**.

A list of all available element functions of the standard class "Directory" is provided in chapter "Directory: Overview" on page 196.

4.6 STANDARD CLASS "STREAM"

The standard class "Stream" is provided in **NextView®4 Script** to access text files and serial interfaces. An object of the class "Stream" cannot be created with the command **New**. It is returned by the function **Stream.FileStream** instead.

To make the standard class "Stream" available in NextView®4 Script, the module Operating System (OS) must be imported at the beginning of the script (see "IMPORT", p. 167).

The following example creates a text file **datafile.txt**, adds data to the file, then reads the file, and prints the content:

```
' Import Stream support at the beginning of the NextView Script
Import "os"
...

Sub Test ()
  Dim fio as Stream
  Dim i as Integer

  ' Create text file datafile.txt with writing access
  set fio = FileStream("datafile.txt", "w")
  if fio is Nothing then Exit Sub

  fio.Print "This is an example file" & chr(13) & chr(10)
  fio.Close

  ' Open text file datafile.txt to insert something at the end
  set fio = FileStream("datafile.txt", "a")
  if fio is Nothing then Exit Sub

  i = fio.Seek (0, StreamEnd)
  fio.Print "1" & chr(13) & chr(10)
  fio.Print "2" & chr(13) & chr(10)
  fio.Print "3" & chr(13) & chr(10)
  fio.Close
```

```
  ' Open text file datafile.txt with reading access
  set fio = FileStream("datafile.txt", "r")
  if fio is Nothing then Exit Sub

  do while Not(fio.IsEof)
    print fio.gets
  loop
  fio.Close

End Sub
```

The following example reads a line terminated by a carriage return from a serial interface and prints it:

```
' Import Stream support at the beginning of the NextView Script
Import "os"
...

Sub Test ()
  Dim ch as String
  Dim str as String
  Dim timeout as Integer
  Dim fio as Stream

  set fio = FileStream("com1", "w+")
  if fio is Nothing then
    Print "Error: Open COM failed."
    Exit Sub
  End If

  If fio.CommSetup ("38400", CommParityNone) <> 0 Then Exit Sub

  ' Send string to serial device
  fio.Print "Start sequence" & chr(13) & chr(10)

  ' receive the string terminated by a carriage return
  timeout = TickCount

  do while (TickCount < (timeout+5000)) AND ch <> chr(13)
    ch = fio.getc
    if (ch <> "") then
      str = str & ch
      timeout = TickCount
    end if
  loop

  if (TickCount >= (timeout+5000)) then print "Timeout"
  print str
  fio.Close
End Sub
```

4.6.1 STREAM: OVERVIEW

Element function	Description
Stream.FileStream	returns a stream object
Stream.Close	closes the stream
Stream.Seek	changes the writing or reading position in the stream
Stream.Rewind	sets the writing or reading position to the beginning of the stream
Stream.Gets	returns the next line of the stream
Stream.Getc	returns a character of the stream
Stream.IsEof	returns if the end of the stream has been reached
Stream.Print	prints a line on the stream
Stream.Size	returns the size of the stream
Stream.Modtime	returns date and time of the last stream change
Stream.CommSetup	defines the baud rate of the serial COM stream
Stream.CommControl	changes the serial COM lines DTR and RTS
Stream.CommState	gets the state of the serial COM lines CTS, DSR, RING, and RLSD

4.6.2 STREAM.FILESTREAM

```
Function FileStream (path as String, mode as String) As
Stream
```

The function **FileStream** returns the stream with the path **path** using the defined access mode **mode**. The specified path to the stream to be opened can be relative or absolute. Text files as well as serial interfaces (e.g. "COM1") can be opened as stream. The following options are provided as access mode **mode**:

mode	Description
r	Opens a stream with reading access. If the stream does not exist or cannot be found, opening will fail.
w	Generates an empty stream with writing access. If the stream already exists, the content will be lost.
a	Opens a stream to add something. If the file does not exist, it will be created. If the file already exists, the content will not be deleted.
r+	Opens a stream for reading and writing. If the stream does not exist or cannot be found, opening will fail.
w+	Generates an empty stream for reading and writing. If the stream already exists, the content will be lost.
a+	Opens a stream to read and add something. If the file does not exist, it will be created. If the file already exists, the content will not be deleted.

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.3 STREAM.CLOSE

```
Sub Close ()
```

The procedure **Close()** closes an open stream.

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.4 STREAM.SEEK

```
Function Seek (pos as Integer, start as Integer) as Integer
```

The function **Seek()** sets the write/read pointer in the stream to the position **pos** relatively to **start**. The function returns the value of the write/read pointer in the stream after the call.

The following values can be used for **start**:

start	Description
StreamBegin	position is relative to the beginning of the stream
StreamEnd	position is relative to the end of the stream
StreamCurrent	position is relative to the current position in the stream

The command **Seek(0, StreamEnd)**, for example, places the write/read pointer to the end of the stream.

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.5 STREAM.REWIND

```
Sub Rewind ()
```

The procedure **Rewind()** sets the write/read pointer to the beginning of the stream.

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.6 STREAM.GETS

```
Function Gets () as String
```

The function **Gets()** reads out a line from the stream. The characters are read out from the stream until a carriage return or the end of the stream has been reached, or 1024 characters have been read at once.

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.7 STREAM.GETC

```
Function Getc () as String
```

The function **Getc()** reads a character from the stream.

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.8 STREAM.ISEOF

```
Function IsEof () as Boolean
```

The function **IsEof()** returns **True** if the end of the stream has been reached.

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.9 STREAM.PRINT

```
Sub Print (str as String)
```

The procedure **Print(str)** writes the string **str** in the stream at the current position of the write/read pointer.

To add characters to a stream via Print, the stream must have been opened with access mode "a" or "a+" (see "STREAM.FILESTREAM", p. 202) and the write/read pointer must be set to the end of the stream before the Print command (Seek(0,StreamEnd), see "STREAM.SEEK", p. 203).

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.10 STREAM.SIZE

```
Function Size () as Integer
```

The function **Size()** returns the size of the stream.

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.11 STREAM.MODTIME

```
Function Modtime () as String
```

The function **Modtime()** returns date and time of the last change of the stream.

The following format is used:

YYYY-MM-DD hh:mm:ss with: **YYYY: year**
MM: month
DD: day
hh: hour
mm: minute
ss: second

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.12 STREAM.COMMSETUP

```
Function CommSetup (baud as String, Optional parity as
Integer = CommParityNone, Optional stop as Integer =
CommStopBits10) as Integer
```


The function **CommSetup(baud)** sets the baud rate of the serial COM stream to **baud**. The function returns the value **0** if the call was successful.

The following values are possible for the parity bits **parity** and the number of stop bits **stop**:

Stop bits	Value
CommStopBits10	1 stop bit
CommStopBits15	1,5 stop bits
CommStopBits20	2 stop bits

Parity bits	Value
CommParityNone	No parity
CommParityOdd	Odd parity
CommParityEven	Even parity
CommParityMark	Mark parity
CommParitySpace	Space parity

```
Dim fio as Stream
set fio = FileStream("com1", "w+")
If fio.CommSetup ("38400", CommParityOdd, ComStopBits10) <> 0 Then

   ...

End If
```

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.13 STREAM.COMMCONTROL

```
Function CommControl (dtr as Integer, rts as Integer) as
Integer
```

The function **CommControl(dtr,rts)** changes the serial COM lines DTR

(*data-terminal-ready*) and RTS (*request-to-send*). The function returns the value **0** if the call was successful. The respective line can be influenced as follows:

State	Description
-1	no changes for the line
0	line is deleted
1	line is set

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.6.14 STREAM.COMMSTATE

```
Function CommState (Byref cts As Integer, Byref dsr As
Integer, Byref ring As Integer, Byref rlsd As Integer) As
Integer
```

The function **CommState(cts, dsr, ring, rlsd)** gets the state of the serial COM lines CTS (*clear-to-send*), DSR (*data-set-ready*), RING (*ring indicator*), and RLSD (*receive-line-signal-detect*). The function returns the value **0** if the call was successful.

The respective line can have the following states:

State	Description
0	line is not set
1	line is set

A list of all available element functions of the standard class "Stream" is provided in chapter "Stream: Overview" on page 202.

4.7 STANDARD CLASS "CLIPBOARD"

The standard class "Clipboard" is provided in **NextView®4 Script** to access the clipboard of the operating system. An object of the class "Clipboard" is created with the command **New**.

To make the standard class "Clipboard" available in NextView®4 Script, the module Operating System (OS) must be imported at the beginning of the script (see "IMPORT", p. 167).

The following example demonstrates how to use the standard class "Clipboard":

```
' Import Stream support at the beginning of the NextView Script
Import "os"
...
Sub Test ()
  Dim clip as Clipboard
  Dim str as String
  Dim i as Integer

  set clip = new Clipboard
  if clip.Open then
    str = ""
    for i = 1 to 16
      str = str & " " & NvAnalogIn(i).value
    next
    clip.Text = str
    clip.Close ' make accessible to other clipboard
  end if
End Sub
```

4.7.1 CLIPBOARD: OVERVIEW

Element function	Description
Clipboard.Open	opens the clipboard
Clipboard.Text	writes into the clipboard or reads out content
Clipboard.Close	closes the clipboard

4.7.2 CLIPBOARD.OPEN

```
Function Open () As Boolean
```

The function **Open** opens the clipboard of the operating system to be used in **NextView®4 Script**. Other programs can access the clipboard not until it has been closed (see "CLIPBOARD.CLOSE", p. 210).

A list of all available element functions of the standard class "Clipboard" is provided in chapter "Clipboard: Overview" on page 209.

4.7.3 CLIPBOARD.TEXT

```
' Read content of clipboard
  Function Text () As String
' Write content to clipboard
  text = newText
```

The function **text** is used to read out the clipboard or to deposit new content in the clipboard.

A list of all available element functions of the standard class "Clipboard" is provided in chapter "Clipboard: Overview" on page 209.

4.7.4 CLIPBOARD.CLOSE

```
Sub Close()
```

The procedure **Close** shuts the clipboard for **NextView®4 Script**. Other programs can access the clipboard not until then.

A list of all available element functions of the standard class "Clipboard" is provided in chapter "Clipboard: Overview" on page 209.

4.8 STANDARD CLASS "XNF"

The standard class "XNF" (*extended INI File format*) is provided in **NextView®4 Script** to access initialization text files in Windows® INI format. An object of the class "XNF" is created with the command **New**. The class "XNF" features several simple procedures to access INI files.

To make the standard class "XNF" available in NextView®4 Script, the module XNF must be imported at the beginning of the script (see "IMPORT", p. 167).

The following example demonstrates how to use the standard class "XNF" for an initialization file **Init.xnf** in the directory **c:\data** with the following content:

```
[General]
Date=2010-01-22
AnalogOutNumber=2
[AnalogOut1]
Filename=Points1.txt
[AnalogOut2]
Filename=Points2.txt
```

```
Import "XNF"
...
Dim IniFile as XNF
Dim i as Integer
Dim nAnOut as Integer
Dim AnOutFileName(20) as String

Set IniFile = new XNF

If IniFile.LoadParams("c:\data\init.xnf") Then
  Print "Init Date: " & IniFile.GetSectionParam ("General", "Date")
  nAnOut = IniFile.GetSectionParam ("General", "AnalogOutNumber")
  for i = 1 to nAnOut
    AnOutFileName(i) = IniFile.GetSectionParam ("AnalogOut"&i, "Filename")
  Next

Else
  print "Open XNF file failed."
End If
```

4.8.1 XNF: OVERVIEW

Element function	Description
XNF.LoadParams	opens the XNF text file
XNF.SaveParams	writes the XNF text file
XNF.SectionCount	returns the number of XNF sections
XNF.GetSectionName	returns the XNF section name
XNF.SectionLineCount	returns the number of lines in the XNF section
XNF.GetSectionLine	returns a line of the XNF section
XNF.GetSectionParam	reads a parameter of the XNF section
XNF.SetSectionParam	sets a parameter in the XNF section
XNF.DeleteSection	deletes the content of the XNF section

4.8.2 XNF.LOADPARAMS

```
Function LoadParams (filename as String) As Boolean
```

The function **LoadParams** opens an initialization text file written in Windows® INI format.

A list of all available element functions of the standard class "XNF" is provided in chapter "XNF: Overview" on page 212.

4.8.3 XNF.SAVEPARAMS

```
Function SaveParams (filename as String, merge as Integer) as
Boolean
```

Saves the XNF sections in an INI file named **filename**. If **merge** = 1, the INI sections are added to the existing sections in the INI file, otherwise a new file will be created.

```
Dim IniFile as XNF
Set IniFile = new XNF

if IniFile.SetSectionParam ("AnalogOut1","Filename","AnOut1.txt")
then
  print "Set Ok"
End If

if IniFile.SaveParams("c:\data\newinit.xnf",0) then
  print "Save Ok"

End If
```

A list of all available element functions of the standard class "XNF" is provided in chapter "XNF: Overview" on page 212.

4.8.4 XNF.SECTIONCOUNT

```
Function SectionCount () as Integer
```

Returns the number of XNF sections.

```
print "Number of sections: " & IniFile.SectionCount
```

A list of all available element functions of the standard class "XNF" is provided in chapter "XNF: Overview" on page 212.

4.8.5 XNF.GETSECTIONNAME

```
Function GetSectionName (index as Integer) as String
```

Returns the name of the XNF section **index**. It starts with 1 to the number of XNF sections (see "**XNF.SectionCount**", p. 213).

```
print "1. section Name: " & IniFile.GetSectionName(1)
```

A list of all available element functions of the standard class "XNF" is provided in chapter "XNF: Overview" on page 212.

4.8.6 XNF.SECTIONLINECOUNT

```
Function SectionLineCount (section as String) as Integer
```

Returns the number of lines of the XNF section.

A list of all available element functions of the standard class "XNF" is provided in chapter "XNF: Overview" on page 212.

4.8.7 XNF.GETSECTIONLINE

```
Function GetSectionLine (section as String, index as Integer)
as String
```

Returns the line **index** of the XNF section **section**. **Index** starts with 1 to the number of lines in the XNF section (see "**XNF.SectionLineCount**", p. 214).

A list of all available element functions of the standard class "XNF" is provided in chapter "XNF: Overview" on page 212.

4.8.8 XNF.GETSECTIONPARAM

```
Function GetSectionParam (section as String, name as String)
as String
```

Reads out the parameter **name** of the section **section**.

```
 [Data1]
NameX=Testvalue

Dim str as String
str = IniFile.GetSectionParam("Data1","NameX")
Print str ' outputs Testvalue
```

A list of all available element functions of the standard class "XNF" is provided in chapter "XNF: Overview" on page 212.

4.8.9 XNF.SETSECTIONPARAM

```
Function SetSectionParam (section as String, name as String,
value as String) as Boolean
```

Writes the value **value** of the parameter **name** in the XNF section **section**.

```
Dim str as String
If IniFile.SetSectionParam("Section2","NameY","Testwert Y") Then
End If
```

Creates the following section in the XNF file:

```
[Section2]
NameY=Testwert Y
```

A list of all available element functions of the standard class "XNF" is provided in chapter "XNF: Overview" on page 212.

4.8.10 XNF.DELETESECTION

```
Sub DeleteSection (section as String)
```

Removes the content of a section in an XNF object.

A list of all available element functions of the standard class "XNF" is provided in chapter "XNF: Overview" on page 212.

5 INDEX

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y